A novel way to efficiently simulate complex full systems incorporating hardware accelerators

Nikolaos Tampouratzis

Technical University of Crete, Greece
Motivation / The Idea

➢ **Most simulators** used for evaluating the complete user applications (i.e. full-system CPU/Mem/Peripheral simulators) **lack any type of SystemC accelerator support**.

➢ We developed a **novel simulation environment** comprised of a **full system simulator** with a **SystemC cycle-accurate hardware accelerator** device
  ✓ Efficiently and seamlessly supporting communication and synchronisation issues
  ✓ Allows for fast design of any kind of accelerator

➢ **COSSIM** can be **extended** to support our **Simulation environment** to simulate **hundreds of nodes incorporating H/W Accelerators**

➢ Our System utilizes:

  ➢ **GEM5 full-system simulator**: simulates complete systems comprised of numerous CPUs, peripherals running full OS
  ➢ **Accellera [1]**: open-source proof-of-concept **SystemC simulator** (it has been approved by the IEEE Standards Association [2])

---


Our Approach

I. Operating System (OS)
   ✓ a set of device drivers in Kernel mode have been developed
   ✓ our novel approach allows full overlap of the two sub-simulations

II. Memory Bus
    ✓ Bus master port is connected to the peripheral I/O Wrapper (write/read registers)
    ✓ Bus slave port is connected to the GEM5 DMA Wrapper (write/read large amounts of data)
Our Approach (2)

III. Accelerator Wrapper
✓ a mixed C++/SystemC core → connection of the GEM5 C++ functions and the accelerator’s SystemC threads
✓ a large Device Memory to store the data from OS’ memcpy was implemented (simulating the DDR memory of most of the real systems incorporating PCI-connected FPGA and/or GPU boards)

IV. SystemC Accelerator
✓ a reference SystemC accelerator has been developed to evaluate the Accelerator Wrapper and the Linux Kernel Drivers
✓ Our approach is a helpful reference for any synthesizable SystemC accelerator
Implementation
Accelerator Wrapper

- **Buddy dynamic memory allocation algorithm** scheme [3] implemented (similar to the cudaMalloc of NVIDIA GPUs)
- **2 DMA engines** → efficiently transfer high data volumes from the Linux driver to the Accelerator Wrapper and vise versa (cudaMemcpy similar)
- **GEM5 synchronisation event** → triggered at every SystemC accelerator device cycle

- **initialisation thread** → generate the reset and start signals
- **clock thread** → generate the actual clock signal of the accelerator
- **2 MemCpy SystemC threads** → pass the data from the Wrapper Device Memory to synthesisable I/O ports

A reference Controller/Scheduler SystemC → the main SystemC thread

- The user has the ability to create as many individual cores as required

2 SystemC memcpy functions → efficient communication with the Accelerator Wrapper Memory
**User-Friendly Interface (1)**

- **AccelMemcpy** & **AcelCallDevice** → **asynchronous** functions
  - the application running on GEM5 (including the OS) can continue to execute its computational tasks while the SystemC accelerator is also running/simulated.
  - DMA engines are used to transfer the Data from the Host to the Device.

### Code 1: Header of Accelerator Driver

```c
1: typedef uint64_t DevMemAddr;
2: void AccelInitialization();
3: void AccelFinalization();
4: DevMemAddr AccelMalloc(size_t size, const char *label);
5: void AccelFree(DevMemAddr SWAddr);
6: void AccelMemcpy(DevMemAddr SWAddr, void *data, size_t size, uint8_t TransferType);
7: void AccelCallDevice();
```
User-Friendly Interface (2)

Code 2 Segments of SystemC Application

```cpp
1: SC_MODULE(SystemCDevice){
2:     int INTER_MEM[MEM_SIZE]; // Create an Internal Memory
3:     ...
4:     SC_CTOR(SystemCDevice){
5:         SC_CTHREAD(main_thread, clk.pos());
6:         async_reset_signal_is(reset, true);
7:     }
8: }
9: void SystemCDevice::main_thread(){
10:     while(true) {
11:         uint64_t size = 18; // Copy 18 elements
12:         memcpyToDevice(INTER_MEM,'A', 0, size, ACC_INT);
13:         <Some Processing... / Call other computational Threads>
14:         memcpyToHost(INTER_MEM,'A', 0, size, ACC_INT);
15:         wait();
16:     }
17: }
```

memcpyToDevice

memcpyToHost
**Evaluation**

**Conventional Method (top)**
- in every SystemC simulator call, the full-system must boot the OS from scratch
- no notion of synchronisation exists

**Proposed method (bottom)**
- full-system simulator boots the OS only once
- full global synchronization is supported through programmed I/Os and DMA engines

(i) Standard (top) & (ii) Our novel method integrating GEM5 Full System Simulator with Accellera Simulator
Performance Results (1)

System is evaluated on two use cases using Intel i7-3632QM at 2.2GHz with 8GB of RAM as processing platform.


<table>
<thead>
<tr>
<th>Mutual Information</th>
<th>Transfer Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Iterations (Complexity)</td>
<td>Standard Method</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>100</td>
<td>37.3 min</td>
</tr>
<tr>
<td>200</td>
<td>37.5 min</td>
</tr>
<tr>
<td>500</td>
<td>37.9 min</td>
</tr>
</tbody>
</table>

Simulation time of (i) standard and (ii) our method using two use cases (Accelerator is called ten times)

Performance Results (2)

Simulation time of TE using different # of Accelerator calls
Thank You