ÉCOLE DES MINES
DE PARIS

ÉCOLE DES MINES DE PARIS
CENTRE DE RECHERCHE EN INFORMATIQUE

THÈSE
présentée
pour obtenir

le GRADE de DOCTEUR EN SCIENCES
INFORMATIQUE TEMPS RÉEL, ROBOTIQUE ET AUTOMATIQUE
DE L'ÉCOLE DES MINES DE PARIS

par M. **Thierry J-F. OMNÈS**

Sujet :

Acropolis - un précompilateur de spécification pour l'exploration du transfert et du stockage des données en conception de systèmes embarqués à Haut Débit

Soutenue le mercredi 9 mai 2001, 15h00

à l'Écoles des mines de Paris, 60, boulevard Saint-Michel, Paris

Devant le jury composé de :

MM.	Anne MIGNOTTE	Président
	Francky CATTHOOR	Directeur
	Mateo VALERO	Rapporteur
	Jef VAN MEERBERGEN	Rapporteur
	Sanjay RAJOPADHYE	Rapporteur
	Michel DAYDÉ	Rapporteur
	François IRIGOIN	Examinateur
	Fabien COELHO	Examinateur
	César DOUADY	Examinateur
	Jean CASTÉRES	Examinateur

Acropolis - a system-level precompiler for data transfer and storage exploration (DTSE) in High-Throughput embedded system design

PhD dissertation

Thierry J-F. OMNÈS

Sujet principal de la thèse

Pour les applications multimedia embarquées récentes, l'optimisation simultanée de la performance, du nombre de ressources, du coût, de la taille et de la consommation électrique du système est nécessaire. La majorité des techniques d'ordonnancement et de placement existantes sont mal adaptées à ce nouveau contexte car elles ne s'intéressent bien souvent qu'à la performance pure sous des hypothèses simplificatrices telles la régularité temporelle, la linéarité des adresses ou encore l'assignation unique. Les approches s'intéressant à la modélisation du nombre de ressources sont quant à elles limitées à l'utilisation de ressources parfaitement symétriques, ce qui s'accorde mal avec l'hétérogénéité croissante des systèmes émergents. Afin de palier ces limitations sur architecture à mémoire partagée, Sven Wuytack introduit une nouvelle méthodologie d'ordonnancement–placement s'intéressant **globalement** à l'ensemble des corps de boucle de l'application, appelée Storage Cycle Budget Distribution (SCBD). SCBD se base sur l'algorithme Conflict-Directed Ordering (CDO) de complexité $O(n^4m)$ où n est le nombre d'opérations mémoire et m mesure la liberté d'ordonnancement moyenne pour ces opérations exprimée en cycles mémoire.

Dans cette thèse, nous résolvons un problème CDO généralisé. Combiné à des techniques d'approximation spécialisées, nous parvenons ainsi à en réduire la complexité à $O(n^{3/2}m^{1/2}ln(m^{1/2}))$, ce qui autorise une conception interactive et/ou des ré-ordonnancements à la volée. Cette analyse en profondeur nous conduit à l'algorithme d'ordonnancement–placement espace–temps Generalised Conflict-Directed Ordering (GCDO(k)). Combiné au développement de nouvelles bornes d'optimisation coût $\times$ temps (C$\times$T), nous montrons la supériorité et la quasi-optimalité de GCDO(k) pour résoudre des problèmes réels parallèles et/ou de grande taille. Cette contribution ouvre de nouveaux champs d'application prometteurs tout particulièrement pour plates-formes à mémoires distribuées intégrant des contraintes espace–temps fortes, par exemple des entrées/sorties audio–video par streaming et/ou des mémoires de type Synchronous Dynamic RAM (SDRAM).

Mots-clés: algorithmes de graphe dynamiques, problèmes de coloration et clique maximale, techniques de résolution multi-étage et adaptative, approximation de problèmes NP-complet, sélection et identification (multi-dimensionnelles) espace–temps, programmation entière, programmation logique par contraintes, algorithmes d'évolution stochastique, technique de résolution divise–et–règne, exploration du transfert et du stockage des données, systèmes embarqués de faible coût mais Haute Performance, conception de systèmes électroniques de faible consommation assistée par ordinateur, ordonnancement–placement plate-forme dépendant mais logiciel–matériel indépendant **avant** conception conjointe, réorganisation du stockage des données pour utilisation efficace de cache, compilateurs embarqués, parallélisme d'instructions, découplage des accès mémoire par décomposition espace–temps, parallélisme de tâche, parallélisme objet, parallélisation automatique, système opéra–toire embarqué, systèmes distribués, conception de plate-forme, outils d'aide à la décision, technologies de l'information et de la communication, applications au traitement de la parole et à l'audio, applications au graphisme et à la vidéo, application aux protocoles réseau, application au multimedia.

Main Ph.D topic definition

For most recent embedded multimedia applications, optimising simultaneously the raw performance, the number of ressources, the cost, the size and the power consumption of the system is necessary. The majority of existing ordering and mapping techniques are poorly suited for this new context because they often only consider the optimisation of raw performance under simplifying assumptions such as temporal regularity, linear addressing schemes and single assignment. The few ordering and mapping techniques to model the number of ressources are limited to symmetric behaviour, which is poorly suited for the growing heterogeneity of emerging systems. To alleviate all these shortcomings on shared memory architectures, Sven Wuytack introduces a novel ordering and mapping methodology able of taking **globally** all loop bodies as scope, called Storage Cycle Budget Distribution (SCBD). SCBD relies on the Conflict-Directed Ordering (CDO) algorithm, exhibiting a $O(n^4m)$ complexity where n is the number of memory operations and m measures the average scheduling freedom for these memory operations expressed in memory cycles.

In this thesis, we solve a generalised CDO problem. Combined to dedicated approximation techniques, this allows us to reduce its complexity to $O(n^{3/2}m^{1/2}ln(m^{1/2}))$ and therefore to enable interactive design and even on–the–fly reconfiguration. Our in-depth analysis brings us to the introduction of the space–time ordering and mapping algorithm Generalised Conflict-Directed Ordering (GCDO(k)). Combined to the introduction of new cost $\times$ time (C$\times$T) bounds, we show superior and near-optimal behaviour of GCDO(k) to solve large scale and/or (highly) parallel design problems. Our contribution opens new promising application areas in designing distributed platforms under stringent space–time constraints for example arising from audio–video I/O by streaming and/or from the use of Synchronous Dynamic RAMs (SDRAMs).

Keywords: dynamic graph algorithms, colouring and maximum clique problems, multi–level and adaptive solution techniques, approximation to NP-complete problems, (multi–dimensional) space–time identification and selection, Integer–Linear Programming (ILP) technology, Constraint-Logic Programming (CLP) technology, Stochastic Evolution Algorithms (SEAs), divide–and-conquer solution techniques, Data Transfer and Storage Exploration (DTSE), low-cost but still High Performance embedded systems, low-power Electronic Design Automation (EDA), platform-dependent but still HW/SW-independent ordering and mapping **before** co-design, data layout re–organisation for efficient cache utilisation, embedded compilers, Instruction-Level Parallelism (ILP), decoupling of memory accesses using space–time patterns, task versus data parallelism, object parallelism, automatic parallelisation, embedded operating systems, distributed systems, platform-based design, decision–support systems, Information and Communication Technology (ICT), speech and audio applications, graphics and video applications, network protocol applications, multimedia applications.

Curriculum Vitæ

Thierry J-F. Omnès was born in Lannion, Côtes d'Armor (France) in 1970.

Thierry holds an engineering degree in computer science and applied mathematics from INP-ENSEEIHT, Toulouse (France); a DEA in fundamental computation and parallelism from INPT-ENSAE-UPS, Toulouse (France); a Master of Philosophy (MPhil) in electronics and computer science from the University of Southampton (United Kingdom) where he served as staff in the academic year 1996-97; and is presenting this manuscript for obtaining the Ph.D in real-time systems, robotics and automation from Ecole Nationale Supérieure des Mines, Paris (France).

Since 1992, Thierry has been active in High-Performance Computing and Networking (HPCN), Numerical Algorithms and Scientific Computation with a view to industrial applications and technology transfer to Small and Medium Enterprises (SMEs), including STRACO, Compiègne (France); Aérospatiale, Toulouse (France); LMS - Numerical Technologies, Leuven (Belgium); NUMECA, Brussels (Belgium); Bertin & Compagnie, Paris (France) and Teuchos Group, Versailles (France). Thierry has co-authored seven competitive research proposals (of which five have been granted) and participated in four leading European R&D projects involving ESPRIT-IV TOOLSHED, PROMENVIR, HPC-VAO and especially the Micro-Electronics Development for European Applications (MEDEA [1]) phases I. and II. - System-Level Methods and Tools (EUREKA S1535 MEDEA A403 - SMT) project from the pan-European co-operative R&D EUREKA [2] programme, where one main objective is to leverage the capabilities of Electronic Design Automation (EDA) tools for an anticipated increase in system complexity of two orders of magnitude as a result of the System-on-a-Chip (SoC) revolution.

During the period 1992-98, Thierry visited some of the best European teams in this area involving the parallel algorithms team of Iain Duff at Météo France - European Centre for Advanced Research and Training in Scientific Computation (CERFACS), Toulouse (France); the applied artificial intelligence group of Jean-François Cloarec within the French National Centre for Research and Studies in Telecommunications (CNET), Lannion (France); the image processing team of Jean-Claude Grossetie within the Joint Research Centre (JRC) of the European Commission (EC), Ispra (Italy); the numerical algorithms team of Pierre Leca within the French National Aerospace Research Establishment (ONERA), Châtillon (France); the industrial applications team of Mike Surridge at the Parallel Applications Centre (PAC) today IT Innovation Centre, Southampton (United Kingdom) and the numerical analysis group of Paul Van Dooren within the Centre for Systems Engineering and Applied Mechanics (CESAME), Louvain-la-Neuve (Belgium).

Since 1998, Thierry has (slightly) changed his main focus for low-cost but High-Performance (distributed) embedded systems by joining the System Exploration for Memory and Power (SEMP) team of Francky Catthoor within the Design Technology for Integrated Information and Communication Systems (DESICS) division of the Interuniversity Micro-Electronics Centre (IMEC), Leuven (Belgium). Within IMEC-DESICS/SEMP, Thierry is currently software architect and leader for the parallel multimedia precompiler - Acropolis [3] project. Acropolis acts as a precompilation stage to conventional C/C++ software compilation suites, mainly for boosting power, memory, bandwidth and cache performance in an embedded (and parallel) software context. In order to maximise the impact of the Acropolis project, Thierry is responsible for managing research co-operation with ENS-LIP, Lyon (France); INSA, Lyon (France); ENSMP-CRI, Fontainebleau (France); IRISA-COSI, Rennes (France); Philips-NATLAB, Eindhoven (The Netherlands); CSEM, Neuchâtel (Switzerland); UPC, Barcelona (Spain); NexWave Solutions, Montpellier (France); Globepsan - T.Sqware, Guyancourt

1. See: `http://www.medea.org/` for more information.
2. See: `http://www.eureka.be/` for more information.
3. See: `http://www.imec.be/acropolis/` for more information.

(France); Prosilog SA, Évry (France); Polyspace Technologies, Grenoble (France); Esterel Technologies, Nice (France); and PGI, Portland (USA) as well as promoting the project towards many other research centres and companies, involving Cadence Design Systems, San Jose (USA); Philips Semiconductors, Eindhoven (The Netherlands); Motorola, Toulouse (France); ST Micro-Electronics, Grenoble (France); Hewlett Packard Laboratories, Paolo Alto (USA); High-Performance Computing Institute, iHPC (Singapore); Thomson Multimedia Corporate Research, Rennes (France); Netgem R&D, Rennes (France); EolRing, Caen (France); Propsys, Chambéry (France); Clearstream, Luxembourg (Luxembourg); Metrowerks, Bâle (Switzerland); etc.

Acknowledgments

Thierry dedicates this Ph.D to those who have contributed in generating his passion for research and in maintaining this often uncertain and sometimes even destabilizing activity for almost a decade: Philippe Berger, Michel Daydé, Patrick Amestoy, Mario Arioli, Prof. Iain Duff, Pierre-Henri Cros, Albert Feroldi, Jean-François Cloarec, Jean-Claude Grossetie, Prof. Mohamed Ali Hamdi, Prof. Pierre Leca, François-Xavier Roux, Prof. Tony Hey, Mike Surridge, Prof. Paul van Dooren, Prof. Joseph Noailles, Prof. Bernard Philippe, Prof. Jean-Pierre Coyette, Prof. Charles Hirsch, Jean-François Coudert and especially Prof. Francky Catthoor, Ivo Bolsens, Prof. Hugo de Man, Prof. François Irigoin, Fabien Coelho, Prof. Rudi Lauwereins, Prof. Dirk Roose, Prof. Anne Mignotte, Prof. Sanjay Rajopadhye, Prof. Patrice Quinton and Prof. Christian Piguet who are directly involved in the presented work.

Thierry is also most thankful to his (direct) research colleagues along these years, especially Michel Amsellem, Stéphane Morucci, Nick Floros, Yuri Papai, Chidamber Kulkarni, Koen Danckaert, Erik Brockmeyer, Arnout Vandecappelle, Antoine Fraboulet, Fabien Coelho, Denis Vanderstraeten, Thierry Franzetti, Bart Boogaerts and Marc Heijligers for their close co-operation and friendship: making hard challenges sometimes so pleasant that it was in the end real shared fun!

Thierry would finally also like to thank the European Ph.D reviewers and the members of the jury for honouring this work with their free time and attention : Prof. Mateo Valero, Prof. Jef van Meerbergen, Prof. Michel Daydé, Jean Castéres and César Douady.

Let this passion go on ...

Table des matières

Première partie

Objectifs, Méthodologie et Principaux Résultats

CHAPITRE 1

État de l'Art, stratégie et DTSE

1. Vers une conception de systèmes multi-critère, flexible et maîtrisée

Alors que la grande majorité des progrès en architecture de microprocesseurs furent guidés par les applications de calcul scientifique et entier pendant des décennies, les applications multimedia prennent aujourd'hui une importance dominante dans les traitements de nos systèmes d'information et de communication [**68**].

En étudiant plus en détail les caractéristiques de ce domaine applicatif [**103**], on constate une répartition de charge dominée par le transfert et le stockage des données en forte corrélation avec une répartition de consommation électrique également (largement) dominée par le transfert et le stockage des données tant pour la partie programmable [**93**] [**258**] que pour la partie dédiée à des traitements spécifiques [**159**] du système. Ceci s'explique par la nature même des applications multimedia, communicant et retraitant localement de grands volumes d'information.

Afin d'améliorer les performances des systèmes multimedia émergents, il reste bien sûr possible d'adapter des concepts déjà bien étudiés, tels:

(1) **l'élimination des temps morts** par l'introduction d'un *flot de données asynchrone* permettant de minimiser les attentes actives d'accès à des ressources – tels des processeurs vectoriels [**189**], des registres et leur système d'interconnexion [**48**] ou une mémoire partagée [**153**]; voire permettant un découplage traitement–communication parfait sans perte de cycle [**77**], y compris dans le cadre général multi-instructions multi-données (MIMD) [**4**] [**5**].

(2) **la division des temps de cycle** par l'introduction d'unités *pipelinées* pour décupler le débit tant au niveau mémoire [**222**] [**199**] qu'au niveau processeur, pouvant atteindre un nombre d'étages allant jusqu'à 128 et plus dans le cas des architectures vectorielles [**224**].

(3) **la localisation des traitements** par exemple par l'introduction d'une *hiérarchie mémoire* permettant d'améliorer la bande passante processeur–mémoire effective de façon remarquable [**143**], y compris dans un contexte distribué [**259**].

(4) **la multiplication du débit unitaire** en introduisant du parallélisme d'instructions, ce qui est par exemple à la base des architectures very-long instruction word (VLIW); des extensions multimedia des processeurs actuels [**46**] [**47**] [**237**] et des architectures single-instruction multiple data (SIMD) en général.

En combinant ces concepts, il est possible de gagner plusieurs ordres de grandeur en performance [**197**] [**198**] jusqu'à atteindre les 100 TFlop/s soutenus de l'Accelerated Strategic Computing Initiative (ASCI) Américaine prévue pour juin 2004 [**160**].

Il serait pourtant fâcheux d'oublier qu'en basculant du marché élitiste du calcul scientifique et entier intensif au marché grand public du multimedia, la métrique de référence n'est plus seulement la performance mais une approche *multi-critère* entre performance, coût, taille et également consommation électrique. Alors que la mémoire jouait déjà une importance considérable pour des objectifs de performance pure (mesurée en mega-instruction per second, MIPS ou mega floatting-point operation per second, MFlop/s), elle prend aujourd'hui un rôle central pour ces trois nouveaux critères [**146**] (mesurés en millimètre carré de surface de silicium, mm^2 (60-80%); milliwatts de consommation électrique, mW (40-60%) et en dollar, \$).

Il serait également fâcheux d'oublier qu'entre le développement des premiers super-ordinateurs Cray au début des années 1970s et la révolution des télécommunications mobiles ces dernières années, les technologies de la micro–électronique et le visage du calcul à Haute Performance ont considérablement évolué [**161**]. Nous entrons dans l'ère du System–on–a–Chip (SoC), ce qui permet de concevoir un système parallèle multi-instructions multi-données (MIMD), ses mémoires et son réseau d'interconnexion intégré sur un seul circuit, appelé couramment *plate-forme* [**39**].

Dans ce nouveau contexte applicatif, métrique et technologique, la *réutilisation de propriété intellectuelle* souvent standardisée devient essentielle car il est impossible de concevoir des systèmes aussi complexes en partant de rien, même pour les sociétés leader du secteur. La réduction des coûts de fabrication et de commercialisation induite par la *réutilisation* et une meilleure *maîtrise des temps de conception* ne suffisent pas pour autant à rentabiliser rapidement un tel investissement, nécessitant une large *flexibilité* du système afin de couvrir plusieurs générations de produits pendant plusieurs années, comme cela fut le cas pour les fameux Personnel Computer (PCs) programmables qui ont totalement dominé le marché pendant deux décennies consécutives.

Face à cette nouvelle donne – appelée *ère post-PC* [**220**] [**66**] – la *reconfigurabilité* apparaît comme une technique incontournable [**65**] car elle propose des solutions au problème de *flexibilité* imposé par les contraintes économiques du secteur, tout en offrant des opportunités d'optimisation de système multi-critère proches de la conception de circuits intégrés dédiés [**264**], ce qui est depuis peu étendu au niveau mémoire et interconnexion [**261**].

Pour toutes les raisons évoquées ci-dessus, nous sommes convaincus de la nécessité d'introduire une nouvelle méthodologie de conception des systèmes d'information et de communication:

– permettant la *réutilisation* et la *maîtrise du temps de conception*;
– intégrant un moteur d'optimisation *multi-critère*; et
– permettant une certaine *flexibilité* basée sur l'emploi (partiel) de solutions programmables et l'exploitation de nouvelles formes de *reconfigurabilité* au niveau mémoire et interconnexion [**261**] [**109**]

pour réduire la part dominante du transfert et du stockage, tout particulièrement dans un contexte d'applications et d'architecture multimedia dont la complexité dépasse les capacités d'exploration humaine [1].

2. L'approche Data–Transfer and Storage Exploration (DTSE)

Depuis 1988, la division Design Technology for Integrated Information and Communication Systems (DESICS) d'IMEC, Leuven (Belgique) s'intéresse au développement de nouvelles méthodologies de conception de systèmes d'information et de communication. Sous la direction du Prof. Francky Catthoor, également professeur associé à la Katholieke Universiteit Leuven (K.U.Leuven), une méthodologie complète fut introduite [**38**] pour permettre le passage de l'algorithmique à la conception conjointe logiciel–matériel traditionnellement utilisée dans ce domaine [**281**] [**75**]. Au sein de cette méthodologie, l'accent est mis sur une exploration du transfert et du stockage des données – Data Transfer and Storage Exploration (DTSE) en anglais – au niveau d'abstraction des langages de programmation C/C++ **avant** que le partitionnement logiciel–matériel (co-design en anglais) ainsi que tout autre détail d'implémentation ne soient connus. Cette approche nouvelle permet de réduire les besoins en stockage et en bande passante [**273**], qui représentent le goulot d'étranglement typique des applications multimedia de nouvelle génération [**170**]. Nous présentons en Figure 1 la variante de la méthodologie Data Transfer and Storage Exploration (DTSE) pour plates-formes parallèles (partiellement) programmables à mémoires et caches (partiellement) contrôlables et reconfigurables.

1. Ceci étant typiquement le cas dès qu'il s'agit de traiter plus de 1,000 lignes de C/C++ ou de développer ne serait-ce que quelques lignes de code pour une *plate-forme* distribuée reconfigurable.

Dans la suite de ce paragraphe, nous en expliquons les différentes étapes sans entrer dans les détails d'une application réelle, dont nous reportons l'exposé à la Partie II - Chapitre 2.

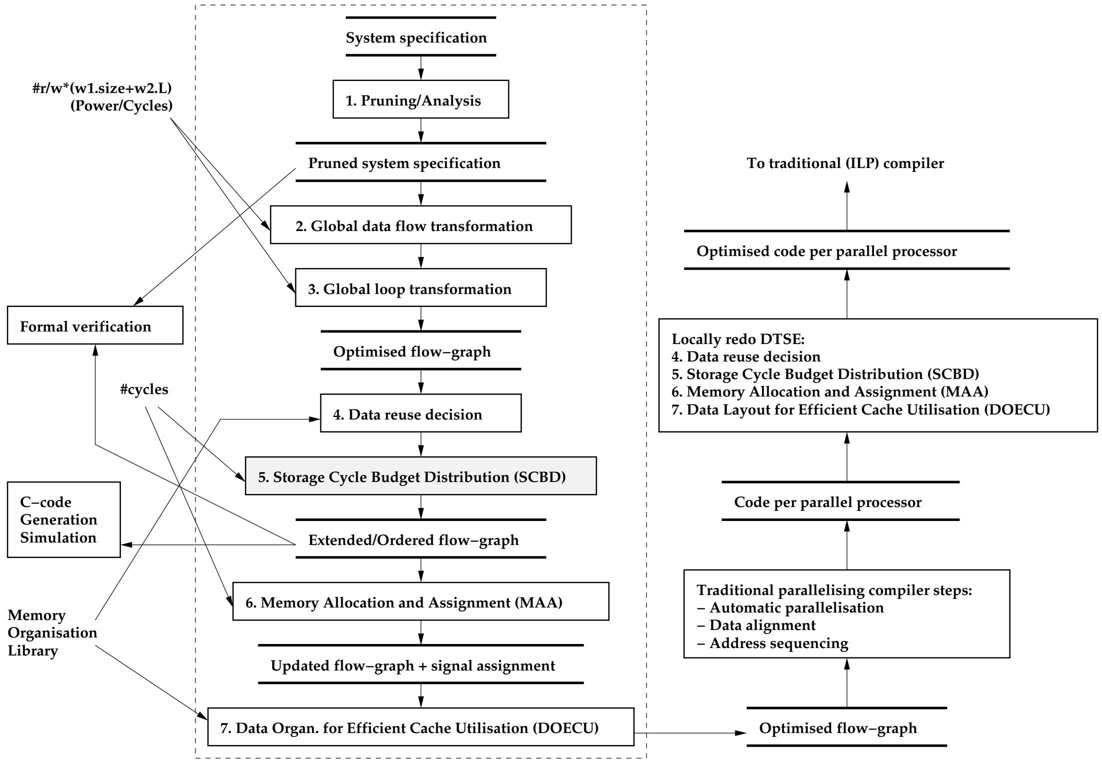


FIG. 1. *Variante de la méthodologie Data Transfer and Storage Exploration (DTSE) pour plates-formes parallèles (partiellement) programmables à mémoires et caches (partiellement) contrôlables et reconfigurables, appelée aussi méthodologie* Acropolis

Partant d'une spécification de système (System Specification en anglais) exprimée actuellement dans un dialecte C ou C++ **analysable statiquement**[2] – pouvant de ce fait encapsuler des composantes de *réutilisation de propriété intellectuelle* –, une première étape appelée *élagage* (Pruning en anglais) consiste à réécrire la spécification originale sous forme de trois couches hiérarchiques [**280**]: le contrôle du système (niveau 1), les nids de boucles (niveau 2) et les fonctions arithmétiques de base non transformables pouvant par exemple être déjà disponibles sur un (co-)processeur dédié (niveau 3). Cette étape est effectuée **manuellement** en partant de codes ANSI-C [**125**] [**62**] ou C++ [**250**]. Dans la mesure où les étapes ultérieures ne s'intéressent qu'à l'optimisation du niveau 2, il est possible de profiter de cette étape pour effectuer un premier compromis entre qualité et temps de conception en déclarant certains nids de boucles peu significatifs en tant que fonctions arithmétiques de base à ne pas réoptimiser[3]. Pour ce faire, la fonction de comptage et de rapport dynamique d'accès mémoire de l'outil ATOMIUM développé par IMEC [**67**] est à la méthodologie DTSE ce qu'un profiler est à l'optimisation de code conventionnelle. Le résultat de cette étape est une *spécification de système élaguée* (Pruned System Specification en anglais) exprimée actuellement dans un dialecte ANSI-C [**64**].

2. Pour les applications dynamiques, voir les extensions de la section suivante.

3. Agissant comme un filtre de complexité en entrée, l'élagage s'avère très utile pour effectuer des optimisations grossières mais néanmoins multi-critère de systèmes très complexes en un temps record.

Une deuxième étape dite *transformation du flot des données globales* (Global data–flow transformation en anglais) consiste à rechercher de nouveaux algorithmes pour le niveau 2 sans en modifier globalement les entrées et les sorties. En premier lieu, il s'agit de supprimer tout code et accès mort[4] – ce qui est automatisé par la fonction d'élagage dynamique de l'outil ATOMIUM développé par IMEC –; ainsi que de simplifier les chaînes de dépendances faisant intervenir un nombre de signaux inutilement grand. En second lieu, il s'agit de rechercher **manuellement** des simplifications basées sur une analyse algorithmique du niveau 2 de l'application permettant de mettre en évidence des techniques de calcul équivalentes mais plus adaptées voire même démontrées optimales [**194**] [**140**] pour certains ou l'ensemble des critères de performance. Ce processus intelligent peut susciter de longues recherches mais conduit généralement à des améliorations impressionnantes impossibles à obtenir autrement [**195**]. Il ne pourrait être que très difficilement et partiellement automatisable par l'utilisation de techniques de transformation de flot pour ses aspects les plus simples et des techniques de calcul symbolique [**107**] pour ses aspects les plus complexes, ce qui n'a pas été entrepris par IMEC.

Une troisième étape dite *transformation de boucles globales* (Global loop transformation en anglais) consiste à rechercher de nouveaux nids de boucles pour le niveau 2 sans en modifier globalement les entrées et les sorties. En premier lieu, il s'agit de chercher à fusionner un maximum de nids de boucles entre eux afin d'accroître les opportunités de réutilisation de données. En second lieu, il s'agit d'appliquer des transformations géométriques (dites de placement) améliorant conjointement la localité, la régularité ainsi que la réutilisation de données. Cette étape est en cours d'automatisation par le projet ACROPOLIS d'IMEC [**55**] [**57**] [**136**] [**56**]. En troisième lieu, il s'agit de rechercher un parcours géométrique permettant de générer automatiquement un nouveau code source de grande qualité [**83**] [**43**]. Cette étape est en cours d'études en collaboration avec l'ENSMP-CRI, Fontainebleau (France) et l'INSA, Lyon (France).

Les étapes 2–3 peuvent être guidées par des estimateurs de parallélisme et de critères de performance de haut-niveau, par exemple le produit du nombre d'accès en lecture ou écriture ($\#r/w$) multiplié par une somme pondérée d'une estimation de la surface ($\omega_1.size$) de silicium et du temps de calcul nécessaire ($\omega_2.L$), ce qui correspond à un compromis multi-critère consommation électrique / nombre de cycles (Power/Cycles en anglais). Le projet ACROPOLIS d'IMEC développe de tels estimateurs de haut-niveau en collaboration avec l'Université de Science et Technologie, Trondheim (Norvège) [**128**]. Le résultat des étapes 2–3 est un *graphe de flot de données optimisé* (Optimized Flow Graph en anglais).

Une quatrième étape dite *décision de réutilisation des données* (Data Reuse Decision en anglais) consiste à explorer la création de mémoires intermédiaires pour le stockage des données réutilisées dans les corps de boucles optimisés pour la localité. Dans le cas de la conception d'un processeur dédié, il s'agit de construire une hiérarchie de caches spécialisée pour réduire à la fois transferts et stockages. Dans le cas d'une architecture comportant déjà une hiérarchie de caches programmables tel le processeur Philips TriMedia [**216**], il s'agit d'introduire de nouvelles variables intermédiaires afin de permettre un contrôle total et explicite des mises à jour pour réduire principalement les transferts. Cette étape est effectuée **manuellement** en l'état actuel des projets ACROPOLIS et ATOMIUM mais pourrait faire l'objet d'une automatisation ultérieurement. Dans ce cas, cette étape devrait construire (dans le cas dédié) ou accepter les contraintes de (dans le cas programmable) l'organisation mémoire, stockée dans une librairie [**266**] (Memory Organisation Library).

Une cinquième étape dite *distribution du budget de cycles* (Storage Cycle Budget Distribution en anglais) consiste à répartir le nombre de cycles mémoire autorisés pour parcourir l'application ($\#cycles$) sur l'ensemble des nids de boucles afin de rechercher un ordonnancement global de coût minimum. Sur la base de cet ordonnancement faible coût et de l'espace d'exploration défini par la librairie d'organisation mémoire (Memory Organisation Library), une sixième étape dite *allocation*

4. Un code mort est une partie d'un programme jamais atteinte. Un accès mort est un élément mémoire jamais utilisé.

et placement mémoire (Memory Allocation and Assignment en anglais) construit une architecture mémoire spécialisée dans le cas reconfigurable (ou réutilise une architecture mémoire prédéfinie dans le cas contraire) pour y placer les différentes données du programme. Pour les applications dominées par les données, ces résultats sont suffisants pour permettre une évaluation (très) précise des différents critères d'optimisation [**268**] **avant** même de connaître tous les éléments constitutifs de la plate-forme, le partitionnement logiciel–matériel ou l'ordonnancement des instructions sur les diverses parties programmables. Nous reviendrons largement sur ces deux étapes qui sont traitées par le projet ACROPOLIS d'IMEC et constituent l'objet principal de ce travail de thèse.

Le résultat des étapes 4–5 est un *graphe de flots de données étendu ou ordonnancé* (Extended/Ordered Flow Graph en anglais). Cette représentation peut servir de base pour générer du code ANSI-C optimisé par exemple afin d'effectuer des simulations (ou des mesures) de performances plus détaillées (C-Code Generation Simulation en anglais). Il est également possible de développer des techniques de vérification formelle pour valider que cette nouvelle représentation est équivalente à la spécification originale [**236**] [**235**] (Formal Verification en anglais).

A partir d'une description de l'architecture cache également stockée dans la librairie d'organisation mémoire (Memory Organisation Library), une septième étape dite *organisation interne des données en mémoire* (Memory Data Layout Organisation en anglais) fusionne les données présentes dans un nid de boucle en un seul tableau pour pouvoir retravailler l'ordre interne de ses éléments afin de régulariser au maximum l'accès à la mémoire. Ceci permet (1) d'optimiser l'utilisation des lignes de cache chargées depuis la mémoire centrale; (2) d'éviter le dépassement intempestif de la capacité maximale du cache en adoptant une stratégie de repliement des données les unes sur les autres; et (3) d'éviter les conflits provenant de l'accès simultané de plusieurs données à une même ligne de cache [**137**]. Cette étape est développée par le projet ACROPOLIS d'IMEC, sans toutefois résoudre le point (2) en l'état actuel – ce qui nécessite un outil de *placement avec repliage* (In-Place Mapping en anglais) basé sur une analyse temporelle fine du problème pouvant également réduire la taille mémoire maximale [**63**] sur architecture mémoire reconfigurable. L'outil de *placement avec repliage* fait l'objet de développements par le projet ATOMIUM d'IMEC.

Dans le cas d'une architecture parallèle (partiellement) programmable, il convient de passer le (ou la partie programmable du) résultat de ces sept étapes à un compilateur–paralléliseur traditionnel. Il s'agit ensuite de réappliquer les étapes 4–7 pour le code présent sur chaque processeur (un seul dans le cas d'une parallélisation single program multiple data - SPMD) avant d'utiliser le compilateur–optimiseur local. C'est pourquoi nous parlons de technique de *pré-compilation*.

En pratique, il convient de raffiner la spécification C/C++ en reprenant les étapes 3–7 + compila–teur–paralléliseur + 4–7 + compilateur–optimiseur à plusieurs reprises afin d'évaluer d'autres alternatives **manuellement** et se convaincre de la qualité de sa solution finale, comme il est d'usage pour un compilateur traditionnel.

Nous venons de survoler en trois pages plus de dix années de recherche (1988–), impliquant actuellement une dizaine de chercheurs pour la méthodologie DTSE statique seule et environ vingt-cinq chercheurs si l'on inclut les extensions dynamiques et de gestion de mémoire physique, l'optimisation d'adresse et les premiers utilisateurs internes à IMEC. Afin d'approfondir la DTSE statique, le lecteur est invité à lire [**38**]. Des cours intitulés "How to write high–performance multimedia" sont également organisés régulièrement par le MTC, Leuven (Belgique) et peuvent être consultés dans la rubrique Training de www.imec.be. Afin de mieux cerner l'état des recherches dans ce domaine, nous mettons régulièrement à jour un rapport interne qui sans être franchement public n'est pas non plus totalement confidentiel [**35**] et peut être demandé auprès du Prof. Francky Catthor: catthoor@imec.be. Par ailleurs, nous exposons en Partie II - Chapitre 2 un exemple d'application manuelle de la DTSE sur un algorithme de détection de cavité provenant du domaine de l'imagerie médicale.

3. Vers un ordonnancement–placement économique

Étant donné un ensemble de n opérations à réaliser (O), des ressources disponibles (R) et des contraintes d'ordonnancement et de placement des opérations sur les ressources (C), l'ordonnancement–placement consiste à trouver un ordonnancement des tâches dans le temps (T) ainsi qu'une allocation de chaque tâche à une ressource. Dans le cas le plus général, il s'agit donc d'identifier n éléments *acceptables* dans un espace tri–dimensionnel $O \times T \times R$ (non borné). Cela dit, disposant d'un nombre de ressources **connues à l'avance** pouvant traiter indifféremment l'une ou l'autre des opérations, l'expression la plus classique de ce problème se limite à traiter exclusivement des contraintes de dépendances entre opérations pour optimiser la vitesse d'exécution – c'est–à–dire minimiser le temps d'exécution global (généralement discrétisé) – sur la base d'un graphe de flots de données mono-dimensionnel [**265**] [**278**]:

$$\begin{aligned} \Phi : O &\rightarrow T \times R \\ o &\rightarrow \Phi(o) = (t,r) \;\; \Phi^T(o) = t \;\; \Phi^R(o) = r \end{aligned}$$

$$(1) \qquad \exists ? \Phi_{min}, \qquad \max_{O}(\Phi^T_{min}(o)) = \min_{C(\Phi)} \max_{O}(\Phi^T(o))$$

ce qui est communément appelé *ordonnancement optimal sous contraintes de ressources* [**157**] [**207**] [**53**] [**130**]. Ce problème concentre la majorité des recherches dans le domaine de l'ordonnancement–placement informatique depuis quarante ans d'autant que sa NP-complétude [**88**] rend le processus d'invention de nouveaux modèles et de nouvelles heuristiques toujours plus rapides et plus proches de l'optimum extrêmement compétitif et (presque) sans fin [**94**] [**152**], ce qui tend à susciter des vocations chez nos chercheurs parmi les plus éminents, par exemple dans [**40**] [**215**] [**8**] [**31**] [**60**].

Intuitivement, nous savons pourtant qu'aller le plus vite possible n'est pas économique. Dans le domaine du multimedia, des expériences ont été récemment conduites par IMEC pour évaluer l'utilisation effective des ressources d'information et de communication par des stratégies d'*ordonnancement optimal sous contraintes de ressources*. Il s'avère que cette utilisation effective décroît avec la complexité des systèmes et passe sous le seuil des 10% pour les applications les plus récentes telles l'utilisation de la norme MPEG-4 en mode 3D [**299**] [**41**] [**141**] [**142**]. En Figure 2, nous illustrons cette (forte) dynamique de charge issue d'une optimisation de système pour la vitesse pure.

Afin d'introduire un ordonnancement–placement nettement plus économique en réservation (inutile) de ressources, il convient d'étudier un problème nettement moins classique d'*optimisation de capacité* – c'est–à–dire de minimisation du nombre maximum de ressources nécessaires à mener globalement communications et traitements. Pour ce faire, de nombreuses techniques existent fondées par exemple sur l'algèbre linéaire à Haute Performance [**300**], la programmation entière [**150**] ou la programmation logique par contraintes [**108**] [**155**]; la dernière offrant une souplesse d'utilisation très intéressante par exemple dans un contexte où des contraintes d'ordonnancement–placement supplémentaires telles la spécialisation soudaine des ressources à certaines opérations (panne partielle de traitement) où des erreurs de livraison (panne partielle de communication) doivent être introduites en cours d'exploitation. Cela dit, l'*optimisation de capacité* suppose que toute opération peut être effectuée indifféremment par toute ressource. Ceci est souvent le cas pour des activités de type logistique où tous les chargements s'opèrent auprès d'un grossiste centralisé et également pour l'ordonnancement–placement d'instructions sur une architecture parallèle parfaitement symétrique. Malheureusement, ces approches se généralisent (très) difficilement au scénario d'une logistique distribuée [**252**] [**15**] ou de l'ordonnancement–placement d'instructions sur une architecture parallèle non symétrique [**82**]. Imaginez que vous disposez d'un réseau de deux entrepôts (respectivement deux mémoires distribuées) disposant chacune d'une capacité de livraison locale (respectivement

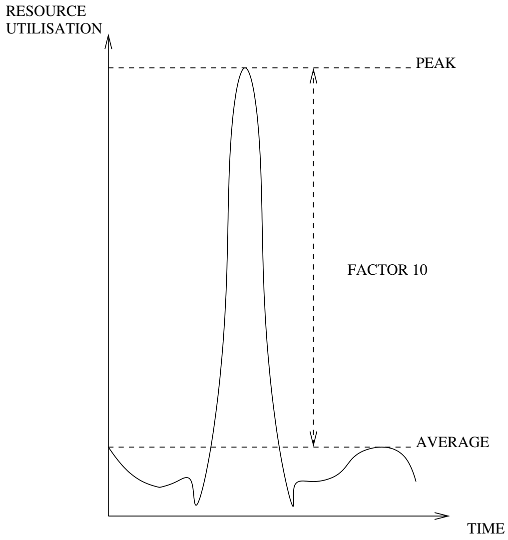


FIG. 2. *Utilisation effective des ressources d'information et de communication dans un système MPEG-4 utilisant une stratégie d'ordonnancement optimal sous contraintes de ressources.*

d'une bande passante locale) limitée. La vitesse et le coût d'exécution des deux livraisons ne seront (très) probablement pas les mêmes selon qu'elles concernent des marchandises (respectivement des données) placées dans le même entrepôts ou dans deux entrepôts distincts. **Afin d'optimiser précisément ces phénomènes, il convient de tenir compte exactement de quelles marchandises (respectivement quelles données) sont chargées (respectivement accédées) de façon concurrente afin d'optimiser globalement la capacité du système – c'est–à–dire économiser sur le nombre de ressources** [**292**]**.** Dans la mesure où cette optimisation **globale** tend à accroître considérablement la complexité du problème initialement considéré, il est essentiel de développer des techniques spécialisées pour y faire face efficacement. A notre connaissance, un seul système de programmation logique par contraintes commercial [**22**] pourrait résoudre de tels problèmes d'optimisation **globale** en logistique distribuée (respectivement en conception de systèmes parallèles non symétriques) pour peu qu'une formulation efficace du problème soit disponible.

L'objectif principal de ce travail de thèse consiste donc à compléter la formulation du problème d'ordonnancement–placement asymétrique entrepris dans [**292**] pour l'optimisation **globale** du transfert et du stockage des données. Une première contribution consiste à réécrire intégralement la formulation *multi-critère* et *flexible* décrite par Sven Wuytack sous forme d'algorithmes de graphe dynamiques (Chapitre 2 Section 1). Une deuxième contribution consiste à généraliser l'approche à un contexte d'identification *espace–temps* plus performant, ce qui est en pratique nécessaire pour permettre l'obtention de résultats quasi–optimum et une meilleure robustesse aux éléments extérieurs tels des erreurs de l'architecte du système ou la réutilisation de composants de *propriété intellectuelle* (Chapitre 2 Section 2). Une troisième contribution consiste à introduire des techniques d'approximation *exacte*, *multi-étage* et *divise–et–règne*, ce qui est en pratique nécessaire pour assister *interactivement* l'architecte du système et/ou permettre l'intégration en boucle fermée avec des outils d'un plus haut niveau d'abstraction [**49**] (Chapitre 2 Section 3). Une quatrième contribution consiste à démontrer la quasi-optimalité de l'approche pour résoudre l'ordonnancement–placement du Segment Protocol Processor d'Alcatel ainsi que d'introduire des éléments formels et intelligents permettant potentiellement d'atteindre et en tout cas de vérifier de tels niveaux de résultats dans le cas général (Chapitre 2 Section 4). Sur la base de ces quatre consolidations majeures, une cinquième et dernière contribution consiste à étudier l'extensibilité de notre approche en direction des *plates-formes* virtuellement partagées (*c.a.d* à mémoires logiquement partagées mais physiquement distribuées) sujettes à des contraintes espace–temps fortes de type streaming [**264**] et mémoire synchrone de type SDRAM (Chapitre 2 Section 5).

4. Ordonnancement–placement multi–critère et flexible

Sur la base des travaux de Paulin et Knight à la fin des années 1980s en *optimisation* **globale** *de capacité* [**213**] [**212**] [**214**] et du transfert de ces techniques pour la *minimisation du nombre d'unités de traitement* dans certains circuits multimedia dédiés conçus par Philips Research [**276**] [**277**] [**274**] dans la première moitié des années 1990s, c'est en 1996 que Sven Wuytack remarque les faiblesses de l'approche en ce qui concerne le transfert et le stockage des données [**294**], notamment en nombre et coût des mémoires distribuées dont l'optimisation **globale** est ignorée par la technique décrite dans [**150**]. C'est alors qu'il pose le principe de base de sa formulation: **afin d'optimiser précisément ces phénomènes, il convient de tenir compte exactement des données accédées de façon concurrente afin d'optimiser globalement la capacité du système – c'est–à–dire minimiser la bande passante** [**292**]. En se limitant à la conception d'un système embarqué disposant de plusieurs unités fonctionnelles connectées à une mémoire partagée, il propose la nouvelle technique d'*optimisation de bande passante* (Storage Bandwidth Optimisation en anglais) présentée en Figure 3.

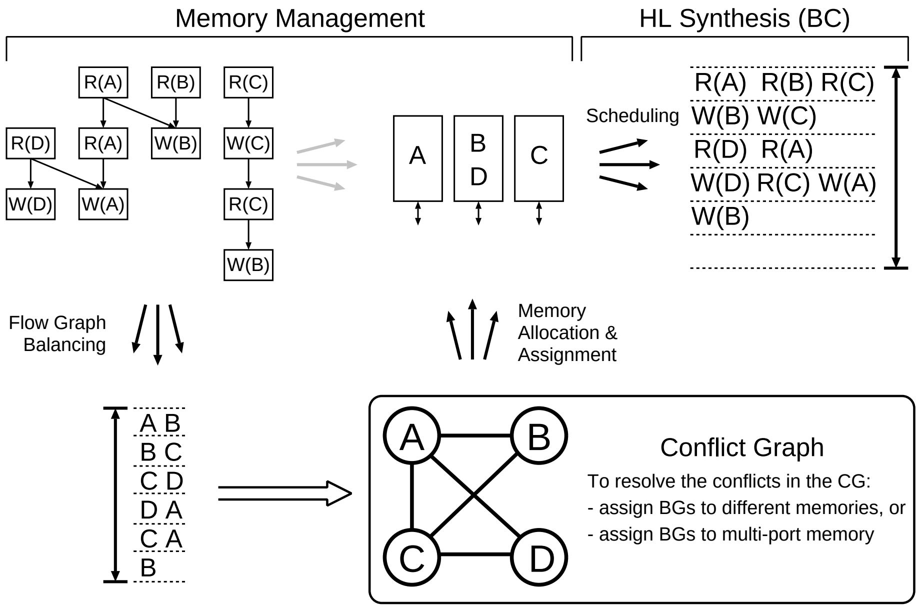


FIG. 3. *Storage Bandwidth Optimisation (SBO) par Sven Wuytack (1996)*

Partant d'un graphe de flot de données mono-dimensionnel (Control–Data Flow Graph en anglais) [**265**] [**278**] et d'un budget temps-réel exprimé en nombre de cycles mémoire **sans** pour autant connaître la moindre ressource à l'avance, une première étape appelée *équilibrage de graphe de flot de données* (Flow Graph Balancing en anglais) [**294**] consiste à rechercher un ordonnancement partiel de faible bande passante respectant le budget temps-réel alloué. Afin de tenir compte **exactement** de quelles données sont accédées de façon concurrente, une arête est ajoutée au *graphe* **global** *des conflits d'accès à la mémoire partagée* (Conflict Graph en anglais) à chaque décision de rendre l'accès à telle ou telle donnée parallèle. Sur la base de ce graphe **global** des conflits d'accès à la mémoire partagée représentant les contraintes d'ordonnancement partiel et de placement physique autorisés, il devient possible d'allouer précisément des mémoires connectées à un bus centralisé et d'allouer les différentes données dans l'une ou l'autre de ces mémoires (Memory Allocation and Assignment en anglais). Une fois l'architecture mémoire optimisée connue, il devient aisé d'utiliser les outils de synthèse de haut–niveau (HL Synthesis en anglais) conventionnels pour concevoir un circuit dédié (respectivement un compilateur embarqué pour programmer un processeur embarqué), qui proposera l'ordonnancement final (Scheduling en anglais) du système. Le lecteur remarquera que dans l'exemple présenté en Figure 3, un ordonnancement qui semblait nécessiter une bande passante de deux accès par cycle mémoire au maximum au niveau de l'*équilibrage de graphe de flot de données* peut présenter un *graphe* **global** *des conflits d'accès à la mémoire partagée* de nombre chromatique supérieur à deux et donc nécessiter une architecture de plus de deux mémoires physiques en pratique. Ceci illustre la faiblesse des approches d'*optimisation de capacité* classiques et justifie l'introduction d'un algorithme d'ordonnancement spécialisé pour architectures mémoires parallèles asymétriques.

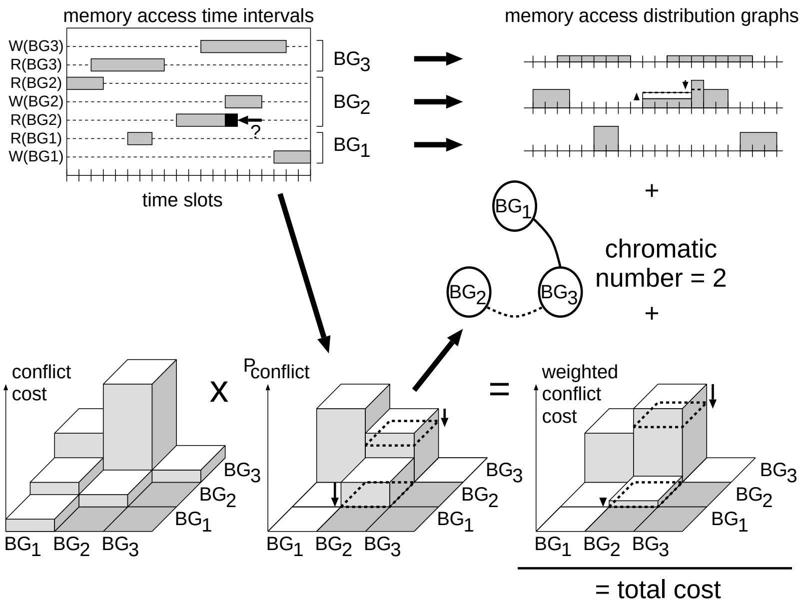


FIG. 4. *Conflict–Directed Ordering (CDO) par Sven Wuytack (1996)*

Afin de résoudre précisément le problème de l'*équilibrage de graphe de flot de données*, Sven Wuytack propose également un algorithme appelé *ordonnancement guidé par les conflits* (Conflict–Directed Ordering ou CDO en anglais) présenté en Figure 4. Avant toute chose, les données de l'application sont partitionnées **manuellement** en *groupes de base* (Basic Group en anglais) [**290**] par définition indivisibles entre plusieurs mémoires. A noter que pour des raisons de facilité, il est pratique de choisir les groupes de base comme étant les tableaux de l'application[5]. Les opérations du graphe de flot de données (mono-dimensionnel) deviennent alors des accès en lecture ou écriture à des groupes de base, par exemple écriture sur le troisième groupe de base W(BG3) et lecture du troisième groupe de base R(BG3). Connaissant le budget temps-réel en nombre de cycles mémoire alloués, les dépendances entre opérations représentées dans le graphe de flot de données (mono-dimensionnel) permettent de déterminer la fenêtre d'ordonnancement au plus tôt et au plus tard (Memory Access Time Interval en anglais) de chaque opération. A partir de cette analyse détaillée de l'espace des ordonnancements acceptables, il devient possible d'evaluer la *probabilité de conflit pour chaque paire*

5. Nous avons validé que ce partitionnement trivial conduit à des résultats très convenables pour 3 des 4 applications réelles illustrant cette thèse. Cet aspect est rediscuté en conclusion.

d'accès (Pairwise Probability of Conflict Computation en anglais). Ce calcul de probabilité peut devenir très complexe [**274**] mais l'expérience montre qu'un calcul de probabilité parfaitement rigoureux est en pratique trop coûteux en temps de calcul pour permettre la résolution de problèmes réels de moyenne et grande taille. Pour cette raison, Sven Wuytack suppose une répartition homogène à l'intérieur de l'intervalle au plus tôt au plus tard, ce qui rend ce calcul $O(1)$. Il peut alors construire en seulement $O(n^2)$ la matrice des probabilités de conflit, ce qui est indiqué par la première flèche noire en Figure 4. Pour chaque probabilité non nulle dans la matrice des probabilités de conflit, il faut introduire une arête entre les deux groupes de base représentés dans le graphe **global** des conflits d'accès à la mémoire partagée, ce qui est indiqué par la deuxième flèche noire en Figure 4. Disposant de cette analyse **globale**, il devient possible d'estimer précisément le coût relatif d'un conflit par rapport à un autre. Pour ce faire, trois principaux cas doivent être considérés:

(1) un conflit d'accès au même groupe de base (Self-Conflict en anglais) est associé à un coût très important car il nécessite l'utilisation de mémoires à plusieurs ports dont le prix et la consommation électrique sont extrêmement défavorables;

(2) un conflit d'accès à deux groupes distincts impliqué dans le sous-graphe chromatique[6] du graphe **global** des conflits d'accès à la mémoire partagée (Chromatic Conflict en anglais) est associé à un coût important car il contribue à l'augmentation du nombre de mémoires physiques et donc à la complexité de l'architecture mémoire, interconnexion incluse;

(3) un conflit d'accès à deux groupes distincts non impliqués dans le sous-graphe chromatique du graphe **global** des conflits d'accès à la mémoire partagée est associé à un coût de base car même si il n'aggrave pas directement l'architecture mémoire, il réduit l'espace des placements admissibles et rend par exemple plus difficile l'équilibrage de la taille des mémoires et la réduction de la taille du réseau d'interconnexion dans un but de faible consommation.

Des extensions de cette fonction coût pondérée à trois termes pour mieux tenir compte de l'importance relative de certains groupes de base (par exemple après une analyse utilisant ATOMIUM [**67**]) ou décourager l'introduction de conflits entre des groupes de base dont les tailles des éléments constitutifs sont complémentaires [**73**] et donc bien adaptés à un regroupement dans une même mémoire sont en cours d'études [**293**]. Nous ne les aborderons pas dans cette thèse car elles en compliqueraient l'exposé sans apporter des contraintes nécessitant une révision fondamentale des contributions exposées, puisque celles-ci n'impliquent en pratique qu'une modification $O(1)$ du module de calcul du *coût individuel du conflit pour chaque paire d'accès* (Weighted Conflict Cost en anglais). Pour l'instant, nous admettons qu'il existe un processus non trivial permettant de passer du graphe **global** des conflits d'accès à la mémoire partagée à la matrice des coûts individuels des conflits pour chaque paire d'accès. Le produit du coût individuel par la probabilité de conflit donne le coût de conflit pondéré (Weighted Conflict Cost en anglais). La somme des éléments de la matrice des coûts de conflit pondérés donne la fonction coût **globale** que nous utiliserons pour guider notre exploration (Total Cost en anglais). Pour une technologie et un budget temps-réel exprimé en cycle mémoire donnés, le lecteur remarquera que les aspects consommation électrique et surface de silicium peuvent être pris en compte de façon précise dans cette fonction puisque la taille des mémoires, le nombre de ports et le nombre d'accès y sont déterminés exactement, ce qui est suffisant pour une modélisation consommation–surface de haut-niveau *cf.* [**38**] Chapitre 4. Nous discutons dans les paragraphes suivants la façon de gérer des critères supplémentaires.

Une fois cette phase d'initialisation achevée et de nouveau dans un souci de facilité, le processus d'exploration consiste à identifier l'extrêmité d'intervalle au plus tôt au plus tard de l'accès à la

6. Le sous-graphe chromatique d'un graphe est l'ensemble des noeuds et des arêtes faisant partie d'au moins un sous-graphe dont le nombre chromatique est égal au nombre chromatique du graphe complet.

mémoire partagée dont le retrait de l'espace de recherche[7] réduit au maximum la fonction coût **globale**, ce qui constitue une technique d'exploration gloutonne avec anticipation d'une étape (Greedy Exploration with One-Step Look-Ahead en anglais). Cette technique d'exploration est illustrée par le point d'interrogation (**?**) de la Figure 4 qui indique un exemple d'anticipation – intensivement répété par l'algorithme CDO – du coût induit par la réduction d'une des extrémités d'un des intervalles au plus tôt au plus tard à un accès. Cette évaluation modifie nécessairement le graphe (espace–temps) de distribution des accès en mémoire (voire les petites flèches d'anticipation) et en conséquence les poids de probabilité et les coûts pondérés (de nouveau illustré par des petites flèches d'anticipation). Dans le cas (rare) où le sous-graphe chromatique est également modifié, un grand nombre de coûts de conflit peuvent soudainement évoluer[8]. Nous démontrons en Partie II - Chapitre 4 que ce processus comporte une complexité de $O(\frac{1}{2}n^4m)$ exprimée en calcul individuel de probabilité de conflit pondérée pour une paire d'accès (appelé abusivement de nouveau Pairwise Probability of Conflict Computation puis PPCC en anglais) où n est le nombre d'accès à la mémoire et m est la taille moyenne de la fenêtre au plus tôt au plus tard au moment de l'initialisation.

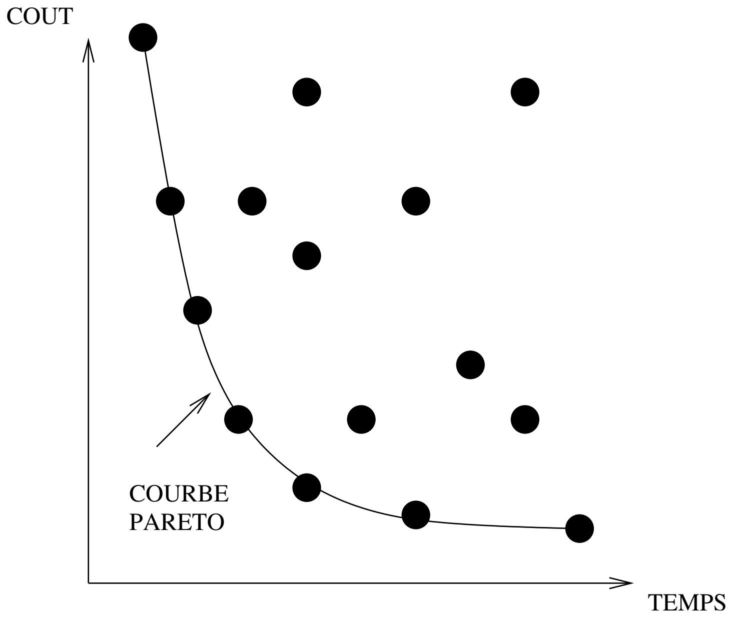


FIG. 5. *Courbe Pareto–optimale obtenue après l'exécution de CDO pour sept budgets temps-réel différents. Les points en dehors de la courbe Pareto–optimale correspondent à des solutions de moins bonne qualité que CDO élimine automatiquement.*

Pour une technologie et un budget temps-réel exprimé en nombre de cycles mémoire donnés, l'algorithme CDO détermine donc un graphe **global** de conflits acceptable correspondant à une optimisation multi-critère entre la bande passante processeur–mémoire imposée par le budget temps-réel, la consommation électrique via le poids des coûts de type (1), α, la complexité du réseau d'interconnexion et la consommation électrique via le poids des coûts de type (2), β[9], et la taille de l'espace

7. nous appelons ce retrait une (opération de) réduction dans le reste cette thèse.

8. ce qui n'est pas implémenté dans l'algorithme CDO original de Sven Wuytack mais que nous considérons systématiquement dans cette thèse pour son impact positif sur la qualité des solutions.

9. Pour un objectif de consommation électrique pur, il existe généralement un nombre de mémoires optimum pour réduire la consommation liée aux accès mémoire sans trop augmenter la consommation liée à l'interconnexion. Pour un objectif purement économique, un nombre minimum de mémoires sera préféré.

des placements admissibles via le poids des coûts de type (3), γ[10] (ce qui influe directement sur la liberté laissée aux autres outils notamment pour réduire la taille des mémoires et donc la surface de silicium). En faisant maintenant varier le budget temps-réel, il est possible de construire des courbes Pareto–optimales [**203**] pour une optimisation multi-objectif [**206**] [**298**] [**205**] [**301**] consommation électrique, complexité du réseau d'interconnexion, surface de silicium et temps, tel que présenté en Figure 5. Le prix du système étant largement déterminé par la technologie utilisée – *c'est–à–dire* grosso-modo le cycle mémoire – le calcul de plusieurs courbes Pareto pour plusieurs technologies disponibles peuvent constituer un outil d'aide à la décision [**102**] [**148**] répondant à l'ensemble des critères décrits en Section 1.1.

Le lecteur remarquera que cette aide à la décision se base sur une analyse constructive, ce qui induit une finesse d'estimation des différents critères de performance sans commune mesure[11] avec les environnements commerciaux existants tel Cierto Virtual Component Co-Design (VCC) de Cadence Design Systems [**32**] [**33**] dont l'objectif est d'obtenir une estimation de l'ordre de grandeur uniquement[12] (erreur relative inférieure à 500%). Comparé aux (rares) prototypes de recherche permettant une analyse globale de plusieurs critères avec une finesse comparable tel MMAlpha [**230**] de l'IRISA, Rennes (France), nous proposons une assistance complète à l'optimisation multi-critère **sans** restriction de format telle l'assignation unique [**34**] avec dépendances uniformes uniquement [**262**]. Tout ceci reste (très) différent du développement d'ordonnanceur pour chacun des critères concernés, tels par exemple la taille mémoire [**223**] ou la vitesse pure [**40**], à partir du langage en assignation unique comme Alpha.

Enfin, dans le cas de plusieurs systèmes représentables par un graphe de flot de données mono-dimensionnel (Control–Data Flow Graph en anglais) exploitant une mémoire partagée commune, des stratégies de type divise-et-règne [**17**] [**167**] (divide–and-conquer en anglais) ont été récemment introduites par le projet ACROPOLIS d'IMEC [**29**] [**28**], ce qui est en général nécessaire à l'optimisation des applications multimedia réelles dont la totalité des nids de boucles significatifs sont difficilement fusionnables en un seul nid de boucle et donc un seul flot de données mono-dimensionnel. Cette extension de CDO appelée *distribution du budget de cycles* (Storage Cycle Budget Distribution en anglais) est la cinquième étape de la méthodologie Data–Transfer and Storage Exploration (DTSE) introduite en Section 1.2.

Pour toutes les raisons évoquées ci–dessus, nous sommes convaincus du bien fondé et de la valeur ajoutée considérable qu'apporte l'approche CDO à la conception de systèmes multimedia embarqués, ce qui justifie largement l'investissement des projets ATOMIUM et ACROPOLIS d'IMEC dans ce

10. Dans les illustrations présentées dans cette thèse, nous recherchons une solution (très) économique minimisant le nombre de mémoires et le système d'interconnexion en interdisant l'utilisation de mémoires multi–port et en préservant autant de liberté que possible pour minimiser la surface de silicium dans les étapes ultérieures (ce qui revient à hiérarchiser les différentes composantes de la fonction coût: $\alpha >> \beta >> \gamma$). Cette utilisation de CDO constitue l'application industrielle la plus simple et (probablement) la plus intéressante de notre environnement ACROPOLIS.

11. L'erreur d'estimation de consommation électrique et de surface induite par les outils SCBD et MAA de l'environnement ACROPOLIS est (vraiment) minime pour la partie mémoire du système en se basant directement sur les meilleurs modèles technologiques disponibles auprès de nos partenaires industriels [**268**]. En ce qui concerne la totalité du système, la partie interconnexion et traitements est actuellement négligée mais nous envisageons des collaborations par exemple avec Pennsylvania State University [**281**] pour y remédier. Globalement, l'erreur d'approximation reste inférieure à 10% en pratique pour les applications dominées à 90% et plus par les données. De plus, cette erreur n'a biensûr aucun effet sur notre ordonnancement qui reste guidé uniquement par la haute précision de notre estimation mémoire. Les effets sur le placement se limitent à des cas où celui-ci rechercherait des solutions exotiques où un compromis mémoire – interconnexion – unités de calcul est nécessaire par exemple en cherchant une diminution du coût global du système en introduisant un nombre (très) important de petites mémoires. Pour l'instant, nous ne nous autorisons pas ces considérations exotiques qui sont d'ailleurs loin de faire l'unanimité chez les industriels.

12. ce niveau de précision est également typiquement le cas pour les outils d'analyse de performance de logiciels distribués [**196**].

domaine depuis cinq années. Cependant, au vu des perspectives d'utilisation décrites au paragraphe précédent, nous nous orientons clairement vers une utilisation nécessitant des centaines voire des milliers d'appels à l'algorithme CDO pour pouvoir faire face au moindre scénario d'aide à la décision industriel. De plus, afin de permettre une certaine interactivité avec cet outil d'aide à la décision, les contraintes de vitesse d'exécution de ces centaines voire de ces milliers d'appels à l'algorithme CDO sont encore plus fortes que ce qu'elles peuvent être aujourd'hui dans un contexte de conception automatique de circuits dédiés en l'espace de quelques jours par l'environnement ATOMIUM d'IMEC. Nous reviendrons longuement sur les techniques d'accélération de CDO, qui constituent la contribution scientifique principale de cette thèse. Ces techniques permettent également de dégager un complémentarité forte entre le projet ACROPOLIS qui s'intéresse aujourd'hui principalement aux extensions vers le monde programmable et l'approximation interactive et le projet ATOMIUM qui s'intéresse plutôt au développement d'un noyau CDO capable de gérer un maximum de spécificités industrielles (technologie mémoire, scénario d'utilisation, etc.) au prix de temps de conception allongés.

Remarque sur l'optimisation conjointe bande passante–nombre d'unités de traitement

Il va de soi que l'*optimisation de bande passante* est plus importante que la minimisation du nombre d'unités de traitement pour l'*optimisation* **globale** *de capacité* des systèmes embarqués dominés par les données. Dans le cas extrême où l'optimisation du transfert et du stockage (DTSE) et l'optimisation des traitements sont effectués par deux outils totalement séparés, l'*optimisation du transfert et du stockage (DTSE)* doit donc se dérouler **avant** la *minimisation du nombre d'unités de traitement*, qui doit en accepter les contraintes. Dans le cas d'un environnement d'*optimisation* **globale** *de capacité* intégré destiné au multimedia, le poids de l'*optimisation de bande passante* doit être supérieur à celui de la *minimisation du nombre d'unités de traitement*. Cette remarque sur la bonne organisation d'une méthodologie de conception des systèmes multimedia dédiés est largement discutée dans [**38**] car son impact sur la hiérarchisation des méthodes et outils de la communauté Electronic Design Automation (EDA) est (très) important. Elle fait également l'objet d'un projet de recherche conjoint entre IMEC et Philips-NATLAB pour le couplage des outils ACROPOLIS [**35**] et PHIDEO [**264**] depuis juillet 2000 [**184**].

CHAPITRE 2

Principales contributions

1. Ré-écriture sous forme d'algorithmes de graphe dynamiques

Quelques algorithmes de graphe de base – tels la recherche du plus court chemin [**95**] ou la recherche d'une fermeture transitive [**165**] dans un graphe orienté – sont utilisés intensivement par de nombreuses applications réelles, tout particulièrement dans les domaines des Réseaux, des Télécom–munications, de la Conception Assistée par Ordinateur (CAO) et des outils d'indexation et de recherche [**193**] (dont les développements récents concernent essentiellement la technologie Internet). L'approche traditionnellement utilisée consiste à développer une librairie de graphes (très) performante. Le meilleur exemple en est le développement de la librairie C/C++ LEDA par le groupe algorithmes et complexité du Max-Planck Institut für Informatik (MPI), Saarbrücken (Allemagne)[1], qui devrait être prochainement intégrée à la Standard Template Library (STL) [**169**]. La disponibilité gratuite d'une telle librairie standard représente un gain de productivité d'autant plus important qu'elle contraint l'utilisateur à ré–utiliser les constructions les plus élégantes [**87**] [**45**] d'un langage [**250**] à mon avis trop complet pour ne pas devenir chaotique si l'on ne se fixe pas des règles strictes de programmation par avance [**163**] [**164**], un peu comme cela fut le cas une demi–décennie plus tôt lors de l'introduction des interfaces LAPACK [**9**] pour l'algèbre linéaire à Haute Performance en langage Fortran 77 [**189**].

Malheureusement, et nous ne le savons que trop bien à IMEC [**134**], séparer le développement d'une application du développement de sa librairie de base conduit généralement à des pertes significatives en performance. Concernant la librairie C/C++ LEDA, le Département d'Informatique de l'Université de Halle remarque depuis 1995 [**104**] [**2**] [**74**] qu'une optimisation croisée application–graphes peut bien souvent réduire une complexité initiale d'ordre n en une complexité finale d'ordre polylogarithmique (grosso–modo $ln(n)$), ce qui conduit à une extension de LEDA très intéressante [**3**] [**110**]. C'est à la fois sur la base de ma méfiance envers C++ exprimée au paragraphe précédent et à la lumière des contributions alléchantes de l'Université de Halle que je propose à Francky Catthoor [**178**] puis à l'IRISA [**192**] et à la division DESICS d'IMEC [**183**] de tenter (entres autres) de réécrire l'ordonnancement–placement multi-critère sous forme d'algorithmes de graphe dynamiques.

Ces résultats sont aujourd'hui publiés principalement dans l'article [**177**] présenté en Partie II – Chapitre 4. Dans la mesure où cet article tend à favoriser le passage de la théorie à l'application[2] parfois au détriment d'une description technique détaillée, nous reprenons dans les sections suivantes les deux clés de voûte de la réécriture de l'ordonnancement–placement multi-critère sous forme d'algorithmes de graphe dynamiques.

1.1. Calcul dynamique des intervalles au plus tôt au plus tard. Tel que présenté en Chapitre 1 Section 4 de cette partie, une composante importante de la phase d'initialisation de l'algorithme CDO proposé par Sven Wuytack consiste à déterminer la fenêtre d'ordonnancement au plus tôt et au plus tard (Memory Access Time Interval en anglais) de chaque opération. Pour chaque étape de

1. Voir `http://www.mpi-sb.mpg.de/LEDA/` pour plus d'information.
2. ce qui est une exigence générale pour toute recherche IMEC.

l'algorithme[3], ces fenêtres d'ordonnancement au plus tôt et au plus tard doivent être calculées pour chaque anticipation et recalculées de nouveau lorsque la meilleure réduction de l'espace de recherche est connue et peut être appliquée (car il serait trop coûteux en mémoire de mémoriser le résultat de toutes les anticipations). En tout, ce calcul doit donc être effectué exactement $nm+1$ fois.

Or, le calcul de l'intervalle au plus tôt au plus tard de n'importe quel noeud d'un graphe orienté peut se réécrire comme une suite de calculs de chemins critiques, plus précisément comme le maximum des chemins critiques de toutes les racines à ce noeud pour la valeur au plus tôt (respectivement comme le minimum du budget temps réel minoré du chemin critique de toutes les feuilles à ce noeud pour la valeur au plus tard). En conséquence, l'algorithme de chemin critique dynamique proposé par l'Université de Halle dans [**95**] s'applique à notre problème et permet d'en faire chuter la complexité en partageant les résultats intermédiaires obtenus pour certains noeuds afin d'en accélérer l'obtention pour d'autres noeuds.

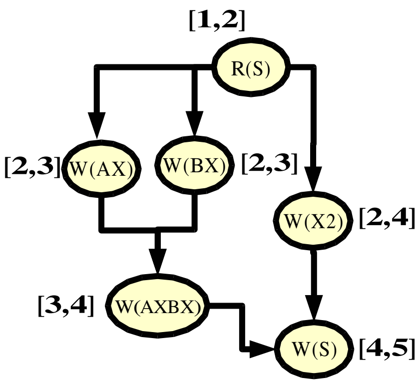


FIG. 1. *Initialisation du graphe de flot de données étendu (Extended Flow Graph, EFG) utilisant les routines de parcours en largeur* `asap` *et* `alap`.

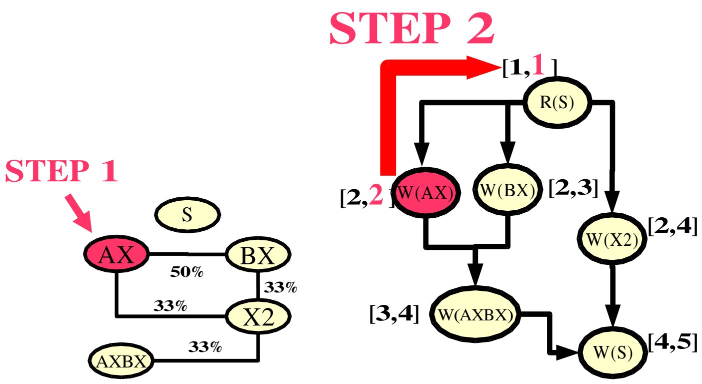


FIG. 2. *tape 1: sélection sur le graphe global des conflits d'accès à la mémoire partagée (Extended Conflict Graph, ECG). tape 2: réduction dynamique du graphe de flot de données étendu (EFG) utilisant l'algorithme dynamique* `dynamic_alap`.

Dans la mesure où CDO effectue ce calcul pour tous les noeuds lors de l'initialisation puis pour des noeuds pris individuellement (selon leur réductibilité) lors de chaque anticipation et pour la réduction, nous avons finalement préféré redévelopper deux jeux de routines spécialisées `asap` et

3. pour une présentation plus détaillée de l'algorithme CDO, voir Partie II - Chapitre 4 Section 5.3.

`alap` (pour l'initialisation) et `dynamic_asap` et `dynamic_alap` (pour les anticipations/réductions), ce qui revient grosso-modo à inliner les nombreux appels à la routine de chemin critique dynamique en une seule routine et permettre des optimisations plus puissantes qu'une réécriture sous forme d'appels à la librairie dynamique de l'Université de Halle. Tel que présenté en Figure 1, l'initialisation est implémentée en exactement $2n$ opérations en appliquant un parcours en largeur du graphe et de son complément. De même, en diffusant en largeur du graphe ou de son complément avec arrêt dès que la propagation ne provoque plus de mises à jour effectives, la complexité de la réduction d'intervalle au plus tôt ou au plus tard tombe statistiquement du nombre de noeud $O(n)$ à seulement une petite partie des voisins du noeud d'ordre environ $O(ln(n))$ [4]. En Figure 2, nous illustrons que la réduction de l'intervalle au plus tôt au plus tard de l'accès en écriture au groupe de base `AX` – *i.e.* W(AX) – nécessite une réduction semblable de l'accès en lecture au groupe de base `S` – *i.e.* R(S). Deux opérations au lieu de $n = 6$ suffisent dans ce cas à une remise à jour complète du graphe de flot de données étendu. Cette implémentation représente déjà une économie de temps d'exécution d'un ordre de grandeur et plus sur les problèmes de moyenne et de grande taille (*ex.* $\frac{156)}{ln(156)} = 30.89$ dans le cas du Segment Protocol Processor d'Alcatel).

1.2. Calcul dynamique des anticipations et des réductions. Nous avons vu en Sous-Section 1.1 de cette section comment gagner un ordre de grandeur en rapidité sur le calcul dynamique des intervalles au plus tôt au plus tard. Le lecteur attentif aura certainement remarqué qu'il n'est pas très utile de mener la mise à jour au plus tôt au plus tard en $O(ln(n))$ au lieu de $O(n)$ si celle-ci est nécessairement suivie d'une mise à jour $O(n^2)$ par la méthode des matrices de Sven Wuytack, ce qui absorbe de toute façon toute la masse de calcul. Dans la mesure où les éléments perturbés dans les matrices sont très étroitement liés aux accès dont les intervalles au plus tôt au plus tard ont été modifiés par cette mise à jour dynamique, il est pourtant possible de développer un calcul dynamique consistant à effectuer une série de mises à jour uniquement sur les sous-matrices (symétriques) correspondant aux accès modifiés et à leurs voisins de conflit.

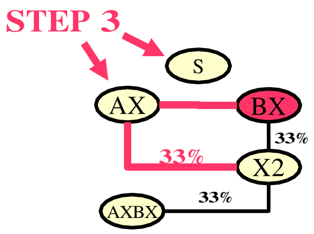


FIG. 3. *tape 3: Mise à jour dynamique des arrêtes du graphe global des conflits d'accès à la mémoire partagée (ECG) en ne visitant que les voisins de conflit.*

Cette idée est excellente mais n'est pas implémentable à l'aide d'une matrice seule car il s'agit clairement d'une technique de calcul matriciel creux **[72]** **[7]** **[282]**. **Nous proposons donc d'associer à chaque groupe de base du graphe global des conflits d'accès à la mémoire partagée la liste de ses opérations d'accès. Partant d'une opération quelconque, il suffit alors de visiter les voisins de conflit du groupe de base puis de déterminer la liste des opérations d'accès concernées pour connaître de façon simple une liste réduite de conflits potentiellement modifiés.** En Figure 3, nous illustrons une telle mise à jour dynamique des arrêtes du graphe global des conflits d'accès à la mémoire partagée en ne visitant que les voisins de conflit (tape 3). Afin de mettre à jour les groupes de base `AX` et `S` réduits précédemment, seules les deux arrêtes `AX-BX` et `AX-X2` sont modifiées. Au total,

4. nous n'avons pas souhaité savoir la complexité exacte de cette implémentation car les résultats de l'Université de Halle nous paraissent (grandement) suffisants pour motiver l'approche.

l'ensemble de la réduction dynamique permet de réduire la complexité de la méthode des matrices de Sven Wuytack de $n+n^2 = 42$ à $2.2 = 4$. Cette performance supplémentaire ne peut être révélée par l'analyse de complexité car il n'existe pas de borne au nombre de voisins de conflit d'un groupe de base. On notera cependant que ce nombre de voisins décroît avec le budget temps-réel alloué. De ce fait, plus le problème est difficile, plus notre réécriture dynamique (LCDO) se distingue de l'algorithme original de Sven Wuytack (CDO). Par ailleurs, l'exemple de la Figure 3 illustre un résultat intéressant. Dans le cas où l'intervalle réduit ([2,3] de `AX`) est inclus au sens large dans l'intervalle de voisins de conflit ([2,3] de `BX` et [2,4] de `X2`), la probabilité de conflit reste constante après réduction – ce qui fait qu'il est inutile de la recalculer en pratique.

$$(2) \qquad [a,b] \subseteq [A,B] \quad \Rightarrow \quad \frac{b-a+1}{(b-a+1)\times(B-A+1)} = \frac{b-a}{(b-a)\times(B-A+1)} = \frac{1}{B-A+1}$$

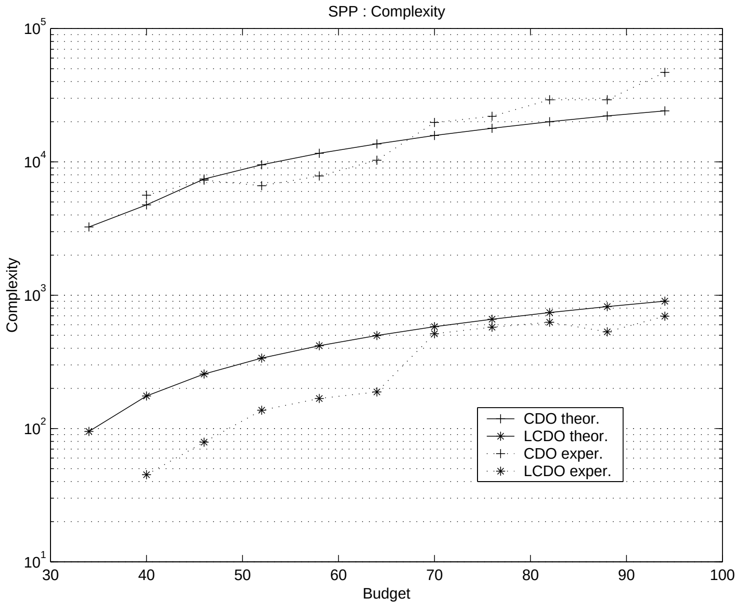


FIG. 4. *Corrélation entre complexités théoriques et temps d'exécution réels pour l'algorithme de Sven Wuytack (CDO) et sa réécriture dynamique (LCDO) appliqués à la conception du Segment Protocol Processor d'Alcatel.*

1.3. Conclusion. Nous venons de réécrire l'algorithme de Sven Wuytack sous forme d'algorithmes de graphe totalement dynamiques, ce qui permet de passer d'une complexité d'exploration globale gloutonne de $O(n^4m)$ à une complexité de seulement $O(n^3mln(n))$ – *ex.* gain de l'ordre de 30.89 dans le cas du Segment Protocol Processor d'Alcatel. Cette implémentation se limite à une modification très faible de l'algorithme et des modèles de calcul proposés initialement par Sven Wutack, en recherchant des gains faciles mais pas trop compliqués par une analyse conjointe complexité-qualité-effort[5]. Nous expliquons plus en détail en Partie II - Chapitre 4 par exemple pourquoi la recherche d'algorithmes dynamiques impliquant plusieurs anticipations/réductions représenterait peu de gain pour beaucoup de développement. Nous illustrons en Figure 4 que la complexité $O(n^3mln(n))$ est (très) pessimiste pour les budgets temps-réel faibles. Le facteur d'accélération peut facilement dépasser 100 dans ce cas alors que la formule de complexité prévoit 30.89. Afin de permettre une

5. ce qui va bien au-delà de l'astuce de codage informatique.

technique de calcul (encore) plus efficace, il nous semble judicieux d'introduire des modèles à Haute Performance dédiés. Cet aspect est discuté dans la prochaine section.

2. Généralisation espace–temps

L'un des problèmes les plus difficiles en conception de systèmes d'information consiste à résoudre le problème de cohérence des données [**71**], ce qui peut entre parenthèse de nouveau s'effectuer de façon dynamique [**256**]. De ce fait, le maintien de modèles de données distincts mais fortement corrélés – à savoir un graphe de flot de données étendu (EFG) et un graphe global des conflits d'accès à la mémoire partagée (ECG) pouvant se déduire entièrement du premier[6] – rend l'implémentation de l'algorithme de Sven Wuytack (CDO) complexe et difficile tant à maintenir qu'à vérifier. Par ailleurs, nous avons vu en Section 1.2 que les modèles EFG/ECG ne permettent d'obtenir qu'un sur-ensemble (très) approximatif des accès concernés par une sélection donnée. Il nous apparaît donc judicieux de rechercher un nouveau modèle unifiant le graphe de flot de données étendu (EFG) et le graphe global des conflits d'accès à la mémoire partagée (ECG) en un seul modèle dont la résolution permettrait de déterminer exactement quels accès sont concernés par une sélection donnée. Un tel modèle simplifierait le développement et permettrait un dynamisme optimal pour chaque décision de réduction et chaque anticipation (simulation d'une étape de réductions).

Les premiers pas dans cette direction sont aujourd'hui publiés dans l'article [**175**] présenté en Chapitre 5 de la Partie II. Une étude des techniques de sélection permettant une meilleure résolution est publiée dans l'article [**176**] présenté en Chapitre 6 de la Partie II. Une étude plus théorique de l'unification EFG/ECG est en cours de publication [**191**] et présentée en Chapitre 7 de la Partie II.

2.1. Résolution de sélection, dynamisme et efficacité de l'exploration EFG/ECG. Commen–çons tout d'abord par illustrer que les modèles EFG/ECG ne permettent d'obtenir qu'un sur-ensemble (très) approximatif des accès concernés par une sélection donnée. Pour ce faire, repartons du graphe de flot de données étendu (EFG) présenté en Figure 1 en supposant maintenant une réduction de l'intervalle au plus tôt au plus tard de l'accès en écriture au groupe de base `X2`. A ce stade de l'analyse, il est impossible de déterminer précisément quel élément de l'intervalle représente le coût le plus important. Afin de déterminer cet élément, l'algorithme CDO effectue une anticipation (simulation de réductions) pour les extrémités de l'intervalle uniquement car il serait beaucoup trop coûteux d'anticiper la réduction de tous les éléments de l'intervalle. En Figure 5, nous illustrons le cas de l'anticipation au plus tard. Pour celle-ci, nous remarquons que les modèles EFG/ECG ne permettent pas de détecter que l'accès en écriture au groupe de base `X2` – *i.e.* W(X2) – n'est en conflit potentiel qu'avec l'accès en écriture au groupe de base `AXBX` – *i.e.* W(AXBX). Du fait de cette faible résolution lors de la sélection, la mise à jour dynamique lors de l'anticipation est exagérée en recalculant totalement les probabilités de conflit entre `X2` et ses voisins sur-approximés `AX`, `BX` et `AXBX`. Dans ce cas précis, nous illustrons en Figure 6 qu'une mise à échelle telle que celle présentée en Formule (2) aurait suffi pour traiter `AX` et `BX` (en pointillés). Par ailleurs, si l'on avait pu détecter avant cette anticipation que l'accès en écriture au groupe de base `X2` – *i.e.* W(X2) – n'est en conflit potentiel qu'avec l'accès en écriture au groupe de base `AXBX` – *i.e.* W(AXBX), il aurait été possible de déterminer sans anticipation que l'élément au plus tôt est plus coûteux que l'élément au plus tard.

A la lumière de cet exemple, nous voyons qu'il existe un lien intime entre la résolution de la sélection, le dynamisme – *i.e.* la rapidité de l'exploration et l'efficacité de celle-ci. En introduisant une sélection plus précise, il paraît possible d'améliorer la vitesse d'exécution et la qualité des résultats au-delà de la réécriture dynamique de CDO (LCDO).

6. Voir Chapitre 4 Définition 6 pour le passage ECG(EFG).

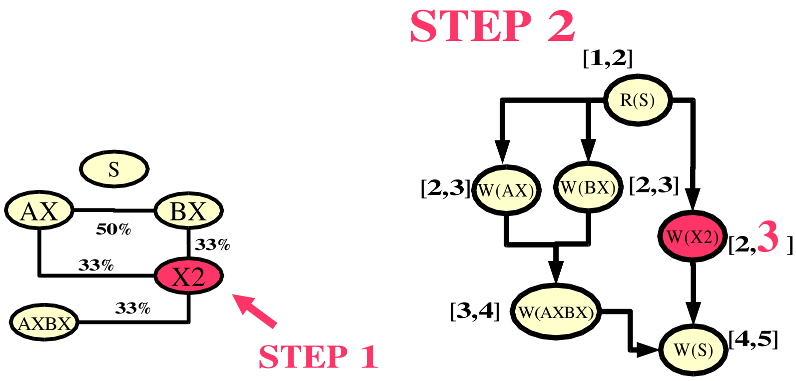


FIG. 5. *tape 1: sélection sur le graphe global des conflits d'accès à la mémoire partagée (Extended Conflict Graph, ECG). tape 2: réduction de faible résolution du graphe de flot de données étendu (EFG).*

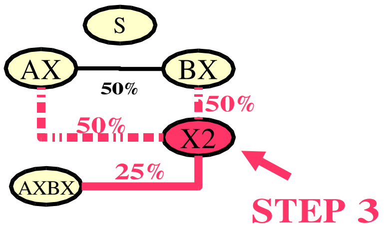


FIG. 6. *tape 3: Mise à jour de faible résolution des arrêtes du graphe global des conflits d'accès à la mémoire partagée (ECG).*

2.2. Recherche d'un dynamisme optimal. Afin de permettre la connaissance exacte des accès concernés par une sélection donnée (*ex.* ASAP ou ALAP de `AX`), il faut déterminer les accès pouvant s'effectuer en parallèle avec cette sélection (*ex.* `BX` et `X2`). Il s'agit donc de construire un modèle capable d'implémenter la fonction *concurrent* : $V \times [1,T] \rightarrow V^n$ de telle sorte que $concurrent(x,t) = \{v_1, v_2, v_3, \ldots, v_n\}$ donne l'ensemble des accès concurrents à x au temps t. Une telle fonction est extrême–ment dynamique puisqu'à chaque réduction seulement quelques résultats d'appels sont modifiés. Dans ces conditions, il paraît d'emblée raisonnable de mémoriser les nm appels possibles à cette fonction utilisés n^2m fois par la réécriture dynamique de CDO (LCDO). Pour un temps fixé t, nous remarquons que l'ensemble des accès concurrents à un accès x ($concurrent(x,t)$) est déterminé essentiellement par trois facteurs:

(1) L'existence de (x,t) sans laquelle aucun accès concurrent n'existe;
(2) L'ensemble des accès présents au temps t;
(3) Les dépendances entre ces accès, pouvant invalider certaines concurrences apparentes.

Pour chaque temps t, le meilleur moyen de mémoriser les appels à la fonction $concurrent(x,t)$ consiste en un graphe comportant tous les noeuds (x,t) existants assortis des dépendances entre ces différents noeuds. $concurrent(x,t)$ retourne la liste des noeuds indépendants de x. Afin de faciliter la détermina–tion dynamique de cette liste, il est possible d'introduire une arrête de conflit entre x et tous les noeuds indépendants. Cette représentation naturelle, extrêmement précise et dynamique de l'espace de recherche conduit à l'introduction de notre graphe de conflit espace–temps étendu (Extended Space–Time Conflict Graph, ESTCG) présenté en Figure 7.

Nous démontrons dans [**191**] présenté en Chapitre 7 de la Partie II que notre graphe de conflit espace-temps étendu (ESTCG) constitue une unification du modèle de dépendance de données (EFG)

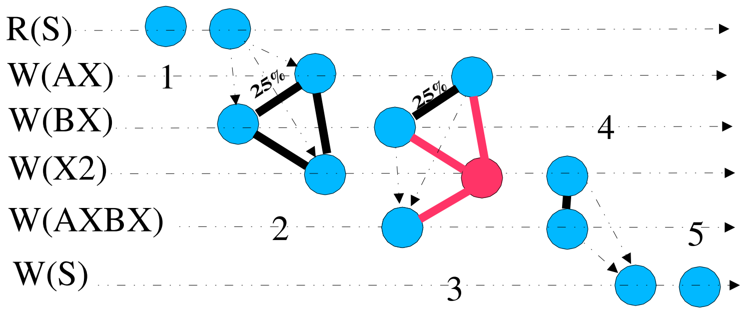


FIG. 7. *Graphe de conflit espace-temps étendu (Extended Space–Time Conflict Graph, ESTCG).*

et du modèle de détection des conflits (ECG) original de Sven Wuytack permettant une exploration globale de complexité comparable à celle de CDO pour les budgets temps-réel faibles. En permettant une sélection de haute résolution y compris à l'intérieur des intervalles explorés, nous remarquons que notre graphe de conflit espace-temps étendu (ESTCG) n'a plus besoin de compenser un manque de résolution par des anticipations coûteuses (simulation d'une étape de réductions). Dans ce cadre d'utilisation appelé GCDO (Generalised Conflict Directed Ordering), la réécriture dynamique de CDO (LCDO) est battue en $O(n^2m^2)$ – soit un gain de 2.26 pour le Segment Protocol Processor d'Alcatel – tout en facilitant le développement et la maintenance logicielle ainsi que la vérification des résultats obtenus.

En 1999, le cheminement présenté en Sections 2.1 et 2.2 de ce chapitre et l'observation de facteurs d'accélération supérieurs à 100 nous avait rendu trop optimistes sur la valeur de nos contributions pour la conception de systèmes multimedia. Premièrement et nous y reviendrons en conclusion de cette section, un gain de deux ordres de grandeur en temps de conception ne signifie pas un gain de deux ordres de grandeur en taille des systèmes concevables – ce qui est la mesure intéressante pour le concepteur. Deuxièmement, si le graphe de conflit espace-temps étendu (ESTCG) généralise les modèles de dépendance de données (EFG) et de détection des conflits (ECG), il n'est pas satisfaisant de réutiliser les techniques de sélection et de réduction mono-dimensionnelles pour notre nouveau contexte multi-dimensionnel espace–temps. De ce fait, la découverte du modèle ESTCG est plus une ouverture vers de nouvelles perspectives de recherche qu'un aboutissement en soi. Le reste de cette section présente quelques aspects de ces perspectives en **analyse espace-temps**.

2.3. Techniques de sélection espace–temps. Reprenons l'exemple de la Section 2.1. En Figure 5, la sélection mono-dimensionnelle EFG/ECG choisit de réduire l'élément au plus tôt de l'accès en écriture au groupe de base `X2` – *i.e* W(X2) – après une étape d'anticipations (simulation de réductions) telle que celle présentée en Figure 6.

Si nous calquons maintenant ce raisonnement sur le graphe de conflit espace-temps étendu (ESTCG) de la Figure 7, il est possible de détecter sans anticipation que l'élément au plus tôt ($t = 2$) est associé à un graphe local de conflit plus dense que l'élément au plus tard ($t = 4$). Mais pourquoi s'arrêter là? Une analyse espace–temps nous permet d'affiner cette décision en confirmant que l'accès en écriture au groupe de base `X2` est effectivement le plus impliqué dans des conflits et que le maximum de ces conflits intervient au milieu de l'intervalle – *c.a.d* au temps $t = 3$. Dans ce cas,

pourquoi ne pas réduire à l'intérieur de l'intervalle?

Afin de répondre à cette question, il faut malheureusement passer par un peu de formalisme repris principalement en Chapitres 4 et 7 de la Partie II avant de pouvoir établir au Théorème 1 du Chapitre 7 qu'une réduction à l'intérieur de l'intervalle est parfaitement acceptable. De fait, ce résultat définit une réduction espace–temps. Afin de tirer parti de cette nouvelle liberté de décision, une sélection espace–temps capable de détecter automatiquement une situation de type (X2,3) doit être développée.

Fort heureusement, nous ne voyons que peu d'implémentations caractéristiques pour une telle sélection espace–temps.

(1) La plus simple consiste à considérer que toutes les arrêtes de conflit du graphe espace-temps étendu (ESTCG) correspondent à un coût unitaire (Deterministic Selection, DS, en anglais) sans aucune considération des probabilités d'occurrence des noeuds (x,t). Nous montrons en Chapitre 6 Section 4.1 de la Partie II que cette implémentation conduit à des résultats catastrophiques[7].

(2) La seconde consiste à calculer les probabilités d'occurrence de toutes les arrêtes et de déterminer l'accès puis le temps le plus coûteux – ce qui constitue exactement le mécanisme LCDO généralisé présenté au paragraphe précédent (Space-Hashed Probabilistic Selection, SHPS, en anglais).

(3) La troisième consiste à déterminer le temps puis l'accès le plus coûteux (Time-Hashed Probabilistic Selection, THPS, en anglais).

(4) La quatrième consiste à rechercher le noeud (x,t) le plus coûteux sans aucune projection préalable en espace ou temps (Multi-Dimensional Selection, MDS, en anglais).

L'expérience nous a montré que le choix d'une technique de sélection espace-temps particulière dépend des caractéristiques de l'application. Pour les applications de grande taille intégrant beaucoup de parallélisme comme le Segment Protocol Processor d'Alcatel, il n'est pas étonnant de constater que les approches (3) et (4) sont les meilleures. Un problème de protocole réseau dispose d'une flexibilité remarquable concernant le choix des données concurrentes en espace (parallélisme de données trivial) mais nécessite une attention toute particulière pour ne pas dépasser une bande passante mémoire limite à chaque pas de temps.

Dans la mesure où les implémentations (2), (3) et (4) affichent une complexité et des temps d'exécution semblables en pratique, il est intéressant de rechercher des critères permettant de choisir une variante plutôt qu'une autre pour une application donnée. Cet aspect est rediscuté en Section 4.

2.4. Techniques de propagation de contraintes. Dans nos diverses contributions et expériences, une tendance semble maintenant clairement se dessiner:

plus l'espace de recherche est caractérisé de façon précise à chaque étape de décision, plus le processus de conception assistée par ordinateur (CAO) se révèle compétifif en temps d'exécution et en qualité des résultats obtenus

Nous insistons sur cette tendance car l'intuition voudrait (tout particulièrement dans le domaine de la micro-électronique et des Digital Signal Processors, DSPs) qu'une précision accrue ne soit pas vraiment nécessaire et qu'elle ralentisse l'exécution. Nous venons de montrer que cette affirmation est fausse en tout cas dans le domaine des outils d'aide à la décision et par extension dans les domaines de la conception et de l'ingénierie assistée par ordinateur (CAO/IAO). Sur la base de cette observation, le reste de cette thèse concerne tout naturellement la recherche d'une analyse espace-temps de résolution maximale permettant une caractérisation extrêmement fine de l'espace de recherche à chaque étape de décision.

La première idée dans cette direction est de s'inspirer des techniques d'intelligence artificielle déjà utilisées dans le domaine des outils d'aide à la décision [**148**] [**18**][8] et tout particulièrement la propagation de contraintes globales [**22**]. Dans cette approche, l'outil exploite toutes les informations relatives à la position courante dans l'arbre d'exploration pour réduire au maximum l'espace

7. ce qui nous avait beaucoup refroidi sur les espoirs que nous mettions sur l'ESTCG au début de l'année 2000.

8. il est amusant de constater que cette idée fut également explorée par Philips Research en 2000 [**21**] et sous un angle différent très original par l'IRISA [**166**] à la même époque.

de recherche. Bien sûr, l'introduction de notre graphe de conflit espace-temps (ESTCG) nous a déjà permis d'acquérir une finesse optimale (de niveau espace–temps) pour la connaissance des couples (x,t) acceptables. Cependant, lors du calcul de la probabilité de conflit entre deux atomes (x_1,t_1) et (x_2,t_2), il n'est pas tenu compte du degré de risque **global** que celui-ci représente. Afin d'évaluer plus précisément ce facteur, nous proposons deux schémas très simples:

(1) Pour chaque conflit **global** entre deux groupes de base, seules les instances de plus forte probabilité entre deux atomes espace–temps (x_1,t_1) et (x_2,t_2) sont considérées, les probabilités des autres instances du même conflit global étant mises à probabilité nulle – *i.e.* il ne paraît spéculativement pas très utile de réduire les instances de faible probabilité avant celles de probabilité dominante (Probabilistic or Speculative Constraint Propagation, SCP, en anglais);

(2) Pour chaque conflit **global** disposant d'au moins une instance de probabilité 100% entre deux atomes, toutes les instances moins probables sont mises à probabilité nulle – *i.e.* il est certain que réduire les instances de faible probabilité n'a plus aucun sens (Deterministic Constraint Propagation, DCP, en anglais).

Nous illustrons en Chapitre 6 Sections 4.4 et 4.5 de la Partie II une nette amélioration de la qualité des résultats par application des techniques (1) et (2). De façon générale, l'approche (2) se comporte mieux. Nous pensons que cette observation reflète l'extrême difficulté à battre des approches déterministes de type (2) par des approches spéculatives de type (1) en conception de systèmes à faible coût/performance. Ces résultats sont détaillés en Chapitre 6 Sections 4.2 et 4.3 et repris sous un angle plus fondamental en Chapitre 7 de la Partie II.

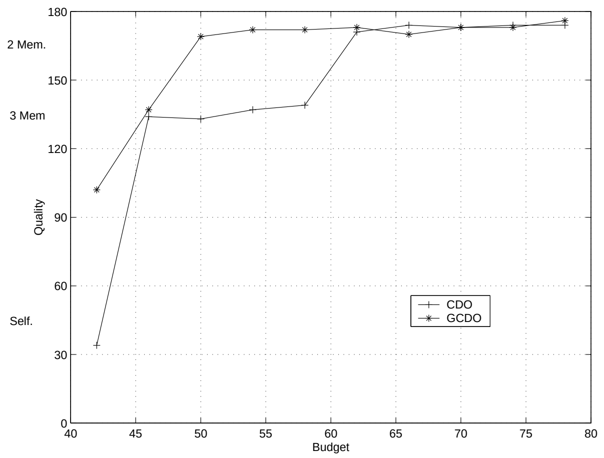


FIG. 8. *Qualité de GCDO utilisant la variante de modèle unifié MDS-DCP-ESTCG par rapport à LCDO utilisant le modèle EFG/ECG pour la conception du Segment Protocol Processor d'Alcatel.*

2.5. Exploration globale, précise et non biaisée. En Figure 8, nous illustrons la qualité de GCDO utilisant la variante de modèle unifié MDS-DCP-ESTCG par rapport à LCDO utilisant le modèle EFG/ECG pour la conception du Segment Protocol Processor (SPP) d'Alcatel. Malgré tous les facteurs en faveur du modèle ESTCG présents dans l'application SPP, nous sommes agréablement surpris de parvenir à une implémentation n'utilisant que deux mémoires à un port au lieu de trois pour un budget de 50 cycles mémoire. Cette solution obtenue 350 fois plus rapidement par GCDO que par l'algorithme original de Sven Wuytack conduit à une architecture mémoire **20%** (respectivement **62%**) plus efficace énergétiquement que celle obtenue par utilisation de l'algorithme LCDO (respectivement LS-LA) pour un budget temps-réel critique de 50 cycles mémoire. En corrélant ces résultats à la répartition énergétique globale d'un commutateur ATM[9], nous venons de découvrir une solution **68%** plus énergétiquement efficace qu'une compilation traditionnelle (type LS-LA). Pour cette application, il n'existe pas de comparatif possible avec une optimisation manuelle considérée par Alcatel comme sinon impossible du moins non économiquement viable.

Ce résultat (très) prometteur qui peut paraître à première vue une exception liée à l'application SPP n'en est rien. Non seulement nous avons observé que GCDO permet de **repousser le mûr mémoire** sur de nombreuses applications mais la suite de cette thèse montre que des facteurs bien supérieurs à 20% peuvent être obtenus. Pourtant, personne ne semble vouloir y croire! En interne IMEC, il paraît difficile de croire qu'il soit possible de ré-implémenter CDO en si peu de temps avec autant d'améliorations. En externe IMEC, il est impensable d'obtenir des résultats nettement supérieurs à ceux de la communauté des compilateurs conscients de la mémoire (Memory-aware Compilers en anglais) tels ceux de Pennsylvania State University [**204**] ou UC Irvine [**202**]. Par ailleurs, puisque GCDO repousse le mûr mémoire pour un nombre de mémoires données, il est incroyable qu'il parvienne à battre les techniques d'ordonnancement sous contraintes de ressources les plus sophistiquées tel FACTS de Frontier Design et Philips Research [**21**]. Alors, quel est le secret de la supériorité de GCDO? Quel niveau de crédibilité accorder à ces résultats?

Afin d'expliquer en partie la supériorité de notre modèle ESTCG par rapport à l'État de l'Art, nous voyons trois arguments principaux:

(1) **globalité**: nous explorons l'ensemble des solutions possibles sans aucune restriction autre que les dépendances de données, ce qui nous conduit au développement d'un ordonnanceur non-linéaire. Cette voie fut explorée pour la première fois en 1995 par Sven Wuytack à IMEC. Entre temps, elle fut abandonnée par les autres groupes de recherche pour des raisons de difficulté de modélisation et de temps de calcul prohibitifs[10], surtout lorsque son utilisation n'est pas limitée à un aspect de l'ordonnancement–placement spécifique tel le stockage et les transferts en mémoire;

(2) **précision**: nous modélisons avec une précision espace–temps optimale les effets de conflits souvent asymétriques entre accès à des données, ce qui est unique au monde;

(3) **absence de biais et self-adaptativité:** en cas d'erreur lors d'une décision d'exploration globale, notre environnement n'est affecté que (très) localement. En particulier, une suite d'erreurs de décision ne biaise pas de façon globale l'accessibilité aux solutions autres que celles explicitement écartées par la suite d'erreurs elle-même, ce qui rend notre environnement extrêmement robuste car les décisions suivantes tendent à compenser voire à corriger totalement ces erreurs. Ceci n'est pas le cas des modèles de flot de données conventionnels tel EFG.

Concernant le point (1) largement discuté au Chapitre 1, nos observations rejoignent celles de la communauté de la programmation logique par contraintes [**22**] [**18**]. Concernant le point (2) largement discuté dans cette section, nos observations rejoignent celles de la communauté des ondelettes et de l'identification espace–temps [**113**] [**118**] [**233**]. Concernant le point (3) démontré au théorème 5 du Chapitre 7 de la Partie II, nos observations rejoignent la statistique de base [**42**] et la communauté du calcul adaptatif [**25**] [**246**] [**14**] [**272**]. Bien entendu, **il est remarquable qu'autant de travaux relatent à notre problème et viennent chacun y apporter une contribution et un fondement scientifique partiel mais à la lumière de la richesse et de la puissance de ces fondements, les résultats de nos expériences n'ont à notre avis plus rien ni de scientifiquement mystérieux ni de scientifiquement étonnant.**

---

9. répartition énergétique d'un commutateur ATM: transfert 70%, horloge 21%, logique 6%, mémoire 3%.
10. il a quand même fallu sept années à IMEC (1995–2001) pour résoudre ce problème.

Dans les prochaines sections, nous montrerons qu'il est en fait possible de surpasser les techniques de compilation conventionnelles (telles LS-LA) par d'importants facteurs voire un ordre de magnitude dans le cas de mémoires distribuées. A cette occasion, nous introduirons des limites théoriques pour l'ordonnancement-placement multi-critère prouvant la capacité de GCDO à trouver des solutions quasi-optimales. A notre connaissance, des résultats aussi proches des limites théoriques prévues par l'informatique fondamentale [**94**] [**152**] sont observés pour la première fois dans le domaine de l'ordonnancement-placement des transferts et des stockages.

2.6. Vers une conception interactive, contrôlable et simplifiant considérablement les problèmes d'intégration. En conception automatique de systèmes électroniques (Electronic Design Automation, EDA, en anglais) comme dans la majorité des secteurs d'application de la Conception Assistée par Ordinateur (CAO), un point essentiel est la capacité de l'outil à interagir le plus naturellement possible avec un utilisateur expert, appelé en l'occurrence architecte de système.

Lorsqu'un nouveau projet de conception de système multimedia débute, une spécification exécutable du système n'existe généralement pas et la réutilisation de solutions préexistantes – dites propriétés intellectuelles [**238**] [**280**] – est souvent envisagée. Cela signifie que l'architecture commence généralement par des expériences sur les nouveaux sous-systèmes qu'il souhaite introduire puis raffine sa solution en explorant différentes implémentations et différents schémas de réutilisation jusqu'à se convaincre d'une bonne solution. Afin de permettre des raffinements systématiques et rapides, une estimation de la qualité de la solution courante est nécessaire à un haut niveau d'abstraction. A partir de cette information, l'architecte doit pouvoir raffiner sa spécification mais également guider l'outil vers une autre solution pour la même spécification voire étudier l'intégration du système en cours d'études avec des solutions partielles préexistantes.

Pour ces raisons, un outil de conception de système moderne doit être entièrement interactif et contrôlable par l'architecte. L'interactivité est rendue possible par un temps d'exécution faible et une présentation intuitive des résultats. La contrôlabilité nécessite de définir un espace de paramètres permettant à l'architecte de contrôler sa solution. Nous montrons en Chapitre 7 Sections 3.2 et 3.4 de la Partie II que le modèle espace–temps ESTCG permet une interactivité de grande qualité ainsi qu'une contrôlabilité totale alors que les modèles EFG/ECG ne le permettent pas. De plus, il découle conjointement de la propriété de contrôlabilité et de la propriété de self-adaptativité présentée en section précédente que l'intégration d'un composant fixe (Black-box IP model en anglais) puisse se faire par simple concaténation à la solution courante. En spécifiant par des contraintes espace–temps les libertés d'exploration autorisées pour un composant flexible (Grey-box IP model en anglais), il est possible de gérer naturellement et simplement des schémas d'intégration réputés complexes et habituellement traités principalement à la main [11]. Cet aspect est rediscuté et illustré en Section 5.1.

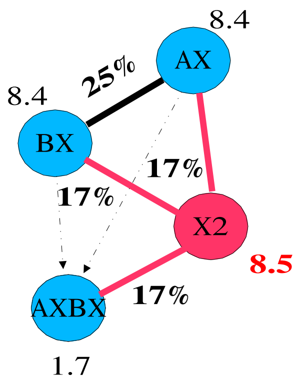
2.7. Conclusion. L'ensemble des techniques présentées dans ce chapitre permettent de construire de nombreuses variantes du modèle espace–temps unifié ESTCG, ce qui laisse la liberté à l'utilisateur d'adopter un ordonnancement–placement adapté aux caractéristiques de son application. **Les résultats prometteurs observés ainsi que les améliorations fondamentales démontrées, notamment en ce qui concerne l'utilisabilité par un architecte de système, nous convainquent de la supériorité du modèle ESTCG et nous conduisent à recommander l'abandon des modèles de flot de données et de conflit conventionnels tels EFG/ECG pour l'ordonnancement-placement multi-critère des stockages et des transferts.** Sur la base de ce nouveau modèle, l'introduction de

11. Des expériences sont en cours pour prouver la faisabilité d'intégrer l'outil Phideo de Philips Research comme un composant flexible appelé par l'outil Acropolis d'IMEC. Cela pourrait résoudre les contraintes de streaming ainsi que d'éviter la gestion de données partagées nécessaire lorsque le composant et le système global ne sont pas raffinés conjointement.

l'algorithme GCDO permet des gains en temps de conception de deux ordres de magnitude par rapport à l'utilisation de l'algorithme original CDO (prototype `FGB` d'IMEC). Sur base d'une interaction de l'ordre de la minute avec l'utilisateur, ces gains ne signifient malheureusement en pratique qu'un facteur 2 à 3 en taille des systèmes concevables. Nous devrons donc nous intéresser à des techniques d'accélération plus agressives dans les prochaines sections afin de s'accomoder de l'augmentation de la taille des systèmes de deux ordres de grandeur et plus lors du passage à la technologie System-on-Chip (SoC) [12].

3. Techniques d'approximation

Lorsqu'un modèle de calcul est trop coûteux, le moyen le plus élégant d'y remédier est le recours à la théorie de l'approximation. Il existe des formes très variées d'approximation notamment pour l'intelligence artificielle [**156**], les problèmes NP-complets [**88**] et l'algèbre [**86**]. Dans cette section, nous étudions l'application de techniques d'approximation dites exacte [**243**] [**50**] multi-étage [**90**] [**96**] et divise–et–règne [**69**] pour accélérer notre algorithme GCDO.



$$cost(ECG) = \sum_E \gamma.p(e) + \sum_{chromatic(E)\backslash self(E)} (\beta-\gamma).p(e)$$
$$+ \sum_{self(E)\backslash chromatic(E)} (\alpha-\gamma).p(e) + \sum_{chromatic(E)\cap self(E)} (\delta-\gamma).p(e)$$

FIG. 9. *Desambiguation de la fonction coût à trois termes de Sven Wuytack en utilisant quatre paramètres* $(\alpha = 50, \beta = 20, \gamma = 10, \delta = 70)$ *pour contrôler précisément le coût au niveau d'abstraction ECG.*

3.1. Approximation exacte de la fonction coût. Afin d'estimer précisément le coût relatif d'un conflit par rapport à un autre, il faut classer tous les conflits (*c.a.d* toutes les arrêtes du graphe **global** des conflits d'accès à la mémoire partagée) à chaque anticipation/réduction – soit de l'ordre de n^2m classifications globales – en utilisant les trois catégories définies en Chapitre 1 Section 5 de cette partie dont le calcul peut être desambigué en utilisant les quatre termes présentés en Figure 9. Tout puriste aura probablement déjà remarqué depuis le premier chapitre le caractère ambitieux de cette classification, dans la mesure où il s'agit d'une complication du problème de coloration déjà NP-complet [**88**] placé de façon (presque) irresponsable pour guider un algorithme d'ordonnancement–placement également NP-complet, ce qui semble donner naissance à une nouvelle catégorie d'algorithmes super-NP-complets!

---

12. Source: roadmap du Projet Européen System-level Methods and Tools (SMT) [**255**].

Imaginez que nous utilisions ne serait-ce qu'un algorithme de coloration basé sur une approche gloutonne en $O(bln(b))$ où b est le nombre de groupes de base, par exemple [**219**] ou [**101**]. Il faut alors une complexité d'au moins $O(n^2mbln(b))$ pour calculer les n^2m classifications globales dont le coût s'obtient en $O(n^2)$ par la méthode des matrices de Sven Wuytack. Le tout représente donc une complexité record de $O(n^4mbln(b))$, ce qui fait de cette implémentation de CDO un algorithme d'ordonnancement–placement difficilement utilisable, comme cela est malheureusement déjà le cas pour la version optimisée de FDS appelée Improved Force–Directed Scheduling (IFDS) par Philips Research pourtant annoncée $O(n^2m^2)$ dans [**274**] [13].

Afin d'améliorer cette situation, il est naturel de penser à une implémentation dynamique de la coloration mais cela s'avère malheureusement irréalisable car il est impossible de décider si l'ajout d'une arrête dans un graphe modifie le nombre chromatique sans effectuer une nouvelle coloration complète de tous les noeuds [14]. Il nous faut donc rechercher une technique d'approximation du nombre chromatique mieux adaptée à nos besoins. Il existe une littérature (très) abondante sur ce sujet. Le nombre et la variété des approches sont tels que l'ACM dû organiser en 1996 un concours du meilleur algorithme de coloration sur une collection des graphes parmi les plus difficiles issus de problèmes réels. Le gagnant fut E. C. Sewell [**243**] dont l'algorithme pu trouver le nombre chromatique exact pour tous les graphes proposés. Cet algorithme comporte deux étapes. La première recherche la clique maximale [**19**]. La deuxième évalue la différence entre la taille de la clique maximale et le nombre chromatique (généralement faible). Sur la base de ces résultats, il est légitime de penser que la taille de la clique maximale constitue une approximation idéale du nombre chromatique. Les meilleurs experts se penchent sur ce problème depuis des années sans réponse [**154**] [**106**], ce qui fait que l'algorithme d'E. C. Sewell ne dispose toujours d'aucun fondement scientifique aujourd'hui.

Pour autant, avez-vous déjà essayé de piéger une clique maximale de taille n en ajoutant des arrêtes jusqu'au moment où le nombre chromatique n'est plus égal à sa taille? Pour $n = 2$, il faut construire au moins un hexagone en ajoutant 3 noeuds et 4 arrêtes, ce qui correspond à une augmentation minimale de 200% du nombre d'arêtes placées exactement là où il ne faut pas. Le cas du triangle ($n = 3$) a été particulièrement étudié car ses applications en modélisation 2D [**249**] et 3D [**239**] ont justifié le développement d'algorithmes de coloration spécifiques. Dans ce cas, il faut construire un grand triangle à partir de trois petits en ajoutant 1 noeud et 3 arrêtes, ce qui correspond à une augmentation de 100% du nombre d'arrêtes. Pour $n = 4$, il faut construire un parallélépipède de base triangulaire mais en ajoutant cette fois 2 noeuds et 10 arrêtes, ce qui correspond à une augmentation de 166% du nombre d'arêtes. **Pour $n \leq 4$, il faut donc au moins trois cliques de taille maximale connectées entre elles exactement comme il ne faut pas pour invalider l'égalité du nombre chromatique à la taille de la clique maximale.** Ce scénario paraissant (très) improbable, Sven Wuytack et d'autres membres de la communauté de la compilation [**98**] considèrent que le risque d'approximer le nombre chromatique par la clique maximale est minime.

Revenons maintenant à nos algorithmes d'exploration. Pour peu que le budget temps-réel donné soit un peu surestimé ou pour peu que l'application soit fortement parallèle (ce qui est souvent le cas en traitement du signal et multimedia), un grand nombre d'accès à la mémoire disposent de grandes fenêtres au plus tôt au plus tard. En conséquence, un grand nombre de probabilités de conflit entre accès sont non nulles dans la matrice des probabilités de conflit et le graphe **global** des conflits d'accès à la mémoire est (presque) plein. Il faut un certain temps pour comprendre que nous touchons là la **faiblesse majeure** de l'algorithme CDO, dont les conséquences négatives sont à mon avis catastrophiques au regard des objectifs annoncés au chapitre précédent.

---

13. je soupçonne cette formule de complexité de ne pas tenir compte de la fonction coût, qui comporte en pratique une détection de conflits basée sur un solveur en nombre entier [**275**], ce qui est nécessairement beaucoup plus lent que le calcul matriciel de Sven Wuytack.

14. ce résultat illustre bien le caractère (très) global du nombre chromatique et la difficulté d'approximation qui en découle.

Premièrement, puisque les décisions d'exploration sont basées majoritairement sinon totalement sur le graphe **global** d'accès à la mémoire partagée (ce qui part d'un excellent principe), le fait que celui-ci reste plein pour un grand nombre des premières itérations de l'algorithme prouve que CDO commence par effectuer une réduction aléatoire de son espace de recherche, ce qui est bien entendu (très) préjudiciable pour la qualité de la solution finale et le temps d'exécution. Deuxièmement, lorsque le graphe **global** d'accès à la mémoire partagée commence à se creuser un peu, les décisions se basent pour une part très importante sur l'approximation du nombre chromatique par la taille de la clique maximale, qui est (très) probablement fausse sur un graphe à la limite d'être plein. Ce n'est donc qu'à l'occasion des dernières itérations que le graphe **global** d'accès à la mémoire partagée devient enfin creux et que CDO commence à prendre des décisions plus raisonnables. En pratique, cela lui vaut de fournir des résultats souvent moins bons qu'un algorithme traditionnel d'ordonnancement par liste [**20**] [**117**]. Convaincus de la valeur de l'algorithme, Sven Wuytack et Erik Brockmeyer recommandent alors d'utiliser une optimisation des paramètres $(\alpha,\beta,\gamma,\delta)$ de la fonction coût pour guider l'algorithme vers une meilleure solution, au prix bien sûr de temps d'exécution encore plus pénalisants. Nous avons essayé de construire une technique d'optimisation de paramètres au dessus de CDO en collaboration avec des experts du domaine [**182**] mais il s'avère qu'en l'état le comportement vis-à-vis de ces paramètres est trop chaotique pour permettre l'utilisation d'outils d'optimisation **globale** aussi sophistiqués fussent-ils pour assister sa robustesse. En l'état, l'algorithme n'est donc absolument pas viable [15].

Comment se sortir des pathologies de notre fonction coût?

Pour ce faire, commençons par remarquer que l'erreur consistant à trop se focaliser sur le nombre chromatique lors des premières itérations ainsi que la règle de bonne utilisation de l'approximation du nombre chromatique par la taille de la clique maximale vont toutes les deux dans le même sens de ne plus recourir à une analyse chromatique sur un graphe **global** des conflits d'accès à la mémoire partagée plein lors des premières itérations. Puisqu'il serait d'un même ordre pathologique de supprimer l'analyse chromatique lors des premières itérations (ce qui guiderait mal l'algorithme lors de ses premières décisions), le seul point sur lequel nous puissions influer reste donc l'aspect plein. Pourquoi CDO construit-il ce graphe plein et que signifie-t-il? En fait, il dit juste que, sur base individuelle, l'hypothèse qu'un conflit entre un accès et n'importe quel autre se retrouve dans la solution finale est de probabilité non nulle au début du processus d'optimisation. Heureusement! Mais malheureusement, cela piège techniquement la stratégie d'exploration mise en œuvre. Nous voyons bien pourtant que "l'hypothèse qu'un conflit entre un accès et n'importe quel autre se retrouve dans la solution finale soit de probabilité non nulle" est une information sans intérêt pour le processus d'exploration qui nécessite des dynamiques relatives pour être bien et clairement guidé mais s'accorde mal de ce genre de message flou et bruité. Ce qui est certainement (beaucoup) plus intéressant pour le processus d'exploration est de savoir précisément quel conflit devient si probable qu'il faut absolument soit chercher à en réduire les risques soit s'en accommoder. **Nous proposons donc de ne retenir dans l'analyse chromatique que les conflits inévitables (de probabilité 100%) dont l'exploration doit absolument s'accomoder.** Ceci minimise le nombre d'arêtes et garantit la qualité d'approximation du nombre chromatique par la clique maximale. Thierry Franzetti montre que cette simplification permet de mettre en œuvre un algorithme de clique maximale dynamique **et** exact [**84**] dont l'étude de complexité révèle un comportement NP au pire en $O(2^{\sqrt{b}})$ pour les graphes creux ne comportant pas plus d'arêtes que de noeuds.

15. ceci est la conclusion scientifique franche que nous tirons de douze mois d'études de CDO et ne contient rien de personnel, en particulier à l'encontre de Sven Wuytack dont la formulation originale ne perd rien en pertinence et à laquelle nous consacrons la majorité de cette thèse.

En pratique, nous n'avons trouvé aucun exemple réel où l'utilisateur souhaite explorer le placement de plus de 50 groupes de base en mémoire ou que le graphe **global** des conflits à la mémoire partagée de la solution optimale comporte plus d'arêtes que de noeuds [16]. Nous pouvons ainsi réduire la complexité de l'analyse chromatique par un facteur d'au moins $\frac{50.ln(50)}{2^{\sqrt{50}}} = 1.46$ en éliminant les pathologies de CDO tout en renforçant la qualité d'exploration avec une fonction coût exacte. La complexité au pire du nouveau CDO ainsi construit est $O(2^{\sqrt{b}}n^4m)$. Par rapport à l'algorithme CDO initial, la qualité des résultats est bien meilleure et toujours meilleure qu'un algorithme d'ordonnancement par liste. De plus, le comportement du nouveau CDO est tellement robuste que l'optimisation des paramètres $(\alpha,\beta,\gamma,\delta)$ n'aurait pas de sens pour les applications réelles que nous avons considéré car ces paramètres n'influent presque plus sur la qualité de la solution [17]. Enfin, pour les exemples réels que nous avons traité, l'algorithme de clique maximale de Thierry Franzetti s'est avéré tellement dynamique que le temps d'exécution du nouveau CDO reste distant de moins de 50% du temps d'exécution du même algorithme utilisant cette fois une approximation $O(1)$ (très) inexacte du nombre chromatique par le degré du noeud courant, ce qui en fait un algorithme seulement $O(n^4m)$ pour nos besoins.

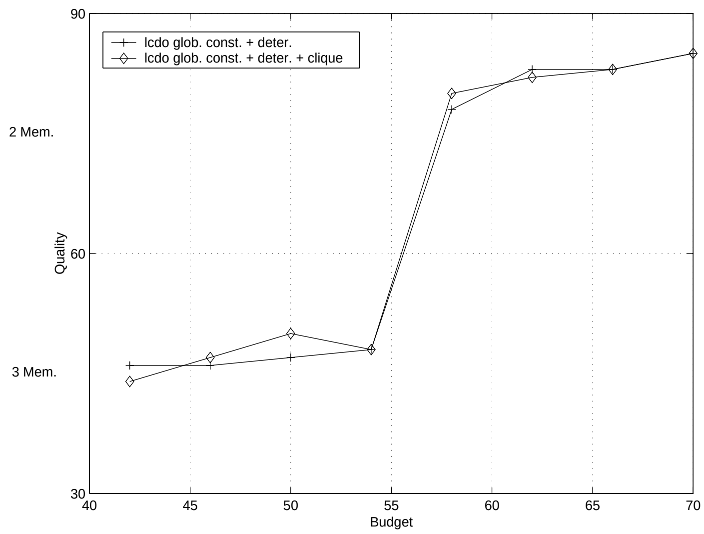


FIG. 10. *Approximation exacte de la fonction coût appliquée à l'algorithme CDO avec comparaison à l'algorithme CDO initial après optimisation des paramètres* $(\alpha,\beta,\gamma,\delta)$ *pour le Segment Protocol Processor d'Alcatel.*

En Figures 10 et 11, nous illustrons les gains en qualité obtenus par utilisation d'une approximation exacte de la fonction coût. Pour CDO, cela permet d'abandonner l'optimisation des paramètres $(\alpha,\beta,\gamma,\delta)$. Pour GCDO, cela permet d'obtenir une solution utilisant deux mémoires à un port jusqu'à un budget temps-réel de 42 cycles mémoire. Par rapport à l'algorithme CDO original, nous trouvons désormais une solution 35% plus rapide sur une architecture utilisant deux mémoires à un port.

16. ce qui reviendrait à imposer de façon assez paradoxale des contraintes temps-réel extrêmes à notre environnement de conception économique ACROPOLIS dont ce n'est bien entendu pas la vocation. Pour cela, l'utilisateur doit recourir à une approche d'optimisation de vitesse pure.

17. ce qui illustre de la possibilité d'optimiser les critères de consommation, mémoire et surface de silicium à un niveau de qualité (très) proche des optimums observés pour chaque critère pris séparément. Ces observations encourageantes ajoutées à celles reprises dans [**204**] concernant consommation et temps laissent penser qu'une optimisation **globale** consommation, mémoire, surface de silicium et temps est compétitive même dans le cas d'objectifs (très) singuliers proches du mono-critère.

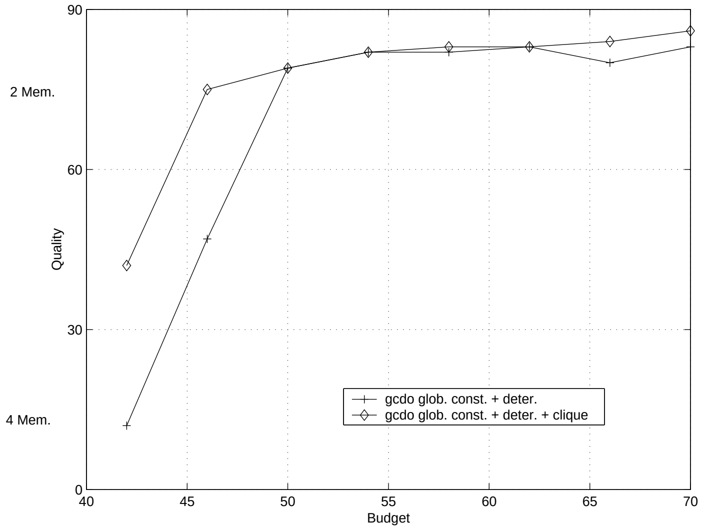


FIG. 11. *Approximation exacte de la fonction coût appliquée à l'algorithme GCDO avec comparaison à l'algorithme GCDO initial pour le Segment Protocol Processor d'Alcatel.*

En supposant maintenant un maximum de 50 cycles mémoire pour implémenter ce système et en corrélant nos résultats à la répartition énergétique globale d'un commutateur ATM, nous venons de découvrir une solution globalement au moins **2.89 fois** plus énergétiquement efficace qu'une compilation traditionnelle (type LS-LA) et au moins **39%** plus énergétiquement efficace que la solution CDO après une fastidieuse optimisation des paramètres $(\alpha,\beta,\gamma,\delta)$. A notre connaissance, de tels gains nettement supérieurs à 100% sont observés pour la première fois dans le domaine de l'ordonnancement-placement des transferts et des stockages, surtout en ce qui concerne un schéma architectural aussi simple qu'un processeur connecté par bus à un nombre variable de mémoires sans présence de phénomènes complexes tels le streaming, l'utilisation de caches ou de technologies mémoire synchrones de type SDRAM.

En Figure 12, nous illustrons la nette supériorité de l'analyse espace–temps pour résoudre des compromis architecturaux dans la zone des budgets temps–reel (très) difficiles. Dans un cas où le premier facteur de compromis serait la vitesse pure, notre solution en 42 cycles mémoire est en fait **57%** plus énergétiquement efficace que la solution trouvée par une compilation traditionnelle (algorithme CDO initial inclus). **Cette efficacité énergétique s'obtient avec une architecture mémoire à deux ports au lieu des huit ports nécessaires par des approches plus conventionnelles, permettant de ce fait le recours à un mécanisme de communication processeur–mémoire–environnement largement simplifié. Cette simplification signifie des gains économiques probablement largement supérieurs aux gains énergétiques que nous présentons.**

3.2. Approximation multi-étage du modèle espace–temps unifié ESTCG. La technique d'accélération d'algorithmes de graphe la plus étudiée dans la littérature est sans aucun doute le recours à l'approximation multi-étage [**263**] [**90**] [**215**] [**120**] [**96**] [**81**] [**123**]. Cette technique consiste à regrouper itérativement les noeuds du graphe afin de parvenir à une représentation plus abstraite du problème, et donc plus rapide à résoudre.

Afin de mettre en œuvre un tel schéma de résolution sur notre graphe de conflit espace–temps étendu (ESTCG), il convient de définir quels regroupements de noeuds sont possibles et la manière de les gérer. Il nous apparaît que deux règles sont essentielles pour autoriser des regroupements:

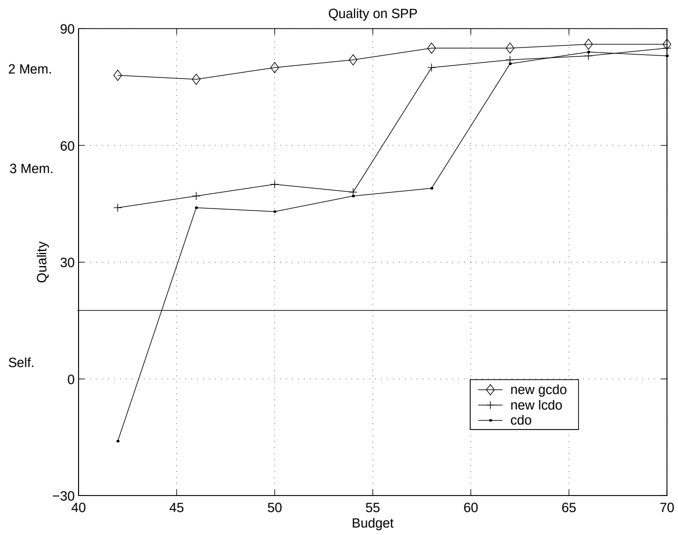


FIG. 12. *Supériorité de l'analyse espace–temps pour résoudre des compromis architecturaux dans la zone des budgets temps–reel (très) difficiles.*

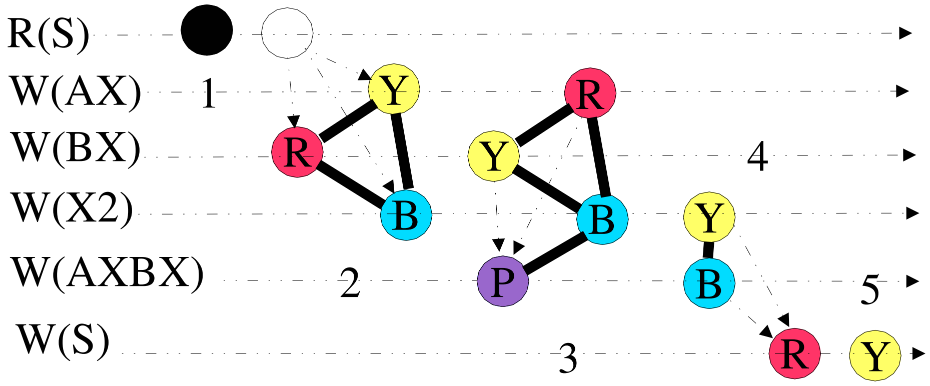


FIG. 13. *Exemple de partitionnement espace–temps.*

(1) il est interdit de placer toutes les occurrences à un même accès dans le même groupe car la décision de supprimer ce groupe supprimerait l'accès, ce qui n'est pas une solution acceptable;

(2) afin de conserver une finesse d'analyse de niveau espace–temps, il convient de placer les occurrences d'accès localement en conflit ou sujet à des dépendances dans des groupes distincts afin de préserver une finesse d'analyse des coûts semblable au niveau des groupes qu'au niveau atomique initial. Dans le cas contraire, un groupe cacherait des conflits et dépendances internes, rendant leur analyse ainsi que leur résolution multi-étage totalement impossible. De ce fait, la possibilité d'atteindre une solution optimale serait perdue par construction.

En Figure 13, nous illustrons le partitionnement espace–temps le plus agressif possible *i.e.* une coloration minimale du graphe de conflit espace–temps étendu (ESTCG) respectant les règles (1) et (2). Une phase de préparation de ce partitionnement peut déduire de l'unicité de la racine R(S) qu'il faut de toute façon ordonnancer celle-ci au temps $t = 1$. Une coloration spécialisée donne quatre couleurs: Y (yellow), R (red), B (blue), P (purple). Une analyse des coûts au niveau de ces couleurs (ou groupes) utilisant la fonction coût de la Figure 9 avec approximation exacte donne Y(6.7), R(5), B(8.5), et P(1.7). De ce fait, l'algorithme multi-étage choisit de réduire B(8.5). Par application de la règle (1), le groupe Y et P doivent être conservés. La dépendance entre Y, R et P au temps $t = 3$ fait que R et Y doivent alors être automatiquement réduits. Dans ce cas, il y a incohérence des règles puisque Y doit être à la fois réduit et conservé. Supposons maintenant que l'algorithme multi-étage puisse revenir sur sa décision de réduction pour Y(6.7). Par application de la règle (1), le groupe R doit être conservé. La dépendance entre R et P au temps $t = 3$ fait que P doit être réduit. Par application de la règle (1), B doit alors être conservé. La dépendance entre B et R au temps $t = 4$ fait que R doit être réduit, ce qui conduit de nouveau à une incohérence. Supposons maintenant que l'algorithme multi-étage puisse revenir sur sa décision de réduction pour R(5) alors Y doit être conservé donc P réduit. Solution finale: Y et B. Cette solution conduit à l'optimum quelque soit le raffinement ultérieur de niveau atomique utilisé. Sur cet exemple, nous avons donc pu trouver une solution optimale en trois réductions de niveau groupe et deux réductions de niveau atomique alors que l'approche GCDO demande sept réductions de niveau atomique. En pratique, il ne paraît cependant pas raisonnable de mettre en œuvre une approche aussi agressive pouvant conduire à des solutions incohérentes.

Comme nous venons de le voir, les incohérences proviennent d'une conjonction malheureuse des règles (1) et (2). De ce fait, il existe deux partitionnements espace–temps sans risque d'incohérence:

(1) Pour chaque temps, regrouper les accès entre lesquels aucune dépendance ni conflit n'existe afin de respecter la règle (2) – *c.a.d* les accès exclusifs présents par exemple dans deux branches disjonctives d'une instruction `if`;

(2) Pour chaque accès, regrouper les temps en au moins deux groupes afin de respecter la règle (1).

Il nous semble que le partitionnement espace–temps (1) est beaucoup plus contraignant que le partitionnement espace–temps (2). C'est pourquoi nous nous sommes intéressés exclusivement au partitionnement espace–temps (2). Ceci nous conduit aux algorithmes adaptatifs LCDO(k) et GCDO(k) capables de partitionner m temps possibles pour un accès en seulement $ln_{\frac{k}{k-1}}(m)$ groupes temporels. Ces techniques sont présentées plus en détail en Chapitre 4 Section 6, et Chapitre 5 Section 2.2.

En Figure 14, nous présentons la dégradation des résultats en appliquant GCDO(k) pour optimiser le Segment Protocol Processor d'Alcatel. Nous constatons que pour $k \geq 6$, les mêmes résultats que GCDO sont obtenus.

En Figure 15, nous présentons la dégradation des résultats en appliquant LCDO(k) pour optimiser le Segment Protocol Processor d'Alcatel. Nous constatons de nouveau que pour $k \geq 6$, les mêmes résultats que LCDO sont obtenus.

En Figure 16, nous comparons les temps d'exécution de CDO, LCDO, GCDO, LCDO(2) et GCDO(2) pour optimiser le Segment Protocol Processor (SPP) d'Alcatel. Nous observons des facteurs d'accélération spectaculaires pour LCDO(2) allant de 3.5 à 20. Nous observons des facteurs d'accélération plus modérés pour GCDO(2) allant de 10% à 2.5. Au final, nous parvenons à réduire le temps d'exécution de CDO d'un facteur 400 en utilisant LCDO(k) alors que la théorie prévoit environ $30.89 \times 8 = 247$. Cette surperformance de LCDO(k) par rapport à sa formule de complexité s'explique par une réduction extrêmement rapide de l'espace de recherche puisque les décisions de niveau groupe impliquent systématiquement de nombreuses mises à jour au plus tôt au plus tard.

Au contraire, nous ne parvenons à réduire le temps d'exécution de CDO que d'un facteur 1,000 en utilisant GCDO(k) alors que la théorie prévoit environ $156^2/8 = 3{,}042$. Cela s'explique bien sûr en partie par des mises à jour au plus tôt au plus tard moins systématiques et donc une réduction de l'espace de recherche moins rapide. Nous pensons cependant que cette sous-performance s'explique principalement par la lourdeur de l'initialisation du modèle espace–temps étendu ESTCG actuellement effectué de façon transparente par création dynamique de nombreux objets espace–temps en C++. Pour SPP, GCDO(k) doit créer un graphe constitué de 15,000 noeuds et environ 2,500,000

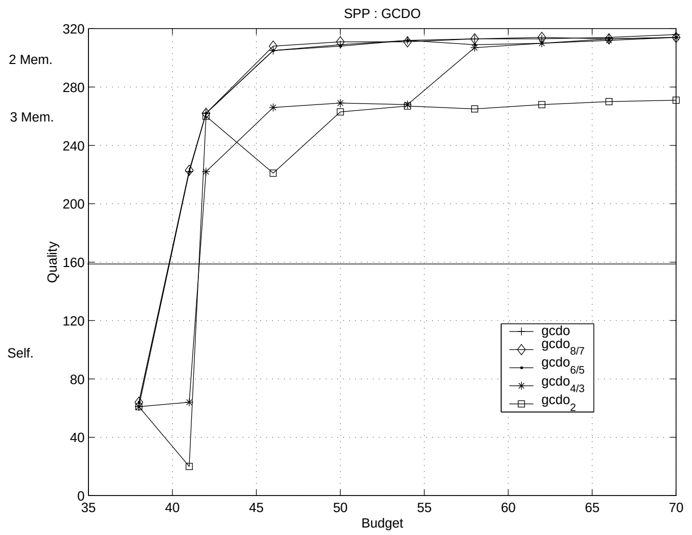


FIG. 14. *Dégradation des résultats en appliquant GCDO(k) pour le Segment Protocol Processor d'Alcatel.*

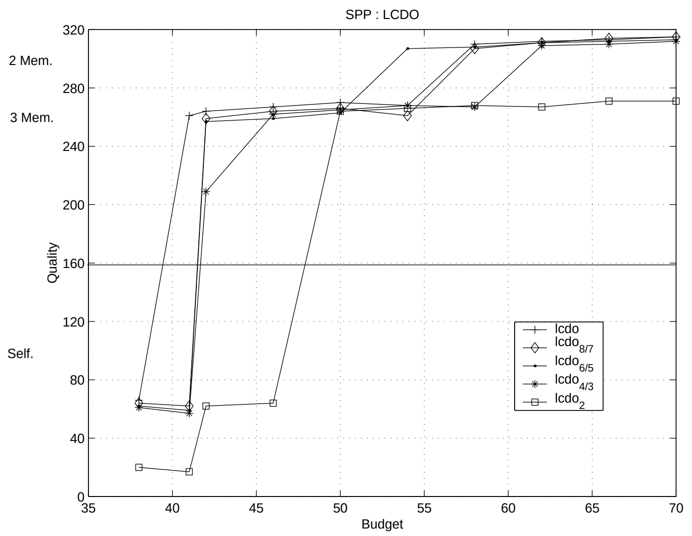


FIG. 15. *Dégradation des résultats en appliquant LCDO(k) pour le Segment Protocol Processor d'Alcatel.*

arrêtes alors que LCDO(k) se contente de manipuler des graphes de 156 noeuds et pas plus d'arêtes. Une meilleure gestion de la mémoire dynamique devrait permettre à GCDO(k) d'atteindre voire de dépasser ses bornes théoriques, ce que nous n'avons pas eu le temps d'entreprendre. Malgré cela, **GCDO(k) reste plus rapide et donne des meilleurs résultats que LCDO(k) en l'occurrence, ce qui reste bien entendu un résultat (très) encourageant. Pour les budgets temps-réel intéressants, nous venons de passer de deux heures à dix secondes de temps d'exécution!**

3.3. Approximation divise–et–règne de l'algorithme GCDO. Afin de traiter des problèmes de grande taille efficacement, la technique divise–et–règne est également largement répandue [**69**]. A IMEC, cette technique fut récemment utilisée pour permettre la manipulation de plusieurs nids

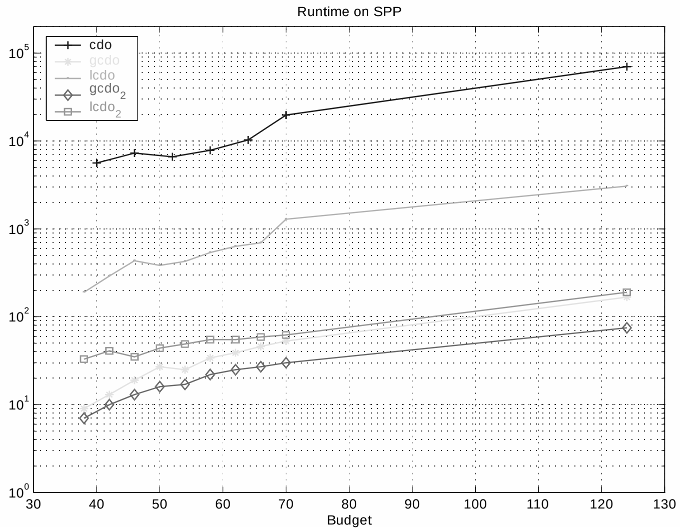


FIG. 16. *Temps d'exécution des algorithmes LCDO, GCDO, LCDO(2) et GCDO(2) comparés à CDO pour le Segment Protocol Processor d'Alcatel.*

de boucles par CDO [**29**] [**28**]. En ordonnancement-placement multi-critère, il n'existe pourtant aucune raison autre que le temps d'exécution obligeant le recours à une approche divise–et–règne. Par ailleurs, cette technique doit être considérée comme une approximation conduisant à une résolution hybride globale–locale et de ce fait dégradant la qualité des résultats. Afin de minimiser ces dégradations, il nous semble que l'application devrait être partitionnée de façon intelligente et pas seulement sur des critères de niveau code source (*ex.* appartenance à un nid de boucle).

Compte tenu des temps d'exécution observés à la section précédente, du risque de dégrader la qualité des résultats et de la difficulté d'un partitionnement hiérarchique automatique [**149**] [**171**] [**121**], nous n'avons pas développé de version divise–et–règne de l'algorithme GCDO (D&C-GCDO). En supposant qu'il soit possible de partitionner un système de taille n en $\sqrt{n}$ systèmes de taille $\sqrt{n}$ sans dégrader la qualité des résultats, nous avons effectué le calcul de complexité présenté en Table 1. Dans un premier temps, cette étude montre que notre algorithme GCDO(k) atteint un niveau de performance tout à fait comparable à la version divise–et–règne de CDO (notée `D&C-FGB`) actuellement utilisée en interne IMEC tout en permettant de préserver les niveaux de qualité de résultats présentés en section précédente. **Nous recommandons donc le remplacement de l'algorithme `D&C-FGB` par l'algorithme GCDO(k).** Dans un deuxième temps, cette étude montre qu'**une version divise–et–règne de GCDO(k) – *i.e.* D&C-GCDO(k) – constitue la clé de la conception interactive de logiciels de 10,000 lignes C/C++ et plus ainsi que la clé du ré-ordonnancement–placement à la volée de noyaux temps-réel jusque 1,000 lignes C/C++.** Si l'on considère que les logiciels de grande taille sont généralement conçus de façon modulaire avec un nombre de variables partagées entre modules (très) restreint alors nous pouvons être optimistes que D&C-GCDO(k) est à notre portée [18]. De tels problèmes logiciels modulaires et de grande taille apparaissent avec les normes multimedia émergentes telles MPEG-4 et les architectures de processeur à base de multi-threading [**247**].

3.4. Conclusion. L'ensemble des techniques présentées dans cette section permettent d'introduire un grand nombre d'algorithmes, ce qui permet d'adopter un ordonnancement–placement totalement maîtrisé (dans les limites du raisonnable) en choisissant le meilleur algorithme adapté à une

18. cette intuition est renforcée par la qualité des résultats obtenus relativement facilement par `D&C-FGB`, selon Erik Brockmeyer.

Algorithme	Complexité	ANSI-C ln/s	ANSI-C ln/mn
D&C-GCDO(k)	$O(n^{3/2}m^{1/2}ln(m^{1/2}))$	926	**18,097**
D&C-LCDO(k)	$O(n^2ln(n^{1/2})ln(m^{1/2}))$	392	2,256
`D&C-FGB`	$O(n^{5/2}m^{1/2})$	131	512
GCDO(k)	$O(n^2mln(m))$	117	409
LCDO(k)	$O(n^3ln(n)ln(m))$	49	152
GCDO	$O(n^2m^2)$	71	197
LCDO	$O(n^3ln(n)m)$	35	91
CDO (`FGB`)	$O(n^4m)$	22	50

TAB. 1. *Principales variantes introduites pour une maîtrise du temps d'exécution de CDO, leurs complexités théoriques et leurs vitesses de conception multi-critère et flexible théoriques exprimées en nombre de lignes ANSI-C par seconde (respectivement par minute). Pour ce calcul, nous supposons un accès à la mémoire partagée par ligne ANSI-C.*

application donnée et un temps de conception aparti. Nous présentons en Table 1 les principales variantes introduites pour maîtriser le temps d'exécution de CDO. Le lecteur remarquera qu'en ce qui concerne la conception *interactive* de logiciels embarqués, notre approche signifie en théorie des gains en taille des systèmes concevables de l'ordre de 50 à 350 par rapport à l'utilisation de CDO (prototype `FGB` d'IMEC) qui représente déjà un gain de productivité significatif pour le concepteur[19], ce qui s'accorde assez bien avec les objectifs du projet Européen System-level Methods and Tools (SMT) [**255**]. En pratique, ces gains ne sont observés que sur les applications de taille suffisamment grande et/ou suffisamment parallèles pour profiter pleinement de nos améliorations[20]. Nous n'y voyons aucun inconvénient dans la mesure où les applications qui profitent peu de nos contributions n'ont pas de problème de temps de conception. Nous sommes également convaincus que l'avenir de la conception de logiciels embarqués concernera de plus en plus d'applications de grande taille et/ou parallèles, ce qui rendra une maîtrise du temps de conception ainsi que le recours à la Conception Assistée par Ordinateur (CAO) incontournables.

4. Éléments formels et intelligents

4.1. Exemple de GCDO sur le Segment Protocol Processor. Nous avons observé en section 3.1 des gains en consommation électrique de 2.89 en utilisant l'algorithme GCDO sur le Segment Protocol Processor d'Alcatel. Afin de mieux évaluer la limite de ces gains et leur proximité à une solution optimale, commençons par faire un bilan des informations dont nous pouvons disposer facilement concernant cet exemple:

(1) Le nombre de noeuds est $n = 156$ et le nombre de conflits exclusifs est $x = 3314$, ce qui signifie que le degré d'exclusivité est très élevé avec un niveau de $x/[n(n-1)/2] = 27.41\%$;
(2) Le chemin critique est 38 cycles mémoire;
(3) GCDO trouve la meilleure solution en 42 cycles mémoire pour 2 ports mémoire;

19. le scénario typique d'utilisation du prototype `FGB` d'IMEC consiste à définir en quelques heures son problème et de laisser tourner plusieurs explorations pendant quelques heures voire quelques jours, ce qui représente un temps (souvent) comparable à une optimisation manuelle mais avec un niveau d'effort bien plus faible et une estimation de la consommation électrique et de la surface (beaucoup) plus précise au final. Jouer avec les paramètres (α,β,γ) revient à effectuer l'exploration à la main en aveugle par rapport à une exploration directement sur le code source. Cela est suffisant pour commencer à attirer des utilisateurs.

20. nous envisageons un travail d'interfaçage à Haute Performance pour améliorer l'impact de nos contributions sur les applications de taille moyenne. En l'état, trop de temps est passé dans l'échange de données entre outils.

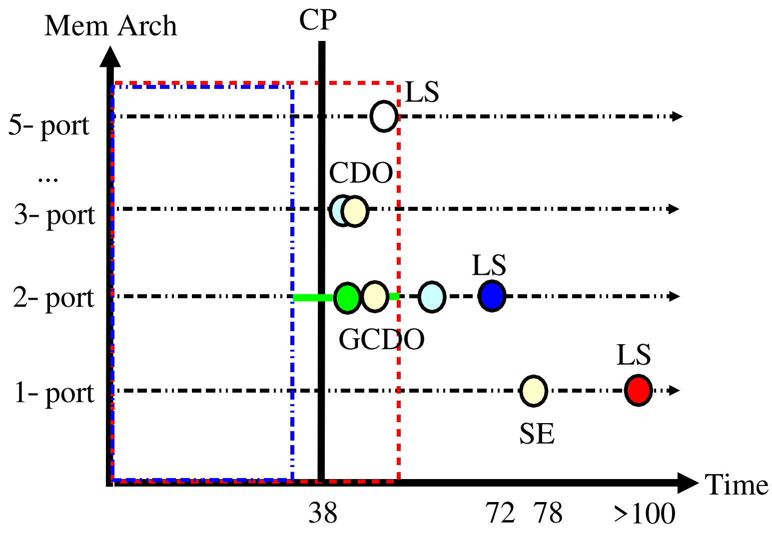


FIG. 17. *Raisonnement formel conduisant à la démonstration de la quasi-optimalité de la solution obtenue par GCDO sur le Segment Protocol Processor d'Alcatel.*

(4) LS-LA trouve une solution en 72 cycles mémoire pour 2 ports mémoire;

(5) LS-LA trouve une solution en plus de 100 cycles mémoire pour 1 port mémoire.

De ce fait, GCDO trouve une solution au maximum à une distance relative de $(42-38)/38 = 10.52\%$ du budget optimum. Par ailleurs, d'après le résultat de Graham [**94**], il ne peut pas exister une solution à 1 port en dessous de $100/2 = 50$ cycles mémoire. De ce fait, la solution GCDO est optimale en nombre de ports mémoire utilisés et seulement 10.52% approchée en temps. Nous illustrons en Figure 17 le raisonnement formel conduisant à la démonstration de quasi-optimalité de la solution obtenue par GCDO sur le Segment Protocol Processor d'Alcatel.

4.2. Méthodologie d'exploration formelle. A la lumière de cet exemple, nous voyons l'intérêt à la fois pratique et formel de commencer les explorations en utilisant un algorithme de liste[21]. A partir du nombre de cycles mémoire nécessaires pour exécuter l'ordonnancement trouvé sur une architecture à 1-port, 2-port, 3-port, etc. et connaissant le chemin critique, il est possible de construire des intervalles de budgets pour lesquels l'architecture mémoire est optimale en utilisant la borne de Graham [**94**]. Pour les applications parallèles comportant un faible degré d'exclusivité, des bornes plus précises peuvent être obtenues en divisant le nombre d'accès par le nombre de ports. Connaissant ces intervalles et les objectifs de l'optimisation, il est possible de décider de la faisabilité et des gains à attendre de l'utilisation d'algorithmes avancés tels LCDO ou GCDO.

Dans le cas du Segment Protocol Processor par exemple, il aurait été possible de détecter un potentiel de réduction de 5 à 2 ports mémoire pour un budget de 50 cycles mémoire, ce qui aurait sans aucun doute intéressé le concepteur. **Contrairement à ce qui est le cas pour un objectif temporel pur, nous devinons ainsi qu'il n'existe pas de limite aux gains possibles en utilisant une exploration globale par rapport à une approche locale en optimisation coût $\times$ temps (C$\times$T).** Pour une application infiniment grande comportant une infinité de groupes de base et une infinité de conflits entre eux, les réductions de coûts possibles sont potentiellement infinis à proximité du chemin critique.

En supposant maintenant l'utilisation de mémoires à 1 port uniquement et connaissant un intervalle d'optimalité architectural $[T_{min}, T_{max}]$ pour un nombre total de ports p, il est possible d'estimer à

21. de préférence le plus mauvais possible afin d'obtenir les bornes inférieures en temps les plus précises possibles.

Algorithme	Complexité	ANSI-C ln/s	ANSI-C ln/mn
LS-LA	$O(nln(n))$	6,382,030	-
CD-SE	$O(nm)$	22,361	173,206
D&C-ACLS	$O(n^{3/2})$	**10,000**	153,262
D&C-GCDO(k)	$O(n^{3/2}m^{1/2}ln(m^{1/2}))$	926	**18,097**
AC-LS	$O(n^2)$	1,000	7,746
BC-LS	$O(n^2)$	1,000	7,746
D&C-LCDO(k)	$O(n^2ln(n^{1/2})ln(m^{1/2}))$	392	2,256
D&C-FGB	$O(n^{5/2}m^{1/2})$	131	512
GCDO(k)	$O(n^2mln(m))$	117	409
LCDO(k)	$O(n^3ln(n)ln(m))$	49	152
GCDO	$O(n^2m^2)$	71	197
LCDO	$O(n^3ln(n)m)$	35	91
CDO (FGB)	$O(n^4m)$	22	50

TAB. 2. *Principaux algorithmes de la boîte à outils d'ordonnancement–placement d'ACROPOLIS, leurs complexités théoriques et leurs vitesses de conception multicritère et flexible théoriques exprimées en nombre de lignes ANSI-C par seconde (respectivement par minute). Pour ce calcul, nous supposons un accès à la mémoire partagée par ligne ANSI-C.*

un haut niveau d'abstraction par exemple la consommation électrique relative de cette solution courante par rapport à la solution (quasi-)optimale:

$$\frac{P(T,p)}{P_{opt}(T_{min},p_{min})} \approx \omega_{mem} \times [\frac{p_{min}}{p}] + \omega_{bus} \times [\frac{T^2}{T_{min}^2}\sqrt{\frac{p}{p_{min}}}] + (\omega_{logic} + \omega_{clock}) \times [\frac{T^2}{T_{min}^2}] \tag{3}$$

En appliquant cette formule à la répartition énergétique d'un commutateur ATM[22] et à la comparaison des solutions LS et GCDO sur architecture à 2 ports mémoire, nous obtenons:

$$\frac{P(72,2)}{P_{opt}(42,2)} \approx 3\% \times [\frac{875mJ}{750mJ}] + 97\% \times \frac{72^2}{42^2} = 2.89 \tag{4}$$

En comparant les solutions CDO et GCDO sur architecture à 2 ports mémoire, nous obtenons:

$$\frac{P(58,2)}{P_{opt}(42,2)} \approx 3\% \times [\frac{750mJ}{750mJ}] + 97\% \times \frac{72^2}{58^2} = +57\% \tag{5}$$

En comparant LS et GCDO pour une synthèse d'architecture à 42 cycles mémoire, nous obtenons:

$$\frac{P(42,8)}{P_{opt}(42,2)} \approx 3\% \times [\frac{1500mJ}{750mJ}] + (70\% \times \sqrt{\frac{8}{2}} + 27\%) = +73\% \tag{6}$$

A partir de ces estimations, **nous concluons que GCDO est un algorithme extrêmement compétitif à la fois pour la programmation d'architectures fixes et la synthèse d'architectures spécialisées. En unifiant ces deux problèmes sans perte sur la qualité des résultats, GCDO est à notre avis un candidat idéal pour la conception de plates-formes à mémoire et interconnexion (partiellement) configurables et programmables** [**39**] [**184**]**.** De telles plates-formes apparaissent actuellement dans le secteur du multimedia [**109**].

4.3. Méthodologie d'exploration intelligente. Nous avons introduit au long de ce chapitre l'existence de nombreuses variantes algorithmiques pour un ordonnancement–placement multi-critère, flexible et maîtrisé. Afin de comparer ces algorithmes à l'Etat de l'Art en compilation embarquée [**270**] et en exploration en général, nous avons développé plusieurs algorithmes d'ordonnancement par liste et également un algorithme d'évolution stochastique dont nous reportons une présentation détaillée au Chapitre 8 de la Partie II. En Table 2, nous présentons les performances des principaux algorithmes d'ordonnancement–placement ainsi obtenus, que nous avons intégré dans une boîte à outils utilisable par notre environnement ACROPOLIS.

Les techniques de liste [**20**] comportent une hypothèse implicite sur la solution recherchée car elles partent du principe qu'un problème résolu localement en temps n'aura pas de conséquences (encore) plus défavorables dans un temps éloigné, même lorsque la fonction coût est étendue pour tenir compte **globalement** des problèmes de transfert et de stockage [**117**]. De même, les techniques génériques issues de l'intelligence artificielle [**139**] [**289**] [**108**] [**49**] [**8**] [**18**] et l'évolution stochastique en particulier [**232**] [**61**] [**254**] [**301**] traversent partiellement l'espace de recherche pour ne retenir rien d'autre que le meilleur minimum local rencontré. Pour ces raisons, nous considérons que les approches autres que CDO n'effectuent qu'une exploration *partielle* de l'espace de recherche [**85**]. Cependant, nous ne pouvons pas ignorer leurs performances (en temps d'exécution **et** qualité de solution) parfois éblouissantes. Après de nombreuses expériences sur des applications réelles et des cas théoriques que nous avons construits d'abord sur papier [**186**] puis en utilisant les extensions interface utilisateur LEDA [**158**] comme laboratoire d'expérimentation placé au coeur de notre environnement, nous sommes arrivés à la conclusion que chaque algorithme dispose en pratique d'une niche applicative pour laquelle il est le meilleur choix (au sens d'aide à la décision multi-critère qualité $>>$ temps).

Nous pouvons ainsi proposer une classification (très) pragmatique de l'ensemble des algorithmes d'ordonnancement–placement multi-critère et flexibles actuels, ce qui est illustré en Figure 18. Afin d'utiliser au mieux notre boîte à outils ACROPOLIS, nous utilisons activement cette classification. Ainsi, nous proposons de toujours commencer un nouveau problème de conception par l'utilisation d'un algorithme de liste car il donnera (très) rapidement une bonne idée sur les problèmes posés par la spécification et une bonne idée sur la faisabilité de les résoudre en utilisant une extension des bornes théoriques[23] de l'analyse par *graphe de flots de données* étudiées en détail dans [**94**] [**152**] au nouveau contexte de l'optimisation du produit temps $\times$ coût $(C \times T)$[24]. Pour nos besoins de recherche propre, ces bornes se sont avérées essentielles pour mettre en évidence les failles de nos premiers prototypes d'ordonnancement–placement [**179**]. Les bornes $C \times T$ ajoutées au développement de cas théoriques dont la solution est connue à l'avance [**186**] nous ont également permis de contourner les (nombreuses) difficultés (à notre avis prohibitives) posées par la programmation entière et les autres approches dites de référence pour ré-implémenter un ordonnancement-placement multi-critère [**185**] afin de déterminer de façon précise le niveau de qualité atteint par nos algorithmes. Pour le concepteur, ces bornes devraient être toutes aussi utiles pour décider (très) rapidement de s'investir ou non dans un problème d'optimisation de système dont il connaîtra à l'avance les chances de succès.

Nous supposons maintenant que le concepteur décide de mener son optimisation au bout. Tout particulièrement dans le cas d'applications (très) séquentielles où l'algorithme de liste se fait beaucoup piéger, nous lui conseillons d'échanger l'algorithme de liste par un algorithme d'évolution stochastique dès la deuxième tentative. Celui-ci est légèrement plus lent mais toujours meilleur sur les

22. $\omega_{bus} = 70\%$, $\omega_{clock} = 21\%$, $\omega_{logic} = 6\%$, $\omega_{mem} = 3\%$ [**234**].

23. je remercie Mario Arioli pour me sensibiliser à l'importance des bornes théoriques depuis 1992 notamment pour arrêter des processus itératifs [**12**] et pour améliorer la qualité des approximations [**11**].

24. Pour ce faire, une borne pratique immédiate est le produit du temps d'exécution minimum donné par la borne de Graham [**94**] par le coût de l'architecture mémoire minimale que l'on obtient facilement en explorant le placement des données sans aucune contrainte de conflit.

cas plutôt séquentiels. Si la solution est acceptable, on arrête bien sûr immédiatement le processus en un temps comparable à une compilation C/C++ classique.

Si la solution n'est toujours pas acceptable, il va falloir utiliser l'artillerie lourde de cette thèse! Premièrement, on distingue le cas séquentiel du cas parallèle. Dans le cas d'une application plutôt séquentielle, on raffinera alors la solution via la version divise-et-règne D&C-CDO(*k*) puis la version multi-étage CDO(*k*) puis la version CDO dynamique si nécessaire. Dans le cas d'une application plutôt parallèle, on raffinera alors la solution via la version divise-et-règne D&C-GCDO(*k*) puis la version multi-étage GCDO(*k*) puis la version GCDO dynamique si nécessaire. Plus le problème est difficile[25], moins grand il pourra être mais nous restons optimistes car les problèmes de grandes tailles résultent généralement d'une accumulation de nombreuses composantes offrant des opportunités d'optimisations croisées généreuses.

La stratégie d'utilisation de la boîte à outils d'ordonnancement–placement ACROPOLIS décrite ci-dessus étant parfaitement automatisable, nous proposons de l'utiliser pour créer un outil d'aide à la décision interactif et intelligent pour un ordonnancement–placement multi-critère, flexible, maîtrisé et agréable à utiliser.

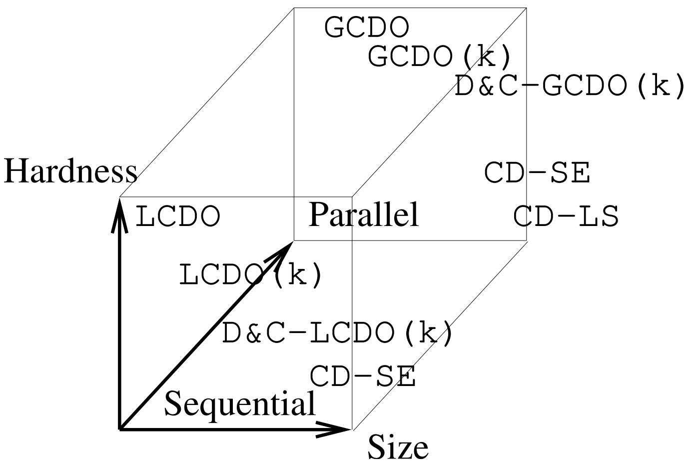


FIG. 18. *Classification des algorithmes d'ordonnancement–placement multi-critère et flexibles*

5. Nouveaux champs d'application

Nous venons de décliner une solution particulièrement intéressante au problème de l'ordonnancement-placement multi-critère et flexible pour architectures constituées d'un processeur et d'un bus et de mémoires (partiellement) configurables. Si de tels systèmes existent bien entendu en pratique, l'une des voies architecturales les plus prometteuses pour les années à venir consiste en une plate-forme multimedia parallèle de type System-on-Chip (SoC) disposant de mémoires et d'interconnexions configurables et d'un petit nombre de processeurs (typiquement 4–8). Les communications de type streaming sont intégrées directement au système et une mémoire SDRAM off-chip est utilisée pour augmenter les capacités de stockage. Nous présentons en Figure 19 une telle plate-forme. Dans la suite de cette section, nous présentons des schémas d'utilisation de l'analyse espace–temps pour ce nouveau contexte.

25. une bonne mesure de difficulté est le rapport entre le nombre de solutions acceptables et le nombre de solutions quasi-optimales (à un ε d'erreur relative près).

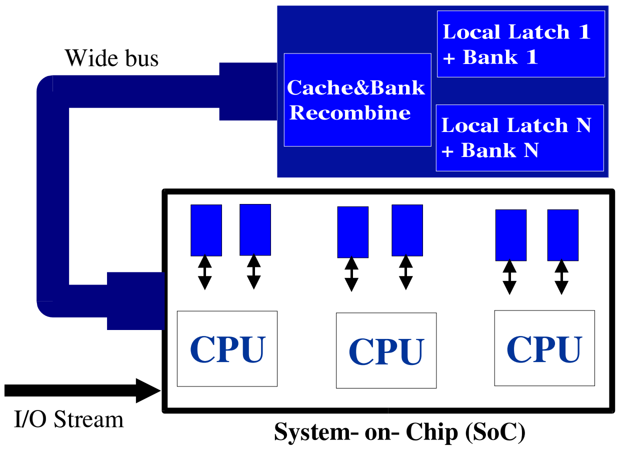


FIG. 19. *Plate-forme multimedia parallèle de type System-on-Chip (SoC) disposant de mémoires et d'interconnexions configurables et d'un petit nombre de processeurs (typiquement 4–8). Les communications de type streaming sont intégrées directement au système et une mémoires SDRAM off-chip est utilisée pour augmenter les capacités de stockage.*

5.1. Entrées/Sorties synchrones de type streaming. Les Entrées/Sorties synchrones de type streaming représentent une contrainte importante pour la conception de systèmes multimedia intégrés [**248**] par exemple utilisant la technologie System-on-Chip (SoC) [**190**] [**211**]. Des techniques tel Philips Research - PHIDEO [**151**] [**264**] existent pour permettre une prise en compte efficace de cette contrainte pour la synthèse de systèmes d'Entrées/Sorties dédiés. Récemment, des techniques logicielles de niveau système opératoire furent introduites pour tenir compte de cette spécificité lors de la programmation de systèmes embarqués [**144**]. Cela dit, il n'existe pas à ce jour de méthodologie unifiant les aspects logiciels et matériels pour permettre la conception d'un système multimedia complet soumis à des contraintes de streaming.

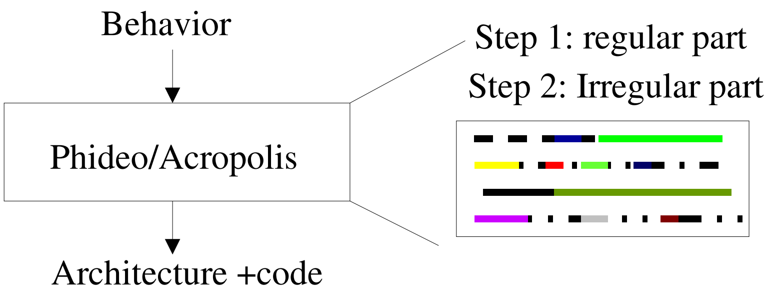


FIG. 20. *tape 1: ordonnancement-placement de la partie régulière en utilisant Philips Research - PHIDEO. tape 2: intégration automatique du sous-système régulier et ordonnancement-placement du système global sans aucune restriction par GCDO de IMEC - ACROPOLIS.*

Nous avons vu en sous-section 2.6 que notre algorithme d'ordonnancement–placement multicritère GCDO simplifie les problèmes d'intégration et la prise en compte d'un nouveau composant (sous-système extérieur). Sur la base de cette simplification, nous proposons en Figure 20 d'utiliser Philips Research - PHIDEO pour l'ordonnancement de la composante (ou sous-système) d'Entrées/

Sorties synchrones. Une fois ce sous-système optimisé, nous proposons d'utiliser GCDO pour effectuer l'ordonnancement-placement du système global. **Ce schéma permet d'unifier les solutions logicielles et matérielles actuellement utilisées pour gérer le streaming. En appliquant cette unification, nous attendons des résultats nettement supérieurs à l'approche classique qui consiste à introduire deux systèmes séparés pour le calcul et pour les Entrées/Sorties synchrones.** Ce résultat attendu de la coopération IMEC-Philips est présenté en Figure 21.

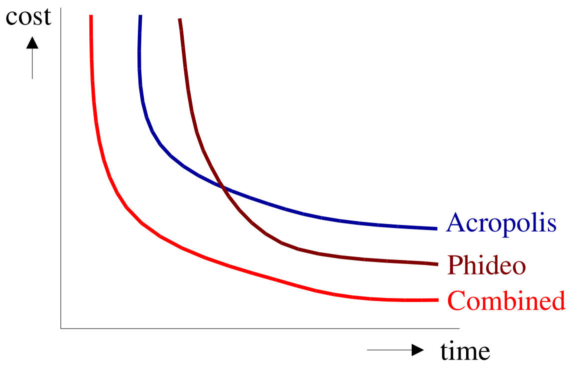


FIG. 21. *Réduction des coûts induite par la conception conjointe du système multimedia et du système de communication synchrone associé par utilisation de IMEC - ACROPOLIS et Philips Research - PHIDEO.*

5.2. Distribution des données. Contrairement aux idées reçues, la technique la plus efficace pour réduire la consommation électrique et le coût des systèmes en général est le parallélisme [**217**] [**168**]. Bien qu'il soit possible de paralléliser l'application avant d'effectuer l'ordonnancement–placement au niveau processeur comme cela fut présenté en Chapitre 1 Section 2, nous montrons en Chapitre 9 de la Partie II qu'il n'est pas judicieux de séparer le partitionnement des données d'une connaissance précise des contraintes architecturales qu'elles représentent. **De ce fait, lors de la compilation d'une application sur une plate-forme constituée de plusieurs processeurs disposant d'une mémoire locale et d'un mécanisme d'adressage global (*ex.* la plupart des machines parallèles commerciales [4], un System-on-Chip (SoC) disposant d'un mécanisme de cohérence mémoire géré par hardware et/ou le système opératoire [39] voire même Internet [259]), il convient d'explorer la parallélisation et le partitionnement des données conjointement.**

En Figure 22, nous présentons un exemple de compilation sur une plate-forme disposant de bus processeur–mémoire configurables et constituée de trois processeurs et six mémoires à 1-port. Pour ce problème apparemment trivial, nous montrons qu'une décomposition de la solution ESTCG trouvée par GCDO en motifs d'accès fortement décorrélés permet de guider la parallélisation et le partitionnement des données vers une solution dont l'efficacité énergétique est 3.08 fois meilleure que celle de la solution mono-processeur et environ 2.25 fois plus efficace qu'une solution parallèle conventionnelle. De telles techniques permettant d'obtenir des gains super-linéaires en parallélisme sont suffisamment rares pour être soulignées [**10**]. De tels dépassements des lois classiques de l'informatique fondamentale – dont la loi d'Amdahl – s'expliquent par une efficacité d'accès aux données nettement plus efficace dans le cas distribué [**288**] (Parallel I/O en anglais).

5.3. Mémoires dynamiques synchrones (SDRAM). Nous présentons en chapitre 2 de la Partie II une synthèse des principaux concepts architecturaux liés aux processeurs multimedia actuels. En section 3 de ce chapitre, nous introduisons les mémoires dynamiques synchrones (SDRAM). En Figure 23, nous rappelons que l'utilisation d'une SDRAM s'accompagne de deux effets indésirables:

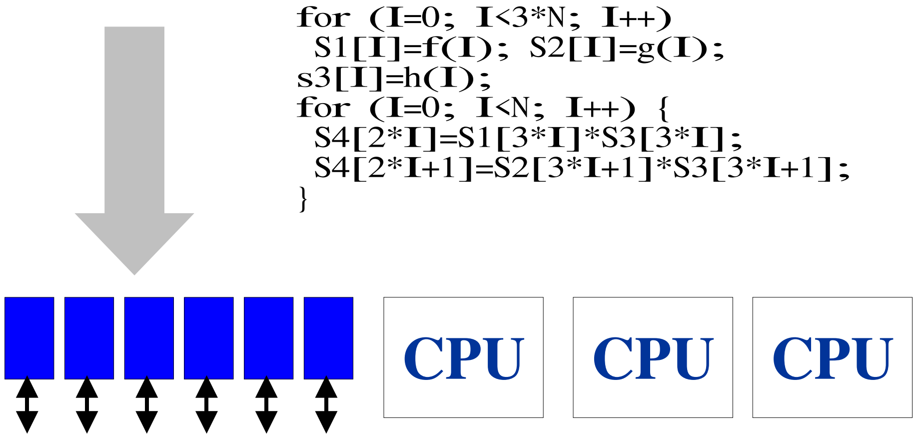


FIG. 22. *Problème d'ordonnancement–placement sur une plate-forme distribuée.*

(1) les données sont stockées dans b bancs mémoire disponibles au mieux tous les b cycles. Afin d'atteindre ce niveau de performance, une parfaite régularité d'utilisation des bancs est nécessaire (Burst-Mode en anglais);
(2) les échanges entre les bancs et le système se font par un bus (très) large provoquant le transfert de nombreux éléments du banc à chaque accès.

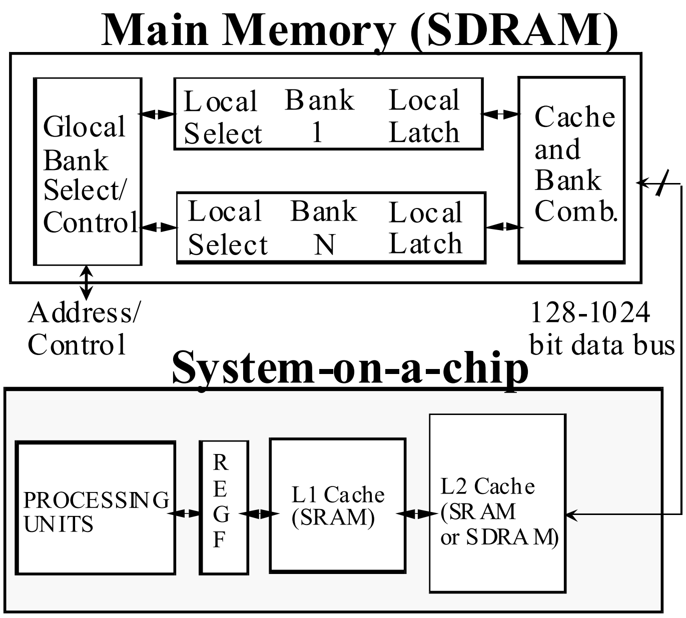


FIG. 23. *Goulot d'étranglement provoqué par l'utilisation d'une mémoire dynamique synchrone (SDRAM) externe.*

Depuis 2000, de nombreuses tentatives en interne IMEC visent à rechercher une solution systématique à ces problèmes [**267**] [**293**]. Une énergie considérable est déployée pour tenter d'étendre les modèles EFG/ECG pour tenir compte des phénomènes espace–temps liés aux SDRAMs. Pourtant, il me semble qu'une SDRAM se caractérise par trois règles simples:

(1) les accès ne sont plus atomiques – *c.a.d* plusieurs accès formant une ligne de page sont systématiquement chargés en parallèle;
(2) chaque page n'est disponible que tous les b cycles – *c.a.d* les accès à des pages différentes sont exclusifs;
(3) les accès aux bancs doivent être régularisés, ce qui peut s'effectuer partiellement avant (partitionnement en bancs) et partiellement après (régularisation à l'intérieur du banc) la phase d'ordonnancement.

Une analyse multi-étage de niveau ligne de page (partitionnement en espace imposé) résout (1). Par ailleurs, une initialisation particulière du graphe de conflit espace–temps (ESTCG) ne faisant apparaître la disponibilité des pages que tous les b cycles résout (2). En première approximation, l'objectif global peut consister à réduire le nombre de ports maximum pour l'ensemble des bancs mémoire, ce qui ne nécessite aucune modification au niveau de GCDO en forçant le choix de paramètres $(\alpha = 1, \beta = 0, \gamma = 0, \delta = 1)$. **L'extension de l'environnement de résolution basé sur le graphe de conflit espace–temps (ESTCG) pour intégrer les phénomènes espace–temps de type SDRAM à bancs symétriques est simple et intuitif, ce qui est loin d'être le cas sur un modèle conventionnel de type EFG/ECG.** En Figure 24, nous présentons un graphe de conflit espace–temps (ESTCG) initialisé pour l'ordonnancement multi-critère des accès à une SDRAM à deux bancs.

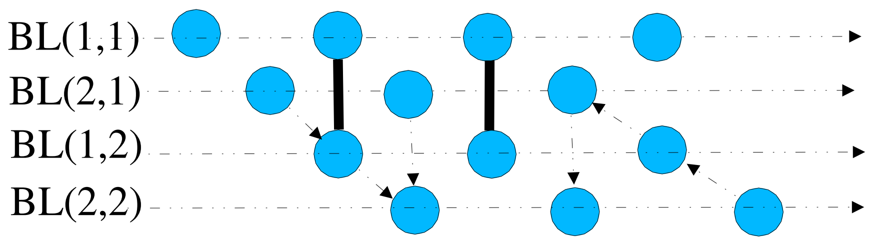


FIG. 24. *Exemple de graphe de conflit espace–temps (ESTCG) initialisé pour l'ordonnancement multi-critère des accès à une SDRAM à deux bancs.*

6. Conclusion

Pour les applications multimedia embarquées récentes, l'optimisation simultanée de la performance, du nombre de ressources, du coût, de la taille et de la consommation électrique du système est nécessaire. La majorité des techniques de compilation existantes sont mal adaptées à ce nouveau contexte car elles ne s'intéressent bien souvent qu'à la performance pure sous des hypothèses simplificatrices telles la régularité temporelle, la linéarité des adresses ou encore l'assignation unique. Les approches s'intéressant à la modélisation du nombre de ressources sont quant à elles limitées à l'utilisation de ressources parfaitement symétriques, ce qui s'accorde mal avec l'hétérogénéité croissante des systèmes émergents. Afin de palier ces limitations sur architecture à mémoire partagée, Sven Wuytack introduit une nouvelle méthodologie d'ordonnancement–placement intéressant **globalement** à l'ensemble des corps de boucle de l'application, appelée Storage Cycle Budget Distribution (SCBD). SCBD se base sur l'algorithme Conflict-Directed Ordering (CDO) de complexité $O(n^4 m)$ où n est le nombre d'opérations mémoire et m mesure la liberté d'ordonnancement moyenne pour ces opérations exprimée en cycles mémoire.

Dans cette thèse, nous avons résolu un problème CDO généralisé. Combiné à des techniques d'approximation spécialisées, nous parvenons ainsi à en réduire la complexité à $O(n^{3/2} m^{1/2} ln(m^{1/2}))$, ce qui autorise une conception interactive et/ou des ré-ordonnancements à la volée. Cette analyse en profondeur nous conduit à l'algorithme d'ordonnancement–placement espace–temps Generalised

Contribution	Maturité	Compilateur	Avant	Après
OPMCF	Industriel	1.00	+30.81%	$\times$ 2.89
Streaming	Théorique	1.00	1.00	+50%
Distribution	Expérimental	2.25	2.25	3.08
SDRAM	Théorique	+90%	1.00	$\geq$ +90%
Bilan 1	Expérimental	4.27	2.94	25.37
Autre outil	Maturité	Compilateur	Avant	Après
CCDL-ADOPT	Expérimental	1.00	3.26	3.26
Bilan 2	Expérimental	4.27	9.58	82.70

TAB. 3. *Comparaison et extrapolation des facteurs de gains énergétiques obtenus par un ordonnancement–placement utilisant un compilateur-paralléliseur conscient de la mémoire,* Acropolis *avant les contributions de cette thèse et* Acropolis *après les contributions de cette thèse.*

Conflict-Directed Ordering (GCDO(k)). Combiné au développement de nouvelles bornes d'optimisation coût $\times$ temps (C$\times$T), nous montrons la supériorité et la quasi-optimalité de GCDO(k) pour résoudre des problèmes réels parallèles et/ou de grande taille.

Sur la base de ces résultats prometteurs, nous montrons la faisabilité d'utiliser GCDO(k) pour l'ordonnancement–placement sur plates-formes multimedia parallèles de type System-on-Chip (SoC) disposant de mémoires et d'interconnexion configurables et d'un petit nombre de processeurs (typiquement 4–8). En supposant des communications de type streaming intégrées directement au système et l'utilisation de mémoires SDRAM off-chip pour augmenter les capacités de stockage, nous proposons des extensions simples et efficaces pour permettre de tirer le meilleur parti de GCDO(k). Pour ce nouveau contexte initialement imprévu mais néanmoins représentatif des réalités industrielles actuelles, nous évaluons en Table 3 nos contributions par rapport à l'Etat de l'Art et aux travaux entrepris précédemment en interne IMEC. En ordonnancement–placement multi-critère et flexible (OPMCF), nos contributions peuvent représenter des gains de 2.89 soit 2.20 fois mieux que la situation avant cette thèse. En streaming, nos contributions permettent de traiter des problèmes auparavant impossibles à traiter avec un potentiel de gain de 50%[26]. En distribution des données, nos contributions peuvent signifier des gains super-linéaires de 3.08 sur trois processeurs. En ce qui concerne les SDRAMs, nous attendons de notre modèle GCDO(k) une résolution bien meilleure que celle des compilateurs conscients de la mémoire et donc d'obtenir un gain au moins équivalent de l'ordre de 40% en temps, ce qui signifie 90% de gains énergétiques pour un commutateur ATM. **En extrapolant l'ensemble de ces résultats pour la conception d'un SoC complexe (Bilan 1), nous montrons l'importance de nos contributions pour maintenir la compétitivité de la technologie** Acropolis **par rapport à l'Etat de l'Art en compilation–parallélisation consciente de la mémoire. Dans ce contexte, nous prévoyons la capacité de GCDO(k) à réduire la consommation électrique de systèmes (très) complexes par un ordre de magnitude et plus, comparé à une solution mono-processeur utilisant un compilateur conventionnel.** En l'état de l'environnement Acropolis au début de cette thèse, de tels gains d'un ordre de magnitude en consommation électrique ne pouvaient être envisagés qu'au prix de dix-huit mois d'efforts d'un architecte système à temps plein. En ajoutant maintenant l'outil d'optimisation de cache CCDL-ADOPT (Cache Conscious Data Layout and ADdress OPTimisation) présenté en Section 6 du Chapitre 2 de la Partie II, nous illustrons le potentiel de l'approche sur une application SoC dominée par les communications externes et SDRAM qui, de plus, offrirait de belles

26. En supposant que le système multimédia et le système d'entrées/sorties consomment autant d'énergie, leur fusion peut au mieux permettre de diviser la consommation totale par deux et au pire de ne rien changer, soit une moyenne de 50%.

opportunités de réutilisation de données dans des caches mémoire. Nous donnons cette dernière extrapolation (Bilan 2) afin de confirmer l'absence de limite aux gains possibles en optimisation coût $\times$ temps (C$\times$T) sur applications et plates-formes complexes, sans que de tels niveaux de résultat n'aient été encore observés manuellement ou à l'aide d'outils par IMEC. L'objectif est de convaincre le lecteur du fort potentiel de recherche encore présent dans ce domaine.

En particulier, nous avons rencontré – en Section 3.2 pour les aspects multi-étage, en Section 3.3 pour les aspects divise–et–règne, en Section 5.2 pour les aspects distribution de données et enfin en section 5.3 pour la définition de groupes de données de base placés en bancs mémoire – différents aspects et critères de **partitionnement espace–temps**. A ce jour, l'état des recherches dans ce domaine est insuffisant pour pouvoir proposer une méthodologie et des techniques permettant d'unifier ou du moins de systématiser une résolution de ces problèmes de **partitionnement espace–temps** conjointement. Pourtant, en maîtrisant le partitionnement en groupes de base et le placement dans les bancs, en décomposant rapidement le problème en sous–problèmes décorrélés pour permettre une résolution divise–et–règne puis en repoussant les limites de la résolution multi-étage pour chacun de ces sous-systèmes, nous pensons que des potentiels de gains importants en qualité et surtout en complexité des systèmes concevables sont encore possibles. Afin d'atteindre de tels niveaux de Haute Performance, nous pensons qu'une étude de l'interaction entre ces différents aspects et critères de **partitionnement espace–temps** serait nécessaire afin de déterminer le compromis le plus efficace possible pour une application donnée. En repoussant cette limite de complexité, la limite des réductions de coût possibles sur base automatique serait bien entendu également repoussée. Comme nous l'avons illustré et pratiquement démontré, ces deux facteurs vont de paire en optimisation coût $\times$ temps (C$\times$T).

Deuxième partie

International Publications

CHAPITRE 1

International Workshops

1. INRIA-CRI-PRiSM Project A3 Seminar

This is the abstract of our INRIA-CRI-PRiSM Project A3 Seminar given at Rocquencourt, France on January 25, 1999 [**37**].

Pour la plupart des applications multimédia temps-réel récentes (telles que la vidéo, le traitement d'images et les télécommunications), la manipulation de données complexes est l'un des (voire même le) critère majeur du coût global du système. C'est aussi le cas lorsqu'une telle application est compilée sur une architecture parallèle multi-processeurs (comme le Trimedia de Philips, le TI C60, le C80 ou le Chromatic/Mpact) ou sur un processeur générique avec des extensions multimédias (tel que le MMX d'Intel). Ce phénomène est principalement dû à l'impact de l'organisation de la mémoire et aux transferts de données entre les éléments de stockage des processeurs et les parties opératives. La gestion de la mémoire a ainsi un effet direct sur le temps de calcul de l'application et sur la consommation du système.

Habituellement, les concepteurs de systèmes choisissent une solution unique parmi l'énorme espace de solutions possibles pour la spécification en langage C d'une spécification abstraite.

Cette solution est ensuite compilée grâce à des outils classiques de compilation parallèle. Pour pouvoir améliorer la spécification initiale du système, il est aujourd'hui nécessaire de pouvoir aider un concepteur pendant la compilation par des retours rapides sur les choix qu'il opère à un niveau algorithmique, sans avoir besoin d'implanter le système à un plus bas niveau. En effet, dans ce domaine, pour des questions d'efficacité la compilation génère conjointement des parties logicielles et des parties matérielles. Ces dernières sont difficiles à caractériser (en consommation, surface et performances) avant leur conception définitive.

Notre équipe a déjà une grande expérience de l'influence de la structure du système et de la description de l'application sur le temps de calcul et sur la consommation; nous avons développé un outil de gestion de la mémoire pour la compilation sur silicium (ATOMIUM) qui travaille à un plus bas niveau qu'un compilateur parallèle de systèmes. Partant de cette expérience, nous développons à l'IMEC un outil d'aide à la compilation conjointe (ACROPOLIS), qui permettra de prendre les bonnes décisions, en explorant efficacement l'espace de solutions et en déduisant une spécification en C optimisée. Nous avons obtenu des résultats prometteurs en appliquant cette approche manuellement à des algorithmes assez complexes de traitement d'image.

2. Dagstuhl Seminar on Instruction-Level Parallelism and Parallelizing Compilation

This abstract was published in [**181**] *following the Instruction Level Parallelism and Parallelizing Compilation Seminar, Dagstuhl, Germany, April 18–23, 1999.*

For most advanced real-time multimedia processing applications (video, imaging, telecom), the manipulation of complex data has a major or even dominant effect on the cost of the global system, also when compiled to (parallel) multi-media processors like the Philips' TriMedia, TI C60 or C80, Chromatic/Mpact and even on general-purpose processors with multi-media extensions like the Intel MMX. This is mainly due to the large impact of the memory organisation and the global data transfer between processor storage and data-path units on the cycle count and system power consumption.

Currently, due to design time restrictions, the system designer usually has to select – on an ad-hoc basis – a single promising path in the huge decision tree from abstract specification to more refined C specification, which is then input to the conventional (parallel) compiler environments. To alleviate this situation, there is a need for more support during the compilation and also for fast and early feedback at the algorithm level without going all the way to the final code. When the design space has been sufficiently explored at a high level and when the most promising candidate has been identified, also the final choice has to be coded in the transformed (C) specification.

Therefore, based on our large experience in hardware oriented memory management (see Atomium project), we are developing a tuned methodology, formal models and system-level compilation techniques which are intended as a precompilation stage to the conventional (parallel) compiler environments. In addition, we are working on prototype tools and an environment supporting the most error-prone and critical tasks in this data storage and transfer exploration and optimisation. Promising results have been achieved by applying this approach manually to a number of complex real-life video and image processing algorithms.

3. INRIA-ICPS-ENSL Workshop on Compilation and Automatic Parallelisation

This abstract was published in [**180**] *following the INRIA-ICPS-ENSL Workshop on Compilation and Automatic Parallelisation, St. Narbor, France, November 18–20, 1999.*

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight in [**212**] to minimize the number of resources required in the high-level synthesis of high-throughput application-specific integrated circuits (ASICs). FDS was later chosen as the basis of the Philips Research - PHIDEO silicon compiler [**151**], which resulted in an in-depth study of FDS implementation issues in [**276**] [**278**] [**274**] [**275**]. State-of-the-art in FDS reports a runtime complexity of $O(m^2n^2)$, where n is the number of operations and m is the average time window of each operation, and a feasibility of solving problems of n equals 100 in one minute using the computers of that time [**277**].

Recently, based on the FDS algorithm, an extension called conflict-directed ordering (CDO) was proposed in [**290**] to minimize the required memory bandwidth of embedded multimedia systems. The CDO algorithm involves an important step in the IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration approaches [**170**] [**38**]. Because the CDO solution process requires global system-level analysis, pre-existing optimisation techniques do not fully hold and even a *localised* (also called *dynamic* in the litterature, [**2**] [**110**] [**74**]) implementation is left with a runtime complexity of $O(mn^3 ln(n))$. This allows to solve problems for $n = 100$ in one hour, even using today's workstation series. Moreover, because the number of operations found in real-life embedded applications is today growing well beyond n equals 100 [**54**] and also because parallelism is often introduced which further increases the number of concurrent operations, a novel fast and effective algorithm for the CDO kernel is needed. Also to improve the designer interaction further, it is desirable to have a controlable trade-off between user time and cost effect [**268**].

In this presentation, we revisit the original idea of Paulin and Knight [**212**] to show that FDS is in fact a projected solution to a more general (*multi-dimensional*) *space–time* partitioning problem. Based on this *space–time* formulation, we introduce a generalized conflict-directed ordering (GCDO) algorithm that instead of transforming data that statistically require too much bandwidth can go down to the precision of transforming (data,time) tuples directly. The high resolution of this technique allows (at worst[1]) a runtime complexity of $O(mn^3)$ because a look-ahead is not mandatory any more for precision enhancement. Moreover, the new size of order 10,000 of the GCDO multi-dimensional search space allows to fully benefit from the latest large-scale scheduling and partitioning techniques ([**227**] [**90**] [**122**] [**96**]) for a *multilevel* k-way parametrizable runtime complexity of $O(kn^2)$. Experiments of using the original CDO and GCDO(k) on two real-life multimedia applications confirm that the design time complexity can be reduced by two orders of magnitude and more. Moreover, an interactive choice of parameter k is shown to be an appropriate solution for designing large-scale and/or parallel data-dominated systems under stringent real-time constraints.

Acknowledgments

We thank Denis Vanderstraeten (K.U Leuven), Fabien Coelho (E.N.S.M Paris) and Antoine Fraboulet (E.N.S. Lyon) for their input on graph algorithms, ad-hoc clustering and polynomial-time discrete optimisation. We also wish to thank the European Commission for partly funding this work in the frame of the MEDEA System-Level Methods and Tools (SMT) project.

1. In the current GCDO prototype, each iteration is not fully localised yet and an improved complexity of $O(mn^2 ln(n))$ can still be achieved.

CHAPITRE 2

Architecture Issues and Exploration in Multimedia Processors

This is the fully revised version of our contribution accepted for publication as chapter in the book "Programmable Digital Signal Processors: Architecture, Programming, and Applications", Y. H. Yu (ed.) [**36**].

Storage technology takes the center stage [**146**] in more and more systems because of the eternal push for more complex applications with especially larger and more complicated data types. In addition, the access speed, size and power consumption associated with this storage form a severe bottleneck in these systems (especially in an embedded context). In this chapter, several building blocks for memory storage will be investigated, with the emphasis on internal architectural organization. After a general classification of the memory hierarchy components in section 1, cache architecture issues will be treated in section 2, followed by main memory organization aspects in section 3. The main emphasis will lie on modern multi-media and telecom oriented processors, both of the microprocessor and DSP type.

Apart from the storage architecture itself, the way data are mapped to these architecture components are as important for a good overall memory management solution. Actually, these issues are gaining in importance in the current age of deep submicron technologies where technology and circuit solutions are not sufficient on their own to solve the system design bottlenecks. Therefore, the last 3 sections are devoted to different aspects of data transfer and storage exploration: source code transformations (section 4), task versus data parallelism exploitation (section 5) and memory data layout organisation (section 6). Realistic multimedia and telecom applications will be used to demonstrate the impressive effects of such techniques.

1. HIERARCHICAL MEMORY ORGANISATION IN PROCESSORS

The goal of a storage device is in general to store a number of n-bit data words for a short or long term. These data words are sent to processing units (processors) at the appropriate point in time (cycle) and the results of the operations are then written back in the storage device for future use. Due to the different characteristics of the storage and access, different styles of devices have been developed.

1.1. General principles and storage classification. A very important difference can be made between memories (for frequent and repetitive use) for short term and long term storage. The former are in general located very close to the operators and require a very small access time. Consequently, they should be limited to a relatively small capacity ($<$32 words typically) and are taken up usually in feedback loops over the operators (*e.g.* RA-EXU-TRIA-BusA-RA in Figure 1). The devices for longer term storage are in general meant for (much) larger capacities (from 64 to 16M words) and take a separate cycle for read or write access (Figure 2). Both categories will be described in more detail in the next subsections. A full memory hierarchy chain is illustrated in Figure 3 (for details see section 3.3.).

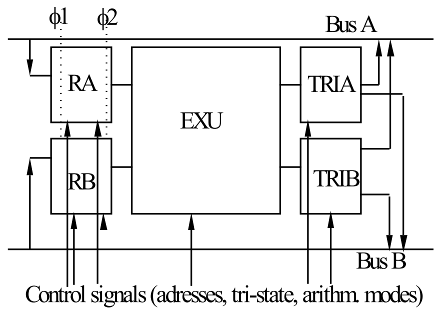


FIG. 1. *Two register files RA and RB in feedback loop of data-path*

Six other important distinctions can be made using the tree-like "genealogy" of storage devices presented in Figure 4.

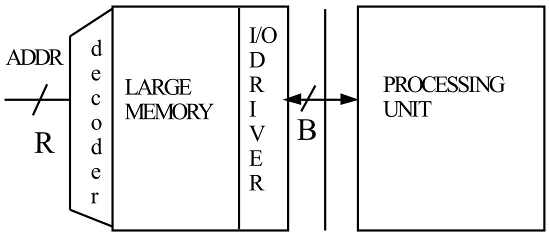


FIG. 2. *Large capacity memory communicating with processing unit*

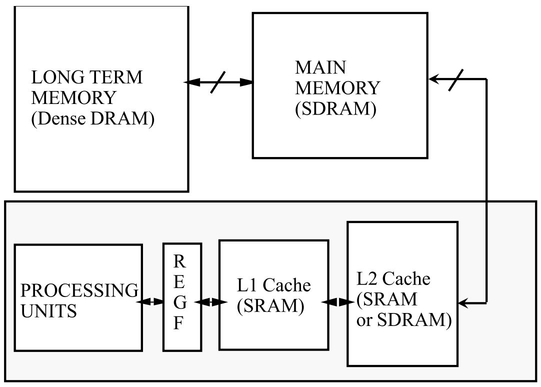


FIG. 3. *Typical memory hierarchy in processors*

(1) **Read-only or Read/write (R/W) access:** some memories are used only to store constant data (such as ROMs). Good alternatives for this are e.g. programmable logic arrays (PLA) or multi-level logic circuits, especially when the amount of data is relatively small. In most cases, data needs to be overwritable at high speeds which means that read and write are treated with the same priority (R/W access) such as in random-access memories or RAMs. In some cases, the ROMs can be made "Electrically Alterable" (=Write-few) with high energies (EAROM) or "Programmable" by means of e.g. fuses (PROM). Only the R/W memories will be discussed further on.

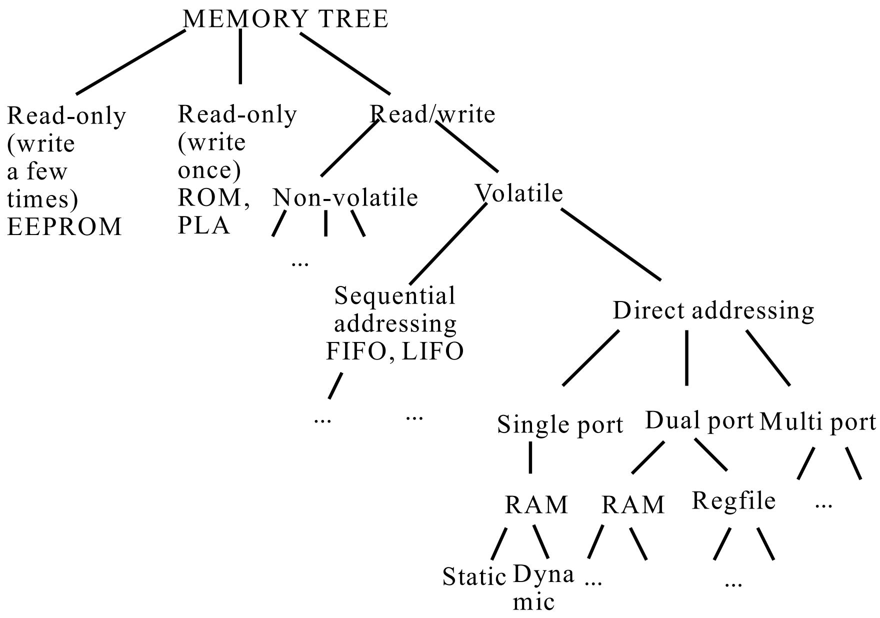


FIG. 4. *Storage classification*

(2) **Volatile or not:** for R/W memories, usually, the data are removed once the power goes down. In some cases, this can be avoided but these non volatile options are expensive and slow. Examples are magnetic media and tapes which are intended for slow access of mass data. We will restrict ourselves to the most common case on chip, namely volatile.

(3) **Address mechanism:** some devices require only sequential addressing, such as the first-in first-out (FIFO) queue or first-in last out (FILO) or stack structures discussed in subsection 1.3, which put a severe restriction on the order in which the data are read out. Still, this restriction is acceptable for many applications. A more general, but still sequential access order is available in a pointer-addressed memory (PAM). In the PAM. The main limitation is that each data value is both written and read once in any statically predefined order. However, in most cases the address sequence should be random (including repetition). Usually, this is implemented with a direct addressing scheme (typically called a Random Access Memory or RAM). Then, an important requirement is that in this case, the access time should be independent of the address selected. In many programmable processors, a special case of random access based buffering is realized, exploiting comparisons of tags and usually also including (full or partial) associativity (in a so-called cache buffer).

(4) **The number of independent addresses and corresponding gateways (busses) for access:** this parameter can be one (single-port), two (dual-port) or even more (multi-port). Any of these ports can be for reading only, writing only or R/W. Of course, the area occupied will increase considerably with the number of ports.

(5) **Internal organization of the memories:** the memory can be meant for capacities which remain small or which can become large. Here a trade-off is usually involved between speed and area efficiency. The register-files in subsection 1.2 constitute an example of the fast small capacity organizations which are usually also dual- or even multi-ported. The queues and stacks in section 1.3 are meant for medium-sized capacities. The RAMs in section 3 can become extremely large (up to 256 Mbit for the state-of-the-art) but are also much slower in random access.

(6) **Static or dynamic:** for R/W memories, the data can remain valid as long as VDD is on (static cell) or the data should be refreshed about every ms (dynamic cell). Circuit level issues are discussed in overview articles like [**80**] for SRAMs, and [**112**] [**221**] for DRAMs.

In the following (sub)sections, the most important read/write type memories and their characteristics will be investigated in more detail.

1.2. Register-file and local memory organization. In this subsection, we discuss the register-file and local memory organization. An illustrative organization for a dual port register-file with two addresses where the separate read and write addresses are generated from an instruction ROM, is shown in Figure 5. In this case, 2 busses (A and B) are used but only in one direction so the write and read addresses directly control the port access. In general, the number of addresses can be smaller than the number of port(s) and the busses can be bi-directional. Additional control signals decide whether to write or read and for which port the address applies. The register-file of Figure 5 can be used very efficiently in the feedback loop of a data-path as already illustrated in Figure 1. In general it is used only for the storage of temporary variables in the application running on the data-path (sometimes also referred to as execution unit). This is true both for most modern general-purpose RISCs and especially for modern multi-media oriented signal processors which have regfiles up to 128 locations. For multi-media oriented VLIW processors or modern super-scalar processors, regfiles with a very large access bandwidth are provided, up to 17-ports (see e.g. [**116**]). Application-specific instruction-set processors (ASIPs) and custom instruction-set processors make heavy use of regfiles for the same purpose. It should be noted that although it has the clear advantage of very fast access, the number of data words to be stored should be minimized as much as possible due to the relatively area intensive structure of such register-files (both due to the decoder and the cell area overhead). Detailed circuit issues will not be discussed here (see e.g. [**285**]). After this brief discussion of the local foreground memories, we will now proceed with background memories of the sequential addressing type.

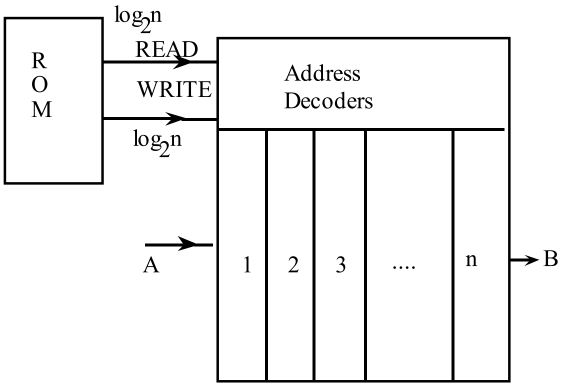


FIG. 5. *Regfile with both R and W address*

1.3. Sequentially addressable memories. Only two of the variety of possible types will be discussed. The memory matrix itself is the same for both and is also identical to the ones used for (multi-port) RAMs.

1.3.1. *First-in first out structures.* Such a FIFO is sometimes also referred to as a queue. In general (Figure 6), it can be used to handle the communication between processing units P1 (source) and P2 (destination) which:

(1) exhibit the same average data throughput

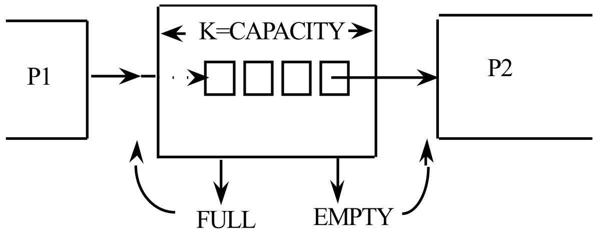


FIG. 6. *Communication between FIFO and 2 processors*

(2) require different production (P1) and consumption (P2) cycles
(3) treat the data in the same sequence

The required capacity K depends on the maximal skew between the consumption and production of the data. Optimization of K is possible if the ”schedules” of P1 and P2 can be delayed relative to one another. The main principles of the organization of a dual-port FIFO, with one read RPTR and one write bus, are shown in Figure 7. Notice the circular shift-registers (pointers) for both read and write address WPTR selection which contain a single 1 which is shifted to consecutive register locations, controlled by a SHR or SHW signal. The latter are based on external R or W commands. A small FSM is in general provided to supervize the FIFO operation. The result of a check whether the position of the 2 pointers is identical, is usually also sent as a flag to this FSM. In this way, also the fact whether the FIFO is ”full” or ”empty” can be monitored and broadcasted to the ”periphery”.

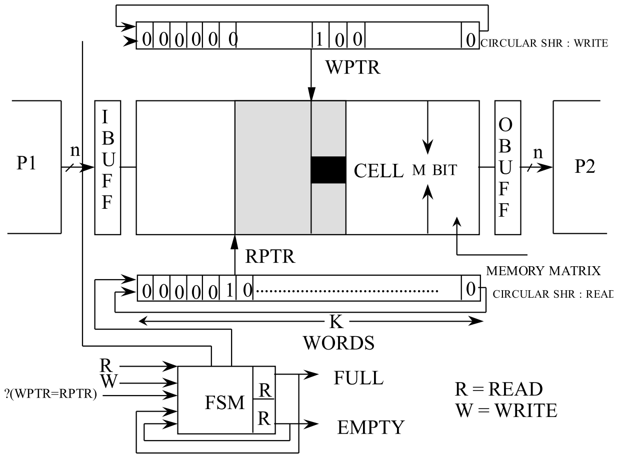


FIG. 7. *Internal FIFO organization*

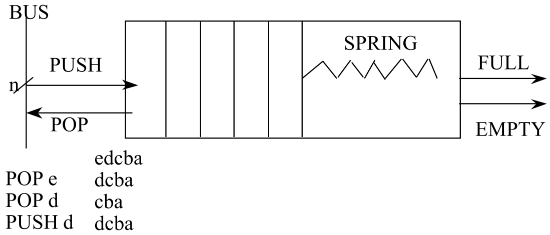


FIG. 8. *Dynamic FILO*

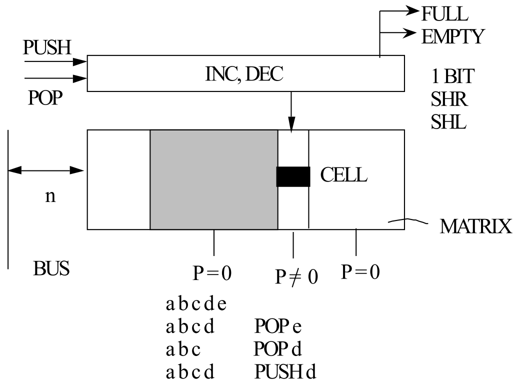


FIG. 9. *Static FILO*

1.3.2. *First-in last-out structures.* Such a FILO is sometimes also referred to as a stack. In general, it can be used to handle the storage of data which has to be temporarily stored ("pushed" on the stack) and read out ("popped" off the stack) in the reverse order. It is regularly used in the design of recursive algorithms, and especially in subroutine stacks.

In principle, the stack can be made "dynamic" (Figure 8) where the data are pushed and popped in such a way that all data move (as if a spring were present). This leads to a tremendous waste of power in CMOS and should be used only in other technologies. A better solution in CMOS is to make the stack "static" as in Figure 9. Here, the only moves are made in a shift register (pointer) which can now move in 2 directions as opposed to the unidirectional shift in the FIFO case.

2. CACHE MEMORY ORGANIZATION

The objectives of this section are : (1) to discuss the fundamental issues about how cache memories operate; (2) to discuss the characteristic parameters of a cache memory and their impact on performance as well as power; (3) briefly introduce the three types of cache misses; and (4) to discuss the differences between hardware and software controlled cache for the current state-of-the-art media and DSP processors.

2.1. Basic cache architecture. In this section a brief overview of the different steps involved in the operation of a cache are presented. We will use a direct mapped cache as shown in Figure 10 for explaining the operation of a cache but the basic principles remain the same for other types of cache memories [**210**]. The following steps happen whenever there is a read/write from/to the cache: (1) Address decoding (2) Selection based on index and/or block offset (3) Tag comparison and (4) Data transfer. These steps are highlighted using ellipse in Figure 10 and 11.

The first step (address decoding) performs the task of decoding the address, supplied by the CPU, into the block address and the block offset. The block address is further divided into tag and index address of the cache. Once the address decoding is complete the index part of the block address is used to obtain the particular cache line demanded by the CPU. This line is chosen only if the valid bit is set and the tag portion of data cache and the block address match with each other. This involves comparison of all the individual bits. Note that for a direct mapped cache we have only one tag comparison. Once this process is done the block offset is used to obtain the particular data element in the chosen cache line. This data element is now transferred to/from the CPU.

In Figure 10, we have used a direct mapped cache, whereas if we had used a n-way set associative cache the following differences will be observed: (1) n tag comparisons instead of one (2) less index bits and more tag bits. This is shown in Figure 11, which shows a 2-way set associative cache. We will briefly discuss some of the issues that need to be considered during the design of cache memories in the next subsection.

2.2. Design choices. In this section we explain in brief some of the important decisions involved in the design of a cache. We shall discuss the choice of line size, degree of associativity, updating policy and replacement policy.

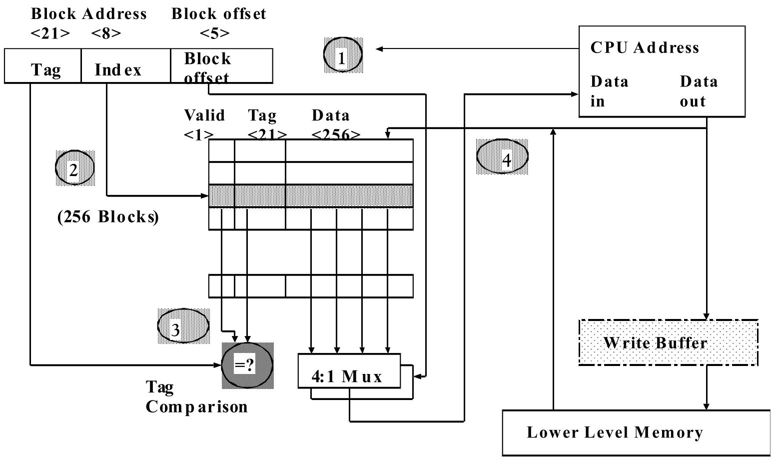


FIG. 10. *8 KB direct mapped cache with 32 byte blocks*

(1) **Line size:** The line size is the unit of data transfer between the cache and the main memory. The line size is also referred to as block size in the literature. As the line size increases from very small to very large, the miss ratio will initially decrease, since a miss will fetch more data at a time. Further increases will then cause the miss ratio to increase, as the probability of using the newly fetched (additional) information becomes less than the probability of reusing the data that is replaced [**245**]. The optimal line size in the above cases is completely algorithm dependent. In general very large line sizes are not preferred since this contributes to larger load latencies and increased cache pollution. This is true for both the general purpose and embedded applications.

(2) **Mapping and Associativity:** The process of transferring data elements from main memory to the cache is termed as "mapping". Associativity refers to the process of retrieving number of cache lines and then determining if any of them is the target. The degree of associativity and the type of mapping having significant impact on the cache performance. Most caches are set associative [**210**] which means that an address is mapped into a set, and then an associative search is made of that set (see Figure 10 and 11). Empirically, and as one would expect, increasing the degree of associativity decreases the miss ratio. The highest miss ratios are observed for direct mapped cache, 2-way associativity is significantly better and 4-way is slightly better still. Further increase in associativity only slowly decreases the misses. Nevertheless, cache with a larger associativity requires more tag comparisons and these comparators constitute a significant amount of total power consumption in the memories. Thus for embedded applications where power consumption is an important consideration, associativities larger than four or eight are not commonly observed. Some architectural techniques for low power cache are presented in [**251**] [**129**].

(3) **Updating policy:** The process of maintaining coherency between two consecutive memory levels is termed as updating policy. There are two basic approaches to updating the main

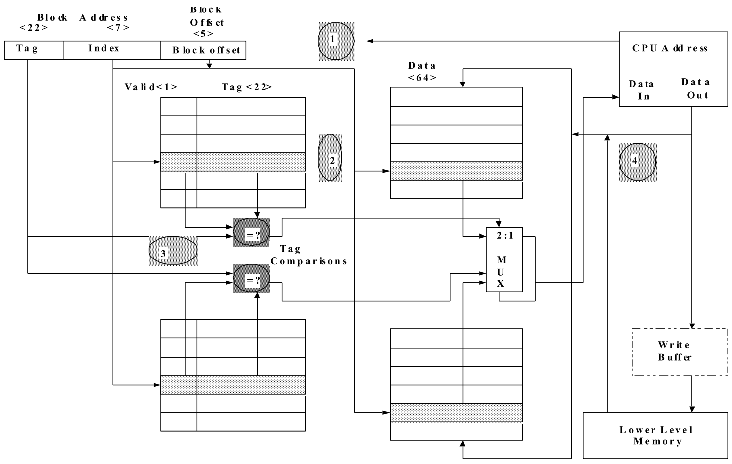


FIG. 11. *8 KB 2-way associative cache with 32 byte blocks*

memory: write through and write back. With write through, all the writes are immediately transmitted to the main memory (apart from writing to the cache); when using write back, (most) writes are written to the cache, and then are copied back to the main memory as those lines are replaced. Initially, write through was the preferred updating policy since it is very easy to implement and solves the problems of coherence of data. But it also generates lot of traffic between various levels of memories. Hence most current state-of-the-art media and DSP processors use a write back policy. A trend towards giving control of the updating policy to the user (or compiler) is observed nowadays [**216**]. Which can be effectively exploited to reduce power, due to reduced write-backs, by compile-time analysis.

(4) **Replacement policy:** The replacement policy of a cache refers to the type of protocol used to replace a (partial or complete) line in the cache on a cache miss. Typically a LRU (least recently used) type of policy is preferred, since it has acceptable results. Current state-of-the-art general purpose as well as DSP processors have hardware controlled replacement policies *i.e.* they have hardware counters to monitor the least recently used data in the cache. In general we have observed that a policy based on compile-time analysis will always have significantly better results than fixed statistical decisions.

2.3. Classification of cache misses. A cache miss is said to have occurred whenever a data requested by the CPU is not found in the cache. There are essentially three types of cache misses namely compulsory, capacity and conflict misses.

(1) **Compulsory misses:** The first access to a block of data is not in the cache, so the block must be brought into the cache. These cache misses are termed as compulsory cache misses.
(2) **Capacity misses:** If the cache cannot contain all the blocks needed during the execution of a program, then some blocks need to be discarded and retrieved later. The cache misses due to discarding of these blocks are termed as capacity misses. Alternatively the total number of misses in a fully associative cache are termed as capacity misses.
(3) **Conflict misses:** For a set associative cache, if too many blocks get mapped to the same set then some of the blocks in the cache need to be discarded. These cache misses are termed as conflict misses. The difference between the total number of cache misses in a direct mapped or a set-associative cache and that of a fully associative cache are termed as conflict misses.

For most real-life applications the capacity and conflict misses are dominant. Hence reducing these misses is vital to achieving better performance and reducing the power consumption. Figure

12 illustrates the detailed cache states for a fully associative cache. Note that, a diagonal bar on an element indicates that the particular element was replaced by the next element, which is in the next column of the same row without a bar, due to the (hardware) cache mapping policy. Hence every diagonal bar represents a cache miss. The main memory layout is assumed to be single contiguous namely array a[] is residing in locations 0 to 10 and array b[] in locations 11 to 21.

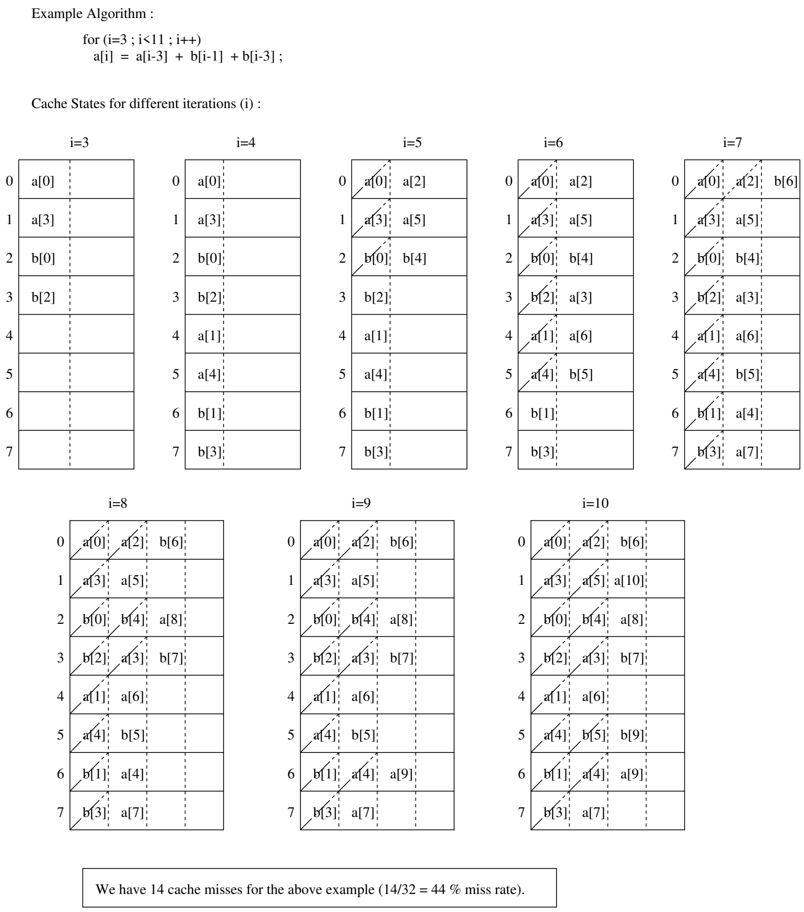


FIG. 12. *Initial algorithm and the corresponding cache states for a fully associative cache.*

We observe from figure 2.3 and figure 2.4 that the algorithm needs 32 data accesses. To complete these 32 data accesses the fully associative cache requires 14 cache misses. Out of these 14 cache misses, 12 are compulsory misses and the remaining 2 are capacity misses. In contrast, the direct mapped cache requires 24 cache misses, as seen in figure 13. This means that out of 32 data accesses, 24 accesses are made to the off-chip memory and the remaining 8 are due to data reuse in the cache (on-chip). Thus for the algorithm in our example we have 10 conflict misses, 2 capacity misses and 12 compulsory misses.

2.4. Hardware versus software caches. Table 1 lists the major differences between hardware and software controlled caches for the current state-of-the-art multimedia and DSP processors. We will briefly discuss these differences:

(1) The hardware controlled caches rely on the hardware to do the data cache management. Hence to perform this task, the hardware uses basic concepts of "cache lines" and "sets".

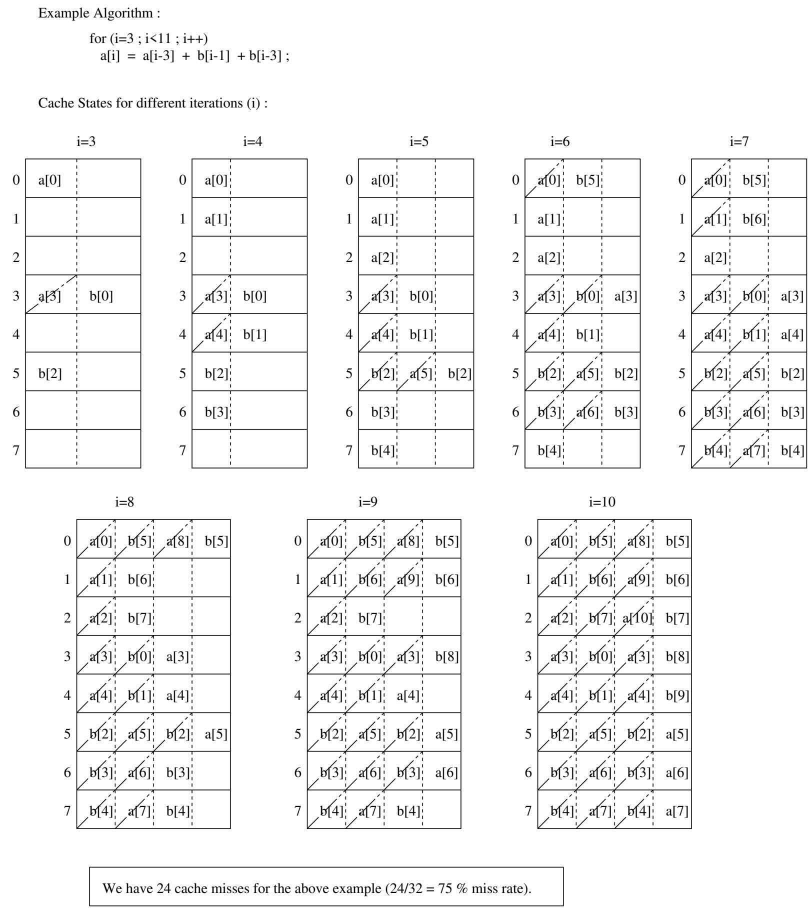


FIG. 13. *Initial algorithm and the corresponding cache states for a direct mapped cache.*

	Hardware Controlled Cache	**Software Controlled Cache**
Basic Concepts	Lines and Sets	Lines
Data Transfer	Hardware	Partly Software
Updating Policy	Hardware	Software
Replacement Policy	Hardware	NA

TAB. 1. *Differences between hardware and software caches for current state-of-the-art multimedia and DSP processors.*

Whereas, for software controlled cache the cache management is done by the compiler (or the user). The complexity of designing a software cache is much less and it requires only a basic concept of unit data transfer which is ”cache lines”.

(2) The hardware performs the data transfer based on the execution order of the algorithm at run time using fixed statistical measures. Whereas, for the software controlled cache this task is performed either by the compiler or the user. This is currently possible using high-level compile-time directives like ”ALLOCATE()” and link time options like ”LOCK” - to lock certain data in part of the cache, through the compiler/linker [**216**].

(3) The most important difference between hardware and software controlled cache is in the way the next higher-level of memory is updated namely the way coherence of data is maintained. For hardware controlled cache the hardware writes data to the next higher level of memory either every time a write occurs or when the particular cache line is evicted. Whereas, for the software controlled cache the compiler decides when and whether or not to write back a particular data element [**216**]. This results in large reduction in the amount of data transfers between different levels of memory hierarchy, which also contributes to lower power and reduced bandwidth usage by the algorithm.

(4) The hardware controlled cache needs extra bit for every cache line for determining the least recently used data, which will be replaced on a cache miss. For software controlled cache, since the compiler manages the data transfer, there is no need for additional bits or a particular replacement policy.

3. MAIN MEMORY ORGANISATION

A large variety of possible types of RAMs for use as main memories has been proposed in the literature and research on RAM technology is still very active, as demonstrated by the results in e.g. the proceedings of the latest International Solid- State Circuits (ISSCC) and Custom Integrated Circuit (CICC) conferences. Summary articles are available in [**44**] [**221**] [**199**]. The general organization will depend on the number of ports but usually, single port structures are encountered in the large density memories. This will also be the restriction here. Most other distinguishing characteristics are related to the circuit design (and the technological issues) of the RAM cell, the decoder and the auxiliary R/W devices. In this section, only the general principles will be discussed. Detailed circuit issues are outside the scope, as mentioned earlier.

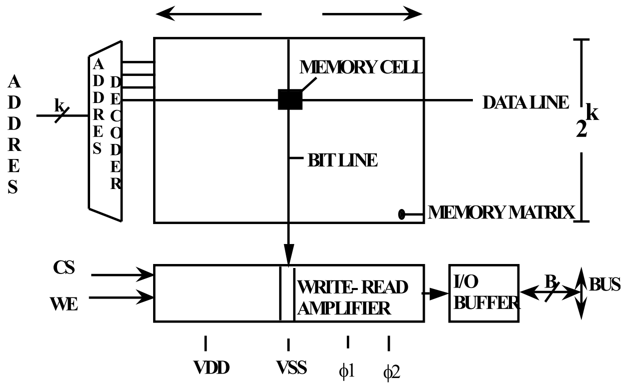


FIG. 14. *Basic floor-plan for B-bit RAM with 2k words*

3.1. Floor-plan issues. For a B-bit organized RAM with a capacity of 2k words, the floor-plan in Figure 14 is the basis for everything. Notice the presence of read and write amplifiers which are necessary to drive or sense the (very) long bit-lines in the vertical direction. Notice also the presence of write-enable WE (for controlling the R/W option) and a chip select (CS) control signal which is mainly used to save power. The different options and their effects are summarized in Figure 15.

The CS signal is also necessary to allow the use of memory banks (on separate chips) as needed for personal computers or work-stations (Figure 16). Notice the fact that the CS1 and CS2 control signals can be considered as the most significant part of the 18-bit address. Indeed, in a way the address space is split up vertically over 2 separate memory planes. Moreover, every RAM in a horizontal slice contributes only a single data bit. For large capacity RAMs, the basic floor-plan of Figure 14 leads

CS	WE	IN	OUT
0	X	X	Z
1	0	X	valid
1	1	valid	Z

FIG. 15. *Summary of control options for Figure 14*

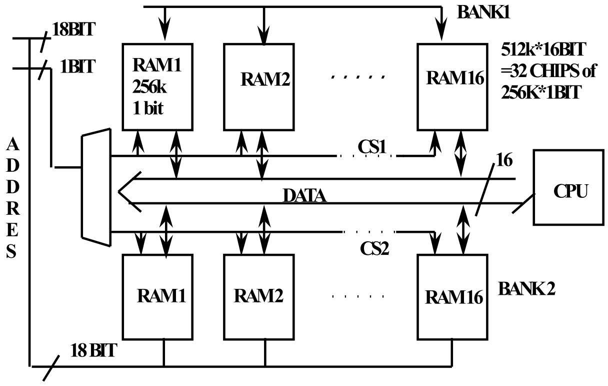


FIG. 16. *Example of Memory bank organization*

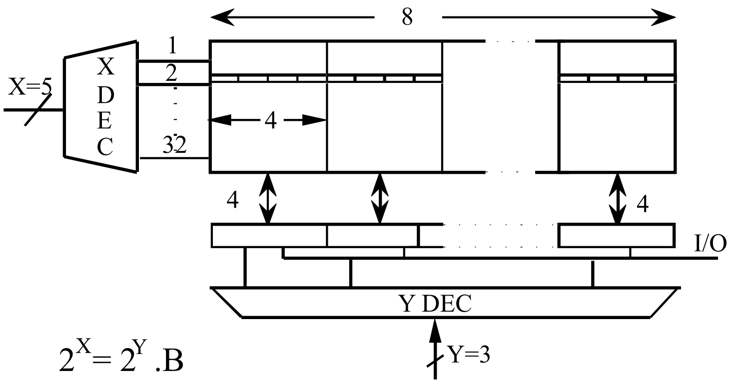


FIG. 17. *Example of post decoding*

to a very slow realisation because of the too long bitlines. For this purpose, the same principle as in large ROMs namely postdecoding is applied. This leads to the use of an X and a Y decoder (Figure 17) where the flexibility of the floor-plan shape is now used to end up with a near square (dimensions x by y), which makes use of the chip area in the most optimal way, and which reduces the access time and power (wire length related). In order to achieve this, the following equations can be applied:

(7) $$x + y = k \quad and \quad x = y + log_2(B)$$

leading to:

(8) $$x = (log_2(B) + k)/2 \quad and \quad y = k - x \; for \; maximal \; \text{"squareness"}$$

This break-up of a large memory plane into several subplanes is very important also for low-power memories. In that case however, care should be taken to enable only the memory plane which contains data needed in a particular address cycle. If possible, the data used in successive cycles should also come from the same plane because activating a new plane takes up a significant amount of extra precharge power. An example floor-plan in Figure 17 is drawn for k=8 and B=4. The memory matrix can be split up in 2 or more parts too, to reduce the length of the word lines also by a factor 2 or more. This results in a typically heavily partitioned floor-plan [**112**] as shown in a simplified form in Figure 18.

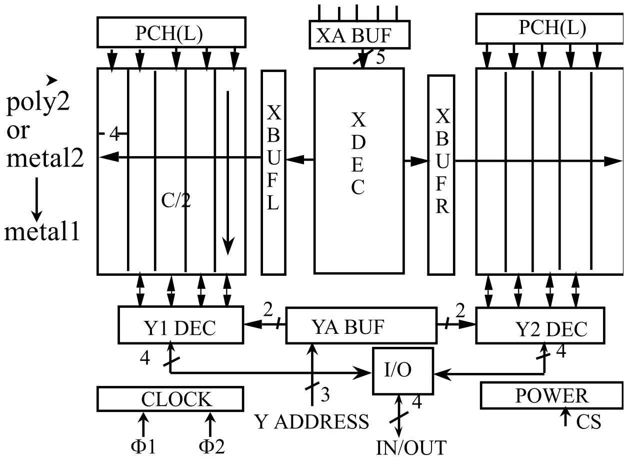


FIG. 18. *Partitioning of memory matrix combined with post decoding*

It should be noted also that the word length of data stored in the RAM is usually matched to the requirement of the application in the case of an on-chip RAM embedded in an ASIC. For RAM chips, this word organisation normally has been standardized to a few choices only. Most large RAMs are 1-bit RAMs. However, with the push for more application-specific hardware, also 4-bit (nibble) and 8-bit (byte) RAMs are now commonly produced. In the near future one can expect also other formats to appear.

3.2. Synchronisation and access times. An important aspect of RAMs are the access times, both for read and write. In principle, these should be balanced as much as possible as the worst case determines the maximal clock rate from the point of view of the periphery. It should be noted that the RAM access itself can be either "asynchronous" or "clocked". Most individual RAM chips are of the asynchronous type. The evolution of putting more and more RAM on chip has lead to a state where stand-alone memories are nearly only DRAMs. Within that category, an important subclass is formed by the so-called synchronous DRAMs or SDRAMs (see e.g. [**79**] [**127**]). In that case the bus protocol is fully synchronous but the internal operation of the DRAM is still partly asynchronous. For the more conventional DRAMs which also have an asynchronous interface, the internal organisation involves special flags, which signal the completion of a read or write and thus the "readiness" for a new data and/or address word. In this case, a distinction has to be made between the address access delay tAA and the chip (RAM) access delay tACS as illustrated in Figure 19. Ideally, tACS = tAA but in practice tACS is the largest. So special tricks have to be applied to approach this ideal.

For on-chip (embedded) RAMs as used in ASICs, typically a clocked RAM is preferred as it is embedded in the rest of the (usually synchronous) architecture. These are nearly always SRAMs. A possible timing (clock) diagram for such a RAM is illustrated in Figure 20.

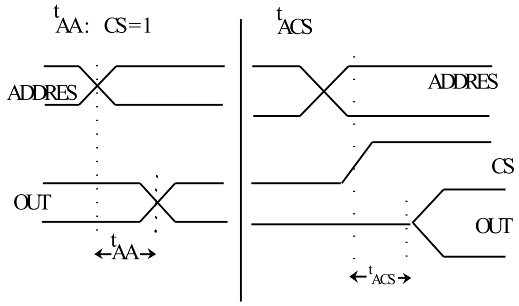


FIG. 19. *Distinction between tAA and tACS for asynchronous RAM*

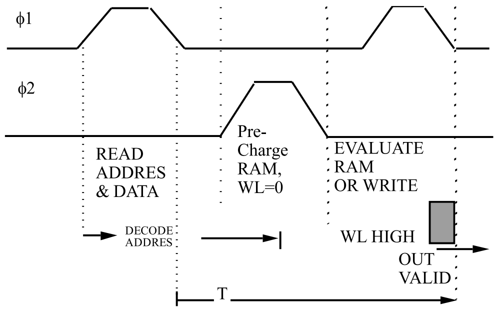


FIG. 20. *Timing diagram for clocked RAM*

3.3. The external data access bottleneck. In most cases, a processor requires one or more large external memories to store the long-term data (mostly of the DRAM type). For data-dominated applications, the total system power cost in the past was for a large part due to the presence of these external memories on the board. Because of the heavy push towards lower power solutions in order to keep the package costs low, and recently also for mobile applications or due to reliability issues, the power consumption of such external DRAMs has been reduced significantly Apart from circuit and internal organization techniques [**297**] [**111**], also technology modifications such as the switch to SOI (Silicon-On-Insulator) [**131**] are considered. Because of all these principles to distribute the power consumption from a few "hot spots" to all parts of the architecture, the end result is indeed a very optimized design for power where every part of the memory organization consumes a similar amount [**241**] [**111**].

It is expected however that not much more can be gained because the "bag of tricks" is now containing only the more complex solutions with a smaller return-on-investment. Note however that the combination of all these approaches indicates a very advanced circuit technology, which still outperforms the current state-of-the-art in data-path and logic circuits for low power design (at least in industry). Hence, it can be expected that the relative power in the non-storage parts can be more drastically reduced still towards the future (on condition that similar investments are done). Combined with the advance in process technology, all this has lead to a remarkable reduction of the DRAM related power: from several W for the 16-32 Mb generation to about 100 mW for 100 MHz operation in a 256 Mb DRAM. Hence, modern stand-alone DRAM chips, which are of the so-called synchronous (SDRAM) type, already offer low power solutions but this comes at a price. Internally they contain banks and a small cache with a (very) wide width connected to the external high-speed bus (see Figure 21) [**126**] [**127**]. So the low power operation per bit is only feasible when they operate in burst mode with large data widths. This is not directly compatible with the actual use of the data in the processor data-paths so without a buffer to the processors, most of the bits that are exchanged would be useless

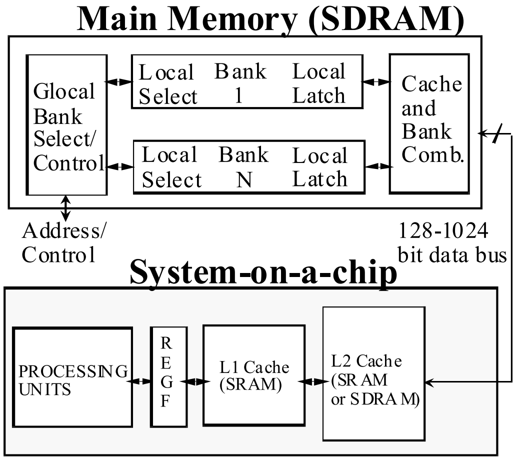


FIG. 21. *External data access bottleneck illustration with SDRAM*

(and discarded). Obviously, the effective energy consumption per useful bit becomes very high in that case and also the effective bandwidth is quite low. Therefore, a hierarchical and typically much more power-hungry intermediate memory organization is needed to match the central DRAM to the data ordering and bandwidth requirements of the processor data-paths. This is also illustrated in Figure 30. The decrease of the power consumption in fast random–access memories is not as advanced yet as in DRAMs but also that one is saturating, because many circuit and technology level tricks have already been applied also in SRAMs. As a result, fast SRAMs keep on consuming on the order of Watt's for high-speed operation around 500 MHz. So the memory related system power bottleneck remains a very critical issue for data-dominated applications.

From the process technology point of view this is not so surprising, especially for submicron technologies. The relative power cost of interconnections is increasing rapidly compared to the transistor (active circuit) related components. Clearly, local data-path and controllers themselves contribute little to this overall interconnect compared to the major data/instruction busses and the internal connections in the large memories. Hence, if all other parameters remain constant, the energy consumption (and also the delay or area) in the storage and transfer organization will become even more dominant in the future, especially for deep submicron technologies.The remaining basic limitation lies in transporting the data and the control (like addresses and internal signals) over large on-chip distances, and in storing them.

One last technological recourse to try to alleviate the energy-delay bottleneck is to embed the memories as much as possible on-chip. This has been the focus of several recent activities, e.g. in the Mitsubishi announcement of an SIMD processor with a large distributed DRAM in 1996 [**260**] (followed by the offering of the "embedded DRAM" technology by several other vendors) and the IRAM initiative of Dave Patterson's group at U.C.Berkeley [**209**]. The results show that the option of embedding logic on a DRAM process leads to a reduced power cost and an increased bandwidth between the central DRAM and the rest of the system. This is indeed true for applications where the increased processing cost is allowed [**284**]. However, it is a one-time drop after which the widening energy-delay gap between the storage and the logic will keep on progressing, due to the unavoidable evolution of the relative interconnect contributions (see above). So on the longer term, the bottleneck should be broken also by other means. In sections 4-6, it will be shown that this is feasible with quite spectacular effects at the level of the system design methodology. The price paid there will be increased design complexity, which can however be offset with appropriate design methodology support tools. In addition to the main-stream evolution of these SRAMs, DRAMs and SDRAMs, also more customized large capacity high band-width memories are proposed, intended for more specialized purposes. Examples are [**221**]: Video RAMs, very wide DRAMs, and SRAMs with more than 2 ports (see e.g. the 8-port SRAM in [**253**]).

4. CODE REWRITING TECHNIQUES FOR ACCESS LOCALITY

Code rewriting techniques, consisting of loop and data flow transformations, are an essential part of modern optimizing and parallelizing compilers. They are mainly used to enhance the temporal and spatial locality for cache performance, and to expose the inherent parallelism of the algorithm to the outer (for asynchronous parallelism) or inner (for synchronous parallelism) loop nests [**6**] [**286**] [**16**]. Other application areas are communication-free data allocation techniques [**244**] and optimizing communications in general [**97**].

It is thus no surprise that these code rewriting techniques are also at the heart of our DTSE methodology. As the first step (after the preprocessing and pruning) in the script, they are able to significantly reduce the required amount of storage and transfers. As such however, they only increase the locality and regularity of the code. This enables later steps in the script (notably the data reuse, memory (hierarchy) assignment and in-place mapping steps) to arrive at the desired reduction of storage and transfers.

Crucial in our methodology is that these transformations have to be applied globally, *i.e.* with the entire algorithm as scope. This is in contrast with most existing loop transformation research, where the scope is limited to one procedure or even one loop nest. This can enhance the locality (and parallelization possibilities) within that loop nest, but it does not change the global data flow and associated buffer space needed between the loop nests or procedures. In this section, we will also illustrate our preprocessing and pruning step, which is essential to apply global transformations.

In section 4.1, we will first give a very simple example to show how loop transformations can significantly reduce the data storage and transfer requirements of an algorithm. Next, we will demonstrate our approach by applying it to a cavity detection application for medical imaging. This application is introduced in section 4.2, and the code rewriting techniques are applied in section 4.3. Finally (section 4.4) we will also give a brief overview of how we want to perform global loop transformations automatically in the DTSE context.

4.1. Simple Example. This example consists of two loops: the first loop produces an array `A[]`, and the second loop reads `A[]` to produce an array `B[]`. Only the `B[]` values have to be kept in memory afterwards.

```
for (i=1; i<=N; ++i) {
A[i] = ...;
}
for (i=1; i<=N; ++i) {
B[i] = f(A[i]);
}
```

Should this algorithm be implemented directly, it would result in high storage and bandwidth requirements (assuming that N is large), since all `A[]` signals have to be written to an off-chip background memory in the first loop, and read back in the second loop. Rewriting the code using a loop merging transformation, gives the following:

```
for (i=1; i<=N; ++i) {
A[i] = ...;
B[i] = f(A[i]);
}
```

In this transformed version, the `A[]` signals can be stored in registers since they are immediately consumed after they have been produced, and since they are not needed anymore afterwards. In the overall algorithm, this significantly reduces storage and bandwidth requirements.

4.2. The cavity detection demonstrator. The cavity detection algorithm is a medical image processing application which extracts contours from images to help physicians detect brain tumors. The initial algorithm consists of a number of functions, each of which has an image frame as input and one as output, as shown in Figure 22. In the first function, a horizontal and vertical gauss-blurring step is performed, in which each pixel is replaced by a weighted average of itself and its neighbours. In the second function (`ComputeEdges()`), the difference with all eight neighbours is computed for each pixel, and this pixel is replaced by the maximum of those differences. In the last function (`DetectRoots()`), the image is first reversed. To this end, the maximum value of the image is computed, and each pixel is replaced by the difference between this maximum value and itself. Next, we look for each pixel whether a neighbour pixel is larger than itself. If this is the case, the output pixel is false, otherwise it is true. The complete cavity detection algorithm contains some more functions, but these have been left out here for simplicity. The initial code looks as follows:

```
void GaussBlur (unsigned char image_in[M][N], unsigned char gxy[M][N]) {
unsigned char gx[M][N];
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
gx[y][x] = ... // Apply horizontal gaussblurring
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
gxy[y][x] = ... // Apply vertical gaussblurring
}

void ComputeEdges (unsigned char gxy[M][N], unsigned char ce[M][N]) {
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
ce[y][x] =... // Replace pixel with the maximum difference with its neighbors
}

void Reverse (unsigned char ce[M][N], unsigned ce_rev[M][N]) {
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
maxval = ... // Compute maximum value of the image
// Subtract every pixelvalue from this maximum value
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
ce_rev[y][x] = maxval - ce[y][x];
}

void DetectRoots (unsigned char ce[M][N], unsigned char image_out[M][N]) {
unsigned char ce_rev[M][N];
Reverse (ce, ce_rev);   // Reverse image
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
image_out[y][x] =... // Is true if no neighbors are bigger than current pixel
}

void main () {
```

```
unsigned char image_in[M][N], gxy[M][N], ce[M][N], image_out[M][N];
// ... (read image)
GaussBlur(image_in, gxy);
ComputeEdges(gxy, ce);
DetectRoots (ce, image_out);
}
```

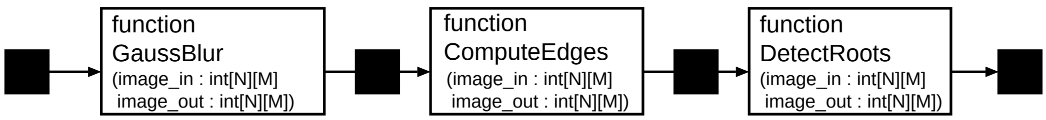


FIG. 22. *Initial cavity detection algorithm*

4.3. Code rewriting for the cavity detection demonstrator. For the initial cavity detection algorithm, as given in Figure 22, the data transfer and storage requirements are very high. The main reason is that each of the functions reads an image from off-chip memory, and writes the result back to this memory. After applying our DTSE methodology, these off-chip memories and transfers will be heavily reduced, resulting in much less off-chip data storage and transfers. Note that all steps will be performed in an application-independent systematic way.

Preprocessing and pruning

First of all, the code is rewritten in a 3-level hierarchy. The top level (level 1) contains system-level functions between which no optimizations are possible. In level 2, all relevant data-dominated computations are combined into one single procedure, which is more easily analyzable than a set of procedures/functions. Level 3 contains all low-level (*e.g.* mathematical) functions, which are not relevant for the data-flow. Thus all further optimizations are applied to the level 2 function. This is a key feature of our approach because it allows to expose the available freedom for the actual exploration steps. The code shown further on is always extracted from this level 2 description.

Next, the data-flow is analyzed and all pointers are substituted by indexed arrays, and the code is transformed into single assignment code such that the flow dependencies become fully explicit. This will allow for more aggressive data-flow and loop transformations. Furthermore, it will also lead to more freedom for our data-reuse and in-place mapping stages later on. This will allow to further compact the data in memory, in a more global and more efficient way than in the initial algorithm code.

Global data flow transformations

In the initial algorithm, there is a function `Reverse()` which computes the maximum value of the whole image. This computation is a real bottleneck for DTSE. From the point of view of computations, it is almost negligible, but from the point of view of transfers, it is crucial, as the whole image has to be written to off-chip memory before this computation, and read back afterwards. However, in this case this computation can be removed by a data-flow transformation.

Indeed, the function `Reverse()` is a direct translation from an original system-level description of the algorithm, where specific functions have been reused. It can be avoided by adapting the next step of the algorithm (`DetectRoots()`) by means of a data flow transformation. Instead of "`image_out[y][x] = if (p>q) ...`" where p,q are pixel elements produced by `Reverse()`, we can write "`image_out[y][x] = if (-p<-q) ...`" or "`image_out[y][x] = if(c-p<c-q) ...`" where `c=maxval` is a constant. So instead of performing the `Reverse()` function and implementing the original `DetectRoots()`, we will omit the `Reverse()` function and implement instead:

```
void cav_detect (unsigned char image_in[M][N], unsigned char image_out[M][N]) {
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
gx[y][x] = ... // Apply horizontal gaussblurring
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
gxy[y][x] = ... // Apply vertical gaussblurring
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
ce[y][x] =... // Replace pixel with the maximum difference with its neighbors
for (y=0; y<M; ++y)
for (x=0; x<N; ++x)
image_out[y][x] =... // Is true if no neighbors are smaller than current pixel
}
```

Next, another data-flow transformation can be performed to reduce the initializations. In the initial version, these are always done for the entire image frame. This is not needed, only the borders have to be initialized, which saves a lot of costly memory accesses. Designers are in principle aware of this, but we have found that in practice the original code usually contains still a large amount of redundant accesses. By systematically analyzing the code for this (which is heavily enabled by the preprocessing phase), we can identify all redundancy in a controlled way.

Global loop transformations

The loop transformations which we apply in our methodology, are relatively conventional as such, but we apply them much more globally (over all relevant loop nests together) than conventionally done, which is crucial to optimize the global data transfers. So the steering guidelines for this clearly differ from the traditional compiler approach. In our example, a global `y`-loop folding and merging transformation is first applied. The resulting computational flow is depicted in Figure 23: it is a line-based pipelining scheme. This is possible since, after the data-flow transformations, all computations in the algorithm are neighborhood computations. The code now looks as follows (after leaving out some conditions on y to reduce the complexity):

```
void cav_detect (unsigned char in_image[M][N], unsigned char out_image[M][N]) {
for (y=0; y<M+3; ++y)
   for (x=0; x<N; ++x)
gx[y][x] =... // Apply horizontal gaussblurring
for (x=0; x<N; ++x)
gxy[y-1][x] =... // Apply vertical gaussblurring
for (x=0; x<N; ++x)
ce[y-2][x] =... // Replace pixel with max. difference with its neighbors
for (x=0; x<N; ++x)
image_out[y-3][x] =... // Is true if no neighbors are smaller than this pixel
}
```

A global `x`-loop folding and merging transformation is applied too. This further increases the locality of the code, and thus the possibilities for data reuse. As a result, the computations are now performed according to a fine-grain (pixel-based) pipelining scheme (see Figure 24). The code now looks as follows (also here, conditions on `x` and `y` have been left out from the code):

```
void cav_detect (unsigned char in_image[M][N], unsigned char out_image[M][N]) {
for (y=0; y<M+3; ++y)
for (x=0; x<N+2; ++x) {
```

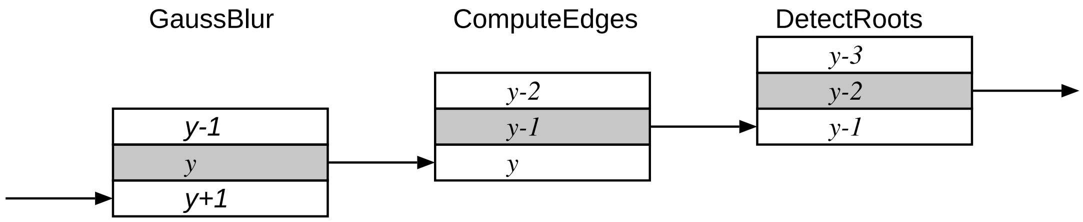


FIG. 23. *Cavity detection algorithm after y-loop transformation*

```
gx[y][x] =... // Apply horizontal gaussblurring
gxy[y-1][x] = ... // Apply vertical gaussblurring
ce[y-2][x-1] = ... // Replace pixel with max. difference with its neighbors
image_out[y-3][x-2] = ... // Is true if no neighbors are smaller than this pixel
    }
}
```

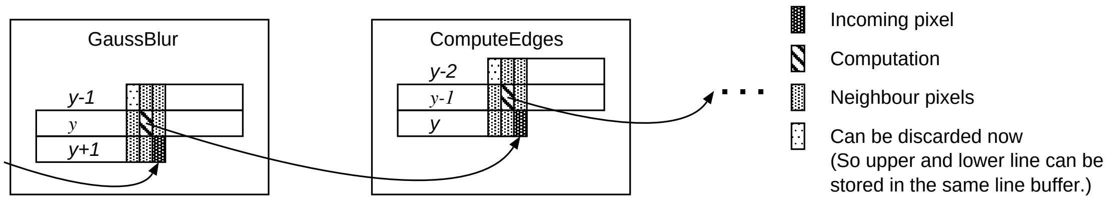


FIG. 24. *Cavity detection algorithm after x-loop transformation*

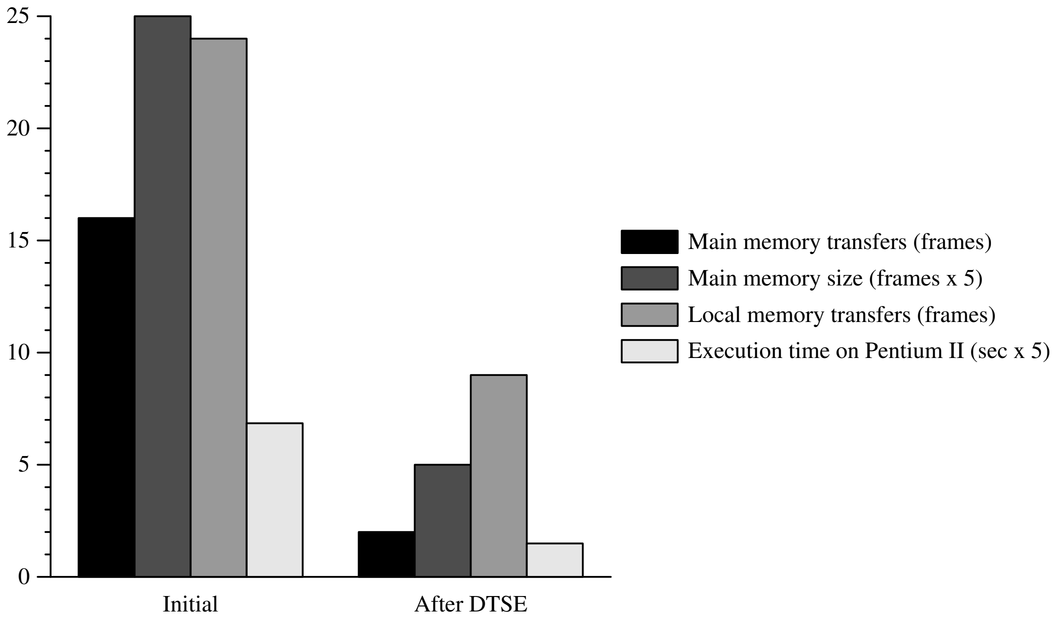


FIG. 25. *Results for cavity detection application*

The result of applying these transformations is a greatly improved locality, which will be exploited by the data reuse and in-place mapping steps to reduce the storage and bandwidth requirements of the application. It is clear that only three line buffers per function are needed for the intermediate memory between the different steps of the algorithm (as opposed to the initial version, which needed frame memories between the functions). In-place mapping can further reduce this to two line buffers per function.

The final results for the cavity detection application are given in Figure 25. The figure shows that the required main memory size has been reduced with a factor of 8. This is especially important for embedded applications, where the chip area is an important cost factor. Moreover, the accesses to both main memory and local memory have been heavily reduced, which will result in a significantly lower power consumption. Because of the increased locality, also the number of cache misses (*e.g.* on a Pentium II processor) is much lower, resulting in a performance speedup with a factor of 4.

4.4. A methodology for automating the loop transformations. To automate the loop transformations, we make use of a methodology based on the polytope model [**147**] [**265**] . In this model, each n-level loop nest is represented geometrically by an n-dimensional polytope. An example is given at the top of Figure 26, where *e.g.* the loop nest with label A is two-dimensional and has a triangular polytope representation, because the inner loop bound is dependent on the value of the outer loop index. The arrows in the figure represent the data dependencies; they are drawn in the direction of the data flow. The order in which the iterations are executed can be represented by an ordering vector which traverses the polytope.

To perform global loop transformations, we have developed a two-phase approach. In the first phase, all polytopes are placed in one common iteration space. During this phase, the polytopes are merely considered as geometrical objects, without execution semantics. In the second phase, a global ordering vector is defined in this global iteration space. In Figure 26, an example of this methodology is given. At the top, the initial specification of a simple algorithm is shown; at the bottom left, the polytopes of this algorithm are placed in the common iteration space in an optimal way, and at the bottom right an optimal ordering vector is defined and the corresponding code is derived.

Most existing loop transformation strategies work directly on the code. Moreover, they typically work on single loop nests, thereby omitting the global transformations which are crucial for storage and transfers. Many of these techniques also consider the body of each loop nest as one union [**59**], whereas we have a polytope for each statement, which allows more aggressive transformations. An exception are "*affine-by-statement*" [**58**] techniques which transform each statement seperately, but our two-phase approach still allows a more global view on the data transfer and storage issues.

5. TASK VERSUS DATA-PARALLELISM EXPLOITATION

Parallelization is a standard way to improve the performance of a system when a single processor cannot do the job. However, it is well known that this is not obvious, since many possibilities exist to parallelize a system, which can severely differ in performance. That is also true for the impact on data storage and transfers.

Most of the research effort in this area addresses the problem of parallelization and processor partitioning [**6**] [**172**] [**218**]. These approaches do not take into account the background storage related cost when applied on data dominated applications. Only speed is optimized and not the power or memory size. The data communication between processors is usually taken into account in most recent methods [**1**] but they use an abstract model (i.e. a virtual processor grid, which has no relation with the final number of processors and memories). A first approach for more global memory optimization in a parallel processor context was described in [**55**], where we showed that an extensive loop reorganization has to be applied before the parallelization steps. A methodology to combine parallelization with code rewriting for DTSE was presented in [**56**].

In this section, we will focus on the parallelization itself. Two main alternatives are usually task and data parallelism. In task parallelism, the different subsystems of an application are assigned to

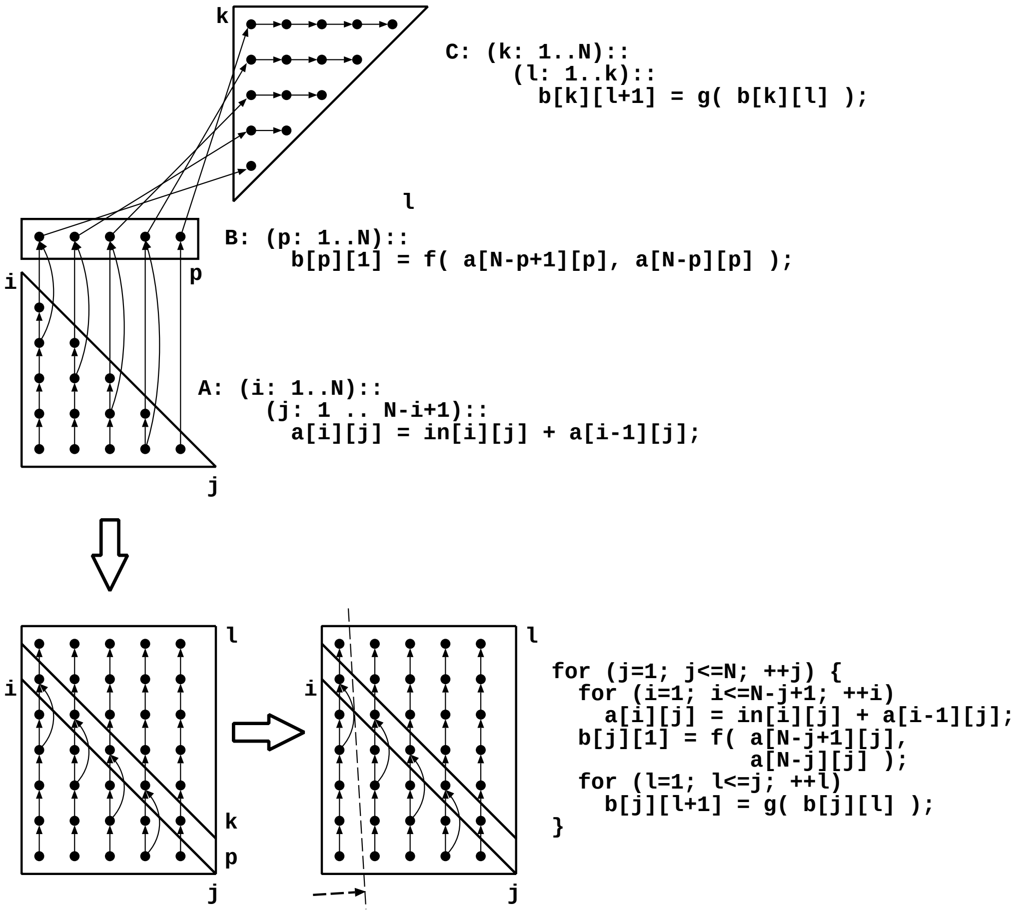


FIG. 26. *Example of automatable loop transformation methodology*

different processors. In data parallelism, each processor executes the whole algorithm, but only to a part of the data. Also hybrid task-data parallel alternatives are possible though. When data transfer and storage optimization is an issue, even more attention has to be paid to the way in which the algorithm is parallelized. In this section, we will illustrate this on two examples, namely the cavity detection algorithm and an algorithm for video compression (QSDPCM).

5.1. Illustration on cavity detection application. We will look at some ways to parallelize the cavity detection algorithm, assuming that a speedup of about 3 is required. In practice most image processing algorithms do not require large speedups so this is more realistic than massive speedup. In the following paragraphs, two versions (initial and globally optimized) of the algorithm are parallelized in two ways: task and data parallelism.

Initial algorithm

Applying task parallelization to the initial algorithm leads to a coarse grain pipelining solution (at the level of the image frames). This can work well for load balancing on a three-processor system, but it is clearly an unacceptable method if we have efficient memory management in mind.

Data parallelism is a better choice. Neglecting some border effects, the cavity detection algorithm lends itself very well to this kind of parallelism, as there are only neighbour-to-neighbour dependencies. So each processor can work more or less independently from the others, except at the boundaries

Version + parallelism	Frame mem	Frame transfers	Line buffers
Initial, data	2	30	0
Initial, task	6	30	0
Transformed, data	0	0	6 RAM
Transformed, task	0	0	6 FIFO

TAB. 2. *Results for cavity detection application*

where some idle synchronization and transfer cycles will occur. Each processor will still need two frame buffers, but now these buffers are only 1/3rd of a frame. So the data parallel solution will in fact require the same amount of buffers as the mono-processor case.

Globally optimized algorithm

Applying task parallelization consists of assigning each of the steps of the algorithm to a different processor, but now we arrive at a fine-grain pipelining solution. Processor one has a buffer of two lines ($y-1$ and y). Line $y+1$ enters the processor as a scalar stream; synchronously the GaussBlur step can be performed on line y, the result of which can be sent to the second processor as a scalar stream. This one can concurrently (and synchronously) apply the `ComputeEdges` step to line $y-1$ and so on. In this way we only need a buffer of two lines per processor, or 6 in total! This is the same amount as we needed for the mono-processor case. So we have achieved what we were looking for: improved performance without sacrificing storage and transfer overhead (which would translate in area and power overhead). Since the line buffer accesses are in FIFO order, cheap FIFO buffers can be selected to implement them.

When we use data parallelism with the globally optimized version, we will need 6 line buffers per processor, or 18 in total. The th of the buffers will depend on the way in which we partition the image. If we use a row-wise partitioning (e.g. the first processor processes the upper third of the image,...), the 18 buffers are of the same size as in the mono-processor case. A column-wise partitioning yields better results: we still need 18 buffers, but their th is only a third of a line (thus equivalent to 6 line buffers of full th). Because the accesses are not FIFO compatible in this case, the buffers will have to be organized as SRAMs, which are more expensive than FIFOs.

The results are summarized in Table 2. It is clear that the task-parallel version is the optimal solution here. Note that the load balancing is less ideal than for the data-parallel version, but it is important to trade off performance and DTSE. E.g., if we can avoid a buffer of 32 Kbit by using an extra processor, this can be advantageous even if this processor is idle 90we have a very bad load balance), because the cost of this extra processor in terms of area and power is less than the cost of a 32 Kbit on-chip memory.

5.2. QSDPCM video processing application. The QSDPCM (Quadtree Structured Difference Pulse Code Modulation) technique is an interframe compression technique for video images. It involves a motion estimation step, and a quadtree based encoding of the motion compensated frame-to-frame difference signal. A global view of the algorithm is given in Figure 27. We will not explain the functionality of the different subsystems in detail here.

The partitioning of the QSDPCM application is based on the computational load, *i.e.* on the number of operations to be executed on each processor. To keep the load balanced, approximately the same amount of computation must be assigned to each of the processors. The total number of operations for the coding of one frame is 12988 K. We will assign about 1000 K operations to each processor, therefore we need 13 processors. The total size of the array signals in the application is 1011 K words. The number of accesses to these signals is 9800 K per frame. The array signals can be divided in two categories:

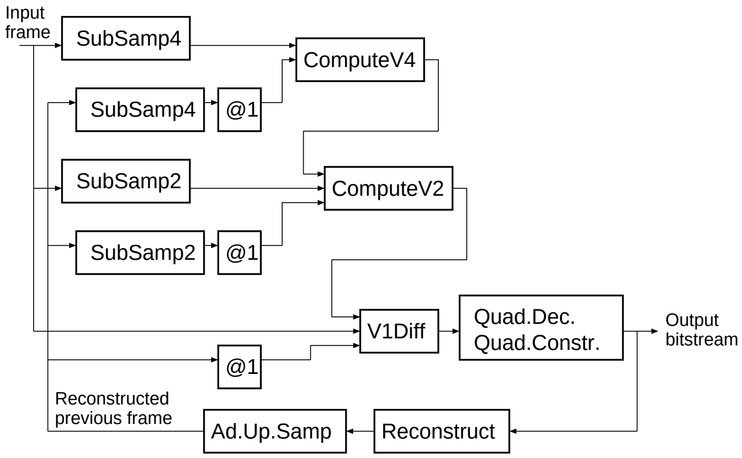


FIG. 27. *QSDPCM application*

- **Category A (532 K):** those which are either inputs or outputs of the algorithm on a per frame basis (coding of one frame).
- **Category B (479 K):** those which are intermediate results during the processing of one frame.

The category A signals can be reduced to 206 K by aggressive in-place mapping, but they are always needed, independent of the paritioning. Therefore, our comparisons further on will consider the category B signals.

We will compare (*w.r.t.* DTSE) different parallelization alternatives for this application: a pure data level partitioning, a modified task level partitioning, a pure task level partitioning, and two hybrid versions (all of these have been obtained by hand). Except for the first partitioning, all alternatives have been evaluated for an initial and a globally transformed version of the code.

Pure data level partitioning

In data level partitioning, each of the 13 processors will perform the whole algorithm on its own part of the current frame (so each processor will work on approximately 46 blocks of 8x8 pixels). The basic advantage of this approach is that it is simple to program. It also requires reduced communication between the processors. On the other hand, each processor has to run the full code for all the tasks.

The memory space required for this partitioning is 541 K words (42 K per processor), while the number of accesses for the processing of one frame is 12860 K. The main reason for these huge memory space requirements is that each processor operates on a significant part of the incoming frame and thus buffers are required to store the intermediate results between two sub-modules of the QSDPCM algorithm. Because of their large size (541 K) these buffers cannot be stored on chip. The increase of the number of accesses (from 9900K in the non-parallel solution, to 12860K) is due to the overlapping motion estimation regions of the blocks in the boundaries of neighbouring frame areas.

Modified task level partitioning

In task level partitioning, the different functions (or parts of them) of the QSDPCM algorithm are assigned in parts of about 1 million operations over the 13 processors. This partitioning is more complex than data level partitioning and for this reason requires more design time. It also requires increased communication (through double buffers) between processors. However, each processor runs only a small part of the code of the QSDPCM application. Here, a modified task level solution is discussed, *i.e.* one which is in fact not a pure task level partitioning as it already includes some data parallelism. A pure task level partitioning will be discussed in the next subsection.

Figure 28 shows the modified task level partitioning. The required memory space for the category B signals is 287 K words, which is significantly smaller than for the data level partitioning (541 K). The main reason for this is that all the buffers storing the intermediate signals are only present between two processors. In data level partitioning these buffers are present in each processor. The number of memory accesses required for the processing of one frame in the task level case (13640 K) is increased in comparison to the data level case (12860 K). This is a result of the increased communication between processors (through double buffers).

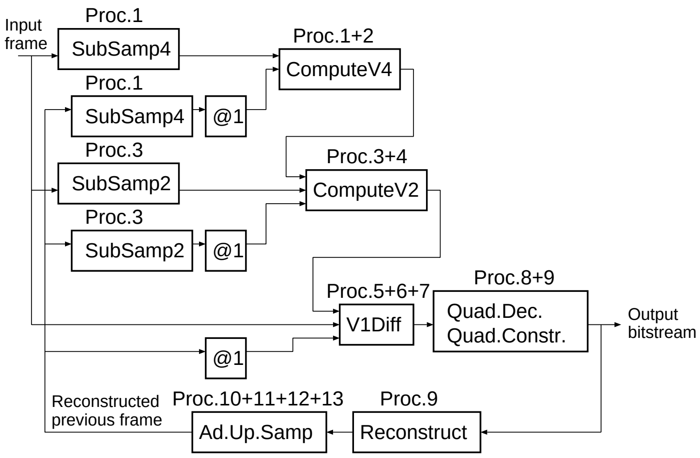


FIG. 28. *Modified task level partitioning*

Pure task level partitioning

There are two points in the previous partitioning in which data level partitioning is performed for specific groups of processors (see Figure 28). Using task level partitioning at these points too, requires 210 K words for the storage of the difference blocks, as well as 2737 K accesses to these words. This is an overhead in comparison to the previous solution. Similar conclusions hold for processors 10, 11, 12, 13. Thus it is clear that a pure task level partitioning of the QSDPCM algorithm is very inefficient in terms of area and power.

Modified task level partitioning based on a reorganized description

Now the modified task level partitioning is performed again, but after applying extensive loop transformations in the initial description (as described in the previous section). The aim of the loop reorganization is to reduce the memory required to store the intermediate arrays (signals of category B) and the number of off-chip memory accesses, since on-chip storage of these signals will become possible after the size reduction. As a result of the loop transformations, the intermediate array signals require only 1190 words to be stored. This means that on-chip storage is indeed possible.

Pure task level partitioning based on the reorganized description

The pure task level partitioning has been analyzed for the loop reorganized description too. The result is that it imposes an overhead of 1024 words and 2737 K accesses to these words. So it is clear that the overhead is significant.

Hybrid partitioning 1, based on the initial description

This hybrid task-data level partitioning is based on a combination of the pure task and data level partitionings. The functions of the QSDPCM application are divided into groups which are executed by different groups of processors. However, within each group of processors each processor performs all the functions of the group to a different part of the data. The proposed hybrid partitioning is described in Figure 29. As far as the intermediate array signals are concerned, the memory size required for their storage (245 K) is smaller than the size required by the modified task level partitioning (287 K). In the same way the number of accesses to these signals is reduced.

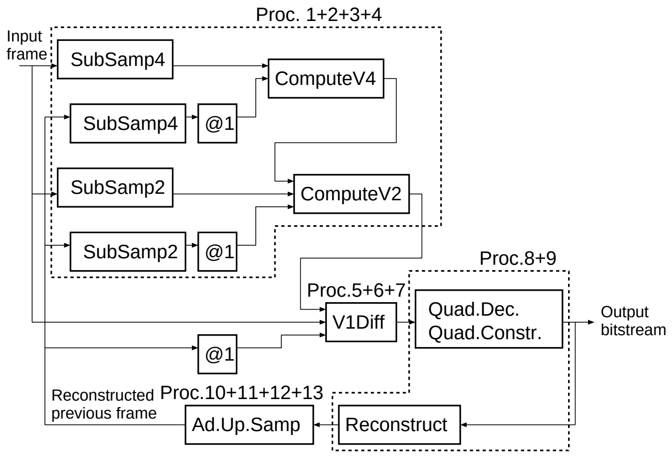


FIG. 29. *Hybrid partitioning 1*

Hybrid partitioning 1, based on the loop-reorganized description

For this version, the memory size required for the storage of the intermediate array signals is 1872 words. This memory size is higher in comparison to the corresponding size required by the task level partitioning (1190 words). Although some buffers between procedures `SubSamp4` and `V4` as well as between `SubSamp2` and `V2` were eliminated, the gain was offset by replicates of array signals with the same functionality in all the processors performing the same task on different data.

Hybrid partitioning 2, based on the initial description

The second hybrid partitioning alternative is oriented more towards the modified task level partitioning. The only difference is in the assignment of tasks to processors 1 and 2. The memory size required by the intermediate array signals is now 282 K words and is slightly reduced in comparison to the task level partitioning. The number of accesses to these signals is also reduced by 10 K in comparison to the task level partitioning as a result of the reduced communication between processors 1 and 2. This second hybrid partitioning requires more words for the storage of the intermediate array signals and more accesses to these signals in comparison to the first hybrid partitioning.

Hybrid partitioning 2, based on the loop-reorganized description

In this case, the number of words required to store the intermediate signals (1334 words) is slightly larger than for the corresponding task level partitioning (1190 words). However this memory size is now smaller than in the first hybrid partitioning. The number of memory accesses to these signals is smaller than in task level partitioning but larger than in the first hybrid partitioning.

Version	Partitioning	Area	Power
Initial	Pure data	1	1
	Pure task	0.92	1.33
	Modified task	0.53	0.64
	Hybrid 1	0.45	0.51
	Hybrid 2	0.52	0.63
Reorganized	Pure task	0.0041	0.0080
	Modified task	0.0022	0.0040
	Hybrid 1	0.0030	0.0050
	Hybrid 2	0.0024	0.0045

TAB. 3. *Results for QSDPCM*

5.3. Conclusions. As far as the memory size required for the storage of the intermediate array signals is concerned, the results of the partitionings based on the initial description prove that this size is reduced when the partitioning becomes more data oriented. This size is smaller for the first hybrid partitioning (245 K) which is more data oriented than the second hybrid partitioning (282 K) and the task level partitioning (287 K).

For the reorganized description, the results indicate the opposite. In terms of the number of memory accesses to the intermediate signals, the situation is simpler: this number always decreases as the partitioning becomes more data oriented.

Table 3 shows an overview of the achieved results. The estimated area and power figures were obtained using a model of Motorola (this model is proprietary so we can only give relative values). From this table it is clear that the rankings for the different alternatives (initial and reorganized) are clearly distinct. For the reorganized description, the task level oriented hybrids are better. This is true because this kind of partitioning keeps the balance between double buffers (present in task level partitioning) and replicates of array signals with the same functionality in different processors (present in data level partitioning). However, we believe that the optimal partitioning depends highly on the number of submodules of the application and on the number of processors.

6. DATA LAYOUT REORGANIZATION FOR REDUCED CACHE MISSES

In section 2, we introduced the three types of cache misses and identified that conflict misses are one of the major hurdles to achieving better cache utilization. In the past source-level program transformations to modify execution order to enhance cache utilization by improving the data locality have been proposed [**119**] [**143**] [**138**]. Storage order optimizations are also helpful in reducing cache misses [**201**] [**63**]. However existing approaches do not eliminate majority of conflict misses. Besides [**119**] [**201**] very little has been done to measure the impact of data layout optimization on the cache performance. Thus advanced data layout techniques need to be identified to eliminate conflict misses and improve the cache performance.

In this section we discuss the memory data layout organization (MDO) technique. This technique allows an application designer to remove most of the conflict cache misses. Apart from this, MDO also helps in reducing the required bandwidth between different levels of memory hierarchy due to increased spatial locality.

First, we will briefly introduce the basic principle behind memory data layout organization with an example illustration. This is followed by the problem formulation and a brief discussion of the solution to this problem. Experimental results using a source-to-source compiler for performing data layout optimization and related discussions are presented to conclude this section.

6.1. Example Illustration. Figure 30 illustrates the basic principle behind MDO. The initial algorithm needs three arrays to execute the complete program. Note that the initial memory data layout is single contiguous irrespective of the array and cache sizes. The initial algorithm can have 3N (cross-) conflict cache misses for a direct mapped cache, in the worst case *i.e.* when each of the arrays are placed at an (base) address, which is a multiple of the cache size. Thus to eliminate all the conflict cache misses it is necessary that none of the three arrays get mapped to the same cache locations, in this example.

The MDO optimized algorithm will have no conflict cache misses. This is because, in the MDO optimized algorithm the arrays always get mapped to fixed non-overlapping locations in the cache. This happens because of the way the data is stored in the main memory, as shown in Figure 30. To obtain this modified data layout, the following steps are carried out : (1) The initial arrays are split into sub-arrays of equal size. The size of each sub-array is called as tile-size. (2) Merge different arrays so that the sum of their tile-size's equals cache size. Now store the merged array(s) recursively till all the arrays concerned are mapped completely in the main memory. Thus we now have a new array which comprises all the arrays but the constituent arrays are stored in such a way that they get mapped into cache so as to remove conflict misses and increase spatial locality. This new array is represented by "x[]" in Figure 30.

In Figure 30, two important observations need to be made: (1) there is a recursive allocation of different array data, with each recursion equal to the cache size and (2) the generated addressing, which is used to impose the modified data layout on the linker.

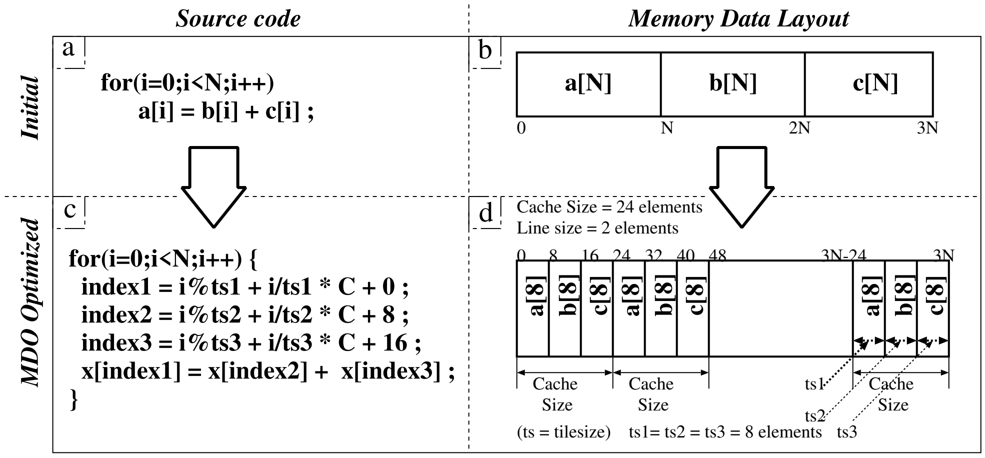


FIG. 30. *Example illustration of MDO optimization on a simple case. Note the source code generated and the modified data layout*

6.2. The General Problem. In this section, we will first present a complete problem definition and then discuss the potential solutions. The general memory data layout organization problem for efficient cache utilization (DOECU) can be stated as, "For a given program with *m*-loop nests and *n*-variables (arrays), obtain a data layout which has the least possible conflict cache misses". This problem has two sub problems. First, the tile size evaluation problem and secondly the array merging/clustering problem.

6.2.1. *Tile Size Evaluation Problem.* Let X_i be the tile size of the array i and C be the cache size. For a given program we need to solve the m equations as below to obtain the needed (optimal) tile sizes. This is required because of two reasons. Firstly, an array can have different effective size in different loop nests. We define effective size as "the number of elements of an array accessed in a loop nest". This number can thus represent either the complete array size or a partial size and is represented as $effsize$. The second reason being that different loop nests have different number of arrays which are simultaneously alive.

$$\text{(9)} \qquad \begin{cases} L_1 &= & x_1^0 + x_2^0 + x_3 + \ldots + x_n^0 & \leq C \\ L_2 &= & x_1^1 + x_2^1 + x_3^1 + \ldots + x_n^1 & \leq C \\ \ldots & & & \\ L_m &= & x_1^{(m-1)} + x_2^{(m-1)} + x_3^{(m-1)} + \ldots + x_n^{(m-1)} & \leq C \end{cases}$$

The above equations need to be solved so as to minimize the number of conflict misses. The conflict misses can be estimated using techniques such as cache miss equations [**91**]. In this paper, we assume that all the arrays which are simultaneously alive have an equal probability to conflict (in the cache). The optimal solution to this problem comprises solving ILP problem [**145**] [**173**], which requires large CPU time. Hence we have developed heuristics which provide good results in a reasonable CPU time. A more detailed discussion of this topic can be found in [**135**] [**137**].

6.2.2. *Array Merging/Clustering Problem.* We now further formulate the general problem using the loop weights. The weight in this context is the probability of conflict misses calculated based on the simultaneous existence of arrays for a particular loop-nest *i.e.* sum of effective sizes of all the arrays as given below :

$$\text{(10)} \qquad L_{wk} = \sum_{i=1}^{n} effsize_i$$

Hence, now the problem to be solved is, which variables to be clustered or merged and in what order *i.e.* from which loop-nest onwards so as minimize the cost function. Note that we have to formulate the array merging problem this way because, we have many tile sizes for each array [1] and there are different number of arrays alive in different loop nest. Thus, using above loop weights we can identify loop nests which can potentially have more conflict misses and focus on clustering arrays in these loop nests.

6.3. The Pragmatic Solution. We now discuss some pragmatic solutions for the above problem. These solutions comprise heuristics, which are less complex and faster from the point of view of automation. First, we briefly discuss how the two stages of the problem are solved as below :

(1) The first step involves evaluation of the effective sizes for each array instances in the program. Next, we perform a proportionate allocation based on the effective size of every array in every loop nest. This means that arrays with larger effective sizes get larger tile sizes and vice-versa. Thus the remaining problem is the merging of different arrays.
(2) The second step involves the merging/clustering of different arrays with their tile sizes. To achieve this we first arrange all the loop nests (in our internal model), in ascending order of their loop weights as calculated earlier. Next, we start merging arrays from the loop nest with highest loop weight and go on till the last remaining array has been merged. Note that once the total tile size is equal to the cache size, we start a second cluster and so on. This is done in a relatively greedy way, since we do not explore for the best possible solution extensively.

We have automated two heuristics in a prototype tool, which is a source-to-source (C-to-C) precompiler step. The basic principle of these two heuristics are given below :

1. In the worst case, one tile size for every loop nest in which the array is alive.

	Initial	DOECU-I	DOECU-II
Avg Mem Acc Time	0.48218	0.2031	0.187943
L1 Cache Line Reuse	423.2192	481.1721	471.0985
L2 Cache Line Reuse	4.960771	16.65545	23.19886
L1 Cache Hit Rate	0.997643	0.997926	0.997882
L2 Cache Hit Rate	0.832236	0.94336	0.958676
L1-L2 BW (Mb/s)	13.58003	4.828789	4.697513
L2-Mem BW (Mb/s)	8.781437	1.017692	0.776886
L1-L2 Data trfs (Mb)	6.94	4.02	3.7
L2-Mem Data trfs (Mb)	4.48	0.84	0.61

TAB. 4. *Experimental results on cavity detection algorithm using MIPS R10000 processor.*

(1) **DOECU I:** In the first heuristic, the tile size is evaluated individually for each loop nest which means that the proportionate allocation is performed based on the effective sizes of each array in the particular loop nest itself. Thus we have many alternatives[2] for choosing the tile size for an array. In the next step, we start merging the arrays from the loop nest with the highest weight, as calculated earlier, and move to the loop nest with the next highest weight and so on till all the arrays are merged. In summary, we evaluate the tile sizes locally but perform the merging globally based on loop weights.

(2) **DOECU II:** In the second heuristic, the tile sizes are evaluated by a more global method. Here we first accumulate the effective sizes for every array over all the loop nests. Next we perform the proportionate allocation for every loop nest based on the accumulated effective sizes. This results in fewer difference between tile size evaluated for an array in one loop nest compared to the one in another loop nest. This is necessary since sub-optimal tile sizes can result in larger self conflict misses. The merging of different arrays is done in a similar was as in the first heuristic.

6.4. Experimental Results. This section presents the experimental results of applying MDO, using the prototype DOECU tool, on three different real-life test-vehicles namely a cavity detection algorithm used in medical imaging, a voice-coder algorithm which is widely used in speech processing and a motion estimation algorithm used commonly in video processing applications. Cavity detection algorithm is explained in section 4.2. We will not explain the other algorithms here but it is important to note that they are data dominated and comprise 2-10 pages of C code.

The initial C source code is transformed using the prototype DOECU tool, which also generates back the transformed C code. These two C codes, initial and MDO optimized, are then compiled and executed on the Origin 2000 machine and the performance monitoring tool "perfex" is used to read the hardware counters on the MIPS R10000 processor.

Table 4, 5 and 6 give the obtained results for the different measures for all the three applications. Note that table 6.3 has same result for both the heuristics since the motion estimation algorithm has only one (large) loop nest with a depth of six namely six nested loops with one body

The main observations from all the three tables are: MDO optimized code has a larger spatial reuse of data both in the L1 and L2 cache. This increase in spatial reuse is due to the recursive allocation of simultaneously alive data for a particular cache size. This is observed from the L1 and L2 cache line reuse values. The L1 and L2 cache hit rates are consistently greater too, which indicates that the tile size evaluated by the tool were nearly optimal since for sub-optimal tile sizes there will more self conflict cache misses.

2. We could have in the worst case, a different tile size for every array in every loop nest for the given program.

	Initial	DOECU-I	DOECU-II
Avg Mem Acc Time	0.458275	0.293109	0.244632
L1 Cache Line Reuse	37.30549	72.85424	50.88378
L2 Cache Line Reuse	48.51464	253.4508	564.5843
L1 Cache Hit Rate	0.973894	0.98646	0.980726
L2 Cache Hit Rate	0.979804	0.99607	0.998232
L1-L2 BW (Mb/s)	115.4314	43.47385	49.82194
L2-Mem BW (Mb/s)	10.13004	0.707163	0.31599
L1-L2 Data trfs (Mb)	17.03	10.18	9.77
L2-Mem Data trfs (Mb)	1.52	0.16	0.06

TAB. 5. *Experimental results on voice coder algorithm using MIPS R10000 processor.*

	Initial	DOECU-I/II
Avg Mem Acc Time	0.782636	0.28985
L1 Cache Line Reuse	9132.917	13106.61
L2 Cache Line Reuse	13.5	24.22857
L1 Cache Hit Rate	0.999891	0.999924
L1-L2 BW (Mb/s)	0.991855	0.299435
L2-Mem BW (Mb/s)	0.31127	0.113689
L1-L2 Data trfs (Mb)	0.62	0.22
L2-Mem Data trfs (Mb)	0.2	0.08

TAB. 6. *Experimental results on motion estimation algorithm using MIPS R10000 processor.*

Since the spatial reuse of data is increased, the memory access time is reduced by an average factor 2 all the time. Similarly the bandwidth used between L1-L2 cache is reduced by 40% to factor 2.5 and the bandwidth between L2 cache - main memory is reduced by factor 2-20. This indicates that though the initial algorithm had larger hit rates, the hardware was still performing many redundant data transfers between different levels of the memory hierarchy. These redundant transfers are removed by the modified data layout and heavily decreased the system bus loading. This has a large impact on the global system performance, since most multimedia applications are required to operate with peripheral devices connected using the off-chip bus.

Since we generate complex addressing, we also perform address optimizations [**99**] to remove the addressing overhead. Our studies have shown that we are able to not only remove the complete overhead in addressing but gain by upto 20% in the final execution time, on MIPS R10000 and HP PA-8000 processors, compared to the initial algorithm apart from obtaining the large gains in the cache and memory hierarchy.

7. CONCLUSIONS

Storage technology is crucial in more and more systems because of the eternal push for more complex applications with especially larger and more complicated data types. Moreover, the access speed, size and power consumption associated with this storage form a severe bottleneck in these systems (especially in an embedded context). In this chapter, we have discussed several building blocks for memory storage, with the emphasis on internal architectural organization for both caches and main memories. The main emphasis has been on modern multimedia and telecom oriented processors, both of the microprocessor and DSP type. In addition, the way date are mapped is as crucial. So we have also discussed data transfer and storage optimizing source code transformations, task versus data parallelism exploitation and memory data layout organisation issues. Several realistic applications have demonstrated the impressive effects of such techniques.

CHAPITRE 3

Low-Power Software for System-on-a-Chips (SOCs)

This chapter is the final version of our accepted special session on System-on-a-Chips (SoCs) proposal at the 4th ACM/IEEE Design and Test in Europe Conf. (DATE), Munich, Germany, March 2001 [**190**].

Organiser: Christian Piguet[1]
Presenters: Christian Piguet[1]*, Marc Renaudin*[2] *and Thierry J-F. Omnès*[3]

[1]*CSEM, Rue Jaquet-Droz 1, P.O. Box* **CH**-*2007 Neuchâtel*
[2]*Laboratoire TIMA, INPG, 46 Avenue Félix Viallet,* **F**-*38031 Grenoble Cédex*
[3]*Interuniversity Micro-Electronics Centre (IMEC), Kapeldreef 75,* **B**-*3001 Leuven*

1. Low-power issues for SOCs

1.1. Introduction. According to the SIA 1999-2000 Roadmap, the move toward embedded Systems on Chip is quite clear. Systems on Chip (SoCs) consist to integrate several components on the same chip in order to improve performances and reduce the cost. Few years ago, a system was a multichip device containing a microprocessor, several memory chips, DRAM chips, FPGA chips, RF (BiCMOS) or analog chips. More and more, all these components will be integrated on a single chip, including microprocessors, glue logic, memories, ROM and SRAM, conventional analog blocks, DRAM memories, FPGA-like logic, RF receiver or transmitter and finally MEMS. The total number of transistors on a single chip could be over one billion (predicted [**242**] to be between 4 to 19 billions in 2014 depending in the circuit type).

For SoCs, very important design problems have to be solved. They are mainly the silicon complexity (reliability, power, interconnect), the system complexity (logic, MPU, memories, RF, FPGA, RF, MEMS), the design procedures (300 to 800 people for the design of a single chip, I.P re-use, design levels), verification and test. Some partial answers have been given to the complexity problem, such as design re-use. The SIA Roadmap [**242**] predicts that in 2012 reuse of processors, logic blocks, and peripherals, would reach about 90% of the embedded logic on the chip.

1.2. Low-power Design Methodologies and CAD tools. Future SoCs will contain several different processor cores on a single chip. It results in parallel architectures, which are known to be less power hungry than fully sequential architectures based on a single processor [**217**]. The design of such architectures has to start with very high-level models in languages such as System C, DL or MATLAB. The very difficult task is then to translate such very high-level models in application software in C and in RTL languages (VHDL, Verilog) to be able to implement the system on several processors.

One could think that many tasks running on many processors require a multitask but centralised operating system (OS), but regarding low-power, it would be better to have tiny OS (2K or 4K instructions) for each processor [**114**], assuming that each processor executes several tasks. Obviously, this solution is easier as each processor is different even if performances could be reduced due to the inactivity of a processor that has nothing to do at a given time frame.

Each more or less specialised processor will be programmed in C and will execute after compilation its own code. Low-power software techniques have to be applied to each piece of software, including pruning, in-lining, loop unrolling and so on. For re-configurable process or cores, retargetable compilers have to be available. The parallel execution of all these tasks has to be synchronised through communication links between processors and peripherals. It results that the co-simulation development tools have to deal with several pieces of software running on different processors and communicating between each other. Such a tool has to provide a high-level power estimation tool to check which are the power hungry processors, memories or peripherals as well as the power hungry software routines or loops.

Such a tool is far from being commercially available. Embedded low-power software emerges as a key design problem. The software content of SoC will increase as well as the cost of its development.

1.3. Low-power SoC Architectures. Low-power SoCs will be based on low-power components, such as processor cores, memories and libraries that are available with nice performances, *i.e.* 20'000 to 100'000 MIPS/watt [**13**] for some cores (using LP techniques such as gated clocks). However, memories are the main consumers on a SoC and several techniques at the architecture and electrical levels have to be applied to reduce their power consumption (generally based on caches, DWL, bitline splitting, low swing).

However, low-power architectures have also to be used. As mentioned earlier, parallel architectures with various specialised cores [**217**] exploit a natural parallelism, which is not based on processor arrays for which software parallelising is too difficult and for which the communication cost is too high. Furthermore, the use of specialised and re-configurable cores could improve performances in such a way that supply voltage could be reduced resulting in lower power consumption. Using a very dedicated co-processor to a given task could improve the speed/power performances of several orders of magnitude.

The 1997 SIA Roadmap [**242**] recognises that, in 2007, asynchronous design will be used in many designs. If ∂ is the distance travelled by a signal in one clock cycle, due to higher frequencies and increasing interconnect delays, a chip will contain several time zones of sizes $\partial * \partial$. Due also to the increasing die size, this number of time zones will grow very rapidly with the new technologies, up to 10'000 zones in 2012. To synchronise 10'000 time zones is a true asynchronous problem [**242**]. It is why asynchronous design is strongly required for chip architectures in the future [**229**].

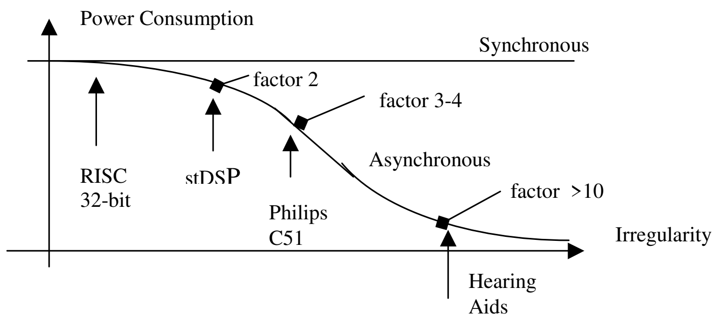


FIG. 1. *Comparison of the Power Consumption of Asynchronous versus Synchronous Architectures*

Asynchronous architectures are often presented as being capable of reducing significantly the power consumption [**229**]. Looking at the results from many papers, it turns out that the largest power savings are obtained with circuits or applications that present a very irregular behaviour. An irregular behaviour is, for instance, a processor that presents a quite tricky instruction set with multi-bytes instructions executed in a various number of cycles and phases. Or an application for which the controller has to very often stop and restart depending on the application. A very regular behaviour is a 32-bit RISC core for which all instructions are always executed in one clock. Figure 1 illustrates this basic law. Basically, SoCs will present more an irregular behaviour than a regular one.

At the architecture level, power and Vdd management with behaviour prediction of the user will be used extensively, as well as low-power communication protocols between the various processors on a single chip. These protocols have to be kept simple and will be asynchronous due to the fact that the various cores will be clocked (if not asynchronous cores) with many different frequencies.

1.4. Low-power Design and Testability Issue. At the low-level, low-power libraries and logic synthesis embedded with place and route are required, as well as estimation of interconnect delays with copper, low-k and SOI. However, the main issue is Vdd as low as 0.6 to 0.3 Volt in 2014. With very deep sub-micron technologies and very low Vdd, the static power will increase significantly due to low Vt. Several techniques with double Vt, source impedance, well polarisation, dynamic regulation of Vt are today under investigation and will be necessarily used in the future.

SoCs testability and debug when first silicon has been returned from the foundry are a main issue. Generally, the mean time to fix a bug is one week. Today, it is not possible do determine if more bugs will be present in a one billion transistor chip, and if a one week per bug is realistic or not. However, it has to be mentioned that it should be much more difficult to fix a bug in I.P. blocks that you do not have designed than current practise. In this new design context, it is estimated that half of the total design effort will be devoted to verification tasks, including debug of the embedded software [**242**].

2. Asynchronous and Locally Synchronous low-power SOCs

2.1. Introduction. SoCs chips will contain with many processors, co-processors, memories and peripherals. As it is quite difficult today to design a clock tree for a single processor (clock skew, power consumption), one can be easily convinced that the situation will be worse in a few years, due to interconnect delays, many processors and different frequencies. It results that future SoCs cannot be synchronised with a single master clock. It is therefore mandatory to propose other clocking schemes including fully asynchronous architectures.

2.2. Clocking schemes. One has fundamentally the choice between two clocking schemes:

– GALS (Globally Asynchronous Locally Synchronous)
– fully asynchronous architectures.

The first choice results from the evolution of synchronous architectures. Basic building blocks are synchronous processors and peripherals, but due to the clock distribution problem, the communications between processors are performed asynchronously. The second choice is a more radical choice that selects asynchronous cores as well as asynchronous communications. From the conceptual point of view, this solution is better as it is a coherent technique and not a mix of synchronous/asynchronous techniques as proposed by the first solution.

2.3. Globally Asynchronous Locally Synchronous SoCs. The high-level model of GALS is many processor cores that can be clocked at any and possibly variable frequency. There can also be asynchronous logic blocks on the same SoC. These cores and blocks have to communicate through asynchronous interfaces. There is no relationship between the frequency of a given block that has to send some messages to any other (or others) block(s). There are two basic solutions to allow the blocks to communicate properly:

– to synchronise their clocks by adjusting the PLL of the two (or more) blocks. This solution proposed in [**100**] consists in N*N PLL for N*N time zones, two adjacent PLL being synchronised by PFDs (Phase Frequency Detectors) in case of a communication.

– to perform an asynchronous communication for which meta-stability has to be avoided. One has to check the data and clock timing to avoid collisions, by introducing a clock delay (stretchable or pausable clock). Several registers clocked by delayed clocks generated locally avoid metastability. Such a technique works well for two communicating blocks, but is not suited for communicating from one to several blocks which could result in a clock-data collision for the chosen delayed clock. It is why delaying data instead of the clock is better [**92**], but it increases hardware.

2.4. Fully Asynchronous SoCs. A more drastic change is to adopt asynchronous logic by default for designing all the components of a SoC and use the "clock" as a resource only when required. Such an approach requires to master asynchronous circuit design, which is not yet an easy task, but on the other hand brings very interesting facilities that the goal of this session is to discuss. It is shown that asynchronous blocks are very easy to reuse and easy to integrate in a complex heterogeneous integrated system because they are locally controlled, they provide flexible and efficient interfacing mechanisms, they have a lower mean power consumption while providing maximum performance, and finally they generate lower electromagnetic noise and smaller currents peaks in the supply [**229**] [**174**]. Significant prototypes designed by industrial and academic teams are reported and discussed to illustrate these properties.

2.5. Power consumption issues in GALS and fully asynchronous SoCs. Comparison:

– GALS: less area than fully asynchronous, due to larger area of fully asynchronous cores
– GALS: more power than fully asynchronous, PLL are large consumers, delayed clocks or data generally require adjustable delays to take into account process variations, temperature,..., but these delays require consuming hardware. Furthermore, for irregular behaviour, fully asynchronous cores consume less than synchronous cores. And SoCs will be mainly designed for applications with irregular behaviour.
– GALA SoCs are by construction consuming minimum energy, their power consumption may be automatically and dynamically adapted to the processing power required by the data stream and they do not require specific hardware/software layers to turn-off inactive processing parts or peripherals.

3. Low-power software for SOCs

For embedded SoCs containing several processors, one has to write several pieces of software for each processor starting typically from a high-level specification using the C/C++ language. Because the cost-critical part of this high-level specification is the data-dominated part [**258**], we have developed a systematic script of code transformations known as the Data Transfer and Storage Exploration (DTSE) methodology for power, area, price and speed multi-objective data-dominated embedded software design [**36**]. The most tedious and error–prone parts of the DTSE methodology are being supported by IMEC's ACROPOLIS/ATOMIUM computer-aided design (CAD) environment [**187**] [**67**]. In this paper, we discuss the main steps of DTSE involved in producing power vs. speed software for each processor of a programmable platform [**39**] consisting of several processors and offering memory and interconnect reconfiguration opportunities [**261**] in the ideal case.

3.1. Data-locality improving Loop Transformations. Loop transformations are at the heart of DTSE. As one of the first steps (after preprocessing and pruning) in the script, they are able to significantly reduce the required amount of storage and transfers. As such however, they only increase the locality and regularity of the code. This enables later steps in the script (notably the data reuse, memory (hierarchy) assignment and in-place mapping steps) to arrive at the desired reduction of storage and transfers.

Crucial in our methodology is that the code transformations have to be applied globally, *i.e.* with the entire algorithm as scope. This is in contrast with most existing loop transformation research, where the scope is limited to one procedure or even one loop nest. This can enhance the locality within that loop nest, but it does not solve the global data flow and associated buffer space needed between the loop nests or procedures. In order to allow these global transformations, the step has to be applied before any partitioning decision is taken *w.r.t.* the HW/SW-system being designed [**57**].

To perform global loop transformations, we make use of a methodology based on the polytope model [**147, 265**]. In this model, each n-level loop nest is represented geometrically by an n-dimensional polytope. The order in which the iterations are executed can be represented by an ordering vector which traverses the polytope. To perform the transformations, we have developed a two-phase approach. In the first phase, all polytopes are placed in one common iteration space. During this phase, the polytopes are merely considered as geometrical objects, without execution semantics. In the second phase, a global ordering vector is defined in this global iteration space.

3.2. Logically Shared but Physically Distributed Data Mapping. After data-locality improving loop transformations have been applied, the cost of a data-dominated Logically Shared but Physically Distributed (LSPD) memory system is determined mostly by:

– The inter-processor bandwidth, which is given by the maximum sum of all ingoing and outgoing dependency edges from each partition;
– The intra-processor bandwidth, which is given by global basic group level conflict analysis at the processor-level [**294**].

The conventional approach used by automatic parallelisation compilers is to partition the iteration space in a Single Program Multiple Data (SPMD) fashion, as it appears after data-locality improving loop transformations. This often results in a high intra-processor bandwidth requirement by the principle of pushing the bottleneck one abstraction-level further down. To alleviate this situation where parallelisation is performed regardless of processor-level constraints and processor-level optimisation is performed only locally we believe that *space–time* analysis of (partly) unrolled loop bodies can help in:

– Reducing the intra-processor bandwidth requirements with limited impact on data locality by re-mapping data accesses onto processors using the full Multiple Instruction Multiple Data (MIMD) capabilities of modern distributed systems [**287**] [**51**];
– Discovering low-cost *space–time* computation *patterns*, that can serve as the basis for (very) low-cost distributed object architectures in platform-based design.

Using our proposed *pattern* analysis, we show (at least) an overall time $\times$ consumption (C $\times$ T) improvement above 100% compared to a custom shared memory approach on a small C/C++ platform-based design example [**188**]. By construction, this technique supports the *interactive* embedded software design [**175**] (even) of large-scale and parallel multimedia and is being integrated within our prototype ACROPOLIS environment.

3.3. Cycle Budget vs. Power trade-offs. In contrast to current design practise for (programmable) processor mapping, which mainly targets performance, we focus on a systematic trade-off between cycle budget and energy consumed in the background memory organisation. The latter is a crucial component in many of today's designs, including multi-media, network protocols and telecom signal processing. We have a systematic way and tool to explore both freedoms and to arrive at Pareto charts, in which for a given application the lowest cost implementation of the memory organisation is plotted against the available cycle budget per sub-module. This by making optimal usage of a parallelised memory architecture. We indicate, with results on a digital audio broadcasting receiver and an image compression demonstrator, how to effectively use the Pareto plot to gain significantly in overall system energy consumption within the global real-time constraints.

The new prototype tool has been applied to drivers from multiple application domains to prove the effectiveness, as demonstrated by results in other recent papers [**26, 30, 27, 268**]. The actual technique underlying the SCBD exploration is discussed in [**28**]. Here we will analyse the results and the detailed evolution of the SCBD tool for the Binary Tree Predictive Coder driver only. The experiment clearly shows the trade-off between memory organisation cost (power and area) and the memory subsystem cycle budget. All the results are obtained within reasonable tool execution time (several minutes) on a Pentium II-400. A Motorola library memory model is used to estimate the on-chip memory cost (see [**38**]). For the off-chip components, we have used an EDO DRAM series of Siemens.

Binary Tree Predictive Coding (BTPC) [**231**] is a loss-less or lossy image compression algorithm based on multi resolution. The image is successively split into a high resolution image and a low resolution quarter image, where the low-resolution image is split up further. The pixels in the high-resolution image are predicted based on patterns in the neighbouring pixels. The remaining error is then expected to achieve high compression ratios with a Huffman coder. The power numbers in this section are based on real memory models.

Figure 2 shows the trade-off for the complete cycle budget range. This is obtained by letting the tool explore the cycle budget starting from the fully sequential budget, and then progress through the most interesting memory organisations (from a cost point of view) to reduce the cycle budget for multiple differently optimised implementations [**27**]. The number of allocated on chip memories is four for the entire graph shown in this figure. In the sequential budget (about 18M cycles), only single-port memories are employed. When a dual-ported memory has to be added, a clear discontinuity is present in the energy function. In order to reduce the budget below 8M cycles, dual-ported memories are needed though. The allocation of two dual-ported memories allows to decrease the cycle budget up to the critical path (6.5M cycles). The worst case needed “image” bandwidth can be guaranteed by inserting three intermediate on-chip memories (two dual port and one single port). These three intermediate memories can deliver up to five pixels per memory cycle without the cost (or infeasibility) of a five-port memory.

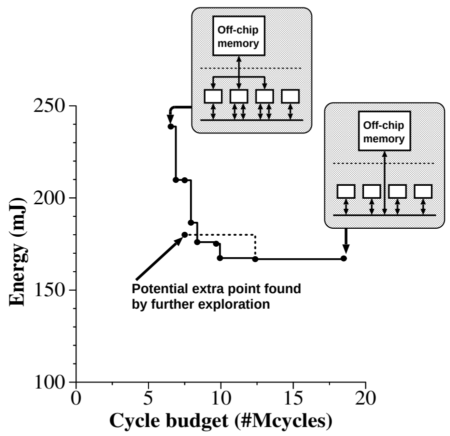


FIG. 2. *Pareto curve for Binary Tree Predictive Coder.*

The memory subsystem cost can be traded of for costs in the other subsystems. The global system cost in terms of power/energy consumption, but also chip/board area or dollars, can be significantly reduced at the price of an acceptable increase in cycles spent in a particular sub-module at the cost of less cycles for another less cost-sensitive sub-module. Exploiting this trade-off is only feasible for real complex systems if it can be systematically explored and when the useful trade-off space is effectively visualised so that a designer can quickly evaluate the cost of different cycle budgets for a given sub-module. We have a methodology supported by tools to effectively achieve this. It has been demonstrated for several other real-life applications (from different domains) like DAB channel decoder, cavity detector and an IP-switch.

3.4. Cache conscious Main Memory Data Layout. Cache misses form a major bottleneck for real-time multimedia applications due to the off-chip accesses to the main memory. This results in both a major access bandwidth overhead and related power consumption as well as performance penalties.

In this section, we illustrate a new technique for organising data in the main memory for data dominated multimedia applications so as to reduce the majority of the conflict cache misses. The main goal of this step is to statically organise data in the main memory taking into account the cache parameters and program characteristics. This involves a combination of array splitting and array interleaving (or array merging) so as to recursively allocate data in the main memory, where each recursive size is equal to the cache size [**137**]. The main advantage of this technique is the ability to trade-off memory size versus power versus performance as compared to other existing techniques. A more detailed discussion of this step is available in [**133**]. This step has been automated as a prototype tool in the ACROPOLIS project at IMEC.

Experiments on real-life demonstrators illustrate that we are able to reduce up to 82% of the conflict misses for applications which are already aggressively transformed at source-level. At the same time, we also reduce the off-chip data accesses by up to 78%. Thus our approach is complimentary to the more conventional way of reducing misses by reorganising the execution order.

3.5. Conclusion. For embedded SoCs containing several processors, designing efficient software for each processor is a (very) difficult and (very) time-consuming task that can be performed manually by a trained designer (only) at the price of a (very) significant degradation of the quality of his final implementation. Embedded software design tools that could help in this process are far from commercially available today. In order to tackle this problem for data-dominated applications, we propose the Data Transfer and Storage Exploration (DTSE) methodology to systematise the organisation of the designer work. In addition, we show that novel CAD support consisting of four main steps can be introduced to automate (almost) all the critical work involved in this multi-objective application-specific platform mapping task **before** applying conventional hardware-software co-design [**32**] [**52**]. This support is currently exactly the purpose of IMEC's ACROPOLIS/ATOMIUM environment [**187**] [**67**].

CHAPITRE 4

A Dynamic Algorithm for Minimizing the Memory Bandwidth of Embedded Systems

This chapter is the extended version of our paper accepted for publication in the special issue of Technique et Science Informatiques (TSI) on Techniques de parallélisation automatique, Éditions Hermès, following the Workshop on Compilation and Automatic Parallelisation, St. Narbor, France, November 18–20, 1999 [**177**].

L'idée d'un ordonnancement guidé par la pression en nombre de ressources (FDS) fut introduite pour la première fois par Paulin et Knight [**212**] pour permettre une synthèse de circuits dédiés à haut débit à un haut niveau d'abstraction. Dans le cadre de nos recherches récentes dans le domaine de l'exploration du transfert et du stockage des données (DTSE) [**38, 290**], nous avons étendu FDS pour l'ordonnancement–placement faible coût en synthèse de systèmes embarqués temps-réels. Dans ce papier, nous reformulons l'itération interne du processus d'optimisation FDS sous forme d'un algorithme de graphe dynamique. Son implémentation permet la conception interactive de systèmes logiciels de télécommunications, de traitement de la parole, de traitement de l'image et de vidéo de faible coût mais (très) haut débit.

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight [**212**] to minimize the number of resources required in the high-level synthesis of high-throughput ASICs. In the frame of our recent Data Transfer and Storage Exploration (DTSE) research [**38, 290**], we have extended FDS for low-cost[1] ordering and mapping in real-time embedded system synthesis. In this paper, we reformulate the inside iteration of the FDS optimization process as a *dynamic* graph algorithm. This new implementation enables the interactive design of low-cost but (very) high-throughput telecom networks, speech, image and video embedded software systems.

1. Context

Many important software applications deal with large amounts of data, which in turn have to be mapped onto a (partly) predefined hardware platform. This is the case both in multi-dimensional signal processing applications such as video and image processing, which handle indexed signals in the context of loops, and in communication network protocols, which handle large sets of records organized in tables. For these multimedia systems, typically a (very) large part of the area cost is due to memory units. The power is also heavily dominated by the storage and transfers. Hence, we believe that the dominating factor in the system-level design is provided by the organization of the global communication and data storage, and related algorithmic transformations. Therefore we have proposed a systematic design methodology in which the memory organization is optimized as a first

1. By low-cost embedded system, we refer to an embedded system exhibiting a good compromise between low-power, low-bandwidth, low-memory size (*i.e.* low-area) and also low-runtime (*i.e.* fast). In our DTSE methodology, this compromise is parameterizable and is a pragmatic choice left to the designer.

step [**273**], **before** the detailed scheduling and mapping onto a (partly) predefined hardware platform is done. Support for this Data Transfer and Storage Exploration (DTSE) methodology is under development in our system-level exploration environment ACROPOLIS [**170, 38, 36**].

Storage bandwidth optimization [**290**] is one of the main steps in our DTSE script. It is executed prior to the Memory Allocation and Assignment (MAA) step [**150, 268**]. The goal of the step is to minimize the required memory bandwidth within the given cycle budget, by adding ordering constraints in the memory access flow-Graph. This allows the memory allocation/assignment tasks to come up with a lower cost memory architecture with a small number of memories and memory ports.

Our previous technique to solve the storage bandwidth optimization step [**290**], namely Conflict Directed Ordering (CDO), has resulted in good quality of results but at the cost of a high computer-aided design runtime for large designs. Moreover, the designer had no control of the quality versus runtime tradeoff. In this paper, we will address these shortcomings and the resulting novel technique will be employed in a more general context than only the storage-bandwidth optimization step.

2. Related work

Scheduling techniques optimizing the number of resources required to meet a deadline have been an active area of research for decades, involving a surprisingly large variety of research communities ranging from operations research for logistics, planning and other military applications; graph theory for high-performance computer-aided design; embedded synthesis and compiler research for more flexible and powerful computer technology to applied artificial intelligence research for next-generation yet more intelligent programming environments. All this research relates to some extent to our Conflict-Directed Ordering (CDO) technique although, as will be developing in the coming sections, our particular storage bandwidth optimization formulation leads to a novel and yet still unsolved constrained scheduling problem.

First, the operations research community aims at seeking highly accurate solutions often at the price of complex modeling and large runtime [**8**]. When computer-aided engineering runtime becomes impractical, using exact solution schemes to a derived subproblem is generally proposed. Within this community, resource-minimal scheduling is typically referred to as a *complex* quadratic-assignment problem [**300**]. Hence, for practical runtime, *only* a quadratic-assignment problem is solved, which in turn optimizes the use of the bandwidth without analyzing *which* data is being accessed simultaneously.

In the embedded synthesis and compiler domain, most techniques operate on the scalar-level [**212, 274**], which in turn lead to impractical runtime for real-life applications. Exceptions currently are the PHIDEO stream scheduler [**274**], the Notre-Dame rotation scheduler [**207**] and DRAM-aware optimizations from the University of California at Irvine [**202**]. Many of these techniques try to reduce the memory related cost by estimating the required number of registers for a given schedule. Only few of them try to reduce the memory bandwidth, which they do by minimizing the *number* of simultaneous accesses [**274**]. They do not take into account *which* data is being accessed simultaneously. Also no real effort is spent to minimize the memory access conflict graphs such that subsequent register/memory allocation can do a better job.

In the artificial intelligence domain, it was recently shown that next-generation programming environments [**49, 240**] incorporate features that can solve problems such as the fast synthesis of digital signal processing systems [**254**]. Such constraint-logic programming environments exhibit powerful global optimization capabilities but, to our knowledge, were never applied for advanced memory bandwidth optimization. We believe this is mainly because a fast and effective problem implementation is required in a first step.

The main difference between our storage bandwidth optimization step and the related work discussed here is that we try to minimize the required memory bandwidth statically in advance by optimizing the access conflict graph for groups of scalars within a given cycle budget. We do this by putting ordering constraints on the flow graph, taking into account *which* memory accesses are being put in parallel (*i.e.* will show up as a conflict in the access conflict graph). Last but not least, our techniques operate on groups of scalars instead of on individual scalars [**202**], which we show in this paper can lead to fast and flexible solutions when combined with state-of-the-art graph heuristics [**122, 110**].

3. Objectives and contributions

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight [**212**] to minimize the number of resources required in the high-level synthesis of high-throughput Application-Specific Integrated Circuits (ASICs). FDS was later chosen as the basis of the Philips Research - PHIDEO silicon compiler, which resulted in an in-depth study of FDS implementation issues in [**274**]. State-of-the-art in FDS reports a runtime complexity of $O(m^2n^2)$, where n is the number of operations and m is the average time window of each operation[2], and a feasibility of solving problems of n equals 100 in one minute using the computers of the mid 90s.

Recently, based on the FDS algorithm, an extension called Conflict-Directed Ordering (CDO) was proposed [**290**] to minimize the required memory bandwidth of embedded multimedia systems. The CDO algorithm involves an important step in the IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration approaches [**170, 38**]. Because the CDO solution process is based on global system-level analysis as opposed to the more local features of FDS, we prove in Section 5.3 that its straightforward implementation exhibits a runtime complexity of $O(n^4m)$. This allows to solve problems for n equals 100 in only ten hours, even using today's workstation series. Moreover, the growing complexity of embedded multimedia systems results in embedded kernels exhibiting a number of memory operations well beyond n equals 100 [**54**] hence necessitating the introduction of instruction-level parallelism to still meet stringent deadlines, which again contributes to growing time windows m. This trend for growing system complexity leads to (very) large-scale scheduling problems, which necessitate novel, fast and effective solution techniques. To improve the designer interaction further [**268**], it is also desirable to have a controllable trade-off between user time and the quality of the solution (which directly influences the cost of the final solution).

In our recent research, we have revisited the original idea of Paulin and Knight [**212**] to show that FDS is a *dynamic* graph algorithm [**175, 74, 110**]. Based on this property, we have introduced a Localized Conflict-Directed Ordering (LCDO) algorithm that instead of performing global system exploration at each optimization step relies on local system explorations, without any loss in the solution quality. This local reduction of the search space allows to reduce the overall complexity by one order of complexity, from $O(n^4m)$ to $O(n^3ln(n)m)$. Moreover, the large size of the remaining LCDO search space allows to also benefit from the latest scheduling and partitioning techniques ([**90, 122, 96**]) for a *k-parameterizable* computer-aided design runtime complexity of $O(kn^2ln(n))$. Experiments of using the original CDO and LCDO on three real-life applications confirm that the design time complexity can be reduced by two orders of magnitude and more. Moreover, an interactive choice of the parameter k is shown to be an appropriate solution for designing large-scale and/or parallel data-dominated systems under stringent real-time constraints. These issues are discussed in more detail in the subsequent sections.

2. For a formal definition of n and m, please consult Section 5.1

4. Illustration and motivation

Imagine that you have at your disposal two 1-port memories to compute $2n$ terms of the following four series[3] in $4n+2$ memory cycles.

$$
\begin{array}{lll}
s_1^{(0)} = 5, & s_1^{(2i-1)} = (s_1^{(2i-2)} - 2) \times 3 & s_1^{(2i)} = (s_1^{(2i-1)} - 1) \times 2 \\
s_2^{(0)} = 5, & s_2^{(2i-1)} = (s_2^{(2i-2)} - 3) \times 4 & s_2^{(2i)} = (s_2^{(2i-1)} - 1) \times 2 \\
s_3^{(0)} = 5, & s_3^{(2i-1)} = (s_3^{(2i-2)} - 3) \times 4 & s_3^{(2i)} = (s_3^{(2i-1)} - 2) \times 3 \\
s_4^{(0)} = 5, & s_4^{(2i-1)} = (s_3^{(2i-2)} - 2) \times 3 & s_4^{(2i)} = (s_4^{(2i-1)} - 1) \times 2
\end{array}
$$

You would probably start by writing the following C code:

```
float S1[2*N+1],S2[2*N+1],S3[2*N+1],S4[2*N+1];
S1[0]=S2[0]=S3[0]=S4[0]=5.;
for (int i=1; i<N+1; i++) {
  S1[2*i-1]=(S1[2*i-2]-2.)*3.; S2[2*i-1]=(S2[2*i-2]-3.)*4.;
  S3[2*i-1]=(S3[2*i-2]-3.)*4.; S4[2*i-1]=(S3[2*i-2]-2.)*3.;
  S1[2*i]=(S1[2*i-1]-1.)*2.;   S2[2*i]=(S2[2*i-1]-1.)*2.;
  S3[2*i]=(S3[2*i-1]-2.)*3.;   S4[2*i]=(S4[2*i-1]-1.)*2.;
};
```

The scheduler might come up with the following solution:

```
1: Load S1[0];  Load S2[0];   2: Load S3[0];  Store S1[1];
3: Store S2[1]; Store S3[0];  4: Store S1[2]; Store S2[2];
5: Store S4[1]; Store S3[2];  6: Store S4[2]; Store S1[3];
7: Store S2[3]; Store S3[3];  8: Store S1[4]; Store S2[4];
9: Store S4[3]; Store S3[4]; 10: Store S4[4]; etc ...
...
2N+1: Store S4[2*N-1]; Store S3[2*N];
2N+2: Store S4[2*N];
```

This ordering does not violate data-flow dependencies, nor does it exceed a memory bandwidth of 2 and nor does it try to place two memory accesses to the same array within the same cycle, hence requiring expensive dual-port memories. However, because of array-to-memory placement constraints, this solution is not feasible using two 1-port memories. Hence, a third 1-port memory has to be added. In order to break this access bottleneck, one needs to find the following schedule:

```
1: Load S1[0];  Load S2[0];  2: Load S3[0];  Store S1[1];
3: Store S1[2]; Store S3[1]; 4: Store S2[1]; Store S4[1];
5: Store S2[2]; Store S4[2]; 6: Store S3[2];  etc ...
...
2N+1: Store S2[2*N]; Store S4[2*N];
2N+2: Store S3[2*N];
```

This solution is feasible using two 1-port memories by carefully assigning `S1` and `S4` to the first memory and `S2` and `S3` to the second memory. In our DTSE script, this post-assignment is the scope of the Memory Allocation and Assignment (MAA) step [**268**], which we do not address in this paper.

As a conclusion, schedulers targeted at embedded software **must** take into account *which* data is being placed in parallel in order to come up with effective solutions. As an illustration, we will

3. Those series can for example arise in financial modeling. s_1 is a strategy consisting in alternating between medium and safe investment. s_2 is a strategy consisting in alternating between risky and safe. s_3 is a strategy consisting in alternating between risky and medium. s_4 is about leaving the most risky strategy s_3 for the safest strategy s_1 in the last round. In the near future, this type of application will be available in embedded decision support systems.

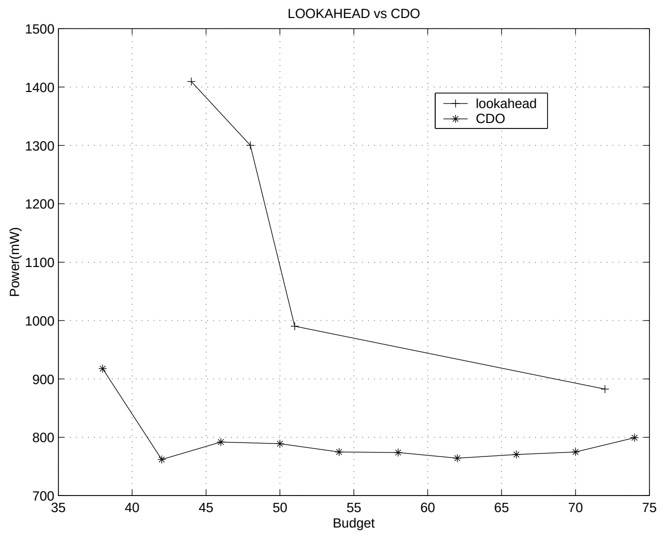


FIG. 1. *List Scheduling with Look-Ahead (LS-LA) vs. our CDO*

now evaluate the well-known List Scheduling (LS) [**20, 117, 31**] algorithm. LS is often cited by both the compiler and system synthesis communities as a reference ordering technique [**270**] Chapter 6. We applied a robust List Scheduling with Look-Ahead (LS-LA) to the design of Alcatel's Segment Protocol Processor (see Section 7.1 for a more detailed description of this system). We measured the power consumption produced by LS-LA and compared it to the power-consumption of our reference, Conflict-Directed Ordering (CDO). The results presented in Figure 1 clearly indicate three trends:

(1) For soft real-time constraints (*i.e.* for a real-time budget higher than 50 memory cycles in our example), LS-LA provides a low-quality solution, already at a 10-20% distance from that of CDO;

(2) For medium real-time constraints (*i.e.* for a real-time budget in the range of 44 to 50), LS-LA provides ineffective solutions, typically exhibiting twice the power consumption of CDO;

(3) For hard real-time constraints (*i.e.* for a real-time budget below 44), LS-LA fails to avoid an extensive use of expensive multi-port memories while CDO can find a smart solution for a real-time budget as low as 38.

As a conclusion, conventional *scheduling-for-speed* algorithms (like LS-LA) are very ineffective for designing embedded systems under hard real-time constraints. In this context, as shown on our SPP example, conventional schedulers (like LS-LA) typically lead to solutions exhibiting twice the necessary power-consumption under real-time constraints relaxed by 10% and more. Moreover, those relaxed constraints are likely to generate an extra factor of 2 increase in cost in practice, by forcing the designer to duplicate the system to still meet his hard real-time requirement, for example using parallel computing techniques. Recent experiments driven by the University of Notre-Dame [**283**] and the University of California at Irvine [**202**] estimate that these shortcomings in the quality of the memory to processor scheduling represent a typical 40% degradation of time performance. In the coming sections, we will therefore develop novel algorithms based on the idea that schedulers **must** take into account *which* data is being placed in parallel.

5. Exploration of access orderings

Our problem is the problem of minimizing the cost of embedded multimedia systems, which can be partly defined as an optimization problem based on a flow-graph, an objective function to minimize and architectural constraints.

Using the example of Section 4., placing `Load S1[0]; Load S2[0];` at memory cycle 1 and then `Load S3[0]; Store S1[1];` at memory cycle 2 and then again `Store S2[1]; Store S3[0];` at memory cycle 3 in parallel requires (at least) a 3-port memory architecture whereas only 2 accesses are being performed in parallel at each memory cycle. This is cost-ineffective by at least 50%.

In the following, we derive ordering techniques to remove this memory bandwidth bottleneck starting from ANSI-C input without any restriction in the input format like for example single-assignment form [**89**], uniform dependences over iterations [**262, 83**], regular-in-time assignment using streams [**150**].

5.1. Problem formulation. Definition 1. *EFG* An extended memory access flow graph $EFG = (V,E,O)$ is a node-weighted directed graph where V is the set of memory accesses, $E \subseteq V \times V$ is the set of precedence constraints, and $O \subseteq V \times B \times T^2$ specifies a time window (or partial ordering) $[t_1,t_2] \in T^2$ for each memory access $v \in V$ to a basic group of scalars in memory $b \in B$.

Remark 1. A basic group of scalars is an indivisible set of data w.r.t its physical mapping onto memories. The choice of the basic groups B is a sensitive choice in the definition of an EFG [**73**].

Terminology 1. *ASAP* and *ALAP* t_1 is referred to as the *As Soon As Possible (ASAP)* timing for v, $ASAP(v)$. t_2 is referred to as the *As Late As Possible (ALAP)* timing for v, $ALAP(v)$. O is referred to as a *partial ordering* of EFG.

Definition 2. *acceptable* Given a memory access flow graph (V,E) to the EFG, an *acceptable* partial ordering O is a partial ordering of EFG meeting all the precedence constraints:

$$\forall e = (v_1,v_2) \in E, \quad t_1(v_2) \geq t_1(v_1) + 1 \quad t_2(v_1) \leq t_2(v_2) - 1$$

Definition 3. *maximal* Given a memory access flow graph (V,E) to the EFG, a *maximal* partial ordering O_{max} is an *acceptable* partial ordering of EFG minimizing $t_1 \in T$ and maximizing $t_2 \in T$ for all memory accesses $v \in V$.

Definition-theorem 4. Unicity of the *maximal* partial ordering Given a memory access flow graph (V,E) to the EFG, for any global timing constraint $T = [s,e]$, there exists a unique *maximal* partial ordering O_{max}.

Corollary to definition-theorem 4. The *maximal* partial ordering of EFG defines the unique maximum possible scheduling freedom for each memory access.

Theorem 1. Necessary inclusion into the *maximal* partial ordering Given a memory access flow graph (V,E) to the EFG, all *acceptable* partial orderings exhibit timing intervals within the range defined by the *maximal* partial ordering.

$$\forall o \in O,[(\forall e = (v_1,v_2) \in E, t_1(v_2) \geq t_1(v_1) + 1\; t_2(v_1) \leq t_2(v_2) - 1)]$$
$$\Rightarrow (\forall (v,b,[t_1,t_2]) \in o, \exists (v,b,[t_{1min},t_{2max}]) \in O_{max}, t_{1min} \leq t_1, t_2 \leq t_{2max})]$$

Terminology 2. *feasible* (or *schedulable*) An extended memory access flow graph EFG is *feasible* (or *schedulable*) if all *maximal* time windows are non empty:

$$\forall (v,b,[t_1,t_2]) \in O_{max}, \quad t_2 \geq t_1$$

and otherwise *infeasible*.

Definition-notation 5. *n* and *m* For any *feasible* extended memory access flow graph EFG, we denote $n = Card(V)$ as the *(total) number of memory accesses* and $m = \sum_{O_{max}} \frac{(t_2 - t_1 + 1)}{n}$ as the *(maximal) average time window* of each memory access.

Definition 6. ***CG(EFG)*** The basic group access conflict graph $CG = (B,C,p)$ of a *feasible* extended memory access flow graph EFG is an edge-weighted undirected graph, where $C \subseteq B \times B$ is the set of basic group access conflicts and p the corresponding probability of conflict. A basic group access conflict between two basic groups of scalars (*i.e.* an edge in the CG) means at least one occurrence of a parallel (or *concurrent*) access to those two basic groups of scalars within the extended memory access flow graph EFG:

$$\forall c = (b_1,b_2) \in C, \exists C(b_1,b_2) = \\ \{((v_1,b_1,[t_{11},t_{12}]),(v_2,b_2,[t_{21},t_{22}])) \in O^2,[t_{11},t_{12}] \cap [t_{21},t_{22}] \neq \emptyset\} \neq \emptyset$$

Terminology 3. ***concurrent*** b_1 and b_2 are referred to as two *concurrent* basic groups of scalars.

Definition 7. ***uniform*** $\boldsymbol{p_u}$ The probability of conflict p between two *concurrent* basic groups of scalars b_1 and b_2 is obtained by accumulating the probability of conflict of all occurrences of a *concurrent* access to b_1 and b_2. The most standard (and *local*) way of computing the individual probability of conflict between two accesses is by assuming that all conflict times in the two conflicting time windows $[t_{11},t_{12}]$ and $[t_{11},t_{12}]$ exhibit a *uniform* probabilistic repartition:

$$\begin{aligned} m(v_1) &= Card([ASAP,ALAP](v_1)) \\ &= t_{12} - t_{11} + 1 \\ m(v_2) &= Card([ASAP,ALAP](v_2)) \\ &= t_{22} - t_{21} + 1 \\ i(v_1,v_2) &= Card([ASAP,ALAP](v_1) \cap [ASAP,ALAP](v_2)) \\ &= \min(t_{12},t_{22}) - \max(t_{12},t_{22}) + 1 \\ p_u(c) &= \sum_{C(b_1,b_2)} \frac{i(v_1,v_2)}{m(v_1) \times m(v_2)} \end{aligned} \tag{11}$$

Remark 2. Non-*uniform* techniques for computing $p(c)$ are discussed in Sections 5.5 onwards.

Pseudo-definition 8. Global (architecture-aware) cost model The basic group access conflict graph (CG) is a compact representation of the **global** memory architecture constraints. Whenever two basic groups of scalars conflict, the memory architecture must guarantee that the conflicting basic groups of scalars can be accessed in parallel in only one memory cycle. In practice, this is solved by multi-port, multiple or banked memory technologies. Based on the conflict graph representation, exploration and accurate cost estimation of memory architectures have been developed [**268**]. It is also possible to approximate this cost by a three-term weighted sum of the number of edges, number of self-edges and chromatic number of the conflict graph [**290**], which represents an implementation challenge in practice [**187**] Section I-2.1.2. In the remaining of this paper, $cost(CG)$ is regarded as a plug-in function call used by our algorithms. The choice of a particular cost function has no major impact on the relative quality or runtime trends presented in the following sections.

Terminology 4. ***cost(CG)*** **or** ***cost(EFG)*** In the following, we will sometimes refer to the above cost model as $cost(CG)$ because it is based on the basic group access conflict graph (CG). For convenience of presentation, we will also refer to it as $cost(EFG)$ since Definition 6 defines a one-to-one match between an EFG and its CG(EFG).

Pseudo-definition 9. Exploration of memory access orderings Given a memory access flow graph (V,E) to the EFG, finding a low-cost memory architecture is the graph optimization problem consisting in identifying the cheapest *acceptable* partial ordering w.r.t the real technological cost of meeting the memory architecture constraints represented in the basic group access conflict graph (CG).

Definition 10. Time interval reduction of EFG Given a memory access $v \in EFG$ such that $O(v) = (b,[t_1,t_2])$ and a time interval $I \subset [t_1,t_2]$ containing (at least) t_1 or t_2, we denote $EFG \backslash t(v)$ the operator consisting in updating the partial ordering O such that I is removed from the time window of v. By definition, after applying the time interval reduction operator $EFG\backslash t(v)$, the new partial ordering of memory access v becomes $O(v) = (b,[t_1,t_2]\backslash I)$.

Theorem 2. Time interval reduction of EFG is strictly inclusive for partial orderings For any time interval reduction of EFG, the new partial ordering is strictly included in the previous one in the sense of the inclusion theorem 1.

$$\forall v \in EFG, \forall t \in T^2(v), \qquad O(EFG\backslash t(v)) \subset O(EFG) \tag{12}$$

Remark 3. Greedy exploration by time interval reductions Algorithms presented in Sections 5.3 onwards are based on a greedy exploration of partial orderings. Starting from $O_0 = O_{max}$, a series of partial orderings are built such that each iteration of the exploration guarantees that the new partial ordering is strictly included in the former one *i.e* $O_{i+1} \subset O_i$ using the time interval reductions of Definition 10. Such algorithms converge in at most $n(m-1)$ and at least 0 iteration(s) while still spanning over the full range of *acceptable* orderings, provided that the EFG is *feasible* (*i.e* that the *maximal* partial ordering is at least *acceptable* and most likely *acceptable* and *reducible*). In the following, we always assume that EFG is *feasible*. In practice, unfortunately, there exists a class of real-time applications that are beyond today's technological capabilities, which of course are *infeasible* and cannot be solved by our techniques alone.

5.2. A 1D digital filter example. In this section, we present our Conflict-Directed Ordering (CDO) technique by the example of power-optimizing the one-dimensional digital filter $p(x) = (\alpha x + \beta\frac{1}{x}) + x^2$. This polynomial filter can be specified as follows using the C language:

```
float S[N]; AX[N]; BX[N]; AXBX[N]; X2[N];
for (int i=0; i<N+1; i++) {
  AX[i]=alpha*S[i]; BX[i]=beta/S[i]; X2[i]=S[i]*S[i];
  AXBX[i]=AX[N-i+1]+BX[N-i+1]; S[i]=AXBX[N-i+1]+X2[N-i+1];
}
```

The above specification is certainly sub-optimal in terms of memory usage and could be improved by blocking the `i`-loop so as to minimize the size of the intermediate `AX`, `BX`, `AXBX` and `X2` arrays. However, in order to also minimize the power consumption, it is desirable to exploit as much as possible the underlying parallelism so as to potentially reduce the clock rate by concurrent memory to processing units data-paths. For this, the computation of `AX` and `BX` have to be done in parallel. The same applies for the computation of `AXBX` and `X2`. Moreover, the computations can be highly pipelined on the i-axis as shown in Figure 2.

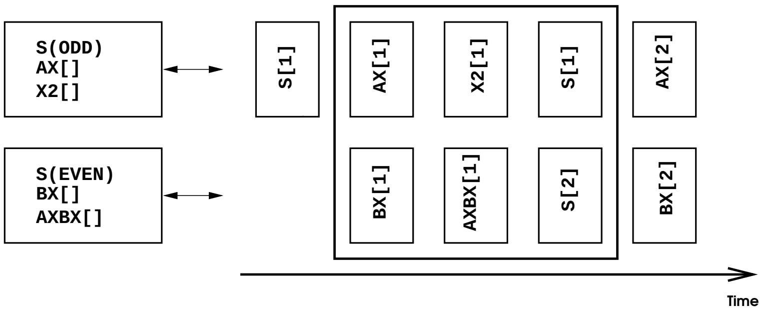


FIG. 2. *Optimal low-cost ordering using two 1-port memories*

The filter throughput is proportional to the inverse of the time required between two consecutive outputs of `S[i]` which corresponds exactly to the time required for one iteration of loop `i`. Using the above optimization, this is achieved in three memory cycles. Suppose we build a perfectly balanced memory-processor architecture in such a way that the processing units can deliver results in one memory cycle (even the division) when used in pipelined arithmetic and burst memory access modes while offering an instruction-level parallelism of at least 2. Based on the proposed loop pipelining scheme, such high-performance computing modes can be easily and fully exploited. As a result, the time required between two consecutive outputs of `S[i]` can be almost (including border effects) as low as three burst-mode memory cycles or three pipelined arithmetic processing cycles (which are equal on a perfectly balanced architecture). Therefore, the filter throughput can reach almost one third of the theoretical pipelined memory to processor throughput, which corresponds to an optimal exploitation of resources for the given application. This last statement remains valid even for a non-perfectly balanced memory-processor architecture offering an instruction-level parallelism of at least 2, which is always the case on modern general-purpose programmable platforms.

For any of the above architectural scenarios, the choice of the blocking factor remains a tradeoff between the pipelined memory to processor characteristics and the memory cost[4]. Last but not least, this tradeoff is a good example of a platform-dependent tuning that is best solved by a designer-in-the-loop use of the scheduler, as enabled by our IMEC - ATOMIUM/ACROPOLIS environment.

5.3. Conflict-directed ordering (CDO). In order to attain this optimal area vs. power solution, the original Conflict Directed Ordering (CDO) solution approach [**290**] works on a flow graph (FG) representation. In a first step, the as soon as possible (ASAP) and as late as possible (ALAP) scheduling interval is being initialized. From the ASAP-ALAP annotated FG, also called extended flow graph (EFG), a conflict graph (CG) incorporating the basic group-level pairwise probability of conflict is being constructed, as shown in Figure 3. After this initialization phase, the set of time reducible

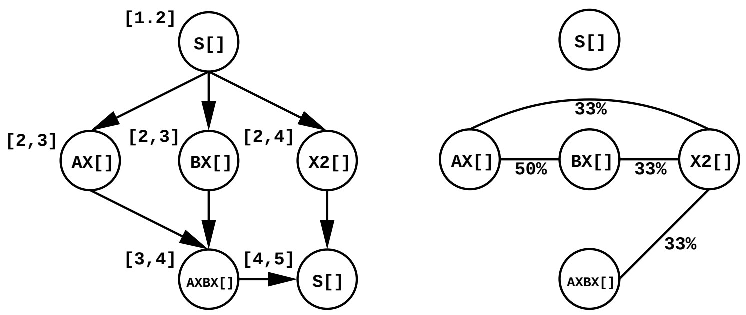


FIG. 3. *Extended flow graph (EFG) and conflict graph (CG) for our 1D filter*

nodes is being updated before entering the iteration process. The pseudo-algorithm presented below shows that each iteration of CDO consists in identifying the most conflicting data access using a single step look-ahead. The [ASAP,ALAP] scheduling interval of the identified data access is then reduced by one time interval.

Because of this unit reduction of the search space and because our 1D digital filter example exhibits a search space of size $n(m-1) = 7$, CDO can in the worst-case reach the optimal solution in seven steps. Moreover, at each step, two look-ahead costs - $cost(EFG \backslash ASAP(v))$ and $cost(EFG \backslash ALAP(v))$ - have to be computed. In the worst-case again, each look-ahead can necessitate to recompute the

4. In pipelined computation theory, runtime performance improves with the number of computed elements *i.e.* in our case the blocking factor. Regarding our low-power objective, decreasing the number of memory cycles allows to reduce the memory clock rate hence the power consumption. In memory optimization theory, performance decreases with the number of intermediate computation arrays *i.e.* in our case the blocking factor. Regarding our low-power objective, decreasing the number of intermediate arrays allows to come up with a smaller and less power-consuming memory architecture. Hence, for power-optimization purposes, there exists an optimal blocking factor.

```
begin
  initialize EFG with the maximal ordering O_max
  reducible := {v ∈ EFG,ASAP(v) < ALAP(v)};                              O(n)
  while (Card(reducible) > 1)                                  O(nm) outer-loops
        cost(EFG\selected) := ∞;
        for (v ∈ reducible) do                      2.||reducible|| inner-loops
            if (cost(EFG\ASAP(v)) < cost(EFG\selected))
               selected := ASAP(v);
            end
            if (cost(EFG\ALAP(v)) < cost(EFG\selected))
               selected := ALAP(v);
            end;
        end;
        EFG := EFG\selected;
        reducible := {v ∈ EFG,ASAP(v) < ALAP(v)};
  end;
end
```

TAB. 1. *Localized Conflict-Directed Ordering (LCDO).*

Conflict Graph (CG) in full using $\frac{n(n+1)}{2}$ pairwise probability of conflict computation (PPCC). Hence, CDO exhibits a worst-case complexity (in PPCC) of $O(n^4m)$.

$$\sum_{i=1}^{n(m-1)} \frac{i}{m-1} \times n(n+1) = \frac{1}{2}n^4m + \ldots \tag{13}$$

For $n(m-1) = 7$, we can estimate a complexity of 1,456 PPCCs.

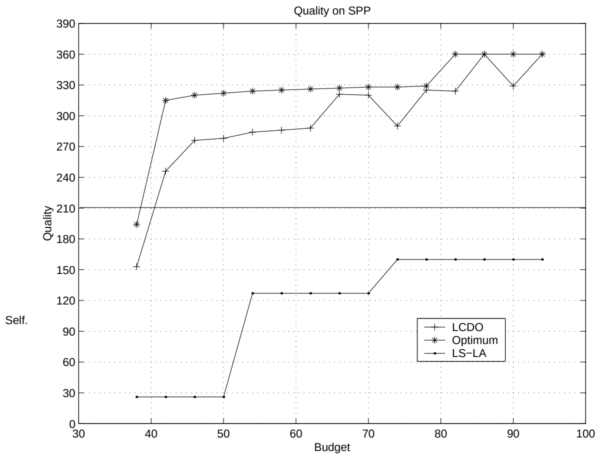


FIG. 4. *LS-LA vs. CDO vs. near-optimal solution quality using our global architecture-aware cost model. Below 210, a self-conflict in the CG is introduced, requiring the use of expensive multi-port memories.*

In Figure 4, we present the results of applying Conflict-Directed Ordering (CDO) to the design of Alcatel's Segment Protocol Processor (SPP). As already introduced in Section 4., it should be first noticed that CDO exhibits a much higher quality of solution w.r.t to DTSE optimizations than conventional ordering techniques like List-Scheduling with Look-Ahead (LS-LA). This motivates the need for low-cost ordering techniques. In a second step, it should be stressed that the initial CDO algorithm is still within a 10-20% margin from the optimal solution. The above combined to an initial high polynomial complexity of $O(n^4m)$ motivates the need for both improvement in quality **and** reduction in design runtime for low-cost ordering techniques like CDO. In the following sections, we will address both needs in detail.

5.4. Localized Conflict Directed Ordering (LCDO). The complexity of CDO can be largely reduced by applying *dynamic* graph techniques [**74, 110**]. Indeed, when comparing $cost(EFG\backslash ALAP(v))$ to
$cost(EFG\backslash selected)$, only a relative cost is required. Hence, it is possible to compare only the difference in cost w.r.t the initial cost(EFG) at the beginning of the current iteration. To do this, only the sons $s(v)$ (respectively only the fathers $f(v)$) of v are involved for an ASAP (respectively ALAP) look-ahead. This involves an update of the set of fathers and sons of the memory access $v \in EFG$, $sf(v) = s(v) \cup f(v)$, as opposed to a full update of the n nodes of EFG in the earlier version. Moreover, only the neighbors of those $sf(v)$ updated nodes have to be considered while updating the CG. This involves an average degree of CG nodes at step i, $d_i \leq n$, updates. Hence, a realistic upper-bound complexity (in PPCC) for a dynamic (or localized) CDO is $O(n^3 ln(n)m)$.

$$\begin{aligned} cplx_1(EFG) &= \sum_{v \in EFG} [ALAP(v) - ASAP(v)] \times sf(v) \\ &\equiv O(nln(n)m) \end{aligned} \tag{14}$$

$$\begin{aligned} & \sum_{i=1}^{n(m-1)} \frac{i}{m-1} \times 2 \times \frac{1}{n(m-1)} \times cplx_1(EFG) \times d_i \\ \leq \; & n^2(1 + \frac{1}{n(m-1)}) \times cplx_1(EFG) \\ \equiv \; & O(n^3 ln(n)m) \end{aligned} \tag{15}$$

When applied to our example, this gives a complexity (in PPCCs) of:

$$6^2(1 + \frac{1}{7})(1.6 + 1.4 + 1.4 + 1.5 + 1.6 + 2.3) = 41{,}14.31 = 1{,}275$$

Therefore, even on this simplistic example, LCDO can be $\frac{1{,}456 - 1{,}275}{1{,}456} = 12\%$ faster than CDO. In practice, it is possible to make LCDO even more dynamic by visiting only those fathers and sons that actually need an update. This data-dependent propagation combined to the strong approximation $d_i \leq n$ means that the above complexity formula (15) is very pessimistic. This also brings an end to all possible dynamic pairwise probability of conflict computation (PPCC) scenarios within one CDO iteration.

In Figure 5, we compare the runtime of Localized CDO (LCDO) to that of CDO on a range of optimization problems arising from the design of Alcatel's Segment Protocol Processor (SPP). On this real-life example, we observe gains in runtime of order 100 which matches well the theoretical predictions. For low values of the real-time budget (*i.e.* for low values of m), real LCDO outperforms the theoretical prediction. This is mainly because the average degree of CG nodes at step i, $d_i \leq n$, decreases much faster in that case. Hence, almost another order of magnitude in the complexity can be gained (i.e. $\equiv O(nln(n)m)$). In simple terms, LCDO is even more dynamic on hard real-time problems.

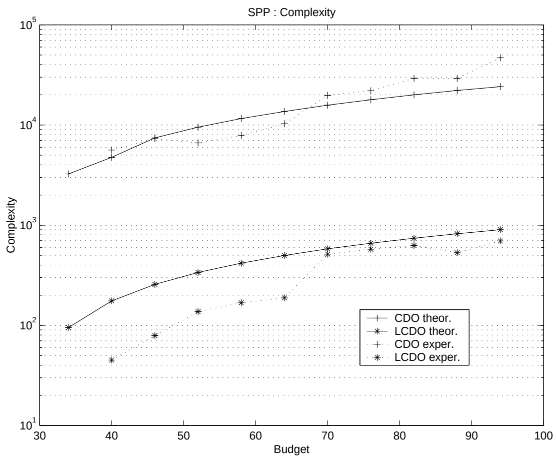


FIG. 5. *Theoretical and real runtime for CDO and LCDO. Time is presented in seconds on the vertical logarithmic scale while the real-time budget $T = [0,B]$ is presented on the horizontal axis.*

Though LCDO can be very fast in practice, it should be stressed that it inherits *exactly* the quality of CDO results. Therefore. solutions are typically found within a 10-20% distance from the optimum. On most (and even simple) cases, CDO and LCDO are unable to find the optimal solution[5]. This motivates the need for higher-quality orderings. Provided *dynamic* techniques can already speed CDO to a reasonable runtime, we will focus on the need for quality improvement in the coming sections. Hopefully, this will inspire further runtime improvements.

5.5. LCDO with global constraint propagation (GCP). When designing large systems, it is common practice to first divide the overall system into p smaller subsystems. When applying automatic design techniques such as LCDO to those subsystems, it becomes much easier to design each subsystem independently in $O(pn^3 ln(n)m)$ than to design and optimize the overall system in full in $O((pn)^3 ln(n)m)$. This divide and conquer mechanism is especially interesting in the case of local optimization techniques, typically found in *scheduling-for-speed* heuristics, because nothing more than just p fully concurrent designs are required in that case.

Unfortunately, as introduced in Section 5.1., optimizing for making systems as cost-effective as possible, requires a **global** architecture-aware cost function. In the case of CDO for example, deciding *which* data can be put in parallel requires (at least) a **global** track of *which* data have already been put in parallel on other parts of the system. Hence, it remains possible to apply divide and conquer techniques to LCDO **only** if **global** constraints are propagated from one subsystem to another. This is for example the case in our data transfer and storage exploration (DTSE) methodology while optimizing over several nested loops [**29**]. Each nested loop is regarded as an individual system requiring an individual scheduling.

We propose to model global constraint propagation by introducing a $\delta(c)$ in the cost function. This δ means that the knowledge of prior decisions applied to other parts of the system guides to either take into account a cost factor ($\delta(c) = 1$) because it is new or to cancel it ($\delta(c) = 0$) because it has already been taken into account previously.

5. For example, CDO does not find the optimal solution for the polynomial filter example presented at the beginning of this section.

$$cost(CG) = \sum_{c=(b_1,b_2)\in CG} \omega(c).p_n(c) \tag{16}$$
$$p_n(c) = \delta(c).p_u(c)$$

By default, all $\delta(c)$ are equal to 1. In a first step, introducing δ in the LCDO algorithm impacts the cost of the individual CG edge cost computation, which remains a $O(1)$ process in PPCCs anyway. In a second step, added computation is also introduced when deciding to keep a conflict, hence necessitating to switch $\delta(c)$ to 0, again in $O(1)$ PPCCs provided the right data structure for the edge cost computation is put into practice[6]. Hence, constraint propagation exhibits the same upper-bound complexity as conventional LCDO. Some overhead can however be expected in practice.

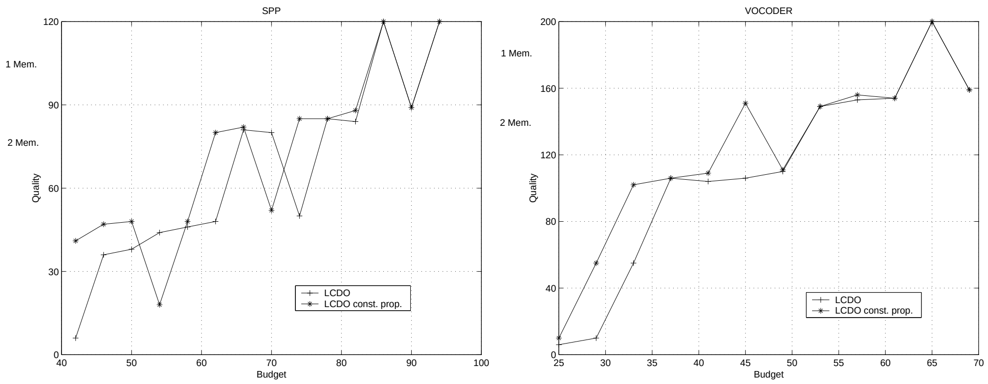


FIG. 6. *Quality for LCDO with CP vs. CDO using our global architecture-aware cost model. For the SPP driver (left), only 1 memory is needed above a quality of 90, 2 concurrent memories above 60 and more below. For the VoCoder driver (right), only 1 memory is needed above a quality of 160, 2 concurrent memories above 120 and more below.*

In Figure 6, we show that global constraint propagation has a very positive impact on the quality of LCDO. Indeed, it really makes sense to re-use the knowledge of problems encountered on other parts of the system to guide new optimizations for new subsystems.

5.6. LCDO with deterministic GCP. It is also possible to implement global constraint propagation on a deterministic conflict graph (CG), that is on a CG where edges are introduced only when a conflict is 100% certain. In that way, new LCDO optimizations leave out only those access nodes that are certain to generate a conflict. By introducing a deterministic CG, the LCDO upper-bound complexity is not affected.

Surprisingly, as shown in Figure 7, a deterministic CG can contribute to further improve the quality of LCDO. Indeed, it really makes sense to leave out only those situations that cannot be solved as opposed to the many that are likely not to be solved. In simpler terms, exploration decisions based on crude facts are better than those based on higher-resolution predictions of future refinements.

6. Faster but still effective exploration

One obvious way to enhance the CDO solution technique is to allow multiple reductions at each iteration. This technique corresponds exactly to that of Kernighan and Lin for fast graph partitioning [**124**]. Multiple reductions are also encountered in improved force-directed scheduling (IFDS) [**274**]. In the following, we introduce LCDO(k), the *clustered* (*i.e.* multiple reductions) version of CDO:

6. A two-level pointing technique to compute $\omega(c).\delta(c)$ "on the fly".

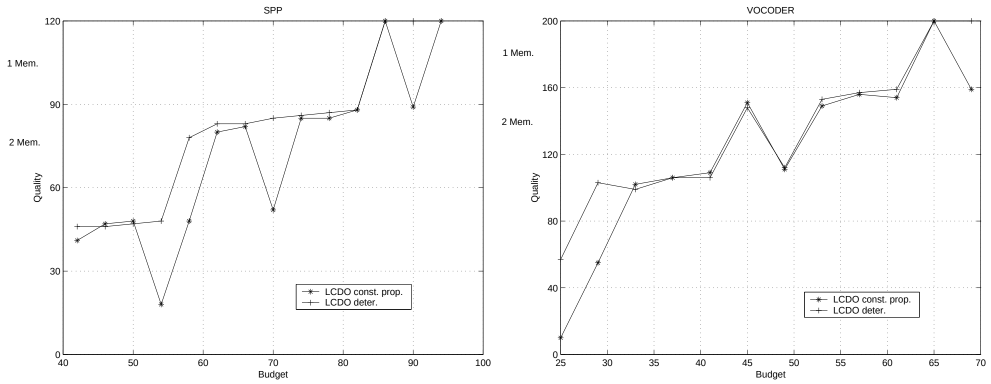


FIG. 7. *Quality for LCDO with deterministic CP vs. CDO-CP using our global architecture-aware cost model. For the SPP driver (left), only 1 memory is needed above a quality of 90, 2 concurrent memories above 60 and more below. For the Vo-Coder driver (right), only 1 memory is needed above a quality of 160, 2 concurrent memories above 120 and more below.*

begin
initialize EFG with the maximal ordering O_{max};
for $(v \in EFG)$ **do** $O(nmln(m))$
$clusters(v) := k$-cluster$([ASAP,ALAP](v),k)$;
$reducible(v) := \{cluster \in clusters(v), ASAP(v), ALAP(v) \in cluster\}$;
end
while $(Card(reducible) > 1)$ $O(k)$ outer-loops
$cost(EFG \backslash selected) := \infty$;
for $(v \in EFG)$ **do** $\|clusters\|$ inner-loops
for $(cluster \in reducible(v))$ **do**
if $(cost(EFG \backslash cluster) < cost(EFG \backslash selected))$
$selected := cluster$;
end;
end;
end
$EFG := EFG \backslash cluster$;
for $(v \in EFG)$ **do**
$reducible(v) := \{cluster \in clusters(v), ASAP(v), ALAP(v) \in cluster\}$;
end
end;
end

TAB. 2. *(Multi-Level) Conflict-Directed Ordering (CDO(k)).*

First, the time domain is decoupled into k clusters with as little inter-cluster conflicts as possible [**105**]. In the general case, this step can be automated by a k-clustering algorithm, for which state-of-the-art favors a k-way partitioning implementation on the complement graph [**105**] pp. 359–360. In the case of CDO(k), this clustering is made very easy because two different times can never conflict. Therefore, the k-clustering has no constraint on the choice of the partitions. In practice, we therefore prefer to choose clusters in the following ad-hoc way [**84**]:

$$
\begin{aligned}
\bigcup_i C_i(v) &= [ASAP,ALAP](v) \\
l(v) &= [0,ALAP-ASAP](v) \\
C_i(v) &= ASAP(v)+[l(v).(1-\alpha_k)^{i-1} \backslash l(v).(1-\alpha)^{i}] \;\; i \in [1,k-1] \\
C_k(v) &= ASAP(v)+l(v).(1-\alpha_k)^{k-1}
\end{aligned} \tag{17}
$$

This ad-hoc choice has the nice property of making the reductions *adaptive* by being less and less aggressive as the optimization progresses if the reduction factor $\alpha_k = \alpha$ is chosen as a constant in [0,1]. We admit here that there exists a direct mapping between the reduction factor α and the number of clusters k. In practice, this ad-hoc clustering technique can be made even more adaptive by alternating the α-reduction from the ASAP and ALAP side dynamically at runtime.

Second, those clusters can serve as input to the clustered version of CDO [**96**], which means that all times within a cluster are always reduced simultaneously.

Therefore, the complexity of LCDO(k) in pairwise probability of conflict computations (PPCC) is given by:

$$
\begin{aligned}
cplx_2(EFG) &= \sum_{v \in EFG} k.sf(v) \\
&\equiv O(knln(n))
\end{aligned} \tag{18}
$$

$$
\begin{aligned}
& \sum_{i=1}^{n(k-1)} \frac{i}{k-1} \times 2 \times \frac{1}{n(k-1)} \times cplx_2(EFG) \times d_i \\
\leq \;& n(1+\frac{1}{(k-1)}) \times cplx_2(EFG) \\
\equiv \;& O(kn^2ln(n))
\end{aligned} \tag{19}
$$

Because the number of clusters k can be controlled by the user, it is possible to reduce the CDO(k) runtime to a minimum.

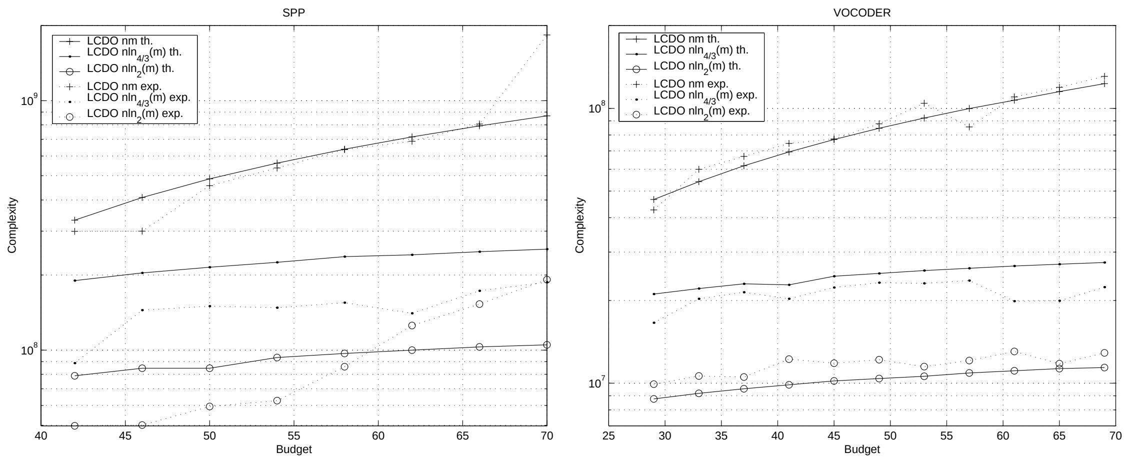


FIG. 8. *Theoretical and real runtime for LCDO and LCDO(k). Time is presented in seconds on the vertical logarithmic scale while the real-time budget $T = [0,B]$ is presented on the horizontal axis.*

In Figure 8, we compare the runtime of various LCDO(k) choices to that of LCDO on a range of optimization problems arising from the design of Alcatel's Segment Protocol Processor (SPP). On this real-life example, we observe gains of order 10-15 which matches well the theoretical predictions. This result added to the gains observed in Figure 5. means an overall speed-up of order 100-1,000 compared to CDO.

For low values of k, real LCDO(k) outperforms the theoretical prediction. This is mainly because the real number of steps can become much lower than $n(k-1)$ when aggressive clustering is used. In that case, the chances of observing a super-linear decline in the number of reducible clusters are much higher because of aggressive multiple reductions.

7. Experiments on real-life examples

7.1. Telecom networks - Segment Protocol Processor (SPP). In this section, we apply LCDO and LCDO(k) to the design of a large-scale system, namely the Segment Protocol Processor (SPP), which is part of the Asynchronous Transfer Mode (ATM) switching mechanism from Alcatel [**257**]. SPP is a Telecom module involving the dynamic routing and buffering of ATM messages by successively decoding parts (also called segments) of its self-contained routing information. After applying our Dynamic Memory Management (DMM) script to the ATM routing standard [**279, 295**], SPP is refined to a specification exhibiting a complex conditional control flow with a total of 156 accesses to 14 arrays. In this context, the role of our storage-bandwidth optimization step is to multiplex the 156 accesses taking into account their exclusive patterns to minimize the required memory bandwidth and cost.

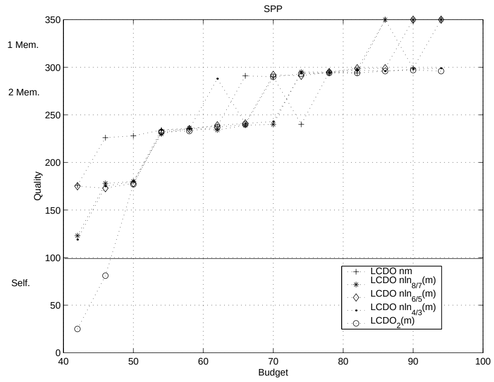


FIG. 9. *LCDO quality on Alcatel's SPP using our global architecture-aware cost model. Only 1 memory is needed above a quality of 300, 2 concurrent memories above 250 and more below. Below 100, the introduction of expensive multi-port memories becomes unavoidable.*

When designing under stringent real-time constraints, much parallelism has to be introduced in the data-path, involving little scheduling freedom (or, in other words, a small search space characterized by a low value of m and a low value of the real-time budget). By applying this stringent scenario to the design of SPP (and as already mentioned in Section 4.), we observe in Figure 9 that LCDO is able to solve problems with a real-time budget lower than 44 memory cycles without introducing self-edges in the Conflict Graph (CG), which would turn up as prohibitively expensive memories. The optimal tradeoff between quality and runtime is met with $k = nln_{6/5}(n)$.

Moreover, the *clustered* reduction capability of LCDO(k) allows to improve the design runtime as illustrated in Figure 10. This improvement factor reaches 300 for a real-time budget of 70, without any loss in the quality of the solution.

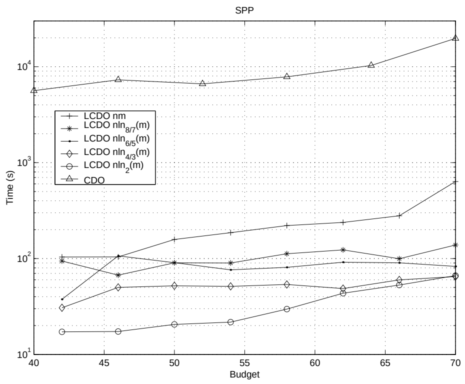


FIG. 10. *Low-cost ordering runtime for Alcatel's SPP. Time is presented in seconds on the vertical logarithmic scale while the real-time budget $T = [0,B]$ is presented on the horizontal axis.*

7.2. Speech processing - Voice Coder (VoCoder). In this section, we apply LCDO and LCDO(k) to the design of a Linear Predictive Coding (LPC) VoCoder. LPC provides an analysis-synthesis approach for speech signals [**132, 271, 226**]. In the LPC algorithm, speech characteristics (energy, transfer function and pitch) are extracted from a frame of 240 consecutive samples of speech. Consecutive frames share 60 overlapping samples. All information to reproduce the speech signal with an acceptable quality is coded in 54 bits. It is worth noting that the steps LPC analysis and Pitch detection use autocorrelation to obtain the best matching predictor values and pitch (using Hamming window). The Schur's algorithm is used to obtain signal in baseband region in LPC analysis and to obtain high and low amplitudes in Pitch analysis. Both autocorrelation and Schur's algorithm have irregular access structures due to algebraic manipulations. Moreover, because of the manipulation of many samples concurrently, the LPC exhibits a highly parallel control flow involving many concurrent arrays. In this context, the role of our storage-bandwidth optimization step is to reduce as much as possible the number of memories by carefully multiplexing the accesses taking into account *which* data is being placed in parallel.

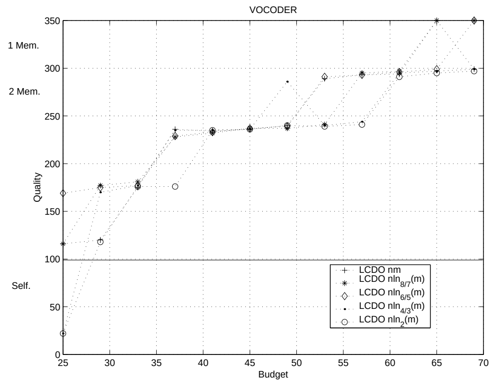


FIG. 11. *LCDO(k) quality for SAIT Electronics's VoCoder using our global architecture-aware cost model. Only 1 memory is needed above a quality of 300, 2 concurrent memories above 250 and more below. Below 100, the introduction of expensive multi-port memories becomes unavoidable.*

In Figure 11, we compare CDO to various LCDO(k) choices on a range of optimization problems arising from the design of SAIT Electronics's VoCoder. When applying stringent real-time constraints, we observe that CDO is able to solve problems with a real-time budget as low as 25.

Moreover, the *clustered* reduction capability of CDO(k) allows to further improve those results. For an extremely low budget of 25, we observe an optimal parameter tuning for $k = nln_{6/5}(m)$. This proves that at least for very difficult problems like VoCoder, the choice of parameter k can help in finding the optimal granularity of the problem formulation w.r.t the solving capabilities of the underlying heuristics.

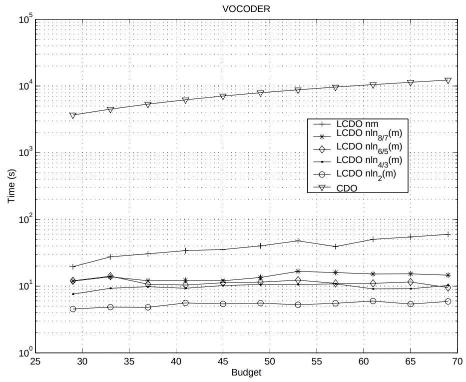


FIG. 12. *Low-cost ordering runtime for SAIT Electronics's VoCoder. Time is presented in seconds on the vertical logarithmic scale while the real-time budget $T = [0,B]$ is presented on the horizontal axis.*

Let us now study the VoCoder design over a wide range of real-time constraints. As shown in Figure 12, our algorithms are clearly classified in two categories from a runtime perspective:

– The conventional category, involving CDO;
– The dynamic category, involving LCDO;
– The fast category, involving LCDO(k).

Those results confirm our complexity analysis presented in Section 5. for LCDO and CDO and Section 6. for LCDO(k). For a real-time budget of 70, LCDO(k) can find a solution 2,000 times faster than a straightforward implementation of CDO (which requires 3 hours and 30 minutes of computation on an HP-PA1000 processor!). This runtime of around 5 seconds for a search space of size 10,000 supports that CDO(k) is, to our knowledge, one of the fastest existing low-cost ordering techniques for large-scale, real-time and parallel programs.

7.3. Image and video processing - Binary Tree Predictive Coding (BTPC). In this section, we apply LCDO and LCDO(k) to the design of Binary Tree Predictive Coding (BTPC), which is used for the compression of images in offset printing. Because BTPC involves a complex algorithm with neighborhood operators at high image rate, it is infeasible to meet stringent real-time constraints without accessing this data in parallel. In this context, the main role of our access ordering technique is to interleave loads, intermediate computations and stores such that only one access to the main image data is required at each cycle, hence avoiding the need for expensive multi-port memories for at least the main memory storage.

In Figure 13, we indicate the circumstances under which LCDO is unable to find the desired low-cost ordering. Below a real-time budget of 25 memory cycles, LCDO is unable to find a good solution without introducing self-edges in the Conflict Graph (CG), which turn up as prohibitively expensive memories in practice. Among the LCDO(k) variants, the best choice is again $k = nlog_{6/5}(n)$.

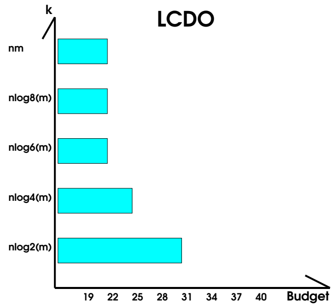


FIG. 13. *Quality results for Binary Tree Predictive Coding (BTPC). For several choices of k on the vertical axis, we present the real-time budget region for which the use of expensive multi-port memories are unavoidable.*

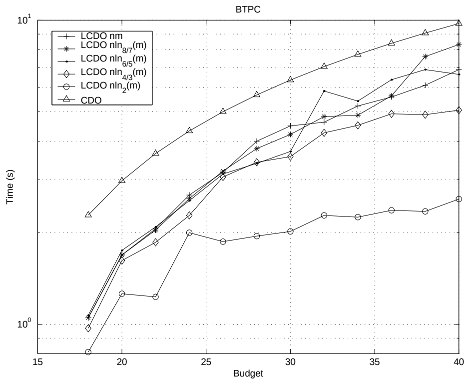


FIG. 14. *Low-cost scheduling runtime for BTPC. Time is presented in seconds on the vertical logarithmic scale while the real-time budget $T = [0,B]$ is presented on the horizontal axis.*

In Figure 14, we measure the runtime of our algorithms for designing the Binary Tree Predictive Coding (BTPC). Even on this small example, LCDO(k) is found to be the fastest solver in about two seconds, improving the CDO runtime by a factor of 4.

8. Conclusion

In this paper, we have introduced three new extensions to the original Force-Directed Scheduling (FDS) algorithm of Paulin and Knight [**212**], mainly for use in the memory access ordering step of our IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration (DTSE) approaches [**170, 38, 36**]. The initial motivation for extending FDS is to take into account *which* memory accesses are being put in parallel. Experiments show the positive impact on the global system cost [**290**]. The second motivation for extending FDS is to introduce *dynamic* graph techniques. Experiments on a wide range of applications, including Telecom networks, speech, image and video processing confirm further benefits, mainly in design runtime. The third motivation for extending FDS is to allow *clustered* analysis. Experiments indicate extra benefits in design runtime and quality for large-scale and/or parallel systems under stringent real-time constraints. By allowing the designer to tradeoff runtime vs. quality through the choice of parameter k, our LCDO(k) algorithm enables the interactive design of low-cost but still data-dominated systems. Using these extensions, experiments indicate spectacular benefits in design runtime and quality for large-scale and/or parallel systems under stringent real-time constraints.

Acknowledgments

We thank Arnout Vandecappelle and Erik Brockmeyer for providing support to the use of their tools and Chidamber Kulkarni for providing support with SAIT Electronics's VoCoder at IMEC. We thank Denis Vanderstraeten (K.U Leuven), Sanjay Rajopadhye (IRISA Rennes) and Alain Darte (E.N.S Lyon) for their input on graph algorithms, problem formulation and reference scheduling techniques. We also wish to thank G. de Jong from Alcatel for use of the Segment Protocol Processor (SPP) application. Our research has been partly supported by the SMT project from the MEDEA-EUREKA programme of the European Commission.

CHAPITRE 5

Interactive Algorithms for Minimizing Memory Bandwidth in High Throughput Telecom and Multimedia

This chapter is the final version of our paper accepted for presentation and publication as a short paper at the 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 5–9, 2000 [**175**].

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight to minimize the number of resources required in the high-level synthesis of high-throughput ASICs [**214**]. In this paper, we reformulate the original FDS algorithm to address the interactive hardware and software design of low-cost but still high-throughput embedded multimedia systems.

1. Context and objectives

Conventional software compilers typically introduce Instruction-Level Parallelism (ILP) in flat Control Data Flow Graphs (CDFGs) by means of List Scheduling (LS) techniques [**214**]. A (very) similar LS approach is typically used for the dynamic runtime scheduling of Real-Time Operating Systems (RTOS) at the task-level. Unfortunately, when applied to the high-level design of embedded systems with finite memory architecture constraints, it is known that such greedy **and** local software optimization techniques shift the makespan (or schedule length) within a factor of 2 from the optimum [**94, 152**] **and** do not utilize resources effectively, which leads to further performance-degrading conflict resolution at runtime [**296**].

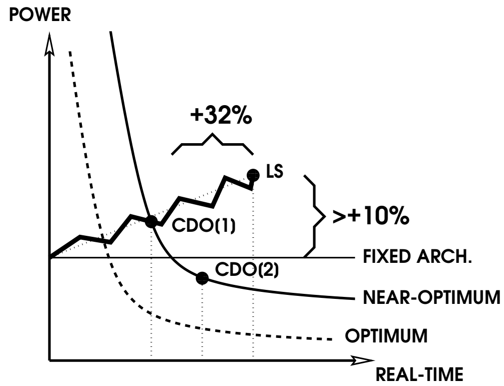


FIG. 1. *Pareto-based iterative refinement of flat CDFG scheduling for improved real-time response and removal of memory resource conflicts*

For (data-dominated) embedded multimedia systems like the dedicated ATM switching Segment Protocol Processor (SPP) from Alcatel [**257**], it was recently estimated [**268, 177**] that the two above shortcomings represent (at least) a cost overhead of 10-100% (in power) for hardware and around 30% (in clock rate) for software. It is however worth mentioning that for hard embedded system designs relying on cutting-edge technology, it is impossible to absorb these overheads. In that case, **only** a near-optimal schedule can deliver results in due-time, which represents a vital competitive advantage to the designer.

In order to address near-optimal scheduling under finite memory architecture constraints, we believe that high-level synthesis techniques like Force-Directed Scheduling (FDS) [**214**] are very adequate because they propose a **global** exploration of the search-space, which is known to be essential for coming closer to the optimal implementation. State-of-the-art in FDS [**274**] reports a runtime complexity of $O(m^2n^2)$, where n is the number of operations and m is the average time window of each operation, and a feasibility of solving problems of n equals 100 in one minute using the computers of the mid 90s.

Recently, based on the FDS algorithm, we proposed an extension called Conflict-Directed Ordering (CDO) [**296**] to address the high-level synthesis of embedded multimedia ASICs. State-of-the art in CDO [**177**] reports a runtime complexity of $O(n^3 ln(n)m)$ using advanced *dynamic* or *localized* graph techniques [**74**].

In the following, we propose to adapt our *localized* CDO (LCDO) technique to also address multimedia software systems using the design process illustrated in Figure 1. In a first step, a conventional scheduler like LS can be run to estimate a reasonable real-time budget. From the thereafter known [lower-bound,initial] real-time budget interval, an iterative optimization scheme can retrieve a near-optimal makespan **and** a conflict-free memory bandwidth utilization scheme. Compared to conventional embedded software design, this enables to utilize a less powerful and more cost-effective processor to address the same specification (procurement support); underload the initial processor to make it more reactive (added value) and/or perform a clock frequency reduction on the initial processor (power reduction). Hence, we believe that our proposed approach can open designers to a range of software design competitive advantage. In this context however, LCDO is run many times [**29**] and must compare to conventional ILP compilers in runtime to attract users. In the following, we therefore develop fast and runtime parameterizable extensions of our LCDO technique [**177**] mainly for extending our IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration (DTSE) approaches [**170, 38**] in a software context.

2. Illustration and contribution

In this section, we apply our technique to the problem of power-optimizing the one-dimensional digital filter $p(x) = (\alpha x + \beta\frac{1}{x}) + x^2$. This filter can be specified using the following C program [1]:

```
for (i=0; j<N+1; i++) {
   AX[i]=alpha*S[i];
   BX[i]=beta/S[i];
   AXBX[i]=AX[i]+BX[i];
   X2[i]=S[i]*S[i];
   S[i]=AXBX[i]+X2[i]; }
```

The above specification is certainly sub-optimal in terms of memory usage and could be improved by blocking the i-loop so as to minimize the size of the intermediate `AX`, `BX`, `AXBX` and `X2` arrays. However, in order to also minimise the power consumption, it is desirable to exploit as much as possible the underlying parallelism so as to potentially reduce the clock rate by concurrent memory to processing units datapaths. For this, the computation of `AX` and `BX` have to be done in parallel. The same applies for the computation of `AXBX` and `X2`. Moreover, the computations can be highly pipelined on the i-axis. Hence, the pixel throughput can reach almost one third of the theoretical

1. For the sake of this presentation, we assume that all arrays have to be stored in main memory. This assumption still leads to an optimal implementation if the values of `AX`, `BX`, `AXBX` and `X2` are re-usable many times by other parts of the system.

pipelined arithmetic throughput while using only two 1-port memories, as illustrated in Figure 2. `AX[i]`, `BX[i]`, `AXBX[i]` and `X2[i]` can be computed in three storage cycles (excluding the `S[i]` read and `S[i]` write, that can also be pipelined in one cycle from one iteration to another). The blocking factor remains a tradeoff between the pipelined arithmetic characteristics and the memory cost.

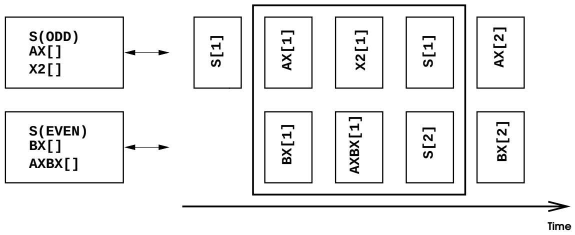


FIG. 2. *Optimal ordering using two 1-port memories*

2.1. Conflict-directed ordering (CDO). In order to attain this optimal area vs. power solution, the original Conflict Directed Ordering (CDO) solution approach [**296**] works on a signal access flow graph (SAFG) representation. In a first step, the as soon as possible (ASAP) and as late as possible (ALAP) scheduling interval is being initialized. From the ASAP-ALAP annotated FG, also called extended flow graph (EFG), a conflict graph (CG) incorporating the basic group-level pairwise probability of conflict is being constructed, as shown in Figure 3. After this initialization phase, the

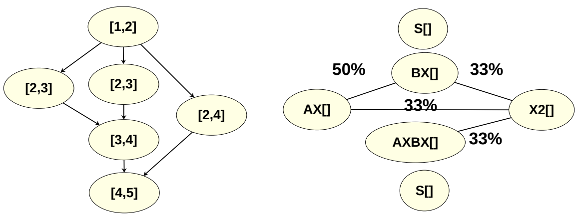


FIG. 3. *Extended flow graph (EFG) and conflict graph (CG)*

set of time reducible nodes is being updated before entering the iteration process. Each iteration of CDO consists in identifying the most conflicting data access using a single step look-ahead. The [ASAP,ALAP] scheduling interval of the identified data access is then reduced by one time interval.

Because of this unit reduction of the search space and because our 1D digital filter example exhibits a search space of size $n(m-1) = 7$, CDO can in the worst-case reach the optimal solution in seven steps.

2.2. Generalized CDO (G-CDO). One obvious way to enhance the CDO solution technique is to allow multiple reductions at each iteration. This technique corresponds exactly to that of Kernighan and Lin for fast graph partitioning [**124**]. Multiple reductions are also encountered in improved force-directed scheduling (IFDS) [**274**]. Unfortunately, because it is always possible to select several nodes in graph partitioning and because IFDS is limited to the selection of a single access node for multiple time reductions, neither of these approaches solve the general problem of multiple selections and reductions (MSMR) in the *multi-dimensional space–time* context of CDO. In order to allow MSMR in full, we propose to introduce a *space–time* conflict graph (ST-CG). As shown in Figure 4, an ST-CG is a two-dimensional graph with accesses on the vertical axis and time on the horizontal axis. The freedom of scheduling access x at time t is (simply) represented by a two-dimensional (x,t) node.

Provided this data model is initialised with a feasible schedule, there are two simple rules for solving the reducibility and feasibility problems:

(1) **Reducibility**: never remove an access from the initial specification i.e. never empty a row;
(2) **Feasibility**: always satisfy dependencies i.e. keep an access only after removing all concurrently scheduled fathers and sons.

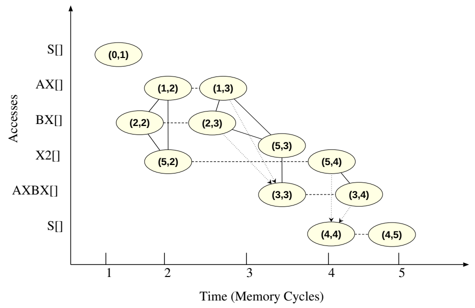


FIG. 4. *ST-CG for our 1D filter*

Based on those two rules, we propose the following generalized conflict directed ordering (G-CDO(k)) algorithm. First, the *space–time* domain is decoupled into k clusters with as little inter-cluster conflicts as possible. In the general case, this step can be automated by a k-clustering algorithm, for which state-of-the-art favours a k-way partitioning implementation on the complement graph. In practice, for pure acceleration purposes, we choose clusters in an ad-hoc way. Second, those clusters can serve as the basis for a *multi-level* version of LCDO[2] or G-CDO[3] [**90, 96**], which means that all nodes within a cluster are always reduced simultaneously.

Because the number of clusters k can be controlled by the user, it is possible to reduce the G-CDO(k) runtime to a minimum. For example, using a greedy coloring technique for k-clustering (after isolating initially non reducible nodes in C1), our 1D digital filter ST-CG results in five clusters:

$$C1 = \{(0,1)\} \tag{20}$$

$$C2 = \{(1,2),(2,3),(5,4),(4,5)\} \tag{21}$$

$$C3 = \{(2,2),(1,3),(4,4)\} \tag{22}$$

$$C4 = \{(5,2),(5,3),(3,4)\} \tag{23}$$

$$C5 = \{(3,3)\} \tag{24}$$

Let us now assume a clever cost function to select and remove $C3$ from the first iteration. For reducibility, $C2$ has to be kept because $(4,5) \in C2$ is now the only node left in the 4th row. For feasibility, $C5$ has to be removed because $(3,3) \in C5$ is a concurrent son of $(2,3) \in C2$. For reducibility, $C4$ has to be kept because $(3,4) \in C4$ is now the only node left in the 3rd row. Hence, there is no reducible cluster left and the algorithm stops. Surprisingly enough, this aggressive use of G-CDO(k) still finds the optimal solution and we will see that this property will also hold in real-life multimedia.

3. Segment Protocol Processor

In this section, we apply the original CDO, LCDO(k) and G-CDO(k) to the design of a large-scale system, namely the Segment Protocol Processor (SPP), which is part of the Asynchronous Transfer Mode (ATM) switching mechanism from Alcatel [**257**]. SPP is a telecom module involving the dynamic routing and buffering of ATM messages by successively decoding parts (also called segments)

2. denoted LCDO(k).
3. denoted G-CDO(k).

of its self-contained routing information. After applying our Dynamic Memory Management (DMM) script to the ATM routing standard, SPP is refined to a specification exhibiting a complex conditional control flow with a total of 156 accesses to 14 arrays. In this context, the role of our access ordering step is to multiplex the 156 accesses taking into account their exclusive patterns to minimize the required memory bandwidth and cost.

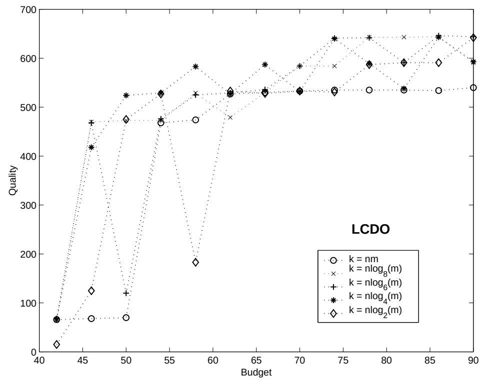


FIG. 5. *LCDO(k) quality on SPP*

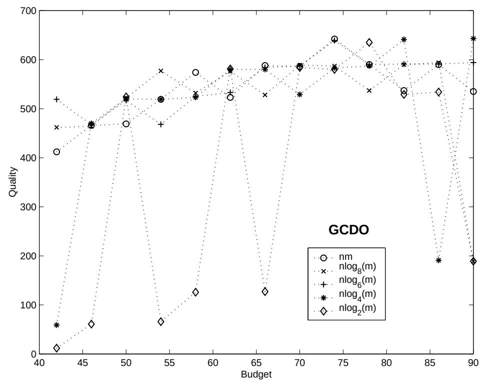


FIG. 6. *G-CDO(k) quality on SPP*

When designing under stringent real-time constraints, much parallelism has to be introduced in the datapath, involving little scheduling freedom (or, in other words, a small search space characterized by a low value of m and a low value of the real-time budget). By applying this stringent scenario to the design of SPP, we observe in Figure 5 that CDO(k) is unable to solve problems with a real-time budget lower than 54 memory cycles while G-CDO (k) can solve problems for a budget as low as 46. This proves that the *multi-dimensional* selection process of G-CDO can achieve better results than its *mono-dimensional* counterpart implemented in LCDO or the original CDO.

Moreover, the *multi-level* reduction capability of G-CDO(k) allows to further improve those results. For an extremely low budget of 42, we observe an optimal parameter tuning for $k = nlog_6(n)$. This proves that at least for large-scale problems like SPP, the choice of parameter k can help in finding the optimal granularity of the problem formulation w.r.t the solving capabilities of the underlying heuristics.

For a real-time budget of 104, G-CDO(k) can find a low-cost scheduling 1500 times faster than a straightforward implementation of CDO (which requires more than 35 hours of computation on an

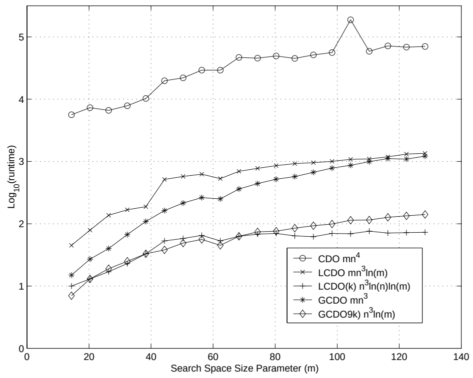


FIG. 7. *CDO, LCDO(k) and G-CDO(k) runtime on SPP*

HP-PA1000 processor!). This runtime of around ten seconds for low budgets and one minute for a search space of size $mn > 15{,}000$ supports that G-CDO(k) is, to our knowledge, the fastest existing low-cost ordering technique for large-scale parallel programs in O(knm) while remaining robust for k chosen as O($nln(m)$).

4. Binary Tree Predictive Coding

In this section, we apply CDO, LCDO(k) and G-CDO(k) to the design of a Binary Tree Predictive Coding (BTPC), which is used for the compression of images in offset printing. Because BTPC involves a complex algorithm with neighbourood operators at high image rate, it is infeasible to meet stringent real-time constraints without accessing this data in parallel. In this context, the main role of our access ordering technique is to interleave loads, intermediate computations and stores such that only one access to the main image data is required at each cycle, hence avoiding the need for expensive multi-port memories for at least the main memory storage.

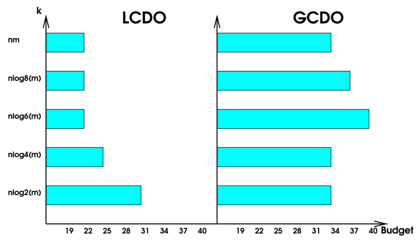


FIG. 8. *LCDO(k) and G-CDO(k) quality on BTPC*

In Figure 8, we indicate the circumstances under which LCDO and G-CDO are unable to find the desired low-cost ordering. Below a real-time budget of 25 memory cycles, none of the algorithms can find a good solution and it is likely that a good solution does not exist. From a budget of 25 to 37, LCDO clearly exhibits better results than the current version of G-CDO(k). This is because the nature of our problem is clearly *mono-dimensional*, involving the need to order carefully only one dataset. Among the LCDO(k) variants, the best choice is $k = nlog_6(m)$.

In Figure 9, we measure the runtime of our algorithms for designing the Binary Tree Predictive Coding (BTPC). LCDO(k) is found to be the fastest solver in about two seconds whereas G-CDO(k) pays a *multi-dimensional* overhead of ten seconds. This difference may be important in practice when

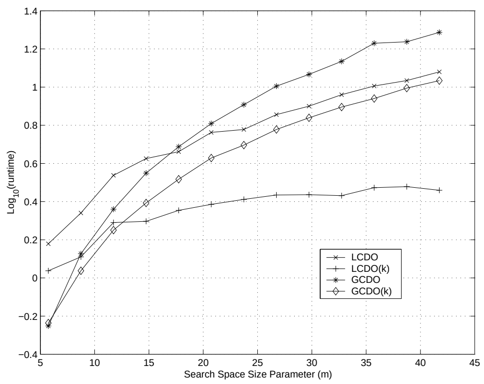


FIG. 9. *CDO, LCDO(k) and G-CDO(k) runtime on BTPC*

designing an even larger system consisting of many such subsystems. In that case, a global optimization strategy typically consists in optimizing the subsystems many times before arriving at a good real-time and cost balance among them.

5. Conclusion and perspectives

In this paper, we have introduced new extensions to the original Force-Directed Scheduling (FDS) algorithm of Paulin and Knight [**214**], mainly for use in the memory access ordering step of our IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration (DTSE) approaches [**170, 38**]. By allowing the designer to tradeoff runtime vs. quality through the choice of parameter *k*, our two algorithms CDO(*k*) and GCDO(*k*) enable the interactive hardware and software design of low-cost but still data-dominated multimedia systems.

Acknowledgments

We thank Denis Vanderstraeten (K.U Leuven), Fabien Coelho (ENSM Paris) and Antoine Fraboulet (ENS Lyon) for their input on graph algorithms, ad-hoc clustering and polynomial-time discrete optimization. We also wish to thank the European Commission for partly funding this work in the frame of the MEDEA System-Level Methods and Tools (SMT) project and G. de Jong from Alcatel for use of the Segment Protocol Processor (SPP) application.

CHAPITRE 6

Multi-Dimensional Selection Techniques for Minimizing Memory Bandwidth in High-Throughput Embedded Systems

This chapter is the original version of our paper accepted for a presentation and an unaltered publication in the proceedings of the 7th Int. High Performance Computing (HiPC) Conference, Bangalore, India, December 17–20, 2000.

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight [**212**] to minimize the number of resources required in the high-level synthesis of high-throughput ASICs. In the frame of our recent Data Transfer and Storage Exploration (DTSE) research [**290, 38, 36**], we have extended FDS for low-cost [1] scheduling in real-time embedded system synthesis. We have shown that FDS is in fact a projected solution to a more general *multi-dimensional space–time* scheduling problem [**175**]. By using this reformulation and by introducing (very) low-complexity *dynamic* and *clustering* graph techniques [**177**], we have shown that the interactive design of low-cost but still high-throughput telecom networks, speech, image and video embedded systems is feasible using a runtime parameterizable Generalized Conflict-Directed Ordering (G-CDO(k)) algorithm. Because G-CDO(k) is based on a true *multi-dimensional* design space exploration mechanism, it is in principle able to analyze design bottlenecks with a much higher resolution than any other technique. In this paper, we develop novel *multi-dimensional* selection techniques to allow this powerful feature. Experiments of redesigning the large-scale and parallel Segment Protocol Processor (SPP) from Alcatel give promising results.

1. Context

Many important software applications deal with large amounts of data, which in turn have to be mapped onto a (partly) predefined hardware platform. This is the case both in multi-dimensio–nal signal processing applications such as video and image processing, which handle indexed signals in the context of loops, and in communication network protocols, which handle large sets of records organized in tables. For these multimedia systems, typically a (very) large part of the area cost is due to memory units. The power is also heavily dominated by the storage and transfers. Hence, we believe that the dominating factor in the system-level design is provided by the organization of the global communication and data storage, and related algorithmic transformations. Therefore we have proposed a systematic design methodology in which the memory organization is optimized as a first step [**273**], **before** the detailed scheduling and mapping onto a (partly) predefined hardware platform is done. Support for this Data Transfer and Storage Exploration (DTSE) methodology is under development in our system-level exploration environment ACROPOLIS [**170, 38, 36**].

1. By low-cost embedded system, we refer to an embedded system exhibiting a good compromise between low-power, low-bandwidth, low-memory size (*i.e.* low-area) and also low-runtime (*i.e.* fast). In our DTSE methodology, this compromise is parameterizable and is a pragmatic choice left to the designer.

Storage bandwidth optimization [**290**] is one of the main steps in our DTSE script. It is executed prior to the Memory Allocation and Assignment (MAA) step [**150, 268**]. The goal of the step is to minimize the required memory bandwidth within the given cycle budget, by adding ordering constraints in the memory access flow-Graph. This allows the memory allocation/assignment tasks to come up with a lower cost memory architecture with a small number of memories and memory ports. Our previous technique to solve the storage-bandwidth optimization step, namely Conflict Directed Ordering (CDO), has resulted in good quality of results for designing large-scale and/or parallel embedded systems under hard real-time constraints [**175, 84, 177**]. In this paper, we will enhance the quality of CDO by introducing *multi-dimensional* selection to enhance the quality of results further.

2. Related work

Scheduling techniques optimizing the number of resources required to meet a deadline have been an active area of research for decades, involving a surprisingly large variety of research communities ranging from operations research for logistics, planning and other military applications; graph theory for high-performance computer-aided design; embedded synthesis and compiler research for more flexible and powerful computer technology to applied artificial intelligence research for next-generation programming environments. All this research relates to some extent to our Conflict-Directed Ordering (CDO) technique although, as will be developing in the coming sections, our particular storage bandwidth optimization formulation leads to a novel and yet still unsolved constrained scheduling problem.

First, the operations research community aims at seeking highly accurate solutions often at the price of complex modeling and large runtime [**8**]. When computer-aided engineering runtime becomes impractical, using exact solution schemes to a derived subproblem is generally proposed. Within this community, resource-minimal scheduling is typically referred to as a *complex* quadratic-assignment problem [**300**]. Hence, for practical runtime, *only* a quadratic-assignment problem is solved, which in turn optimizes the use of the bandwidth without analyzing *which* data is being accessed simultaneously.

In the embedded synthesis and compiler domain, most techniques operate on the scalar-level [**212, 274**], which in turn lead to impractical runtime for real-life applications. Exceptions currently are the PHIDEO stream scheduler [**274**], the Notre-Dame rotation scheduler [**207**] and DRAM-aware optimizations from the University of California at Irvine [**202**]. Many of these techniques try to reduce the memory related cost by estimating the required number of registers for a given schedule. Only few of them try to reduce the memory bandwidth, which they do by minimizing the *number* of simultaneous accesses [**274**]. They do not take into account *which* data is being accessed simultaneously. Also no real effort is spent to minimize the memory access conflict graphs such that subsequent register/memory allocation can do a better job.

In the artificial intelligence domain, it was recently shown that next-generation programming environments [**49, 240**] incorporate features that can solve problems such as the fast synthesis of digital signal processing systems [**254**]. Such constraint-logic programming environments exhibit powerful global optimization capabilities but, to our knowledge, were never applied for advanced memory bandwidth optimization. We believe this is mainly because a fast and effective problem implementation is required in a first step.

The main difference between our storage bandwidth optimization step and the related work discussed here is that we try to minimize the required memory bandwidth statically in advance by optimizing the access conflict graph for groups of scalars within a given cycle budget. We do this by putting ordering constraints on the flow graph, taking into account *which* memory accesses are being put in parallel (*i.e.* will show up as a conflict in the access conflict graph). Last but not least, our techniques operate on groups of scalars instead of on individual scalars [**202**], which we show in this paper can lead to highly accurate solutions when combined with *multi-dimensional* selection techniques.

3. Objectives and contributions

The idea of Force-Directed Scheduling (FDS) was first introduced by Paulin and Knight [**212**] to minimize the number of resources required in the high-level synthesis of high-throughput Applica–tion-Specific Integrated Circuits (ASICs). FDS was later chosen as the basis of the Philips Research - PHIDEO silicon compiler, which resulted in an in-depth study of FDS implementation issues in [**274**]. State-of-the-art in FDS reports a runtime complexity of $O(m^2n^2)$, where n is the number of operations and m is the average time window of each operation, and a feasibility of solving problems of n equals 100 in one minute using the computers of the mid 90s.

Recently, based on the FDS algorithm, an extension called Conflict-Di–rected Ordering (CDO) was proposed [**290**] to minimize the required memory bandwidth of embedded multimedia systems. The CDO algorithm involves an important step in the IMEC - ATOMIUM/ACROPOLIS data transfer and storage exploration approaches [**170, 38**]. Because the CDO solution process requires global system-level analysis, pre-exis–ting optimization techniques do not ful–ly hold and a straightforward implementation is left with a runtime complexity of $O(n^4m)$. This allows to solve problems for n equals 100 in ten hours, even using today's workstation series. Moreover, because the number of operations found in real-life embedded applications is today growing well beyond n equals 100 [**54**] and also because parallelism is often introduced which further increases the number of concurrent operations, a novel fast and effective algorithm for the CDO kernel is needed. To improve the designer interaction further [**268**], it is also desirable to have a controllable trade-off between user time and the quality of the solution (which directly influences the cost of the final solution).

In our recent research, we have revisited the original idea of Paulin and Knight [**212**] to show that FDS is in fact a projected solution to a more general (*multi-dimensional*) *space–time* scheduling problem. Based on this *space–time* formulation, we have introduced a Generalized Conflict-Directed Ordering (G-CDO) algorithm that instead of transforming data that statistically require too much bandwidth can go down to the precision of transforming $(data,ti-me)$ tuples directly. The high resolution of this technique allows a runtime complexity of $O(n^2m^2)$ because a look-ahead is not mandatory any more for precision enhancement. Experiments of using the original CDO and GCDO(k) on redesigning Alcatel's Segment Protocol Processor (SPP) give promising results.

4. Generalized Conflict Directed Ordering

In [**177**], we present our Conflict Directed Ordering (CDO) solution technique. We prove that the original CDO algorithm exhibits a worst-case complexity (in Pairwise Probability of Conflict Computations, PPCCs) of $O(n^4m)$, which can be reduced to $O(n^3ln(n)m)$ using *dynamic* graph techniques and further reduced to $O(kn^2ln(n))$ using *clustering* (also called *multilevel*) techniques. Based on those (very) significant quality and runtime improvements, we introduce the LCDO(k) algorithm.

In [**175**], we show that CDO is in fact a projected solution to a more general (*multi-dimensional*) *space–time* scheduling problem. In order to allow for multiple selection and multiple reductions (MSMR) in full, we introduce the multiple reductions (or *clustered* or *multilevel*) version of the Generalized Conflict Directed Ordering algorithm, G-CDO(k), **without** describing its *multi-dimensional* selection process, that is by implicitly reusing 1D selection techniques.

In the following, we apply *multi-dimensional* selection to G-CDO(k). We first show that a deterministic $O(n^2m)$ implementation does not give satisfactory results. We therefore introduce several probabilistic selection techniques and couple them to the other selection improvements proposed earlier for LCDO in [**177**]. Thanks to its high-resolution and our selection tuning, allowing to remove *space–time* background memory conflict **exactly** where they are, we will see that G-CDO(k) outperforms LCDO(k) for designing large scale and/or parallel systems under stringent real-time constraints in runtime and in quality. The design of Alcatel's Segment Protocol Processor (SPP) will be used as a typical illustration throughout the presentation.

4.1. Deterministic selection (DS). G-CDO is a multi-dimensional space/ti–me scheduler based on a Multi-Dimen–sional Conflict Graph (MDCG) [**175**]. As for the localized version of CDO (LCDO), the complexity of G-CDO can be largely reduced by applying *dynamic* graph techniques [**177**]. Let us first study the simplest way of minimizing bandwidth, that is by using the following cost function:

$$c_1(CG) = \max_t \sum_{(x,t)} \sum_{E(x,t)} \omega(e).1 \tag{25}$$

The effect of removing a node (x,t) from MDCG is first to decrease $\sum_{(x,t)}$ and then eventually to also decrement the maximum over t, $\max_t$. Because the selection process always picks-up the node (x,t) implying the highest cost decrease, it is easy to demonstrate that $\max_t$ is decremented only once in a whi–le when all times exhibiting a high bandwidth cost have been decremented. This principle is general and could apply to any constantly decreasing cost function, even in the presence of global information like the chromatic number (number of memories). Hence, it is in general feasible to implement the above G-CDO selection process in $O(ln(m))$ after a clever $O(n^2m)$ initialization phase[2]. In turn, the reduction can involve a local ASAP/ALAP update such as the one of LCDO, leading to the following worst-case complexity:

$$\begin{aligned} w_2(FG) &= \sum_{v \in EFG} \sum_{i=2}^{m(v)} \frac{1}{i} \times sf(v) \\ &\equiv O(nln(n)ln(m)) \end{aligned} \tag{26}$$

$$\begin{aligned} & O(n^2m) + O(nmln(m)) + \\ & \sum_{i=1}^{n(m-1)} (1 + \frac{2}{n(m-1)}.w_2(FG)).d_i^t \\ & \leq O(n^2m + n^2ln(n)ln(m)) \end{aligned} \tag{27}$$

By comparing formula (27) to formula (15) in Chapter 4, we prove that a multi-dimensional scheduler (G-CDO) can be less time consuming than its mono-dimen–sional counterpart LCDO.

Unfortunately, as shown in Figure 1, an optimally dynamic version of G-CDO does not give satisfactory results in practice. A tradeoff between quality and runtime is therefore needed to compete with the original LCDO.

4.2. Time-hashed probabilistic selection (THPS). In order to improve the quality of G-CDO, we propose to also generalize the pairwise probability of conflict computation (PPCC) present in CDO. For pure bandwidth optimization, this is solved by using the following cost function:

$$c_2(CG) = \max_t \sum_{(x,t)} \sum_{E(x,t)} \omega(e).p(x) \tag{28}$$

2. A max selection can be implemented in $O(ln(m))$ by introducing a sorted list of $\sum_{(x,t)}$. The constantly decreasing property can guide the optimization of the sorted list accessor.

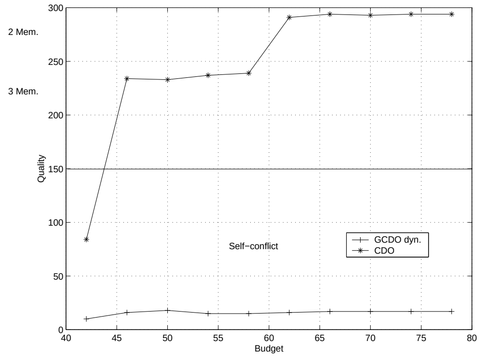


FIG. 1. *Compared quality results for optimally dynamic G-CDO vs. LCDO using our global architecture-aware cost model. Only 2 concurrent memories are needed above 250, 3 concurrent memories above 200 and more below. Below 150, the introduction of expensive multi-port memories becomes unavoidable.*

$$N(x) = \sum_{(x,t)} 1 \tag{29}$$

$$p(x) = \frac{1}{N(x)} \tag{30}$$

Unfortunately, the effect of removing (x,t) from MDCG becomes complex. First, $\sum_{(x,t)}$ is decreased in $O(d_i^t)$, where d_i^t is the degree of the nodes within the time projected conflict graph, CG(t), at step i, $d_i^t \leq n$. Second, the number of time occurrences of access x, $N(x)$, is decreased by 1. Third, for all t where x is present, $p(x)$ is increased by $\frac{1}{N(x)} - \frac{1}{N(x)-1} = \frac{1}{N(N+1)(x)}$, involving $O(d_i^t \times m)$ updates. Fourth, $\max_t$ has to be recomputed. In turn, maintaining feasibility of the solution is ideally achieved by local ASAP/ALAPs relying on $d_i^t \times m$ individual reductions in the loop:

$$\begin{aligned} & O(n^2m) + O(nmln(m)) + \\ & \sum_{i=1}^{n(m-1)} (1 + \frac{2}{n(m-1)}.c_2(FG)).d_i^t.m \\ & \leq O(n^2m^2 + n^2mln(n)ln(m)) \end{aligned} \tag{31}$$

By comparing formula (31) to formula (15) in Chapter 4, we prove that a fully generalized CDO (G-CDO) is slightly less time consuming than LCDO. This situation is improved when multi-level techniques are applied [**175**], thanks to a larger multi-dimensional search space.

Fortunately, as shown in Figure 2, our time-hashed selection technique already gives better quality of results than CDO while still remaining dynamic.

4.3. Multi-dimensional probabilistic selection (MDPS). In order to improve further the quality of G-CDO, it is possible to introduce a summation-based cost function, modeling a multi-dimensional selection process:

$$c_3(CG) \quad = \quad \sum_t \sum_{(x,t)} \sum_{E(x,t)} \omega(e).p(x) \tag{32}$$

The runtime complexity induced by c_3 is very similar to that of c_2, except that the selection becomes more complex. Indeed, in order to keep track of the node to be selected, one needs to first find the

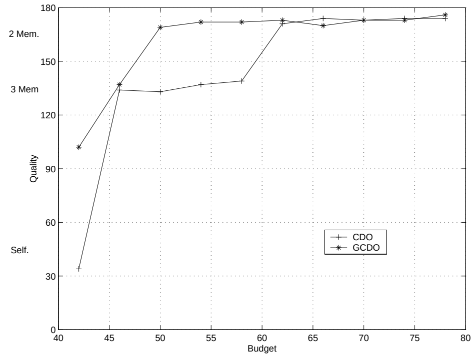


FIG. 2. *Compared quality results for dynamic G-CDO vs. LCDO. using our global architecture-aware cost model. Only 2 concurrent memories are needed above 150, 3 concurrent memories above 120 and more below.*

right t in $O(ln(m))$ and then the right (x,t) in $O(ln(n))$.

$$O(n^2m) + O(nmln(n)) + \sum_{i=1}^{n(m-1)} (1 + \frac{2}{n(m-1)}.w_2(FG)).d_i^t.m \leq O(n^2m^2 + n^2mln(n)ln(m)) \tag{33}$$

By comparing formula (33) to formula (15) in Chapter 4, we prove that a multi-dimensional selection does not impact the runtime complexity of G-CDO.

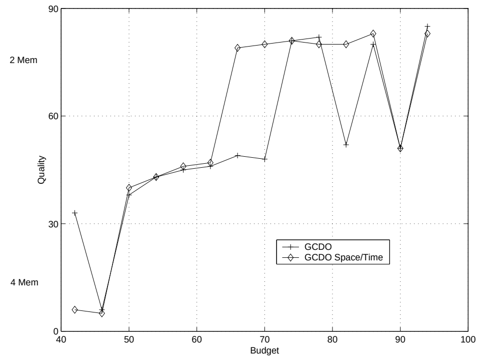


FIG. 3. *Quality enhancement of multi-dimensional selection using our global architecture-aware cost model. Only 2 concurrent memories are needed above 60 and more below.*

Moreover, as shown in Figure 3, mul– ti-dimensional selection can in some ca–ses improve the quality of the original time-hashed solution.

4.4. THPS with probabilistic constraint propagation (CP). When designing large systems, it is common practice to first divide the overall system into p smaller subsystems. When applying automatic design techniques such as G-CDO to those subsystems, it becomes much easier to design each subsystem independently in $O(pn^2m^2)$ than to design and optimize the overall system in full in $O((pn)^2m^2)$. This divide and conquer mechanism is especially interesting in the case of local optimization techniques, typically found in the heuristics used to make a system as fast as possible, because nothing more than just designing or optimizing p times is needed.

Unfortunately, optimizing for making a system as cost-effective as possible, requires **global** optimization techniques. In the case of CDO for example, deciding *which* data can be put in parallel requires (at least) a global track of *which* data have already been put in parallel on other parts of the system. Hence, it is still possible to apply divide and conquer techniques to G-CDO **only** if global constraints are propagated from one subsystem to another. This is for example the case in our data transfer and storage exploration (DTSE) meth–odology while optimizing over several nested loops [**29**]. Each nested loop is regarded as an individual system requiring an individual scheduling.

We propose to model constraint pro–pagation by introducing a $\delta(e)$ in the cost function. This δ means that the knowledge of previous work realized on other parts of the system guides to either take into account a cost factor ($\delta(e) = 1$) because it is new or to cancel it ($\delta(e) = 0$) because it has already been taken into account previously.

$$c_4(CG) = \max_t \sum_{(x,t)} \omega.\delta(e).p(x) \tag{34}$$

$$N(x) = \sum_{(x,t)} 1 \tag{35}$$

$$p(x) = \frac{1}{N(x)} \tag{36}$$

By default, all $\delta(e)$ are equal to 1. In a first step, introducing δ in the G-CDO algorithm impacts the cost of the individual MDCG edge cost computation (ECC), which remains a $O(1)$ process anyway. In a second step, added computation is also introduced when deciding of keeping a conflict, hence necessitating to switch $\delta(e)$ to 0, again in $O(1)$ provided the right data structure for ECC is put into practice[3]. Hence, constraint propagation exhibits the same upper-bound complexity than conventional G-CDO. Some overhead can however be expected in practice.

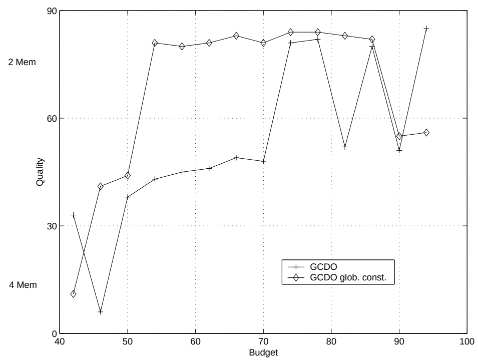


FIG. 4. *Quality enhancement of constraint propagation using our global architecture-aware cost model. Only 2 concurrent memories are needed above 60, 3 concurrent memories above 30 and more below.*

In Figure 4, we show that constraint propagation has a very positive impact on the quality of G-CDO. Indeed, it really makes sense to re-use the knowledge of problems encountered on other parts of the system to guide new optimizations for new subsystems.

4.5. THPS with deterministic constraint propagation (CP). It is also possible to implement constraint propagation on a deterministic conflict graph (CG), that is on a CG where edges are introduced only when a conflict is 100% certain. In that way, the G-CDO algorithm focuses on reducing

3. A two-level pointing technique to compute $\omega(e).\delta(e)$ “on the fly”.

nodes that are certain to generate a conflict. By introducing a deterministic CG, the G-CDO upper-bound complexity is not affected.

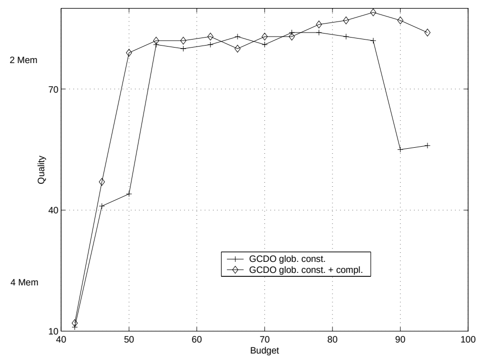


FIG. 5. *Quality enhancement of deterministic constraint propagation reusing the axis definition of Figure 4.*

Surprisingly, as shown in Figure 5, a deterministic CG can contribute to further improve the quality of G-CDO. Indeed, it really makes sense to focus only on those design problems that have 100% chance to occur.

5. Conclusion and perspectives

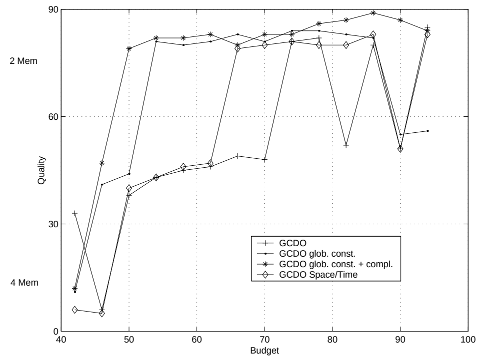


FIG. 6. *Quality results for all presented G-CDO variants reusing the axis definition of Figure 4.*

In Figure 6, we summarize the quality of our four G-CDO variants. By combining all quality enhancement techni–ques, our solution is very significantly improved for hard real-time designs. In particular, designing Alcatel's Segment Protocol Processor (SPP) for a real-time budget in the range of 50 to 70 requires two instead of initially three single-port memories.

Those experiments confirm that *mul–ti-dimensional* selection can be a powerful quality-enhancing technique for designing large-scale and/or parallel low-cost but still high-throughput embedded systems under hard real-time constraints.

Acknowledgments

We thank Arnout Vandecappelle and Erik Brockmeyer for providing support to the use of their tools at IMEC. We wish to thank the European Commission for partly funding this work in the frame of the MEDEA System-Level Methods and Tools (SMT) project and G. de Jong from Alcatel for use of the Segment Protocol Processor (SPP) application.

CHAPITRE 7

A Unified Data-Flow and Resource Conflict Detection Model for Higher Quality Data Access Modelling in the Space–Time Domain

This chapter is the extended version of our paper submitted for a presentation at the EuroPar Conference, Manchester, UK, September 2001 [**191**].

We introduce a unified data-flow and resource conflict detection model exhibiting superior robustness, interactivity, runtime and flexibility fundamentals over the traditional memory-aware extension approach for high-throughput embedded system design. Our model supports data access modeling in the space–time domain without restriction, leading to a time $\times$ consumption gain of 72% over earlier techniques on a real-life network protocol application.

1. Context

Many important software applications deal with large amounts of data, which in turn have to be mapped onto a (partly) predefined hardware platform. This is the case both in multi-dimensional signal processing applications such as video and image processing, which handle indexed signals in the context of loops, and in communication network protocols, which handle large sets of records organized in tables. For these multimedia systems, typically a (very) large part of the area cost is due to memory units. The power is also heavily dominated by the storage and transfers. Hence, we believe that the dominating factor in the system-level design is provided by the organization of the global communication and data storage, and related algorithmic transformations. Therefore we have proposed a systematic design methodology in which the memory organization is optimized as a first step [**273**], **before** the detailed scheduling and mapping onto a (partly) predefined hardware platform is done. Support for this Data Transfer and Storage Exploration (DTSE) methodology is under development in our system-level exploration environment ACROPOLIS [**170**] [**38**] [**36**].

2. Motivation

A growing demand exists for the high-performance computer-aided design (CAD) of high-throughput embedded systems. Earlier work in this area has mainly addressed memory-aware extensions to conventional control data-flow graph (CDFG) models [**150**] [**294**] [**202**] under system specification restrictions such as the single-assignment form [**40**], uniform dependences over iterations [**262**] [**83**] or regular-in-time assignments using stream modeling [**150**]. This has been succesful in demonstrating that a careful ordering of memory accesses can lead to about 30% time performance improvement and about 30% savings in power consumption[1] leading to an overall time $\times$ consumption (C $\times$ T) improvement lower than 2. Unfortunately, the proposed system-level design prototypes were often too slow, too unrobust, and required a high-level of technical expertise which limited their scope to research experiments on applications where near-optimality or CAD runtime is not a major issue. In

1. See for example CDO(k) results in Sections 4.3 and 5 for the latest improvements over [**175**] and [**177**].

order to bridge the gap between the user expectations and the level of maturity of such prototypes, we believe that four major points have to be met:

(1) **High Robustness:** the tool must be robust enough to achieve time $\times$ consumption (C $\times$ T) improvements from the first run compared to existing compiler technology, which otherwise is very discouraging for the user;

(2) **Full interactivity:** a trained designer must be able to achieve a similar or even better result than a CAD expert based on his application knowledge. Moreover, this result should remain within a 10% margin from the ideal solution, leading to C $\times$ T improvements of 3 and more;

(3) **Low Runtime:** the tool must be interactive, that is both fast and able to provide useful feedback to steer further optimisations designer-in-the-loop;

(4) **High Flexibility:** it should also be flexible enough to adapt to existing compilers and tools without restriction.

In order to address these challenges, conventional control data-flow graph (CDFG) models have several drawbacks. First of all, they typically necessitate the addition of a memory resource conflict-detection mechanism either explicitly by introducing a conflict graph [**294**] or implicitly by calling an external conflict-detection routine [**274**] which leads to both an important overhead in software architecture and another overhead in CAD system complexity. Second, conventional CDFG models do not offer by construction the necessary feedback and interaction parameters necessary to reach all acceptable solutions, which means that they take the implicit risk of disregarding a large part of the search space[2]. Moreover, even for the truly explored part of the search space, they typically exhibit non-uniform chances of success w.r.t the solution topology, that is they implicitly incorporate an ad-hoc expectation on the searched solution[3]. Third, once extended for conflict-detection, conventional CDFG models do not easily adapt to existing compilers and tools, especially whenever time- or space-time domain constraints have to be inserted.

To alleviate the three above shortcomings, we introduce a unified space–time and resource conflict detection (ESTCG) model. Compared to conventional CDFG techniques requiring complex CAD extensions for resource-conflict detection, our model can be implemented in no more than 3,000 lines of C++ using open software resources – namely the LEDA Platform for Combinatorial and Geometric Computing [**110**] [**158**] – while achieving a 2 to 5 orders of magnitude lower runtime than existing ordering techniques depending on the problem size (typically within a few seconds of response time). Moreover, we prove in this paper that it provides the necessary feedback and interaction parameters to guarantee by construction that all acceptable solutions can be reached designer-in-the-loop to achieve full interactivity. In practice, this leads to time $\times$ consumption (C $\times$ T) improvements over existing compiler technology of 2 and more from the first run, already outperforming pre-existing memory-aware techniques by about 50% on hard and parallel applications in (very) early user-interaction stages. The proposed framework also offers the full flexibility of propagating to the inside or the outside any local or global, space– time– or space–time constraint that would result either from an advanced interaction with the designer or from the integration with existing compiler and tool environments.

3. Basic framework

In this section, we extend our original definition of the greedy exploration of memory access partial orderings by successive time interval reductions [**177**] to *space–time* exploration using multi-dimensional selection techniques [**176**]. In Subsection 3.1, we generalize our original problem formulation to the *space–time* domain. In subsection 3.2, we demonstrate two superior fundamentals of

2. This will be demonstrated in Section 3. - Theorem 4.

3. This will be demonstrated in Section 3. - Theorem 5.

space–time exploration over conventional approaches. In Subsection 3.3, we apply and evaluate the new concept on a small C/C++ example. In Subsection 3.4, we illustrate its application as the kernel of a high-quality system design framework.

3.1. Problem formulation. **Definition 1.** ***ESTCG*** An extended *space–time* memory access conflict graph $ESTCG = (V,E,C,b,p)$ is a node-weighted and bi-typed edge-weighted graph where $V \subseteq X \times T$ is a set of *acceptable space–time* memory access nodes $v = (x,t)$ where x is a memory access and t is a time $t \in T$; $b \in B^V$ provides the basic group of scalars $b(v) \in B$ of any node; $E \subseteq V \times V$ is a set of unweighted directed edges representing precedence constraints; $C \subseteq V \times V$ is a set of weighted undirected edges representing memory resource conflicts and p the corresponding probability of conflict $p(c)$.

Definition 2. ESTCG(EFG) Starting from a *feasible* extended memory access flow graph $EFG = (V,E,O)$ and an *acceptable* partial ordering O, an extended *space–time* memory access conflict graph $ESTCG = (V_{ST},E_{ST},C_{ST},b,p)$ can be constructed as follows:

$$V_{ST} : \forall v \in V, \forall (v,b,S \subseteq T) \in O, \forall t \in S, \exists! v = (x,t) \in V_{ST}, b(v) = b$$

$$E_{ST} : \forall e = (v_1,v_2) \in E, O(v_1) = (b_1,S_1 \subseteq T), O(v_2) = (b_2,S_2 \subseteq T),$$

$$((x_1, \min(S_1)),(x_2, \min(S_2))) \textbf{ and } ((x_1, \max(S_1)),(x_2, \max(S_2))) \in E_{ST}$$

$$C_{ST} : \forall (v_1 = (x_1,t),v_2 = (x_2,t)) \in (V_{ST})^2, \exists (v_1,v_2) \in C_{ST}$$

Remark 1. The construction of $p(c)$ requires the definition of a multi-dimensional selection strategy [**176**] in a first step. Based on this definition, removing c from C_{ST} whenever $p(c) = 0$ can improve CAD runtime without affecting the algorithms. This is for example an efficient way of solving exclusive accesses in conditional branches, implementing quality-improving global constraint propagations [**176**] and removing impossible conflicts between dependency fathers and sons.

Definition 3. Reduction of an ESTCG Given a *space–time* node $v = (x,t) \in ESTCG$, we denote $EFG \backslash v$ the operator consisting in updating the extended *space–time* memory access conflict graph such that v is removed from the *acceptable* partial ordering represented by the *space–time* nodes.

Theorem 1. Space–time reducibility It is *acceptable* to remove any node $v = (x,t)$ from the $ESTCG$ unless a space row is being emptied.

Proof. Let $v = (x,t)$ a node to be removed. Let $R(v = (x,t)) = V_{ST} \circ V_{ST}^{-1}(v)$ the space row of v. Let $ASAP(R(v)) = \min\{t, (x,t) \in R(v)\}$ the As Soon As Possible (ASAP) timing of space row $R(v)$. Let $ALAP(R(v)) = \max\{t, (x,t) \in R(v)\}$ the As Late As Possible (ALAP) timing of space row $R(v)$. Let us first assume that the space row $R(v)$ is not being emptied – *i.e.* $ASAP(R(v)) \neq ALAP(R(v))$. If $t = ASAP(R(v))$ then $EFG \backslash v$ is *acceptable* because an ALAP scheduling of all space rows is still possible. If $t = ALAP(R(v))$ then $EFG \backslash v$ is *acceptable* because an ASAP scheduling of all space rows is still possible. Otherwise, both the ASAP and ALAP scheduling of all space rows remain possible.

Theorem 2. Sufficient implementation of an acceptable *space–time* reduction In order to implement any reduction $EFG \backslash v$, it is necessary and sufficient:

(1) to remove v from the *space–time* memory access conflict graph $ESTCG$;
(2) if v is an ASAP node, to recursively remove all concurrent dependency sons and to move the dependency edges to the new ASAP node of the current space row $R(v)$;
(3) if v is an ALAP node, to recursively remove all concurrent dependency fathers and to move the dependency edges to the new ALAP node of the current space row $R(v)$.

Remark 2. In a conventional data-flow model, updating the ASAP or the ALAP timing of an access node requires $O(ln(n))$ operations which happens for every reduction. In our unified model, updating the ASAP or the ALAP timing of an access node still requires $O(ln(n))$ operations but the probability of this occurring is low. By carefully comparing the $O(n^3 ln(n) m)$ complexity of our original data-flow technique (see Formula 5. in [**177**]) to the new $O(n^2m^2)$ complexity of its *space–time* implementation (see Formula 9. in [**176**]), we see that the $O(nln(n)/m)$ difference in CAD runtime is exactly the result of a $O(n)$ saving by removing look-aheads, a $O(ln(n))$ saving by pushing ASAP and ALAP updates to become a non-dominant factor and a $O(m)$ overhead for space row traversal in the multi-dimensional probabilistic selection case.

Theorem 3. Unification ESTCG is a unified data-flow and resource conflict detection model

Proof. Starting from any *feasible* extended flow-graph (EFG) exhibiting an *acceptable* partial ordering O, the ESTCG model is sufficient to implement a *space–time* reduction operator and multi-dimensional selection techniques for identifying resource bottlenecks. Therefore, it is unifying the conventional extended flow-graph (EFG) and conflict-graph (CG) models into a single model for the exploration of memory access partial orderings by successive time interval reductions [**177**].

3.2. Main features of the unified model. Theorem 4. Reachability of any solution by steering costs of the *space–time* nodes (Full Interactivity) For any *acceptable* solution, there always exists an *acceptable* perturbation of the initial problem formulation such that our *space–time* ordering techniques are guaranteed to find an equivalent *acceptable* solution.

Proof. Let S be an *acceptable space–time* ordering.

$$S = \{v = (x,t) \in V,\ v\ is\ acceptable\}$$

Let Z be the set of nodes exhibiting an overall cost of zero, so called *free* nodes.

$$Z = \{v = (x,t) \in V,\ \sum_{E(x,t)} \omega(e).p(x) = 0\}$$

Let ε be the minimal cost of non-*free* nodes and M be the maximal cost of non-*free* nodes at extended *space–time* memory access conflict graph (ESTCG) construction-time, for example using Definition 2.

$$\varepsilon = \min_{V\backslash Z}\Big(\sum_{E(x,t)} \omega(e).p(x)\Big) \quad M = \max_{V\backslash Z}\Big(\sum_{E(x,t)} \omega(e).p(x)\Big)$$

Because the cost of an edge can only increase by at most the real-time budget B using our multi-dimensional selection techniques [**176**], the cost of any node at any iteration is bounded as follows:

$$\forall i, \forall v = (x,t) \in V\backslash Z,\quad \varepsilon \leq \sum_{E(x,t)} \omega(e).p(x) \leq BM$$

Therefore, the cost of any non-*free* time column is bounded as follows:

$$\forall i, \forall v = (x,t) \in V\backslash Z,\quad \varepsilon \leq \max_{t} \sum_{(x,t)} \sum_{E(x,t)} \omega(e).p(x) \leq nBM$$

Therefore, it is sufficient to introduce an extra weight $\omega(v) = (\frac{nBM}{\varepsilon} + 1)$ to all nodes in $V\backslash(Z \cup S)$ to strongly guarantee that the multi-dimensional selection process will remove all nodes in $V\backslash(Z \cup S)$ before all others. The result is an acceptable solution $Z \cup S$ exhibiting the same overall cost as S (equivalent solution).

Remark 3. Space-time ordering techniques are ideal for designer-in-the-loop interaction. After running the computer-aided design (CAD) system once, the designer can evaluate the *space–time* result, add local or global constraints down to the resolution of removing or adding any *space–time* feature he wants. By doing this, he has full control over the final solution. In that case, the extended *space–time* memory access conflict graph (ESTCG) acts as an abstract representation of the problem, which is better suited for making decisions. This is a typical user requirement of both application-specific integrated circuit (ASIC) design [**264**] and distributed software design [**200**] [**78**] environments, which cannot be achieved by *flat*[4] data-flow models, involving at best n memory access node parameters to control $n(m-1)$ design decisions.

Theorem 5. Uniform chances of success w.r.t near-optimal solutions (High Robustness) Only a *space–time* decision process guarantees uniform chances of finding any acceptable solution – including near-optimal ones – using greedy exploration of memory access partial orderings by successive time interval reductions.

Proof. On one hand, by allowing *space–time* decisions at each iteration, the chances of not finding a particular near-optimal solution is $(m-1)\frac{1}{m}$ and therefore the chances of finding a particular near-optimal solution remains $\frac{1}{m}$ independent of its position within the explored interval.

On the other hand, by limiting the decision to the extreme points of the interval, which is the case in conventional exploration techniques [**212**] [**274**] [**291**], the probability of finding the optimal solution $s \in [1,m]$ is

$$(\frac{1}{2})^{m-1}.C^m_{s-1}.C^m_{m-s} = (\frac{1}{2})^{m-1}.\frac{(m-s+1)}{s}.[\frac{(m-s+2)...m}{2.3...(s-1)}]^2$$

If the solution is an extreme point, the chances of finding it is only $(\frac{1}{2})^{m-1}$ which converges to 0 for large-scale problems. If the solution is the middle of the interval, the chances of finding it are much higher than $\frac{1}{m}$.

Remark 4. By allowing uniform chances of success w.r.t near-optimal solutions, *space–time* explorations are more suitable than conventional exploration techniques especially for solving large-scale (often parallel) memory access ordering problems exhibiting near-optimal solutions at the edge of precedence constraints, which is a typical characteristic of hard scheduling problems.

3.3. A parallel and hard computation of four series. Imagine that you have at your disposal two 1-port memories to set $3n$ terms of the following three straightforward series and only $2n$ terms of the following fourth alternating series. At the end of the computation, all $3\times 3n+2n = 11n$ terms must be stored in main memory. For the sake of this presentation, we assume that a call to functions f, g, h takes one memory cycle and that n is a multiple of 18.

$$s_1^{(i)} = f(i) \quad s_2^{(i)} = g(i) \quad s_3^{(i)} = h(i)$$

$$s_4^{(2i)} = s_1^{(3i)} \times s_3^{(3i)} \quad s_4^{(2i+1)} = s_1^{(3i+1)} \times s_3^{(3i+1)}$$

In order to tackle this problem, you will probably start by writing the following C/C++ code:

```
float S1[3*N],S2[3*N],S3[3*N],S4[2*N];
for (int i=0; i<3*N; i++) {
  S1[i]=f(i); S2[i]=g(i); S3[i]=h(i);
};
for (int i=0; i<N; i++) {
  S4[2*i]=S1[3*i]*S3[3*i]; S4[2*i]=S2[3*i+1]*S3[3*i+1];
};
```

In order to avoid loading back $s_1^{(3i)}$,$s_3^{(3i)}$,$s_2^{(3i+1)}$ and $s_3^{(3i+1)}$, you will probably merge the two loops into a single loop nest:

4. By *flat* vs. *space–time* data-flow models, we mean non *space–time* expanded data-flow models (within the same abstraction level). This should not confuse the reader with *flat* vs. *hierarchical* data-flow models [**29**] that provide a mechanism for higher-level abstractions.

Algorithm	B	T	Architecture	C	C×T
G-CDO(k)	36	+0.0%	2 1-port	+0.0%	+0.0%
BC-LS	36	+0.0%	2 1-port	+0.0%	+0.0%
AC-LS	36	+0.0%	2 1-port	+0.0%	+0.0%
CDO(k)	43	+19.4%	3 1-port	≈ +20.0%	+43.3%
Conv.LS	48	+33.3%	4 1-port	≈ +40.0%	+86.7%

TAB. 1. *Distance to the optimum solution after applying access ordering techniques to our C/C++ hard and parallel system-level specification example. Design optimization runtime 14.6s. on a 533MHz Celeron processor or equivalent ($n = 66, m = 36, n^2m^2 = 6 \times 10^6$). B (real-time Budget), T (Time), C (Cost).*

```
float S1[3*N],S2[3*N],S3[3*N],S4[2*N];
for (int i=0; i<N; i++) {
  S1[3*i]=f(3*i); S2[3*i]=g(3*i); S3[3*i]=h(3*i);
  S4[2*i]=S1[3*i]*S3[3*i];
  S1[3*i+1]=f(3*i+1); S2[3*i+1]=g(3*i+1); S3[3*i+1]=h(3*i+1);
  S4[2*i+1]=S1[3*i+1]*S3[3*i+1];
  S1[3*i+2]=f(3*i+2); S2[3*i+2]=g(3*i+2); S3[3*i+2]=h(3*i+2);
};
```

This results in 11 memory writes per loop nest, which cannot be divided by 2. Given the two 1-port memories, you will probably think of unrolling for improving performance:

```
float S1[3*N],S2[3*N],S3[3*N],S4[2*N];
for (int i=0; i<N/2; i++) {
  S1[6*i]=f(6*i); S2[6*i]=g(6*i); S3[6*i]=h(6*i);
  S4[4*i]=S1[6*i]*S3[6*i];
  S1[6*i+1]=f(6*i+1); S2[6*i+1]=g(6*i+1); S3[6*i+1]=h(6*i+1);
  S4[4*i+1]=S1[6*i+1]*S3[6*i+1];
  S1[6*i+2]=f(6*i+2); S2[6*i+2]=g(3*i+2); S3[6*i+2]=h(3*i+2);
  S1[6*i+3]=f(6*i+3); S2[6*i+3]=g(6*i+3); S3[6*i]=h(6*i);
  S4[4*i+2]=S1[6*i+3]*S3[6*i+3];
  S1[6*i+4]=f(6*i+4); S2[6*i+4]=g(6*i+4); S3[6*i+4]=h(6*i+4);
  S4[4*i+3]=S1[6*i+4]*S3[6*i+4];
  S1[6*i+5]=f(6*i+5); S2[6*i+5]=g(6*i+5); S3[6*i+5]=h(6*i+5);
};
```

This results in 22 memory writes per loop nest. Given the two 1-port memories, this computation can in principle be achieved in 11 memory cycles. However, although this example exhibits full parallelism of the memory writes, this ideal bandwidth cannot be met because arrays have to be mapped in one or the other 1-port memory. This means that only 8 of the 16 possible `Si[.]`–`Sj[.]` concurrent writes are allowed in practice. Let us assume that `S2` and `S4` are placed in one 1-port memory and that `S1` and `S3` are placed in the other 1-port memory. In that case, the 10 accesses to `S2` and `S4` can be made concurrent with 10 of the 12 accesses to `S1` and `S3`. However, the 2 remaining accesses to `S1` and `S3` have to be sequential, leading to a practical optimum of only 12 memory cycles. A better solution could be achieved for example by breaking `S1`, `S2` and `S3` into sub–arrays [**225**].

In order to build a C/C++ system-level specification example reaching a realistic size, we have further unrolled the merged loop by a factor of 3, leading to 66 memory accesses per loop nest. Using this hard and parallel C/C++ system-level specification example for which the practical optimum solution is known well in advance, we have applied several memory ordering techniques. In Table 1, we present the results of this experiment, which clearly shows that our *space–time* ordering technique G-CDO(k) finds the practical optimum and outperforms flat data-flow techniques like CDO(k) or conventional list-scheduling available from conventional compilers by 30% and more even **before** any user-interaction is started. This clearly illustrates some of the most interesting features of our unified data-flow and resource conflict detection model.

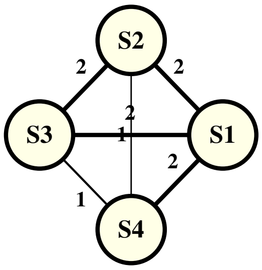


FIG. 1. *Global memory access to basic groups of scalars conflict-graph (BG-CG) obtained after applying a conventional list scheduling algorithm to our unrolled C/C++ system specification involving 22 memory writes per loop nest*

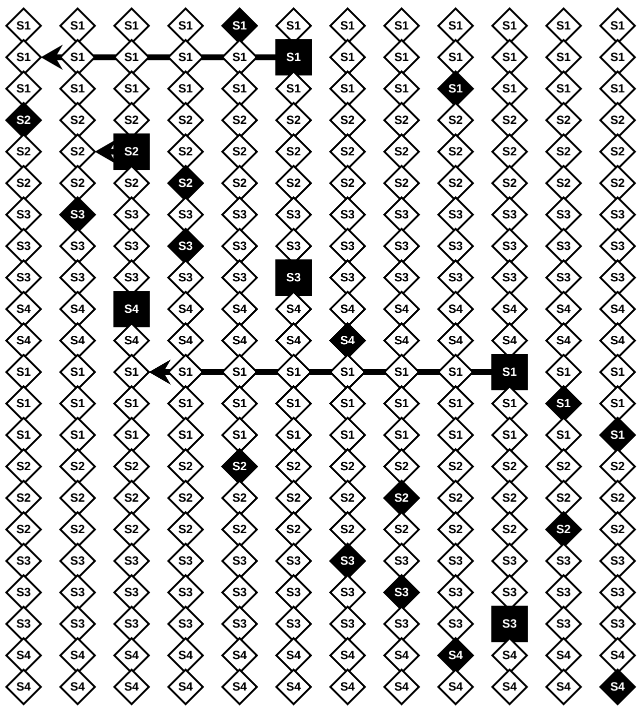


FIG. 2. *Drag-and-drop decisions of the trained designer after selecting edges in the global memory access to basic groups of scalars conflict-graph (BG-CG) and studying* space–time *rescheduling opportunities*

3.4. A high-quality interactive system design scenario. Let us now assume that the designer is not using a near-optimal solver like our GCDO(k) algorithm. Starting from the unrolled C/C++

Algorithm	B	T	Architecture	C	C×T
G-CDO(k)	45	+0.0%	2 1-port (750mW)	+0.0%	+0.0%
AC-LS	43	-4.4%	3 1-port (950mW)	+26.7%	+21.0%
CDO(k)	58	+34.9%	1 2-p. + 3 1-p. (1000mW)	+33.3%	+71.85%
BC-LS	63	+46.5%	p multi-port (>1450mW)	>+93.3%	>+170.6%
Conv.LS	72	+67.5%	p multi-port (>1450mW)	>+93.3%	>+209.3%

TAB. 2. *System performance and cost degradation compared to using space–time ordering techniques for Alcatel's Segment Protocol Processor (SPP) C/C++ system-level specification. Design optimization runtime is 14.7s. on a 533Mhz Celeron processor or equivalent ($n = 156, m = 20, n^2m^2 = 9{,}61 \times 10^6$). B (real-time Budget), T (Time), C (Cost).*

system specification involving 22 memory writes per loop nest, he might be fed back a **global** memory access to basic groups of scalars conflict-graph (BG-CG) requiring the use of an expensive four 1-port memories architecture. This is for example the case with the BG-CG resulting of a run of a conventional list scheduling algorithm (Conv.LS) presented in Figure 1.

From this (very) useful information, a trained designer should immediately realize (or could even be automatically notified) that the expensive use of a four 1-port memories architecture comes from any two non-adjacent edges out of the six edges present in the BG-CG. He should also quickly realize that in order to remove two non-adjacent edges, at least three memory accesses have to be rescheduled just by looking at the occurrence numbers on the edges of the BG-CG. Therefore, he should realize that a good idea to improve the solution is for example by deciding to remove all conflicts between `S2` and `S4` and all conflicts between `S3` and `S1`. He will therefore click on the `S2-S4` and `S3-S1` edges of the BG-CG to notify his intention to explore the removal of those two bottlenecks. As a result, the *space–time* memory access diagram presented in Figure 2. pops up. White *space–time* nodes indicate the original ordering search space. Black *space–time* nodes indicate the current solution. Black *space–time* nodes emphasized in a large square box indicate *space–time* nodes to be explored in more detail. By clicking on any *space–time* node to be explored, the designer can be notified all *acceptable* positions in its *space–time* row satisfying the wanted global cost reduction. In our example, he will therefore (very) quickly realize that by dragging three *space–time* nodes to a new position he can solve his design problem. Hence, a trained designer can be as efficient as our GCDO(k) algorithm to retrieve the opportunities left out by conventional compilers. Moreover, by doing this, he can impose any local or global, space- time- or *space–time* constraint he wants to impose on the final solution.

4. Case study: Segment Protocol Processor (SPP)

In this section, we evaluate the results of redesigning a large-scale embedded system, namely the Segment Protocol Processor (SPP), which is part of the Asynchronous Transfer Mode (ATM) switching mechanism from Alcatel Micro-Electronics [**257**]. SPP is a Telecom module involving the dynamic routing and buffering of ATM messages by successively decoding parts (also called segments) of its self-contained routing information. After applying our Dynamic Memory Management (DMM) script to the ATM routing standard [**279**] [**295**], SPP is refined to a specification exhibiting a complex conditional control flow with a total of 156 accesses to 14 arrays. In this context, the role of our storage-bandwidth optimization step is to multiplex the 156 accesses taking into account their exclusive patterns to minimize the required memory bandwidth and cost.

In Table 2, we present the results of redesigning SPP for a programmable platform with two 1-port memories using our in-house toolbox of memory access ordering algorithms [**84**]. For this case study, our space–time ordering technique G-CDO(k) [**175**] [**176**] can be 67.5% faster and 93.3% more cost-effective than flat data-flow techniques like conflict-directed ordering (CDO) [**294**], budget-constraint list-scheduling (BC-LS) [**85**], architecture-constraint list-scheduling (AC-LS) [**264**] [**85**] or conventional list-scheduling (Conv.LS) [**20**] as available from most compilers **before** any user-interaction happens.

Starting from these initial high-level specification refinements, should the designer have to meet the most stringent real-time constraints (say $B = 43$), he is likely to install the fastest version of his algorithm onto the specified two 1-port memories programmable platform. In that scenario, time is going to be degraded **and** time- and energy- consuming conflict resolution is going to happen at runtime [**175**]. We believe that the product C$\times$T is an optimistic estimator of the performance degradation induced by this crude user-interaction scenario. Using the C$\times$T estimator, we predict a performance degradation of up to 209.3% (*i.e.* pessimistic performance improvement of 3.1) if using conventional compiler techniques like List Scheduling (Conv.LS) instead of our proposed space–time ordering techniques for re-designing Alcatel's Segment Protocol Processor (SPP). In [**202**], the University of California at Irvine observes similar time performance improvements of 40% and more by applying high-level memory optimization techniques to a set of ten small application benchmarks. This also confirms our earlier prediction that the product C$\times$T can improve by more than a factor of 2 by deploying near-optimal memory-aware system-level techniques as opposed to conventional scheduling-for-speed approaches [**175**].

5. Conclusion

In this paper, we have introduced a new *space–time* model able of unifying data-flow and resource conflict detection for the high-quality design of high-throughput embedded systems. Features of this new model include:

(1) **High Robustness:** a systematic and significant time $\times$ consumption (C $\times$ T) improvement compared to existing compiler technology from the very first system-level specification refinement as illustrated in Sections 3.3 and 4;

(2) **Full Interactivity:** a full controllability over the final solution by the trained designer as demonstrated in Section 3.2 and illustrated in Section 3.4, involving time $\times$ consumption (C $\times$ T) improvements above a factor 3;

(3) **Ultra-Low Runtime:** an extremely fast and accurate *space–time* feedback to steer further system improvements designer-in-the-loop which is a popular interaction scenario both within the distributed software design [**200**] [**78**] and the application-specific IC [**264**] computer-aided design (CAD) communities;

(4) **High Flexibility:** a flexible software architecture able of integrating with third-party compilers and tools by means of any local or global, space– time– or *space–time* constraint propagation, reaching the level of flexibility of today's most intelligent decision-support environments [**49**] [**240**] [**18**].

Experiments prove that our *space–time* model allows time $\times$ consumption (C $\times$ T) improvements beyond a factor of 2 on an automatic basis starting from C/C++ specifications. To our knowledge, this is the first time such high-quality results are met by any memory-aware precompiler on large-scale industrial applications.

CHAPITRE 8

Partial Exploration Techniques for Minimizing Memory Bandwidth in High-Throughput Telecom Networks and Speech Embedded Systems

This chapter is the original version of our paper rejected for a presentation at the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD'2000) and later again rejected at the VLSI'2001 Workshop. It seems to be extremely difficult to publish any work on List Scheduling (LS) nowadays, so we have given up on publishing this material which remains an IMEC-DESICS Internal Report [**85**].

The major problem faced in designing next generation multimedia is to bridge the gap between high-performance memory to processor communication specifications and the (very) low-cost requirements of the mass market. One important step to address this challenge is to introduce novel scheduling techniques still exhibiting high-performance but at the price of much less resource utilisation. In practice, because human decisions are involved in the tradeoff between cost and functionality, such techniques ideally fit into interactive precompilation environments - such as the IMEC - ATOMIMUM/ACROPOLIS data transfer and storage exploration (DTSE) environment [**170, 38**].

Unfortunately, in order to make this interactivity possible, (very) fast scheduling heuristics have to be introduced [**175, 177**] and some quality of solution is sacrificed. On one hand, in order to come up with a solution as close as possible to the optimal implementation, it makes sense to explore the design space in *full* [**175, 177**] using elementary algorithms, such as a greedy search. On the other hand, by prefering *partial* exploration of the design space, less optimal solutions can be found with the higher accuracy of advanced search techniques such as evolutionary algorithms.

In this paper, we introduce three promising techniques for minimizing memory bandwidth in high-throughput telecom networks and speech embedded systems using *partial* exploration techniques.

1. Introduction

To date, state-of-the-art in *full* design space exploration for memory bandwidth minimization involves the *divide-and-conquer* implementation of *dynamic* and *adaptive* algorithms exhibiting an overall $O(n^2 ln(n) ln(m))$ runtime complexity [**177**] [1], where n is the number of memory accesses and m is the average scheduling freedom per access of the embedded system being designed. For this *full* search strategy, we have recently demonstrated a theoretical runtime complexity upper-bound of $O(nln(n) \times mln(m))$ [**177**]. However, it is worth mentioning that this *ultrafast* solution can be achieved **only** at the price of large computer-aided design (CAD) software development efforts [**110**] and also some loss in the quality of the final design solution [**177**].

In order to alleviate this situation, we have therefore also investigated alternative *partial* design space exploration techniques for memory bandwidth minimization. By introducing Conflict-Directed

1. The $O(n^2 ln(n) ln(m))$ complexity is an extension of the divide-and-conquer complexity result presented in [**177**] Section 5.3 under the assumption of a symmetric *fractal* growth of embedded software — *i.e.* the number of subsystems grows as fast as the size of individual subsystems.

List Scheduling (CDLS) and Stochastic Evolution-based Conflict-Directed Ordering (SE-CDO) algorithms, we show that a *partial* space exploration can reach an *ultra-low* runtime complexity of $O(nm)$ while still being implementable at the price of (very) small CAD software development efforts.

Experiments of using CDLS and SE-CDO on real-life telecom networks and speech applications exhibit very encouraging quality of solutions, often outperforming the conventional *full* design space exploration techniques. This result suggests that memory bandwidth optimization can take advantage of **many** near-optimal solutions scattered all over the design space, allowing *partial* design space exploration to compete and even outperform *full* exploration techniques.

We therefore believe that CDLS and SE-CDO are very good candidates to complement our LCDO(k) [**177**] and G-CDO(k) [**175**] algorithms within the low-cost scheduling toolbox of our ACROPOLIS data transfer and storage exploration (DTSE) environment [**170, 38**].

See `http://www.imec.be/acropolis/` for on-line download and demo.

2. List Scheduling

In this section, we introduce two novel ordering algorithms mainly based on the List Scheduling (LS) paradigm [**20**]. Each of them responds to a different strategy. Architecture Constrained Conflict Directed List Scheduling (AC-CDLS) takes a memory architecture as an input and finds the best scheduling — *i.e* with the lowest time budget — regarding these constraints. On the opposite, Budget Constrained Conflict Directed List Scheduling (BC-CDLS) tries to find the best memory architecture within a given time budget. The main difference between the conventional list scheduling approach and ours is that we introduce a global view of the problem [**213**] by using a Conflict Graph (CG). List Scheduling has only a local impact, but in order to minimize the memory requirements we have to look at *which* data is accessed. During the execution when two accesses are being placed in parallel, we update the CG by adding an extra edge between the two corresponding basic groups of scalars (*e.g.* arrays). This allows to memorize all the created conflicts and as a result better organize the future scheduling [**177**].

2.1. AC-CDLS. At each iteration, AC-CDLS performs a lookahead on all schedulable nodes. If none of them meets the memory constraints we add a time to the budget. If at least one of them meets the constraints, the one with the highest priority (ALAP minimum, force calculations, ...) is scheduled.

Memory is constrained by a Conflict Graph that we provide as an input. But if the memory architecture is not yet fully defined it is also possible to constrain search by giving more ”general” constraints:

(1) Checking that no self conflicts are created to avoid multi-port memories;
(2) Checking that no more than nm memories are created(by analyzing the chromatic number of the CG;
(3) Checking that no more than nc conflicts are created.

2.2. BC-CDLS. The determination of the optimum time is based on two criteria :

(1) the number of conflicts increase;
(2) the later a will be scheduled, the more it will reduce the ASAP/ALAP interval of its descendants, and so there will be less freedom to schedule them in the future.

This leads to the following cost function:

Let b be the time budget.

Let $C1(t)$ be the number of conflicts created when scheduling a on time t.

```
begin
   Perform ALAP analysis;
   S := {roots of EFG}
   reducible := {nodes of EFG};
   t := 1;
   while (!reducible.empty) do
          acceptable.clear;
          for (v ∈ S) do
               try schedule v on time t;
               if (memory constraints satisfied)
                  acceptable := acceptable + {v};
               end
               if (acceptable.empty)
                  t := t + 1;
            else schedule highest priority;
                  update memory conflicts;
                  reducible.delete(w);
                  S = {v ∈ reducible, f(v) not reducible};
               end
          end
   end
   budget_required = t;
end
```

TAB. 1. *Architecture-Constrained Conflict-Directed List Scheduling (AC-CDLS).*

```
begin
   Perform ALAP analysis;
   S := {roots of EFG}
   reducible := {nodes EFG};
   t := 1;
   while (!reducible.empty) do
          find a ∈ S with minimum ALAP;
          Find optimum t ∈ [ASAP,ALAP](a);
          update memory conflicts;
          reducible.delete(a);
          S = {v ∈ reducible, f(v) not reducible};
   end
end
```

TAB. 2. *Budget-Constrained Conflict-Directed List Scheduling (BC-CDLS).*

Let $C2(t)$ estimate the freedom that will be remove when scheduling a on time t. A good approximation of this number is to make it proportional to $t - ASAP(a) + 1$, but in order to increase the accuracy we can also multiply it by its number of sons $s(a)$. This gives $C2(t) = (t - ASAP(a) + 1) \times s(a)$.

Indeed, we can also approximate $s(a)$ by $b - ALAP(a)$. Potentially, the more $ALAP(a)$ is close to b, the less it will have descendants.

Finally the C2 cost function is becoming :

$$C2(t) = \frac{t - ASAP(a) + 1}{ALAP(a) - ASAP(a) + 1} \times \frac{b - ALAP(a)}{b} \tag{37}$$

So, our internal cost function is:

$$C(t) \quad = \quad C1(t) + k \times C2(t) \tag{38}$$

where k is a parameter to balance $C1$ and $C2$. But we also have to check that no self-conflicts are created and that the number of memories is not increased. These two tests are similar to those made in AC-CDLS. Here, the priority is:

(1) no self-conflict creation;
(2) no extra memory required;
(3) minimization of the cost function C(t).

We can illustrate the behavior of the algorithm when k is varying. If k is equal to 0, BC-CDLS(0) only tries to reduce the number of conflicts. From a global point of view, it introduces as less parallelism as possible but this quickly reduces the ASAP/ALAP intervals such that there is no way to avoid costly conflicts at the end of the scheduling. Therefore BC-CDLS(0) tends to introduce a high concentration of accesses at the end of the schedule and a low concentration at the beginning, as shown in Figure 1. On the contrary, for k maximum, BC-CDLS(k_{max}) tries to introduce as much

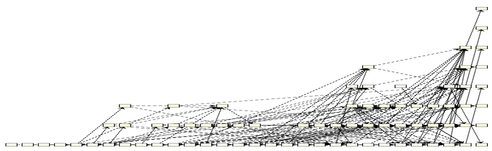


FIG. 1. *Space–Time spectra for BC-CDLS(0)*

parallelism as possible without optimizing the number of conflicts. Therefore, BC-CDLS(k_{max}) finds tends to introduce a high concentration of accesses at the beginning of the schedule and a low concentration at the end, as shown in Figure 2. In Figure 3, we show that it is possible to tradeoff between

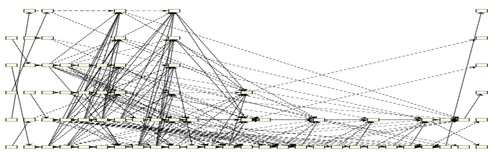


FIG. 2. *Space–Time spectra for BC-CDLS(k_{max})*

the two extreme scenarios of BC-CDLS(0) and BC-CDLS(k_{max}) by choosing k in $[0, k_{max}]$. For the optimal choice of k, BC-CDLS(k_{opt}) can maintain the same parallelism during all the schedule, hence minimizing the required memory bandwidth as shown in Figure 4.

3. Stochastic Evolution

We have also explored the class of evolution algorithms (EAs) for the solution of the CDO problem [**228**]. Indeed, EAs represent a powerful alternative to descent algorithms or list scheduling techniques thanks to the possibility of escaping local optima of the cost function. In particular, we

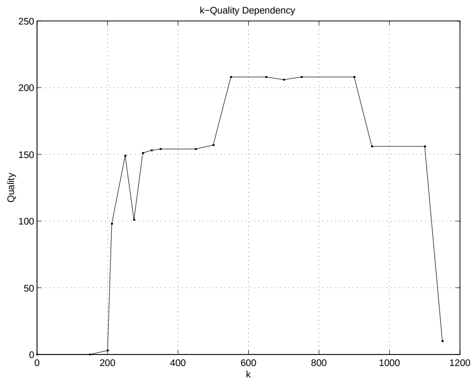


FIG. 3. *BC-CDLS(k) quality of solutions*

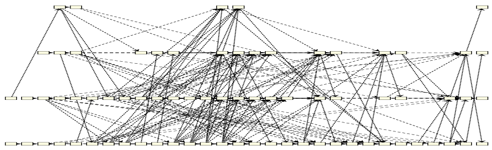


FIG. 4. *Space–Time spectra for BC-CDLS(k_{opt})*

have investigated a modified version of the *Simulated Annealing* algorithm [**115**] (called *Stochastic Evolution* in [**232**] and further developed in [**269**]).

Consider the *solution space* as the set of solutions — *i.e.* any scheduling of the nodes that satisfies the dependency constraints and assume that a cost function to be minimized is defined on the solution space. The idea behind the algorithm is to explore the solution space favoring directions where the cost of the solutions seems low. In Stochastic Evolution, the exploration selects randomly a neighbor of the current solution and evaluates the *gain* Δ, defined as the difference in cost. If the cost of the neighbor is lower, it is accepted with probability 1 and becomes the new current solution. Otherwise, the probability of acceptance is determined by the gain and by a control parameter K called *temperature*. At the beginning, the temperature is high and neighboring solutions have a high probability of being accepted even if they increase the cost function. Every n proposed neighboring solutions, the temperature is gradually decreased up to a point where the system freezes, accepting only neighboring solutions that decrease the cost function. The frozen system is re-heated $M-1$ times to allow the exploration of several local minima.

In our implementation, we start from the ASAP solution and scale it such that the entire budget is covered. A *neighboring* solution is obtained by re-scheduling one single node of the current solution; the new time is chosen to satisfy the dependency constraints. The *cost* of a schedule is evaluated from the conflict graph associated with the schedule. It is defined as the weighted sum of the number of self-conflicts, the number of edges of the conflict graph, and the total weight of the edges (the weight of an edge represents the number of times a given conflict appears in the scheduling). We have decided not to include the chromatic number of the conflict graph in our cost function. Its evaluation is computationally expensive while estimates are not sufficiently accurate to provide any valuable information. On the other hand, we consider the total weight of the edges because its addition provides some variability to the cost function and increases the quality of the solutions.

The pseudo-code of the algorithm in Table 3. The parameters K_{ini}, K_{min}, and $\rho < 1$ define the cooling strategy of the temperature for each pass while M denotes the number of minima to be explored.

```
begin
  S := get_an_initial_sol()
  S_best := S
  for (i ∈ [1,M]) do
     K := K_ini
     while (K > K_min) do
        for (j ∈ [1,n]) do
           S_neigh := get_a_neighbor(S)
           Δ := cost(S_neigh) − cost(S)
           if (Δ/K ≤ rand[0...1])
              S := S_neigh;
           end
           K = ρ K;
        end
        if (cost(S) < cost(S_best))
           S_best := S;
        end
     end
  end
```

TAB. 3. *Conflict-Directed Stochastic Evolution (CD-SE).*

The simplicity of the algorithm makes it powerful in terms of complexity: The main operations consist in updating the set of neighbors of the current solution and in computing the gain. Updating the set of neighbors requires to recompute the ASAP/ALAP values of the sons and fathers of the node being rescheduled . This can be performed in $O(d)$ operations, where d is the average degree of the nodes in the EFG graph. The computation of the gain is based on the conflict graph. It merely consists in a loop over the possible edges adjacent to the "memory" corresponding to the node being rescheduled and and the complexity grows like $O(b_g)$ with b_g representing the total number of memories ($b_g \ll n$).

Finally, the succession of operation "chose a neighbor, evaluate the gain and update the set of neighbors" is performed n times in the inner loop of Table 3. The outer loop depends on M and on the cooling strategy only; it is problem independent. Hence, the global complexity becomes

$$\text{complex}(SE) = O\left(n \times \max(b_g,d)\right) \equiv O\left(nm\right)$$

4. Experimental results

The Stochastic Evaluation approach has been tested on the SPP and VOCODER problems with 156 and 76 nodes respectively

Quality results are presented in Fig. 5 and 6 and timings, obtained on a XX computer, are shown in Fig. 7 and 8. Because the algorithm contains a random function, the quality of the final solution may vary from one execution to another. Therefore, we have performed 4 executions for each data set and retained the best of the four solutions.

Compared to the other approaches, the Stochastic Evolution algorithm (noted SE) performs reasonably well. It also exhibits a reasonable robustness, i.e. the number of self-conflict is minimized and the number of colors is small. Remember though that the number of colors is *not* an objective in the cost function. Unfortunately, our experience with evolution algorithms suggests that robustness is not scalable for large values of n since the location of the initial solution may become too far from a good solution.

For the SPP problem, SE proposes about 70 000 neighboring solutions and accepts 40% of them. Also, 15 minima are explored for each execution. For Vocoder, about 40 000 neighbors are selected with again a probability of acceptance around 40%. This is one of the key features of SE: by proposing and accepting a large amount of neighbors, it is able to cover an extended area of the search space, therefore increasing the chance to find a "deep" minimum.

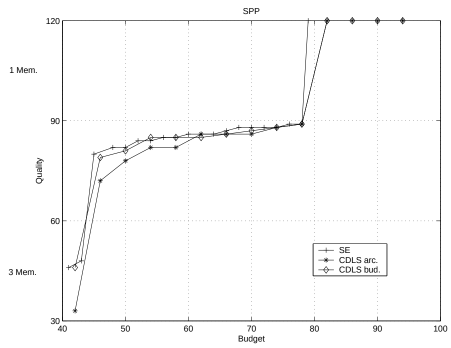


FIG. 5. *SE-CDO quality for Alcatel's SPP*

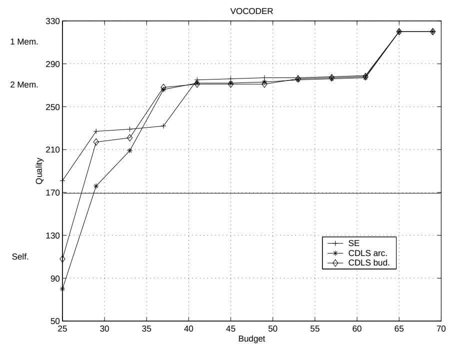


FIG. 6. *SE-CDO quality for SAIT Electronics's Vocoder*

From a complexity perspective, it can observed that the computation time with SE is independent of the budget. This is in agreement with our complexity analysis. Variations are due to the fact that extra-computation is allowed every time a "so-far" global optimum is attained (by increasing the parameter M).

5. Conclusion

In this paper, we have introduced three promising techniques - namely AC-CDLS, BC-CDLS and CD-SE - for minimizing memory bandwidth in high-throughput telecom networks and speech embedded systems using *partial* exploration techniques. We have illustrated very good runtime and quality of results on two real-life examples. We therefore believe that AC-CDLS, BC-CDLS and CD-SE are very good candidates to complement our LCDO(k) [**177**] and G-CDO(k) [**175**] algorithms within the low-cost scheduling toolbox of our IMEC - ACROPOLIS data transfer and storage exploration (DTSE) environment [**170, 38**].

We are currently tuning an intelligent approach, able to identify the best scheduling strategy to be used given application characteristics. This will be discussed in future work.

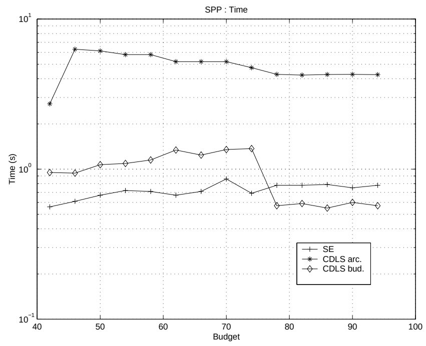


FIG. 7. *SE-CDO runtime for Alcatel's SPP*

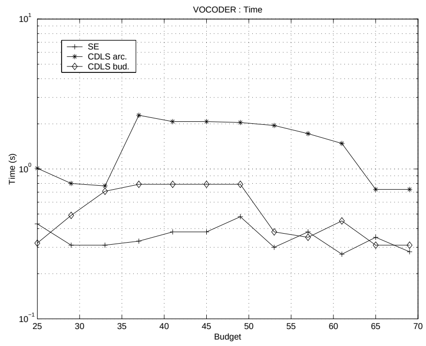


FIG. 8. *SE-CDO runtime for Alcatel's SPP*

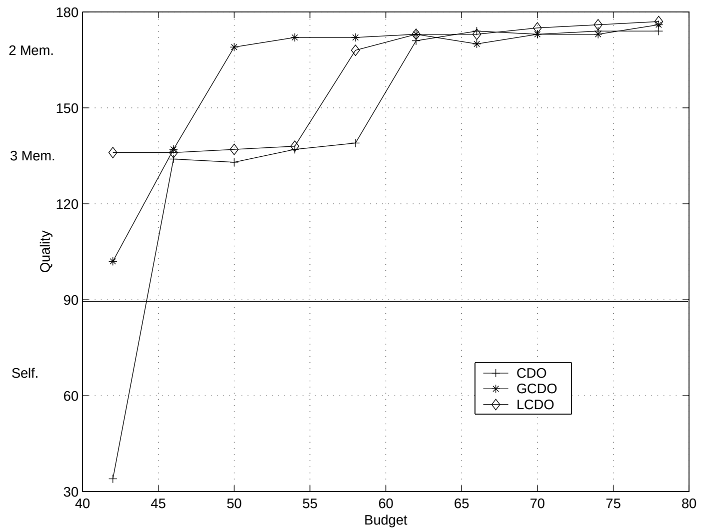


FIG. 9. *Quality results for best CDO variants*

Acknowledgments

Our research has been partly supported by the SMT project of the Flemish Community - IWT and the UIAP P4/02 Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's

Office for Science Technology and Culture and by the *Fonds voor Wetenschappelijk Onderzoek* (FWO project G0344.98). The scientific responsibility rests with its authors.

CHAPITRE 9

Space–Time Memory Access Patterns: a Step towards the Object-Oriented Design of Low-Cost Distributed Platforms

This chapter is the extended version of our paper submitted to the 5th Int. Wsh. on Software and COmPilers for Embedded Systems (SCOPES), St Goar, Germany, March 20–22, 2001 [**188**].

After data-locality improving loop transformations have been applied, the cost of a data-dominated distributed system is determined mostly by:

- The inter-processor bandwidth, which is given by the maximum sum of **all** ingoing and outgoing dependency edges from each partition;
- The intra-processor bandwidth, which is given by **global** basic group level conflict analysis at the processor-level [**294**].

The conventional approach used by automatic parallelization compilers is to partition the iteration space in a Single Program Multiple Data (SPMD) fashion, as it appears after data-locality improving loop transformations. This often results in a high intra-processor bandwidth requirement by the principle of pushing the bottleneck one abstraction-level further down. To alleviate this situation where parallelization is performed regardless of processor-level constraints and processor-level optimizations are performed **only** locally we believe that *space–time* analysis of (partly) unrolled loop bodies can help in:

- Reducing the intra-processor bandwidth requirements with limited impact on data locality by remapping data accesses onto processors using the full Multiple Instruction Multiple Data (MIMD) capabilities of modern distributed systems;
- Discovering low-cost *space–time* computation *patterns*, that can serve as the basis for (very) low-cost distributed object architectures in platform-based design.

Using our proposed *pattern* analysis, we show (at least) an overall time $\times$ consumption (C $\times$ T) improvement above 100% compared to a custom shared memory approach on a small C/C++ platform-based design example.

1. A full custom design using shared memories

Imagine that you have at your disposal $11n/2$ memory cycles to set $3n$ terms of the following three straightforward series and only $2n$ terms of the following fourth alternating series.

$$s_1^{(i)} = f(i) \quad s_2^{(i)} = g(i) \quad s_3^{(i)} = h(i)$$

$$s_4^{(2i)} = s_1^{(3i)} \times s_3^{(3i)} \quad s_4^{(2i+1)} = s_1^{(3i+1)} \times s_3^{(3i+1)}$$

In order to tackle this problem, you will probably start by writing the following C/C++ code:

```
float S1[3*N],S2[3*N],S3[3*N],S4[2*N];
for (int i=0; i<3*N; i++) {
  S1[i]=f(i); S2[i]=g(i); S3[i]=h(i);
};
```

```
for (int i=0; i<N; i++) {
  S4[2*i]=S1[3*i]*S3[3*i]; S4[2*i]=S2[3*i+1]*S3[3*i+1];
};
```

In order to avoid loading back $s_1^{(3i)}, s_3^{(3i)}, s_2^{(3i+1)}$ and $s_3^{(3i+1)}$ and also to introduce an even number of memory writes in the internal loop body, you will probably merge and unroll the two original loop bodies:

```
float S1[3*N],S2[3*N],S3[3*N],S4[2*N];
for (int i=0; i<N/2; i++) {
  S1[6*i]=f(6*i); S2[6*i]=g(6*i); S3[6*i]=h(6*i);
  S4[4*i]=S1[6*i]*S3[6*i];
  S1[6*i+1]=f(6*i+1); S2[6*i+1]=g(6*i+1); S3[6*i+1]=h(6*i+1);
  S4[4*i+1]=S1[6*i+1]*S3[6*i+1];
  S1[6*i+2]=f(6*i+2); S2[6*i+2]=g(3*i+2); S3[6*i+2]=h(3*i+2);
  S1[6*i+3]=f(6*i+3); S2[6*i+3]=g(6*i+3); S3[6*i]=h(6*i);
  S4[4*i+2]=S1[6*i+3]*S3[6*i+3];
  S1[6*i+4]=f(6*i+4); S2[6*i+4]=g(6*i+4); S3[6*i+4]=h(6*i+4);
  S4[4*i+3]=S1[6*i+4]*S3[6*i+4];
  S1[6*i+5]=f(6*i+5); S2[6*i+5]=g(6*i+5); S3[6*i+5]=h(6*i+5);
};
```

This results in 22 memory writes per loop body. Given the (extremely) high degree of concurrency present in this loop body, you might (legitimately) think that it can be computed in 11 memory cycles using a two 1-port memory architecture. Unfortunately, such an ideal bandwidth cannot be met because arrays have to be mapped in one or the other 1-port memory. This means that only 8 of the 16 possible `Si[.]`–`Sj[.]` concurrent writes are allowed, leading to a practical optimum of **only** 12 memory cycles using two 1-port memories.

The non-trivial hardness of this ordering problem is confirmed by the Extended Space–Time Conflict Graph (ESTCG) solution [1] found by our generalized conflict-directed ordering (GCDO) algorithm [**175**] [**176**] in Figure 1. This (optimal) solution requires three 1-port memories for a memory cycle budget of 11 because the chromatic number of the basic groups of scalars conflict-graph (BG-CG) is 3, leading to the shared memory architecture and mapping of arrays presented in Figure 2.

2. A platform-based design using distributed memories

Imagine that you decide to further reduce the power consumption and also potentially increase the throughput of this system by mapping it onto a (partly) predefined platform [**39**] composed of maximally six 1-port memories, three computational units and a custom interconnect.

2.1. Single Program Multiple Data (SPMD) execution mode. The conventional approach used by automatic parallelization compilers is to partition the iteration space in a Single Program Multiple Data (SPMD) fashion, as it appears after data-locality improving loop transformations [**83**] [**43**] [**56**]. This technique solves the problem of concurrent accesses to data [**153**] **but** results in a replication of the shared memory architecture among distributed components, as presented in Figure 3.

2.2. Multiple Instruction Multiple Data (MIMD) execution mode. In order to address an MIMD execution mode, we propose to first decompose (or decouple [**76**] [**77**] or cluster) the ESTCG into seven local computation *patterns*, as shown in Figure 4. These *patterns* are indivisible sets of computation and data – *i.e.* small basic objects [**162**] [**287**] – that can be freely combined for example

1. As a reminder, in an Extended Space–Time Conflict Graph (ESTCG), time is the horizontal axis and memory accesses are on the vertical axis; resource conflicts are represented by (plain in Figure 1) undirected edges and dependencies are represented by directed (dotted in Figure 1) edges. For a formal description of the ESTCG model, please read [**191**].

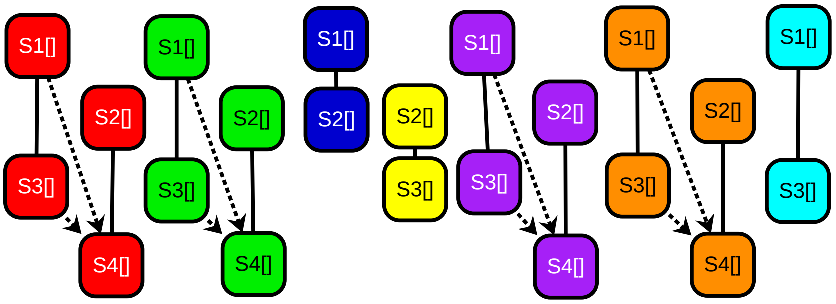


FIG. 1. *Extended Space–Time Conflict Graph (ESTCG) obtained after applying our GCDO(k) ordering algorithm to the unrolled C/C++ system specification involving 22 memory writes per loop body using a memory cycle budget of 11.*

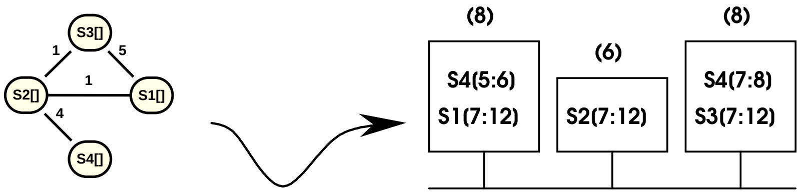


FIG. 2. *Global basic groups of scalars conflict-graph (BG-CG), shared memory architecture and mapping of arrays obtained after applying our GCDO(k) ordering algorithm to the unrolled C/C++ system specification involving 22 memory writes per loop body using a memory cycle budget of 11.*

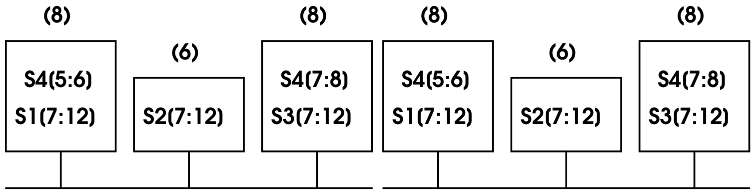


FIG. 3. *Memory architecture and mapping obtained after combining GCDO(k) to conventional SPMD parallelization techniques on our unrolled C/C++ system specification involving 44 memory writes per loop body using a memory cycle budget of 22.*

to further optimize the distributed MIMD architecture throughput. In our example, we immediately see that the high chromatic number (responsible for a three 1-port memories architecture) comes from keeping *patterns* (3) and (4) connected to the same shared memory. By unrolling three times the merged loop and by imposing the first distributed system to be responsible for *patterns* of type (3); the second distributed system to be responsible for *patterns* of type (4); and the last distributed system to be responsible for *patterns* of type (7), it is possible to reduce the distributed memory architecture from 2 × three 1-port memories to **only** 3 × two 1-port memories, as shown in Figure 5. Moreover, this mapping can be explicitly programmed by encapsulating our small basic object *patterns* into larger distributed objects for object parallelism compilation [**24**] [**23**] [**70**].

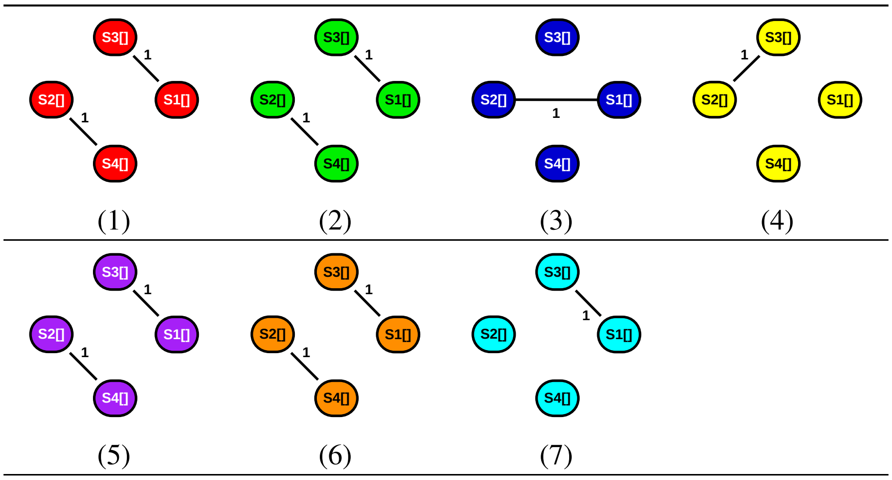


FIG. 4. *Decoupling the basic groups of scalars conflict-graph (BG-CG) into seven local and low-cost computation* patterns

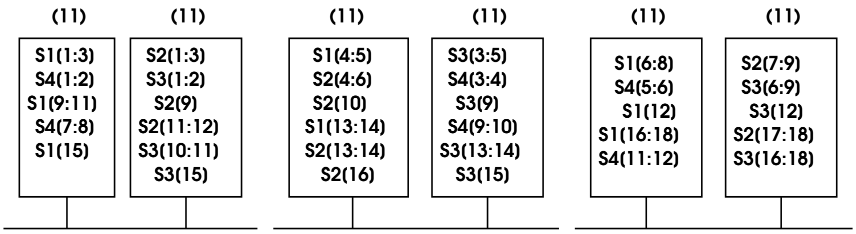


FIG. 5. *Memory architecture and mapping obtained after extending GCDO(k) with* pattern *analysis of the unrolled C/C++ system specification involving 66 memory writes per loop body using a memory cycle budget of 33.*

Component	Shared Custom	Distributed SPMD	Distributed MIMD	Gain
Storage	$2.16^2 + 12^2 = 656$	$2(2.8^2 + 6^2) = 328$	$6(22/3)^2 = 323$	**-50.76%**
Transfer	$(2.2 + \sqrt{3}).2 = 11.464$	$2(2.\sqrt{2} + \sqrt{3/2}) = 8.106$	$6.\sqrt{22/12} = 8.124$	**-29.13%**
Datapath	2	$1 + 1 = 2$	$1 + 1 + 1 = 3$	**+50.00%**
Throughput	$11n/2$	$11n/2$	$11n/3$	**+50.00%**

TAB. 1. *Relative power and time performance using a* pattern-*based distributed MIMD implementation instead of a conventional SPMD distribution of computation for exploiting a maximum number of six 1-port memories.*

2.3. Performance evaluation. Under the assumption that the technology is fixed, it is possible to derive high-level estimates from the physically accurate power models used for storage, interconnect and datapath – see *e.g.* [**38**] Chapter 4. As a first approximation, the power consumption of the storage is proportional to the product of its size $\times$ the number of accesses; the power consumption of a bus is proportional to its length $\times$ the number of accesses; and the power consumption of the datapath is proportional to the clock rate.

Using these high-level power models, we first show in Table 1 that, compared to the original shared memory solution (Shared Custom), a distributed SPMD version of our system (Distributed SPMD) consumes twice less power for storage and about 30% less power for the interconnect [**208**] while maintaining the same throughput on the specified platform already at half the clock rate (which has extra positive power effects which unfortunately are very difficult to model at a high level of abstraction).

In a second step, we also show that *pattern* analysis is a (very) smart technique for boosting by 50% the throughput of the Distributed SPMD implementation **without** affecting (or even slightly improving) its storage and transfer power characteristics. If an increase in throughput is not needed, *pattern* analysis can still be used to lower the clock rate of the platform for further power reduction (which again is very difficult to model at a high-level).

3. Conclusion

In this paper, we illustrate on a small but hard C/C++ example how *space–time* memory access *patterns* can be used as an alternative to custom shared memory or even distributed SPMD techniques for platform-based design. This leads to an estimated time $\times$ consumption (C $\times$ T) improvement above 100%[2] on a six 1-port memories, three processor and custom interconnect platform over the traditional custom shared memory approach, while also outperforming the SPMD parallelization paradigm in data transfer and storage related power consumption and overall throughput. These results are clearly in favour of the introduction of a novel object-oriented design technology based on the *space–time* memory access *patterns* paradigm for low-cost distributed multimedia platform design [**51**].

2. This C $\times$ T gain will increase when designing larger systems under hard real-time constraints, which will necessarily exhibit more opportunities for reducing the cost of mapping onto a larger distributed platform. Our small example is not at all an ideal case from a C $\times$ T improvement perspective.

Bibliographie

[1] A. AGARWAL, D. KRANTZ, AND V. NATARANJAN, *Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors*, IEEE Trans. on Parallel and Distributed Systems, 6 (1995), pp. 943–962.

[2] D. ALBERTS, G. CATTANEO, AND G. F. ITALIANO, *An empirical study of dynamic graph algorithms*, in Proc. of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM/SIAM, ed., New York, Philadelphia, 1996, pp. 192–201.

[3] D. ALBERTS, G.CATTANEO, G. F. ITALIANO, U. NANNI, AND C. D. ZAROLIAGIS, *A software library of dynamic graph algorithms*, in Proc. Algorithms and Experiments (ALEX'98) Conf., Trento, Italy, Feb. 1998, pp. 129–136.

[4] G. ALVERSON, P. BRIGGS, S. COATNEY, S. KAHAN, AND R. KORRY, *Tera hardware-software cooperation*, in Proceedings of Supercomputing'97, San Jose, CA, Nov. 1997, ACM SIGARCH and IEEE. Tera Computer Company.

[5] G. A. ALVERSON, W. G. GRISWOLD, C. LIN, D. NOTKIN, AND L. SNYDER, *Abstractions for portable, scalable parallel programming*, IEEE Transactions on Parallel and Distributed Systems, 9 (1998), pp. 71–86.

[6] S. AMARASINGHE, J. ANDERSON, M. LAM, AND C. TSENG, *The SUIF compiler for scalable parallel machines*, in Proc. 7th SIAM Conf. on Parallel Processing for Scientific Computing, 1995.

[7] P. R. AMESTOY, *Factorization of Large Sparse Matrices Based on a Multifrontal Approach in a Multiprocessor Environment*, Ph. D. thesis, CERFACS, Toulouse, France, 1991.

[8] C. ANCOURT, D. BARTHOU, C. GUETTIER, F. IRIGOIN, B. JEANNET, J. JOURDAN, AND J. MATTIOLI, *Automatic data mapping of signal processing applications*, in Proc. Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP'97), Swiss Federal Institute of Technology, Zurich, Switzerland, July 14–16, 1997.

[9] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, AND D. SORENSEN, LAPACK *: A portable linear algebra library for high-performance computers*, SIAM, Philadelphia, 1992. Also: LAPACK Working Note 20. Available at: `netlib2.cs.utk.edu:/lapack/lawns/lawn20.ps`.

[10] D. ANDRE AND J. R. KOZA, *A parallel implementation of genetic programming that achieves super-linear performance*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, H. R. Arabnia, ed., vol. III, Sunnyvale, 9-11 Aug. 1996, CSREA, pp. 1163–1174.

[11] M. ARIOLI, B. CODENOTTI, AND C. FASSINO, *The Padé method for computing the matrix exponential*, Tech. Rep. TR/PA/92/84, CERFACS, Toulouse, France, 1992.

[12] M. ARIOLI, I. S. DUFF, AND D. RUIZ, *Stopping criteria for iterative solvers*, Tech. Rep. TR/PA/91/58, CERFACS, Toulouse, France, 1991.

[13] C. ARM, J.-M. MASGONTY, AND C. PIGUET, *Double-latch clocking scheme for low-power I.P. cores*, in Proc. PATMOS'2000, Goettingen, Germany, September 13–15, 2000.

[14] T. BÄCK, *Self-adaptation*, in Handbook of Evolutionary Computation, T. Bäck, D. B. Fogel, and Z. Michalewicz, eds., Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997, pp. C7.1:1–15.

[15] C. BACKSTEIN, G. KRAETZSHMAR, AND J. SCHNEEBERGER, *Distributed plan maintenance for scheduling and execution*, in Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93), C. Bäckström and E. Sandewell, eds., Vadstena, Sweden, Dec. 1994, IOS Press (Amsterdam), pp. 74–86.

[16] U. BANARJEE, R. EIGENMANN, A. NICOLAU, AND D. PADUA, *Automatic program parallelization*, in Proc. of the IEEE, vol. 81, Feb. 1993.

[17] J. F. BARD, *A multiobjective methodology for selecting subsystem automation options*, Management Science, 32 (1986), pp. 1628–1641.

[18] R. BARTÀK, *Constraint programming: In pursuit of the Holy Grail*, in Proc. WDS'99, Prague, Tchekoslovakia, 1999.

[19] R. BATTITI AND M. PROTASI, *Reactive local search for the maximum clique problem*, Technical Report TR-95-052, International Computer Science Institute, Berkeley, CA, Sept. 1995.

[20] S. J. BEATY, *List scheduling: Alone, with foresight, and with lookahead*, in Conf. on Massively Parallel Computing Systems (MPCS): the Challenges of General-Purpose and Special-Purpose Computing, May 1994.

[21] M. BEKOOIJ, B. MESMAN, J. VAN MEERBERGEN, AND J. JESS, *Tightly coupled operation assignment and scheduling for VLIW processors with FACTS*, in 5th Int. Wsh. on Software and COmPilers for Embedded Systems (SCOPES), St Goar, Germany, March 20-22, 2001.

[22] N. BELDICEANU AND E. CONTEJEAN, *Introducing global constraints in* CHIP, Mathl. Comput. Modelling, 20 (1994), pp. 97–123.

[23] F. BODIN, *Directions in parallel programming HPF, shared virtual memory and object parallelism in pC++*, NASA contractor report CR-194943, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, 1994.

[24] F. BODIN, P. BECKMAN, D. GANNON, S. YANG, S. KESAVAN, A. MALONY, AND B. MOHR, *Implementing a parallel C++ runtime system for scalable parallel systems*, in Proc. Supercomputing Conference, Portland, Oregon, 1993.

[25] A. BRANDT, *Multi–level adaptive finite–element methods. I. Variational problems*, in Special Topics of Applied Mathematics, J. Frehse, D. Pallaschke, and U. Trotenberg, eds., North–Holland, Amsterdam, 1979, pp. 91–128.

[26] E. BROCKMEYER, J. D'EER, N. BUSÀ, F. CATTHOOR, P. LIPPENS, AND J. HUISKENS, *Code transformations for reduced data transfer and storage in low power realization of DAB synchro core*, in Proc. PATMOS'99, Kos, Greece, 1999.

[27] E. BROCKMEYER, A. VANDECAPPELLE, AND F. CATTHOOR, *Systematic cycle budget versus system power trade-off: a new perspective on system exploration of real-time data-dominated applications*, in Int. Symp. on Low Power Electronics and Design (ISLPED), Rapallo, Italy, July 26–27, 2000, pp. 137–142.

[28] E. BROCKMEYER, A. VANDECAPPELLE, S. WUYTACK, AND F. CATTHOOR, *Low power storage cycle budget distribution tool support for hierarchical graphs*, in 13th Int. Symp. on System Synthesis (ISSS), Madrid, Spain, Sept. 20–22, 2000, pp. 137–142.

[29] E. BROCKMEYER, S. WUYTACK, A. VANDECAPPELLE, AND F. CATTHOOR, *Low power storage cycle budget distribution for hierarchical graphs*, in 2nd ACM/IEEE Design and Test in Europe Conf. (DATE) Userforum, Paris, France, Apr. 1999, pp. 249–254.

[30] ———, *Low power storage for hierarchical graphs*, in Proc. 3rd ACM/IEEE Design and Test in Europe Conf. (DATE), Paris, France, Mar. 2000.

[31] P. BRUCKER, *Scheduling algorithms*, Springer, ISBN 3-540-64105-X, 2nd ed., 1998.

[32] CADENCE DESIGN SYSTEMS, *Cierto virtual component co-design (VCC) environment*, http://www.cadence.com/datasheets/vcc_environment.html, 2000.

[33] ———, *Cierto virtual component co-design (VCC) links to implementation*, http://www.cadence.com/datasheets/vcc_implementation.html, 2000.

[34] L. CARTER, B. SIMON, B. CALDER, L. CARTER, AND J. FERRANTE, *Predicated static single assignment*, in Proc. Int. Conference on Parallel Architectures and Compilation Techniques (PACT), Newport Beach, California, Oct. 12–16, 1999, IEEE Computer Society Press, pp. 245–255.

[35] F. CATTHOOR, E. BROCKMEYER, M. CUPAK, K. DANCKAERT, E. DE GREEF, J.-P. DIGUET, S. JANSSENS, C. KULKARNI, K. MASSELOS, T. OMNÈS, AND S. WUYTACK, *Global transfer and storage exploration for power and performance in multi–media applications on (parallel) programmable processors*, Internal Report, IMEC-DESICS, Leuven, Belgium, Feb. 2000.

[36] F. CATTHOOR, K. DANCKAERT, C. KULKARNI, AND T. J.-F. OMNÈS, *Data transfer and storage architecture issues and exploration in modern DSPs*, in Programmable Digital Signal Processors: Architecture, Programming, and Applications, Y. H. Yu, ed., Marcel Dekker, Inc., New York, USA, Dec. 2000.

[37] F. CATTHOOR AND T. J.-F. OMNÈS, Acropolis*: global storage and transfer exploration for power and performance in multi-media applications on (parallel) programmable processors*, in INRIA-CRI-PRiSM Project A3 Seminar, C. Eisenbeis, F. Irigoin, and P. Feautrier, eds., INRIA, Rocquencourt, France, January 25, 1999.

[38] F. CATTHOOR, S. WUYTACK, E. DE GREEF, F. BALASA, L. NACHTERGAELE, AND A. VANDECAPPELLE, *Custom Memory Management Methodology - Exploration of Memory Organisation for Embedded Multimedia System Design*, Kluwer Academic Publishers, ISBN 3-540-64105-X, 1998.

[39] H. CHANG, L. COOKE, M. HUNT, G. MARTIN, A. MCNELLY, AND L. TODD, *Surviving the SOC Revolution : A Guide to Platform-Based Design*, Kluwer Academic Publishers, ISBN 0-7923-8679-5, Nov. 1999.

[40] F. CHAROT, P. FRISON, E. GAUTRIN, D. LAVENIER, P. QUINTON, AND C. WAGNER, *From equations to hardware. Towards the systematic mapping of algorithms onto parallel architectures*, Int. Journal of Pattern Recognition and A.I., 8 (1994).

[41] CHEN, YANG, AND OUHYOUNG, *JavaGL: A 3D Graphics Library in Java for Internet Browsers*, IEEE Transactions on Comsumer Electronics, 43 (1997), pp. 271–278.

[42] J. S. CHIPMAN, *Linear restrictions, rank reduction, and biased estimation in linear regression*, Linear Algebra and its Applications, 289 (1999), pp. 55–74.

[43] F. COELHO, *Génération de code faible consommation pour stencils*, in Workshop on Compilation and Automatic Parallelisation, no. 1999-53 in Research Report, INRIA, St. Narbor, France, November 18–20, 1999.

[44] R. COMERFORD AND G. WATSON, *Memory catches up*, IEEE Spectrum, (1992), pp. 34–57.

[45] J. O. COPLIEN, *Advanced C++ programming styles and idioms*, Addison-Wesley Publishing Company, ISBN 0-201-54855-0, 1994.

[46] J. CORBAL, R. ESPASA, AND M. VALERO, *Exploiting a new level of DLP in multimedia applications*, in Proc. MICRO'99, 1999, pp. 72–81.

[47] J. CORBAL, R. ESPASA, AND M. VALERO, *MOM: a matrix SIMD instruction set architecture for multimedia applications*, in Proc. SuperComputing SC'99, Portland, Oregon, November 11–18, 1999, ACM Press and IEEE Computer Society Press.

[48] H. CORPORAAL, *Microprocessor Architectures – from VLIW to TTA*, John Wiley & Sons, 1998.

[49] COSYTEC, *Application development with the* CHIP *system*, Technical Report, Cosytec S.A., France, 1997.

[50] O. COUDERT, *Exact coloring for real-life graphs is easy*, in Proc. 34th ACM/IEEE Design Automation Conf. (DAC), 1997.

[51] G. COULSON AND S. BAICHOO, *Experiences in implementing a distributed object platform for multimedia applications*, Software Practice and Experience, 30 (2000), pp. 663–683.

[52] COWARE, *Coware N2C design system overview*, `http://www.coware.com/`, 2000.

[53] J. CRAWFORD, *An approach to resource constrained project scheduling*, in Proc. Artificial Intelligence and Manufacturing Research Planning Wsh, 1996.

[54] J. L. DA SILVA, C. YKMAN-COUVREUR, M. MIRANDA, K. CROES, S. WUYTACK, G. DE JONG, F. CATTHOOR, D. VERKEST, P. SIX, AND H. DE MAN, *Efficient system exploration and synthesis of applications with dynamic data storage and intensive data transfer*, in Proc. 35th ACM/IEEE Design Automation Conf. (DAC), 1998, pp. 76–81.

[55] K. DANCKAERT, F. CATTHOOR, AND H. DE MAN, *System-level memory management for weakly parallel image processing*, in Proc. Euro-Par Conf., vol. 1124 of Lecture Notes in Computer Science, Springer Verlag, Lyon, France, Aug. 1996, pp. 217–225.

[56] ———, *A loop transformation approach for combined parallelization and data transfer and storage optimization*, in Proc. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA), vol. V, Las Vegas, NV, June 2000, pp. 2591–2597.

[57] K. DANCKAERT, K. MASSELOS, F. CATTHOOR, H. DE MAN, AND C. GOUTIS, *Strategy for power efficient design of parallel systems*, IEEE Trans. on VLSI Systems, 7 (1999), pp. 258–265.

[58] A. DARTE AND Y. ROBERT, *Affine-by-statement scheduling of uniform loop nests over parametric domains*, Internal report 92-16, ENSL/IMAG, Lyon, France, Apr. 1992.

[59] ———, *Scheduling uniform loop nests*, Internal report 92-10, ENSL/IMAG, Lyon, France, Feb. 1992.

[60] A. DARTE, Y. ROBERT, AND F. VIVIEN, *Scheduling and Automatic Parallelisation*, Birkhauser, 2000.

[61] L. DAVIS, *Handbook of Genetic Algorithms*, Thomson Computer Press, 1996.

[62] E. DE GREEF, *Pruning in a High-Level Memory Management Context*, Internal Report, IMEC, VSDM, Leuven, Belgium, Dec. 1993.

[63] E. DE GREEF, *Storage size reduction for multimedia applications*, Ph.D thesis, ESAT, EE Department, Katholieke Universiteit, Leuven, Belgium, 1998.

[64] E. DE GREEF, M. EYCKMANS, AND S. WUYTACK, *ANSI-C usage restrictions for ATOMIUM applications*, Internal Report Version 1.0, IMEC-VSDM, Leuven, Belgium, May 1998.

[65] A. DE HON, *Reconfigurable Architectures for General-Purpose Computing*, Ph.D thesis, MIT, Artificial Intelligence Laboratory, Sept. 1996.

[66] H. DE MAN, *System Design Challenges in the Post-PC Era*, in Phil Kaufmann nominee talk, 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 2000.

[67] K. DENOLF, P. VOS, J. BORMANS, AND I. BOLSENS, *Cost-efficient C-Level Design of an MPEG-4 Video Decoder*, in Proc. Int. Wsh. on Power and Timing Modeling, Optimization and Simulation, Goettingen, Germany, Sept. 13-15, 2000.

[68] K. DIEFENDORFF AND P. K. DUBEY, *How multimedia workloads will change processor design*, in Int. Symp. on Microarchitecture (MICRO-30), Sept. 1997.

[69] T. G. DIETTERICH, *The divide-and-conquer manifesto*, in Algorithmic Learning Theory, 11th International Conference, ALT 2000, Sydney Australia, December 2000, Proceedings, vol. 1968 of Lecture Notes in Artificial Intelligence, Springer, Berlin, 2000, pp. 13–26.

[70] S. DIWAN AND D. GANNON, *A capabilities based communication model for high-performance distributed applications: The open HPC++ approach*, in Proc. Int. Parallel Processing Symposium (IPPS) and Symposium on Parallel and Distributed (SPDP), Apr. 1999.

[71] M. DUBOIS, J. SKEPPSTEDT, AND P. STRENSTROM, *Essential misses and data traffic in coherence protocols*, Journal of Parallel and Distributed Computing, 29 (1995), pp. 108–125.

[72] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, UK, 1986.

[73] P. ELLERVEE, M. MIRANDA, F. CATTHOOR, AND A. HEMANI, *System-level data format exploration for dynamically allocated data structures*, in Proc. 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 2000, pp. 556–559.

[74] D. EPPSTEIN, Z. GALIL, AND G. ITALIANO, *Dynamic graph algorithms (chapter 8)*, in CRC Handbook of Algorithms and Theory of Computation, 1999.

[75] G. ESAKKIMUTHU, N. VIJAYKRISHNAN, M. KANDEMIR, AND M. J. IRWIN, *Memory system energy: Influence of hardware-software optimizations*, in Int. Symp. on Low Power Electronics and Design (ISLPED), Rapallo, Italy, July 26–27, 2000, pp. 244–246.

[76] R. ESPASA AND M. VALERO, *Decoupled vector architectures*, in Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA '96), Washington - Brussels - Tokyo, Feb. 1996, IEEE, pp. 281–290.

[77] R. ESPASA AND M. VALERO, *A simulation study of decoupled vector architectures*, The Journal of Supercomputing, 14 (1999), pp. 124–152.

[78] ETNUS, TimeScan 3.0 *Multiprocessor Event Analyser*, Product Overview, Etnus LLC, Framingham, MA, 2000.

[79] S. ETO ET AL., *A 1-GB SDRAM with ground-level precharged bit-line and nonboosted 2.1V word-line*, IEEE J. Solid-state Circuits, 33 (1998), pp. 1697–1702.

[80] R. EVANS AND P. FRANZON, *Energy consumption modeling and optimization for SRAMs*, IEEE J. of Solid-State Circuits, 30 (1995), pp. 571–579.

[81] G. EVEN, J. S. NAOR, S. RAO, AND B. SCHIEBER, *Fast approximate graph partitioning algorithms*, in Proc. 8th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, Jan. 1997, pp. 639–648.

[82] D. G. FEITELSON, *A Survey of Scheduling in Multiprogrammed Parallel Systems*, Research Report RC 19790 (87657), IBM T.J. Watson Research Center, Yorktown Heights, NY, Feb. 1995.

[83] A. FRABOULET, G. HUARD, AND A. MIGNOTTE, *Loop alignment for memory accesses optimization*, in Proc. 12th ACM/IEEE International Symposium on System-Level Synthesis (ISSS), 1999.

[84] T. FRANZETTI, *Survey of scheduling algorithms for the design of embedded multimedia systems*, Master thesis, Computer Science and Applied Mathematics Department of INP-ENSEEIHT, Toulouse, France, June 2000.

[85] T. FRANZETTI, D. VANDERSTRAETEN, AND T. OMNÈS, *Partial exploration techniques for minimizing memory bandwidth in high-throughput telecom networks and speech embedded systems*, Joint technical report, IMEC-DESICS and Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, Mar. 2000.

[86] K. GALLIVAN, E. GRIMME, AND P. VAN DOOREN, *A rational Lanczos algorithm for model reduction*, Numerical Algorithms, 12 (1996), pp. 33–63.

[87] E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Elements of reusable object-oriented software*, Addison-Wesley Professional Computing Series, ISBN 0-201-63361-2, 1994.

[88] M. R. GAREY AND D. S. JOHNSON, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, New York, NY, 2nd ed., 1983.

[89] T. GAUTIER ET AL., *Le projet Cairn: vers la conception d'architectures à partir de* Signal *et* Alpha, Collection technique et scientifique des télécommunications, (1997).

[90] A. GERASOULIS, J. JIAO, AND T. YANG, *A multistage approach to scheduling task graphs*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 22 (1995), pp. 81–103.

[91] S. GHOSCH, M. MARTONOSI, AND S. MALIK, *Cache miss equations : a compiler framework for analyzing and tuning memory behaviour*, ACM Trans. on Programmable Languages and Systems, 21 (1999), pp. 702–746.

[92] R. GINOSAR ET AL., *Adaptive synchronization*, in Proc. AINT 2000, Delft, Netherlands, July 18–20 2000.

[93] R. GONZALES AND M. HOROWITZ, *Energy dissipation in general-purpose microprocessors*, IEEE J. Solid-state Circuits, 31 (1996), pp. 1277–1283.

[94] R. L. GRAHAM, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., (1970), pp. 291–308.

[95] D. GRIGIONI, M. IOFFREDA, U. NANNI, AND G. PASQUALONE, *Experimental analysis of dynamic algorithms for the single shortest path problems*, in Proc. of Wsh. on Algorithm Engineering (WAE'97), Sept. 1997.

[96] A. GUPTA, *Fast and effective algorithms for graph partitioning and sparse-matrix ordering*, IBM J. of Research and Development, 41 (1997).

[97] M. GUPTA, E. SCHONBERG, AND H. SRINIVASAN, *A unified framework for optimizing communication in data-parallel programs*, IEEE Trans. on Parallel and Distributed Systems, 7 (1996), pp. 689–704.

[98] R. GUPTA, M. L. SOFFA, AND D. OMBRES, *Efficient register allocation via coloring using clique separators*, ACM Transactions on Programming Languages and Systems, 16 (1994), pp. 370–386.

[99] S. GUPTA, M. MIRANDA, F. CATTHOOR, AND R. GUPTA, *Analysis of high-level address code transformations for programmable processors*, in Proc. 3rd ACM/IEEE Design and Test in Europe Conf. (DATE), Paris, France, Mar. 2000.

[100] V. GUTNIK AND A. CHANDRAKASAN ET AL., *Active GHz clock network using distributed PLLs*, in Proc. IEEE Int. Solid-state Circuits Conf. (ISSCC), San Francisco, USA, Feb. 2000.

[101] D. L. HARRAR, *Orderings, multicoloring, and consistently ordered matrices*, SIAM J. on Matrix Analysis and Applications, 14 (1993), pp. 259–278.

[102] P. HÄTTENSCHWILER, M. MORESINO, AND A. SCHROFF, *Rapid prototyping of decision support systems*, in Engineering of Intelligent Systems EIS'98, ICSC, 1998, pp. 295–302. ISBN 3-906454-12-6.

[103] J. H. HENNING, *SPEC CPU2000: Measuring CPU Performance in the New Millennium*, IEEE Computer Magazine, 40 (2000), pp. 28–35.

[104] M. R. HEZINGER AND V. KING, *Randomized dynamic graph algorithms with polylogarithmic time per operation*, in Proceedings of the 27th Symposium on Theory of Computing, 1995, pp. 519–527.

[105] D. S. HOCHBAUM, *Approximation Algorithms for NP-hard problems*, PWS Publishing Company, ISBN 0-534-94968-1, 1997.

[106] ———, *Hardness of approximations: comments on the reduction of the maximum clique problem to that of the chromatic number*, in Approximation Algorithms for NP-hard problems, PWS Publishing Company, ISBN 0-534-94968-1, 1997, p. 430.

[107] H. HONG, A. NEUBACHER, AND W. SCHREINER, *The Design of the* SACLIB/PACLIB *Kernels*, in DISCO 93 — International Symposium on Design and Implementation of Symbolic Computation System, A. Miola, ed., vol. 722 of Lecture Notes in Computer Science, Gmunden, Austria, September 15–17, 1993, Springer, Berlin, pp. 288–302. Also: Technical Report 93-30, RISC-Linz, Johannes Kepler University, Linz, Austria, June 1993. Source code at: `ftp.risc.uni-linz.ac.at/saclib`.

[108] ILOG, *Optimization technology white paper: a comparative study of optimization techniques*, White Paper, ILOG S.A., Gentilly, France, 1997.

[109] IMPROV SYSTEMS INC., *Improv systems overview*, White Paper, USA, 2000. Available at: `http://www.improvsys.com`.

[110] P. ITALIANO, P. CATTANEO, U. NANNI, G. PASQUALONE, C. ZAROLIAGIS, AND D. ALBERTS, LEDA *extension package for dynamic graph algorithms*, Technical Report, Department of Computer Science, University of Halle, Germany, Mar. 1998.

[111] K. ITOH, *Low voltage memory design*, in IEEE Int. Symp. on Low Power Design (ISLPD), Tutorial on Low Voltage Technologies and Circuits, Aug. 1997.

[112] K. ITOH, Y. NAKAGOME, S. KIMURA, AND T. WATANABE, *Limitations and challenges of multi-gigabit DRAM chip design*, IEEE J. of Solid-state Circuits, 26 (1997).

[113] B. JAWERTH AND W. SWELDENS, *Wavelet multiresolution analyses adapted for the fast solution of boundary value ordinary differential equations*, in Sixth Copper Mountain Conference on Multigrid Methods, N. D. Melson, T. A. Manteuffel, and S. F. McCormick, eds., vol. CP 3224, Hampton, VA, 1993, NASA, pp. 259–273.

[114] A. JERRAYA, *Hardware/software codesign*, in Summer Course, Orebro, Sweden, August 14–16, 2000.

[115] D. S. JOHNSON, C. R. ARAGON, L. A. MCGEOCH, AND C. SHEVON, *Optimization by simulated annealing: An experimental evaluation Part I – graph partitioning*, Opns. Res., 37 (1989), pp. 865–892.

[116] R. JOLLY, *A 9ns 1.4GB/s 17-ported CMOS register file*, IEEE J. of Solid-state Circuits, 26 (1991), pp. 1407–1412.

[117] J. JONSSON AND J. VASELL, *On objective function selection in list scheduling algorithms for digital signal processing applications*, in Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 1997, pp. 667–670.

[118] A. JUDITSKY, Q. ZHANG, B. DELYON, P.-Y. GLORENNEC, AND A. BENVENISTE, *Wavelets in identification: wavelets, splines, neurons, fuzzies: how good for identification?*, INRIA Report 2315, Sept. 1994.

[119] M. KANDEMIR, J. RAMANUJAM, AND A. CHOUDHARY, *Improving cache locality by a combination of loop and data transformations*, IEEE Trans. on Computers, 48 (1999), pp. 159–167.

[120] G. KARYPIS, R. AGGARWAL, V. KUMAR, AND S. SHEKHAR, *Multilevel hypergraph partitioning: applications in VLSI domain*, in Proc. 34th ACM/IEEE Design Automation Conf. (DAC), 1997.

[121] G. KARYPIS, E.-H. S. HAN, AND V. KUMAR, *Multilevel refinement for hierarchical clustering*, Technical Report number 99-020, University of Minnesotta, 1999.

[122] G. KARYPIS AND V. KUMAR, METIS*: Unstructured graph partitioning and sparse matrix ordering system. version 2.0*, Technical Report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, Aug. 1995.

[123] G. KARYPIS AND V. KUMAR, hMETIS*, multilevel k-way hypergraph partitioning*, Technical Report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, 1998.

[124] B. W. KERNIGHAN AND S. LIN, *An effective heuristic procedure for partitioning graphs*, The Bell System Technical Journal, (1970), pp. 291–308.

[125] B. W. KERNIGHAN AND D. M. RITCHIE, *The C programming language*, Prentice-Hall, ISBN 0-13-110163-3, 1978.

[126] C. KIM ET AL., *A 64 Mbit, 640 MB/s bidirectional data-strobed double data rate SDRAM with a 40mW DLL for a 256 MB memory system*, IEEE J. Solid-state Circuits, 33 (1998), pp. 1703–1710.

[127] T. KIRIHATA ET AL., *A 220 mm^2, four- and eight-bank, 256 Mb SDRAM with single-sided stiched WL architecture*, IEEE J. Solid-state Circuits, 33 (1998), pp. 1711–1719.

[128] P. G. KJELDSBERG, F. CATTHOOR, AND E. J. AAS, *Dependency size estimation with a partially fixed execution ordering*, in Proc. Int. Conf. on Computer Aided Design (ICCAD), San Jose, USA, Nov. 2000.

[129] U. KO, P. T. BALSARA, AND A. NANDA, *Energy optimization of multi-level processor cache architectures*, in Proc. Int. Conf. on Low Power Electronics and Design (ICLPED), Dana Point, CA, 1995, pp. 63–68.

[130] KOVALYOV AND SHAFRANSKY, *Uniform machine scheduling of unit-time jobs subject to resource constraints*, Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science (DAMATH), 84 (1998).

[131] S. KUGE, F. MORISHITA, T. SURUDA, S. TOMISHIMA, N. TSUKUDE, T. YAMAGATA, AND K. ARIMOTO, *SOI-DRAM circuit technologies for low power high speed multigiga scale memories*, IEEE J. Solid-state Circuits, 31 (1996), pp. 586–596.

[132] C. KULKARNI, *Caching strategies for voice coder application on embedded processors*, Master thesis, Dept. of Electrical Engineering, Katholieke Universiteit, Leuven, Belgium, June 1997.

[133] ———, *Cache optimization for multimedia applications*, PhD thesis, ESAT, EE Department, Katholieke Universiteit, Leuven, Belgium, Feb. 2001.

[134] C. KULKARNI, F. CATTHOOR, AND H. DE MAN, *Optimizing graphics applications : A data transfer and storage exploration perspective*, in Wsh. on Media Processors and DSP's, Int. Symp. on Microarchitecture (MICRO-32), Haifa, Israel, Nov. 1999.

[135] ———, *Advanced data layout optimization for multimedia applications*, in Proc. Wsh. on Parallel and Distributed Computing in Image, Video and Multimedia Processing (PDIVM) of IPDPS 2000, vol. 1800 of Lecture Notes in Computer Science, Springer Verlag, May 2000, pp. 186–193.

[136] C. KULKARNI, K. DANCKAERT, F. CATTHOOR, AND M. GUPTA, *Interaction between data parallel compilation and data transfer and storage cost minimization for multimedia applications*, in Int. Conf. on Parallel Proc. in Europe (EuroPar'99), Toulouse, France, Sept. 1999, pp. 668–676.

[137] C. KULKARNI, C. GHEZ, M. MIRANDA, F. CATTHOOR, AND H. DE MAN, *Cache conscious data layout organization for embedded multimedia applications*, in Proc. 4th ACM/IEEE Design Automation and Test in Europe Conference (DATE), Munich, Germany, Mar. 2001.

[138] D. KULKARNI AND M. STUMM, *Linear loop transformations in optimizing compilers for parallel machines*, The Australian Computer Journal, (1995), pp. 41–50.

[139] V. KUMAR, *Algorithms for constraint satisfaction problems: A survey*, AI Magazine, 13 (1992), pp. 32–44.

[140] G. LAFRUIT, L. NACHTERGAELE, J. BORMANS, M. ENGELS, AND I. BOLSENS, *Optimal memory organization for scalable texture codecs in MPEG-4*, Internal Report, IMEC-DESICS, Leuven, Belgium, Mar. 2000.

[141] G. LAFRUIT, L. NACHTERGAELE, K. DENOLF, AND J. BORMANS, *3D computational graceful degradation*, in ISCAS 2000, Workshop and Exhibition on MPEG-4, May 28–31, 2000, pp. III–547–III–550.

[142] G. LAFRUIT AND W. RAEMDONC, *High-level Content-related Complexity Parameters for Real-time 3D Rendering*, in Proc. SIGGRAPH, 2001.

[143] M. LAM, E. ROTHBERG, AND M. WOLF, *The cache performance and optimization of blocked algorithms*, in Proc. Int. Conf. on Architectural Support for Programming Languages and Operating Systems, Santa Clara, CA, 1991, pp. 63–74.

[144] F. LaMonica, *Streaming media*, Linux Journal, 81 (2001), pp. 108, 110, 112–114.

[145] C. L. Lawson and R. J. Hanson, *Solving Least-Square Problems*, Classics in Applied Mathematics, SIAM, Philadelphia, USA, 1995.

[146] G. Lawton, *Storage technology takes the center stage*, IEEE Computer Magazine, 32 (1999), pp. 10–13.

[147] C. Lengauer, *Loop parallelization in the polytope model*, in Proc. 4th Int. Conf. on Concurrency Theory (CONCUR), Hildesheim, Germany, Aug. 1993.

[148] M. Lenz, E. Auriol, and M. Manago, *Diagnosis and decision support*, Lecture Notes in Computer Science, 1400 (1998), pp. 51–??

[149] D. Li and Y. Y. Haimes, *Hierarchical generating method for large-scale multiobjective systems*, J. Optimization Theory and Applications, 21 (1987), pp. 303–333.

[150] P. Lippens, J. van Meerbergen, W. Verhaegh, and A. van der Werf, *Allocation of multi-port memories for hierarchical data streams*, in Proc. IEEE Int. Conf. Comp. Aided Design (ICCAD), Santa Clara, USA, Nov. 1993, pp. 728–735.

[151] P. E. R. Lippens, J. L. van Meerbergen, A. van der Werf, W. F. J. Verhaegh, M. B. T., J. O. Huisken, and O. P. McArdle, PHIDEO *: a silicon compiler for high-speed algorithms*, in Proc. of Eur. Conf. on Design Automation, 1991, pp. 436–441.

[152] Z. Liu, *Worst-case analysis of scheduling heuristics of parallel systems*, Parallel Computing, 24 (1998), pp. 863–891.

[153] S. Lumetta and D. Culler, *Managing concurrent access for shared memory active messages*, in Proc. Int. Parallel Processing Symp. (IPPS) and Symp. Parallel and Distributed (SPDP), Apr. 1998, pp. 272–279.

[154] C. Lund and M. Yannakis, *On the hardness of approximating minimization problems*, Journal of the ACM, 41 (1994), pp. 960–981.

[155] I. J. Lustig and J.-F. Puget, *Program != Program: Constraint Programming and its Relationship to Mathematical Programming*, Technical Report, ILOG S.A., Gentilly, France, 1997.

[156] K. Marriott, *Abstract interpretation: A theory of approximate computation*, Lecture Notes in Computer Science, 1302 (1997), pp. 367–??

[157] J. B. Mazzola and A. W. Neebe, *Resource-constrained assignment scheduling*, Operations Research, 34 (1986), pp. 560–572.

[158] K. Mehlhorn and S. Naher, LEDA*: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, ISBN 0521563291, 1999.

[159] T. H. Meng, B. Gordon, E. Tsern, and A. Hung, *Portable video–on–demand in wireless communication*, in special issue on “Low-power electronics” of the Proc. of the IEEE, vol. 83, Apr. 1995, pp. 659–680.

[160] P. Messina, *ASCI hits its stride*, in Proceedings from the Conference on High Speed Computing: The End of the Beginning, K. P. Hirons, M. Vigil, and R. Carlson, eds., Los Alamos, NM, Nov. 1999, LANL, pp. 69–91. LA-13637-C.

[161] H. W. Meuer and E. Strohmaier, *The changing faces of supercomputing*, Scientific computing world, 55 (2000), pp. 12–15.

[162] B. Meyer, *Object oriented software construction*, in C. A. R. Hoare series editor, Series in Computer Science, Prentice Hall Int’l Inc., ISBN 0-13-629049-3, Englewood cliffs, NJ, 1988.

[163] S. Meyers, *Effective C++ : 50 specific ways to improve your programs and designs*, Addison-Wesley Reading (Mass.), 2nd ed., 1998.

[164] ———, *More effective C++: 35 new ways to improve your programs and designs*, Addison-Wesley Reading (Mass.), 1st ed., 1998.

[165] T. Miller and C. Zaroliagis, *A first experimental study of a dynamic transitive closure algorithm*, in Proc. of Wsh. on Algorithm Engineering (WAE’97), Sept. 1997.

[166] A. Monsifrot and F. Bodin, *Computer Aided Hand Tuning (CAHT): Applying case-based reasoning to performance tuning*, Technical Report, IRISA, Rennes, France, February 13, 2001.

[167] J. Montusiewicz and A. Osyczka, *A decomposition strategy for multicriteria optimization with application to machine tool design*, Engineering Costs and Production Economics, 20 (1990), pp. 191–202.

[168] T. Mudge, *Invited talk*, in 7th ACM/IEEE Int. High Performance Computing (HiPC) Conf., vol. 1970 of Lecture Notes in Computer Science, Springer Verlag, Bangalore, India, 2000.

[169] D. R. Musser, A. Saini, and A. Stepanov, *STL tutorial and reference guide: C++ programming with the standard template library*, Addison-Wesley Reading (Mass.), 1996.

[170] L. Nachtergaele, F. Catthoor, F. Balasa, F. M. Franssen, E. de Greef, H. Samson, and H. de Man, *Optimization of memory organization and partitioning for decreased size and power in video and image processing systems*, in IEEE Int. Workshop on Memory Technology, May 1994, pp. 82–87.

[171] K. S. NARENDA AND K. PARATHASARATHY, *Learning automata approach to hierarchical multiobjective analysis*, IEEE Trans. Systems, Man, Cybernetics, 21 (1991), pp. 263–273.

[172] M. NEERACHER AND R. RUHL, *Automatic parallelization of LINPACK routines on distributed memory parallel processors*, in Proc. IEEE Int. Parallel Proc. Symp. (IPPS), Newport Beach, CA, Apr. 1993.

[173] G. L. NEMHAUSER AND L. A. WOLSEY, *Integer and Combinatorial Optimizations*, J. Wiley & sons, New York, USA, 1988.

[174] S. M. NOWICK, M. B. JOSEPPHS, AND C. H. V. BERKEL, *Special Issue of the Proceedings of the IEEE on "Asynchronous Circuits and Systems"*, Feb. 1999.

[175] T. OMNÈS, T. FRANZETTI, AND F. CATTHOOR, *Interactive algorithms for minimizing memory bandwidth in high throughput telecom and multimedia*, in Proc. 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 2000, pp. 328–331.

[176] ———, *Multi-dimensional selection techniques for minimizing memory bandwidth in high-throughput embedded systems*, in Proc. 7th ACM/IEEE Int. High Performance Computing (HiPC) Conf., vol. 1970 of Lecture Notes in Computer Science, Springer Verlag, Bangalore, India, 2000, pp. 323–334.

[177] T. OMNÈS, T. FRANZETTI, AND F. CATTHOOR, *A dynamic algorithm for minimizing the memory bandwidth of embedded systems*, Technique et Science Informatiques (TSI), 20 (2001), pp. 1–25.

[178] T. OMNÈS, A. VANDECAPPELLE, E. BROCKMEYER, C. KULKARNI, AND F. CATTHOOR, Acropolis *α-2000 project planning (1999–2000)*, Internal Report, IMEC-VSDM, Leuven, Belgium, Dec. 1998.

[179] T. J.-F. OMNÈS, *Generalized dynamic multi-dimensional and multi-level - power-oriented scheduling and mapping:* GDMDML-POSM *is fast ... but is it optimal?*, Presentation to KUL - Scientific Computing Group, Design Technology for Integrated Information and Communication Systems (DESICS) division of IMEC, Leuven, Belgium, Feb. 1999.

[180] ———, *Interactive algorithms for low-cost scheduling in* Acropolis, in Workshop on Compilation and Automatic Parallelisation, no. 1999-53 in Research Report, INRIA-ICPS-ENSL, St. Narbor, France, November 18–20, 1999, pp. 29–31.

[181] ———, *The multimedia compilation project* Acropolis, in Instruction Level Parallelism and Parallelizing Compilation, Dagstuhl-Seminar-Report 237 (99161), D. Arvind, K. Ebcioglu, C. Lengauer, K. Pingali, and R. Schreiber, eds., Dagstuhl International Conference Center, Germany, April 18–23, 1999, p. 15.

[182] ———, *Parameter tuning in generalized dynamic multi-dimensional and multi-level - power-oriented scheduling and mapping:* GDMDML-POSM, IMEC-OPTIMUS Meeting, Design Technology for Integrated Information and Communication Systems (DESICS) division of IMEC, Leuven, Belgium, November 4, 1999.

[183] ———, Acropolis *α-2000 - a parallel multimedia software precompiler*, Inter-Group Meeting Slides, Design Technology for Integrated Information and Communication Systems (DESICS) division of IMEC, Leuven, Belgium, March 26, 1999.

[184] ———, *Analysis of* Phideo *and* Acropolis, in Runtime exploitation of reconfigurable platforms for real-time applications Workgroup, M. Boonen, M. Heijligers, P. Lippens, J. van Meerbergen, T. Omnès, P. Schaumont, P. Vanderbroek, and D. Verkest, eds., IMEC-DESICS and Philips-NATLAB, Leuven, Belgium and Eindhoven, The Netherlands, July 2000.

[185] ———, *A survey of Constraint-Logic Programming (CLP)*, IMEC Internal Report, Design Technology for Integrated Information and Communication Systems (DESICS) division of IMEC, Leuven, Belgium, Feb. 2000.

[186] ———, *Theoretical drivers for motivating, tuning and identifying new research directions for Conflict Directed Ordering (CDO)*, IMEC Internal Report, Design Technology for Integrated Information and Communication Systems (DESICS) division of IMEC, Leuven, Belgium, Feb. 2000.

[187] ———, Acropolis*: a System-Level Precompiler for Data Transfer and Storage Exploration (DTSE) in High-Throughput Embedded System Design*, Ph.D dissertation in Real-time Systems, Automation and Robotics, Centre de Recherche en Informatique de l'Ecole Nationale Supérieure des Mines, Paris, France, May 2001.

[188] ———, *Space–time memory access* patterns*: a step towards the object-oriented design of low-cost distributed platforms*, in 5th Int. Wsh. on Software and COmPilers for Embedded Systems (SCOPES), St Goar, Germany, March 20-22, 2001.

[189] T. J.-F. OMNÈS, P. R. AMESTOY, M. J. DAYDÉ, AND I. S. DUFF, *Pipelining the* **LU**-*factorization of full linear systems on memory shared multiprocessors*, Internal report, CERFACS, Parallel Algorithms Group, Toulouse, France, Mar. 1993.

[190] T. J.-F. OMNÈS, E. BROCKMEYER, C. KULKARNI, AND K. DANCKAERT, *Low power software for system-on-a-chips (SoCs)*, in 4th ACM/IEEE Design and Test in Europe Conf. (DATE), special session on low-power System-on-a-Chips (SoCs), M. Renaudin, T. J.-F. Omnès, and C. Piguet, eds., Munich, Germany, Mar. 2001.

[191] T. J.-F. OMNÈS AND F. CATTHOOR, *A unified data-flow and resource conflict detection model for higher-quality data access modeling in the space–time domain*, in submitted to EuroPar'2001, Manchester, UK, Sept. 2001.

[192] T. J.-F. OMNÈS, F. CATTHOOR, AND S. RAJOPADHYE, *Multi-level conflict-based scheduling and mapping: application to embedded multi-media system design*, Confidential Ph.D topic definition, IMEC-IRISA, Jan. 1999.

[193] T. J.-F. OMNÈS AND A. FEROLDI, *LaTeX and Framemaker automatic report generation based on the Conception Tracing Tool (OSC) database*, Internal report, France Télécom CNET, Applied Artificial Intelligence Group, Lannion, France, Aug. 1993.

[194] T. J.-F. OMNÈS AND J.-C. GROSSETIE, *Space-product vectorized computation of singular boundary integral Helmholtz operators*, in Interactive image synthesis and processing on innovative computer architecture, Human Capital and Mobility (HCM) Final Meeting, Swiss Center for Scientific Computation (SCSC), Manno, Switzerland, April 18–20, 1995.

[195] ———, *The generalized (*$\mathbf{Ax}=\lambda.\mathbf{Bx}$*) spectral problem on the interval and the Fast Fourier Transform (FFT)*, DEA thesis in fundamental computer science and parallelism, INPT-ENSAE-UPS, Toulouse, France, Sept. 1996.

[196] T. J.-F. OMNÈS AND R. M. KENDRICK, *European performance measurement and analysis tools survey*, Report to the European Commission (EC), Parallel Applications Center (PAC), Southampton, United Kingdom, Dec. 1996.

[197] T. J.-F. OMNÈS, L. MEBAREK, F.-X. ROUX, Y. LIU, P. LECA, AND M. A. HAMDI, *High Performance BEM-3D acoustics*, Master thesis, Computer Science and Applied Mathematics Department of INP-ENSEEIHT, Toulouse, France, June 1996.

[198] T. J.-F. OMNÈS, P. STEINFELD, AND N. FLOROS, CALIFE *advanced performance modelling*, ESPRIT-IV TOOLS for High-productivity Engineering Design (TOOLSHED) deliverable report, Parallel Applications Center (PAC), Southampton, United Kingdom, Sept. 1997.

[199] Y. OSHIMA, B. SHEU, AND S. JEN, *High speed memory architectures for multimedia applications*, IEEE Circuits and Devices Magazine, (1997), pp. 8–13.

[200] PALLAS, Vampir 2.0 *tutorial*, Technical Report, Pallas GmbH, Bruhl, Germany, Dec. 1998.

[201] P. R. PANDA, N. D. DUTT, AND A. NICOLAU, *Memory data organization of improved cache performance in embedded processor applications*, in Proc. 9th ACM/IEEE International Symposium on System-Level Synthesis (ISSS), La Jolla, CA, 1996, pp. 90–95.

[202] ———, *Incorporating DRAM access modes into high-level synthesis*, IEEE Trans. CAD Integrated Circuits and Systems, 17 (1998), pp. 96–109.

[203] V. PARETO, *Cours d'économie politique*, Rouge, Lausanne, Switzerland, 1896.

[204] A. PARIKH, M. KANDEMIR, N. VIJAYKRISHNAN, AND M. J. IRWIN, *Energy-aware instruction scheduling*, in Proc. 7th ACM/IEEE Int. High Performance Computing (HiPC) Conf., vol. 1970 of Lecture Notes in Computer Science, Springer Verlag, Bangalore, India, 2000, pp. 335–344.

[205] K.-W. PARK AND D. E. GRIERSON, *Pareto-Optimal Conceptual Design of the Structural Layout of Buildings Using a Multicriteria Genetic Algorithm*, Computer-Aided Civil and Infrastructure Engineering, 14 (1999), pp. 163–170.

[206] S.-J. PARK, *A Data Allocation Methodology Using the Multiple Aspects Petri Net and the Pareto Genetic Algorithm in Distributed Databases*, PhD thesis, Korea University, 1997.

[207] N. PASSOS AND E. SHA, *Push-up scheduling : optimal polynomial-time resource constrained scheduling for multi-dimensional applications*, in Proc. IEEE Int. Conf. Comp. Aided Design (ICCAD), San Jose, CA, Nov. 1995, pp. 588–591.

[208] C. S. PATEL, S. M. CHAI, S. YALAMANCHILI, AND D. E. SCHIMMEL, *Power constrained design of multiprocessor interconnection networks*, in International Conference on Computer Design: VLSI in Computers and Processors (ICCD '97), Washington - Brussels - Tokyo, Oct. 1997, IEEE, pp. 408–416.

[209] D. PATTERSON, T. ANDERSON, N. CARDWELL, R. FROMM, K. KEETON, C. KOZYRAKIS, R. THOMAS, AND K. YELICK, *Intelligent RAM (IRAM): chips that remember and compute*, in Proc. IEEE Int. Solid-state Circuits Conf. (ISSCC), San Francisco, CA, Feb. 1997, pp. 224–225.

[210] D. PATTERSON AND J. HENNESSEY, *Computer architecture : a quantitative approach*, Morgan Kaufmann Publisher Inc., San Francisco, CA, 1996.

[211] P. PAULIN, F. KARIM, AND P. BROMLEY, *Network processors: A perspective on market requirements, processor architectures and embedded s/w tools*, in 4th ACM/IEEE Design and Test in Europe Conf. (DATE), special session on System-on-a-Chips (SoCs), Munich, Germany, Mar. 2001, pp. 420–427.

[212] P. G. PAULIN AND J. P. KNIGHT, *Force-Directed Scheduling in automatic data path synthesis*, in Proc. 24th ACM/IEEE Design Automation Conf. (DAC), I. C. S. Press, ed., Miami Beach, FL, June 1987, pp. 195–202.

[213] ———, *High-level synthesis benchmark results using a global scheduling algorithm*, 1988.

[214] ———, *Algorithms for high-level synthesis*, IEEE Design and Test of Computers, (1989), pp. 18–31.

[215] F. PELLEGRINI AND J. ROMAN, SCOTCH *: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, in Proc. High-Performance Computing and Networking (HPCN), vol. 1067 of Lecture Notes in Computer Science, Springer Verlag, Brussels, Belgium, Apr. 1996, pp. 493–498.

[216] PHILIPS, *TriMedia TM1000 data book*, Philips Semiconductors, Sunnyvale, CA, 1997.

[217] C. PIGUET, *Parallelism and low-power*, in Proc. Symp. Architectures de Machines (SymA'99), Rennes, France, June 8, 1999.

[218] C. POLYCHRONOPOULOS, *Compiler optimizations for enhancing parallelism and their impact on the architecture design*, IEEE Trans. on Computers, 37 (1988), pp. 991–1004.

[219] C. POMMERELL, M. ANNARATONE, AND W. FICHTNER, *A set of new mapping and coloring heuristics for distributed–memory parallel processors*, in Proceedings of Copper Mountain Conference on Iterative Methods, vol. 414, 1990, pp. 1–27.

[220] L. PRESS, *Personal computing: the post-PC era*, Communications of the ACM, 42 (1999), pp. 21–24.

[221] B. PRINCE, *Memory in the fast lane*, IEEE Spectrum, (1994), pp. 38–41.

[222] ———, *High Performance Memories*, J. Wiley, 1996.

[223] F. QUILLÉRÉ, S. RAJOPADHYE, AND D. WILDE, *Gestion de la mémoire dans un compilateur d'équations récurrentes*, in Techniques de parallélisation automatique, C. Mongenet, S. Rajopadhye, and Y. Robert., eds., special issue of Technique et Science Informatiques (TSI), Éditions Hermès, Paris, France, Jan. 2001.

[224] F. QUINTANA, J. CORBAL, R. ESPASA, AND M. VALERO, *Adding a vector unit on a superscalar processor*, in 13th Int. Conf. on supercomuting, June 1999.

[225] P. QUINTON, S. RAJOPADHYE, AND T. RISSET, *On manipulating Z-polyhedra*, Tech. Rep. 1016, IRISA, Rennes, France, 1996.

[226] L. R. RABINER AND R. W. SCHAFER, *Digital signal processing of speech signals*, Prentice Hall Int'l Inc., Englewood cliffs, NJ, 1988.

[227] R. RAJARAMAN AND D. F. WONG, *Optimum clustering for delay minimization*, in Proc. 30th ACM/IEEE Design Automation Conf. (DAC), June 1998.

[228] C. R. REEVES, *Modern Heuristic Techniques for Combinatorial Problems*, Scientific Publications, 1993.

[229] M. RENAUDIN, *Asynchronous circuits and systems : a promising design alternative*, in MIGAS Summer School on Microelectronics for Telecommunications : Managing High Complexity and Mobility, Autrans, France, June 28 – July 4, 2000.

[230] T. RISSET, *Contribution à la compilation de nids de boucles sur silicium*, Habilitation à diriger des recherches, Université de Rennes I - IFSIC, Rennes, France, Nov. 2000.

[231] J. ROBINSON, *Efficient general-purpose image compression with binary tree predictive coding*, IEEE Trans. on Image Processing, 6 (1997), pp. 601–608.

[232] Y. G. SAAB AND V. B. RAO, *Combinatorial optimization by stochastic evolution*, IEEE Trans. on CAD, 10 (1991), pp. 525–535.

[233] N. SAITO, *Classification of geophysical acoustic waveforms using time-frequency atoms*, in Proc. Stat. Computing, A. S. Assoc., ed., 1996.

[234] SAKURAI AND KURODA, *Tutorial d: Low power circuit design multimedia LSI's*, in Proc. ED&TC, 1997.

[235] H. SAMSON, *Formal verification and transformation of video and image processing applications*, PhD thesis, ESAT, EE Department, Katholieke Universiteit, Leuven, Belgium, Sept. 1995.

[236] H. SAMSON, F. FRANSSEN, F. CATTHOOR, AND H. DE MAN, *Verification of loop transformations for real–time signal processing applications*, in VLSI Signal Processing VII, J. Rabaey, P. Chau, and J. Eldon, eds., New York, USA, 1994, IEEE Press, pp. 208–217.

[237] R. SCHAFFER, F.CATTHOOR, AND R.MERKER, *Combining background memory management and regular array co-partitioning, illustrated on a full motion estimation kernel*, J. of Parallel Algorithms and Applications, (2000).

[238] P. SCHAUMONT, R. CMAR, S. VERNALDE, M. ENGELS, AND I. BOLSENS, *Hardware reuse at the behavioral level*, in 36th IEEE/ACM Design Automation Conf. (DAC), New Orleans, LA, June 1999, pp. 327–332.

[239] SCHIERMEYER, *Deciding 3-colorability in less than* $o(1.415^n)$ *steps*, in WG: Graph-Theoretic Concepts in Computer Science, International Workshop WG, 1993.

[240] C. SCHULTE, Oz Explorer*: A visual constraint programming tool*, in Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 1997, pp. 667–670.

[241] T. SEKI, E. ITOH, C. FURUKAWA, I. MAENO, T. OZAWA, H. SANO, AND N. SUZUKI, *A 6-ns 1-Mb CMOS SRAM with latched sense amplifier*, IEEE J. of Solid-state Circuits, 28 (1993), pp. 478–483.

[242] SEMATECH, *SIA rodmap*, Technical Report, 1997. Available at `http://notes.sematech.org/ntrs/rdmpmem.nsf`.

[243] E. C. SEWELL, *An improved algorithm for exact graph coloring*, DIMACS series in discrete mathematics and theoretical computer science, 26 (1996).

[244] T.-S. SHEN AND J.-P. SHEU, *Communication free data allocation techniques for parallelizing compilers on multi-computers*, IEEE Trans. on Parallel and Distributed Systems, 5 (1994), pp. 924–938.

[245] A. J. SMITH, *Line size choice for CPU cache memories*, IEEE Transactions on computers, 36 (1987).

[246] S. L. SMITH, *Adaptative asynchronous parallel algorithms in distributed computation*, PhD thesis, University of Colorado, Boulder, Colorado, 1991.

[247] G. S. SOHI, *Microporcessors — 10 years back, 10 years ahead*, in Informatics – 10 Years Back, 10 Years Ahead, R. Wilhelm, ed., Lecture Notes in Computer Science, Springer-Verlag, Berlin Germany, 2001, pp. 208–218.

[248] R. STEPHENS, *A survey of stream processing*, Acta Informatica, 34 (1997), pp. 491–542.

[249] L. STOCKMEYER, *Planar 3-colorability is polynomial complete*, ACM SIGACT News, 5 (1973), pp. 19–25. NCPY.

[250] B. STROUSTUP, *C++ programming language*, Addison-Wesley, ISBN 0-201-88954-4, 3rd ed., 1997.

[251] C.-L. SU AND A. DESPAIN, *Cache design trade-offs for power and performance optimization: a case study*, in Proc. Int. Conf. on Low Power Electronics and Design (ICLPED), Dana Point, CA, 1995, pp. 63–68.

[252] K. P. SYCARA, S. P. ROTH, N. SADEH, AND M. S. FOX, *Resource allocation in distributed factory scheduling*, IEEE Expert, 6 (1991), pp. 29–40.

[253] T. TAKAYANAGI ET AL., *350 MHz time-multiplexed 8-port SRAM and word-size variable multiplier for multimedia DSP*, in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), San Francisco, CA, Feb. 1996, pp. 150–151.

[254] J. TEICH, E. ZITZLER, AND S. BHATTACHARYYA, *Optimized software synthesis for digital signal processing algorithms - an evolutionary approach*, in Proc. IEEE Workshop on Signal Processing Systems (SiPS), 1998.

[255] THE MEDEA DESIGN AUTOMATION ROADMAP, *Design Automation solutions for Europe*, Technical Report, Apr. 2000. Available at `http://www.medea.org`.

[256] O. E. THEEL AND B. D. FLEISCH, *A dynamic coherence protocol for distributed shared memory enforcing high data availability at low costs*, IEEE Trans. on Parallel and Distributed Systems, 7 (1996), pp. 915–930.

[257] Y. THERASSE, G. H. PETIT, AND M. DELVAUX, *VLSI architecture of a SMDS/ATM router*, Annales des Télécommunications, 48 (1993), pp. 166–180.

[258] V. TIWARI, S. MALIK, AND A. WOLFE, *Power analysis of embedded software: a first step towards software power optimisation*, in Proc. 31th ACM/IEEE Design Automation Conf. (DAC), Santa Clara, CA, Nov. 1994, pp. 384–390.

[259] G. TOMLINSON, D. MAJOR, AND R. LEE, *High-capacity Internet middleware: Internet cache system architectural overview*, in Proceedings of the Workshop on Internet Server Performance (WISP99), May 1999.

[260] T. TSURUDA, M. KOBAYASHI, T.TSUKUDE, T. YAMAGATA, AND K. ARIMOTO, *High-speed, high bandwidth design methodologies for on-chip DRAM core multimedia system LSIs*, in Proc. IEEE Custom Integrated Circuits Conf. (CICC), San Francisco, CA, May 1996, pp. 265–268.

[261] C. USSERY, *Configurable Platforms: The ASIC Revolution*, in Invited talk, 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 2000. Slides available at: `http://www.improvsys.com`.

[262] V. VAN DONGEN AND P. QUINTON, *Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays*, in Proc. Int. Conf. on Systolic Arrays, I. C. S. Press, ed., June 1988, pp. 473–482.

[263] R. VAN DRIESSCHE AND D. ROOSE, *A graph contraction algorithm for the calculation of eigenvectors of the Laplacian matrix of a graph with a multi-level method*, Report TW 209, Katholieke Universiteit, Department of Computing Science, Leuven, Belgium, Sept. 1994.

[264] J. VAN MEERBERGEN, P. LIPPENS, W. VERHAEGH, AND A. VAN DER WERF, PHIDEO*: high-level synthesis for high throughput applications*, Journal of VLSI signal processing, 9 (1995), pp. 89–104.

[265] M. VAN SWAAIJ, F. FRANSSEN, F. CATTHOOR, AND H. DE MAN, *Modeling data and control flow for high-level memory management*, in Proc. 3rd ACM/IEEE European Design Automation Conf. (EDAC), Brussels, Belgium, Mar. 1992, pp. 8–13.

[266] A. VANDECAPPELLE, *Physical Memory Management (PMM) Memory Library: Architectural Design Document*, Version 1.6, IMEC-VSDM, Leuven, Belgium, Aug. 1997.

[267] A. VANDECAPPELLE, F. CATTHOOR, E. BROCKMEYER, AND S. WUYTACK, *User requirements for exploitation of special memory access modes*, Internal confidential report, IMEC-DESICS, Leuven, Belgium, June 2000.

[268] A. VANDECAPPELLE, M. MIRANDA, E. BROCKMEYER, F. CATTHOOR, AND D. VERKEST, *Global multimedia system design exploration using accurate memory organization feedback*, in 36th IEEE/ACM Design Automation Conf. (DAC), New Orleans, LA, June 1999, pp. 327–332.

[269] D. VANDERSTRAETEN AND R. KEUNINGS, *Optimized partitioning of unstructured finite element meshes*, Int. J. Num. Meth. Engrg., 38 (1995), pp. 433–450.

[270] J. VANHOOF, K. VAN ROMPAEY, I. BOLSENS, G. GOOSSENS, AND H. DE MAN, *High-level synthesis for real-time digital signal processing*, Kluwer Academic Publishers, ISBN 0-7923-9313-9, 1993.

[271] B. VANHOOF ET AL., *Design of a voice coding ASIC with the* Cathedral-II *silicon compiler*, in Proc. Euro-ASIC Conf., Paris, France, Feb. 1993.

[272] F. VAUGHAN AND D. MUNRO, *Self-adaptive compliant persisent architectures*, in Proceedings of the Seventh IDEA Workshop, Feb. 2000, pp. 5–10.

[273] I. VERBAUWHEDE, F. CATTHOOR, J. VANDEWALLE, AND H. DE MAN, *Background memory management for the synthesis of algebraic algorithms on multi-processor DSP chips*, in Proc. VLSI'89, Munich, Germany, Aug. 1989, pp. 2098–2118.

[274] W. VERHAEGH, E. AARTS, J. KORST, AND P. LIPPENS, *Improved Force-Directed Scheduling (IFDS) in high-throughput digital signal processing*, IEEE Transactions on Computer-Aided Design, 14 (1995), pp. 945–960.

[275] W. F. J. VERHAEGH, *Multidimensional periodic scheduling*, PhD thesis, Eindhoven University of Technology, Philips Electronics, Eindhoven, The Nertherlands, 1995.

[276] W. F. J. VERHAEGH, E. H. L. AARTS, J. H. M. KORST, AND P. E. R. LIPPENS, *Improved Force-Directed Scheduling (IFDS)*, in Proc. of Eur. Conf. on Design Automation, 1991, pp. 430–435.

[277] W. F. J. VERHAEGH, P. E. R. LIPPENS, E. H. L. AARTS, J. H. M. KORST, A. VAN DER WERF, AND J. L. VAN MEERBERGEN, *Efficiency Improvements for Force-Directed Scheduling (IFDS)*, in Proc. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD), IEEE Computer Society Press, Nov. 1992, pp. 286–291.

[278] ———, *Modeling data and control flow for high-level memory management*, in Proc. 3rd ACM/IEEE European Design Automation Conf. (EDAC), Brussels, Belgium, Mar. 1992.

[279] D. VERKEST, J. DA SILVA, C. YKMAN, K. CROES, M. MIRANDA, S. WUYTACK, G. DE JONG, F. CATTHOOR, AND H. DE MAN, Matisse*: A system-on-chip design methodology emphasizing dynamic memory management*, Journal of VLSI Signal Processing, 21 (1999), pp. 277–291.

[280] F. VERMEULEN, F. CATTHOOR, D. VERKEST, AND H. DE MAN, *Formalized three-layer system-level model and reuse methodology for embedded data-dominated applications*, IEEE Trans. on VLSI Systems, 8 (2000), pp. 207–216.

[281] N. VIJAYJRISHNAN, M. KANDEMIR, M. J. IRWIN, H. KIM, AND W. YE, *A unified energy estimation framework with integrated hardware-software optimizations*, tech. rep., Dept. of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802, 2000.

[282] F. VOISIN AND G.-R. PERRIN, *Sparse computations with PEI*, Lecture Notes in Computer Science, 1586 (1999), pp. 1006–??

[283] Z. WANG, M. KIRKPATRICK, AND E. H.-M. SHA, *Optimal two level partitioning and loop scheduling for hiding memory latency for DSP applications*, in Proc. 37th ACM/IEEE Design Automation Conf. (DAC), Los Angeles, USA, June 2000, pp. 556–559.

[284] N. WEHN AND S. HEIN, *Embedded DRAM architectural trade-offs*, in Proc. 1st ACM/IEEE Design and Test in Europe Conf. (DATE), Paris, France, Feb. 1998, pp. 704–708.

[285] N. WESTE AND K. ESHARAGHIAN, *Principles of CMOS VLSI design*, Addison-Wesley, 2nd ed., 1993.

[286] M. WOLF, *Improving data locality and parallelism in nested loops*, PhD thesis, Stanford University, Stanford, USA, Aug. 1992.

[287] W. WOLF, *Object-oriented cosynthesis of distributed embedded systems*, ACM Transactions on Design Automation of Electronic Systems, 1 (1996), pp. 301–314.

[288] D. E. WOMBLE AND D. S. GREENBERG, *Parallel I/O*, Parallel Computing, 23 (1997), p. iii. Introduction to a special issue.

[289] J. WURTZ, *Constrained based scheduling in* Oz, in Proc. Operations Research 1996 (OR'96), Springer-Verlag, 1997.

[290] S. WUYTACK, *Summary of discussion on use of basic set analysis and other domain or signal set partitioning*, Internal Report, IMEC-VSDM, Leuven, Belgium, Mar. 1996.

[291] S. WUYTACK, *Storage-Bandwidth Optimization for Flat Control Flow Graphs (Chapter 7)*, in Custom Memory Management Methodology - Exploration of Memory Organisation for Embedded Multimedia System Design, Kluwer Academic Publishers, ISBN 3-540-64105-X, 1998, pp. 141–170.

[292] S. WUYTACK, *System-level power optimization of data storage and transfer*, Ph.D thesis, ESAT, EE Department, Katholieke Universiteit, Leuven, Belgium, Oct. 1998.

[293] ———, *Storage bandwidth optimization in ATOMIUM: Developer's view*, Internal confidential report version 0.9, IMEC-DESICS, Leuven, Belgium, Oct. 2000.

[294] S. WUYTACK, F. CATTHOOR, G. DE JONG, B. LIN, AND H. DE MAN, *Flow graph balancing for minimizing the required memory bandwidth*, in Proc. 9th ACM/IEEE International Symposium on System-Level Synthesis (ISSS), La Jolla, CA, Nov. 1996, pp. 127–132.

[295] S. WUYTACK ET AL., *Memory management for embedded network applications*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18 (1999), pp. 533–544.

[296] ———, *Minimizing the required memory bandwidth in VLSI system realizations*, IEEE Trans. on VLSI Systems, 7 (1999), pp. 433–441.

[297] T. YAMADA (SONY), *Digital storage media in the digital highway era*, in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), San Francisco, CA, Feb. 1995, pp. 16–20.

[298] J. YANG AND T.-S. CHANG, *Multiobjective scheduling for IC sort and test with a simulation testbed*, IEEE Trans. Semiconductor Manufacturing, 11 (1998), pp. 304–315.

[299] P. YUAN, M. GREEN, AND R. W. H. LAU, *A Framework for Performance Evaluation of Real-time Rendering Algorithms in Virtual Reality*, in Proc. ACM Symp. on Virtual Reality Software and Technology, 1997, pp. 51–58.

[300] Q. ZHAO, S. KARISCH, F. RENDL, AND H. WOLKOWICZ, *Semidefinite programming relaxations for the quadratic assignment problem*, Joint technical report CORR-95-27 and DIKU TR-96/32, University of Waterloo and University of Copenhagen, Canada and Danemark, Feb. 1998.

[301] E. ZITZLER AND L. THIELE, *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*, IEEE Transactions on Evolutionary Computation, 3 (1999), pp. 257–271.