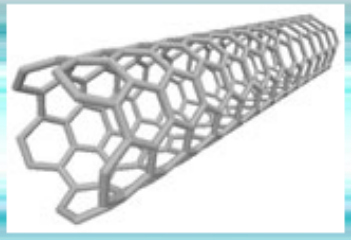
TIMA LABORATORY

Techniques of Informatics and Microelectronics for integrated systems Architecture



Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation

Marius Gligor, Nicolas Fournel and Frédéric Pétrot

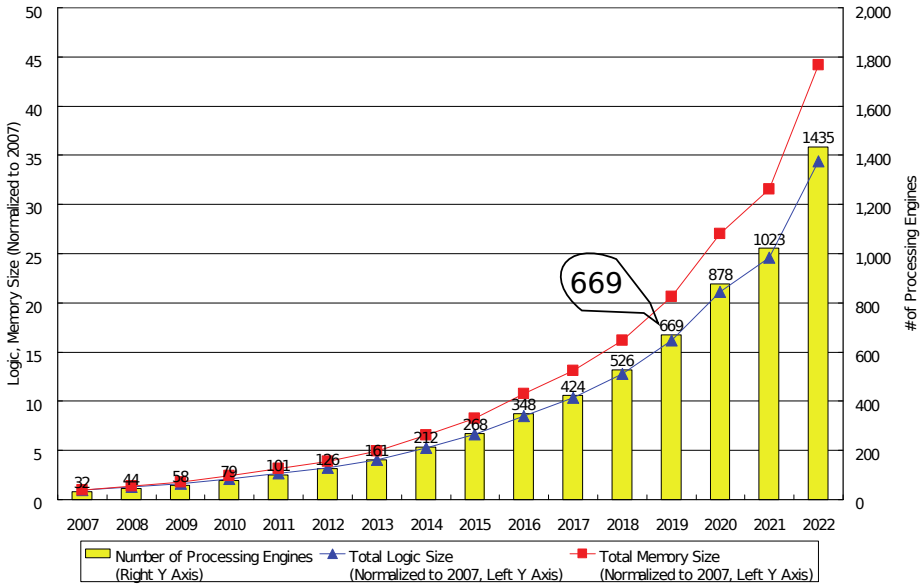
TIMA Laboratory - SLS Group, INP-UJF-CNRS
Grenoble, France

October 18, 2010

Outline

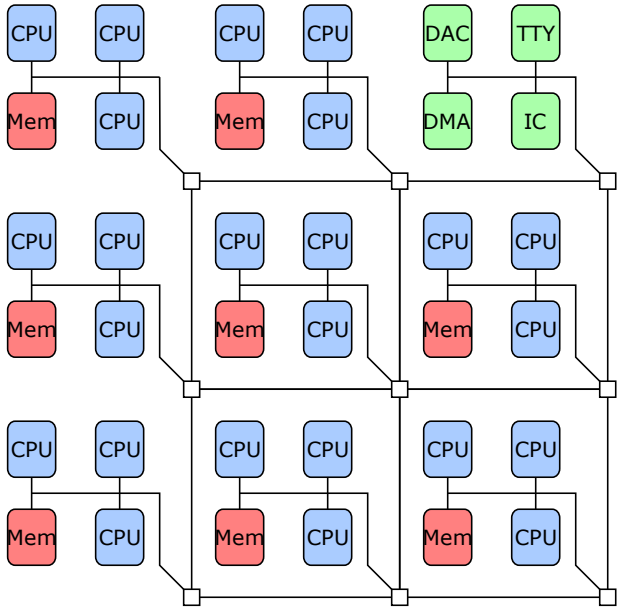
MPSoC Perspectives

ITRS Roadmap for the number of cores in consumer devices



Characteristics of future SoCs architecture

- More and more homogeneous
- Massively parallel
- Programmability is a major concern



Problems

MPSoCs based on GPPs

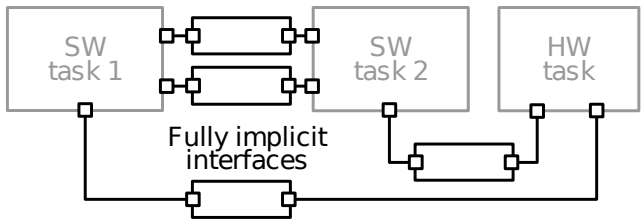
- Great flexibility, short time-to-market
- Great reusability
  - Use GPP and legacy code (*e.g.* OS)

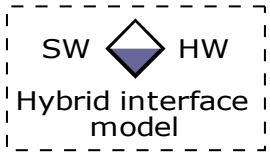
System simulation

- Architecture validation and early software coding
- Optimize hardware/software performances and consumption through architecture exploration
- Speed/accuracy trade-off required
  - Several abstraction levels

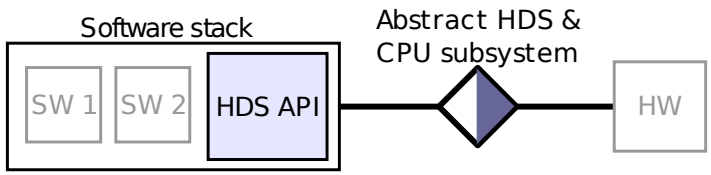
Abstraction Levels

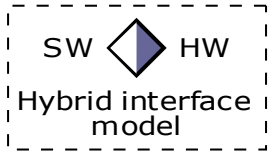
System



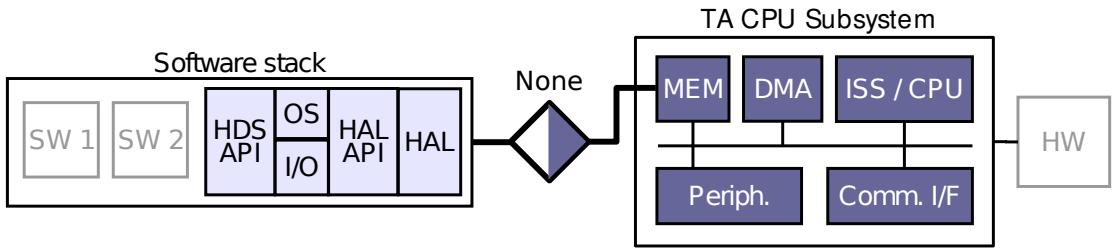


Virtual architecture

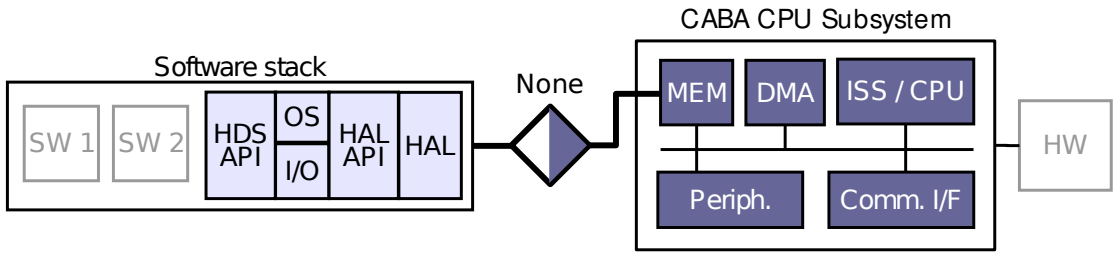




Transaction accurate

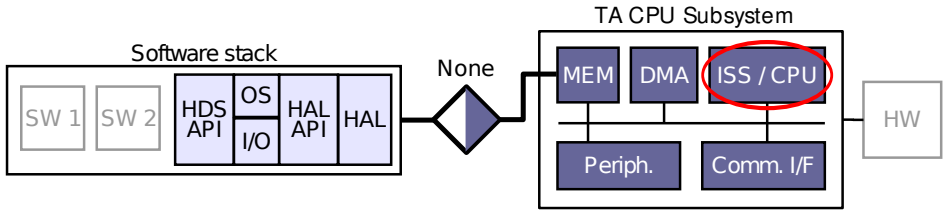


Cycle approximate / accurate



Motivations

Classical TA solution: interpretive ISS



- Pros: flexible and precise
- Cons: low simulation speed

Binary translation based ISS

- Pros: fast
- Cons: no time notion for the simulated platform

Solution: Transform a binary translation based ISS into an accurate component that can be used in an event-driven simulation

Possible candidates

- Binary translation based ISS: **QEMU**, Bochs, ...
- Event driven simulator: **SystemC** is the current solution for MPSoC simulation

Outline

QEMU / SystemC TLM

QEMU

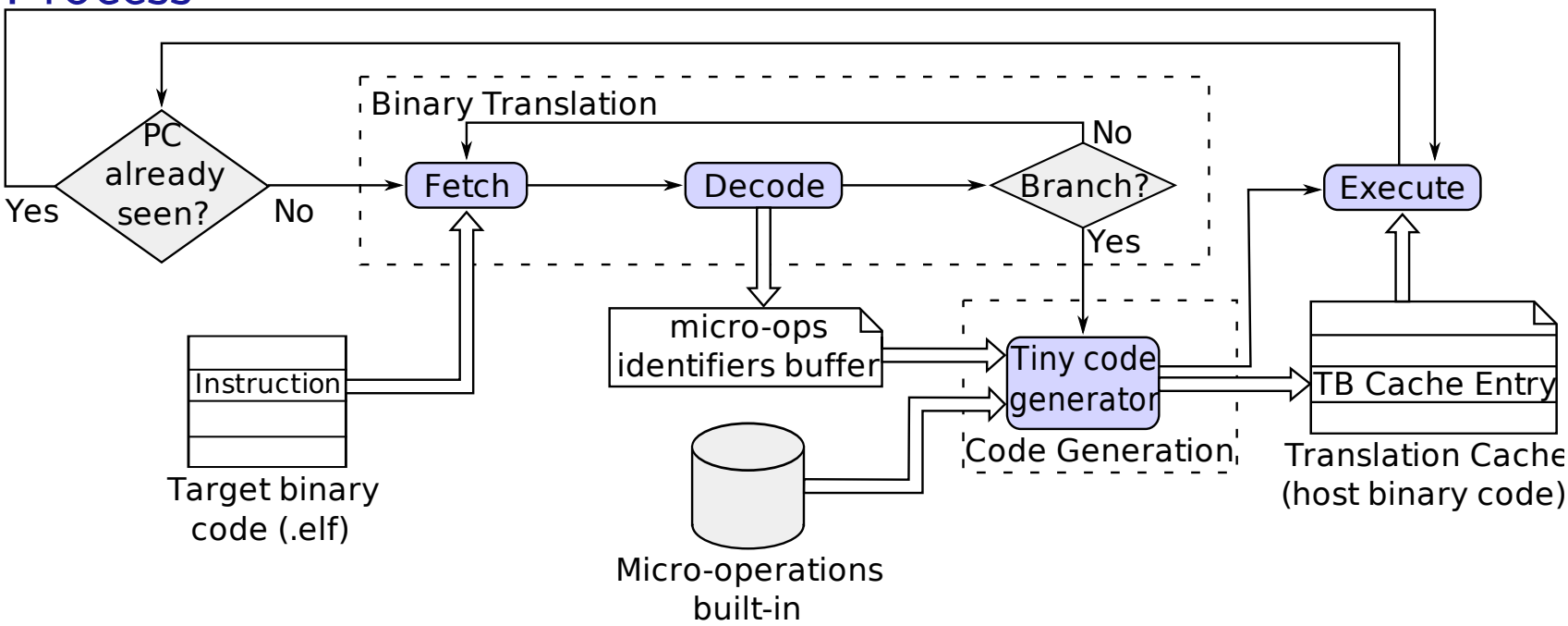
- Fast and portable emulator
- Emulates multiple target architectures (*e.g.* x86, ARM, SPARC) on multiple host architectures
- Based on dynamic binary translation and dynamic code generation
- 5 to 20 times slower than native code execution

SystemC TLM

- Use of transactions for data exachange
- Discrete event-driven simulation
  - Concurrency of processes

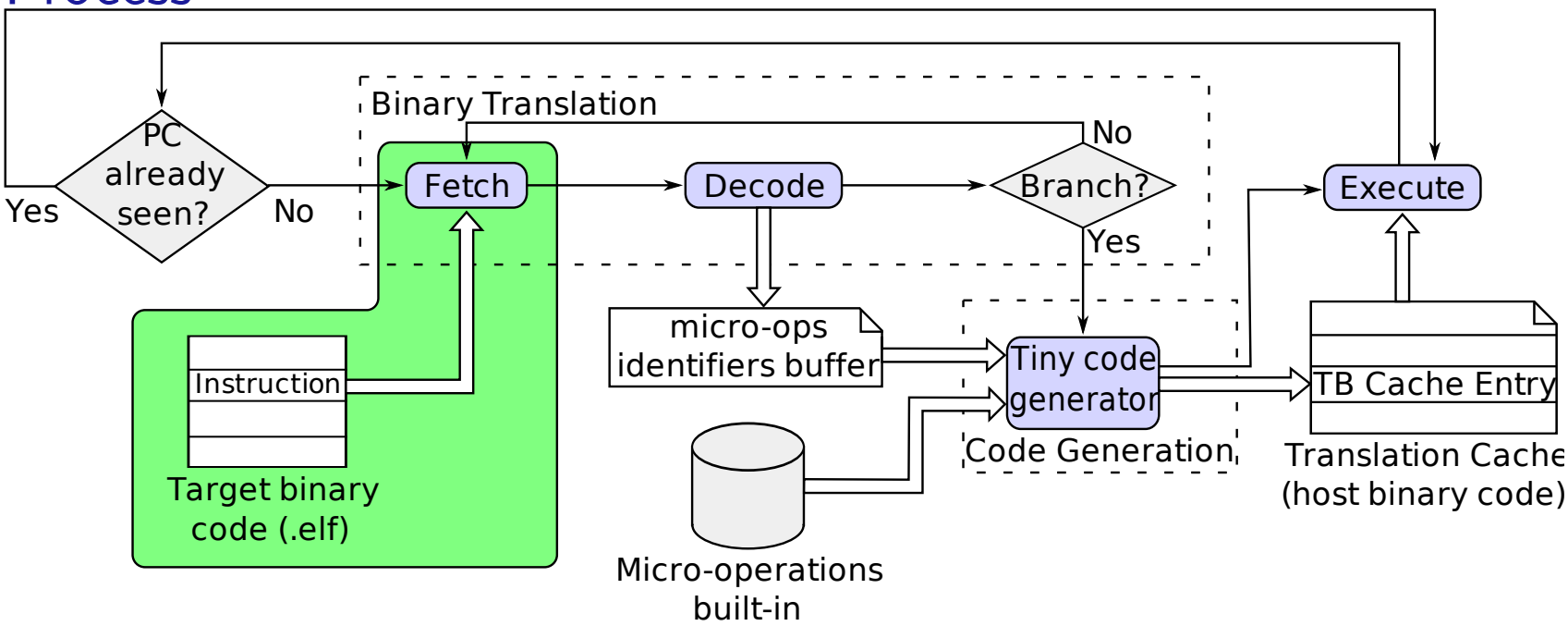
QEMU Emulation Process

Process



QEMU Emulation Process

Process

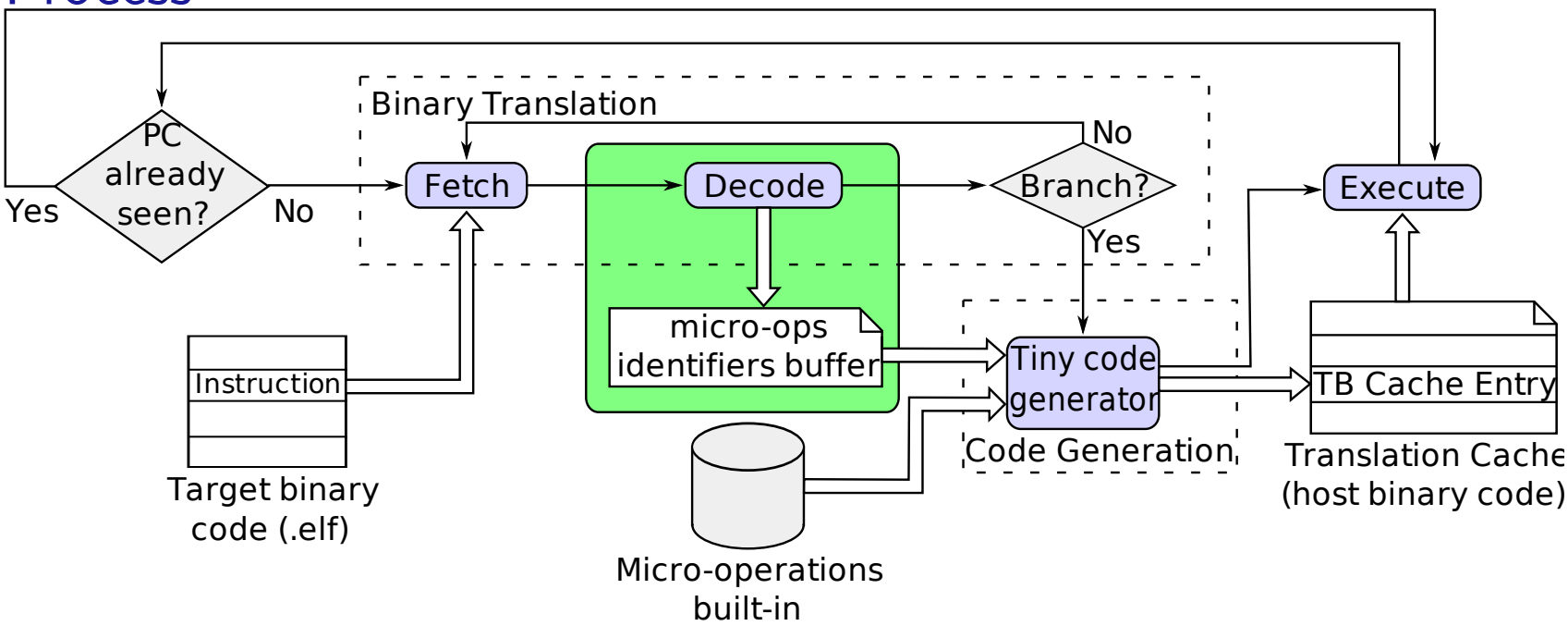


Code generation example

```
18    target_instrX
```

QEMU Emulation Process

Process



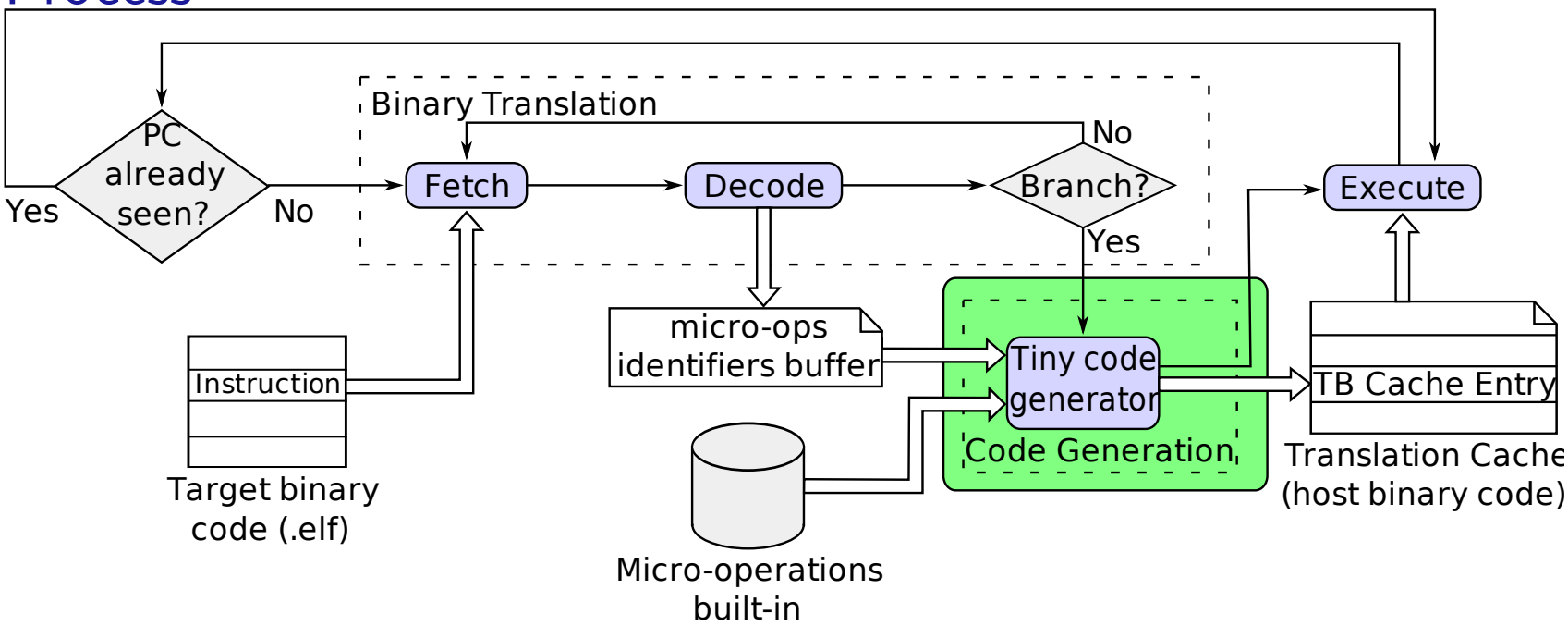
Code generation example

```
18   target_instrX          micro-op1_instrX

                            micro-op2_instrX
```

QEMU Emulation Process

Process



Code generation example

18	target_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
			host_instr2_micro-op1_instrX
			host_instr3_micro-op1_instrX
		micro-op2_instrX	host_instr1_micro-op2_instrX

ISS Wrapping and Connection

ISS SystemC wrapper

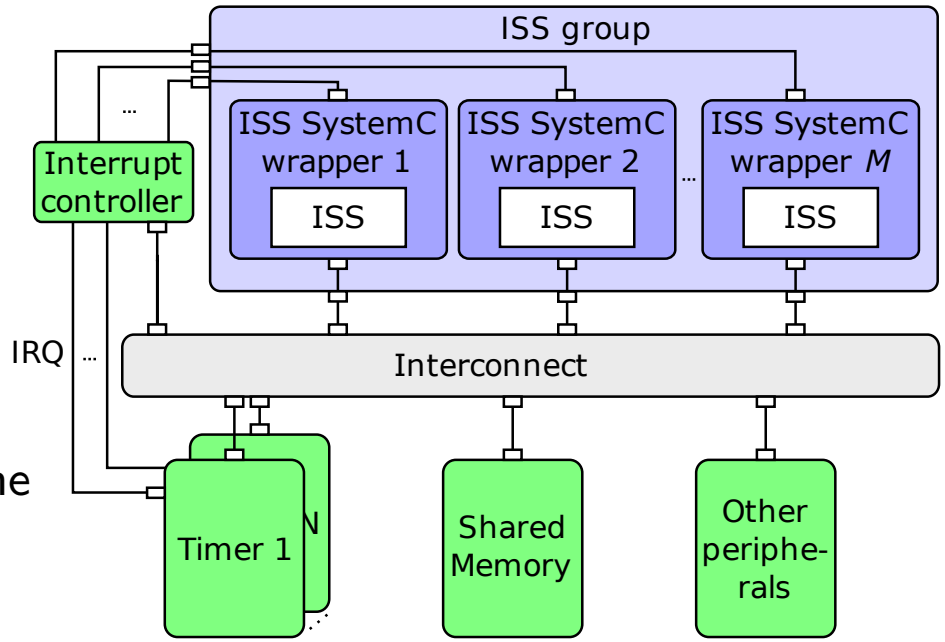
- Simulates independently under the SystemC control in the context of a SystemC thread
- Connected to interconnect
- Implements instruction and data caches
- A SystemC thread - interface between SystemC interrupt signals and ISS

ISS group

- Groups the processors that may share the same translation cache
- Identical processors

SystemC timed TLM components

- Traffic generator, timers, main memory, spinlocks, interconnect, RAMDAC, TTYs



ISS Wrapper/SystemC Synchronization

ISS component must synchronize with SystemC

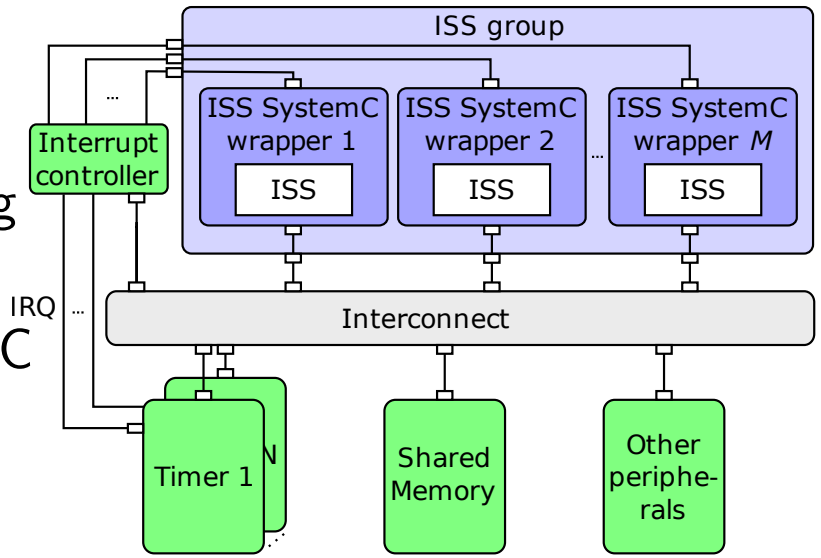
- Consume the time corresponding to the simulated cycles

Synchronization points

- Cache misses (instruction and data caches)
- I/O operations
- Target synchronization instructions (*e.g.* load and store exclusive)
- Predefined period without synchronization

Interrupts forwarding

- Generated by hardware components during the SystemC activities of the processors
- Interrupt treated after the current SystemC activity ends, at the beginning of the next translation block



Code Annotation

Motivation

- Generated code offers no time information about the target execution
- Accurate time modeling of the ISS

Insert micro-operations

- To increment the number of simulated cycles
  - Inserted before the micro-operations of each target instructions
  - Use the target processor datasheet
  - Take into account registers dependencies, branch prediction
- To emulate target caches (instruction and data) and write buffer

Annotation example

Instr address	Target code	Original translation	Annotated translation	Annotated generated code
addr_instrX	target_instrX	micro-op1_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
				host_instr2_micro-op1_instrX
				host_instr3_micro-op1_instrX
		micro-op2_instrX	micro-op_annotation	host_instr1_micro-op_annotation
				host_instr2_micro-op_annotation
			micro-op2_instrX	host_instr1_micro-op2_instrX

Code Annotation: Details & Example

Instruction Cache

- Where?
  - At the beginning of each translation block
  - At the beginning of each cache line
- What?
  - Synchronize simulated cycles
  - Request over the interconnect

Data cache

- Where?
  - At each data access (read and write)
- What?
  - On read miss: synchronize and fill the cache line using the interconnect
  - On write hit: update the value in cache
  - On write: update the value in memory through interconnect (write through policy)

	Instr address	Target code	Original translation	Annotated translation
start_tb:	18	instr1_reg_operation	micro-op1_for_instr1 ..... micro-opN1_for_instr1	instr_cache_verify (18); nb_cycles += cpu_datasheet [instr1]; micro-op1_for_instr1 ..... micro-opN1_for_instr1
	1C	instr2_load_from_1000	micro-op1_for_instr2 ..... micro-opN2_for_instr2	nb_cycles += cpu_datasheet [instr2]; data_cache_verify (1000); micro-op1_for_instr2 ..... micro-opN2_for_instr2
	20	instr3_store_5_to_2000	micro-op1_for_instr3 ..... micro-opN3_for_instr3	instr_cache_verify (20); nb_cycles += cpu_datasheet [instr3]; write_access (2000, 5); micro-op1_for_instr3 ..... micro-opN3_for_instr3

Precision Levels

- Simulation speed/accuracy trade-off
- Caches full
  - Search data and instructions over the interconnect
  - Ignore instructions from cache (instructions are simulated from QEMU translation cache)

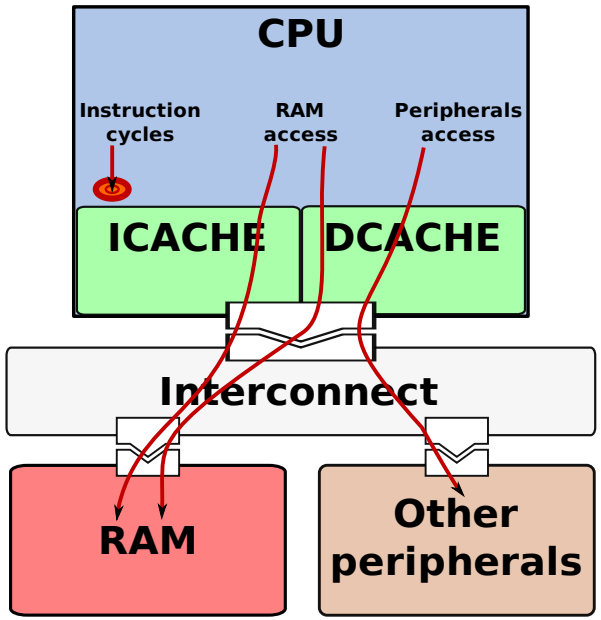


Figure: Cache full

Precision Levels

- Simulation speed/accuracy trade-off
- Caches full
  - Search data and instructions over the interconnect
  - Ignore instructions from cache (instructions are simulated from QEMU translation cache)
- No caches (0 memory access time)

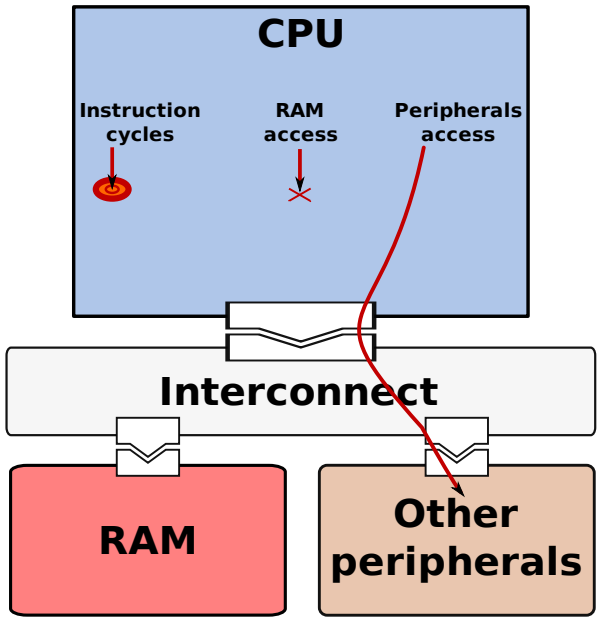


Figure: No cache

Precision Levels

- Simulation speed/accuracy trade-off
- Caches full
  - Search data and instructions over the interconnect
  - Ignore instructions from cache (instructions are simulated from QEMU translation cache)
- No caches (0 memory access time)
- Caches as pure directories
  - Precomputed time consumed
    - Cache wait: when the miss occurs
    - Cache late: at the next synchronization

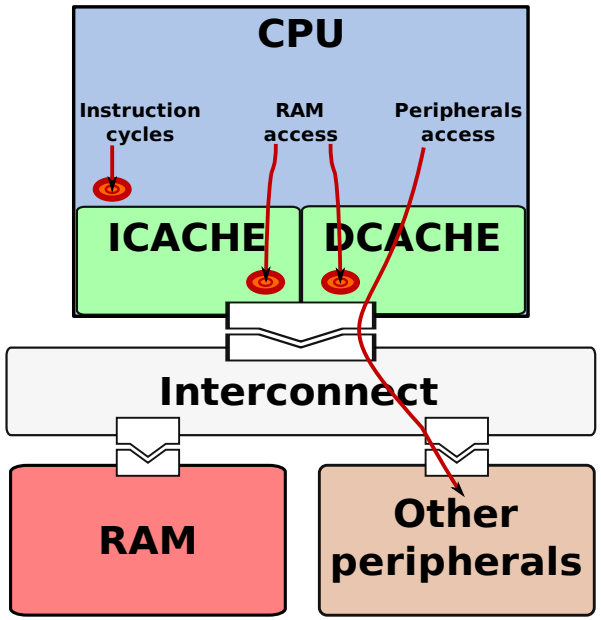
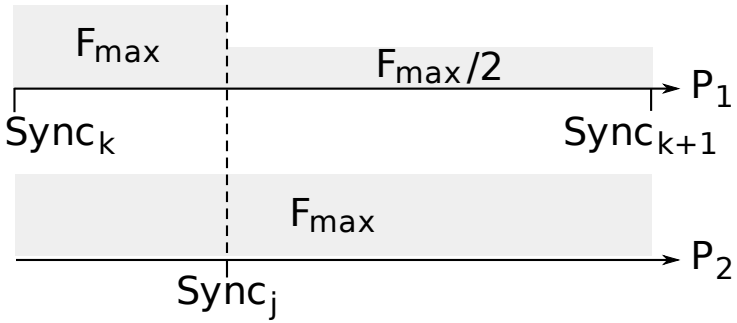


Figure: Cache wait & late

Frequency and Energy Modeling

Frequency modeling

- Each processor can be simulated at a different frequency
- Compute time corresponding to the number of cycles to synchronize ($t = nb\_cycles/fq$)
- A processor can change the frequency of other processors



Energy modeling (in collaboration with Nicolas Fournel)

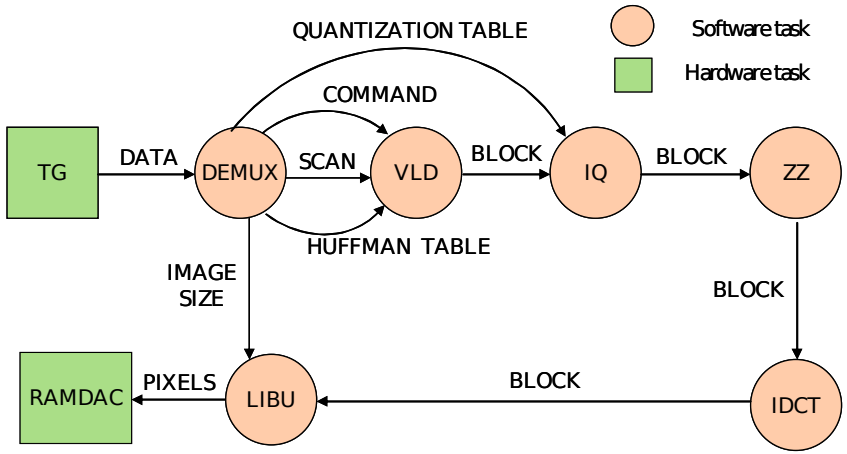
- Power consumed by a processor at a given moment depends on the current activity (e.g. instruction execution, IDLE state) and the current frequency and voltage of the processor
  - $E_t^{CPU} = \sum (P^{CPU}[Activity_i][f, v] \times t_i)$, where $\sum t_i = t$
- Power consumed by a hardware component depends on the current running mode (display device: idle, display): $E_t^{HW} = P^{HW} \times t$
  - For each event $Event_i$ (display device: read, write), energy $E_{Event_i}^{HW}$ is added

Outline

Motion-JPEG Application

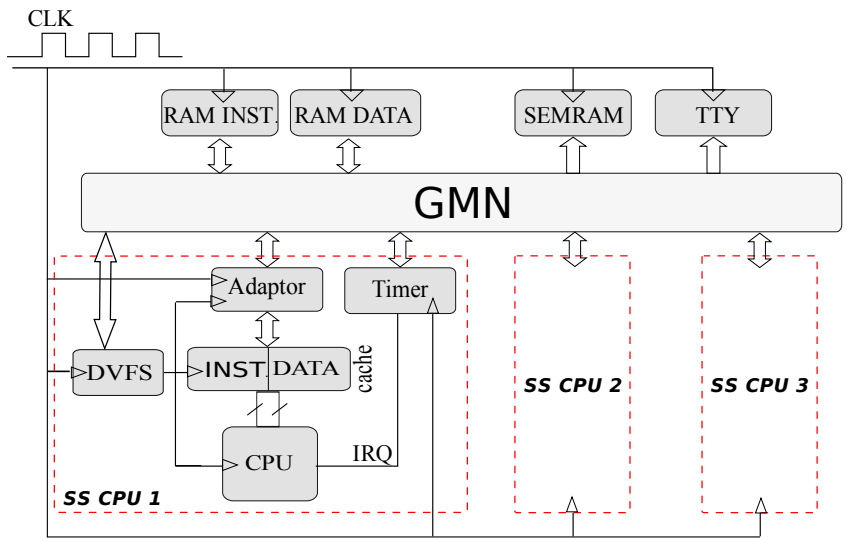
Software stack

- Motion-JPEG decoding application
- Mutek operating system
  - POSIX compliant
  - SMP version



Hardware platform

- Processors
- Caches
- Interconnect
- Memories
- Timers
- DVFSes
- ...



Experimental Results - Accuracy

Virtual Platforms

- Transaction level: modeled with QEMU/SystemC
- CABA reference: modeled with ISS/SoClib/SystemCASS

Table: Monoprocessor results

	No cache	Cache late	Cache wait	Cache full
Instruction cycles error	0.00 %	0.00 %	0.00 %	0.00 %
Uncached accesses error	0.00 %	0.00 %	0.00 %	0.00 %
Simulated time error (accuracy error)	-36.70 %	-0.04 %	-0.04 %	-0.04 %
Sim. speedup (vs. Cache full)	19.38 X	12.48 X	1.94 X	1
Sim. speedup (vs. CABA reference)	553.10 X	356.13 X	55.39 X	28.55 X

Table: Multiprocessor results

	No cache	Cache late	Cache wait	Cache full
Instruction cycles error	35.13%	22.31%	5.24 %	2.56 %
Uncached accesses error	-44.54 %	-9.80 %	2.01 %	8.19 %
Simulated time error (accuracy error)	-21.07 %	1.34 %	-8.44 %	-3.42 %
Sim. speedup (vs. Cache full)	21.45 X	13.87 X	2.03 X	1
Sim. speedup (vs. CABA reference)	381.01 X	246.38 X	35.97 X	17.76 X

Experimental Results - Simulation Speed

Virtual Platforms

- Transaction level: modeled with QEMU/SystemC
- Transaction level reference
  - Interpretative SoClib ISS
  - Rest of the hardware components from our simulation platform
- Results: "Cache full" configuration is **2** times faster than the reference platform

Other simulation speed results ("No cache" configuration)

- Linux boot: **31s** (against 3.3h estimated for CABA)
- H264 frame decoding: **2s** (against 12.7min estimated for CABA)

Limitations

Limitations

- The processors pipeline is not modeled
- Usual drawback of TLM - Time modeling is rough

Outline

- Introduction
- Using binary translation in event driven simulation
- Experimental results
- **Conclusion and Perspectives**

Conclusion and Perspectives

Simulation strategy implemented at the TA abstraction level

- Based on the existing untimed binary translation ISSes
- Allows a simulation speed/accuracy trade-off
- Supports runtime change of the processors frequency

Short term future works

- Integrate the ISS with cycle accurate hardware models of SoCLib

Long term future works

- Improve the accuracy (timing, power) modeling of the processors

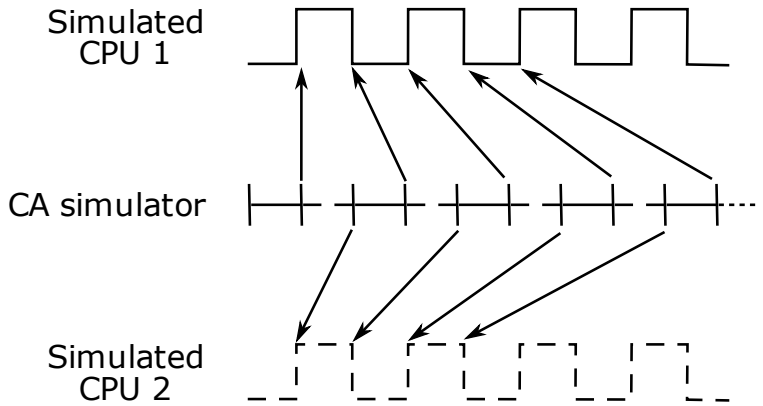
Thank you!

{marius.gligor@imag.fr}

Binary Translation / SystemC Synchronization

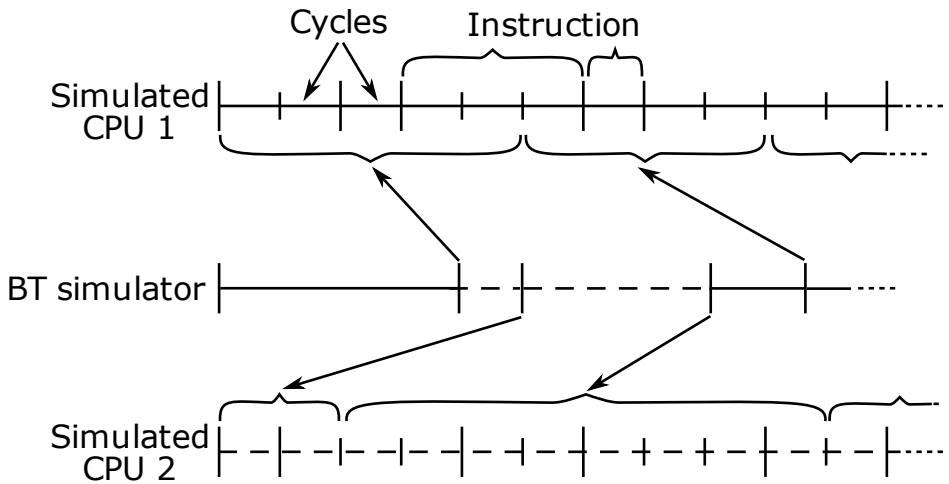
Simulation at CA abstraction level

- Synchronization at least each clock edge



Binary translation based ISS (TLM)

- Synchronization after one or several simulated cycles



QEMU/SystemC Implementation Details