Modeling Exclusive Memory Access for a Time-Decoupled Parallel SystemC Simulator

Jan Henrik Weinstock, Rainer Leupers
(ICE, RWTH Aachen)

Sankt Goar, 02.06.2015
Virtual Platforms

- Virtual platforms are used to execute **target specific** software
  - Non-intrusive debugging
  - Design space exploration

- **SystemC** is the de-facto industry standard for constructing virtual platforms
  - System-level modelling language
  - Abstraction levels (speed vs. accuracy)
Parallel Simulation

- **Number of processors in embedded systems rising:** simulation speed degrades
  - Higher abstraction levels alone not sufficient anymore
  - Multi-core PCs → Parallel simulation

Jan Henrik Weinstock, Christoph Schumacher, Rainer Leupers, Gerd Ascheid and Laura Tosoratto:
*Time-Decoupled Parallel SystemC Simulation*, DATE’14

- **Considerations for models used in a parallel environment**
  - SystemC processes may execute concurrently
  - Local timestamps of SystemC processes may differ
  - Cross-thread communication is stated ahead of time
**Simulation kernel distributes load on module granularity**

- Even load per thread results in better performance

**Black-box CPU model** – no benefit from parallel simulation:
Kernel only sees three modules (cpu, bus, mem) and only one produces significant load (cpu)

**Separate cores model** – good benefit from parallel simulation:
Equal load produced by core 1 and core 2; should be simulated on different host threads.
Modeling Overhead

- **Inter processor communication**
  - Multi-core interrupt controller

- **Caches and Cache Coherency**
  - Not modeled at high abstraction levels

- **Exclusive memory access**
  - Required for synchronized access to shared memory
  - Embedded: Load-Link & Store-Conditional (LL/SC)

<table>
<thead>
<tr>
<th>ISA</th>
<th>Load-Link</th>
<th>Store-Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMv8</td>
<td>LDXR</td>
<td>STXR</td>
</tr>
<tr>
<td>MIPS32</td>
<td>LL</td>
<td>SC</td>
</tr>
<tr>
<td>Power</td>
<td>lwarx</td>
<td>stwcx</td>
</tr>
</tbody>
</table>
Communication Channels

- **TLM Blocking Transport Interface (BT)**
  - Models communication using *interface method calls*

```
transaction tx; sc_time t;
tx.set_address(0xc01f00a);
tx.set_data_ptr(&R[0]);
tx.set_read();
socket->b_transport(tx, t);
```

- **TLM Direct Memory Interface (DMI)**
  - Models communication using *memory pointers*

```
ptr = get_direct_memory_ptr(); // at sim start

memcpy(&R[0],
      ptr + 0xc01f00a,
      sizeof(uint32_t));
```
- **TLM generic payload does not support LL/SC**
  - Extension used to signal Read $\rightarrow$ LL and Write $\rightarrow$ SC
  - Extension holds ID of initiating core
- **Exclusive memory access Monitors used in interconnects**
  - Keeps track of active links
- **TLM generic payload does not support LL/SC**
  - Extension used to signal Read $\rightarrow$ LL and Write $\rightarrow$ SC
  - Extension holds ID of initiating core
- **Exclusive memory access Monitors used in interconnects**
  - Keeps track of active links
- **TLM generic payload does not support LL/SC**
  - Extension used to signal Read $\rightarrow$ LL and Write $\rightarrow$ SC
  - Extension holds ID of initiating core

- **Exclusive memory access** Monitors used in interconnects
  - Keeps track of active links
TLM Blocking Transport Interface

- **TLM generic payload does not support LL/SC**
  - Extension used to signal Read $\rightarrow$ LL and Write $\rightarrow$ SC
  - Extension holds ID of initiating core
- **Exclusive memory access Monitors used in interconnects**
  - Keeps track of active links
- **Problem**: ISS operates directly on memory via pointers (DMI)
  - Monitors will not get notified when links are broken with regular write (store) operations

- **Solution**: exploit simulation host architecture to detect if a store-conditional should fail
  - X86 has no LL/SC, but we can use compare-and-swap (CAS) to emulate LL/SC:

```c
ptr = get_direct_memory_ptr(); // at sim start

Core 0

MEM

Load-Linked [addr]                      Store-Conditional [addr, data]
---                                   ---
uint32_t data;
memcpy(&data, ptr + addr, 4);
core0.link_addr = addr;
core0.link_data = data;
return data;

if (addr != core0.link_addr)
   return false;
else
   return cas(ptr + addr, core0.link_data, data);
```
Mixed Mode Operation

- **Mixed-Mode allows to simulate transaction- and memory-based models together**
  - Useful for integration of models from different vendors
  - Detailed subsystem simulation

- **Memory component gets equipped with a monitor**
  - Monitor supplied together with DMI pointer
    
    ```
    ptr = get_direct_memory_ptr(&monitor); // at sim start
    ```

- DMI accesses (LL, SC, ST) must notify monitor
- Transaction accesses also pass through monitor once they reach memory component
- Offload secondary CPU cores to different simulation thread
OpenRISC Demo Screenshots

- **OpenRISC booting linux-3.18-smp**

![Screenshot of OpenRISC booting Linux](image-url)

```
TCP established hash table entries: 2048 (order: 0, 8192 bytes)
TCP bind hash table entries: 2048 (order: 1, 16384 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 8192 bytes)
UDP-Lite hash table entries: 256 (order: 0, 8192 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
futex hash table entries: 1024 (order: 1, 16384 bytes)
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
of_serial 90000000.serial: ttyS0 at MMIO 0x90000000 (irq = 2, base_baud = 125000)
0) is a 16550A
console [ttyS0] enabled
console [ttyS0] enabled
bootconsole [uart0] disabled
bootconsole [uart0] disabled
libphy: ethoc-mdio: probed
NET: Registered protocol family 17
Freeing unused kernel memory: 7888K (c02f8000 - c0aac000)
```

Please press Enter to activate this console.
OpenRISC Demo Screenshots

- **OpenRISC running process manager** `top`
OpenRISC Demo Screenshots

- OpenRISC pinging **www.google.de** and **www.silexica.com**

```plaintext
weinstoc@fabius: /net/heap/weinstoc/scl/build/release/install/examples/openrisc

weinstoc@fabius: /net/heap/weinstoc/scl/build/release/install/examples/openrisc

/ # ping www.google.de
PING www.google.de (173.194.67.94): 56 data bytes
64 bytes from 173.194.67.94: seq=0 ttl=45 time=2.490 ms
64 bytes from 173.194.67.94: seq=1 ttl=45 time=2.690 ms
64 bytes from 173.194.67.94: seq=2 ttl=45 time=2.570 ms
64 bytes from 173.194.67.94: seq=3 ttl=45 time=2.600 ms
64 bytes from 173.194.67.94: seq=4 ttl=45 time=2.670 ms

--- www.google.de ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 2.490/2.604/2.690 ms

/ # ping www.silexica.com
PING www.silexica.com (195.20.232.45): 56 data bytes
64 bytes from 195.20.232.45: seq=0 ttl=52 time=1.810 ms
64 bytes from 195.20.232.45: seq=1 ttl=52 time=1.980 ms
64 bytes from 195.20.232.45: seq=2 ttl=52 time=1.910 ms
64 bytes from 195.20.232.45: seq=3 ttl=52 time=1.880 ms
64 bytes from 195.20.232.45: seq=4 ttl=52 time=1.880 ms

--- www.silexica.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.810/1.892/1.980 ms
```

16
OpenRISC showing `ls -l` and interrupt status
### Slowdown cause by the model during Linux-SMP boot

- Turn off secondary cores (1 – 3)
- Replace LL → LD, SC → ST for comparison
- Safe during first 3 simulated seconds (no SC failed)

<table>
<thead>
<tr>
<th>Model</th>
<th>Run</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Average</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMI</td>
<td>LDST</td>
<td>17,7197</td>
<td>17,6685</td>
<td>17,7443</td>
<td>17,6699</td>
<td>17,7074</td>
<td>17,7020</td>
<td>0.39%</td>
</tr>
<tr>
<td></td>
<td>LLSC</td>
<td>17,7291</td>
<td>17,8069</td>
<td>17,7145</td>
<td>17,7450</td>
<td>17,8597</td>
<td>17,7710</td>
<td></td>
</tr>
<tr>
<td>TRANS</td>
<td>LDST</td>
<td>77,9926</td>
<td>78,2996</td>
<td>78,0086</td>
<td>77,3476</td>
<td>81,1386</td>
<td>78,5574</td>
<td>1.83%</td>
</tr>
<tr>
<td></td>
<td>LLSC</td>
<td>81,3725</td>
<td>80,1775</td>
<td>79,4534</td>
<td>78,1732</td>
<td>80,7987</td>
<td>79,9951</td>
<td></td>
</tr>
</tbody>
</table>
Performance Results

- **Time-Decoupled parallel simulation vs. sequential simulation**
  - Linux boot (sim. duration = 1 second)
  - Speedup improves with increasing quantum
  - 3.4x speedup using 4 threads for simulation

![Graph showing performance results](image-url)
Conclusions

- Proposed a model for exclusive memory access using SystemC/TLM
- Approach adapts to modeling style:
  - Fast (DMI) vs. Accurate (Transactions)
  - Mixed operation possible
  - No changes required for ISS when using DMI
- Speedup 3.4x possible with separate cores & parallel SystemC kernel

Future Work
- Test model with more architectures/ISS (e.g., ARM)
- Improve model accuracy, e.g. by tracking load-links on cache line granularity
- Simplify integration by moving exclusive access monitors into TLM initiator sockets

Thank You!