Fast Dynamic Streaming Applications Generation for MPSoC Architectures

Béatrice CREUSILLET

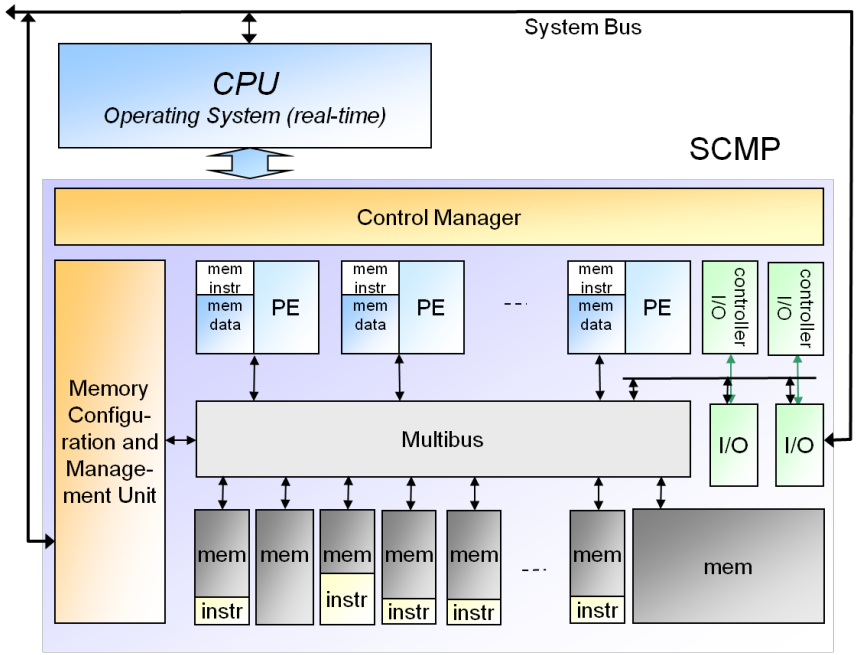
HPC Project

4èmes Rencontres de la Communauté Française de compilation
Saint-Hippolyte 2012

Work funded by the ARTEMIS Joint Undertaking grant agreement No. 1000029,
and achieved in collaboration with N. Ventroux (CEA/List).

General context: Distributed Embedded Systems

- Dynamic streaming applications
  - Well-balanced pipeline
  - Efficient scheduling
- SCMP: Asymmetric MPSoC (CEA)
  - Centralized control core(s) $\longrightarrow$ task scheduling, load balancing
  - Computing cores
  - Sesam Simulator

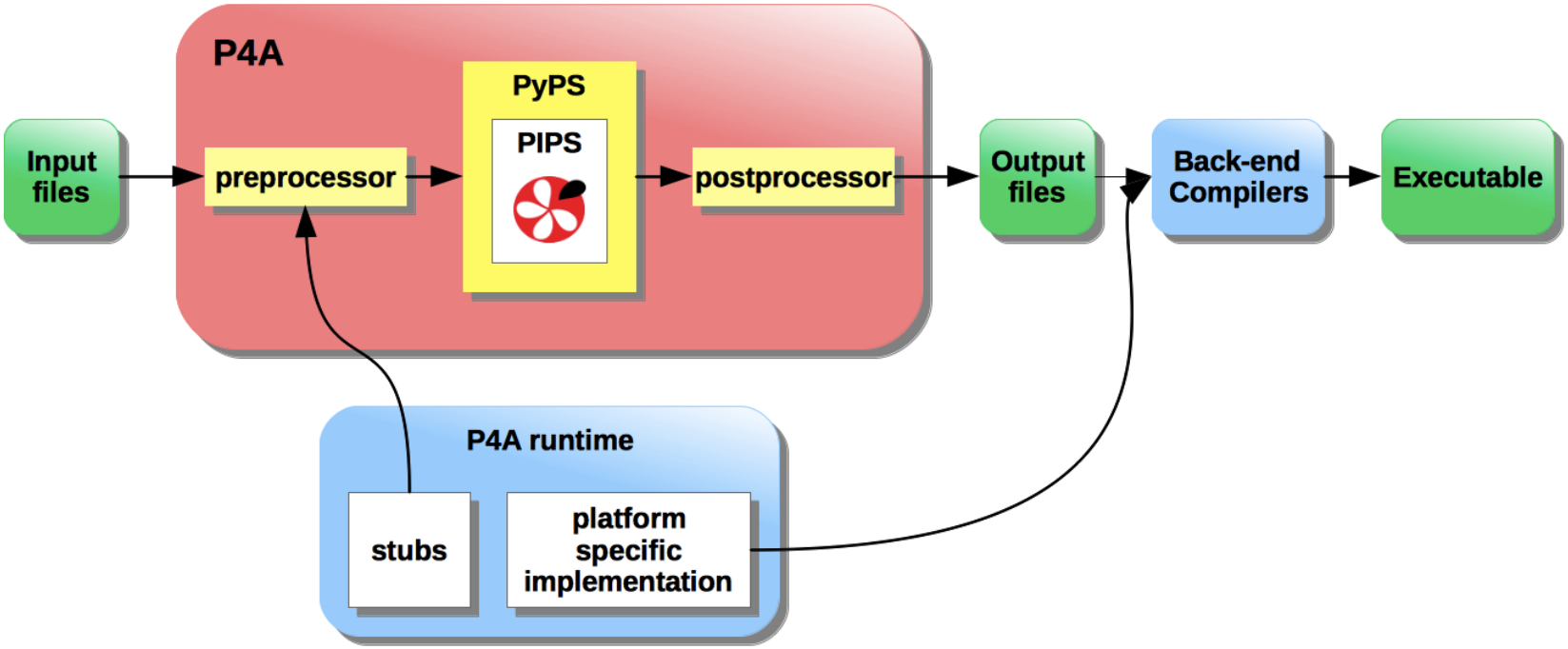


(Reproduced with the authorization of N. Ventroux, CEA)

- Sesam Hardware Abstraction Layer
  - Assembly language for control tasks (CDFG)
  - Explicit synchronizations
  - Communications via shared memory buffers
    - allocation/freeing
    - read/write
    - blocking wait / non blocking send $\longrightarrow$ hand-shake protocol

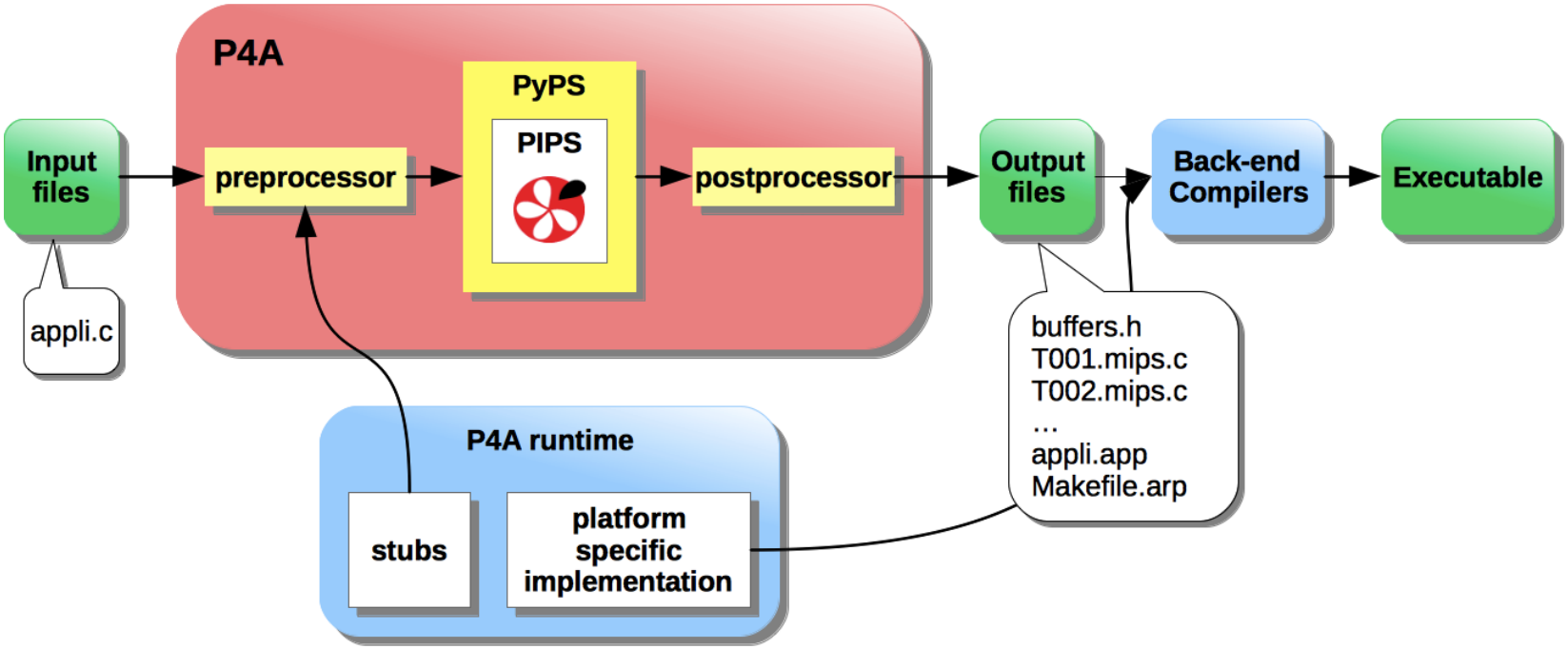
Goal: Testing Different Parallelization Strategies

- Semi-automatic code generation from sequential code
- Manual task kernels identification
  - pragmas
  - without input/output data
- Automatic code generation (Par4All)



Goal: Testing Different Parallelization Strategies

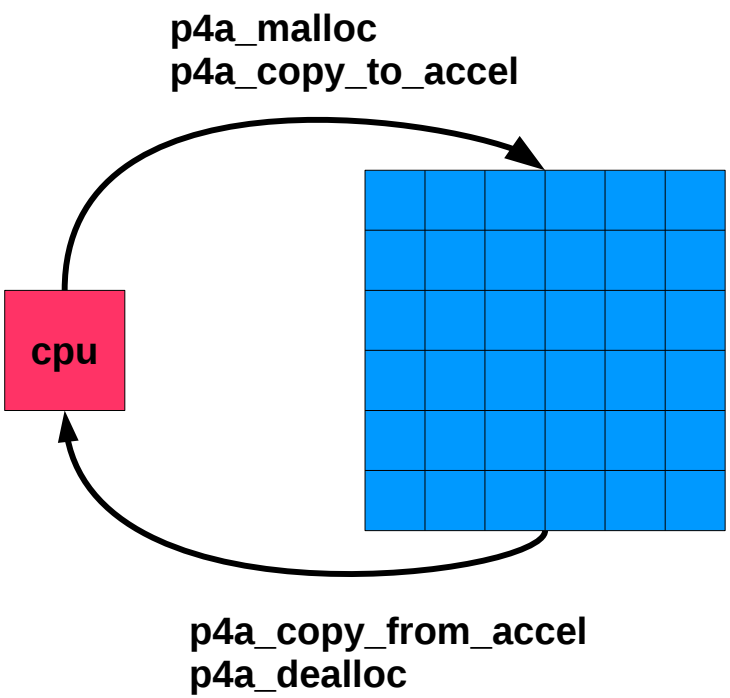
- Semi-automatic code generation from sequential code
- Manual task kernels identification
  - pragmas
  - without input/output data
- Automatic code generation (Par4All)



Gpu Programming model

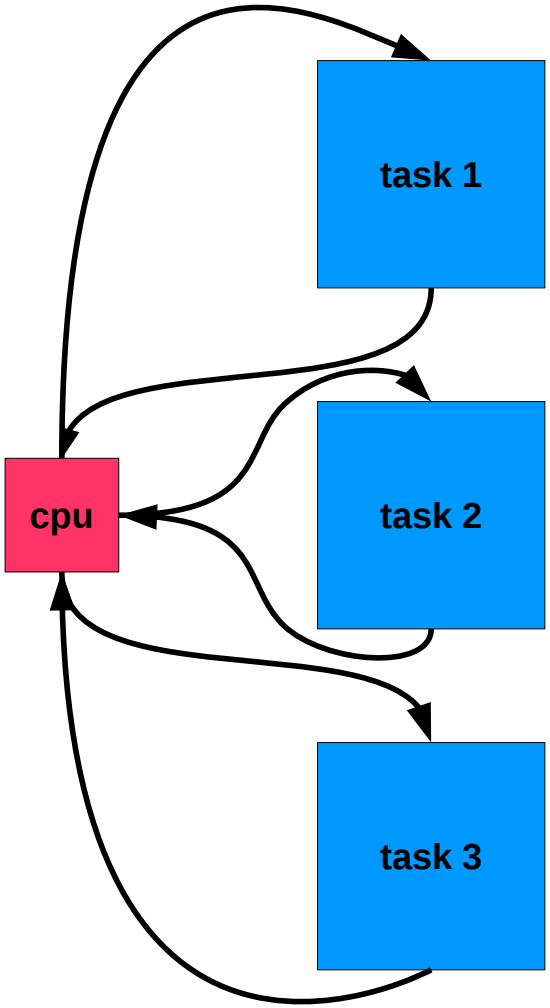
Statement isolation (Serge Guelton's PhD):

- isolate the data of a computation kernel
- externalize its memory space on a distant medium
- based on array regions analyses



- Identical code on host and nodes
- Symmetric runtimes for host and nodes
  - data consistency
- host ≈ data server

Extending GPU Programming model (I)

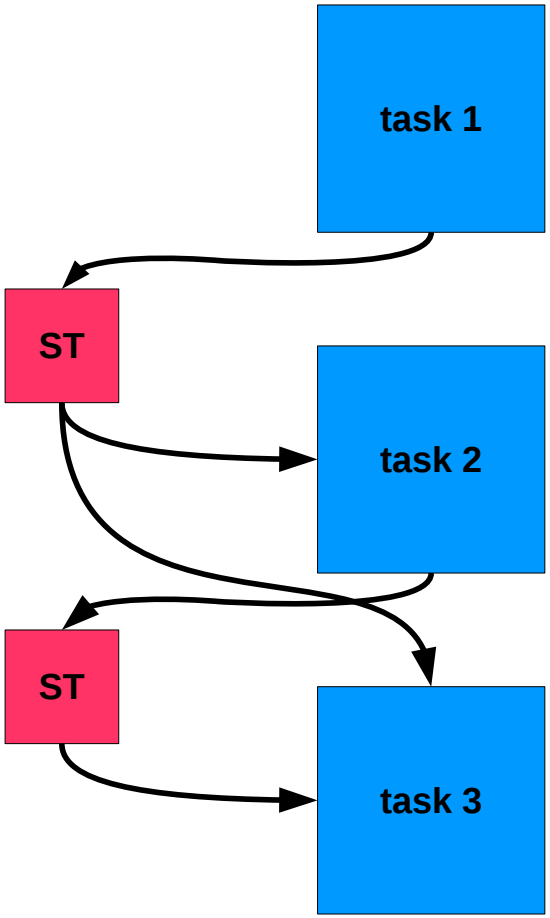


Idea: reuse the SPMD model

- One host task
- Several computing tasks

$\longrightarrow$ sequentialization ☹

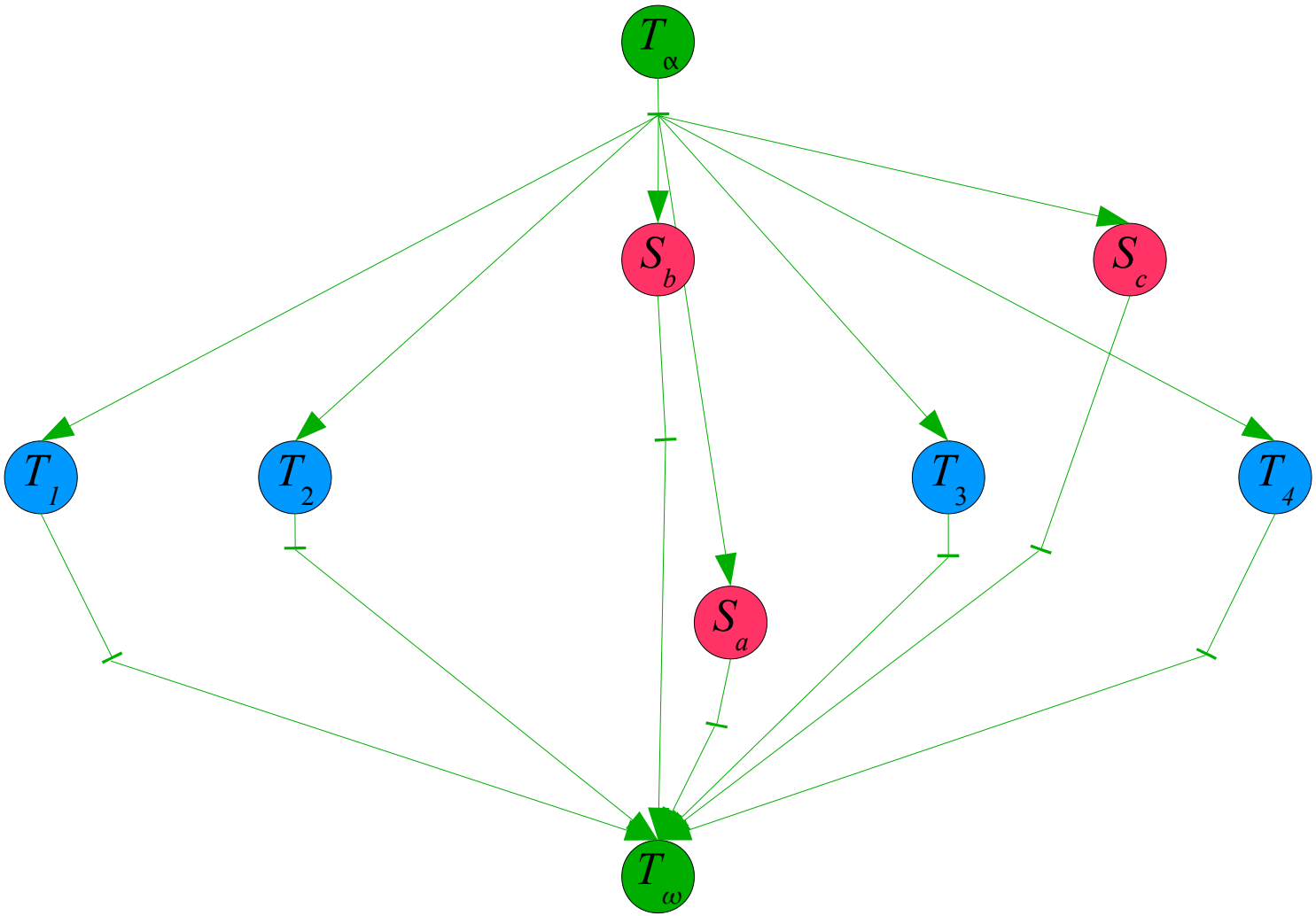
Extending GPU Programming Model (II)



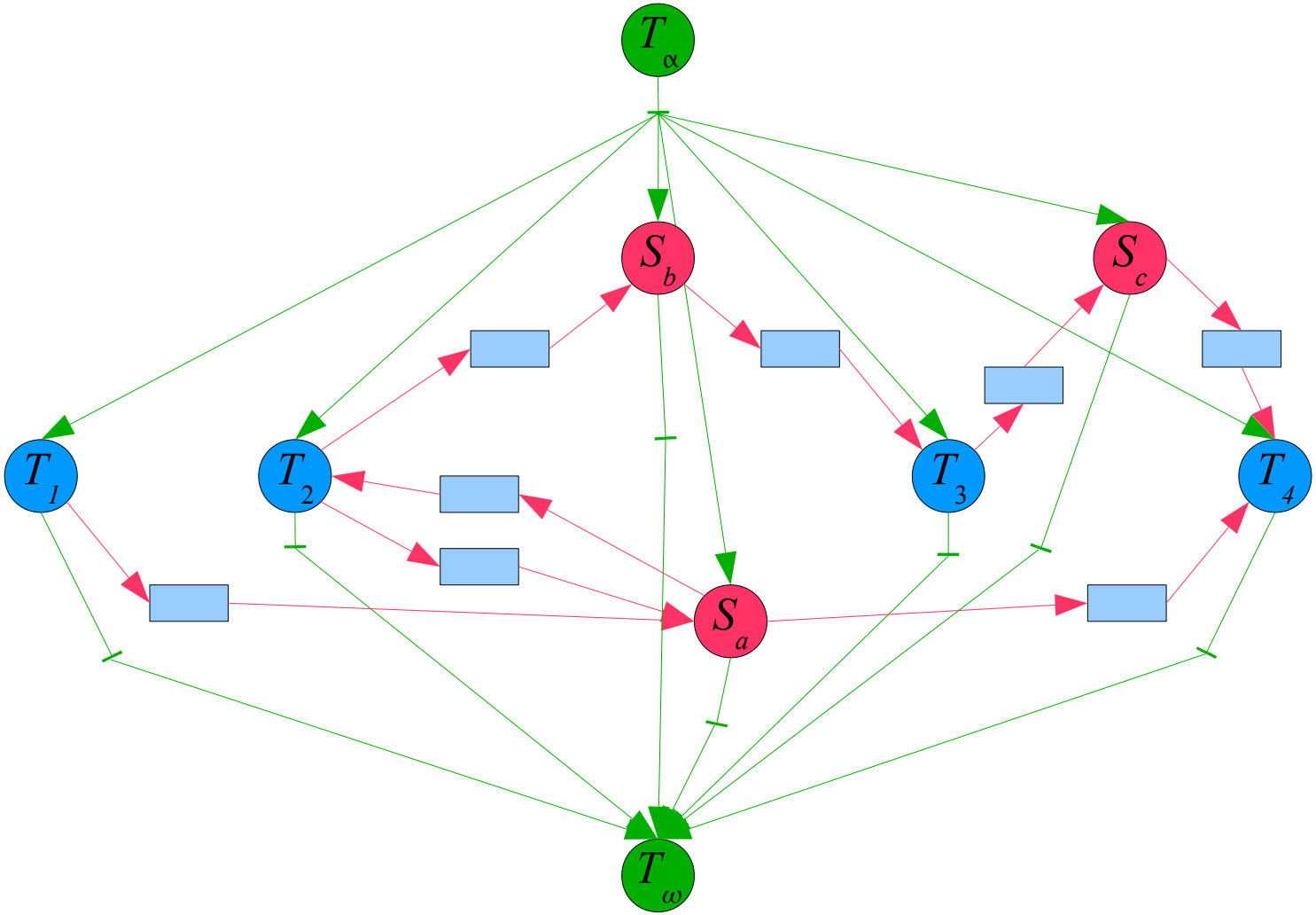
Generalizing the model:

- Several computing tasks
- Several data server tasks:
  - enables task parallelism
  - ensures data consistency (even in case of several producers)
- But potential overhead:
  - time for copying data from/to server task memory space
  - memory

Control Data Flow Graph



Control Data Flow Graph and Communications



Input Code

```
int main() {
  int i, t, a[20], b[20];
  for (t=0; t < 100; t++)
  {
kernel_tasks_1:
    for(i=0; i <10; i++)  a[i] = i+t;

kernel_tasks_2:
    for(i=10; i <20; i++) a[i] = 2*i+t;

kernel_tasks_3:
    for(i=0; i <20; i++)  printf("a[%d] = %d\n", i, a[i]);
   }
   return(0);
}
```

Tasks and buffers description file

```
#define NB_BUFFERS 3
#define P4A__a__0_id 0
#define P4A__a__1_id 1
#define P4A__a__2_id 2
...
#ifdef kernel_task_2
#define kernel_task_2_p 1
#define P4A__a__0_cons_p 0
#define P4A__a__0_prod_p 0
#define P4A__a__1_cons_p 0
#define P4A__a__1_prod_p 1
#define P4A__a__2_cons_p 0
#define P4A__a__2_prod_p 0
#else
#define kernel_task_2_p 0
#endif
...
```

```
#ifdef P4A_sesam_server_a
#define P4A_sesam_server_a_p 1
#define P4A__a__0_cons_p 1
#define P4A__a__0_prod_p 0
#define P4A__a__1_cons_p 1
#define P4A__a__1_prod_p 0
#define P4A__a__2_cons_p 0
#define P4A__a__2_prod_p 1
#else
#define P4A_sesam_server_a_p 0
#endif
```

Code for the second task

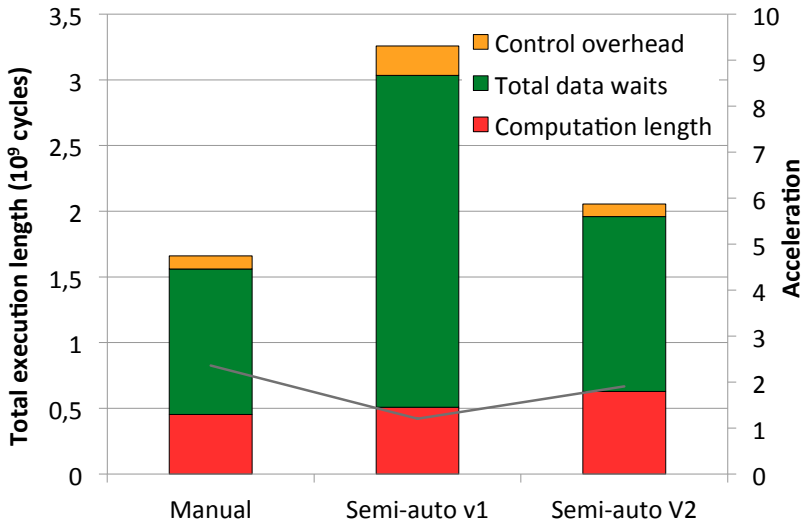
```
#define kernel_task_2
#include "scmp_buffers.h"
int main() {
 P4A_scmp_reset();
 int i, t, a[20], b[20];
 for(t = 0; t <= 99; t += 1) {
  [...]
  {
   int (*P4A__a__1)[10] = (int (*)[10]) 0;
   P4A_accel_malloc((void **) &P4A__a__1, sizeof(int)*10,
       P4A__a__1_id, P4A__a__1_prod_p || P4A__a__1_cons_p,
       P4A__a__1_prod_p);

   if (kernel_task_2_p)
     for(i = 10; i <= 19; i += 1)  (*P4A__a__1)[i-10] = 2*i+t;

   P4A_copy_from_accel_1d(sizeof(int), 20, 10, 10,
       P4A_sesam_server_a_p ? &a[0] : NULL, *P4A__a__1,
       P4A__a__1_id, P4A__a__1_prod_p || P4A__a__1_cons_p);
   P4A_accel_free(P4A__a__1, P4A__a__1_id,
       P4A__a__1_prod_p || P4A__a__1_cons_p,
       P4A__a__1_prod_p);
  }
 [...]
 }
 return(ev_T004);
}
```

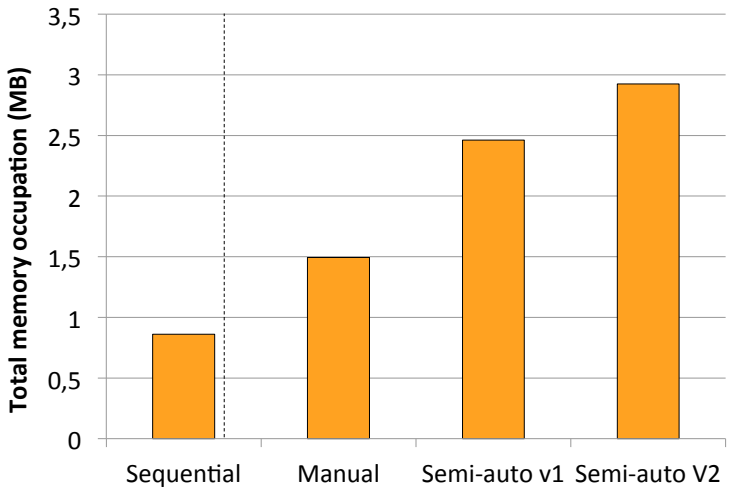
Code for the server task

```
#define P4A_sesam_server_a
#include "scmp_buffers.h"
int main() {
 P4A_scmp_reset();
 int i, t, a[20], b[20];
 for(t = 0; t <= 99; t += 1) {
  [...]
  {
   int (*P4A__a__1)[10] = (int (*)[10]) 0;
   P4A_accel_malloc((void **) &P4A__a__1, sizeof(int)*10,
       P4A__a__1_id, P4A__a__1_prod_p || P4A__a__1_cons_p,
       P4A__a__1_prod_p);

   if (kernel_task_2_p)
     for(i = 10; i <= 19; i += 1)  (*P4A__a__1)[i-10] = 2*i+t;

   P4A_copy_from_accel_1d_server(sizeof(int), 20, 10, 10,
       P4A_sesam_server_a_p ? &a[0] : NULL, *P4A__a__1,
       P4A__a__1_id, P4A__a__1_prod_p || P4A__a__1_cons_p);
   P4A_accel_free(P4A__a__1, P4A__a__1_id,
       P4A__a__1_prod_p || P4A__a__1_cons_p,
       P4A__a__1_prod_p);
  }
 [...]
 }
 return(ev_T004);
}
```

Results on Real Life Sensing Application (Thales)

Parallelized versions	nb computation tasks	dvpt time	acceleration on 8 PEs
Manual (CEA)	18	7 days	2,35
Semi-auto V1	19 (8 kernel + 9 server tasks)	1/2 day	1,20
Semi-auto V2	31 (17 kernel + 12 server tasks)	30 min	1,91





- small computation overhead
- significant memory overhead
- comparable data waits/computation ratio
  $\longrightarrow$ fair image of pipeline balancing

Conclusion

New architectures $\longrightarrow$ lots of opportunities . . . to test!

- Sesam: architecture exploration
- Par4all: code generation exploration
  - generate multiple parallelized versions at low cost
  - moderate impact on performances

Sesam/Par4All: complete exploration chain for asymmetric MPSoCs

Further possible enhancements

- Avoid server tasks
- Avoid local data copies
- Optimize communications (Mehdi Amini - LCPC)
- Further specialize tasks code (dead code elimination)
- Fully automate the process