ERBIUM: A Deterministic, Consurrent Intermediate Representation to Map Data-Flow Tasks to Scalable, Persistent Streaming Processes

Cupertino Miranda *Philippe Dumont* *Albert Cohen*
Marc Duranton *Antoniu Pop*

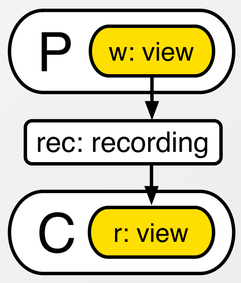
INRIA MINES ParisTech

LANGUAGE

```
main () {
#pragma omp parallel for \
        num_threads(3)
#pragma omp single
   for(int i = 1; i <= 1000; i++)
   {
#pragma omp task output(tmp)
      int tmp = i + external
#pragma omp task input(tmp)
      printf("%d\n",
            tmp * external);
   }
}
```

Front-end Expansion

```
process C (record int rec) {
   view r; int i = 0;
   connect_reader (r, rec);
   set_horizon (r, 1);
   while ((i = update (r, i + 1)) != 0)
      printf ("%d\n",
            r[[i]] * external);
      release (r, i);
   }
}
```

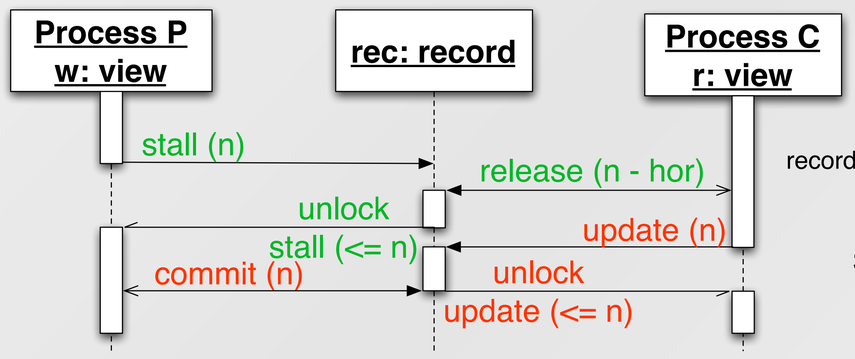


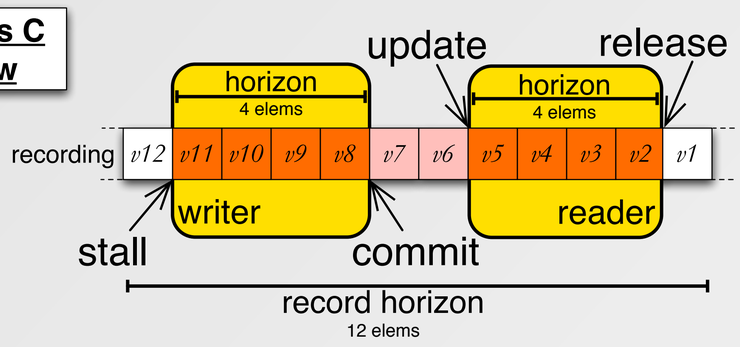

```
process P (record int rec) {
   view w;
   connect writer (w, rec);
   set_horizon (w, 1);
   for (int i = 1; i <= 1000; i++) {
      stall (w, i);
      w[[i]] = i + external;
      commit (w, i);
   }
}

main () {
   record int rec;
   set_number_of_writers(1);
   set_number_of_readers(1);
   run P(rec);
   run C(rec);
}
```

Motivation:

- data-flow semantics for determinism
- task parallelism for extra scalability
- combined data and task parallelism to hide latency (data-driven execution)
- pipeline parallelism to reduce the impact on memory bandwidth





COMPILATION

```
typedef struct {
   record int rec;
} P_params, C_params;

void P (P_params *params) {
   record int rec = params->rec
   view w = alloc_view();
   connect writer (w, rec);
   set_horizon (w, 1);
   for (int i = 1; i <= 1000; i++) {
      stall (w, i);
      w[[i]] = i + external;
      commit (w, i);
   }
   free_view (w);
}
main () {
   int hor = BEST_POWER_OF_2
   void *rec;
   rec = alloc_record(sizeof(int), hor);
   set_number_of_writers(rec, 1);
   set_number_of_readers(rec, 1);
   pthread_create (&P, &{ rec });
   pthread_create (&C, &{ rec });
}
```

Process / Run expansion
Parameter marshaling
Initialization
Termination
Record buffer size

```
void C (P_params *params) {
   record int rec = params->rec
   view r = alloc_view();
   connect_reader (r, rec);
   set_horizon (r, 1);
   while ((i = update (r, i + 1)) != 0)
      printf ("%d\n",
            r[[i]] * external);
   release (r, i);
   }
   free_view (r);
}
```

```
void C (P_params *params) {
   ...
   set_horizon (r, 256);
   int mask = r->position_mask;
   receive (r, i + 16);
   while ((i = update (r, i + 256)) != i + 256)
   {
      receive (r, i + 256);
      for (int j = 0; j < 256; j++)
      {
         int pos = (i - 256) & mask;
         printf ("%d\n", r->data[pos + j] *
                        external);
      }
      release (r, i);
      prev_i = i;
   }
   if(i != 0) {
      int pos = (i - 256) & mask;
      for(int j = prev_i+1; j <= i; j++)
         printf ("%d\n", r->data[j & mask]);
      release (r, i);
   }
   free_view (r);
}
```

Blocking / Windows
DMA / Prefetching
OCC expansion

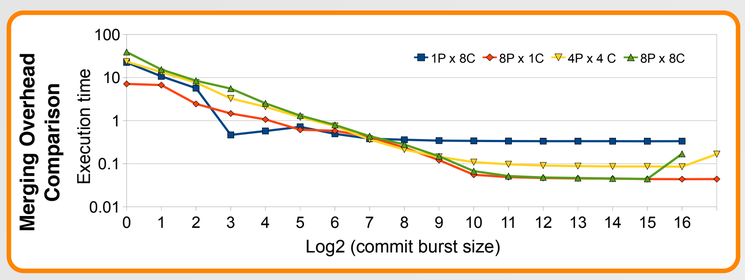
```
...
   while ((i = update (r, i + 256)) != i + 256)
   {
      receive (r, i + 256);
      int pos = (i - 256) & mask;
      for (int j = 0; j < 256; j += vector_factor)
      {
         vector int tmp1 = {external, ... };
         vector int tmp2;
         tmp2 = tmp1 *
               (vector int *) &(r->data[pos + j]);
         for(int h = 0; h < vector_factor; h++)
            printf ("%d\n", tmp2[h]);
      }
      release (r, i);
      prev_i = i;
   }
...
```

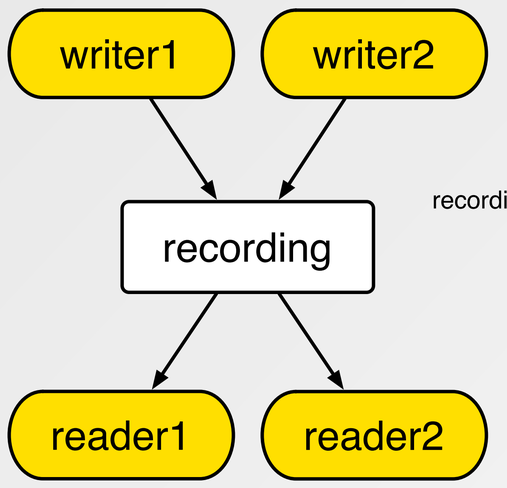
Vectorization
Code Motion

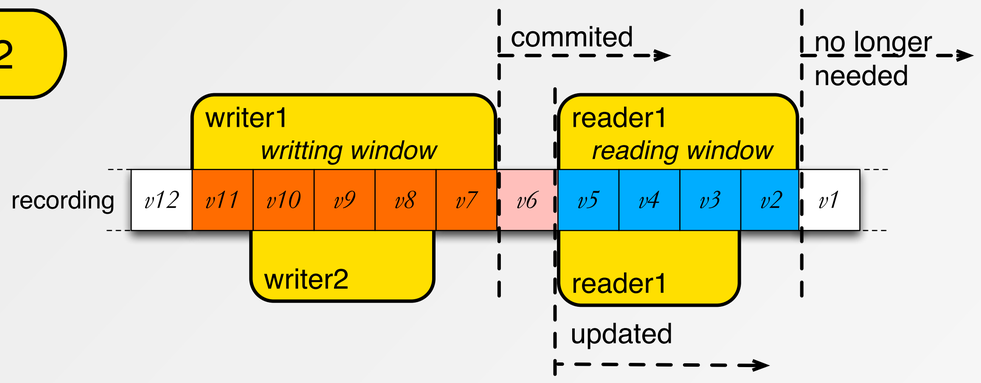
Task fusion
Static scheduling
} New Intermediate Representation

RUNTIME

- Multi-producer, multi-consumer FIFO
- FIFO with random peek/poke access
- Lock-free implementation:
  - neither atomic instruction nor fences
- Cache-conscious





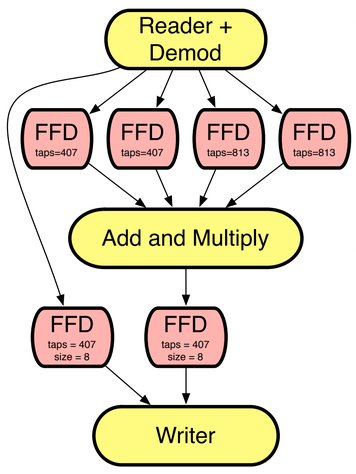


EXECUTION

Applications

GNU FMRadio	task- & data-parallel	+ vectorized
Xeon	10.1	12.6
Opteron	9.52	14.6

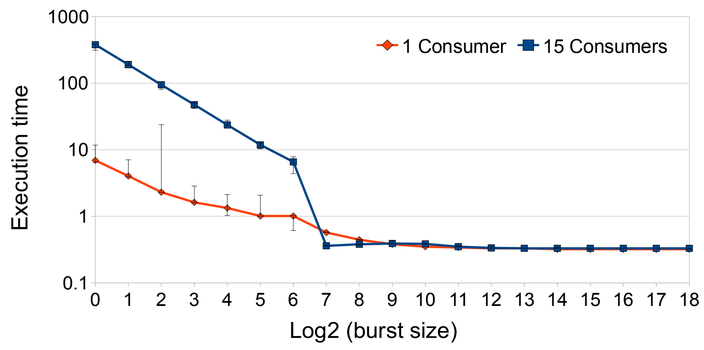
802.11a	task-parallel	data-parallel	combined
Xeon	1.85	1.84	6.67
Opteron	2.73	2.81	7.45



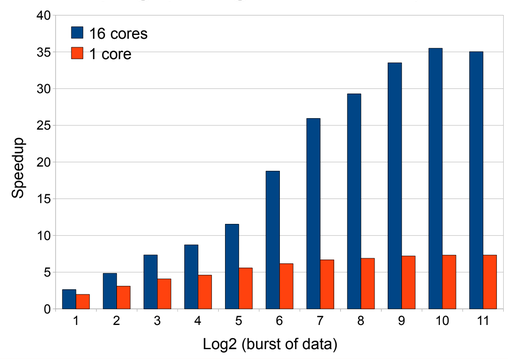
Xeon Machine:
- 4 socket Intel Xeon E7450
- 24 cores at 2.4 GHz
- 64 GB of memory
- 32KB L1 cache,
- 3MB L2 (2 cores)
- 12MB L3

Opteron Machine:
- 4 socket AMD Opteron 8380
- 16 cores at 2.5 GHz
- 64 GB of memory
- 64 KB L1 cache
- 512 KB L2 per core
- 6 MB L3

Overhead / Burst Analysis (Opteron)

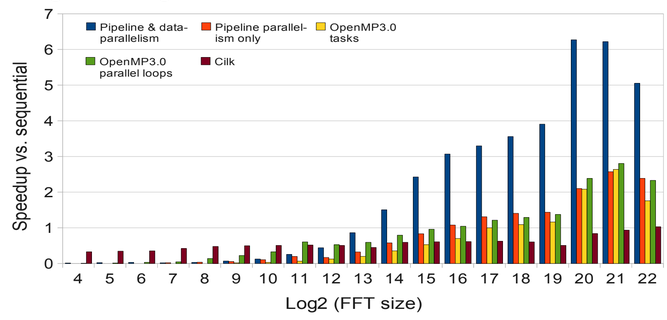


Jpeg (fine grain version)

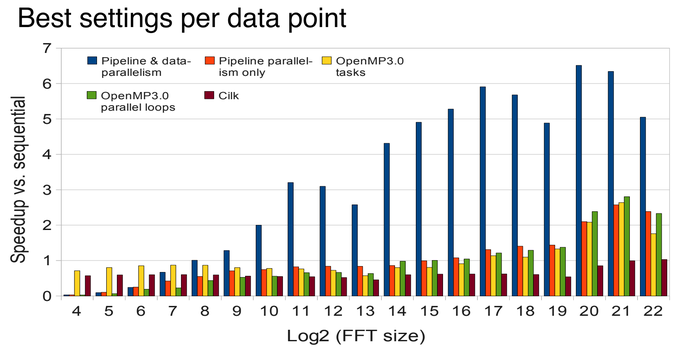


Fine-grain parallelism with FFT (Opteron)

Single settings



Best settings per data point



compilation flow of a data- and task-parallel program