

Reducing fine-grain communication overhead in multithread code generation for heterogeneous MPSoC

Lisane Brisolara¹, Sang-il Han^{2,3}, Xavier Guerin², Luigi Carro¹,
Ricardo Reis¹, Soo-Ik Chae³, Ahmed Jerraya²



SCOPES 2007

Nice, France

¹Instituto de Informática,
UFRGS, Brazil

²TIMA Laboratory,
SLS Group, France

³School of EECS, SDGroup,
Seoul National University, Korea



Grenoble,
France

9107 Km

Seoul,
Korea

10155 Km

Porto Alegre,
Brazil

Introduction

- Heterogeneous MPSoCs are becoming attractive solutions for emerging embedded systems
- Software programming on heterogeneous MPSoC is a complex task
 - High abstraction level and all of automation that you can get

Introduction (2)

- Fine-grain specification provides more optimization possibilities in a MPSOC design
 - such as exploiting fine-grain parallelism, more efficient partitions, and fine-grain memory optimization
- However, fine-grain models may introduce a large number of messages
 - Increasing communication overhead in terms of:
 - Execution time
 - Memory size

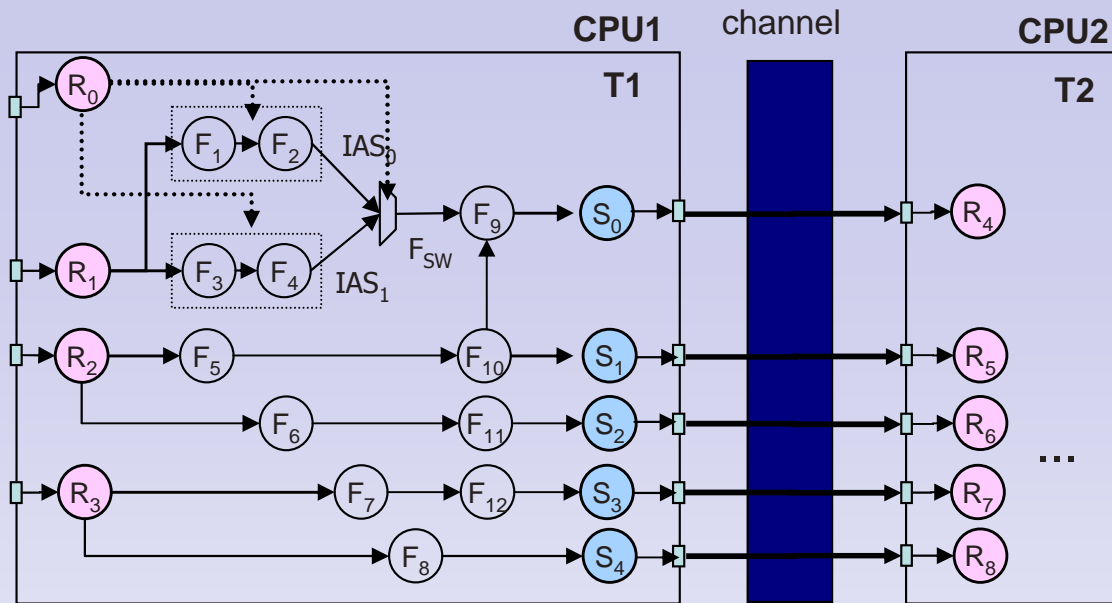
How to reduce the communication overhead at the fine granularity level?

Motivation

- **Message Aggregation (MA)** increases the granularity of the data transfers
- MA merges messages with identical source and destination, reducing:
 - synchronization cost
 - the number of channels required to promote the communication in SW

MA may promote reductions on execution time and memory size by using larger messages

Motivation example

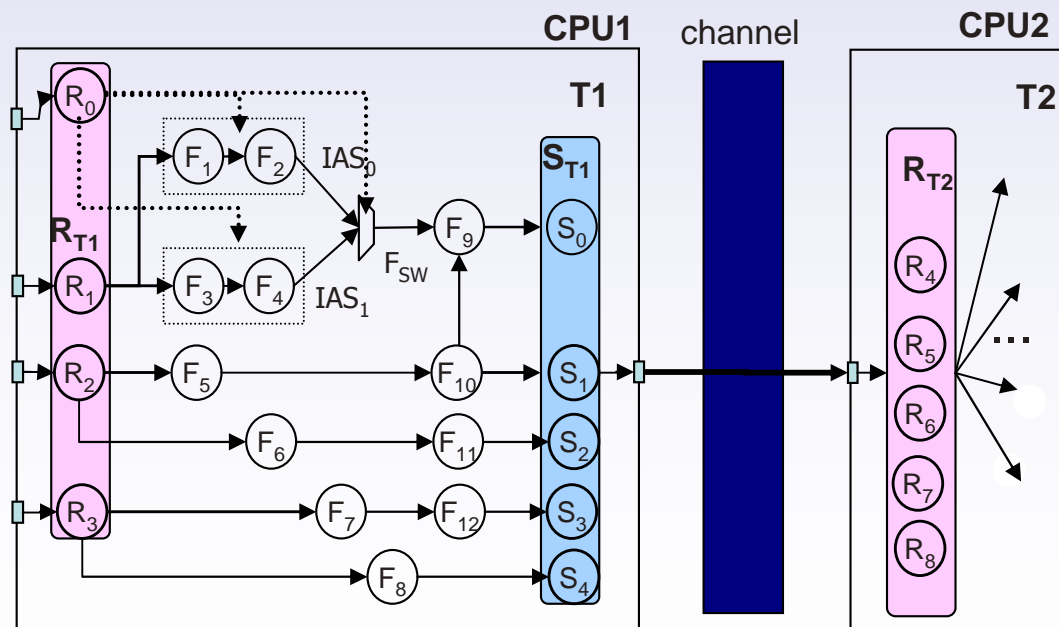


(a) Fine-grain specification

```

T1(){
    recv (R0,8 ); // recv 8 bytes
    recv (R1,8);
    F1();
    ...
    send (S0, 8); // send 8 bytes
    send (S1, 8);
    send (S2, 8);
    send (S3, 8);
    send (S4, 8);
}
    
```

(b) Code without MA



(c) Fine-grain specification after MA

```

T1(){
    recv (RT1, 40 ); // recv 40 bytes
    F1();
    ...
    send (ST1, 40); // send 40 bytes
}
    
```

(d) Code with MA

Motivation (3)

- **Message Aggregation (MA)** technique can:
 - Reduce communication overhead
 - Increase Performance
 - Reduce data memory size
- **What is the problem?**
 - Big number of data transfers
 - Which messages should one aggregate?

Proposed solution

- Integration of the **Message Aggregation** in an automatic code generation flow
- Automatic optimization to support design space exploration

Outline

- Our Approach
 - Multithread code generation flow
 - Message Aggregation
- Case Study
- Conclusions
- Future Work

Our approach

Simulink model

System model in high abstraction level

256 blocks and 286 data links

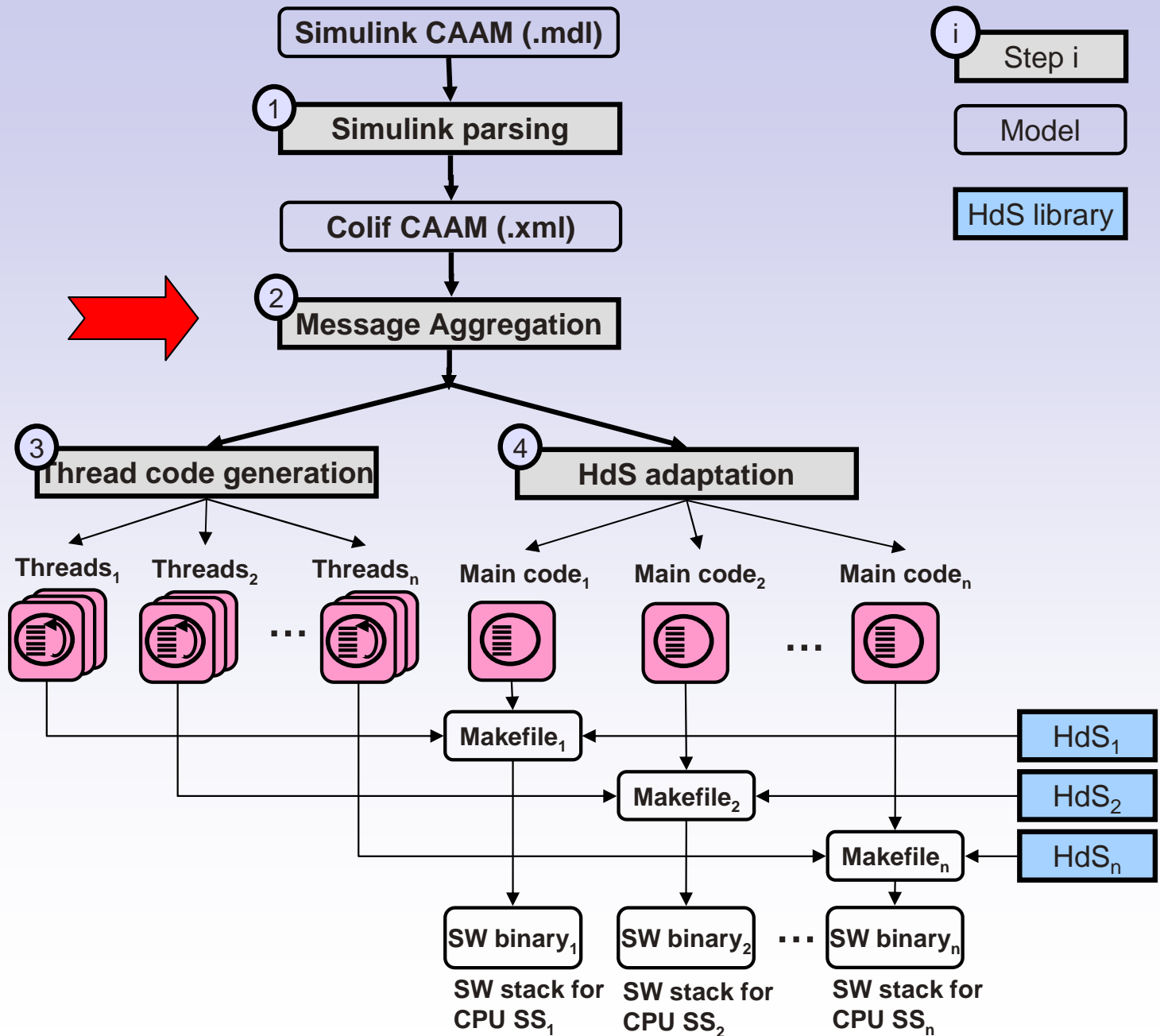
generate

Multithread C code

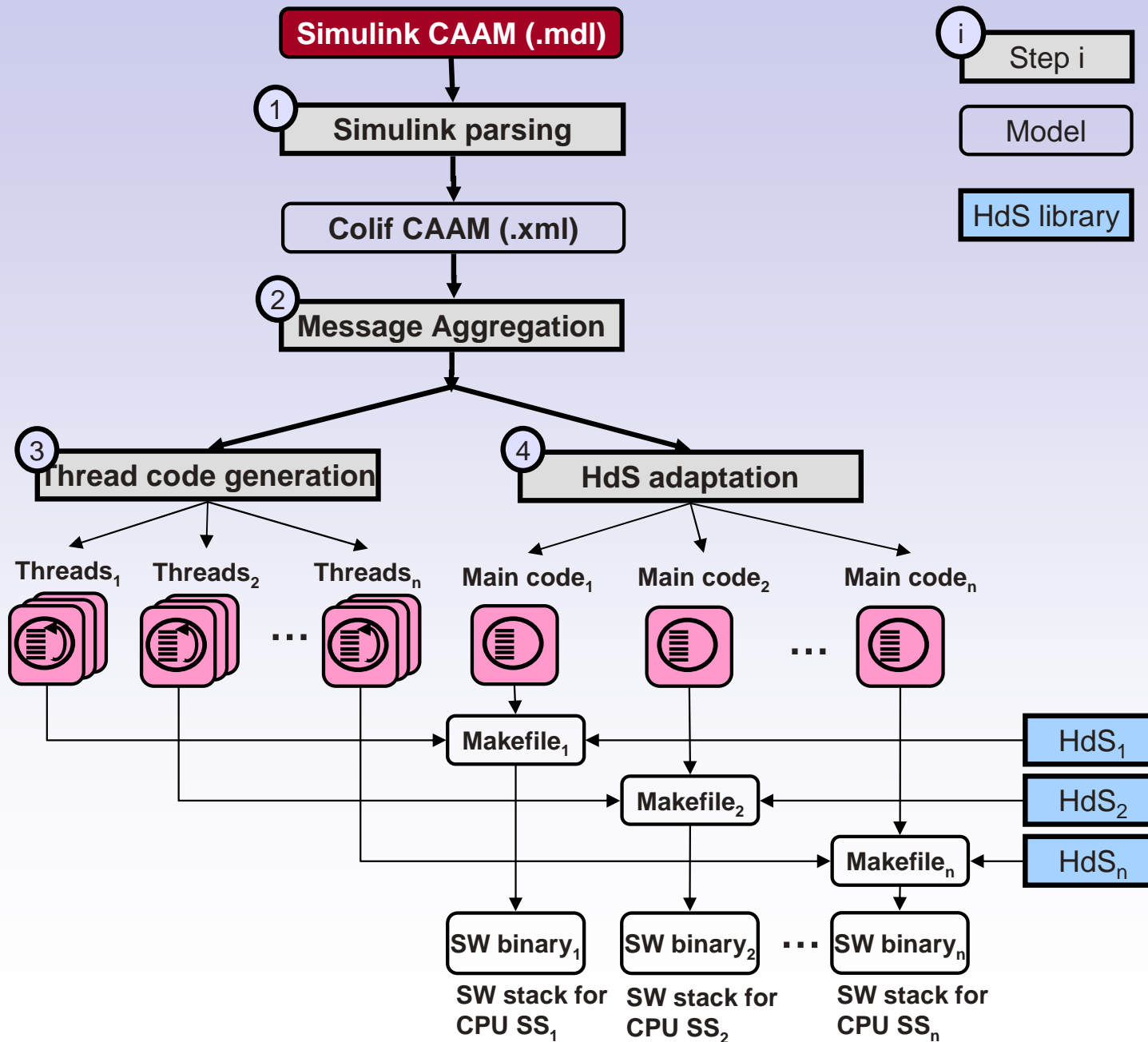
Code targeted to MPSoC arch.

1745 code lines (for an architecture with 2 CPUs)

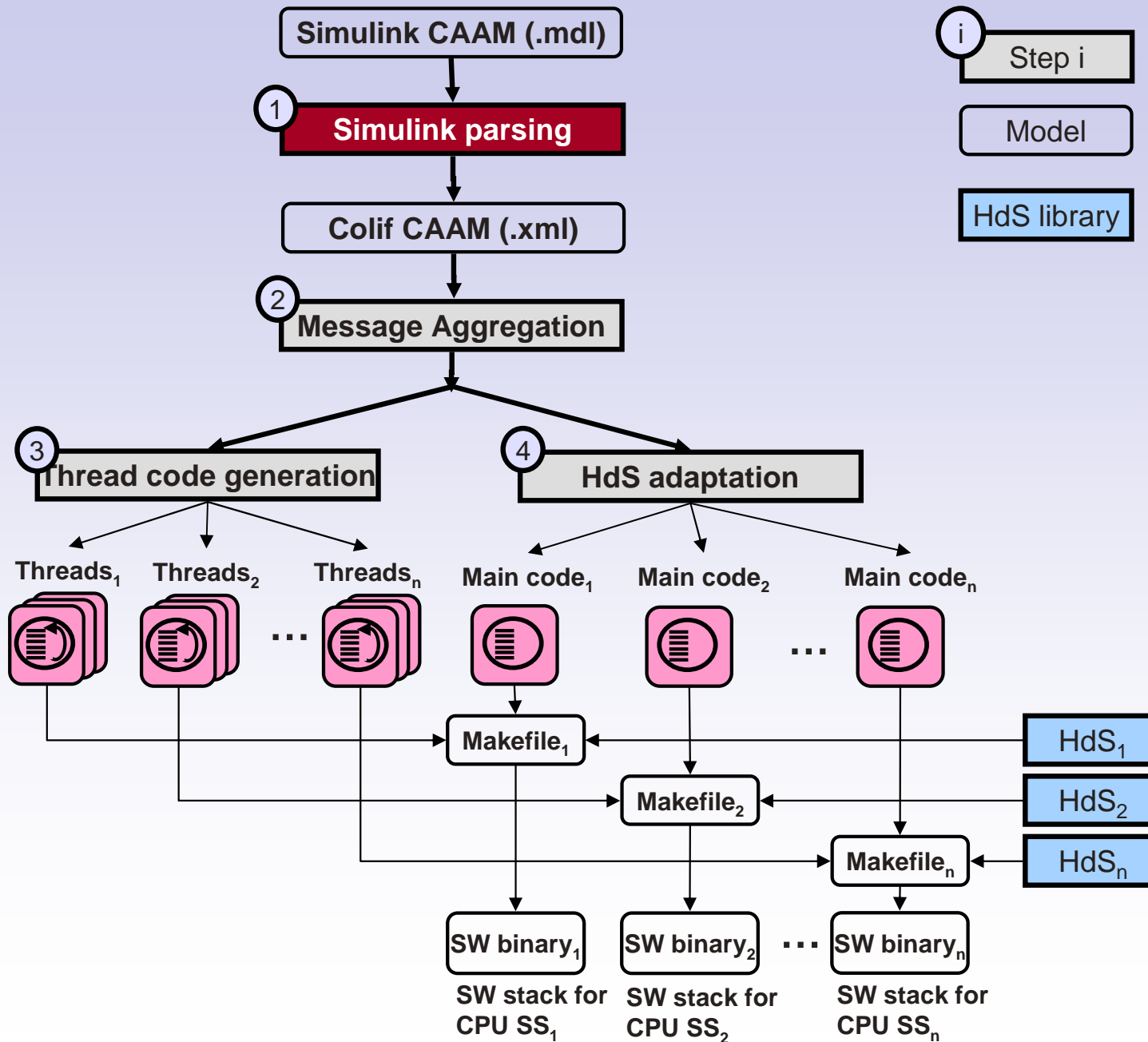
Multithread code generation flow



Multithread code generation flow



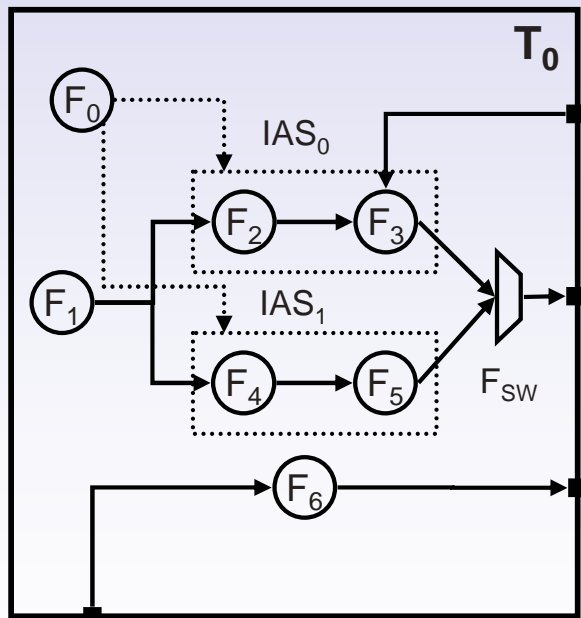
Multithread code generation flow



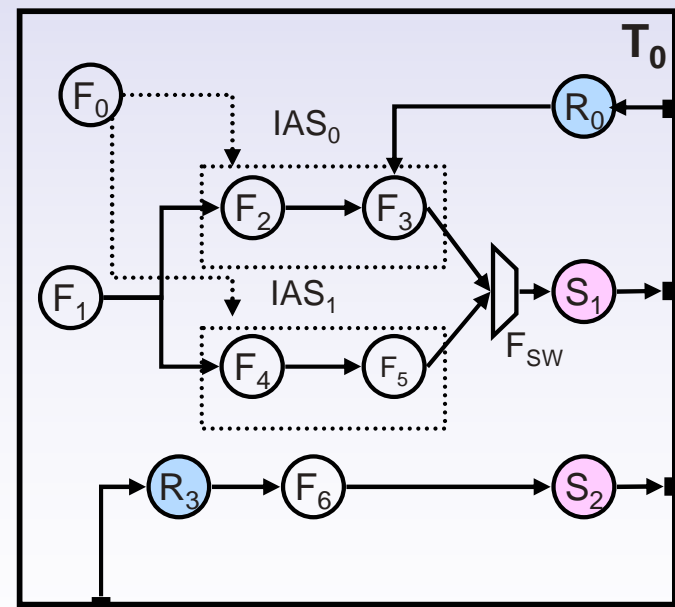
Parsing Simulink CAAM

- This step translates the Simulink model to a Colif model
- It inserts Send/Recv nodes to indicate communication

Thread-SS in a **Simulink CAAM**

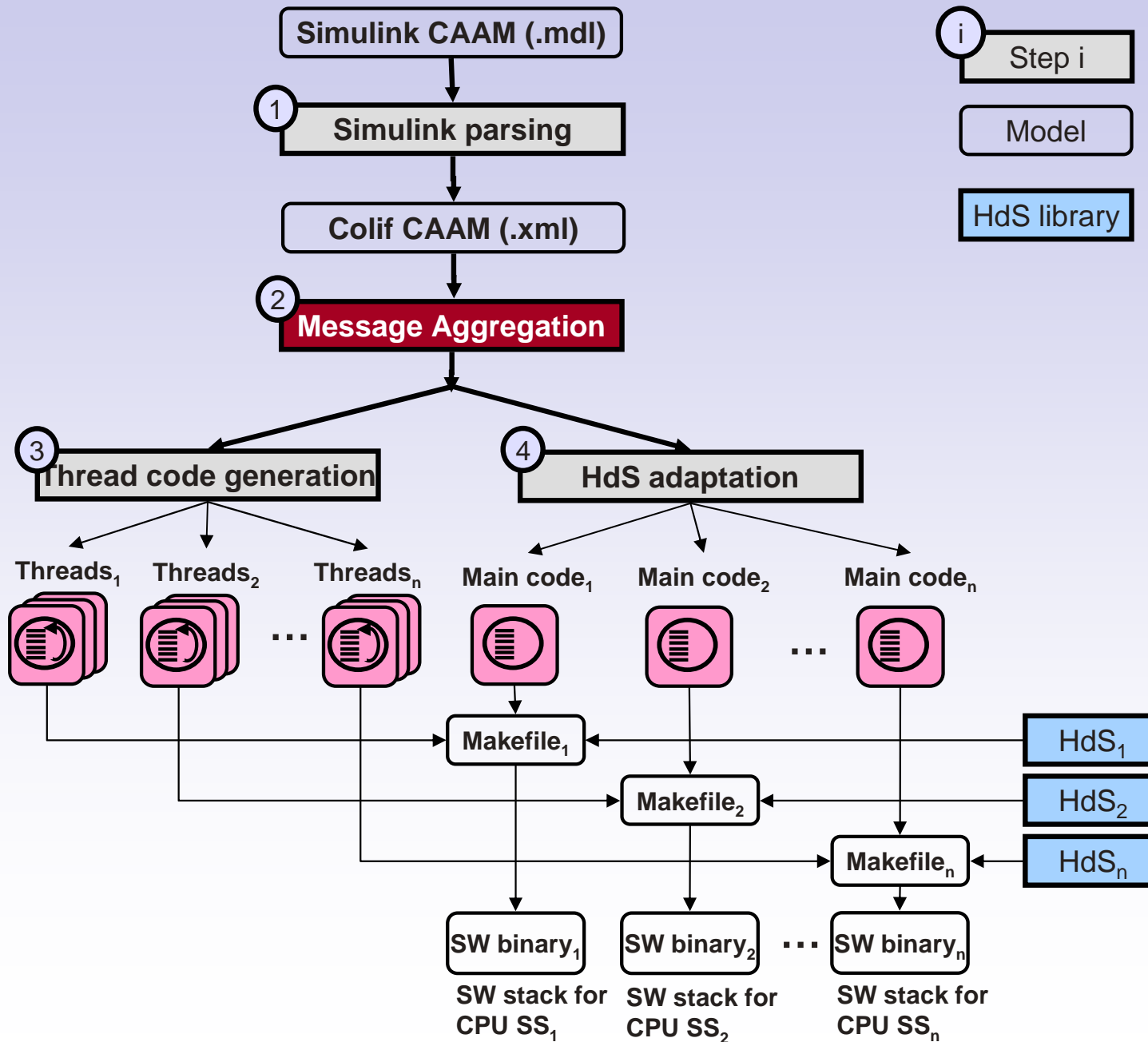


Thread-SS in a **Colif CAAM**



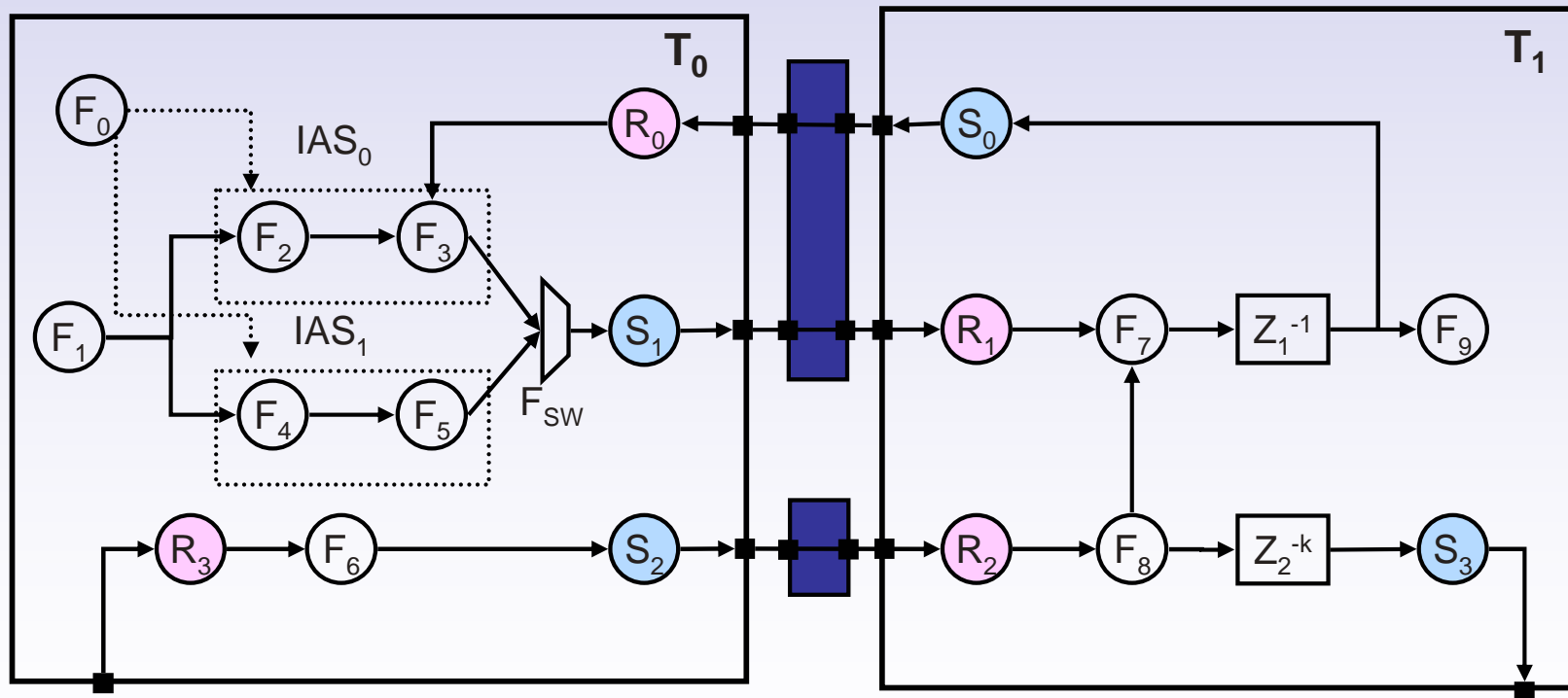
Send and **Recv** nodes are mapped to HdS communication primitives during code generation

Multithread code generation flow



Message Aggregation

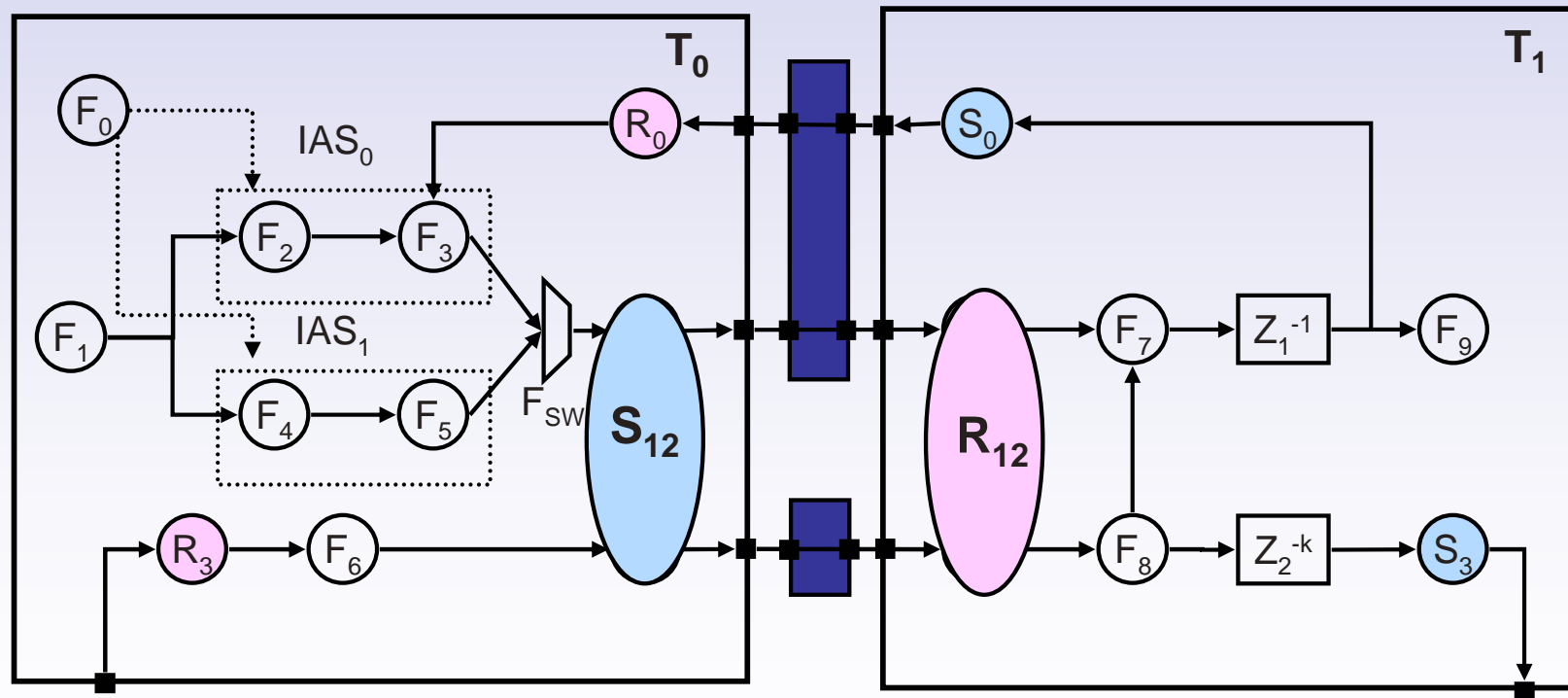
Merging *Send* and *Recv* nodes with the same source and destination



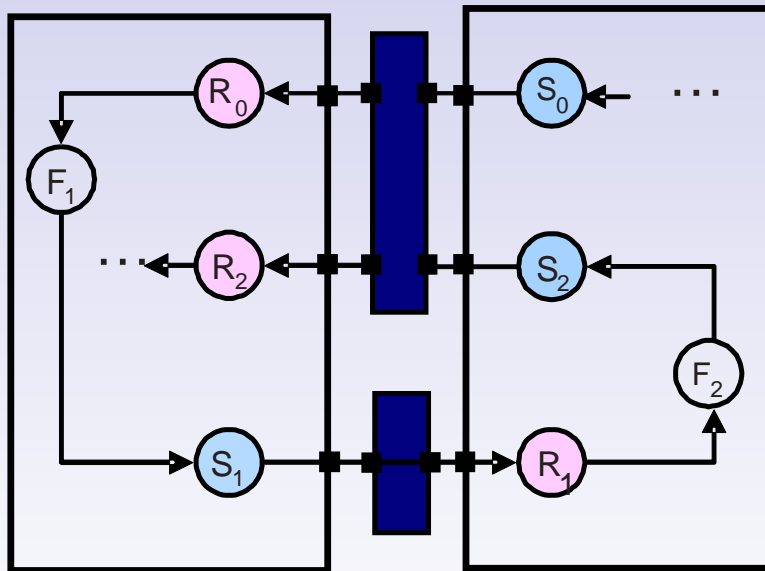
Message Aggregation

Merging *Send* and *Recv* nodes with the same source and destination

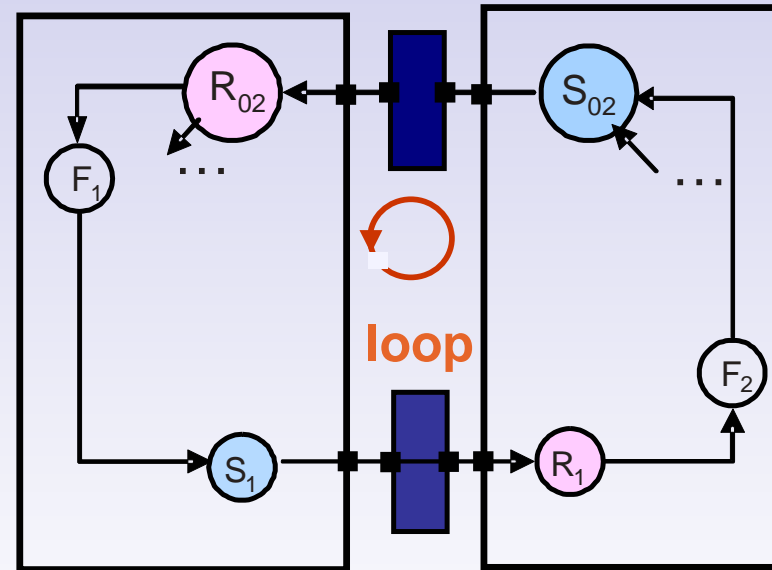
S_1 and S_2 from T_0 have T_1 as destination, so they are merged



Message Aggregation: Deadlock problem



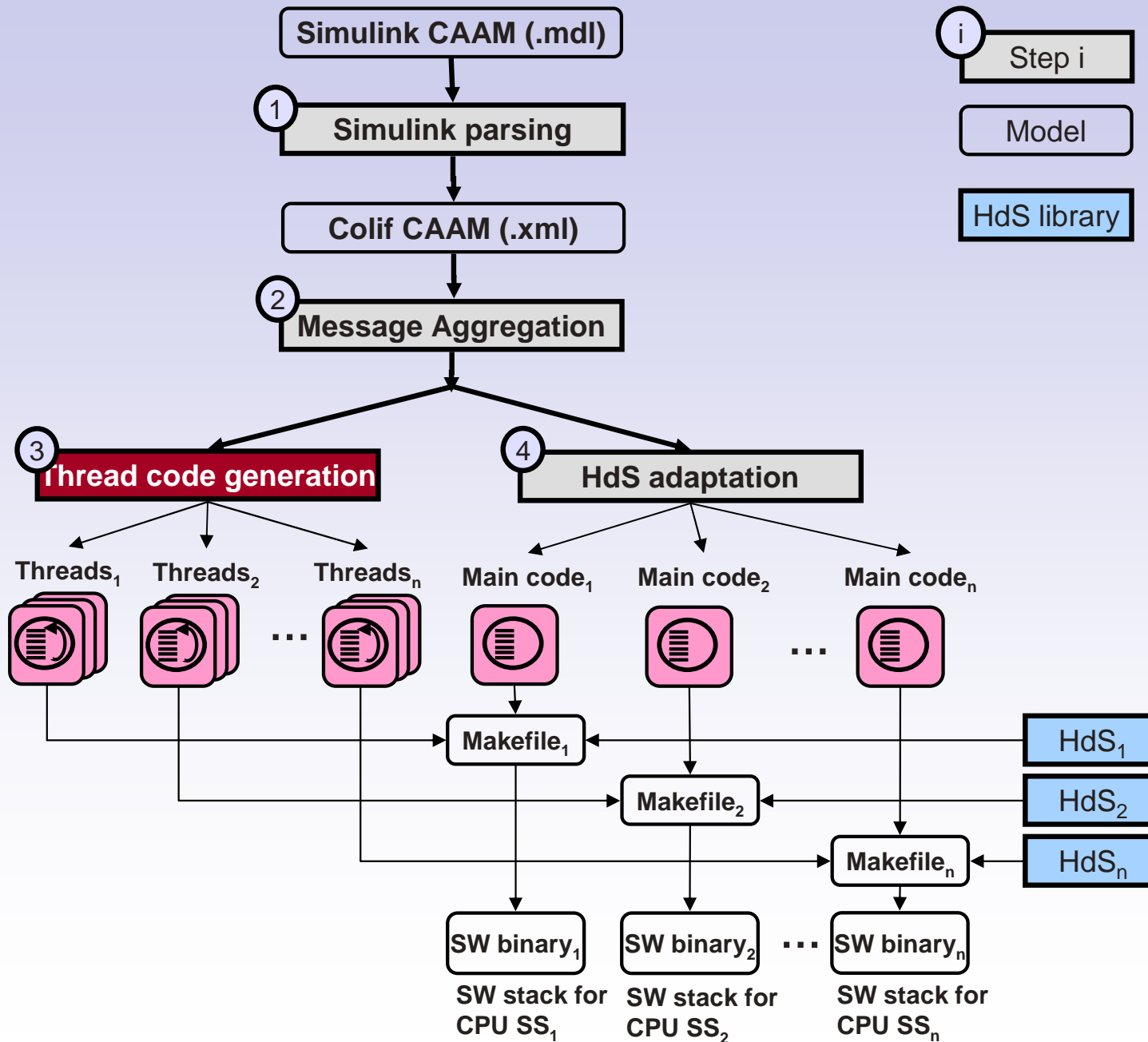
(a) A Colif CAAM



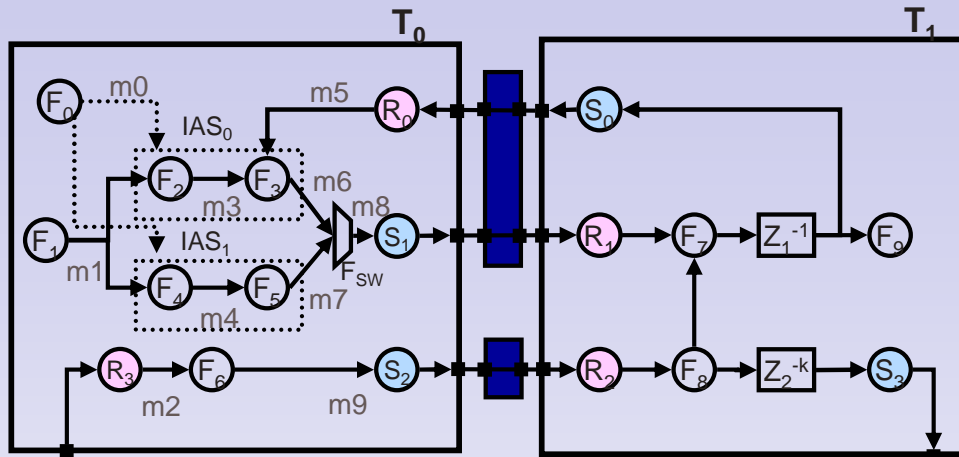
(b) A Colif CAAM with deadlock

To avoid deadlock, nodes are merged only when all of them have no precedent data dependency

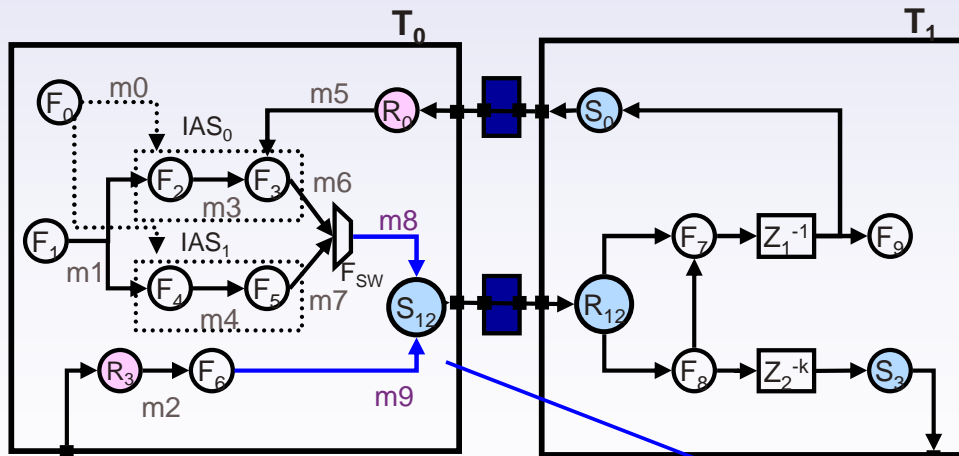
Multithread code generation flow



Thread code generator



(a) Colif CAAM



(b) Colif CAAM after MA

T₀ Code **without** MA

```

1: char m0[1]; int m1[4]; 2: // decl m2,m3,m4...
2: int m8[4]; int m9[8];
3: T0 ( ) {
4: while (1){
5:   F0 (m0); F1 (m1);
6:   recv (m5,8); //R0
7:   if (m0) {
8:     F2(m1,m3); F3(m3,m5,m6); m8=m6;
9:   else
10:    F4(m1,m4); F5(m4,m7); m8=m7;
11:   recv (m2,32); F6(m2, m9);
12:   send ( m8,4); //S1
13:   send ( m9,32); //S2
14: } }

```

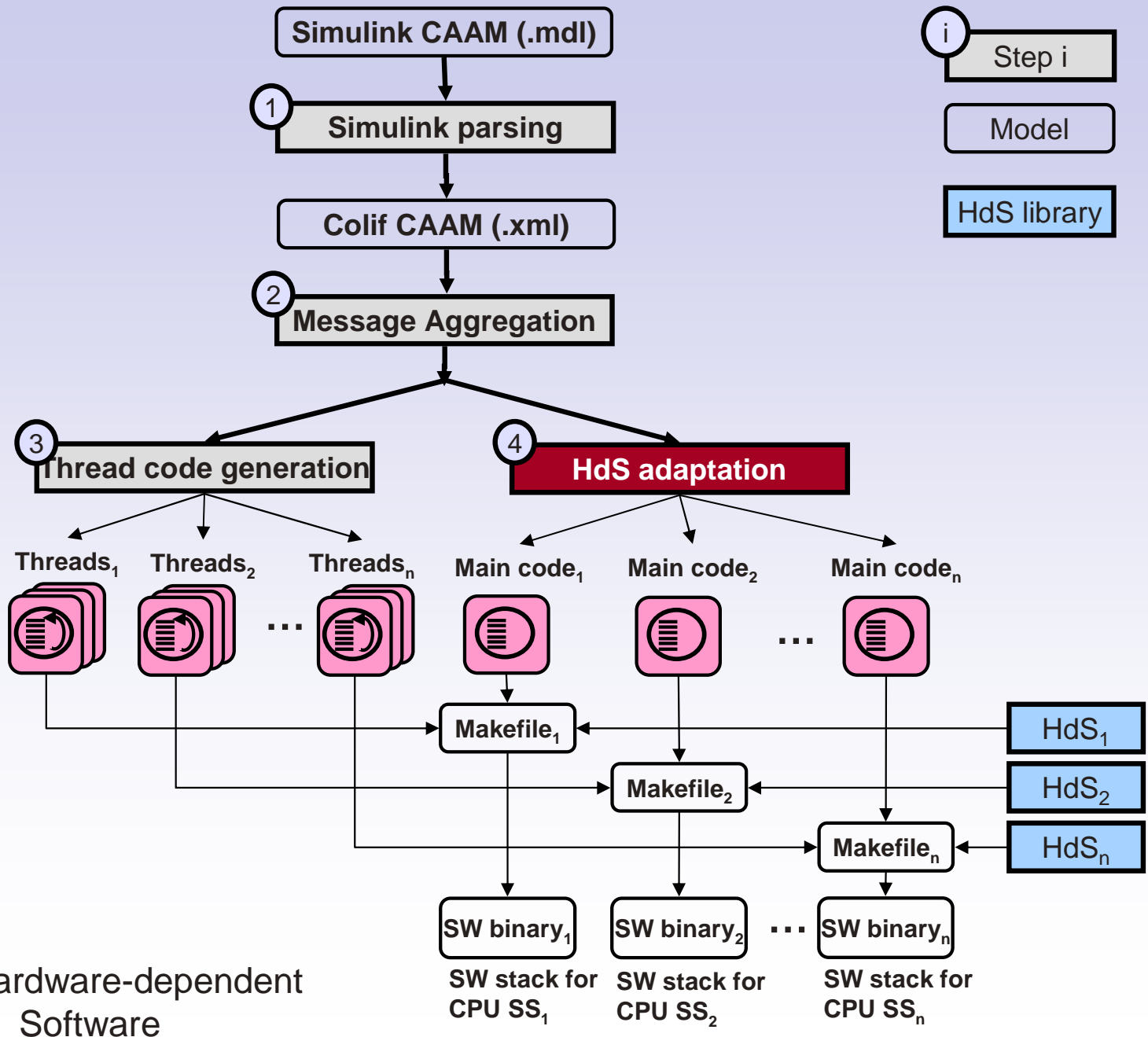
T₀ Code **with** MA

```

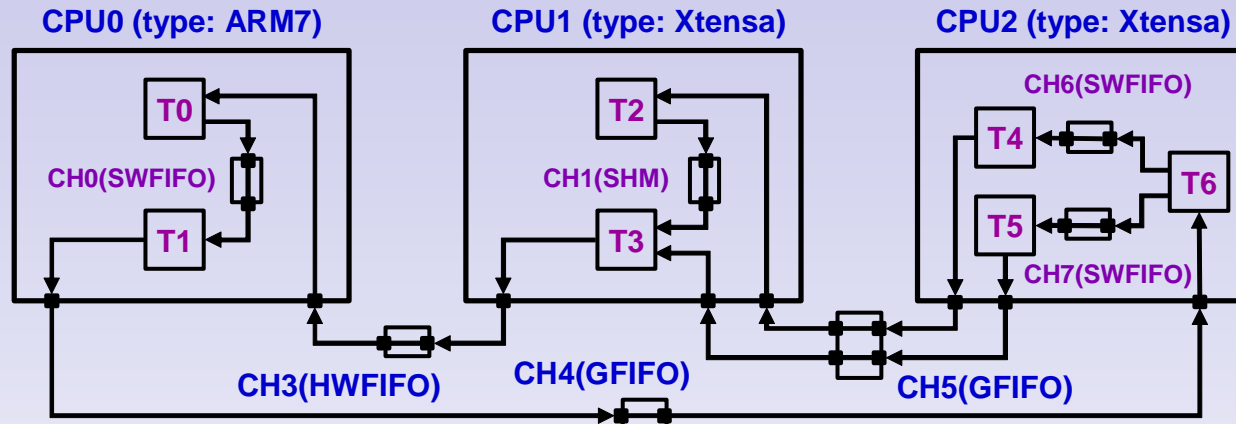
1: char m0[1]; int m1[4]; 2: // decl m2,m3,m4,...
2: struct {int m8[4]; int m9[8]; } m10;
3: T0 ( ) {
4: while (1){
5:   ...
6:   recv (m5,8); //R0
7:   if (m0){
8:     F2(m1,m3); F3(m3,m5,m6); m10.m8=m6;
9:   else
10:    F4(m1,m4); F5(m4,m7); m10.m8=m7;
11:   recv (m2,32); F6(m2, m10.m9);
12:   send (m10,36 ); // S12
13: } }

```

Multithread code generation flow



Main and Makefile code generation



(a) An example of Simulink CAAM

```

1: channel_t *ch4, *ch5, *ch6, *ch7;
2: port_t *p4, p5, p6, p7;
3: void main( ) {
4:   ISR_attach(0, gfifo_isr);
5:   ...
6:   channel_init(&ch6, SWFIFO,...);
7:   port_init(&p6, &ch6, ...);
8:   ...
9:   thread_create(T4, ...);
10:  thread_create(T5, ...);
11:  ... }

```

(b) Main code for CPU2

```

1: CC=xt-xcc // Xtensa-compiler
2: ...
3: SRCS= T4.c T5.c T6.c main.c
4: ...
5: LIBS= libhds-xt.a
6: ...
7: sw.bin: $(OBSJ) $(LIBS)
8: $(CC) -o sw.bin $(OBSJ) $(LIBS)

```

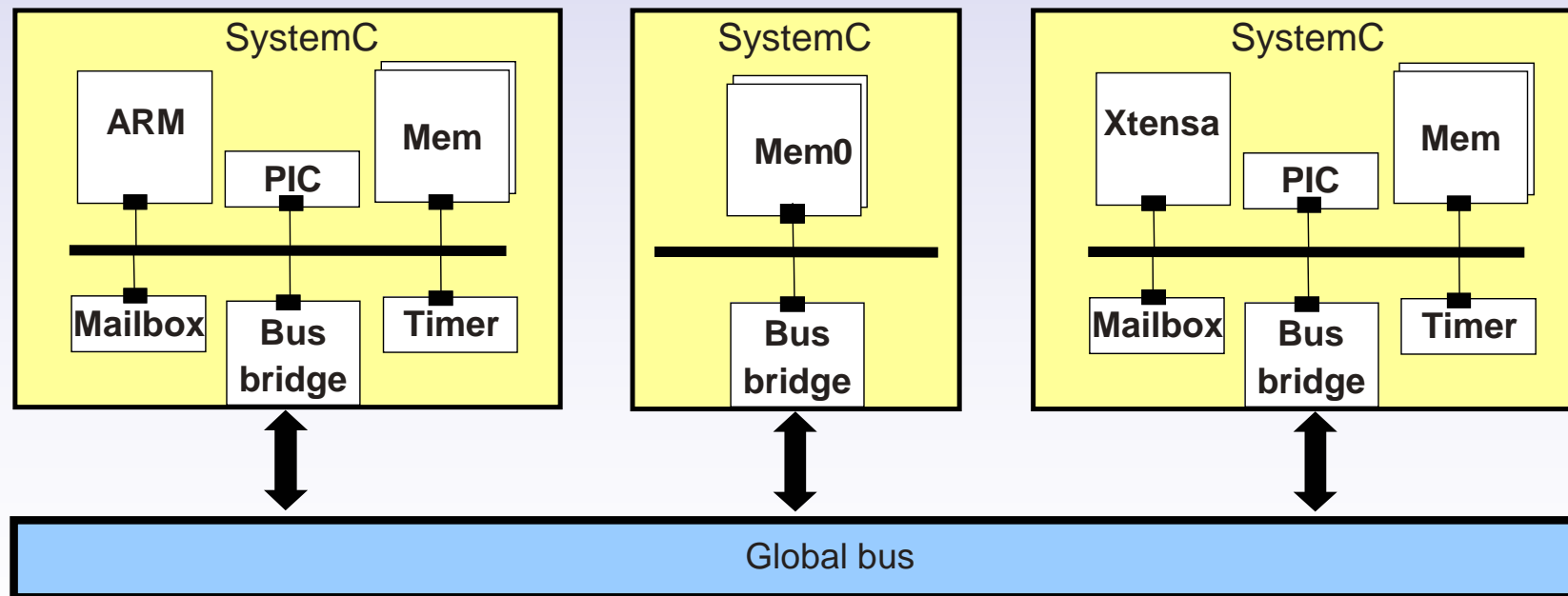
(c) Makefile for CPU2

Case Study

- Application: H-264 video decoder
- Experiment:
 - From the H264 Simulink CAAM, we generated code
 - Evaluation of the impact of MA in terms of:
 - Required communication channels
 - Memory size
 - Execution time
 - Considering different task partitioning (2, 3, 4, 5, and 6 CPUs)

Case Study: Multiprocessor Platform

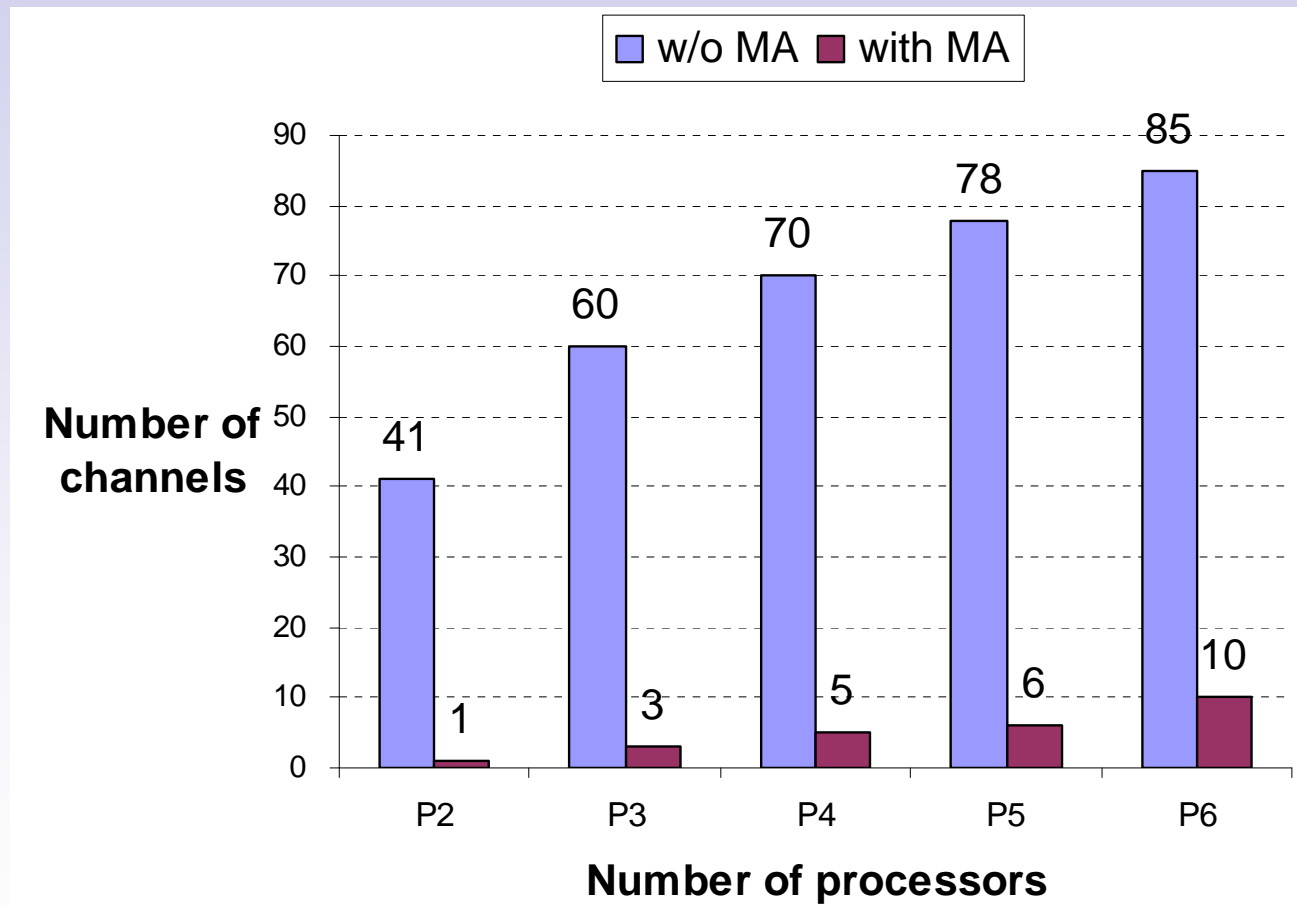
- n CPUs (processor, local mem., Mailbox, bridge, timer, and PIC)
- a Global Bus
- a Global Memory



PIC: Programmable Interrupt Controller

Impact on the number of channels

MA achieved a **reduction around 90%** on the number of required channels



Communication time

Message Aggregation reduces time spent with communication

with MA			w/o MA		
comp	comm	idle	comp	comm	idle
0.763	0.037	0.199	0.705	0.125	0.170
0.590	0.053	0.357	0.550	0.147	0.303
0.642	0.074	0.284	0.587	0.186	0.227
0.576	0.074	0.349	0.499	0.174	0.326
0.448	0.074	0.478	0.486	0.192	0.442



communication time per 1 second

Message Aggregation decreased the time spend with communication

Communication speed

Communication Speed in Bytes/cycle

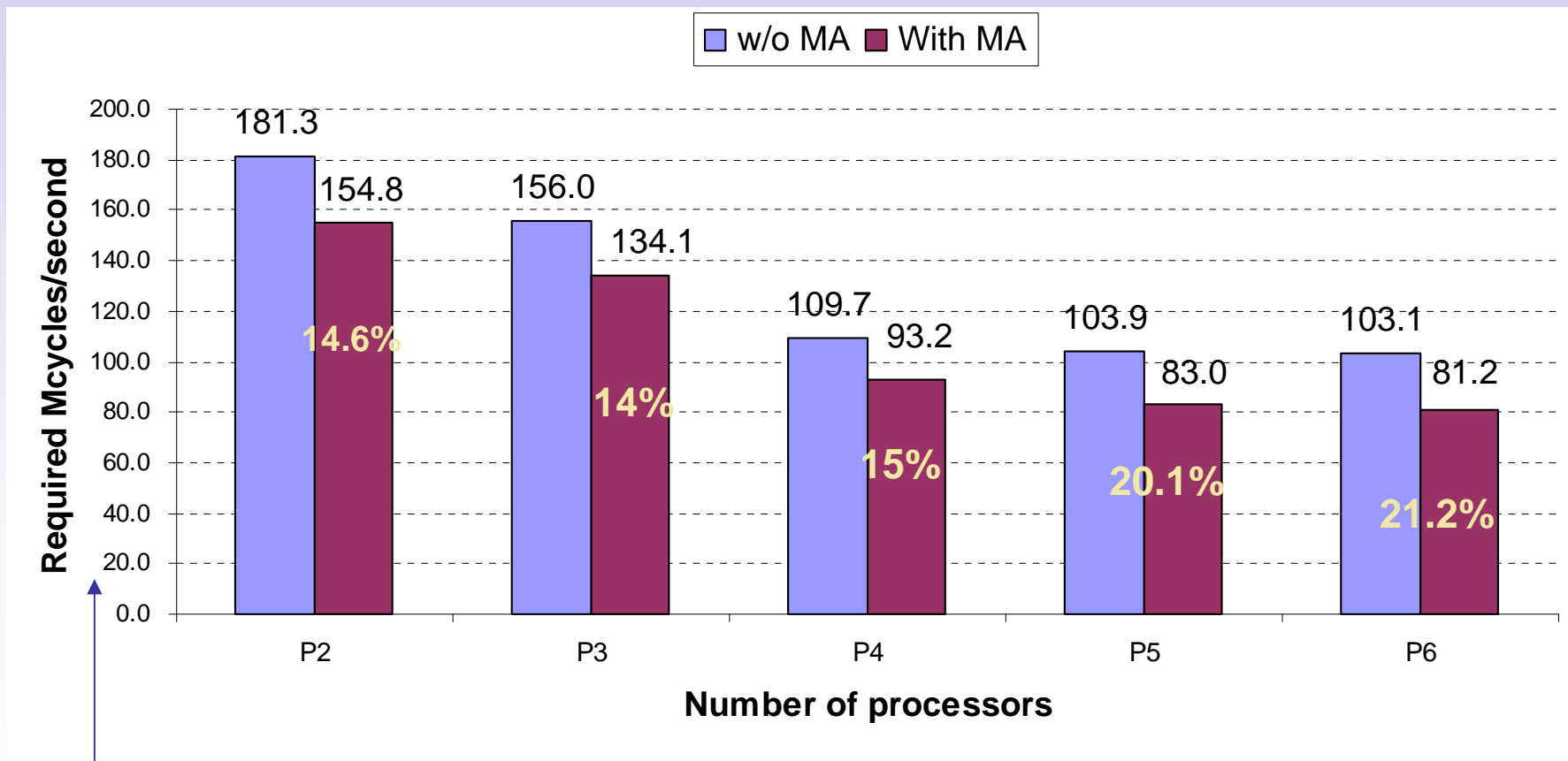
	with MA	w/o MA
P2	0.65	0.17
P3	0.56	0.17
P4	0.57	0.19
P5	0.56	0.19
P6	0.49	0.15

Average for 1 cycle

Message Aggregation accelerated the communication

Performance results

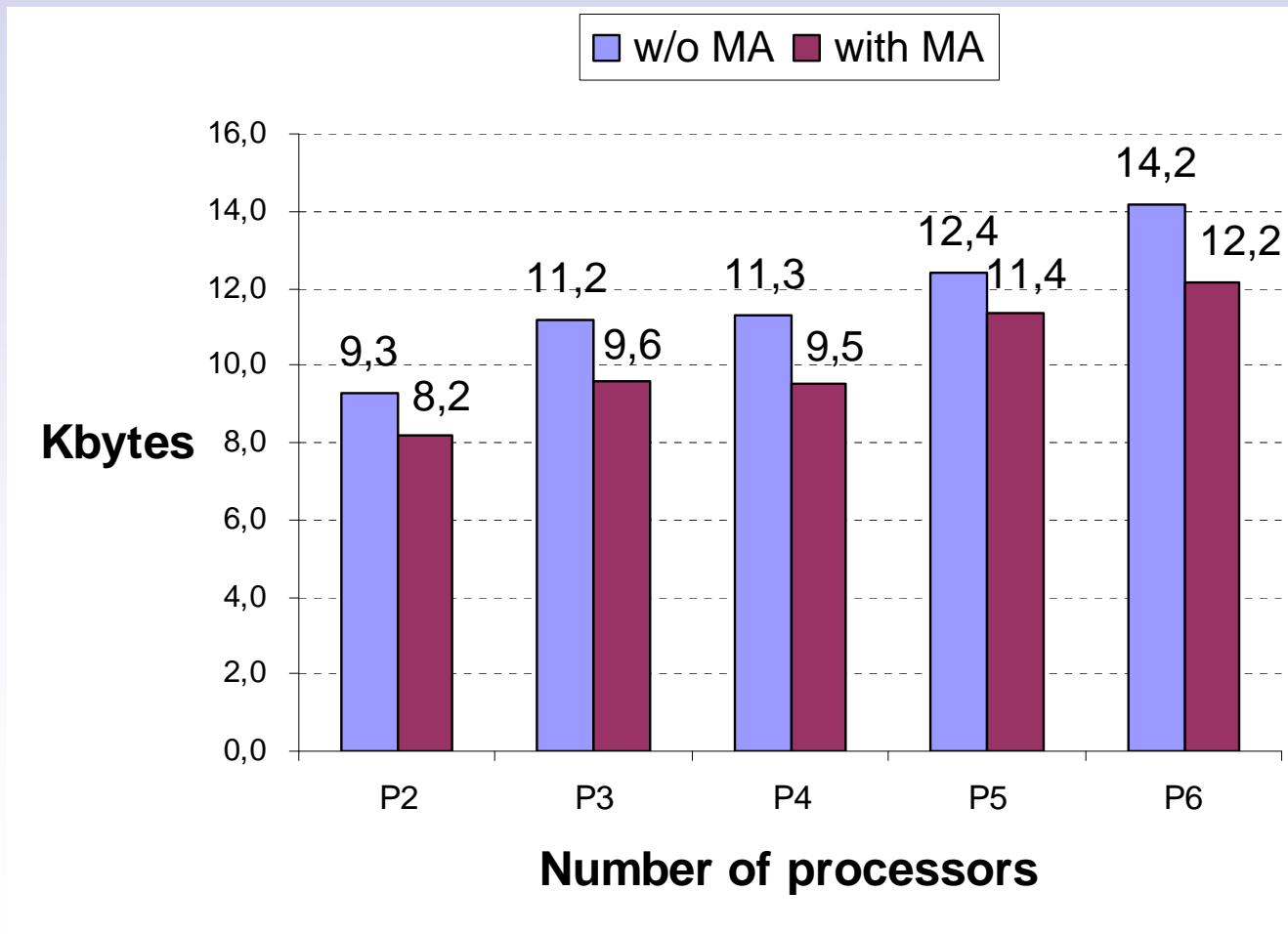
MA achieved improvements on performance **from 14% to 21%**



Consumed Mcycles/s to decode H.264 QCIF 30 fr/sec

Data memory results

MA reduces the required SW communication infrastructure reducing data memory size



Reductions around 15% on the data memory size have been obtained by MA

Conclusions

- This approach may reduce:
 - synchronization costs
 - required communication channels
- **Message Aggregation** achieved improvements:
 - 20% for performance
 - 14% for data memory size
 - 4% for code size

Future works

- **Aspects to be investigated:**
 - Impact on real-time response
 - Message latency potential increase
- A global scheduling policy that consider (in the same time):
 - communication overhead reduction
 - message latency

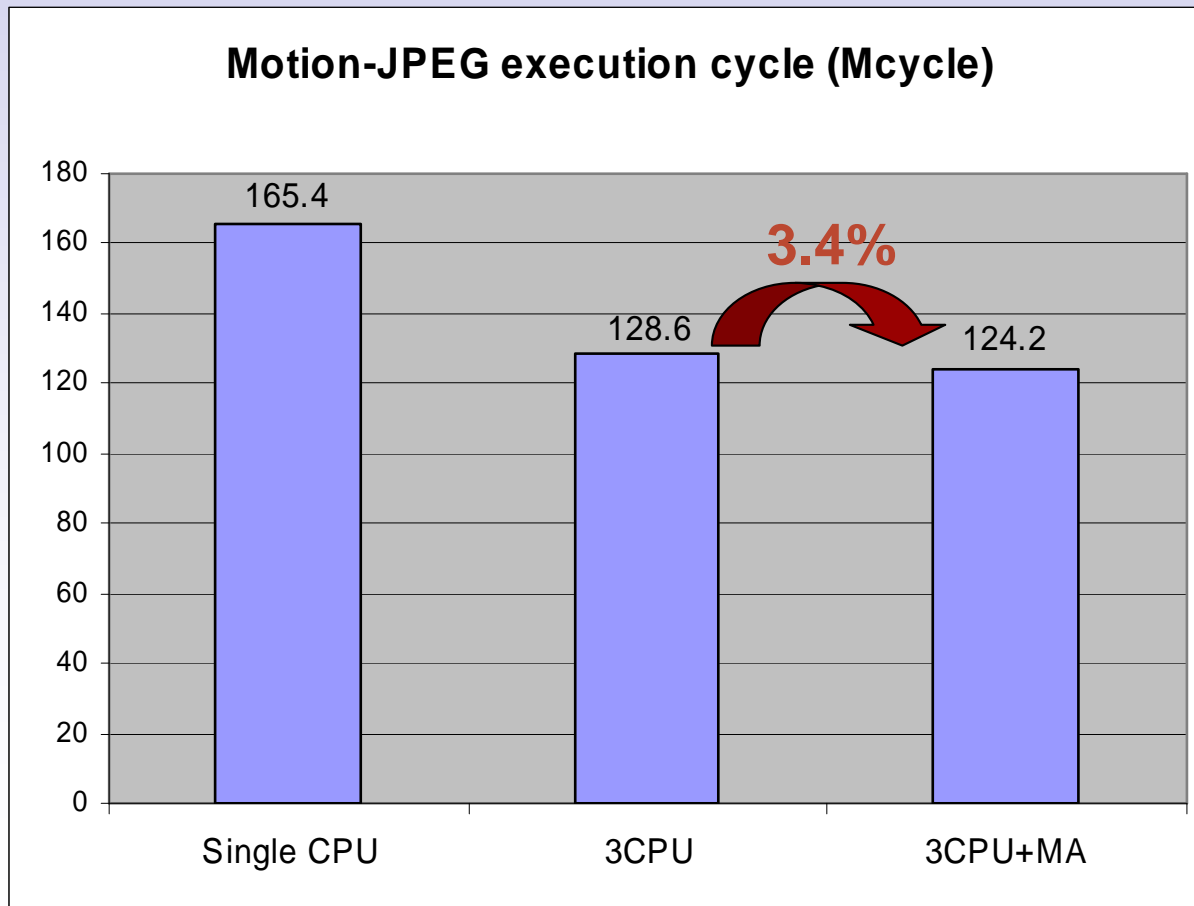
Thank You !

Questions ?

Contact: lisane@inf.ufrgs.br

MJPEG results

3.4% of performance improvement was achieved with MA.



This small improvement is because has a very small number of channels