

A Novel, Simulator for Heterogeneous Cloud Systems that incorporate Custom Hardware Accelerators

Nikolaos Tampouratzis, Ioannis Papaefstathiou

Abstract—The growing use of hardware accelerators in both embedded (e.g. automotive) and high end systems (e.g. Cloud infrastructure) triggers an urgent demand for simulation frameworks that can simulate in an integrated manner all the components (i.e. CPUs, Memories, Networks, Hardware Accelerators) of a system-under-design(SuD). By utilizing such a simulator, software design can proceed in parallel with hardware development which results in the reduction of the so important time-to-market. The main problem, however, is that currently there is a shortage of such simulation frameworks; most simulators used for modelling the user applications (i.e. full-system CPU/Mem/Peripheral simulators) lack any type of support for tailor-made hardware accelerators. The presented ACSIM framework is the first known open-source, high-performance simulator that can handle holistically system-of-systems including processors, peripherals, accelerators and networks; such an approach is, for example, very appealing for the design of Cloud Servers that incorporate FPGAs as PCI-connected accelerators. ACSIM is an extension of the COSSIM simulation framework and it integrates, in a novel and efficient way, a combined system and network simulator with a SystemC simulator, in a transparent to the end-user way. ACSIM has been evaluated when executing several real-world use cases; the end results demonstrate that the presented approach has up to 99% accuracy in the reported SuD aspects (when compared with the corresponding characteristics measured in the real systems), while the overall simulation time can be accelerated almost linearly with the number of CPUs utilized by the simulator. More importantly, the presented interconnection scheme between the Processing and the SystemC simulators is orders of magnitude faster than the existing solutions, while ACSIM can efficiently simulate up to several hundreds of processing nodes with hardware accelerators interconnected together, in a fully distributed manner.

Index Terms—Parallel Distributed Systems Simulator, Heterogeneous Cloud Systems Simulator, Hardware Accelerators.



1 INTRODUCTION

NOWADAYS Highly Parallel and Distributed computing systems (i.e. Clouds and HPC systems) are growing in capability at an extraordinary rate, incorporating processing systems that vary from simple processors to high performance units and hardware accelerators connected with each other through numerous networks. One of the main problems the designers of such heterogeneous systems face is the lack of simulation tools that can offer realistic insights beyond simple functional testing, such as the actual performance of the processing and the hardware accelerators, accurate overall system timing, power/energy estimations etc. On top of that, software design is severally influenced by the high complexity of the new hardware systems, since there is a need for new software that can take advantage of the novel features incorporated into the hardware accelerators. If the software development cannot be done in parallel with the hardware implementation then the overall design cycle is getting prohibitively large.

In this paper, we present the ACSIM Simulation Framework which is an extension of an open-source framework (i.e. COSSIM [1]) that aims to address all the aforementioned

limitations. COSSIM efficiently integrates a series of sub-tools that model the computing devices of the processing nodes as well as the network(s) of the parallel systems. It provides cycle accurate results by simulating the actual application and system software executed on each node, together with the actual networks employed and it provides power/energy consumption estimation for both the processing elements and the network based on the actual dynamic usage scenarios. In this paper, we extend COSSIM by introducing a novel flow that enables the designer to rapidly prototype synthesisable SystemC hardware accelerators in conjunction with the Processing & Network systems without worrying about communication and synchronisation issues. This allows system designers to simulate not only the new software running on the processing units but also the new hardware modules within the same environment; this novel aspect significantly accelerates the final product delivery.

The contribution of this paper can be summarized in the following points.

- Nikolaos Tampouratzis is with the Laboratory of Computer Systems Architecture, Aristotle University of Thessaloniki, Greece, GR, 54124. E-mail: ntampouratzis@ece.auth.gr
- Ioannis Papaefstathiou is with Synelixis Solutions Ltd, Chalkida, Greece, GR, 34100. E-mail: ygp@synelixis.com

- The first known open-source integrated simulation framework which can simulate whole Heterogeneous Cloud Systems that incorporate PCI-connected Hardware Accelerators (such as Amazon EC2 F1 [2], Alibaba Cloud [3]).
- An innovative flow to enable the designer to simulate the complete Cloud System (i.e. CPU, Net-

Manuscript received March 13, 2018; revised October 28, 2018.

work, Hardware Accelerators) within one simulation framework.

- A novel global synchronization scheme which takes into account the trade-off between the simulation speed and the simulation accuracy.
- The introduction of micro-routers so as to allow the support of a wide range of different real network protocols.
- A thorough evaluation of the end system based on several real world use cases and different metrics.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the similar existing approaches, while Section 3 describes the underlying COSSIM simulator. Subsequently, Section 4 presents our novel approach for simulating seamlessly hardware accelerators and Section 5 demonstrates the efficiency of our approach. Finally, Section 6 concludes the paper.

2 RELATED WORK

The ACSIM simulator can be mainly utilized in the development of systems that incorporates PCI-connected accelerators. In this section we list the main existing approaches in this domain.

Starting from the Cloud domain, the most widely used simulation framework is CloudSim [4] and its numerous extensions or derivatives (e.g. CDOSim [5], Network-CloudSim [6], CloudAnalyst [7], CloudSimDisk [8]). Compared to ACSIM, these simulators are far more generic and do not support precise simulation of the actual execution of an application neither they model the exact hardware used (i.e. they use generic models of the underlying platforms). Other cloud simulators focus on specific aspects of the data center that are useful for cloud providers such as, and mainly, resources' provisioning. Those simulators model user and application requirements through stochastic processes or mathematical models in order to predict resource usage and provide optimization insights for avoiding excessive overprovisioning and underutilization (GreenCloud [9], BigHouse [10], CACTOS [11]). ACSIM is orthogonal to those systems; the initial results from those tools, for a specific application, can be optimized by executing the application on top of ACSIM and extract the precise application performance indicators. More importantly, none of those systems support any kind of hardware accelerators.

In addition, there are certain tools that can simulate an accelerator together with a multi-core CPU. Natural [12] supports the simulation of natural language processing in a system comprising of an accelerator interconnected to the last-level cache of a multi-core system. Furthermore Aladdin, [13], is a "pre-RTL" power performance simulator designed to enable rapid design space exploration for accelerator-centric systems". Both of these approaches use the GEM5 processing simulator in its *syscall* emulation mode which means that no operating system can be supported within their simulations making them impractical for general Cloud/HPC system design; moreover they cannot support any kind of networking.

Furthermore, there are two approaches which present synchronization schemes for connecting GEM5 with a SystemC simulator [14], [15]; the main drawback of the first one is that they replace the GEM5 *simulate* function with a

SystemC *simulate* function thus the complete GEM5 environment is heavily slowed down while most of its features cannot be utilized. As a result such an approach cannot be used in highly parallel systems simulations that require fast simulation speeds as well as the full features provided by GEM5. In contrast, in the ACSIM framework, GEM5 is synchronized with SystemC only during accelerator execution. In [15], the authors present a coupling of GEM5 with SystemC in order to describe realistic SystemC models of modern CPUs. However, the main difference with ACSIM is that in [15] the communication between the GEM5 and accelerator is done through a specific NoC, thus such a system can be utilized only on single chip designs; ACSIM can be used for the design of the most widely-used multi-chip/multi-board systems that incorporate PCI-connected H/W accelerators (such as the ones utilized in Amazon EC2 F1, Microsoft Azure, Alibaba Cloud etc.).

Finally, the authors in [16] present *PARADE*, a cycle-accurate full-system simulation platform that enables the design and exploration of the emerging accelerator-rich architectures. *PARADE* can generate dedicated or composable accelerator simulation modules and simulate the management of the accelerators, together with a customizable network-on-chip, at cycle-level. However, they use AutoPilot and they need to synthesize the accelerator, in order to get cycle accurate timing information, and as a result their overall system is much slower than our approach which is based on SystemC high-level system modeling. Finally, *PARADE* supports a shared memory model between the accelerators and the CPU cores and thus it can be used, just as the other systems described above, only in the design of single-chip solutions. Moreover, none of the existing systems supports the simulation of real networks among the GEM5 instances.

Based on all the above, it is clear that currently there is no framework supporting all the features of ACSIM, namely simulation of off-chip hardware accelerators together with CPUs and networks in a fast and fully featured way.

3 COSSIM FRAMEWORK

Since ACSIM is an extension of the COSSIM simulation framework, in this section we introduce the latter as well as the enhancements/modifications developed in order to support different network protocols (those have not been presented before). COSSIM is built upon several established open-source packages [1]. GEM5 [17] is used to simulate the processing components of each node in the system, while OMNET++ [18] is employed to simulate the real networking infrastructure between those nodes. To bind the whole framework together and establish a common notion of time, COSSIM employs the IEEE1516 HLA architecture [19] through the open-source CERTI package [20].

Figure 1 demonstrates the COSSIM simulator with all its components and interfaces. Multiple instances of a node simulator module (i.e. a GEM5-based subsystem called cGEM5) are required for the efficient simulation of the numerous processing nodes of a parallel system. The network that binds together the different nodes is simulated by the network simulation module (i.e. OMNET++ based module called cOMNET++). The processing simulation instances are connected with the network one through IEEE HLA

compliant interfaces (interface #1). Specifically, *COSSIMlib* has been developed so as to enable the interoperability between cGEM5/cOMNET++ and CERTI/HLA. Finally, each cGEM5 instance is connected through a custom XML interface with a McPAT instance to allow the estimation of the energy/power consumption of each node (interface #2), while cOMNET++ employs internally the INET/MiXIM add-on to estimate the power consumption of the network. Section 3.1 introduces the transparent micro-router functionality which is developed in order to allow the support of a wide range of different real network protocols, while Section 3.2 provide a recap of the novel COSSIM Synchronization scheme which heavily affects the evaluation numbers presented in the Validation section.

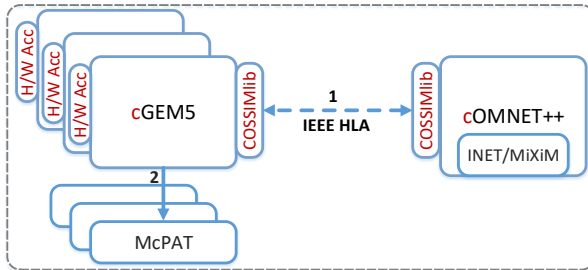


Fig. 1: Top-level view of the COSSIM framework

3.1 Transparent micro-Routers Functionality

As described in [1], GEM5 supports only an Ethernet Network Interface Card. In order to support different network protocols, in the context of ACSIM (such as WiFi protocols), we implemented, within OMNET++, a micro-router functionality allowing user to change the physical medium (and as a result the network adapter) from Ethernet to any other network without affecting at all the GEM5 network card.

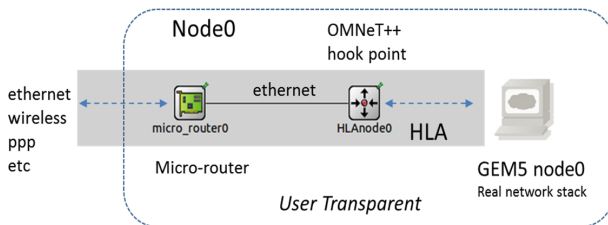


Fig. 2: Network node with micro-router support

ACSIMs transparent micro-routers have two interfaces: an Ethernet one which is 100% compatible with the Linux protocol stack and which is always connected to the cGEM5 node network card and a fully interchangeable interface which supports any possible network. This is demonstrated in Figure 2 (HLA node0 is the connection point of cGEM5 and cOMNET++) where on the left side micro_router0 is a dual interface micro-router which supports any kind of wireless and/or wired protocol on one side and the Ethernet one on the other. As a technical note, the micro-router is implemented as an INET StandardHost enhanced with routing capabilities and multiple interfaces. This approach allows for the full simulation of any interconnection network but when the actual network adapter of the CPU is not an Ethernet one (e.g. InfiniBand) the accuracy may be affected due to the fact that we create Ethernet packets which are then routed over any other network within OMNET++; this can be improved when/if new GEM5 models for NIC cards are developed (e.g. Infiniband).

3.2 Global Synchronization

The actual COSSIM synchronization scheme consists of two levels:

- 1) **Synchronization per node** Each node (i.e. cGEM5 and ACSIM in our case) needs to communicate, in a consistent way, with its corresponding node in the network simulator (i.e. cOMNET++) and exchange data packets. This type of synchronized communication is necessary because certain network data between the two simulators must be exchanged in a manner that will preserve the exact time ordering. For this reason, one federation is created per node pair so as to support full synchronization as illustrated in Figure 3 (a).
- 2) **Global Synchronization** The COSSIM simulator needs to periodically synchronize all nodes. This is because it supports different types of CPUs with potentially different clock cycles and/or different network protocols, all resulting in varying workload for the simulators' engines. Therefore, the simulated time (i.e the time aspect of the modeled system) in each node can be completely different for the same wall-clock time. For this reason, a Global Synchronization federation is created to achieve a unified notion of time which contains all cGEM5 nodes and one OMNET++ helper Node (SynchNode) as illustrated in Figure 3 (b).

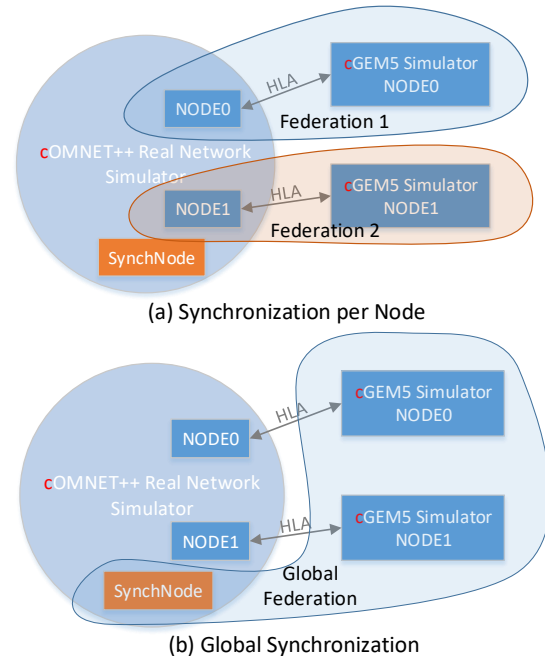


Fig. 3: COSSIM HLA federations

The *Synchronization time per node* and the *Global Synchronization time* are two different entities that can be separately defined by the user. The first one is mostly defined by the latency of the network interface and it does not affect the simulation speed while the second is a trade-off between simulation speed and simulation accuracy.

The proposed global synchronization scheme is responsible for the preservation of the cycle-accurate notion of the simulation process, including the Hardware Accelerators. OMNET++ is natively an event driven simulator, however

by employing the global synchronization scheme, it becomes hooked to the "cycle-events" of each of the cGEM5 simulated nodes. This not only prevents the clocks of all the nodes from any drift but also implicitly "forces" the cOMNET++ to act like a "cycle-driven" event simulator. In this respect every component of the simulated system has the exact same notion of time (in terms of clock cycles).

4 ACSIM FRAMEWORK

4.1 Design

This section describes the most significant enhancements and modifications developed in order to expand the COS-SIM simulator so as to support our novel SystemC accelerator and thus create the ACSIM framework. Figure 1 presents a novel approach (which is developed in ACSIM) for the interconnection of the full system processing simulator with a SystemC cycle-accurate accurate one which allows for the S/W and H/W description to be developed within the same simulation environment. SystemC is selected because of its cycle-accurate simulation features, while it is one of the most widely used input languages for the High Level Synthesis (HLS) tools. In this work, we use the open-source simulator and the SystemC definition as it is provided by the Open SystemC Initiative (OSCI), now known as Accellera [21], which has also been approved by the IEEE Standards Association [22].

The ACSIM framework proposes a novel design flow that can be used in the design of the most widely used multi-chip/multi-board parallel systems that incorporates hardware accelerators. Figure 4 illustrates the integration of our SystemC simulator with COSSIM; in particular the interconnection is between the SystemC simulator and cGEM5 and then there is an interconnection (which is described in the last sections) of cGEM5 with cOMNET++ which allows for the complete simulation of a heterogeneous parallel system.

4.1.1 Operating System (OS)

The OS can be represented as a layered structure, as in Figure 4. It contains the User Space with the user applications and all the appropriate libraries, and the Kernel Space (in our case a Linux Kernel), which is strictly reserved for running a privileged operating system kernel and most of the device drivers. In order to incorporate efficiently the module performing the simulation of the hardware accelerators we have developed a set of device drivers. The accelerator is activated through programmed I/Os that provide the start address and the size of the array used for the descriptors of the accelerator. The CPU can then sleep until an interprocessor interrupt from the accelerator is delivered to indicate task completion; this approach allows for full overlap of the two sub-simulations. In addition, an *ioctl* function has been developed in order to achieve efficient user-kernel space communication; this novel function can mainly perform the following tasks:

- **QUERY_SET_DATA** - Initialise the Direct Memory Access (DMA) copy transaction from the Host (Kernel Space) to the Accelerator Wrapper.
- **QUERY_GET_DATA** - Initialise the DMA copy transaction from the Accelerator Wrapper to the Host (Kernel Space).

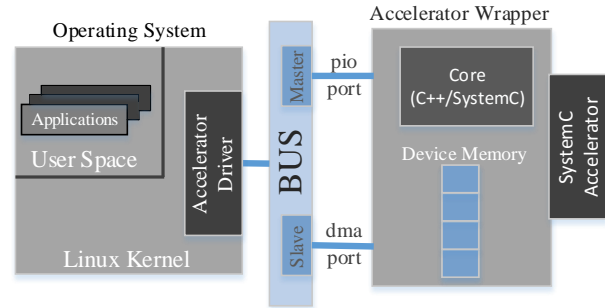


Fig. 4: Integration of SystemC accelerator with cGEM5 (full-system mode)

- **QUERY_CALL_DEVICE** - Trigger the SystemC accelerator to execute a specified application.

Finally, an interrupt handler has been implemented in order to receive appropriate interrupts from the Accelerator Wrapper, such as the SystemC accelerator finish signal, the *mempcy* finish signal, etc.

4.1.2 Memory Bus

cGEM5 provides a Memory Bus to interconnect all architecture components such as CPUs, Caches, RAMs as well as I/O devices using master and slave ports. Our novel Accelerator Wrapper device is attached to cGEM5's interconnection system using one master and one slave port. Specifically, one bus master port is connected to the peripheral I/O Accelerator Wrapper port *pio* in order to read and write into the accelerator's wrapper registers; similarly, one bus slave port is connected to the *DMA* Accelerator Wrapper port in order to write/read large amounts of data to/from the accelerator device memory, as shown in Figure 4.

4.1.3 Accelerator Wrapper

Our novel Accelerator Wrapper device is responsible for the efficient communication and synchronisation of cGEM5 with the SystemC accelerator. For the memory-related tasks (e.g. transferring data from the CPU's memory to the accelerator's memory) we have mimicked the corresponding approaches from CUDA since those are widely used and thus the designers are already familiar with them. Our Accelerator Wrapper inherits all the GEM5 DMA device characteristics so that full DMA transactions utilizing the full operating system can be performed. In addition, it contains a large Device Memory to store the data from the OS *mempcy*, allowing for the accurate simulation of the DDR memory found in most of the real systems incorporating PCI-connected acceleration (e.g. FPGA-based) boards.

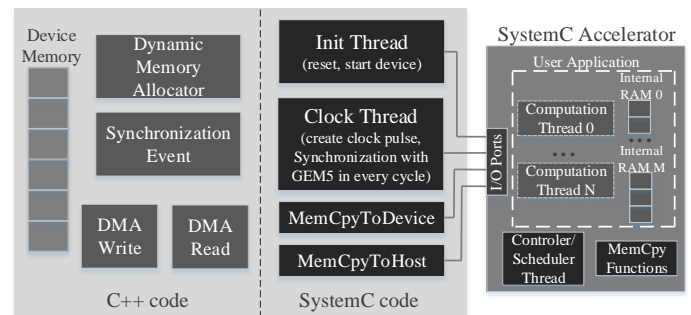


Fig. 5: Accelerator Wrapper Device

The Accelerator Wrapper consists of, in total, eight C++ and SystemC-thread modules as described below (Figure 5):

- 1) **Dynamic Memory Allocator C++ Module** - The Buddy dynamic memory allocation algorithm [23] is implemented in the Accelerator Wrapper to allocate and free Device Memory segments through the cGEM5 operating system; this module is similar to the *cudaMalloc* of NVIDIA's GPUs [24].
- 2) **DMA Write/Read C++ Modules** - Two Direct Memory Access engines have been developed so that the cGEM5 *dma_device* can efficiently transfer high data volumes from the Linux driver to the Accelerator Wrapper and vice versa, similar to the *cudaMemcpy* of NVIDIA GPUs [24]. The user can define, through one parameter, the delay of the DMA data transfer in order to achieve a realistic latency.
- 3) **Synchronisation Event C++ Module** - A cGEM5 synchronisation event function is implemented and it is triggered at every SystemC accelerator device cycle. This function checks whether the SystemC accelerator has reached the next simulation cycle. Finally, in case cGEM5 is faster than the SystemC accelerator¹, it reschedules the synchronisation event function to wait for the SystemC accelerator.
- 4) **Init SystemC Thread** - The initialisation thread is implemented in SystemC so as to generate the *reset* and *start* signals both of which are essential for SystemC's module execution. Furthermore, the thread calls the *sc_pause*² command when the SystemC acceleration task is finished.
- 5) **Clock SystemC Thread** - The clock thread is implemented in SystemC in order to generate the actual *clock* signal of the accelerator when called by cGEM5's OS; the *DeviceClock* option is used in order to define the clock frequency (e.g. --DeviceClock=200MHz). Moreover, at every SystemC cycle, the full synchronisation with cGEM5 is achieved by checking whether the cGEM5 has completed its tasks within this time frame; whenever required, the SystemC accelerator waits for the cGEM5; this is achieved, in our module, by using the *wait* SystemC function at *SC_FS* time granularity.
- 6) **MemCpy ToDevice/ToHost SystemC Threads** - The Two *MemCpy* SystemC threads developed pass the data from the Wrapper Device Memory to the corresponding synthesisable I/O ports, so that they eventually reach the SystemC accelerator. The user can define through one parameter the amount of data to be read/written in one SystemC cycle.

4.1.4 SystemC Accelerator

In order to simulate the hardware accelerator, the Accelera open-source libraries have been incorporated with the cGEM5 SCons construction tool [25]; this allows for the compilation and execution of complete system applications

1. This can happen when cGEM5 has not any processing work to do and SystemC accelerator has to manipulate a lot of threads.

2. *sc_pause* command is selected instead of *sc_stop* because the latest destroys all SystemC structures and as a result, the same SystemC module cannot be called twice.

running on top of ACSIM and utilizing hardware accelerators. Moreover, a reference accelerator module has been developed, in SystemC, in order to evaluate the Accelerator Wrapper and the Linux Kernel Drivers; this module, which is also provided in open-source, can also act as a reference for the designers/users that will develop their own SystemC accelerators on top of ACSIM.

It must be highlighted that the designer can fully simulate synthesizable SystemC, such as the one supported by Xilinx's Vivado HLS or Mentor's Catapult, since the interconnection port modules utilized in the Accelerator Wrapper, are fully synthesizable as well, as illustrated in Figure 5.

The reference SystemC accelerator module consists of a main SystemC thread and two SystemC functions as described below:

- **Controller/Scheduler SystemC Thread** - This is the main SystemC thread which is called by cGEM5's OS while the user has the ability to create as many individual cores as required so as to best serve his/her application (dashed-lines in Figure 5). These threads can be scheduled by the Controller/Scheduler SystemC Thread.
- **MemCpy ToDevice/ToHost SystemC functions** - Two SystemC *memcpy* functions have been implemented which are responsible for the transfer of data from/to the Wrapper Device Memory to/from the accelerator's internal memories. Those functions mainly call the corresponding Wrapper functions (*MemCpy SystemC Threads*), in order to allow for the efficient communication with the Accelerator Wrapper.

4.2 Accelerator Run-Time Environment

This section describes the interface of our integrated system, and how the users can develop their accelerators in SystemC and their application in C/C++ on top of the full operating system running on cGEM5.

Code segment 1 describes the header functions of the developed drivers library. The designers can simply include the *<AccelDriver.h>* in their file so as to utilize them. In more detail, we have implemented two functions (*AccelInitialization* & *AccelFinalization*) which handle the initialisation and the finalisation of the Accelerator Wrapper Device; in other words they initialise and terminate the Kernel Driver, Wrapper Memory Device, etc. Moreover, the *AccelInitialization* function must be called before any SystemC device call, while, the *AccelFinalization* function must be called at the end of the simulation of the SystemC accelerator.

Subsequently, two functions *AccelMalloc* and *AccelFree* were implemented in order to allocate and free space in the Accelerator Wrapper's Device Memory. Specifically, *AccelMalloc* uses the Buddy Memory Allocator to allocate space in the Device Memory, while a single queue has been implemented in order to keep the allocation segment characteristics. The parameters of this function are the allocation *size* (in bytes) and the *label* which is the name assigned to the corresponding segment. The last parameter is used to access the specific segment from within the code of the SystemC accelerator, as presented

in Code segment 2, during the *MemCpy* function. The *AccelFree* function frees up the segment that has been allocated by *AccelMalloc* while its input parameter is of type *DevMemAddr* which is effectively what has been returned by *AccelMalloc*.

Code Segment 1 Header of Accelerator Driver

```

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();

```

Function *AccelMemcpy* has been implemented so as to allow for the efficient copy of the data from cGEM5's memory to the Accelerator Device's Memory and vice versa. This function takes four parameters, *i*) *DevMemAddr* returned from *AccelMalloc*, *ii*) a pointer (void *) to the local data, *iii*) the *size* in bytes which should be copied and *iv*) the *TransferType*; for the last parameter, the user can employ either *SystemCMemcpyHostToDevice* or *SystemCMemcpyDeviceToHost*.

In addition, the *AccelCallDevice* function has been implemented which actually calls the SystemC accelerator and utilizes all the synchronisation mechanisms described in Section 4.1. It must be noted that both *AccelMemcpy* and *AccelCallDevice* functions have been implemented as **asynchronous** functions. As a result, the application running on cGEM5 (including the OS) can continue execute its computational tasks while the SystemC accelerator is also executing tasks. This is possible since the DMA engines used to transfer the Data from the Host to the Device (and vice versa) as well as the SystemC actual simulation are executed on different threads from the cGEM5 one, as analytically described in Section 4.1.

There are also the *MemCpy ToDevice/ToHost* SystemC functions which allow for the efficient data transfer from the Wrapper's Device Memory to the SystemC internal Memory (Code segment 2). The *MemCpy* function takes 5 parameters: *i*) the *name* of the SystemC internal memory, *ii*) the *label* (e.g. 'A'), as it has been declared in the *AccelMalloc* function of the application running on cGEM5, *iii*) the *SWOffset* (in the presented example it is 0) which is the offset of the wrapper Device Memory, *iv*) the *size* of the elements to copy and *v*) the *ElementType* which is the type of elements to be copied. Table 1 summarizes the element types which can be supported by our SystemC Device (the most widely-used types); moreover, it is a trivial process to add any other desired element type.

Moreover, Code Segment 2 shows some essential segments that should be placed on the SystemC file describing the accelerator module. The first eight lines should be placed in the Header SystemC file, while the remaining lines should be put in the implementation file. In the Header file, an internal memory of the accelerator (for example this can be an FPGA's BRAM) is declared in line 2 with a predefined

Data Type	Size	Data Type	Size
ACC_CHAR	1 byte	ACC_INT	4 bytes
ACC_FLOAT	4 bytes	ACC_DOUBLE	8 bytes
ACC_UINT8_T	1 byte	ACC_UINT16_T	2 bytes
ACC_UINT32_T	4 bytes	ACC_UINT64_T	8 bytes

TABLE 1: Type of Elements supported by the custom SystemC application

size, while *SC_CTHREAD* is utilized since it is an integral part of the synthesisable SystemC subset and uses the clock and reset signals that are provided by the Wrapper's Clock & Init threads respectively.

Code Segment 2 Segments of SystemC Application

```

1: SC_MODULE(SystemCDevice){
2:   int INTER_MEM[MEM_SIZE]; ▷ Create an Internal Mem
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: }

```

Finally, lines 9 to 17 describe the functionality of a simple reference accelerator module, using an infinite *while* loop. Since this process describes the intended hardware, the function of the thread never returns, keeping the thread always alive; in order to mark the end of a clock cycle and suspend the process until the next clock event, the *wait()* function is called.

5 VALIDATION AND PERFORMANCE ANALYSIS

This section provides the experimental results of our work. The evaluation of the ACSIM simulation framework has been analysed using light versions of two Linux distributions; Gentoo Base System for x86 processors and BusyBox for ARM ones. Specifically, Section 5.1 & Section 5.2 present the evaluation of the Processing & Network Simulator part of the COSSIM simulator in the context of a heterogeneous parallel system as well as the Performance analysis of the underlying COSSIM framework, upon which ACSIM is built, when utilizing multiple CPU cores in multiple distributed machines. Finally, Section 5.3 presents the evaluation and performance analysis of ACSIM using three real world use cases, while Section 5.4 describes the ACSIM validation. It should be noted that all the results for COSSIM presented here have not been presented in any other paper.

5.1 Evaluation of Processing Simulator sub-system

In terms of the performance of the Processing sub-system, this is based on the performance of GEM5 which varies greatly with the complexity of the processing system that is being simulated. A typical simple CPU (InOrder CPU) can be simulated at a rate of 1 to 3 MIPS (Million Instructions

/ Sec). More complex CPU structures (e.g. Out of Order-OoO CPUs) and memory subsystems can reduce the rate of simulated instructions to as low as 0.1 - 0.3 MIPS [26], [27].

The key concept of ACSIM’s approach is to execute the cOMNET++ simulator in a typical workstation, while the cGEM5 instances together with the simulators of the accelerators are executed in one or more servers (distributed simulation). For this reason, in the following experiments, the cOMNET++ simulation was executed on a workstation utilizing an Intel i5-4590 processor (3.3GHz) running Ubuntu Linux 14.04 with 16GB of RAM. On the other hand, the simulation of the CPUs and the accelerators as well as the HLA Server were executed on a cluster which contains 44 HP Proliant BL465c server blades with two AMD Opteron Model 2218 (2.6GHz), 4GB of RAM and Gigabit Ethernet per blade (totally 176 physical cores are contained with 176GB of RAM running Ubuntu Linux 16.04). The main configuration of the simulated CPUs is described in Table 2.

TABLE 2: The main configuration of the processors simulated

CPU Type	CPU/System Clock	Memory	L1/L1D/L2/L3 Cache Size
X86 Atomic (In Order)	2GHz/1GHz	DDR3 (2048MB)	32KB/64KB/2MB /16MB
ARM Atomic (In Order)	2GHz/1GHz	DDR3 (512MB)	32KB/64KB/2MB /16MB

Figure 6 illustrates the mean instructions per second simulated when several real-world applications are executed in the COSSIM system including the time needed for the negotiations and set-up of the corresponding networks interconnecting the different nodes for both X86 and ARM processors. As illustrated in Figure 6, the performance of the cGEM5 component of the ACSIM simulator is in line with the reported performance of the publicly available version of the simulator. Those numbers do not include any communication with the Accelerator Simulator. Our results demonstrates that as the number of nodes increases, the rate of simulated instructions (thus performance) decreases. This is mainly due to the Global Synchronization schemes for the 2-128 node experiments, while there is a steep performance drop when more than 176 cGEM5/Accelerator instances are executed on the 176-core cluster due to lack of processing resources. For the specific experiment, the synchronization interval is set at 10us. Consequently, after every 10us of simulated time, all nodes are halted so that they can all be synchronized and then resume operation. Apparently, this synchronization interval determines the accuracy of captured events. Figure 7 demonstrates the impact of the synchronization interval in the performance of the overall simulation, as measured by the wall-clock time required to complete a simulation with 16 nodes. There is a dramatic drop in performance for very short interval times (smaller than 100us) mainly due to the increased number of messages that cOMNET++ has to manipulate.

Furthermore, Figure 8 illustrates the simulation time required to boot the OSs with their network card configuration for 16-1024 ARM-based nodes and X86-based nodes using 10us synchronization interval. This is the part needed so as the full OSs is up and running and the simulation of the hardware accelerator can start. Based on those results

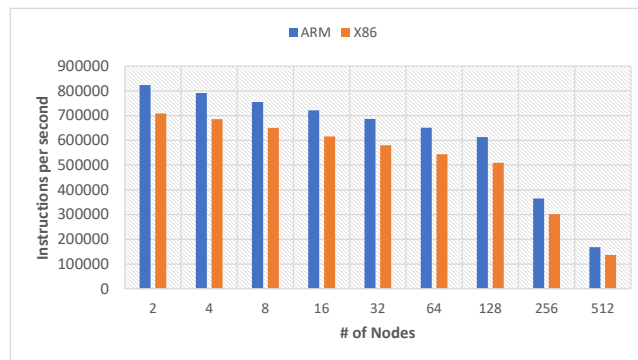


Fig. 6: Performance results using typical GEM5 configuration

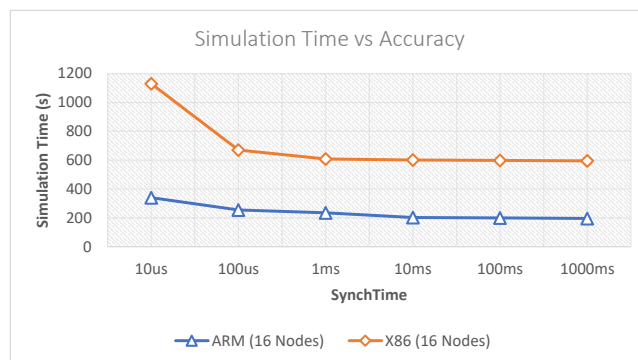


Fig. 7: Simulation time using different synchronization intervals

it becomes evident, as in the previous figure, that performance is heavily impacted when the number of cGEM5 instances spawned becomes higher than the number of physical processor cores of the distributed system on which the simulation is executed. Specifically, in the 16-128 nodes experiments, the simulation time is almost constant, while, after 176 cores, doubling the number of cGEM5s instances triggers an increase in the simulation time by a factor of two. Similar results are obtained in X86-based nodes which require more processing time to boot the OS. Summarizing, the effect of the network has a negligible impact on the overall performance; as a result, COSSIM has good scalability in case the number of cGEM5 instances is lower or equal to the number of the physical cores. As a result, the performance of the parallel combined Processing-Acceleration modules of ACSIM is expected to follow the same trend.

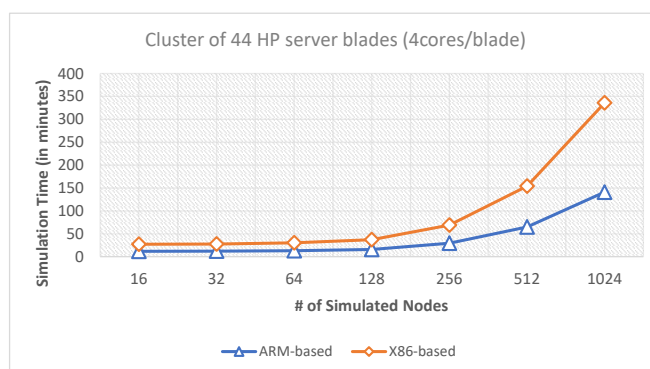


Fig. 8: Simulation time using a cluster of 44 server blades

5.2 Evaluation of Network Simulator Part

The network performance of our approach has been evaluated using the widely used Netperf 2.7.0 benchmark suite and specifically the very demanding TCP_RR benchmark. Netperf [28] was developed by Hewlett-Packard, while it is widely used in measuring the performance of many different types of networks and it provides tests for throughput, and end-to-end latency.

Request/Response performance is often overlooked, but it is just as important as bulk-transfer performance. While things like larger socket buffers and TCP windows, as well as stateless offloads like TSO (TCP Segmentation Offload) and LRO (Large Receive Offload) [29] can cover numerous latency and even path-length sins, those sins do surface in the request/response tests. While in a bulk-transfer test the reporting metric is the units of bits (or bytes) transferred per second, TCP_RR test reports the transactions per second where a transaction is defined as the completed exchange of a request and a response.

Figure 9 illustrates the transactions per second in the case of two COSSIM-simulated x86 systems and two Real x86 systems; the configuration consists of one server and one client running the netperf TCP_RR benchmark and connected through Gigabit Ethernet. In more detail, different request/response packet sizes have been evaluated ranging from 1 byte to 10K bytes within 1 second³ of total time. The following command shows the netperf TCP_RR configuration which is used for a request/response packet size of 100 bytes, while Table 3 summarizes all the parameters of our experiment.

```
netperf -H IP -t TCP_RR -l 1 -- -r 100,100 (1)
```

TABLE 3: TCP_RR benchmark Parameters

Parameter	Description
-H [IP]	Set the name of the remote system
-r [req,resp]	Set the request and response packet size
-l [testlen]	Controls the time of the requested test

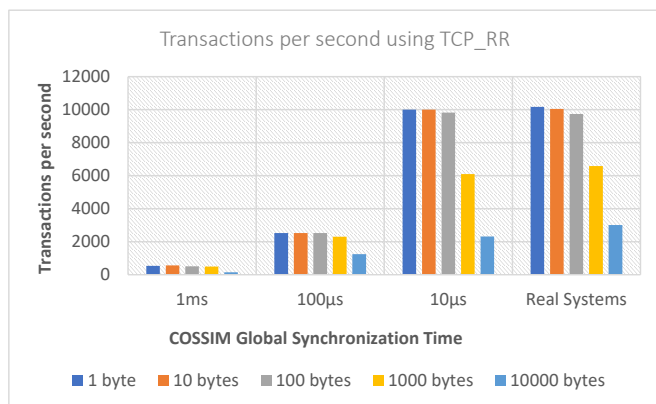


Fig. 9: Transactions per second using 1server/1client with different Global Synchronization intervals

In all the TCP_RR experiments, the synchronization per node interval is set at 10us so as to achieve full Gigabit

3.1 second was selected as adequate quantity of time to reach the maximum speed of experiment.

Ethernet rates between the nodes. Specifically since the maximum Ethernet packet is 1536bytes [30], by selecting a synchronization period of 10us, each node can receive up to 1536bytes/10us or 153.600.000bytes/sec or 1,2Gbits/sec so it can support full Gigabit Ethernet speed. On the other hand, three different intervals are selected for Global Synchronization from 10us to 1ms. Figure 9 demonstrates that as the Global Synchronization decreases, more accurate results are obtained (i.e. much closer to the Real Systems' ones), while when using a Global Synchronization of 10us, the transactions/second is very similar to those achieved by the Real Systems for all request/response sizes. Summarizing, in case the Global Synchronization interval is less than or equal to the Synchronization per node interval, ACSIM guarantees that the simulation is fully accurate.

5.3 Evaluation of ACSIM

In order to evaluate the efficiency of our novel approach we have used two SystemC-described accelerators and the ACSIM approach is compared with the conventional manual approach; both simulations approaches are illustrated in Figure 10. In general, the integrated simulation utilizing a full-system simulator and a SystemC one is not a trivial process mainly due to the data exchange needed between the two stand-alone simulators.

Based on the existing conventional flow, the application executed in the processing simulator (such as GEM5) writes the data that should be fed to the SystemC simulator in files, denoted as Steps 1 & 2 in Figure 10. Then, the full-system simulator has to terminate or stall its execution so that the SystemC simulator can read the data from those files without any synchronization problem, i.e. Step 3.

In contrary, our novel ACSIM method resolves efficiently all those issues since the full-system simulator boots the OS only once and, since it communicates with the SystemC simulator through programmed I/Os and DMA engines, full global synchronisation is supported. Moreover, the proposed approach is orders of magnitude faster than the conventional one as demonstrated in Figure 11.

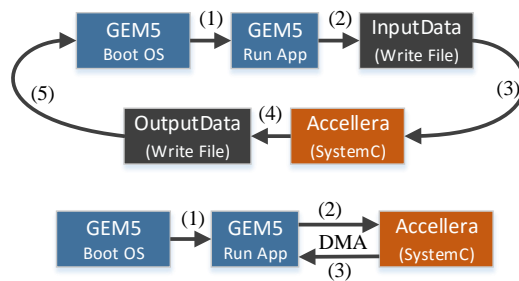
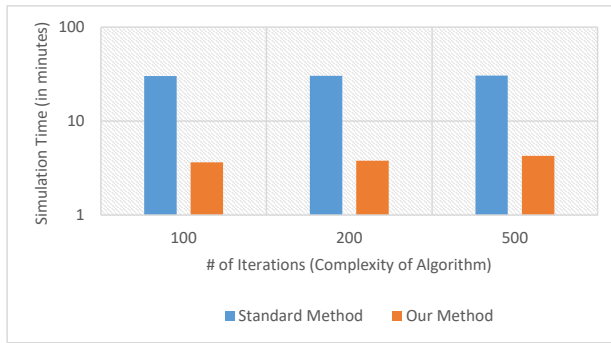
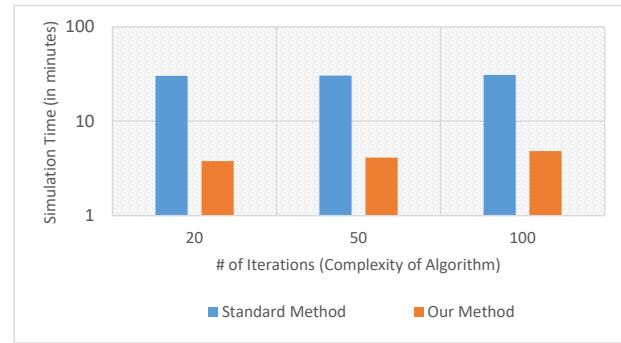


Fig. 10: (i) Standard (top) & (ii) Our novel method efficiently integrating GEM5 Full System Simulator with Accellera Simulator (bottom)

The first two use cases that are utilized in the evaluation of our ACSIM are accelerators that implement the Mutual Information (MI) [31] & Transfer Entropy (TE) [32] algorithms. The main reason behind the selection of those two use cases is that the complexity of the accelerator hardware implementing them depends on the number of iterations supported in each hardware core. The MI and TE results

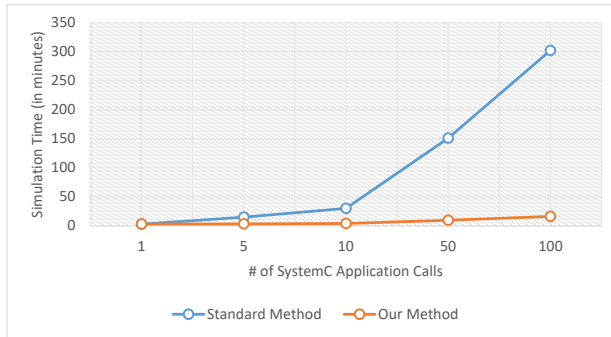


(a) Mutual Information Algorithm

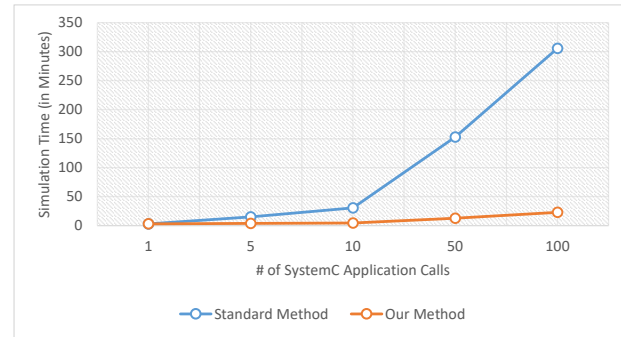


(b) Transfer Entropy Algorithm

Fig. 11: Simulation time of (i) standard and (ii) our method using two use cases (10 # of Accelerator calls are used)



(a) Mutual Information Algorithm



(b) Transfer Entropy Algorithm

Fig. 12: Simulation time of two use cases using different # of Accelerator calls

are based on a typical data set of 40K samples while the number of iterations range from 100 to 500 for MI and from 20 to 100 for TE, as shown in Figure 11; these ranges have been selected due to the fact that they constitute typical values in order to get high accuracy results. In addition, the results of Figure 12 focus only on the two high-end values for both ranges, i.e. 500 iterations for MI and 100 iterations for TE since those are the worst cases for our system. In those two first experiments we concentrate on a single node setting (i.e. just a cGEM5 instance connected to the Accellera simulator).

Figure 11 first illustrates the overall simulation time when calling 10 times the SystemC accelerator (i.e. our MI and TE SystemC code executed on top of the Accellera Simulator). It becomes apparent that the proposed method is one order of magnitude faster than the conventional file-based method in all cases and that is because the cGEM5 simulator boots the OS only once. Furthermore, the overall simulation time does not radically change in this case since the simulation of our very small SystemC-based accelerators is orders of magnitude faster than the actual time needed by cGEM5. Moving to more complicated accelerators (such as for example a TE module handling 100 iterations) we clearly see the efficiency of our approach since the complex code takes slightly more time than a code half or even 5 times smaller (i.e. cores handling 50 or 20 TE iterations respectively) due to our novel synchronization approach.

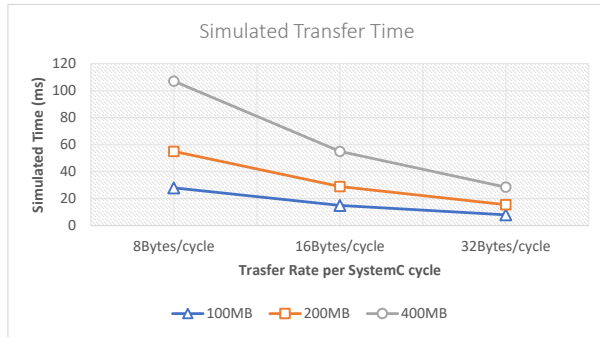
Figure 12 compares the overall simulation times between the conventional and the proposed method using a different number of SystemC accelerator calls, when the complexity of the accelerators remain constant (i.e for the same number of iterations). For the case of the first call, our method needs

slightly more time due to synchronisation issues, in the order of a second, however, as the number of calls to the accelerator increases, it gradually surpasses and eventually dominates over the conventional method. For example, our method is one order of magnitude faster for all three last cases (10 or more SystemC accelerator calls) for both the MI and the TE hardware cores.

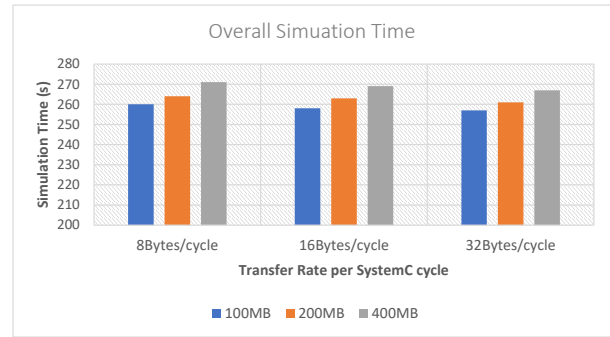
In order to evaluate the proposed host/device inter-communication scheme, we measure both the simulated transfer time from cGEM5 to our accelerator as well as the overall ACSIM simulation time as illustrated in Figure13a and Figure13b respectively. Specifically, in our experiments, three different amounts of data have been used, from 100MB to 400MB, using three different transfer rates (from 8Bytes up to 32Bytes per SystemC cycle⁴). In case 100MB of data is to be transferred at 8Bytes/cycle, for example, ACSIM reports 28ms as simulated time; when the accelerator is working at 500MHz we needed 100MB/8Bytes = 12,5M cycles or 25ms plus the overhead for the transaction initialization and as a result, the simulated time reported by ACSIM is accurate. In addition, as illustrated in Figure13a, the simulated time is decreased approximately by 1,93x when doubling the transfer rate and this is due to the overhead for the initialization of the transaction. Finally we can observe from Figure13b that ACSIM has negligible overhead in the overall simulation irrespective of the amount of data transferred.

Moving to the overall evaluation of ACSIM it is based on a Reservoir Simulation (RS) application which is the state-of-the-art technology used to predict the oil-field perfor-

4. The user can define the transfer rate through a parameter.



(a) ACSIM Simulated Transfer Time



(b) Overall Simulation Time

Fig. 13: ACSIM Simulated & Overall Simulation Time using different transfer rates

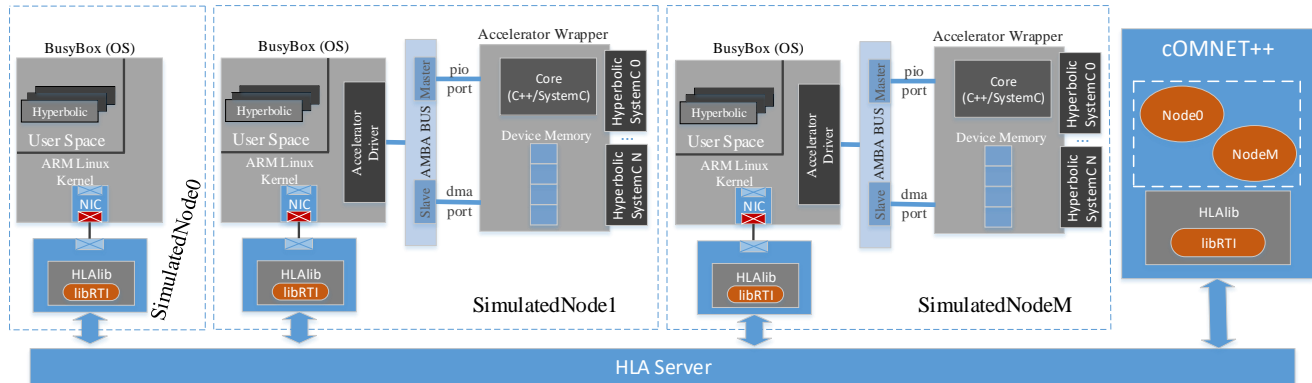
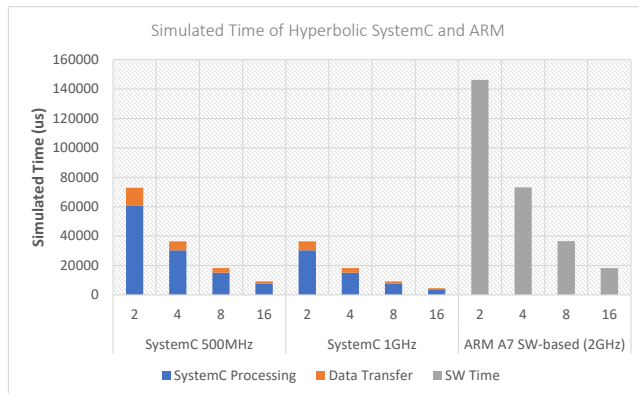
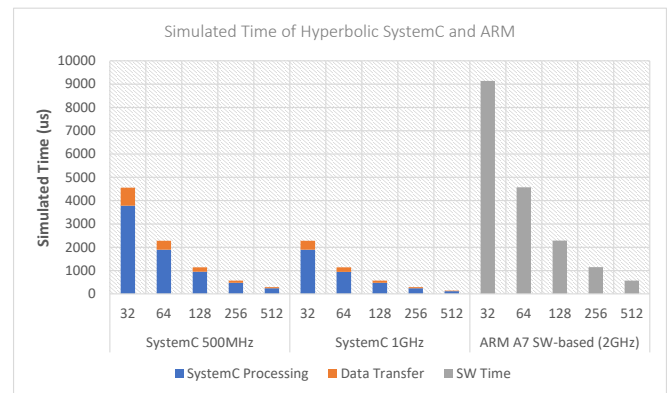


Fig. 14: A representation of ACSIM simulator using Hyperbolic SystemC cores

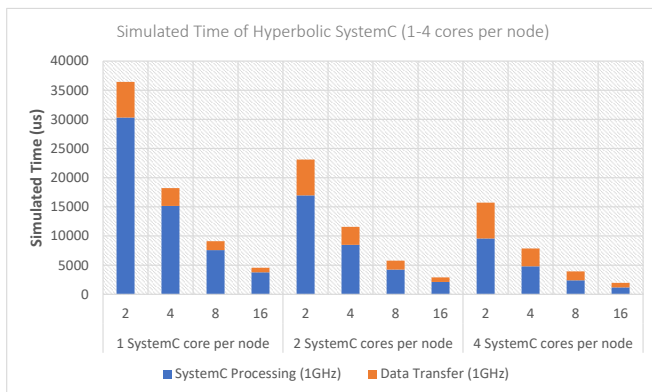


(a) 2-16 Simulated Nodes

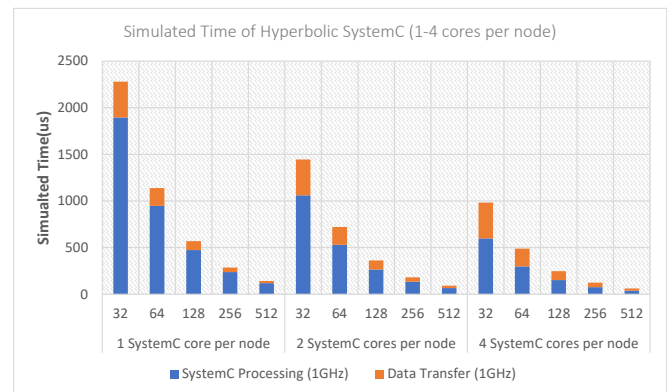


(b) 32-512 Simulated Nodes

Fig. 15: Simulated Time of Hyperbolic SystemC Processing and Data Transfer at 500MHz and 1GHz versus ARM SW-based version at 2GHz using 800K samples



(a) 2-16 Simulated Nodes



(b) 32-512 Simulated Nodes

Fig. 16: Simulated Time of Hyperbolic SystemC Processing and Data Transfer at 1GHz using 1-4 SystemC cores per node

mance under several possible production schemes. Within this application we have developed a SystemC-based accelerator of the Hyperbolic method. The Hyperbolic algorithm facilitates the calculation of the Rachford-Rice [33] (RR) equation which is effectively responsible for providing insight as to the flow of liquids within an oil reservoir and it is used extensively in the field of oil Reservoir Simulation. RR is a variation of the Newton-Raphson [34] method that provide results at less iterations and, therefore, in less time.

Figure 14 illustrates the ACSIM system when simulating the overall RS system including the Hyperbolic acceleration cores. Specifically, each processing node represents an ARM-based System running BusyBox OS tightly interconnected with two SystemC instances of the Hyperbolic algorithm. ARM-based systems are simulated since they have been recently introduced in the ARM server domain [35], [36]; obviously ACSIM seamlessly supports x86-based systems as well. In node0 a single server (producer) is simulated which initializes the overall processing tasks and distributes the data among the rest of the nodes (consumers) through real network protocols (i.e. both Ethernet and wireless). The consumer nodes, in turn, send the corresponding data to the hardware accelerators and after the processing is complete, they return the results back to the Server.

Figure 15 demonstrates the simulated time reported when the whole application is simulated without the accelerators (grey bars) and when ACSIM simulates the accelerators. Those numbers are from 2-512 consumer nodes using 800K samples and 10us as global synchronization interval. The experiment is executed in the cluster described in Section 5.1.

Figure 15, demonstrates that the simulated time is decreased by a factor of two, every time we double the number of consumer nodes to which the data are distributed and processed. Moreover as we can see if the accelerators are simulated at 500Mhz the overall latency reported by ACSIM is 2x times smaller when compared with the purely S/W scenario (this includes the simulation of the required data transfer from the CPU to the Accelerator and back⁵ which is denoted with the Orange line). When doubling the clock frequency of the simulated accelerators we get a 2x increase in the speedup as well. This is inline with the trend reported by a real implementation of the application in FPGA-based parallel heterogeneous systems as described in [37] and show the efficiency of our approach.

In Figure 16 we illustrate the simulated time for the same application when we have from 1 to 4 accelerator cores per processing node. In this experiment we can see that the application latency reported is decreased approximately by 1.7x when we double the accelerators per node; this is a reasonable number since the application execution also involves the data transfer from the single processing system to multiple Accelerators' memories.

From all the above it is clear that ACSIM is a very useful tool for Design Space Exploration in highly parallel heterogeneous systems since different number of nodes, different number of accelerators per node and different

5. In our simulation the DMA engine is used to transfer the data from the memory of cGEM5 to the Accelerator's DRAM, and subsequently the SystemC main thread reads/writes two memory elements in every SystemC cycle.

clock frequencies of the accelerators can all be seamlessly simulated and evaluated.

5.4 ACSIM Validation

In order to validate the ACSIM simulation environment, we ported our application to the Trenz TE0808 boards [38] which utilize the UltraScale+ Xilinx FPGA. In addition the corresponding TEBF0808 carrier board [39] is used allowing a number of peripherals capabilities such as PCI-E communication as illustrated in Figure 17. We have implemented from 1 to 4 accelerator Hyperbolic cores per processing node in this FPGA-based system and the output results we got from the board were identical with the ones reported by the ACSIM and listed in the previous section.

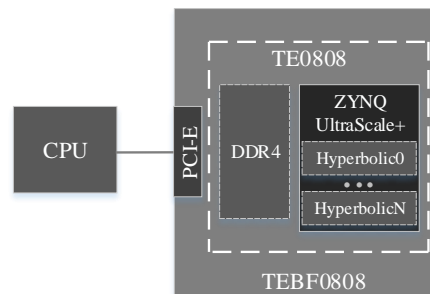


Fig. 17: ACSIM Validation using Trenz TE0808 board

6 CONCLUSIONS

In this paper, we present the ACSIM Simulation Framework, an open-source novel solution that addresses, in a single integrated tool-set, the simulation of the processing and the networking parts of highly parallel systems integrating custom hardware accelerators.

Our novel approach allows for the first time for global synchronization along the three (i.e. Network, Processing and Hardware Accelerator) simulation domains. Such an integrated approach can severally reduce the time needed for design space exploration while also accelerate the overall development and verification process.

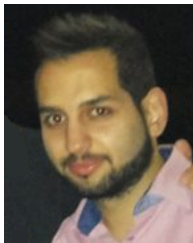
Based on a number of experiments the ACSIM simulator proved to be an order of magnitude faster, in the development process, when compared with the existing conventional approach, while it produces very accurate results in real-world applications. Moreover, it is, to the best of our knowledge, the first ever simulator that can efficiently simulate highly parallel systems (such as those used in Cloud infrastructures) containing several hundreds of processing nodes each one interconnected with a number of hardware accelerators.

As future work, and in order to further extend the users of the Simulation framework, we aim to support additional architectures incorporating accelerators (e.g. accelerators that can get data fully from user-space queues without going through the OS as well as PCI accelerators which have host-visible device memory and device-visible host memory.) In addition, we aim to integrate any newly developed GEM5 NIC so as to be able to natively support different interconnection network infrastructures (e.g. InfiniBand).

In order to further increase the impact of this work, the complete source code (Linux drivers, Accelerator wrapper, etc.) are freely distributed to the community [40].

REFERENCES

- [1] N. Tampouratzis, A. Brokalakis, A. Nikitakis, S. Andrianakis, I. Papaefstathiou, and A. Dollas, "An open-source extendable, highly-accurate and security aware cps simulator," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2017, pp. 81–88.
- [2] Amazon ec2 f1. [Online]. Available: <https://aws.amazon.com/ec2/instance-types/f1/>
- [3] Alibaba cloud. [Online]. Available: <https://www.alibabacloud.com/>
- [4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
- [5] F. Fittkau, S. Frey, and W. Hasselbring, "Cdosim: Simulating cloud deployment options for software migration support," in *2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, 2012, pp. 37–46.
- [6] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, Dec 2011, p. 105.
- [7] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 446–452.
- [8] B. Louis, K. Mitra, S. Saguna, and C. Hlund, "Cloudsimdisk: Energy-aware storage simulation in cloudsim," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec 2015, pp. 11–15.
- [9] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, Dec 2010, pp. 1–5.
- [10] D. Meisner, J. Wu, and T. F. Wenisch, "Bighouse: A simulation infrastructure for data center systems," in *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software*, ser. ISPASS '12, 2012, pp. 35–45.
- [11] P. O. Sberg, H. Groenda, S. Wesner, J. Byrne, and D. S. N. et al., "The cactus vision of context-aware cloud topology optimization and simulation," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, Dec 2014, pp. 26–31.
- [12] P. T. et al., "Hardware acceleration for similarity measurement in natural language processing," in *Low Power Electronics and Design (ISLPED)*, *2013 IEEE International Symposium on*, Sept 2013, pp. 409–414.
- [13] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Y. Wei, and D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.
- [14] M. Yu and J. S. et al., "A fast timing-accurate mpsoc hw/sw co-simulation platform based on a novel synchronization scheme," in *Lecture Notes in Engineering and Computer Science*, 2010.
- [15] C. Menard, J. Castrillon, M. Jung, and N. Wehn, "System simulation with gem5 and systemc: The keystone for full interoperability," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2017, p. 62.
- [16] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, p. 380.
- [17] The gem5 simulator. [Online]. Available: <http://gem5.org/>
- [18] Omnet++ discrete event simulator. [Online]. Available: <https://omnetpp.org/>
- [19] (2010) Ieee 1516-010 - standard for modeling and simulation high level architecture - framework and rules. [Online]. Available: <https://standards.ieee.org/findstds/standard/1516-2010.html>
- [20] (2015) Certi project. [Online]. Available: <http://savannah.nongnu.org/projects/certi>
- [21] "Accellera SystemC wiki website," <https://en.wikipedia.org/wiki/Accellera>
- [22] *IEEE 1666 Standard SystemC Language Reference*, IEEE.
- [23] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.
- [24] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.
- [25] "SCons: A software construction tool official website," <http://scons.org/>.
- [26] A. Saidi, "Accelerating Simulation with Virtual Machines," 2012. [Online]. Available: http://gem5.org/File:2012_12_gem5_workshop_kvm.pdf
- [27] T. Carlson, "Full-System Simulation at Near Native Speed," 2015. [Online]. Available: http://www.gem5.org/wiki/images/4/4f/2015_ws_11_20150614_-_Trevor_E_Carlson_-_gem5_workshop.pptx
- [28] *Network Performance Benchmark*, HP Networking Performance Team, 2015. [Online]. Available: <https://hewlettpackard.github.io/netperf/>
- [29] J. Corbet, 2007. [Online]. Available: <https://lwn.net/Articles/243949/>
- [30] "Ieee standard for ethernet," *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, pp. 1–4017, March 2016.
- [31] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York, NY, USA: Wiley-Interscience, 1991.
- [32] T. Schreiber, "Measuring information transfer," *Phys. Rev. Lett.*, vol. 85, pp. 461–464, Jul 2000. [Online]. Available: <http://link.aps.org/doi/10.1103/PhysRevLett.85.461>
- [33] "The negative flash," *Fluid Phase Equilibria*, vol. 53, pp. 51 – 71, 1989, proceedings of the Fifth International Conference.
- [34] M. Abramowitz, *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables.*, Dover Publications, Incorporated, 1974.
- [35] R. Ammendola, A. Biagioni, P. Cretaro, O. Frezza, and F. L. C. et. al., "The next generation of exascale-class systems: The exanest project," in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 510–515.
- [36] S. RUBENOFF (2017) The year in cloud: Microsoft invests in arm, fpgas and gpus. [Online]. Available: <https://datacenterfrontier.com/the-year-in-cloud-microsoft/>
- [37] K. Georgopoulos, A. Ioannou, P. Malakonakis, I. Mavroidis, L. Lavagno, and I. Papaefstathiou, "Hls algorithmic explorations for hpc execution on reconfigurable hardware ecoscale," *ARC 2018*, May 2018.
- [38] Trezn te0808. [Online]. Available: <https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/TE08XX-Zynq-UltraScale/>
- [39] Trezn tebf0808 baseboard. [Online]. Available: <https://wiki.trenz-electronic.de/display/PD/TEBF0808>
- [40] N. Tampouratzis, "Acsim - a novel, simulator for heterogeneous cloud systems that incorporate custom hardware accelerators," <https://github.com/ntampouratzis/ACSIM>, 2018.



Nikolaos Tampouratzis is a researcher at Aristotle University of Thessaloniki, working on simulation tools for computing systems. He has joined Telecommunication Systems Institute, Technical University of Crete since October 2012 as a research associate, providing research and development services to several EU-funded research projects. He received his BSc from University of Crete, and MSc and Ph.D from Technical University of Crete with specialization in Computer Architecture and Hardware Design.



Ioannis Papaefstathiou received the Ph.D. degree in computer science from the University of Cambridge, Cambridge, U.K. He is a Manager of the HPC and Embedded Systems Group of Synelxis Solutions SA and an Associate Professor at the ECE School, at Aristotle University of Thessaloniki, Greece. His current research interests focus on architectures and efficient implementations of HPC, CPS and reconfigurable systems.