Data and Process Abstraction in PIPS Internal Representation

Fabien Coelho, Pierre Jouvelot, Corinne Ancourt, François Irigoin

MINES ParisTech

Typeset with LaTeX, revision 361

PIPS Overview

project started in 1988, 23 years ago!

interprocedural analyses and transformations

linear algebra based analyses *preconditions, array regions*

par4all HPCProject initiative

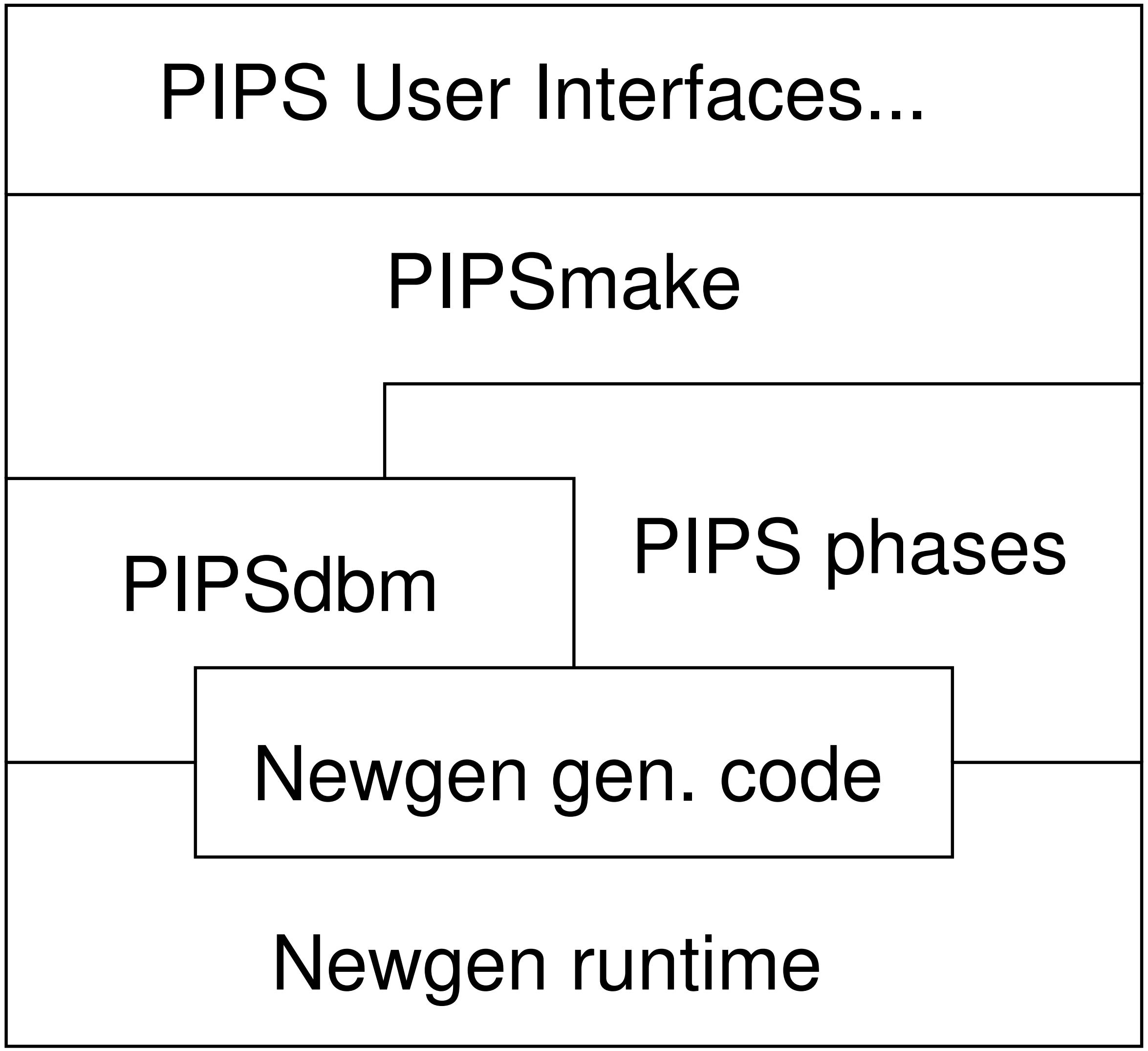
input Fortran 77, Fortran 95, C

output idem, source to source!

Newgen data structure generator, used for IR

PIPSmake on demand dependency handling *à la make*

PIPSdbm database layer for persistence


PIPS User Interfaces...
PIPSmake
PIPS phases
PIPSdbm
Newgen gen. code
Newgen runtime

PIPSmake resource dependencies

- links usage, passes and production
- per PROGRAM or MODULE (fonction)
- transformations: use/produce `MODULE.code`

```
initializer > MODULE.user_file
            > MODULE.initial_file

filter_file > MODULE.source_file
    < MODULE.initial_file
    < MODULE.user_file

bootstrap   > PROGRAM.entities
```

```
parser       > MODULE.parsed_code
             > MODULE.callees
    < PROGRAM.entities
    < MODULE.source_file

controlizer > MODULE.code
    < PROGRAM.entities
    < MODULE.parsed_code

print_code  > MODULE.printed_file
    < PROGRAM.entities
    < MODULE.code
```

Newgen, a Data Description Language (DDL)

- **external** type defined outside of Newgen
- **int bool string** basic types, including enumerations
- **x +** cross product, alternative
- * **[]** {} list, array, set
- -> functional mapping

```
workshop = { WIR, ODES, IMPACT, ACCA };
date = year:int x month:int x day:int;
person = name:string x email:string;
attendee = workshop x person x date;
participants = attendees:attendee*;
```

Newgen code generation for C

- (opaque) data structure definition: both STATIC and DYNAMIC types!

```
#define date_domain 124
typedef struct {
  int type;                // dynamic type tag
  int year, month, day;    // other fields
} * date;
```

- cons/des-tructor, clone `make/free/copy_date(...)`
- accessors `int date_year(date)`
- /de-serialization `write/read_date(FILE *...)`
- typed list constructor, data structure check...

PIPS Intermediate Representation

- **entity** global symbol table
- **statement** Hierarchical Control Flow Graph
- code **decorations** use functional mappings

PIPS Symbol Table

symbol anything with a name!

identifier, type, constant, label, field, parameter, memory location...

tabulated name is a unique key to retrieve an entity

storage in an underlying hashtable for quick retrieval

use prefixes for disambiguation `main:0`result`

global can be large, kept in memory!

necessary for interprocedural analyses

with type, storage, initial value

```
tabulated entity = name:string x type x
  storage x initial:value;

type = statement:unit + area + variable +
  functional + varargs:type + unknown:unit +
  void:qualifier* + struct:entity* +
  union:entity* + enum:entity*;
variable = basic x dimensions:dimension* x
  qualifiers:qualifier*;
basic = int:int + float:int + logical:int +
  overloaded:unit + complex:int + string:value +
  bit:symbolic + pointer:type + derived:entity +
  typedef:entity;
dimension = lower:expression x upper:expression;
```

```
qualifier = const:unit + restrict:unit +
   volatile:unit + register:unit + auto:unit;
functional = parameters:parameter* x result:type;

storage = return:entity + ram + formal + rom:unit;

value = code + symbolic + constant +
  intrinsic:unit + unknown:unit + expression;
```

PIPS code: Hierarchical Control Flow Graph (HCFG)

language C or Fortran, but specific constructs. . .

AST semantical whenever possible

- only for really structured code!
- otherwise false loops are *desugarized*
- can be relied on for semantical analyses

CFG handles goto, exit, continue, return. . .

- with `unstructured control` domains
- *prettyprint* regenerate necessary gotos

PIPS statement definition

```
statement = label:entity x number:int x
  ordering:int x comments:string x instruction x
  declarations:entity* x decls_text:string x
  extensions;

instruction = sequence + test + loop +
  whileloop + goto:statement + call +
  unstructured + multitest + forloop +
  expression;

sequence = statements:statement* ;
```

```
test = condition:expression x true:statement x
  false:statement;

loop = index:entity x range x body:statement x
  label:entity x execution x locals:entity*;

whileloop = condition:expression x
  body:statement x label:entity x evaluation;

call = function:entity x arguments:expression*;

unstructured = entry:control x exit:control;
control = statement x predecessors:control* x
  successors:control*;
```

```
forloop = initialization:expression x
  condition:expression x increment:expression x
  body:statement;

expression = syntax x normalized;
syntax = reference + range + call + cast +
  sizeofexpression + subscript + application +
  va_arg:sizeofexpression*;
reference = variable:entity x indices:expression*;
```

Code decorations

- use pointer to value mappings *hashtable*

```
statement_effects = persistant statement -> effects
```

- serialization: must rely on an absolute identifier

  statement number computed on the code

Newgen generic recursion engine

- use dynamic typing tag to check/guide the recursion
- per-recursion context to pass data structures
- apply functions per newgen domain
- top-down: filter function tells whether to go on
- bottom-up: rewrite function applied if filter said true
- optimization: does not recurse if not needed
- can abort the recursion
- stack query: parent, parent of a given type...

Code examples

1. Index of a loop within a test?
2. (Simple) variable substitution
3. add control counters

Index of a loop within a test?

```
typedef struct {
  entity var;  bool is_index;
} ctx;

static bool loop_flt(loop l, ctx * c) {
  if (loop_index(l)==c->var &&
      gen_get_ancestor(test_domain, l)!=NULL) {
    c->is_index = true;
    gen_recurse_stop(NULL);
  }
  return true;
}
```

```
bool var_is_index_in_test(statement s, entity v)
{
  ctx cs = { v, false };
  gen_context_multi_recurse(s, &cs,
     loop_domain, loop_flt, gen_null,
     NULL);
  return cs.is_index;
}
```

(Simple) variable substitution

```
typedef struct {
  entity from, to;
} ctx;

static void loop_rwt(loop l, ctx * c) {
  if (loop_index(l)==c->from)
    loop_index(l) = c->to;
}

static void ref_rwt(reference r, ctx * c) {
  if (reference_variable(r)==c->from)
    reference_variable(r) = c->to;
```

```
}

void subs_var(statement s, entity from, entity
to) {
  ctx cs = { from, to };
  gen_context_multi_recurse(s, &cs,
    loop_domain, gen_true, loop_rwt,
    reference_domain, gen_true, ref_rwt,
    NULL);
}
```

Control counters: before

```
int compute(int n) {
  int i = 1;
  while (i<n) {
    i<<=1;
    if (rand()) i++;
  }
  return i;
}
```

```
int compute(int n) {
  int i = 1;
  int if_then_0 = 0, if_else_0 = 0, while_0 = 0;
  while (i<n) {
    while_0 = while_0+1;
    i <<= 1;
    if (rand()) {
      if_then_0 = if_then_0+1;
      i++;
    } else
      if_else_0 = if_else_0+1;
  }
  return i;
}
```

```
// File "add_control_counter.c"
#include "..."

statement make_increment(entity var) {
  return make_assign_statement(...);
}

entity create_counter
  (entity module, string name) {
  return ...;
}

// Add Control Counter recursion context
typedef struct { entity module; } acc_ctx;
```

```
// add a new counter at entry of statement "s"
void add_counter
   (acc_ctx * c, string name, statement s)
{
  entity cnt = create_counter(c->module, name);
  insert_statement(s, make_increment(cnt), true);
}

void test_rwt(test t, acc_ctx * c) {
  add_counter(c, "if_then", test_true(t));
  add_counter(c, "if_else", test_false(t));
}
```

```
void loop_rwt(loop l, acc_ctx * c) {
  add_counter(c, "do", loop_body(l));
}

void whileloop_rwt(whileloop w, acc_ctx * c) {
  add_counter(c, "while", whileloop_body(w));
}

void forloop_rwt(forloop f, acc_ctx * c) {
  add_counter(c, "for", forloop_body(f));
}
```

```
// add control counter instrumentation
void add_cnt(entity module, statement root)
{
  acc_ctx c = { module };
  gen_context_multi_recurse
     (root, &c,
      test_domain, gen_true, test_rwt,
      loop_domain, gen_true, loop_rwt,
      whileloop_domain, gen_true, whileloop_rwt,
      forloop_domain, gen_true, forloop_rwt,
      NULL);
}
```

```
// PASS: instrument with control structure counters
bool add_control_counters(string name) {
  entity module = name_to_entity(name);
  statement stat = (statement)
    db_get_memory_resource(DBR_CODE, name, true);
  set_current_module_entity(module);
  set_current_module_statement(stat);
  add_cnt(module, stat);
  DB_PUT_MEMORY_RESOURCE(DBR_CODE, name, stat);
  reset_current_module_entity();
  reset_current_module_statement();
  return true;
}
```

Conclusion

Newgen provides useful services

- results in quite homogeneous code
- powerful recursion engine

HCFG representation for source to source

- extensions from Fortran 77 to C and Fortran 95

23 years later and still going strong!

- a lot of thesis work was capitalized

PAWS PIPS As a Web Service *new web interface*