Combined Use of Dynamic Binary Translation and SystemC for Fast and Accurate MPSoC Simulation

Marius Gligor and Frédéric Pétrot

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

March 18, 2011
Outline

- **Introduction**
  - Context & Motivations

- Using binary translation in event driven simulation

- Experimental results

- Conclusion and Perspectives
Motivations

Classical TA simulation 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
Using binary translation in event driven simulation

Outline

- Introduction
- Using binary translation in event driven simulation
  - Background technologies
  - Multiprocessor modeling
  - Time modeling
  - Frequency and energy modeling
- Experimental results
- Conclusion and Perspectives
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 exchange
- Discrete event-driven simulation
  - Concurrency of processes
QEMU Emulation Process

Process

PC already seen?
Yes
No

Binary Translation

Fetch

Decode

Branch?

Execute

No

Yes

micro-ops
identifiers buffer

Tiny code
generator

Code Generation

Target binary code (.elf)

Micro-operations built-in

Translation Cache (host binary code)

TB Cache Entry
QEMU Emulation Process

Process

PC already seen? Yes | No

Fetch

Instruction

Target binary code (.elf)

Micro-operations built-in

Decode

Branch?

Yes

No

Execute

Translation Cache (host binary code)

TB Cache Entry

Binary Translation

micro-ops identifiers buffer

Tiny code generator

Target binary code (.elf)

Code generation example

18 target_instrX
QEMU Emulation Process

Process

PC already seen? → Yes

Yes → micro-ops identifiers buffer

No → Tiny code generator

Micro-operations built-in

Translation Cache (host binary code)

TB Cache Entry

Instruction

Target binary code (.elf)

Code generation example

18 target_instrX → micro-op1_instrX

micro-op2_instrX
QEMU Emulation Process

Process

PC already seen? Yes → Instruction

PC already seen? No → Binary Translation

Instruction → Fetch

Fetch → Decode

Decode → Branch?

Branch? No → Execute

Branch? Yes → Execute

Execute → Translation Cache

Translation Cache (host binary code)

TB Cache Entry

Tiny code generator

Code Generation

Micro-operations built-in

Target binary code (.elf)

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)
- QEMU normal processor simulation breaks (e.g. interrupt handling)
- Predefined period without synchronization
Binary Translation / SystemC Synchronization

Simulation at CA abstraction level
- Synchronization at least at each clock edge

Binary translation based ISS (TLM)
- Synchronization after one or several simulated cycles
Interrupts Treatment

Interrupts forwarding

- Generated by hardware components during the SystemC activities of the processors
- Interrupt pending flags set during the SystemC activity of the processors
- Flags viewed by QEMU at its activity resume
- Interrupt treated after the current SystemC activity ends, at the beginning of the next translation block
QEMU/SystemC Implementation Details

Using binary translation in event driven simulation -> Multiprocessor modeling

M. Gligor - Combined Use of Dynamic Binary Translation and SystemC for Fast and Accurate MPSoC Simulation
QEMU Isolation and Encapsulation

Motivation

- Simulate several processors of the same/different type in parallel
- QEMU uses global variables
- Processors unscheduling not longer at the translation block boundary
  - Generate problems (e.g. translation cache invalidation)

Isolations

1. Different processor types
   - QEMU compiled as dynamic library for each processors type
2. Processor groups
   - Structure that contains the variables of a processor group
3. Processors in the same processor groups
   - Processor specific global variables saved and restored at each SystemC synchronization
Using binary translation in event driven simulation - Time modeling

**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**

<table>
<thead>
<tr>
<th>Instr address</th>
<th>Target code</th>
<th>Original translation</th>
<th>Annotated translation</th>
<th>Annotated generated code</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr_instrX</td>
<td>target_instrX</td>
<td>micro-op1_instrX</td>
<td>micro-op1_instrX</td>
<td>host_instr1_micro-op1_instrX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>host_instr2_micro-op1_instrX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>host_instr3_micro-op1_instrX</td>
</tr>
<tr>
<td>micro-op2_instrX</td>
<td></td>
<td>micro-op_annotation</td>
<td></td>
<td>host_instr1_micro-op_annotation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>host_instr2_micro-op_annotation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>host_instr1_micro-op2_instrX</td>
</tr>
</tbody>
</table>
## 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)

<table>
<thead>
<tr>
<th>Instr address</th>
<th>Target code</th>
<th>Original translation</th>
<th>Annotated translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_tb:</td>
<td>instr1_reg_operation</td>
<td>micro-op1_for_instr1</td>
<td>instr_cache_verify (18);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>micro-opN1_for_instr1</td>
<td>nb_cycles += cpu_datasheet [instr1];</td>
</tr>
<tr>
<td>1C</td>
<td>instr2_load_from_1000</td>
<td>micro-op1_for_instr2</td>
<td>nb_cycles += cpu_datasheet [instr2];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>micro-opN2_for_instr2</td>
<td>data_cache_verify (1000);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nb_cycles += cpu_datasheet [instr2];</td>
</tr>
<tr>
<td>20</td>
<td>instr3_store_5_to_2000</td>
<td>micro-op1_for_instr3</td>
<td>instr_cache_verify (20);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>micro-opN3_for_instr3</td>
<td>nb_cycles += cpu_datasheet [instr3];</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>write_access (2000, 5);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nb_cycles += cpu_datasheet [instr3];</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>micro-op1_for_instr3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>micro-opN3_for_instr3</td>
</tr>
</tbody>
</table>
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)
- Cache late
  - Caches as pure directories
  - Precomputed time consumed at the next synchronization

**Figure:** Cache late
Using binary translation in event driven simulation - Time modeling

Backdoor Mechanism

Motivation

▶ Allows to access some SystemC modules (e.g. main memory) without advancing the SystemC time
▶ Special interface

Used for

▶ Faster simulation
  ▪ "No cache" and "cache late" precision levels
▶ Correct and accurate simulation
  ▪ Target binary code translation
  ▪ Debug

Accuracy issues

▶ TLB records updated during the translation and debug must be restored
Using binary translation in event driven simulation -> Frequency and energy modeling

**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 \( t = \frac{nb\_cycles}{fq} \)
- A processor can change the frequency of other processors

**Energy modeling**

- 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

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

**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 - Speedup and Accuracy

Virtual Platforms

- Transaction level: modeled with QEMU/SystemC
- 1) CABA reference: modeled with ISS/SoClib/SystemCASS
- 2) Transaction level reference
  - Interpretative SoClib ISS
  - Rest of the hardware components from our simulation platform

”Cache late” configuration results

<table>
<thead>
<tr>
<th></th>
<th>Simulation speedup</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CABA reference</td>
<td>350X</td>
<td>92 %</td>
</tr>
<tr>
<td>Transaction level reference</td>
<td>2X</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Other simulation speed results (”Cache late” configuration)

- Linux boot: **6s**
- H264 frame decoding: **1.8s**
Limitations

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

Using binary translation in event driven simulation

Experimental results

Conclusion and Perspectives
Conclusion and Perspectives

Simulation strategy implemented at TLM

- Based on the existing untimed binary translation ISSes
- Allows a simulation speed/accuracy trade-off
- Supports runtime change of the processors frequency
- Open source available at http://tima-sls.imag.fr/www/research/rabbit

Future works

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

QEMU evolution that we would like to see

- Isolate and make reentrant the ISS
Thank you!

{marius.gligor@imag.fr}