TECHNICAL UNIVERSITY OF CRETE

ELECTRONIC AND COMPUTER ENGINEERING DEPARTMENT

MICROPROCESSOR & HARDWARE LABORATORY

A novel Simulator for Heterogeneous Parallel and Distributed Systems

NIKOLAOS G. TAMPOURATZIS

DISSERTETION SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS OF THE DEGREE OF

DOCTOR OF PHILOSOPHY

AT

TECHNICAL UNIVERSITY OF CRETE

CHANIA, GREECE

JUNE 2018

Doctoral Thesis Committee

Thesis Statement

The astonishing growing development of Heterogeneous Parallel Systems and CPS trigger an emergence need of efficient simulators for such platforms, simulators that are extremely complex and processing hungry.

Abstract

In an era of complex networked heterogeneous systems, simulating independently only parts, components or attributes of a system-under-design is not a viable, accurate or efficient option. By considering each part of a system in an isolated manner, and due to the numerous and highly complicated interactions between the different components, the system optimization capabilities are severely limited. One of the main problems Cyber Physical Systems (CPS) and Highly Parallel Systems (HPS) designers face is the lack of simulation tools and models for system design and analysis. This is mainly because the majority of the existing simulation tools can handle efficiently only parts of a system (e.g. only the processing or only the network). Moreover, most of the existing simulators need extreme amounts of processing resources while faster approaches cannot provide the necessary precision and accuracy.

On top of that, the growing use of hardware accelerators in both embedded systems (e.g. mobile phones) and high-end systems (e.g. HPC/Cloud systems) 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 systems, software design can proceed in parallel with hardware development which will result in the reduction of the so important time-to-market. The main problem, however, is that such systems do not exist; most current 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.

In this thesis we present the COSSIM simulation framework which is the first known open-source, high-performance simulator that can handle holistically system-of-systems including processors, peripherals and networks; such an approach is very appealing to both CPS and Highly Parallel Heterogeneous Systems designers and application developers. In the context of COSSIM, a novel intercommunication and synchronization scheme is developed which is fully compliant with the IEEE HLA standard. Our highly integrated approach is further augmented with accurate power estimation that can tap on all system

components and perform analysis of the overall system under design something which was unfeasible up to know.

In addition, we present the ACSIM framework which is the first known open-source, high-performance simulator that can handle holistically system-of-systems including processors, peripherals, networks and 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. Finally, a sophisticated Eclipse-based GUI has been developed to provide easy simulation set-up, execution and visualization of results.

COSSIM and ACSIM have 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. Finally, the presented ACSIM interconnection scheme between the Processing and the SystemC simulators is orders of magnitude faster than the existing solutions, while our simulation framework can efficiently simulate up to several hundreds of processing nodes with hardware accelerators interconnected together, in a full distributed manner.

Acknowledgements

I am especially grateful to my supervisor, Associate Professor Ioannis Papaefstathiou for his guidance and support during all my years in my Ph.D. I would like to thank Antonios Nikitakis, Andreas Brokalakis, Konstantinos Georgopoulos, Pavlos Malakonakis for the excellent collaboration and their valuable contribution in this thesis work. Moreover, I would like to thank Professors Dionisios Pnevmatikatos, Apostolos Dollas, Kostas Kalaitzakis, Manolis Katevenis, Dimitrios Soudris, Pavlos Mattheakis for their co-advising and their participation in my Ph.D. Thesis committee. Finally, I would like to thank my friends and family, my father George, my mother Irene, my brother Manos, as well as Xrysa for their love, support and encouragement. This work was carried out with the financial support of Telecommunication Systems Institute (Chania Crete Greece) in the framework of the COSSIM and ECOSCALE H2020 European projects with scientific support of Ioannis Papaefstathiou.

Contents

List of Tables

List of Figures

1 Introduction

High Performance Computing (HPC), Cloud Computing and Cyber Physical Systems (CPS) are considered “The Next Computing Revolution” after Mainframe computing (60’s - 70’s), Desktop computing & Internet (80’s - 90’s) and Ubiquitous computing (00’s).

High performance computing refers to the computing system, including several processors as part of a single machine or a cluster of several computers as an individual resource. It owes its feature of high speed computing to its great ability to process information. Therefore, the main methodology that is currently applied to high performance computing is parallel computing. In short, high performance computing is legendary for its processing capacity.

Cloud computing is based on the utility and consumption of Internet services, usually offering visualized resources with dynamic extensible performance through the Internet. According to the definition of the National Institute of Standards and Technology [1], cloud computing is based upon pay-per-use model, allowing available and required network. It allows convenient network access to configurable shared pool of computing resources. These resources include networks, servers, storage, applications and services. The users can access the data center and carry out computing on demand with remote hosts and mobile devices.

Cyber-physical System (CPS) is a system of systems which tightly couple the computing components between the physical components, governed by the underlying processes and policies [2]. So that, CPS integrates networked computational resources into physical processes in order to add new capabilities into the original system and realize real-time perception, dynamic control, information services in large-scale projects.

1.1 Motivation

Nowadays, both Highly Parallel and Distributed Computing Systems (i.e. Clouds and HPC systems) and Cyber Physical Systems (CPS) are growing in capability at an extraordinary rate, incorporating processing systems that vary from simple microcontrollers to high performance units and hardware accelerators connected with each other through numerous networks. In order to meet the growing requirements of automated and assisted features and the inherent complexity, developers have to model and simulate those sophisticated systems at all the design stages. However, 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 hardware accelerators, accurate overall system timing, power/energy estimations and network deployment issues.

On top of that, software design is severally influenced by the high complexity of the new hardware systems, which forced the emergence of new software that can take advantage of the novel features incorporated into 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 thesis, we present the COSSIM Simulation Framework which is an open-source framework that aims to address all the aforementioned limitations. The proposed solution 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, realistic communication modeling of the different networks that are employed and power/energy consumption estimates for both the processing elements and the network based on the actual dynamic usage scenarios.

In addition, ACSIM Simulation Framework is presented which is an extension of COSSIM. Specifically, ACSIM introduces a novel flow that enables the designer to rapidly prototype synthesisable SystemC hardware accelerators in conjunction with the Processing systems without worrying about communication and synchronization 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.

1.2 Contribution

The contributions of this work come mainly from the implementation of highly integrated COSSIM simulation framework (and ACSIM extension) which is the first known open-source, high-performance simulator that can handle holistically system-of-systems including processors, peripherals, hardware accelerators and complex networks; such an approach is very appealing to both Highly Parallel Heterogeneous Systems and CPS designers and application developers. COSSIM/ACSIM offers a series of unique features and advantages compared to what is available in the market, that we believe will make the framework especially attractive as summarized in the following paragraphs.

A Novel High-Performance integrated simulation framework. The COSSIM/ACSIM simulation framework can substantially accelerate the process of HPC, Cloud, CPS systems, applications design and simulation in two significant ways:

- By being a distributed simulation tool, COSSIM/ACSIM can leverage the existing infrastructure to design the next generations of systems. For example, an available highly parallel system can be effectively used to provide the computational resources to design its next iteration. Previously available tools (including the tools that COSSIM/ACSIM also uses, such as GEM5) do not offer this scaling potential and as such they are prohibitive to use. More specifically, COSSIM/ACSIM implements a synchronization and interconnection mechanism to execute in parallel all instances of the processing simulator (a distributed simulation is also available as an option).
- Compared to using tools such as the ones that COSSIM/ACSIM includes, the integrated approach offers seamless real-time communication between components and thus almost eliminates the process of converting results from one tool to inputs for another and setting up series of sequential simulations in order to complete the simulation of all aspects of a System.

A direct result of the increased performance of the framework, is that it enables the much faster completion of the design process of HPC, Cloud and CPS applications or systems. Apparently, a shorter time from concept to market translates also to significantly reduced overall costs.

Additionally, as a consequence of the high accuracy of results that COSSIM/ACSIM provides, system and application tuning can be performed to a large extend before an actual deployment of an HPC, Cloud or CPS system. This reduces significantly the time between a parallel or distributed system is brought up and actual production code is able to be executed efficiently on top of it.

A Novel Application-Centric Approach to Design Highly Parallel Heterogeneous and Cloud Systems. The COSSIM/ACSIM simulation framework focuses on the precise simulation of a specific application. It does not simply model application behavior, since it actually executes all parts and tasks of the application. Therefore, it provides a way to uncover all subtle characteristics of the application and as such provide great insights regarding performance, as well as experimentation to uncover the best possible system infrastructure to support it. Compared to other tools that generally rely on traces or network behavior (as described in detailed in the next chapter), COSSIM/ACSIM can be slower and less general. However, this is only true when general Cloud or HPC systems are designed and currently the trend seems to follow a different path: designing the infrastructure for specific applications in mind. This is where COSSIM/ACSIM excels and can provide excellent value. Additionally, it is the tool that can help Cloud and HPC application designers the most. Typically, those developers have to work with system specifications rather than actual hardware (because they either try to evaluate the cost of a cloud solution or design an application for a parallel system not yet released) and a precise framework, such as COSSIM/ACSIM, can be the best solution to their needs.

A Novel Approach Covering a Gap in Existing CPS Simulation Tools. The COSSIM CPS simulation framework is a collection of already available tools, tightly connected together in a single package. Although there is no novelty in the components of the package itself, the framework proposition as a CPS simulation tool that can accurately simulate the performance, timing and functional behavior of system of systems, estimate power consumption per node and per overall system, provides functionality that no other CPS tool on the market currently offers. Popular CPS simulation tools focus more on CPS functional design by providing models of computation and physical process but not modeling of the actual devices and networks. As such, COSSIM works complimentary to those tools and can be used to verify the code and concepts derived through them without requiring an actual CPS system to be deployed.

Our Approach Provides Reduced Design Effort. After an initial time required to learn how to use the framework and setup simulations, the tight integration of all components in the framework provides a coherent and straightforward way to automate the whole process. The COSSIM framework does not alter the main properties of its components and as a result a designer can take full advantage of previous knowledge (or work) on the widely-used and documented tools, while at the same time he/she does not have to deal with the tedious task of passing information from one tool to the other, properly converting inputs and outputs to compatible representations and performing a simulation in discrete steps (e.g. first simulating a component and then a model of the network). As a result, the simulation process becomes more fluid and productivity rises significantly. In addition, the community has already knowledge of processing and network subsystems which are used, and this knowledge can be leveraged with COSSIM. Finally, our novel approach (i.e. ACSIM) allows for the first time for global synchronization along the three (i.e. Processing, Network 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.

2

Parallel Systems Simulators and COSSIM/ACSIM Approach

The COSSIM (as well as the ACSIM) simulator can be utilized mainly in the development of three classes of systems; HPC, CPS and Cloud systems. This chapter presents an overview of parallel systems simulators from the above domains as well as analytically comparison with our approach. Specifically, Section 2.1 analytically presents the available tools for CPS simulation, while Section 2.2 and Section 2.3 present the state of the art tools in Cloud and HPC domains respectively.

2.1 CPS Simulation Tools

Starting from the CPS domain, the available tools for CPS simulation fall under two main categories. The first one focuses mostly on the functionality of the system under design and thus they try to model different entities in a CPS system: a physical process, an electromechanical physical component, user behavior, events, message exchanges between components etc. Such tools are mostly based on the use of well-defined models of computation. Among them the most profound ones are Ptolemy [3], Matlab Simulink [4] and Modelica-based[1] simulation environments. While these tools can model physical processes, they can only be used during the initial design phase of the CPS application (they can even generate application code) since they mainly offer functional simulation. As a result, they cannot handle cycle-accurate processing simulation, power consumption estimations of the actual processing components nor the simulation of the actual networks that are going to be employed.

Simulators that do handle those aspects can typically be found in the field of Wireless Sensor Networks (WSNs), which is a certain subset of CPS. Most of these tools are designed around a specific WSN-related operating system or a specific device. However, these simulators either support only very simple microcontrollers (for example *ATMega128*) and Real Networks (wireless *IEEE 802.11* and/or *802.15.4* protocols) or very complex CPUs (ARM, MIPS, PowerPC etc.) and dummy network communication.

In the following subsections, the available tools of WSN domain which support both processing and network analytically presented. However, none of them can simulate both the processing and network sub-systems of an actual CPS application.

2.1.1 TOSSIM (extensions: PowerTOSSIM)

TOSSIM [5] is an emulator specifically designed for WSN running on top of the TinyOS, which is an open source operating system targeting embedded low-end systems. TOSSIM is

[1] Modelica is an object-oriented, equation-based language used to construct models of systems. Multiple free or commercial simulation environments that can import Modelica models exist such as OpenModelica, JModelica, CyModelica while even conventional physical simulators, such as Simulink, can handle Modelica models.

a bit-level discrete event network emulator built in Python, a high-level programming language emphasizing code readability, and C++. It includes models for very simple CPUs (*ATMega128* microcontroller), analog-to-digital converters (ADCs), clocks, timers, flash memories and radio components. The network communication over the wireless channel is abstracted as a directed graph, in which each vertex is a processing node, and each edge has a bit error probability. Each processing node has a private piece of state representing what it hears on the radio channel, thus this abstraction allows testing under perfect transmission conditions (bit error rate is zero).

However, this emulator has some limitations. Firstly, TOSSIM is designed to simulate behaviors and applications running on top of *TinyOS* and developed only in *nesC* language, and it is not designed to simulate any other applications/network protocols. Therefore, TOSSIM cannot correctly simulate several issues of the energy consumption in WSN; users can adopt *PowerTOSSIM*, another *TinyOS* simulator extending the power model of TOSSIM, to estimate the power consumption of each node. Furthermore, because TOSSIM is specifically designed for WSN simulation, motes[2]-like nodes are the only processing nodes that TOSSIM can simulate.

2.1.2 ATEMU

ATEMU [6] is an emulator of an AVR processor (MICA 2 single chip microcontroller) implemented in C and being utilized in WSNs. ATEMU can emulate not only the communication among the sensors, but also every instruction executed in each sensor (low-level operations of the processor, timers and radio system are all emulated). ATEMU can simulate multiple sensor nodes at the same time, and each sensor node can run different programs. Moreover, it can emulate power consumptions and radio channels. However, although ATEMU can give highly accuracy results (cycle-accurate), the simulation time is much longer than that of similar simulation tools, while it can support only Mica 2 motes.

[2] Mote is a node in a sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network.

2.1.3 AVRORA

Avrora [7] simulates a network of motes, runs the actual microcontroller programs (rather than models of the software), and runs accurate simulations of the devices as well as the radio communication. *Avrora* is an instruction-level simulator built in Java, which removes the gap between *TOSSIM* and *ATEMU*. The codes in *Avrora* run instruction by instruction, which results in a higher simulation speed and better scalability. *Avrora* provides more accuracy than *TOSSIM* while it scales at the same level as TOSSIM. However, *Avrora* practically supports only the *ATMega128L* microcontroller while it can support only some features of the *IEEE 802.15.4* standard compliant radio chip *CC2420*[3] using *AvroraZ* (an *Avrora* extension).

2.1.4 WorldSens

Worldsens [8] is actually an integrated environment for development and rapid prototyping of wireless sensor network applications. The environment itself consists of two simulator tools which can be used either independently or in a cooperation – *WSim* and *WSNet* as illustrated in Figure 2.1.

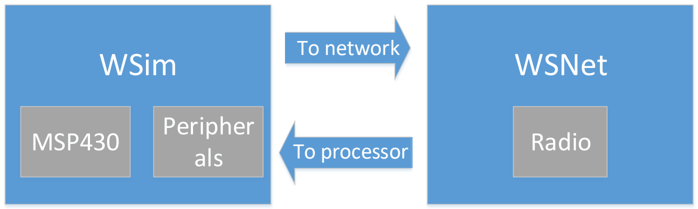


Figure 2.1. WorldSens simulator

The task of **WSim** is to simulate the hardware behavior and the events that occur in the actual hardware platforms. It relies on cycle accurate full platform simulation using microprocessor instruction driven timings. The simulator is able to perform a full simulation of hardware events which allows performance analysis. However, these platforms include only Texas Instrument *MSP430f1611*[4] micro-controller unit including the full instruction set

[3] https://www.ti.com/lit/ds/symlink/cc2420.pdf
[4] https://www.ti.com/lit/ds/symlink/msp430f1611.pdf

as well as all the peripheral digital blocks (timers, basic clock module, serial ports, with UART and SPI modes, etc).

WSNet is a modular event-driven wireless network simulator. Its architecture consists of different blocks that model characteristics and properties of the radio medium. The list of available MAC protocols is also relatively rich containing the *IEEE 802.15.4* wireless protocol. During one simulation, the behavior of a block is specified using a model which is a particular implementation of the block functionalities.

Each node in the **WorldSens** network is simulated by a *WSim*. The *WorldSens* simulator runs the native code as deployed in the sensor hardware without any change and emulates all components embedded in the hardware sensor nodes. Thus, all instructions sending commands to the *CC1100*[5] are executed and the behavior of the *CC1100* is also simulated. When the *CC1100* actually transmits a byte, it is transferred to *WSNet* which simulates the radio propagation and interferences according to its internal models, and finally transmits the data to the simulated *CC1100* RF transceivers in the other *WSim* programs. Finally, it supports simple linear energy consumption model. However, it can simulate only very simple microcontrollers (*MSP430* microcontroller), while the official WorldSens site is not available anymore (and the documentation).

2.1.5 Cooja (Contiki OS)

The **Cooja** [9] simulator is similar to *TOSSIM* since its main purpose is to simulate the behavior of an operating system. *Cooja* is a Java-based simulator developed for simulations of sensor nodes running the Contiki operating system.

The authors of *Cooja* claim that their simulator can work on different levels enabling the so-called *cross level simulations* (Figure 2.2). For example, ns-2 (networking level) is principally a simulator designed for network and application levels without taking the hardware properties into account. On the other hand, *TOSSIM* (operating system level) is

[5] http://www.ti.com/lit/ds/symlink/cc1100.pdf

intended particularly for simulating the behavior of the operating system *TinyOS*, while the purpose of *Avrora* is to simulate at the machine code level (instruction set execution level).

The *Cooja* simulator is implemented in Java, making the simulator easy very extensible, since it also allows the sensor node software to be written in C by using the Java Native Interface. Furthermore, it can execute the Contiki programs in two different ways: either by compiling the application code directly on the host CPU, or by compiling it for the *MSP430* hardware. Moreover, it can simulate *IEEE 802.15.4* wireless protocol, however it cannot provide any power estimations, it can simulate only very simple microcontrollers (*MSP430* microcontroller) and it has low efficiency due to the cross-level simulation extendibility [10].

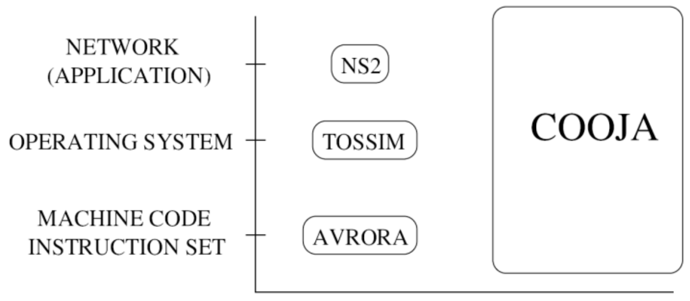


Figure 2.2. Cross-level simulation [132 – D7.1.X]

2.1.6 SunShine (TOSSIM – SimulAVR - GEZEL)

SUNSHINE [11] is the first “sensor-net” simulator that supports joint evaluation and design of sensor hardware and software performance in a networked context. *SUNSHINE* captures the performance of network protocols, software and hardware up to cycle-level accuracy through its seamless integration of three existing sensor-net simulators: a network simulator *TOSSIM*, an instruction-set simulator *SimulAVR* [12] and a hardware simulator *GEZEL* [13] (all simulators are built in C++ programming language).

The **TOSSIM** framework has been described above. Even though TOSSIM is able to capture the sensor motes behaviors and interactions, such as packet transmission, reception and packet losses at a high fidelity, it does not consider the sensor motes processors’ execution time. Therefore, TOSSIM cannot capture the fine-grained timing and interrupt properties of software code.

On the other hand, **SimulAVR** is an instruction-set simulator that supports software simulation for the Atmel AVR family of microcontrollers which are popular choices for processors in sensor systems. *SimulAVR* provides accurate timing of the software execution and can simulate multiple AVR microcontrollers in one simulation.

Finally, **GEZEL** is a hardware domain simulator that includes a simulation kernel and a hardware description language (based on the Finite-State-Machine + Datapath (FSMD) model). In GEZEL, a platform is defined as the combination of a microprocessor connected with one or more other hardware modules. For example, a platform may include a microprocessor, a hardware coprocessor, and a radio chip module. To simulate the operations of such a platform, one has to combine different software modules, which capture software executions over the microprocessor, and hardware simulation domain, which captures the behaviors of hardware modules and their interaction with the microprocessor as illustrated in Figure 2.3.

Cycle-level co-simulation can show details of sensor nodes' behaviors, such as hardware behavior, but are relatively slow to simulate. TOSSIM nodes do not simulate many details of the sensor nodes but are simulated much faster. The mix of cycle-level simulation with event-based simulation ensures that SUNSHINE can leverage the fidelity of cycle-accurate simulation, while still benefiting from the scalability of event-driven simulation.

However GEZEL can support only co-simulations with AVR microcontrollers using *SimulAVR* (e.g. *SunShine*), while the network communication is abstracted (slightly better than TOSSIM).

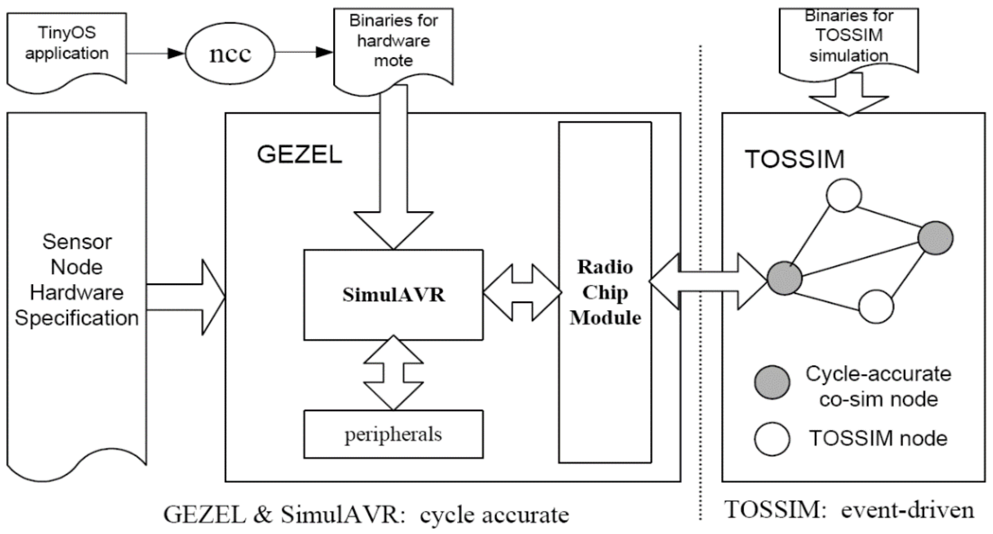

TinyOS application
ncc
Binaries for hardware mote
Binaries for TOSSIM simulation
Sensor Node Hardware Specification
GEZEL
SimulAVR
peripherals
Radio Chip Module
TOSSIM
Cycle-accurate co-sim node
TOSSIM node
GEZEL & SimulAVR: cycle accurate
TOSSIM: event-driven

Figure 2.3. SunShine Architecture

Summarizing from the previous sections, WSN simulators focus on specific micro-controllers or motes (they can model the whole sensor node including sensors and the radio communications chip). These simulators can effectively model both the actual software executed on the processing nodes and the network between the devices, however their applicability cannot be generalized neither in terms of software (e.g. support for different operating systems or software packages), nor in terms of hardware (e.g. support for other more complicated CPUs) or network (e.g. they can only simulate specific wireless protocols). As a result, those tools cannot handle the simulation of general CPS that require the modeling of very diverse processing devices together with a multitude of networks.

Table 2.1 provides a comparative view of the CPS simulation tools that have been analyzed in the previous subsections.

Simulation Tool	Simulation Type	CPU Simulation Support	Network Models	Power Models	OS Support	Application Support
TOSSIM	Discrete Event	Only ATmega128 L (MicaZ mote)	Abstracted as a directed graph (not 802.15.4)	✓ (Power Tossim)	✓ (Only TinyOS)	Only nesC applications
ATEMU	Cycle Accurate	Only ATmega128	Abstracted as a directed graph	Very Simple	✓ (Only TinyOS)	Only nesC applications

		L (Mica2 mote)				
Avrora	Instruction Accurate	Only ATmega128 L (MicaZ & Mica2 motes)	AvroraZ 802.15.4	Very Simple	✗	Only nesC applications
WorldSens	Instruction Accurate	Wsim TI MSP430 microcontroller	WSNet Supports 802.15.4	Linear Energy Model	✗	C applications
Cooja	Instruction Accurate	TI MSP430 microcontroller	Supports real network	✗	✓ (Contiki OS)	Applications compiled for ContikiOS
SunShine	Instruction Accurate / Event-Driven	Microcontroller-type CPUs **AVR** (SimulAVR)/ **ARM** (v5) (SimltARM)	Similar with TOSSIM (with FIFO)	Power Sunshine	Only TinyOS	Only nesC applications
COSSIM/ACSIM	Cycle Accurate / Discrete Event	From simple up to multicore complex ARM & X86 CPUs	Ethernet/ Wireless/ 3G	✓ (McPAT/MiXiM)	✓ (Linux-based)	C/C++/Java applications

Table 2.1. Comparative Analysis of Existing CPS Simulation Tools

2.2 Cloud Simulation tools

In the area of Cloud Simulators, the most powerful and commonly used cloud simulation toolkit is *CloudSim* [14] and its numerous extensions or derivatives which supports the modeling and simulation of Cloud computing architectures and environments for application provisioning. In addition, there are a numerus of simulators (such as *BigHouse* [15], *GreenCloud* [16], *CACTOS* [17]) which focus on specific aspects of the data center that are useful for cloud providers such as, and mainly, resources' provisioning as presented in the next sections.

2.2.1 CloudSim

CloudSim implements generic application provisioning techniques which can be extended with ease and limited effort, and this is the reason many other simulation tools have been implemented as extensions of *CloudSim*. It can support system architecture at the level of physical components, as well as behavior modeling of Cloud computing systems such as data centers, virtual machines, etc. The following subsections present the most widely used of the Cloud simulators that have been developed using the *CloudSim* toolkit. Figure 2.4 depicts the layered architecture of *CloudSim*.

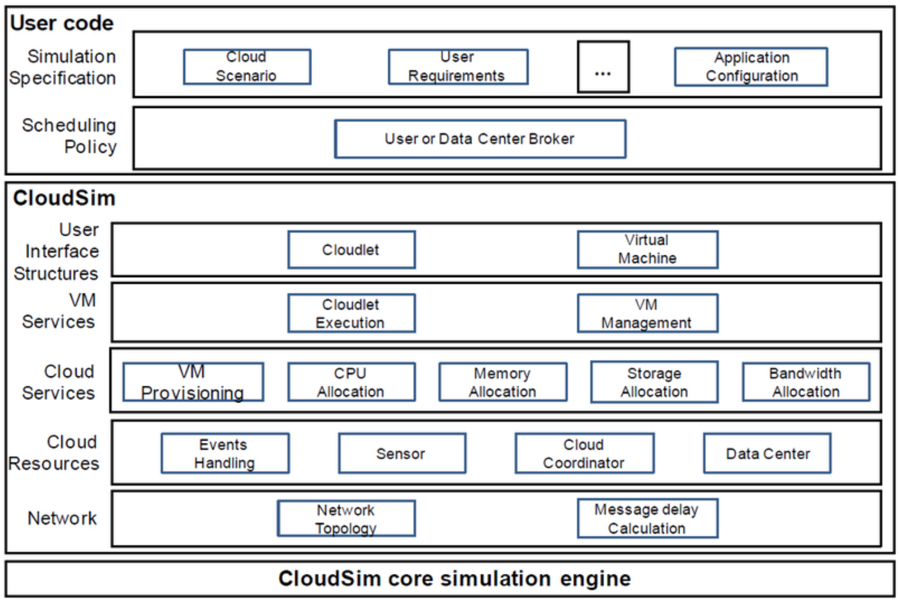


Figure 2.4. CloudSim layered architecture

2.2.1.1 CDOSim

CDOSim is a cloud simulator extending *CloudSim* whose purpose is, as its name indicates, to optimize cloud deployment options. It uses reverse-engineering along with the Instruction Count (IC) measure as input in order to compute response times, Service Level Agreement (SLA) violations, delays and an overall cost which is the sum of the costs for the used bandwidth and virtual machine instances.

2.2.1.2 NetworkCloudSim

NetworkCloudSim [18] is another extension of CloudSim which enables the simulation and modeling of real applications such as cloud data centers and generalized applications such as HPC and e-commerce. It supports scalable network models for cloud data centers and enhances *CloudSim* with complex and more sophisticated application models such as message passing applications and workflows in order to capture the relevant characteristics of real systems.

2.2.1.3 CloudAnalyst

An extension of *CloudSim* which aims to model, analyze and evaluate the requirements of large-scale Cloud applications in terms of geographic distribution of both users and computing infrastructure, is *CloudAnalyst* [19]. Cloud infrastructures can be deployed in different geographic locations and requests to these systems come from distant locations with heterogeneous distributions and sizes.

2.2.1.4 CloudSimDisk

Another energy-aware simulation at the level of storage is *CloudSimDisk* [20]. It extends *CloudSim* with a flexible module which allows energy-aware storage simulations including hard disk drives (HDD) device models, HDD power models, disk array management algorithms and energy-aware data center persistent storage. *CloudSimDisk* is designed to be flexible and it can easily be extended in order to test and validate new algorithms and systems for energy-aware storage in cloud systems. It can model the main characteristics of HDDs like capacity, average rotation latency, average seek time, and maximum internal transfer rate as well as power consumption in a particular operating mode, active or idle.

2.2.1.5 CloudSimSDN

CloudSimSDN [21] is another extension of *CloudSim* for Software-Defined Networking (SDN) enabled cloud environments. It simulates cloud data centers, physical machines, switches, network links and virtual topologies and it can evaluate resource management policies applicable to SDN-enabled cloud data centers. *CloudSimSDN* measures both performance metrics and energy consumption to ensure environment conservation and cost-reduction.

2.2.1.6 SmartSim

SmartSim [22] is a cloud simulation toolkit that has been built on the basis of CloudSim. Its purpose is to model applications on smart mobile devices. It can support both system modeling, which refers to the hardware functionalities such as processor and memory, as well as behavior modeling which is the operational behavior such as resource provisioning mechanisms, dynamic processing management policies, and statistics of resource utilization for computational intensive mobile applications on smart mobile devices.

2.2.1.7 DynamicCloudSim

DynamicCloudSim [23] is an extension of *CloudSim* which introduces the simulation of instabilities coming mainly from the heterogeneity of hardware on top of which VMs are running, such as dynamic changes due to external loads as well as failures in task executions. In order to achieve this goal it takes into account several parameters like: (i) external bandwidth as a requirement of tasks and file I/O as additional performance characteristic of tasks, VMs, and host, so that the execution of different kinds of tasks on VMs with different performances can be simulated, (ii) heterogeneity and randomization among VMs' performance and (iii) dynamic changes at runtime in order to simulate he influence of external loads as a consequence of sharing common resources.

Summarizing, *CloudSim* and its extensions try to model a cloud infrastructure and use application workload models and resource performance models in order to provide information about system and user configurations and requirements. Compared to COSSIM/ACSIM, these simulators are far more generic and do not employ precise simulation of the actual execution of an application neither do they model the exact hardware used but rather generic models of the underlying platforms.

2.2.2 BigHouse – CactoSim - GreenCloud

There are a number of simulators in cloud domain which focus on specific aspects of the data center that are useful for cloud providers such as, and mainly, resources' provisioning. These 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. The following subsections present an overview of the existing such simulators.

2.2.2.1 BigHouse
Bighouse [15] is a simulator infrastructure for data centers which introduces a higher level of abstraction. Its key feature is the usage of a stochastic queueing simulation methodology, by representing the fundamental characteristics of workloads (resource requirements, instructions, memory or disk access, etc.), using random variables. The first module is responsible for the construction of the data centers including some models for the description of the desired architectures which can be extended in order to create new functionality. The second module creates the simulation using the statistical tools, handling the communication and control infrastructure and giving the results of the simulation process.

2.2.2.2 CactoSim
CactoSim simulator [17] is a discrete event simulation framework which enables cost and risk analysis by simulating cloud optimization strategies using a mix of workloads and resource performance evaluating scenarios. It supports both behavior and system modeling of heterogeneous components within a cloud computing environment, taking into account the infrastructure (data centers, clusters, racks, virtual machines, etc.) provisioning policies, providing energy consumption, resource utilization, application response time and costs.

2.2.2.3 GreenCloud
GreenCloud [16] is a simulation environment focusing mostly on energy-aware cloud computing data centers. It models and calculate the energy consumption of each component of a data center such as servers, switches, links, etc, by separating the energy consumption in three types: (i) computing energy, (ii) communication energy, and (iii) the energy consumption related to the physical infrastructure of a data center.

COSSIM/ACSIM on the other hand focuses on specific applications and simulates them with high accuracy. As such, from a cloud provider perspective, it provides greater insight on how to design and implement tiers, evaluate performance and latencies. From a cloud application developer perspective, this increased accuracy and focus on a specific application, provides better knowledge on the impact of using a cloud infrastructure and reveals opportunities and methods to overcome bottlenecks or inherent difficulties.

Therefore, it becomes obvious that COSSIM/ACSIM covers a niche for cloud providers and application developers that target specific use cases rather than general infrastructure.

Table 2.2 provides a comparative view of the most widely used Cloud simulation tools that have been analyzed in the previous subsections.

Simulation Tool	**Underlying Platform**	**Programming Language**	**Cycle-Accurate**	**Network Models**	**Energy Models**	**H/W Accelerators**
CloudSim	SimJava	Java	✗	limited	✓	✗
CloudAnalyst	CloudSim	Java	✗	limited	✓	✗
IcanCloud	CloudSim	C++	✗	✓	✗	✗
NetworkCloudSim	CloudSim	Java	✗	✓	✓	✗
GreenCloud	NS-2	C++	✗	✓	✓	✗
COSSIM/ACSIM	GEM5/OMNET++	C++/python	✓	✓	✓	✓

Table 2.2. Comparative Analysis of the most widely used Cloud Simulation Tools

2.3 HPC Simulation tools

Cloud computing is mainly about sharing resources and making efficient use of computational machines by multiple users and applications. Complimentary to the cloud data servers, while also sharing certain aims and features, are the parallel systems intended to execute highly complex and demanding applications (HPC). The problem in this case is not to design an infrastructure that can be effectively used by multiple users but to make a system of systems that is tuned to execute in the most efficient (performance, energy - or both -wise) way a single (or a few) parallel application(s). The advent of applications that require extreme computational resources (such as deep learning, AI, analytics, real-world physical/chemical/biological simulations, etc.) both for academic/research and commercial purposes makes the design and deployment of such systems and applications a very interesting and expanding area; in HPC there are certain tools such as *SST/GEM5* [24] which model cycle accurate processing unit, while others (e.g. [25], [26]) that focus on the simulation of real networks.

SST/GEM5 is a proposal for a tool which tries to explore key issue about HPC details in terms of performances, energy and reliability. The developers have integrated the gem5 simulator with the Structural Simulation Toolkit (SST) [27], extended it by adding fast-forwarding capabilities, by porting the HPC-oriented Kitten operating system [28], and by adding reliability analysis capabilities. On the other hand, in [25], [26], the authors have developed a simple general-purpose simulator using the OMNeT++/OMNEST simulation framework making assumptions about the applications by using models of computation. As such they either focus on providing high accurate results per node and more simplistic network models or they replace the cycle-accurate simulation with application traces or functional modeling and focus on the network part. COSSIM/ACSIM goes beyond those approached by combining successfully both aspects while providing additional functionality (such as testing the robustness of applications).

Other HPC/Parallel system simulation tools overcome performance and scaling issues by using models of processors, application traces and specific assumptions for the network topologies/technologies and middleware (such as communication libraries like MPI). Example tools belonging in this category are BigSim[29], XSim [30] and MARS [31]. Compared to those tools, COSSIM/ACSIM is more general, being able to support arbitrary network topologies and software tools, while by executing the applications on precise processor models it is far more accurate.

Summarizing, it becomes apparent that no integrated solution exists that can handle the simulation of actual CPS, Cloud and HPC systems, including their complete software stack and network dynamics. In order to address the lack of such tools, COSSIM/ACSIM efficiently integrates several well-established simulation sub-systems in a single framework that works in the transparent to the user way (i.e. as if it was a single tool rather than a framework). COSSIM/ACSIM is the only known tool that can handle the network, processing and power simulation in an integrated and fully distributed manner utilizing custom H/W accelerators allowing for much faster and more accurate development.

Table 2.3 provides a comparative view of the most widely used HPC simulation tools that have been analyzed in the previous subsections.

Simulation Tool	**Cycle-Accurate**	**Network Models**	**Energy Models**	**H/W Accelerators**
SST/gem5	✓	Limited (MPI + external router)	×	×
BigSim	×	Limited (MPI based)	×	×
xSim	×	Limited (MPI based)	×	×
COSSIM/ACSIM	✓	✓	✓	✓

Table 2.3. Comparative Analysis of the most widely used HPC Simulation Tools

3

COSSIM/ACSIM Design Choices

A highly parallel system comprises of a set of nodes and a number of networks that connect those nodes together. The diversity of nodes used in such systems is high; there can be simple micro-controllers that control an actuator device or provide readings from a sensor, network devices, powerful main control units, server systems, etc. Thus, in order to accurately simulate a processing node, a system simulator that is cycle-accurate, Instruction Set Architecture (ISA) independent, configurable (in terms of supported devices and system features), able to boot real-world operating systems and support real network modeling is required. This chapter presents an overview of the existing processing and network simulators as well as the most appropriate solution in order to meet the COSSIM requirements in Sections 3.1 and 3.2 respectively. In addition, Section 3.3 overviews a comparative analysis of existing processing and network simulation tools. Furthermore, Section 3.4 presents the related work about the H/W accelerators in full system simulators. Finally, Section 3.5 describes our solution for the integration of these components examining the most widely-used integration frameworks.

3.1 An Overview of the Processor-Only Simulation Tools

The key concept is not to develop processing and/or network simulators from scratch but to exploit the advantages of the most appropriate existing simulator frameworks. There are numerous processing simulators and emulators that have been developed and implemented during the last decades. This section presents the most widely-used of them focus on the advantages and disadvantages in order to support the CPS, Cloud and HPC domains. In other words, the processing simulation tool must support the following list of requirements/specifications so as to be able to handle efficiently the above domains.

I. Handling multi-core CPUs
II. Being Cycle accurate
III. Able to efficiently simulate a complete system with devices and an operating system (Full System mode)
IV. Integrate a Widely-used user interface (familiar to the community)
V. Should be open-source
VI. The simulation execution time should be high
VII. Integrate/handle power consumption models
VIII. Integrate/handle Hardware Accelerators

More analytically, it must support multi-core CPUs of different ISAs (with ARM and x86 being mandatory due to their wide industry adoption) including several levels of memory hierarchy and complex peripherals. In addition, it must be a cycle accurate simulator because an accurate modeling and simulation environment can reduce the effort and cost of deployment, testing and maintenance as mentioned in [32]. Moreover, since COSSIM/ACSIM aims to model complete Systems, it must support complete processing systems with the I/O devices and at least one operating system, in order to accurately capture realistic interactions of an application running on top of an actual system. Equally important, a tool that can be considered for COSSIM/ACSIM has to be open-source so as to allow modification and adaptations required by the intricacies of the application domain targeted by COSSIM/ACSIM. Lastly it must support state of the art energy models or at least it has to provide hooks so that such tools can be effectively integrated.

3.1.1 Imperas Open Virtual Platforms (OVP)

OVP [33] is a high-performance simulator that can simulate advanced multi-core heterogeneous or homogeneous platforms with complex memory hierarchies, cache systems and peripherals. *OVP* is an instruction-accurate simulator (not cycle accurate) implemented in C. Currently, it can support processor models of ARC, ARM, MIPS, PowerPC, NEC v850, and OpenRisc families with many different types of system components including ram, rom, trap, cache and peripheral models including dma, uart, fifo, etc. However, it does not provide cycle-accurate simulation, power models and cannot model or run an operating system.

3.1.2 SimpleScalar

The *SimpleScalar* [34] tool set is a system software infrastructure used to run modeling applications for program performance analysis, detailed micro-architectural modeling, as well as for hardware-software co-verification. *SimpleScalar* can execute modeling applications that simulate (cycle-accurate) real programs running on a range of modern processors and systems. The toolset can model a variety of platforms ranging from simple unpipelined processors to detailed dynamically scheduled microarchitectures with multiple-level memory hierarchies, including Alpha, Power PC, x86 and ARM. In addition, an architecture-level power modeling framework (Wattch [35]) has been developed in parallel with SimpleScalar to perform power analysis. However, it cannot model or run an operating system, since the simulation speed is low.

3.1.3 CPU Sim

CPU Sim [36] is a Java application that allows users to design simple computer CPUs at the microcode level and to run machine-language or assembly-language programs on those CPUs through simulation. It can be used to simulate a variety of architectures, including accumulator-based, RISC-like, and stack-based (such as the JVM) architectures. However, CPU Sim cannot support very complex CPUs (such as ARM, X86, etc.), while it can run only machine-language or assembly-language programs.

3.1.4 ESCAPE

ESCAPE [37] is a PC-based simulation environment aimed at the support of computer architecture education. The environment can simulate both a microprogrammed architecture and a pipelined architecture with single pipeline. Both architectures are custom-made, with a certain amount of configurability. Other tools, such as a memory monitor, assembler/disassembler and analysis tools, such as on-the-fly generation of pipeline activity and usage diagrams, are integrated with the environment. However, ESCAPE cannot support very complex CPUs (such as ARM, X86, etc.), while it can be used only for education purposes (due to simplicity of supported architecture).

3.1.5 HASE

HASE [38] is a Hierarchical Computer Architecture design and Simulation Environment which allows for the rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software. HASE produces a simulation trace file which can be used to animate the on-screen display of the model so as to show data movements, parameter value updates, state changes, etc. HASE is available mainly for academic purposes and it can support only some MIPS commands and not complex CPUs (such as ARM, X86, etc.).

3.1.6 MikroSim

MikroSim [39] is an educational software computer program for hardware-non-specific explanation of the general functioning and behaviour of a virtual processor, running on the Microsoft Windows operating system. Devices like miniaturized calculators, microcontroller, microprocessors, and computer can be explained on custom-developed instruction code on a register transfer level controlled by sequences of micro instructions (microcode). Based on this it is possible to develop an instruction set to control a virtual application board at higher level of abstraction. **MikroSim** is available mainly for academic purposes and it can support only micro instructions (microcoding) for a virtual control unit and not complex CPUs (such as ARM, X86, etc.).

3.1.7 SimNow

The *AMD SimNow* [40] simulator is an AMD64 technology-compatible x86 platform simulator for AMD's family of processors. It is designed to provide an accurate model of a computer system from the program, OS, and programmer's point of view. It allows fast simulation of an entire computer system, plus standard debugging features such as break-pointing, memory-viewing and single-stepping. The simulator allows such work as BIOS and OS development, memory-parameter tuning, and multi-processor system simulation. However, *SimNow* is not an open-source tool and it can support only AMD-based processors.

3.1.8 Zsim

Zsim [41] is a fast x86-64 simulator. It was originally written to evaluate ZCache (Sanchez and Kozyrakis, MICRO-44, Dec 2010), hence the name, but it has since outgrown its purpose. *Zsim's* main goals are to be fast, simple, and accurate, with a focus on simulating memory hierarchies and large, heterogeneous systems. It is parallel and uses DBT extensively, resulting in speeds of hundreds of millions of instructions/second in a modern multicore host. Unlike conventional simulators, *Zsim* is organized to scale well (almost linearly) with simulated core count. However, *Zsim* cannot support ARM-based architecture.

3.1.9 GEM5

GEM5 simulator [42] is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. It is a cycle-accurate simulator able to model different CPUs/ISAs and system components (full-system and application-only modes are both supported). In addition, it is a widely used processing simulator with active development by contributors from both the academic and the industrial sectors.

The GEM5 simulation infrastructure is the merger of the best aspects of the M5 [43] and GEMS [44] simulators. M5 provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models. GEMS complements these features with a detailed and flexible memory system, including support for multiple cache coherence protocols and interconnect models. Currently, GEM5 supports most commercial ISAs (ARM, x86, ALPHA, MIPS, Power and SPARC), including booting Linux on three of them (ARM, x86 and ALPHA).

More specifically, GEM5's key capabilities are:

- *CPU Model*. The GEM5 simulator currently provides four different CPU models, each of which lie at a unique point in the speed-vs.-accuracy spectrum. *AtomicSimple* is a minimal single IPC CPU model, *TimingSimple* is similar but also simulates the timing of memory references, *InOrder* is a pipelined, in-order CPU, and *O3* is a pipelined, out-of-order CPU model. *GEM5* decouples ISA semantics from its timing CPU models, enabling effective support of multiple ISAs. As a result, through a combination of ISA and CPU model selection (along with proper processor parameters, e.g. size of internal processor core structures, functional units, delays etc), the simulator is able to closely model numerous available CPUs, such as ARM Cortex-M series, ARM Cortex-A series, x86 processors.
- *System Mode*. Each execution-driven CPU model can operate in either of two modes. System-call Emulation (SE) mode avoids the need to model devices or an operating system (OS) by emulating most system-level services. Meanwhile, Full-System (FS)

mode executes both user-level and kernel-level instructions and models a complete system including the OS and I/O devices (it should be noted though that Linux-based OSes are currently only supported, including Android). Specifically, in SE mode, whenever the program executes a system call, GEM5 traps and emulates the call, often by passing it to the host operating system. On the other hand, in FS mode, GEM5 simulates a bare-metal environment suitable for running an OS. This includes support for interrupts, exceptions, privilege levels, I/O devices, etc.

- *Memory System*. The GEM5 simulator includes two different memory system models, *Classic* and *Ruby*. The Classic model (from M5) provides a fast and easily configurable memory system, while the Ruby model (from GEMS) provides a flexible infrastructure capable of accurately simulating a wide variety of cache coherent memory systems. Ruby models inclusive/exclusive cache hierarchies with various replacement policies, coherence protocol implementations, interconnection networks, DMA and memory controllers, various sequencers that initiate memory requests and handle responses.

It should be noted that *GEM5* does not provide network simulation functionality, in order to simulate frameworks of interconnected systems. However, when using FS mode where devices can be added and used from the OS, *GEM5* supports the use of network interface cards that can provide hooks for connection to external network simulator tools. Also, GEM5 does not support natively any power/energy estimations. It is possible though, to link GEM5 with external widely used power estimators, such as *McPAT* [45].

Concerning performance, GEM5 as a cycle-accurate simulator is rather slow. In practice, that means that for certain configurations, GEM5 can be one to two orders of magnitude slower than the actual system it models. Works that attempt to bring the performance of the simulator close enough to the performance of the actual system simulated by trading either accuracy or verbosity level ([46]) have been presented. Other works in progress [47], are focusing on using virtualization to accelerate simulation.

GEM5 is developed in C++, however it uses Python in order to perform all simulation configuration tasks as well as the simulation setup. As a result, using available features of the simulator to configure a system and prepare a simulation run is rather simple and can be accomplished through Python scripts. However, adding new functionality requires extensive lower-level coding in C++ and compiling the whole simulator.

Overall, the following list covers the main advantages of the GEM5 simulator in the context of COSSIM:

i) GEM5 is a cycle-accurate simulator
ii) It can simulate single or multi-core homogeneous or heterogeneous platforms
iii) It can simulate a very broad range of modern CPUs, from microcontrollers to high-performance processors while supporting all major ISAs
iv) It can simulate memory hierarchies, cache systems and peripherals (such as Hardware Accelerators in Full System mode)
v) Simulation configuration and setup can be performed in high-level scripting language (Python)
vi) It supports power modeling through widely used external tools

On the other hand, the only disadvantage is that its performance for complex CPU models can be quite slow. However, it is not a significant disadvantage in the scope of COSSIM/ACSIM since parallelism can be extracted at the process level, since multiple cGEM5 instances can be executed in parallel on the same physical machine or more efficiently in a highly distributed manner.

Among those, *GEM5* only satisfies the largest number of requirements defined for the *COSSIM/ACSIM* processing sub-system. More specifically, while *OVP* presents several key advantages, such as being a fast simulator able to model complex CPUs along with their peripherals, it lacks certain features that are indispensable in the scope of COSSIM. First of all, x86 processors cannot be modeled in *OVP* as required by the COSSIM application domains. Furthermore, it is not a cycle-accurate simulator and cannot execute a full OS stack, as required by COSSIM/ACSIM. Similarly, *SimpleScalar* cannot also execute a full operating system, neither is it possible to extend the devices it models beyond a processor (i.e. it cannot model peripheral devices, such as a network interface card). In addition, *CPU Sim*, *ESCAPE*, *HASE*, *MikroSim* are simulators for academic purposes which support only simple microcontrollers, while *SimNow* and *Zsim* cannot support ARM-based architectures. On the contrary, GEM5 provides both ARM & x86 ISA modeling, cycle accurate simulation, peripheral device modeling and finally OS execution support.

3.2 An Overview of the Network-Only Simulation Tools

GEM5 can provide system simulation up to the level of the Network Interface Card (NIC). It does not support network modeling. Therefore, COSSIM employs a dedicated network simulator that handles all network related modeling above the physical layer of a NIC. This section aims to collect and analyze the requirements of network simulation tool by examining a wide variety of existing network simulation tools. The following subsection describes the main requirements which are needed from the network simulation tool. In other words, the network simulation tool must fulfill a combination of the following requirements/ specifications in order to handle efficiently the CPS, Cloud and HPC domains

I. Support real network protocols
II. Widely-used emulation interface (familiar to the community)
III. Open-source simulation tool
IV. Fast execution time
V. Support power consumption models

Specifically, it must have the same interface with one of the existing widely-used network and/or wireless signal network (WSN) simulation/emulation tools, while it must support complex and heterogeneous real network protocols (e.g. wireless 802.15.41 protocol). Finally, it must be an Open-Source tool to extend it for integration with our simulator, while it must support energy models to measure the consumption of the network modules (radio, physical etc). The following subsections present an overview of the most widely-used existing network simulators.

3.2.1 NS-2

NS-2 [48] is a popular non-specific discrete event real network simulator built in an Object-Oriented extension to the Tool Command Language (TCL) and C++. It provides the most complete support of communication protocol models (such as wireless IEEE 802.11 and 802.15.4 protocols).

However, this simulator has some limitations. Firstly, when compared to other tools, NS-2 has a long learning curve and requires advanced skills to perform meaningful and repeatable simulations. Another drawback of NS-2 is the lack of customization available. Packet formats, energy models, MAC protocols, and the sensing hardware models all differ from those found in most sensors/systems. In addition, since NS-2 is originally targeted to IP networks but not HPC/CPS Systems, there are some limitations when used for such applications. Firstly, NS-2 can simulate the layered protocols but not application behaviors. However, the layered protocols and applications interact and cannot be strictly separated in HPCs/CPSs. So, in this situation, using NS-2 is inappropriate, and it can hardly produce results with adequate accuracy for HPCs/CPSs. Secondly, because NS-2 is designed as a general network simulator, it does not consider some unique characteristics of sensor-based networks. For example, NS-2 cannot simulate problems of the limited available bandwidth and energy resources, typical of the aforementioned applications. Thirdly, NS-2 has scalability issues when HPCs/CPSs applications typically involve a large number of nodes [49], [50]. Finally, increasing the number of nodes simulated in NS-2 results in tracing files that are too large to manage and difficult to parse.

3.2.2 NS-3

NS-3 [51], like NS-2, is an open source discrete-event network simulator. NS-3 is considered as a replacement of NS-2, (not an extension of it) and as a result it is not backwards compatible with NS-2; thus, it cannot directly take advantage of the large base of protocols and models that have been developed for NS-2. However, it is a real network simulator capable of simulating networks such as Wireless Sensor Networks IEEE 802.15.4 and IEEE 802.11. It provides significant improvements in performance, scalability and extensibility compared to

NS-2 simulator. Like its predecessor, NS-3 relies on C++ for the implementation of the simulation models. NS-3 no longer uses Tcl scripts to control the simulation, thus skipping the problems which were introduced by the combination of C++ and Tcl in NS-2. Instead, network simulations in NS-3 can be implemented in pure C++, while parts of the simulation optionally can be realized using Python as well. However, power/energy model is not mentioned in documentation (probably not implemented) while it has very large trace files.

3.2.3 J-sim

J-Sim [52] is an open-source, component-based compositional network simulation environment that is developed entirely in Java. *J-Sim* is a truly platform-neutral, extensible, and reusable environment. *J-Sim* also provides a script interface to allow integration with different script languages such as Perl, Tcl or Python, while in the current release, *J-Sim* is fully integrated with a Java implementation of the Tcl interpreter (with the Tcl/Java extension), called Jacl. So, similar to NS-2, *J-Sim* is a dual-language simulation environment in which classes are written in Java (for NS-2, classes are written in C++) and "glued" together using Tcl/Java. It supports energy modeling and has a component-based architecture, but it does not support radio energy consumption. However, only the IEEE 802.11 MAC protocol has been implemented so far in *J-Sim*, while the *J-Sim* model defines the generic structure of a node (either an end host or a router) without any mention for the type of CPU which can be simulated. Hence, it is not a preferred simulator tool for realistic WSN simulation. Moreover, it introduces some additional overhead and some inefficiencies regarding to the Java programming language [53].

3.2.4 NETSim

NetSim [54] is a popular network simulation and network emulation tool used for network design & planning, defense applications and network R&D. Various technologies such as Cognitive Radio, Wireless Sensor Networks, Wireless LAN, Wi-Max, TCP, IP, etc. are covered in *NetSim*. *NetSim* is a stochastic discrete event simulator developed by Tetcos, in association with Indian Institute of Science, with the first release in June 2002. However, *NetSim* is an application that simulates only Cisco Systems' networking hardware and software and is designed to aid the user in learning the Cisco IOS command structure.

3.2.4 OMNET++

OMNeT++ [55] is also a discrete event simulator; however it is more general than the aforementioned simulators as it is not designed only for network simulations, thus providing great extensibility. OMNeT++ is a general discrete event, component-based (modular) open architecture simulation framework that includes the basic machinery and tools to write network simulations. Although it does not provide any components specifically for computer networks, queuing networks or any other domain, it offers “1-click” extensions to various simulation models and frameworks such as the INET framework, Castalia or and MiXim [56] for accurate modeling. Model frameworks are developed and maintained completely independently of the simulation framework and follow their own release cycles.

Through OMNET++ different network protocols and topologies can be supported and a realistic network behavior of the parallel system can be modeled; devices such as bridges, switches, routers that are part of the infrastructure rather than the CPS/HPC/Cloud system developed can also be modeled so as to further increase the simulation accuracy.

The key advantage of OMNeT++ is that it offers great extensibility not only in the classical network simulation domain but also in the physical/environment domain [53] (i.e node mobility in 3D space). Moreover, its modular and extensible architecture allows it to be seamlessly integrated to a framework such as COSSIM/ACSIM, without compromising future updates and/or backward compatibility. Furthermore, it features a much friendlier graphical user interface compared to its alternatives, which makes tracing and debugging easier.

Among the pure network simulation tools available, OMNET++ only satisfies the largest number of CPS/Cloud/HPC requirements as defined in [57]. More specifically, *NS-2* & *NS-3* suffers from low performance as the number of nodes increases (bad scalability) due to large trace files while *NS-2* has very simple energy model (NS-3 power/energy model is not mentioned). In addition, *Jsim* has low performance due to Java implementation, while *NETSim* is a commercial network simulation tool.

3.3 Comparative Analysis Table of Existing Processing & Network Simulation Tools

Tables 3.1 and 3.2 provide a comparative view of the processing and network simulation tools that have been analyzed in the previous subsections. Tables demonstrate a fragmented market that does not provide any single integrated solution. As a result, COSSIM/ACSIM is the first known simulation framework that allows for the simulation of a complete modern CPS/HPC utilizing complex SoCs interconnected with sophisticated networks combining the features of the widely used GEM5/OMNET++ processing & network simulators. Finally, the COSSIM system support accurate power estimations (using McPAT & INET/MiXiM power simulators) as illustrated in the next chapter.

Simulation Tool	**Type of Tool**	**Simulation Type**	**CPU Simulation Support**	**Network Models**	**Power Models**	**Scalability**
Imperas OVP	Processor	Instruction Accurate	From simple up to multicore complex CPU	Not Mentioned	Not Mentioned	Good
SimpleScalar	Processor	Cycle Accurate	From simple up to multicore complex CPU	Not Mentioned	Wattch	Significant Overhead
CPU Sim	Processor	Instruction Accurate	Simple custom virtual CPUS	Not Mentioned	Not Mentioned	Not Mentioned
ESCAPE	Processor	Instruction Accurate	Simple custom CPUS	Not Mentioned	Not Mentioned	Not Mentioned
HASE	Processor	Instruction Accurate	Simple custom CPUS / MIPS	Not Mentioned	Not Mentioned	Not Mentioned
MikroSim	Processor	Register Transfer Level	Simple microcontrollers	Not Mentioned	Not Mentioned	Not Mentioned

SimNow	Processor	Cycle Accurate	Complex AMD CPUs	Yes/ Network adapter	Yes	Good
Zsim	Processor	Instruction Accurate	Complex x86 CPUs	Not Mentioned	Not Mentioned	Good
GEM5	Processor / System	Cycle Accurate	From simple up to multicore complex CPU	No (includes only NIC)	Through external power estimation tools	Significant Overhead
NS-2	Network	Discrete Event	Not Mentioned	Support 802.15.4	Very Simple	Not (inappro-priate for WSNs)
NS-3	Network	Discrete Event	Not Mentioned	Support 802.15.4	Not Mentioned	Good
Jsim	Network	Discrete Event	Not Mentioned	Only 802.11	Yes (without radio)	Inappropri ate for WSNs
NETSim	Network	Discrete Event	Not Mentioned	802.11/802.15. 4	Yes	Good
OMNET++	Network	Discrete Event	Not Mentioned	**MiXiM** Support 802.15.4	Linear Energy Model **(MiXiM)**	Good
COSSIM/ACSI M	Processor & Network	Cycle Accurate / Discrete Event	From simple up to multicore complex ARM & X86 CPUs	Ethernet/ Wireless/ 3G	Yes (**McPAT/INET -MiXiM**)	Good

Table 3.1. Comparative Analysis of Existing Simulation Tools (Part A)

Simulation Tool	**Tool Adoption**	**Programming Language**	**Source Code Availability**	**OS Support**	**Application Support**
Imperas OVP	Widely Used	C	Open Source	No	C applications
SimpleScalar	Widely Used	C	Open Source	No	Precompiled (no OS dependencies)
CPU Sim	Educational tool	Java	Open Source	No	Machine-Language
ESCAPE	Educational tool	Pascal	Open Source	No	Machine-Language

HASE	Educational tool	Java	Open Source	No	Machine-Language
MikroSim	Educational tool	Not Mentioned	Open Source	No	Machine-Language
SimNow	Widely Used	Not Mentioned	Commercial	Yes	Precompiled (no OS dependencies)
Zsim	Limited	C++	Open Source	No	C/C++/Java
GEM5	Widely Used	Python/C++	Open Source	Yes (Linux-based)	Precompiled linux binaries (SE mode) / All applications supported by OS (FS mode)
NS-2	Widely Used	TCL/C++	Open Source	No	C/C++ applications
NS-3	Widely Used	Python/C++	Open Source	No	C/C++ applications
Jsim	Limited	Java/ Jacl[6]	Open Source	No	C/C++ applications
NETSim	Widely Used	C#	Commercial	No	C/C++ applications
OMNET++	Widely Used	C++	Open Source	No	C/C++ applications
COSSIM/ACSIM	-	C++/Python	Open Source	Yes (Linux-based)	C/C++/Java applications - SystemC H/W Accelerators

Table 3.2. Comparative Analysis of Existing Simulation Tools (Part B)

3.4 H/W accelerators in Full System simulators

On top of that, none of all the listed simulators has any notion or provide any hooks for the co-simulation of hardware accelerators. However, there are also certain tools that can simulate an accelerator together with a multi-core CPU. First of all, authors in [58] present a tool which 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

[6] Tcl Interpreter with TCL/Java extension

Aladdin [59], 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 HPC/CPS/Cloud system design; moreover, they cannot support any kind of networking.

Two approaches which are, in a way, similar to the *ACSIM* which presented in this thesis can be found in [60] and [61]; the first work presents a synchronization scheme for connecting GEM5 with a SystemC simulator. The main drawback is that, in their approach, they replace the GEM5 *simulate* function with a SystemC *simulate* function thus GEM5 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 very fast processing simulation speeds as well as the full features provided by GEM5. In the second work, authors present the *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, in order to use the accelerators, the users must transform their applications in a data flow language that provides the chaining information of the accelerators, thus making their system much more difficult to be used than *ACSIM* which does not require any kind of transformation. Moreover, they use *AutoPilot* tool and they synthesize the accelerator, in order to get accurate timing information, and as a result their overall system is much slower and less general than our approach.

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

3.5 Integration Frameworks

Binding the aforementioned processing and network simulators together is a very complex task, requiring carefully designed communication interfaces and synchronization schemes.

These bidirectional interfaces have to pass information on the type and timing of events as well as to provide a common data representation scheme throughout the framework.

During the last decades, a number of standards for distributed simulation have been developed. Functional Mock-Up Interface (FMI) [62] is a tool-independent standard for the exchange of dynamic models and for co-simulation. The primary goal is to support the exchange of simulation models between suppliers and OEMs while the second goal was the seamless co-simulation using communication technologies like COM/DCOM, CORBA, Windows message based interfaces or signals and events. However, FMI does not currently support any communication protocol for distributed/parallel simulation. Therefore, FMI cannot be considered an appropriate standard for COSSIM's processing and network integration. Moreover, Test and Training Enabling Architecture (TENA) [63] is an architecture proposed so as to bring interoperability to United States Department of Defense test and training systems. TENA is designed to promote integrated testing and simulation-based acquisition through the use of a real-time synthetic environment, which integrates testing, training and simulation. However, TENA does not provide integrated Time Management which is a necessary feature for simulation events because its aim is to integrate real-time systems. Moreover, Distributed Interactive Simulation (DIS) [64] is a (mainly US) government/industry initiative to define an interoperability infrastructure for linking simulations of various types at multiple locations so as to create realistic, virtual worlds for the simulation of highly interactive activities. Furthermore, the ALSP [65] concept was initiated in January 1990 when ARPA sponsored MITRE to analyze the distributed wargaming process with the goal of generalizing and systematizing the design of constructive simulation interfaces. However, DIS & ALSP protocols are currently being replaced by the IEEE High Level Architecture (HLA) [66] standard which unifies their characteristics and mechanisms [67].

As a result, in order to efficiently integrate the Processing and the Network sub-simulators, COSSIM/ACSIM employs the HLA standard. HLA determines the functional entities, design rules and interfaces for each connected simulation system and specifies the communication between the individual components. It requires a certain Run-Time

Infrastructure (RTI) which performs tasks such as synchronization and simulation control as presented in Chapter 5 in detail.

The HLA specifies a software architecture and not an implementation. Although, there exist several commercial as well as open-source RTI implementations [68], the most widely used RTI implementations are Pitch *pRTI* (commercial), *CERTI* and *Portico Project* (non-commercial). First of all, Pitch *pRTI* integrates simulations in an HLA compliant way; the designer can mix different operating systems and programming languages while some of the main advantages of Pitch pRTI™ are [69] that (i) it can provide advanced debugging capabilities, (ii) it can handle unreliable federates gracefully including automatic resign, ownership, time management recovery and (iii) It gives the ability to integrate existing C/C++ simulators with platform-independent Java systems. However, it is a commercial tool and it cannot be used and incorporated with the COSSIM/ACSIM simulation tool because it is a closed-source tool (free license only allows for up to 2 federates/instances).

Subsequently, Portico [70] has been examined as a possible solution but unfortunately it had a bug in the time management services which was discovered during implementation [71]. Looking at CERTI [72], the implementation of the RTI has been used extensively in previous works concerning with simulation integration [71] [73] [74] [75] because it is an open-source, widely used and stable implementation, so CERTI has adopted as the most appropriate HLA implementation for processing and networking integration. Recently, authors in [74] used CERTI in combination with Ptolemy II, an environment for modeling and simulating physical processes of CPSs. As a result, COSSIM Simulation tool can be extended to support Ptolemy II through CERTI HLA so as to become an integrated CPS simulation tool.

4

COSSIM/ACSIM Architecture

This chapter presents the integration of processing and network simulator parts as well as the structural elements which constitute the COSSIM/ACSIM. Initially, a top-level view of the whole COSSIM/ACSIM simulator integration is described in Section 4.1. Subsequently, modifications/adaptations of processing and network parts to achieve integration of them and support network protocols are presented in Section 4.2 and 4.3 respectively. In addition, Section 4.4 presents the pause-resume functionality, while General Purpose Input Output (GPIO) which has been implemented in GEM5 side to include sensor devices in the processing system is presented in Section 4.5. Furthermore, Section 4.6 describes the integration of power estimation tools for the COSSIM/ACSIM framework in order to estimate the energy and power consumption of processing subsystem. Finally, a sophisticated Eclipse-based GUI which has been developed to provide easy simulation set-up, execution and visualization of results is presented in Section 4.7.

4.1 The High-level architecture of the COSSIM simulator

Figure 4.1 demonstrates the COSSIM/ACSIM simulator with all its components and interfaces (components which are implemented/modified in the context of this thesis illustrated with red color). Multiple instances of a node simulator module (i.e. a GEM5-based module 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). Additionally, each cGEM5 instance is connected through a custom XML interface with a McPAT instance to estimate the energy/power consumption of each node (interface #2), while cOMNET++ employs internally the MiXIM add-on to estimate the power consumption of the network. Both cGEM5 and cOMNET++ instances have been properly modified to provide hooks that allow security software components to interact with the simulation process so that security tests can be performed during the simulation (interfaces #3, #4, #5).

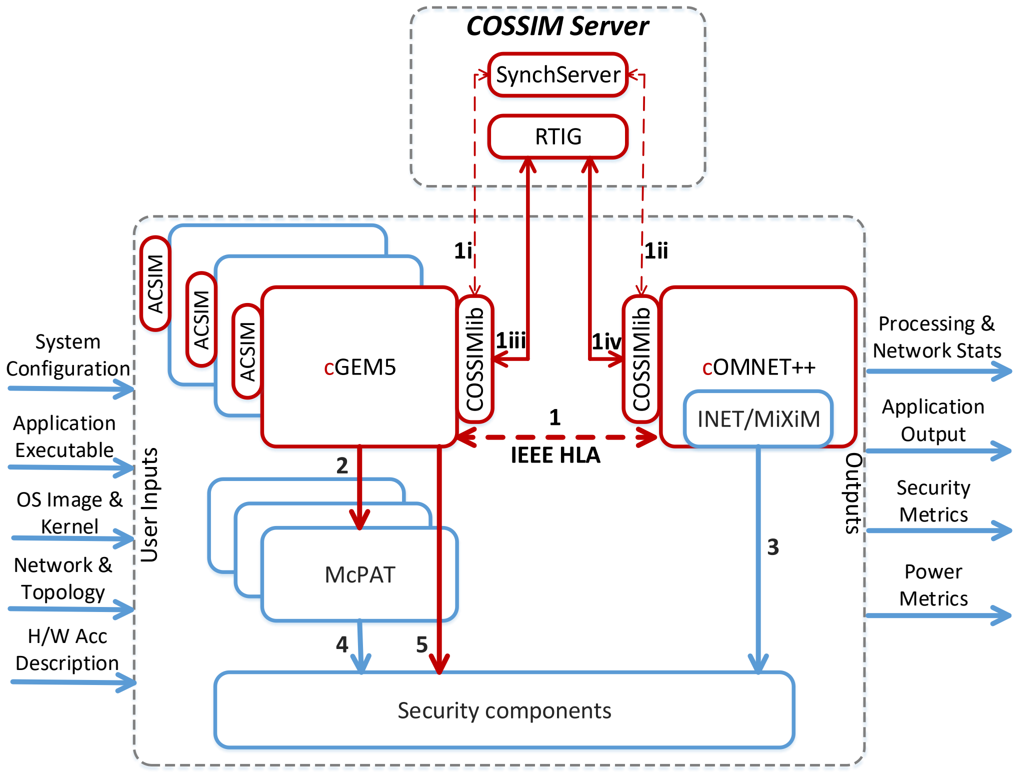


Figure 4.1. Top-level view of the COSSIM framework

Figure 4.1 also illustrates the primary inputs that a user has to provide to the overall system and the primary outputs of the simulation process. Specifically, System Configuration (number of CPUs, memory sub-system, peripherals etc.) have to be defined either using a configuration file or the supplied graphical interface. Furthermore, image files of the OSes that will be executed on the nodes of the simulated platform have to be provided by the user. An image file includes the OS kernel, the application executable[7] and all the libraries that are required. In addition, through a Network & Topology description, the user can define a network topology (distance between nodes, topology, mobility between nodes, etc.) and describe the interconnection channels and the selected network protocols. Finally, the user may define optionally the Hardware Accelerator description in SystemC language.

Upon completion of a simulation, the COSSIM simulator provides a number of outputs. Specifically, the COSSIM output comprises of: a) Processing & Network statistics containing all the measured metrics of the simulation process (e.g. clock ticks, real time execution, cache misses, number of packets sent, number of packet drops, delay of received packets, peak throughput, etc.), b) the output of the actual application that is executed, c) the security metrics (e.g. increase in response time, rejection rate, vulnerability density, system robustness, etc.), d) power and energy estimations for each node (including further details such as peak power, runtime dynamic power, leakage etc) as well as for the network.

Interface #1 is responsible for the integration of the processing and the network simulation instances through HLA. The HLA compliant interfaces that have been added to the GEM5 and OMNET++ simulators implement both the required interconnection and synchronization functions. Interface #1 consists of four sub-parts per node; (i) two responsible for the exchange of the Data Packets and the synchronization of the Processing with the Network sub-tools and (ii) two responsible for the initialization and synchronization of the overall COSSIM framework through custom-made *SynchServer*. Finally, *COSSIMlib* is implemented so as to enable the interoperability between cGEM5/cOMNET++ and CERTI/HLA as described in detailed in the next Chapter.

[7] In the context of this thesis a number of preassembled linux-based OS images is implemented in which the user can execute C/C++ and Java applications.

On top of that, this thesis presents a novel interconnection of processing full system simulator with SystemC cycle-accurate hardware accelerator device providing both S/W and H/W description in the same simulation environment (i.e. ACSIM). SystemC is selected because of its cycle-accurate simulation features, while it is one of the most widely used input languages for the HLS tools. In this work, the official effort for SystemC definition and promotion known as Open SystemC Initiative (OSCI), now known as Accellera [76], provides an open-source proof-of-concept simulator while it has been approved by the IEEE Standards Association [77].

4.2 GEM5 adaptation for COSSIM integration (cGEM5)

In order for GEM5[8] to serve as the processing simulation sub-system in the COSSIM framework, it has properly been extended to support certain additional features. Specifically, in order to be able to simulate a processing node, it should support the simulation of a CPU including several levels of memory hierarchy and complex peripherals (such as network cards, accelerators or other components), as well as the execution of a full operating system on top of the simulated device. Currently, cGEM5 can support single and multi-core ARM and X86 architectures, real network cards and a complete TCP/IP protocol stack included in a Linux-based OS Kernel module with the appropriate drivers. Additionally, cGEM5 can also execute safety-critical systems (such as RTOS) [78].

In the following subsections the limitations of the current, publicly available, version of GEM5 are described in tandem with the modifications and extensions that have been implemented to alleviate those restrictions.

4.2.1 Extending the Network Model of GEM5

In GEM5's publicly available repositories, the only network interface card implemented, tested and verified is the Intel 8254x based gigabit Ethernet adapter. It is provided as a PCI GEM5 network device using the e1000 Linux driver.

However, the latest version of GEM5 supports this *real-network* device only on ARM-based architectures [79]. In the context of this thesis, GEM5 has been recently modified so as

[8] The adapted version GEM5 for COSSIM will be referred as cGEM5

to support the Intel 8254x network card for the x86 ISA as described in the following code-segment. Specifically, line 1 instantiates the *Intel 8254x* device as a normal PCI device to x86 PCI BUS, lines 3-5 connect the network device to the GEM5 memory system using 2 master ports (for configuration) and 1 slave port (for data transfer), while line 7 connects the Intel's 8254x interface with the Etherlink[9] interface.

FSConfig Python File. Connect the Ethernet NIC to X86 Classic System

```
1   self.ethernet=IGbE_e1000(pci_bus=0,pci_dev=0,pci_func=0,InterruptLine=1,InterruptPin=1)
2   ...
3   x86_sys.ethernet.pio    = x86_sys.iobus.master
4   x86_sys.ethernet.config = x86_sys.iobus.master
5   x86_sys.ethernet.dma    = x86_sys.iobus.slave
6   ...
7   self.etherlink.interface = Parent.testsys.ethernet.interface
8   ...
```

On top of that, proper drivers to support it have been built; it should be noted though that the available drivers required Linux kernel 3.x support and therefore such a kernel had to be custom built since the GEM5 repositories only offered Linux kernel 2.x for x86 simulated systems. Our solution is shared in GEM5's community [80].

In addition to the network interface cards, GEM5 supports networking through a simple Etherlink device. Etherlink is a virtual dummy link which emulates a cable over which Ethernet packets are sent and received without any delay (no switching or routing functionality is implemented – Figure 4.2).

In the scope of COSSIM, these limitations are unacceptably restrictive. Therefore, Etherlink could not be used in its current form and thus it had been modified, while NICs supporting more protocols have been developed. Since NIC device models cannot easily be developed without specific information from their manufacturer -the Intel NIC model used in GEM5 has been contributed by Intel itself- and in order to support different physical networks, the COSSIM novel approach is to tap the Ethernet packets from Etherlink and send

[9] Etherlink is a virtual dummy link which connects two homogeneous GEM5 systems.

them to the Networking Simulator modifying at the same time the packets to match the specific network protocol required by the simulated application (i.e. Ethernet, WiFi, 3G, etc).

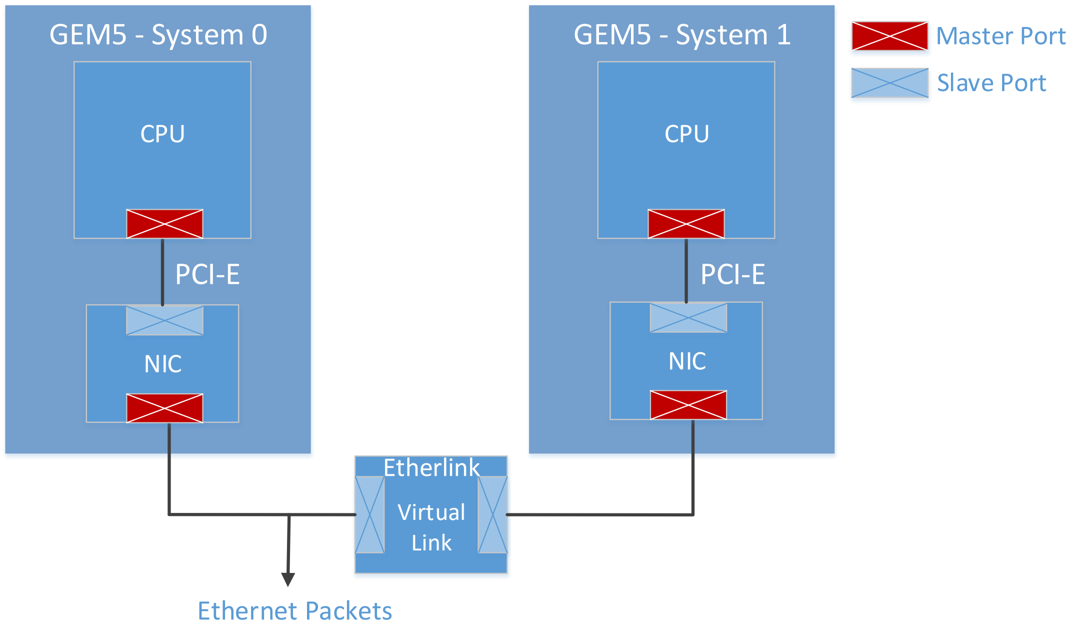


Figure 4.2. GEM5 Systems Interconnection using Etherlink

In order to achieve the aforementioned objectives, the CERTI HLA interface [72] has been employed. Specifically, the COSSIMlib has been implemented and integrated to the main core of the GEM5 system through Etherlink. COSSIMlib is a wrapper to an RTI Ambassador Class which is responsible for the exchange of the messages over the network with the HLA Server via TCP and UDP sockets. COSSIMlib exchanges Ethernet Packets captured from the Etherlink Device and sends (and accordingly receives) them to (from) the HLA Server. The HLA Server forwards these messages to a proper interface of the adopted Network Simulator that implements all the network related functionality. Figure 4.3 provides an overview of the implemented scheme.

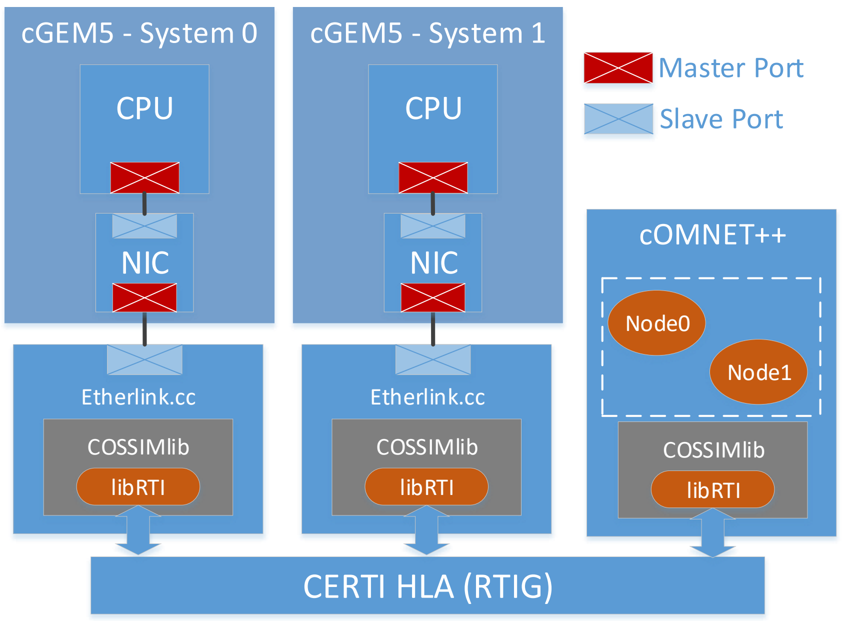


Figure 4.3. COSSIM Systems Interconnection

4.2.2 Supporting Parallel/Distributed Simulation

The simplistic network model of GEM5 has another serious limitation. It only supports the simulation of two identical networked systems (for example two identically configured ARM processors with exactly the same peripherals and memory configuration). Furthermore, the simulation of both systems is executed within the same thread, thus a serious performance penalty is triggered while no synchronization primitives between the two systems are provided.

By using HLA-complaint cGEM5 interfaces combined with a network simulator, as described in the previous subsection, the different cGEM5 instances can be efficiently connected. Each cGEM5 instance models a single node and different GEM5 instances are connected through a simulated network (more precisely through HLA links and a network simulator). As a result, all the following limitations of a conventional GEM5 simulation can be overcome:

- ✓ there are no limitations for identically configured systems.
- ✓ there are no limitations on the number of cGEM5 instances that are interconnected together; the overall system can be scaled to support as many processing nodes as required.

- ✓ parallelism can be extracted at the process level, since multiple cGEM5 instances can be executed in parallel. Furthermore, as CERTI HLA functions over IP, the different cGEM5 instances do not even need to be on the same physical machine and the overall COSSIM simulator can thus be executed efficiently in a highly distributed manner.

The parallel nature of our approach requires a novel mechanism so as to synchronize the different cGEM5 instances as well as the network simulator; the need for synchronization arises from the different notion of time in each GEM5 instance. GEM5 is an event-driven simulator that schedules operations (events) on clock ticks. As a result, different GEM5 instances modeling different systems require different amounts of wall-clock time to reach certain simulation milestones and the notion of the actual time in those GEM5 instances can be different at any given wall-clock period. In addition to the limited accuracy the lack of synchronization triggers, and in our case where the processing nodes are connected through a network, the communication on top of the actual network protocol cannot be performed at all, as ordering or other time-related functions cannot be accomplished. Therefore, COSSIM introduces a novel synchronization system as described in the next Chapter.

Alleviating the above limitations, cGEM5 is able to serve as the processing simulation sub-system in the COSSIM framework. In addition, we extend the publicity available GEM5 compatible images to facilitate the application import in COSSIM simulator, while a X86 GEM5-compatible image with full Ubuntu 12.04 is created from scratch. Specifically, our version of cGEM5 can support:

- ➢ 2 lightweight Linux distributions
  - ➢ Gentoo Base System (v1.12.11.1) for X86 processors (x86_64root.img)
  - ➢ BusyBox (v1.15.3) for ARM processors (linux-aarch32-ael.img & linaro-minimal-aarch64.img)
  - ➢ In both systems Ubuntu-minimal package and JRE7 are installed so as to enable execution of C, C++ and Java applications

- Ubuntu 12.04 Linux distribution are integrated for both X86 (ubuntu-12.04.img) & ARM (aarch32-ubuntu-natty-headless.img & aarch64-ubuntu-trusty-headless.img)
  - Ubuntu-minimal & Ubuntu-essential packages are installed
  - Apt-get is enabled for easy installation

4.3 OMNET++ adaptation for COSSIM integration (cOMNET++)

This section describes the OMNeT++ modifications/adaptations to support completed Ethernet Wifi and 3G network protocols. The key idea behind using a dedicated network simulator in COSSIM is to be able to support multiple network protocols, topologies and devices through which nodes (represented by cGEM5 instances) can be interconnected. OMNeT++ has been chosen as the most capable and feature-rich network simulator in that context, however there are issues that arise. OMNeT++ does not support the real protocol stacks (e.g. Linux ones) as GEM5 and therefore to be able to bridge the two packages and use freely all OMNET++ legacy requires a procedure that encapsulates/decapsulates cGEM5 binary packets into OMNET++ - compatible packets. The latter can then be used throughout OMNET++ and ensure 100% compatibility with all OMNeT++ packages and structures. In addition, in order to support different network protocols (generally compatible till a certain level, such as WiFi protocols) within OMNeT++, a micro-router functionality is implemented as described in the next paragraphs.

4.3.1 HLA Enabled Node Functionality

cGEM5 instances are connected to OMNeT++ through HLA. For each cGEM5 instance, a node is created within OMNeT++, called HLA-Enabled node. These nodes can transparently communicate with cGEM5 through an HLA run-time infrastructure (RTI) wrapper, while encapsulating/decapsulating network packets in a comprehensible form for both simulators. Within OMNeT++, the HLA-Enabled nodes can communicate with any normal OMNeT++ node, e.g. a router or a switch, and thus any kind of topology can be supported. Furthermore, it is even possible to have in the same simulation OMNeT++ nodes that model user-defined systems, if the user of COSSIM is not interested in simulating them in cGEM5.

More specifically a CERTI-HLA compliant wrapper was developed within OMNET++, offering a unique interface to each node simulated in the network subsystem in order to communicate consistently and synchronized with the processing subsystem of the COSSIM simulator. Figure 4.4 presents an overview of the OMNeT++ top level architecture augmented with two HLA-enabled nodes. These nodes communicate (via the HLA sockets) transparently with the processing simulator. All the added functionality was deployed in the user space to assure 100% compatibility with OMNeT++ and its libraries (e.g. INET). As mentioned earlier in order to increase COSSIM's simulation accuracy the upper protocol stack of the network should be simulated in the processing sub-system (that is cGEM5). On the other hand, the network sub-system should be able to forward network packets in the data link layer (L2) or in the network layer (L3) in order for other aspects of the network to be modeled (e.g packet latency). Furthermore OMNeT++, like any other network simulator (e.g ns2), does not produce RFC-compliant packets, meaning that there is no actual payload data (from the application layer for simplicity and performance purposes) and actually, only the payload length field in handled.

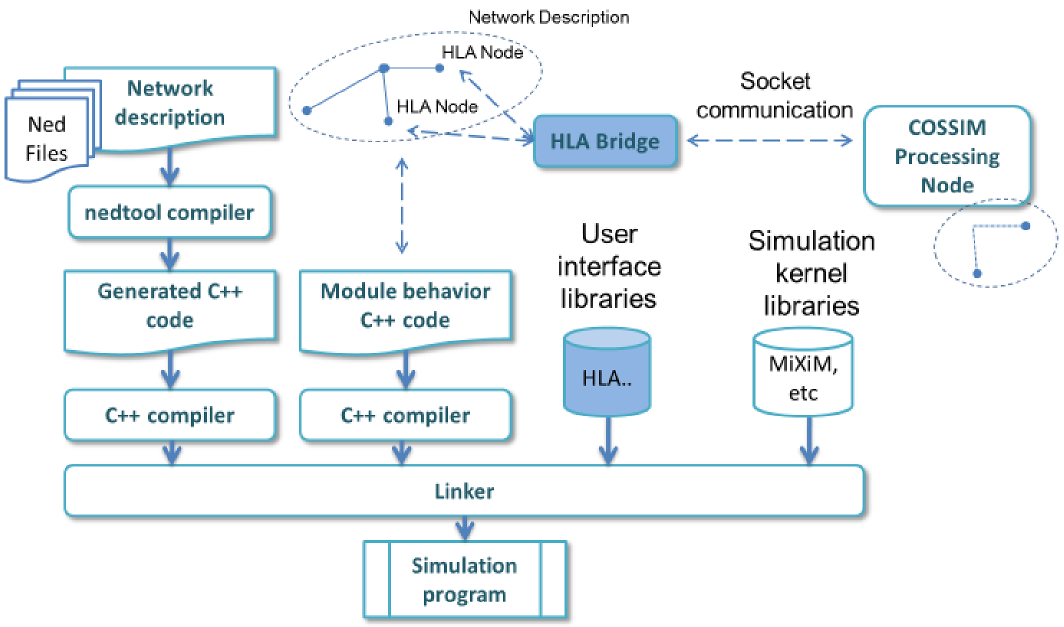


Figure 4.4. COSSIM network subsystem (two HLA-enabled nodes are illustrated)

On the other hand, and since the goal of COSSIM is to provide cycle-accurate simulation, the whole Linux protocol stack is executed within cGEM5, thus producing a fully RFC-compliant binary IP packet. In order for these packets to be forwarded, they have to be properly encapsulated inside the cOMNeT++ L2/L3 packet structure. The first step towards this direction was to modify the standard OMNeT++ cPacket structure to include payload data. The second step was to develop a custom-fit functionality that will be automatically inherited in each of the HLA enabled nodes of the simulation to seamlessly convert the cGEM5 packets to cOMNeT++ packets and vice versa. Each packet sent from the cGEM5 subsystem into the OMNeT++ passes through a sequential de-capsulation procedure (i.e parsing) from the Ethernet/L2 level to the L3 level (the IP level) followed by an encapsulation procedure in the OMNeT++ protocol stack. In the opposite direction (from OMNeT++ to cGEM5) all the L2/L3 fields had to be properly de-serialized into a single RFC-compliant binary packet (i.e. a valid Ethernet packet) that will form the payload for the HLA channel. Figure 4.5 summarizes this procedure. All this functionality is abstracted from the user as it is built as a shared object that can be automatically linked in any simulation scenario that demands precise processing simulation.

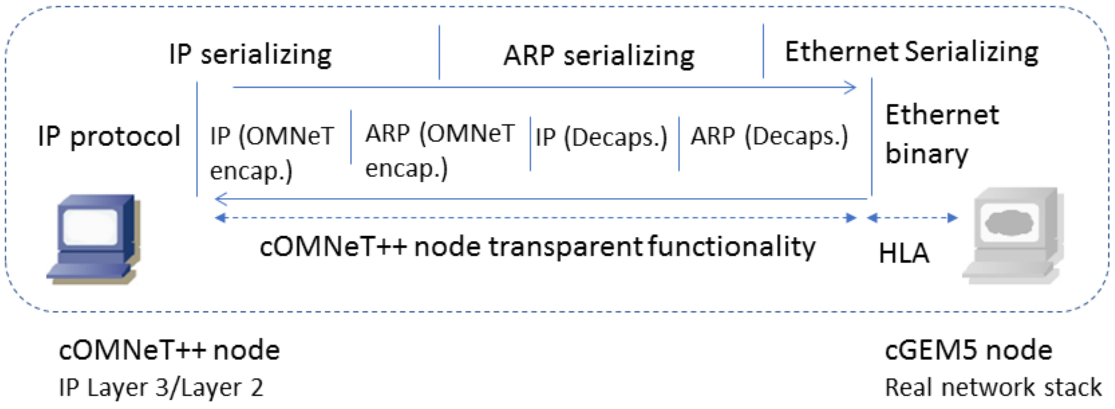


Figure 4.5. The En/De-capsulation vs serialization process inside OMNET++ HLA nodes

In order for the user to define HLA enabled nodes, he/she should only include the HLA enabled node in the OMNeT++ NED description file which describes the network topology. As the HLA Nodes are built as shared object they are able to be linked automatically within any COSSIM simulation project. If the user wishes to add custom functionality, he/she can inherit the basic functionality and extend as desired or directly edit the reference code. In the following snippet it is shown how an HLA enabled node can be included in a network topology. Figure 4.6 illustrates the two HLA-enabled nodes (which are defined from the following code snippet) and two simples original OMNeT++ nodes (Client & Server).

NED file network topology. Two HLA Enabled Nodes instantiated into a simulation network

```
import tic_toc.Txc0;
import tic_toc.Txc1;
import tic_toc.SyncNode;
network ARPTest
{ . . .
submodules:
        tic: Txc0 { @display("i=device/pc");}
        toc: Txc1 { @display("i=device/pc");}
        switch: EtherSwitch { @display("p=202,156"); }
. . .
connections:
        tic.gate <--> ethline_slow <--> switch.ethg++;
        toc.gate <--> ethline_slow <--> switch.ethg++;
. . .}
```

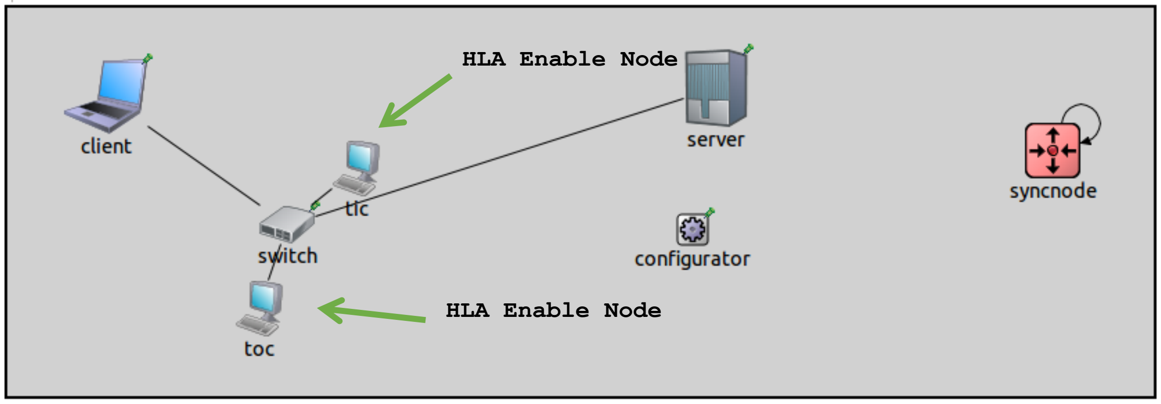


Figure 4.6. A simple Network simulation using two HLA Enabled Nodes

4.3.2 Transparent micro-Routers Functionality

Currently, there are no other Network Interface Cards available for GEM5 besides Ethernet ones. To be able to support different network protocols (generally compatible till a certain level, such as WiFi protocols) within OMNeT++ a micro-router functionality is implemented within OMNeT++ allowing user to change the physical medium (and as a result the network adapter) from Ethernet to any other supported option without affecting at all the cGEM5 node hook point implementation. In this way, the GEM5 nodes could always "sees" an "Ethernet attached" network on the OMNeT++ side.

COSSIM transparent micro-routers from the one side always have an Ethernet interface 100% compatible with the Linux protocol stack talking to the GEM5 node counterpart and from the other side an interchangeable option for the user to switch the network interface.

As shown in the Figure 4.7 the HLAnode0 is the attachment point with the cGEM5 node into the cOMNeT++ functionality. On the left of the same figure the micro_router0 is a multiple interface micro-router which is capable of changing the network interface from Ethernet to wireless, ppp etc. Specifically, the micro-router is implemented as an INET StandardHost enhanced with routing capabilities and multiple interfaces. All the aforementioned functionality is preset and obscured from the user in order to make instantiation of a node with a desired interface as simple as possible.

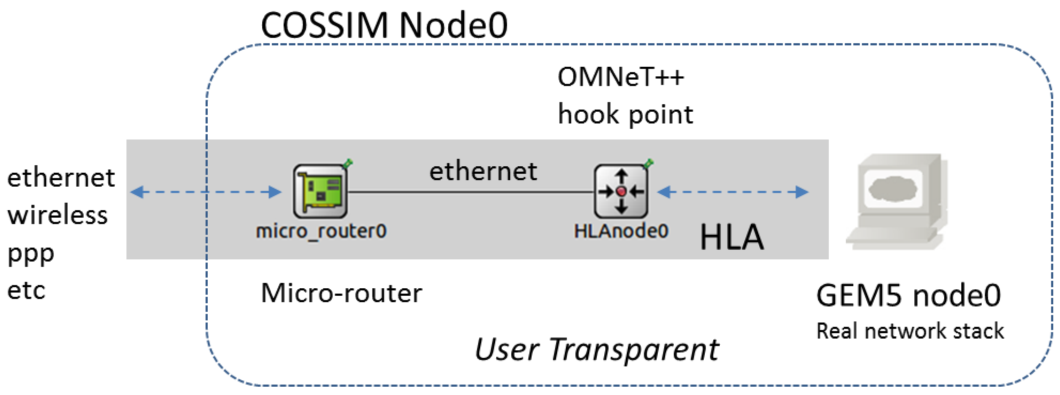


Figure 4.7. Cossim node with micro-router support

Figure 4.8 (left) depicts a scenario where 2 HLA enabled nodes are connected via an Ethernet link, while in Figure 4.8 (right) is depicted a scenario where the COSSIM micro-router links an HLA enabled node to a wireless interface. In both configurations, the IP assignment between the OMNeT++ nodes could be set either automatic or manual with similar results. In the automatic way, the configurator module assigns a whole class C network IP range for each router's separate interface, while in the manual way user specifies exactly the IPs and Subnet Masks according to the underlying network architecture using a custom routing table. This is necessary as OMNeT++ cannot handle the auto-routing in the case that all the IPs in the network are not defined at the configuration time (case where the HLA node is not assigned with an IP inside OMNeT++). The same scenario could be generalized using any other interfaces exposed to the outer network.

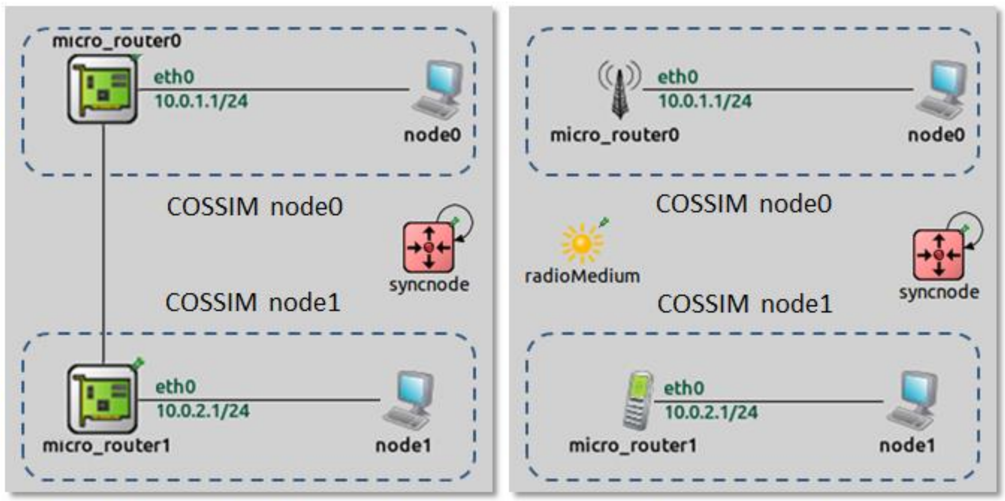


Figure 4.8. Ethernet & Wireless Network simulation with the micro-router functionality exposed

The above shortcoming can be easily addressed by using a relative simple routing configuration. In the following snippet we see the routing configuration for the node0 and node1 respectively considering the above network simulation scenario.

node0. Routing table

```
route:
10.0.2.0  10.0.0.2   255.255.255.0    G  0  wlan0
10.0.1.0  *    255.255.255.0  H  0  eth0
routeend.
```

node1. Routing table

```
route:
10.0.1.0  10.0.0.1   255.255.255.0    G  0  wlan0
10.0.2.0  *    255.255.255.0  H  0  eth0
routeend
```

4.4 Integration of Security Tools & Pause-Resume functionality

The COSSIM simulation framework integrates a number of tools that allow the CPS designer to evaluate the security of the system simulated. COSSIM's security module consists of three main sub-modules; the DoS Testing System (DTS), the Fuzz Testing System (FTS) and the Metrics Management (MM) as illustrated in Figure 4.9. The DTS is responsible for testing the resilience of a parallel system under simulation against various types of denial-of-service (DoS) attacks. As DoS is a network attack, DTS is tightly connected with the network simulation components of the COSSIM framework. The FTS provides automated testing of components' interfaces for discovering vulnerabilities. It produces input test vectors that are fed into the parallel system/application under test. This process (fuzzing) is capable of exposing errors that arise as a result of the processing of these input vectors. Fuzzing is not only used for security testing; it is also employed in quality assurance for evaluating a system's robustness. Finally, the MM observes the state of the simulated parallel system to calculate various types of security metrics, allowing the system developers to assess its behavior in certain situations.

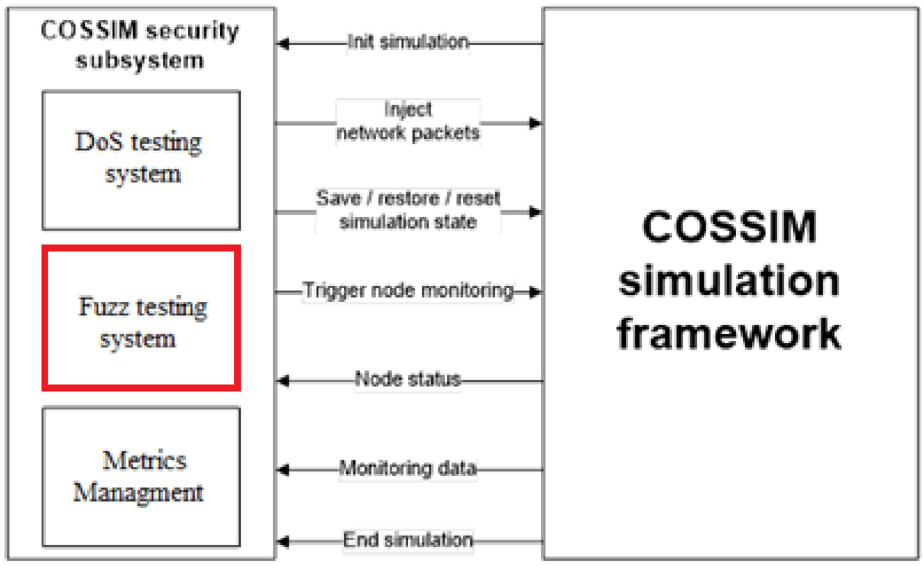


Figure 4.9. Overall concept of the Security Module

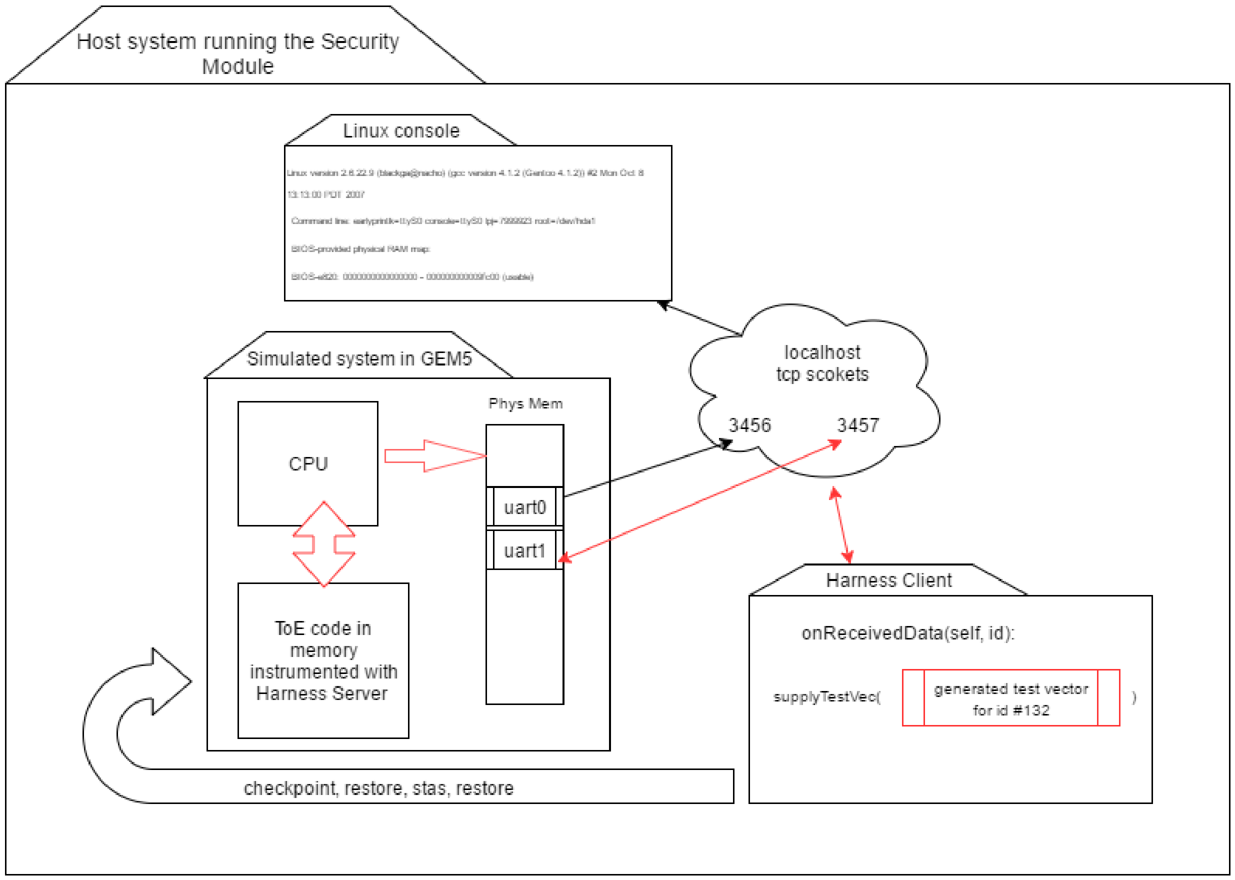
In the context of this thesis, we implement *pause-resume* functionality in order to integrate Fuzz testing system with the COSSIM framework. Initially, an overview of Fuzz Testing component implementation is described, while subsequently *pause-resume* functionality is presented so as to provide cGEM5 state and memory to Fuzz Testing component.

4.4.1 Design and operation of the Fuzz Testing component

As described in previous Section, Fuzzing is an approach to software testing whereby the system being tested is bombarded with test vectors generated by another program. The system is then monitored, in the hope of finding errors that arise as a result of processing this input. It is typically an automated or semi-automated process that involves repeatedly manipulating and supplying data to the software for processing. Thus, fuzz testing can help increase the quality of the software.

COSSIM Fuzz testing is based on [81] and it is made up of two sub-systems: the *Harness Client* and the *Harness Server* as illustrated in Figure 4.10. The Harness Server provides C based interfaces for instrumentation of the code. It is responsible for catching the data on the hooked interfaces, providing them to the *Harness Client*, fetching back the modified / fuzzed values and continuing the execution with these modified data fields. It is also possible to block the target process and wait for an instruction of the Harness Client.

The *Harness Client* manages the fuzzing effort: it has interfaces for providing the test vectors for the data fields that the Harness Server provides id's for. However, it requires to start and restore GEM5 simulations and run the test vectors from the always the same checkpoint, e.g. the same processor state which results in clean test cases. For this reason, *pause-resume* functionality is implemented in which all COSSIM components can be paused and resumed the simulation from the same simulated time as described in the following subsection in detail.


Host system running the Security Module
Linux console
Simulated system in GEM5
CPU
Phys Mem
uart0
uart1
ToE code in memory instrumented with Harness Server
localhost tcp scokets
3456
3457
Harness Client
onReceivedData(self, id):
supplyTestVec(
generated test vector for id #132
)
checkpoint, restore, stas, restore

```
Linux version 2.6.22.9 (blackga@nacho) (gcc version 4.1.2 (Gentoo 4.1.2)) #2 Mon Oct 8
13:13:00 PDT 2007
 Command line: earlyprintk=ttyS0 console=ttyS0 lpj=7999923 root=/dev/hda1
 BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
```

Figure 4.10. Architecture and operation of Fuzz Testing component

4.4.3 COSSIM Pause-Resume Functionality

Pause/resume functionality of COSSIM simulator is developed (without stopping the execution of cGEM5) in order to provide cGEM5 state and memory to Fuzz Testing component. As described in the previous section, the main concept behind fuzz testing is to send a large number of test vectors to the system under test, making sure the system is in the same exact state before each request; sending even a single test vector to the system will change its internal state and have a possible effect on the processing of subsequent test vectors, thus potentially invalidating the results of the test. In other words, it needs to take a cGEM5 checkpoint and restore cGEM5 simulations from the same checkpoint a lot of times. Although, this is possible in standalone GEM5, this is not possible in COSSIM framework because even though GEM5 is able to store and restore its own state (using checkpoints), cOMNeT++ on the other hand cannot support this feature (store/restore).

Moreover, cGEM5 cannot be stopped but paused in every checkpoint. That's why HLA connections will be dropped if some of GEM5 node stops its execution. On the other hand, GEM5 –r (restore) option can be performed only in a new GEM5 execution [82]. To encounter the above limitation, pause-resume functionality is implemented in COSSIM environment. The following steps describe an overview of this functionality.

1) Initially, the pause of particular cGEM5 node is achieved using *m5_checkpoint* inside the simulated application and send a notification to fuzz testing module to inform that there is a checkpoint (step 1). During pausing of particular GEM5 node, all COSSIM environment (other cGEM5s & cOMNET++) will be paused through Global Synchronization.
2) In turn, fuzz testing module checks periodically if there is a checkpoint. If so, it can get information of this checkpoint calling the *startChanges* function. During this state (which all COSSIM subparts are paused), *fuzz testing module* can restore the state of this specific cGEM5 node using *-r* option in a **standalone** GEM5 instance (as many times as needed – steps 2&3) but without network (because it is a standalone GEM5).
3) When fuzz testing finishes the tests, it can resume all COSSIM simulator (step 4) calling the *stopChanges* function.

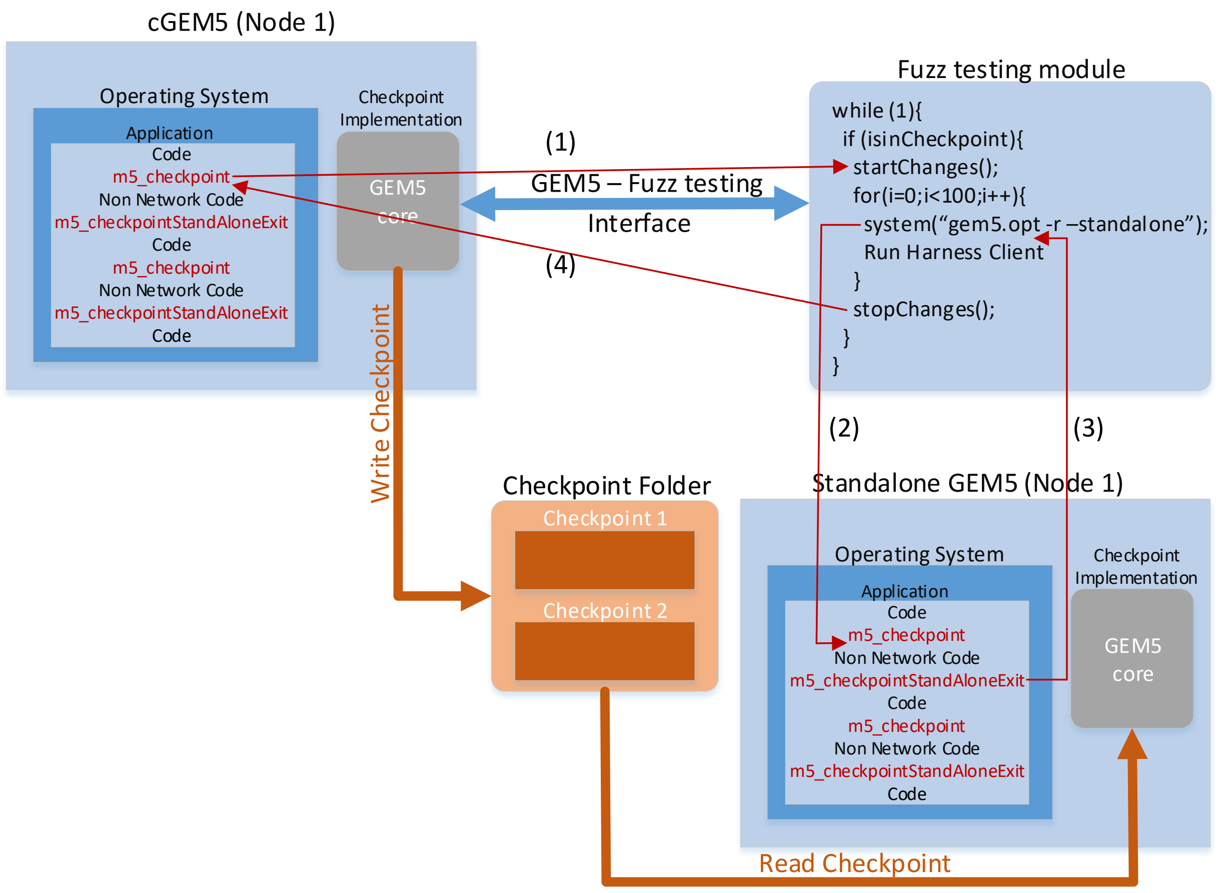


Figure 4.11. COSSIM Pause/Resume Functionality

Additionally, a mechanism to stop the standalone GEM5 is developed before the next checkpoint. For this reason, a new pseudo-instruction (which is called *m5_checkpointStand AloneExit*) is developed to stop GEM5 standalone simulation and exit the standalone execution. This pseudo-instruction is recognized only by **standalone** GEM5 and it is ignored by cGEM5. It is necessary because both cGEM5 and standalone GEM5 have the identical application code as illustrated in Figure 4.11. In other words, standalone GEM5 simulates the application code from *m5_checkpoint* to *m5_checkpointStandAloneExit*. Furthermore, *GEM5-FuzzTesting* interface is developed in order to synchronize the processing with the *Fuzz Testing* component extending the *SynchServer* functionality, while cGEM5 *Terminals* are modified to support the different terminals of each *cGEM5* and *StandAloneGem5* instance. It should be noted that *pause/resume* functionality can be used independently of *Fuzz Testing* component in case of the user needs to pause and resume the whole COSSIM/ACSIM framework modifying the cGEM5 state and memory.

4.5 General Purpose Input Output (GPIO) to include sensor devices to the processing simulator part

In the case of the application of COSSIM in the simulation of CPS, our framework can simulate the computation along with the networking aspects of a CPS in a holistic approach. As a result, COSSIM/ACSIM mainly simulates the cyber part rather than the physical part of a CPS system; in order to support the efficient simulation of the physical part as well, COSSIM provides two main options. The first one, extends cGEM5 by adding support for sensor devices in the processing system. GEM5 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. In COSSIM/ACSIM, each sensor is connected through programmed I/O using one memory bus master port to read the sensor values. To incorporate efficiently each GEM5 with the sensors a set of device drivers has been developed, while an ioctl [83] function was developed to achieve efficient user-kernel space communication. Figure 4.12 illustrates an abstract view of our COSSIM Sensor Device when integrated with cGEM5.

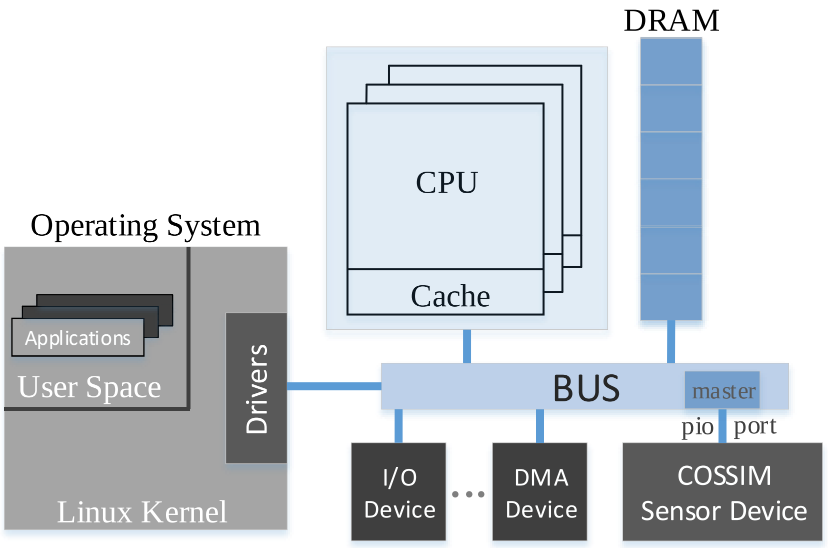


Figure 4.12. Integration of COSSIM Sensor Device with cGEM5 (full-system mode)

In addition, the COSSIM framework is able to seamlessly interconnect with other open-source approaches simulating physical aspects of the system (e.g. Ptolemy II [3]). Specifically, our tool can be connected with any other tool that has HLA interfaces in order (for each processing node) to have access to the physical processes that "generate" the sensing data. Regarding the physical aspects that relate to the network performance, those are already included in the tool through the OMNET++ simulator which offers an extension that models the physical world regarding the reaction between the physical and network processes. For example, we could model the movement of the nodes in the space, obstacles in the wave propagation, attenuation, noise, re-transmissions etc.

4.6 Power/Energy Estimator for the Processing Sub-System of the COSSIM/ACSIM Simulator

This section presents the integration of a widely used Power Estimation tool with our Processing subsystem (i.e. cGEM5) so as to estimate the Energy, Delay and Area metrics for a design with quantitative properties and activity factors[10]. In addition, in the scope of this thesis, an energy estimator is implemented which calculates the energy consumed during a specific period. Finally, a complimentary Power Model for ARM big.LITTLE Architecture based on [84] is integrated in our simulator.

4.6.1 An overview of Research Tools for High-Level Microprocessor Power Estimation

Initially, an overview of the bibliography for the current state of research-oriented tools for modelling the power consumption of microprocessors is presented. It should be noted that the bibliographic references presented cannot be exhaustive since high-level power estimation presents an active challenge in the microprocessor and electronic system design research fields and new works are frequently introduced, while older approaches and tools are deprecated or significantly altered.

[10] Power/Energy estimator for the Network sub-system is calculated from INET/MiXiM OMNET++ extension which is fully integrated in COSSIM/ACSIM.

Starting from the Cache and Memory level, the most well-known tools used are Dramsim2 and Cacti which can be used for timing and energy analysis of DRAMs and caches respectively [85] [86]. In addition, Orion2 is a model used to estimate the energy of Networks on Chip (NOCs) that are the de-facto interconnects used for chip multiprocessors [87].

On the other hand, *Wattch* [35] is a processor energy simulation tool that can provide architects with pre-silicon energy estimation. Wattch works together with the popular microarchitecture simulator SimpleScalar [34] and provides parameterized activity-based power estimates. It is more suitable to microarchitectural research rather than whole system (also since attached to SimpleScalar it does not support OS simulation) and it is not really suitable to describe custom hardware blocks (e.g. an accelerator).

For a wider system approach McPAT can be used to estimate the Energy, Delay and Area metrics for a design with quantitative properties and activity factors as inputs for the simulation [45]. McPAT is based on the aforementioned Cacti tools and tries to map architectural and technological descriptions of a CPU system to Cacti blocks in order to estimate power. The tool is designed to work together with performance simulators in order to receive activity vectors as inputs and provide power estimations that these simulators can use for power-aware functionality (e.g. hotspots, DVFS, power states etc). Gem5 is a full-system performance simulator that is widely used and most often coupled with McPAT, however there is not an automated process to integrate these tools.

4.6.2 McPAT

McPAT is selected as Power Estimation tool for processing simulator subsystem as it is very efficient since it uses the well-established CACTI models [88] and provides accurate and valid results according to the literature. Specifically, McPAT developers have provided validation studies for a number of commercially available architectures [45] [89]. The output of McPAT has been compared against published data for the 90nm Niagara processor running at 1.2 GHz with a 1.2V power supply [90], the 65nm Niagara2 processor running at 1.4 GHz with a 1.1V power supply [91], the 65nm Xeon processor running at 3.4 GHz with a 1.25V power supply [92], the 180nm Alpha 21364 processor running at 1.2 GHz with a 1.5V power supply

[93], the 45nm dual-core Diamondville Atom processor running at 1.6 GHz with a 1.0V power supply [94], and the 40nm Cortex A9 dual-core hard IP implementation running at 2.0 GHz [95]. These include in-order and out-of-order processors, single-threaded and multithreaded processors, as well as high performance and embedded processors in the validation targets. Thus, the validations stress McPAT in a comprehensive and detailed way.

Although there is a number of works which integrate the McPAT with GEM5 simulator[11] [12], this is not a trivial process because there are issues about the compatibility of Gem5 outputs and CACTI thresholds as described in the following subsections. In the scope of this thesis, we provide an integrated solution which seamlessly interconnect the cGem5 with the McPAT. The proposed solution can be used with the generic versions of gem5 and McPAT (available from the official gem5 and McPAT repositories) [96]. Figure 4.13 demonstrates the block diagram of our solution; boxes with dashed lines are implemented in the scope of COSSIM/ACSIM.

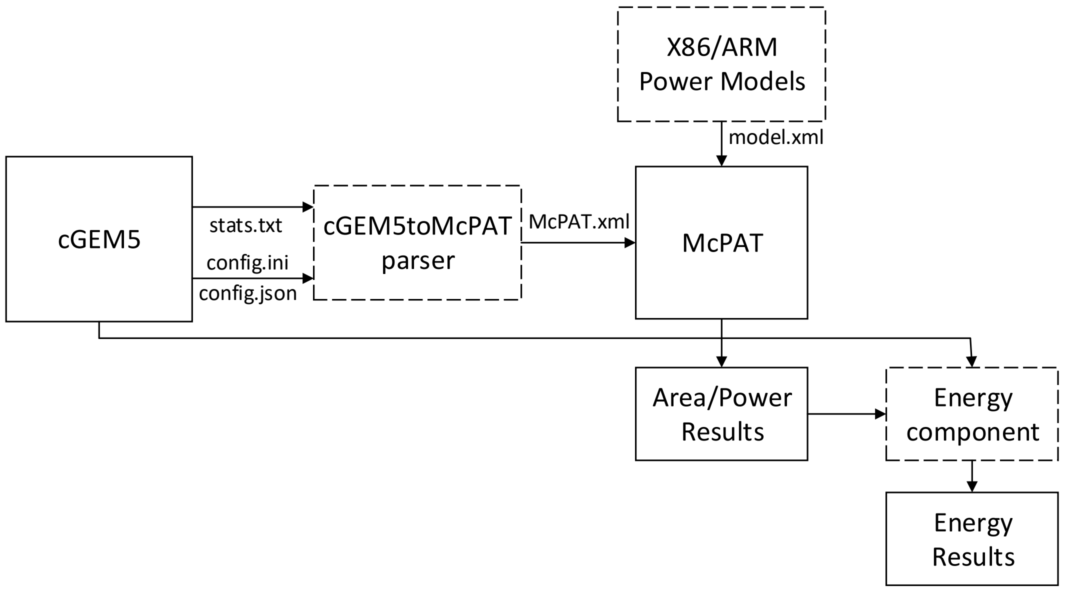


Figure 4.13. Integration of cGEM5 with McPAT and Energy component

[11] https://github.com/markoshorro/gem5McPATparse

[12] https://github.com/harvard-acc/gem5-aladdin/blob/master/sweeps/gem5tomcpat/GEM5ToMcPAT.py

As illustrated from Figure 4.13 McPAT requires as input an XML file that holds both parameter information (at the architectural, circuit and modeling level) and usage (runtime statistics) information. On the other hand, a GEM5 simulation run upon completion will generate three files (among other data that are not relevant in the scope of this thesis):

- stats.txt – a text file containing the statistics of a specific run. The simulation statistics provide run-time usage information of the different units present in the system modeled by GEM5. This run-time usage information are the activity factors that McPAT requires in order to estimate the run-time power consumption.
- config.ini / config.json – files describing the configuration of the system modeled in GEM5 (conveys architectural information).

GEM5toMcPAT parser which is presented in [97] is used and modified. Specifically, a series of changes have been made to address the following issues.

- **Reporting issues in stats.txt output from GEM5** When used with the simpleCPU models, GEM5 presents some values in decimal form while they should be displayed in integer form. Apparently McPAT cannot handle decimal values for cycles. The conversion parser has been properly modified to handle such cases and provide correct roundings (the correct roundings can be found when parsing the stats.txt files and retrieving the overall values for the specific numbers).
- **Issues with McPAT (CACTI)** CACTI will not handle cases where certain properties have values below a threshold. A prime example is the width of the cache lines. While CACTI will prompt an error on such cases, it can handle without issue these corner cases. Therefore, we have patched CACTI (and therefore McPAT) to handle those cases internally without issuing an error and halting execution. We have validated the results and a newer version of McPAT has been produced.

In addition, we have designed two basic processor description templates that can be used with GEM5's simpleCPU models. The first one is for ARM-based processors and the second for x86-64 processors. In order to construct those power models, we have used the configuration and statistics information as provided by GEM5 as well as internal knowledge on how GEM5 models the specific CPUs. The result are two xml template files that can be used with the GEM5-to-McPAT conversion tool that will produce working and acceptable McPAT inputs.

In its current form, McPAT only reports power estimations. For COSSIM/ACSIM though, energy estimations are also required. From the outputs of GEM5 (stats.txt file) the overall time required to execute the desired software application is provided. Similarly, from the output of McPAT, the dynamic power as well as the leakage power are reported. We have developed a simple energy component that parses the outputs of GEM5 and McPAT and calculates the energy consumed during the specific run as:

$$E = \frac{P_{dynamic} + P_{leakage}}{RunTime}$$

The above functionality is executed automatically given the Gem5 requirement outputs, and it is integrated in our Graphical User Interface as described in the next Section.

McPAT consist of two phases: the *chip representation building phase* and the *runtime power computation phase*. For the 1st phase, only the statically config.ini / config.json files are required in order to represent the chip, while in the 2nd phase, the statistics file (stats.txt) is required in order to calculate the final runtime power dissipation. Authors in [98] presents a novel partition of these two phases so as to gain speedup in the whole process. In order to exploit this functionality, cGEM5 is modified to produce gem5 stats in real time and therefore to produce power/energy results in predefined simulated time intervals. Specifically, the *m5 dumpstats* pseudo-instruction has been extended to incude this functionality providing a number of statDump<x>.txt, where x indicates the statDump file number which increments by one for each file generated. As described above, McPAT tool cannot parse directly a statDump<x>.txt files produced by cGEM5. So, each of the statDump<x>.txt files are converted to mcpatIn<x>.xml, which follow the McPAT expected input format, using the GEM5ToMcPAT parser as illustrated in Figure 4.14.

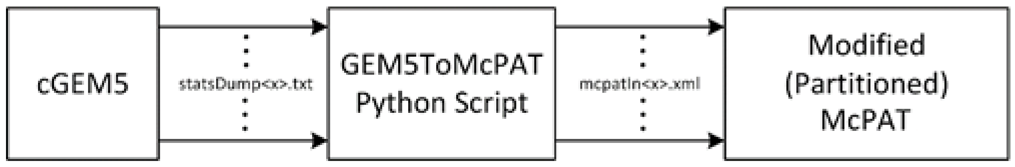


Figure 4.14. Integration of cGEM5 with partitioned McPAT using real-time statistics

4.6.3 Complimentary Power Model for ARM big.LITTLE Architecture

In contrast with McPAT that tries to build a chip representation of the simulated processor, authors in [84] propose power model which requires as input the instruction mix of the program executed. Specifically, they analyze an ARM big.LITTLE architecture and study the performance and energy tradeoffs of the big and the LITTLE ARM cores at different voltage and frequency levels. They investigate how the workload characteristics and their execution on a particular core type affect energy consumption. Finally, they develop a lightweight energy model, suitable for runtime use, using as input parameters only the instructions per cycle (IPC) and instruction mix.

In the context of this thesis, we integrate the big.LITTLE complementary power model with our cGEM5 processing simulator as illustrated in Figure 4.15. As referred in [99] Gem5 can produce a trace file with all simulated commands using appropriate configuration. This functionality is used to get the simulated instruction mix, while stats file is used to get the number of execution cycles. In order to get the number of instructions executed per instruction type and statistics from cGEM5 according to big.LITTLE specific categories[13], GEM5toBigLittle parser is implemented. A significant problem was that the trace file can become very large very quickly due to millions of simulated instructions and it is practically impossible to use the big.LITTLE power model. For this reason, *Pause-resume* and *real-time* statistics functionalities (which are presented in the previous sections) are used in order to pause the whole COSSIM process in predefined intervals, get the appropriate cGEM5 outputs and calculate the power using the big.LITTLE architecture.

[13] Authors categorize the ARM instruction set in a logical way that differentiates instructions based on their functionality and what part of the processor datapath they utilize. Specifically, they differentiate them internally in each category based on operands type and instruction flavor for each case. As they referred, their categorization consists of Arithmetic/Logic, Data Movement, Compare/test and Load/Store instructions.

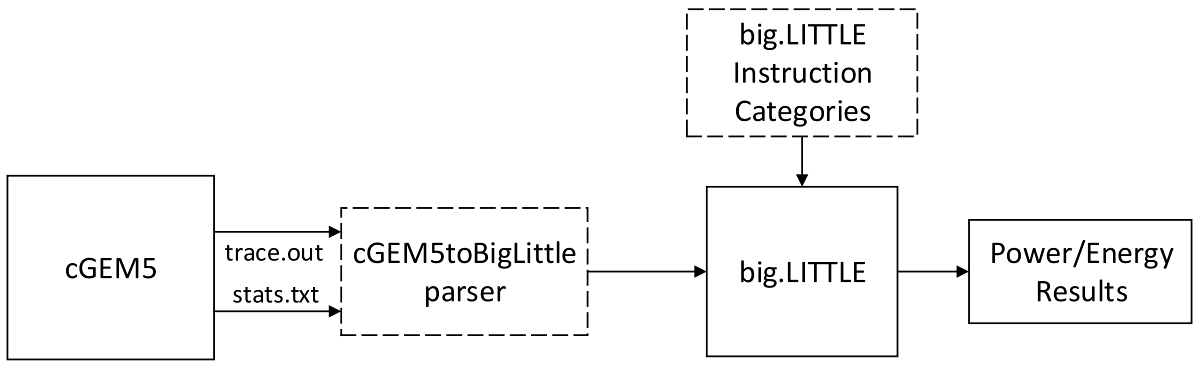


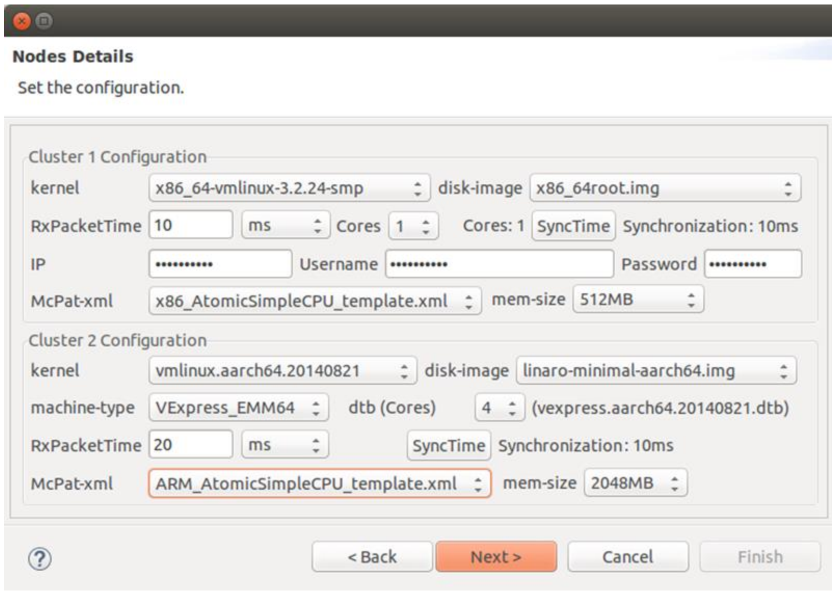
Figure 4.15. Integration of cGEM5 with big.LITTLE power component

4.7 Graphical User Interface

This section presents the simulation configuration and execution monitoring tool. The tool allows for the configuration of the nodes, the network and the selection of the monitoring metrics. According to the simulation configuration only the required parts of the simulator will be engaged in the simulation (e.g., a power consumption model will not be committed if relevant results are not requested) so as to provide reasonable matches between the speed of the simulation and the requested simulation detail.

The Eclipse-based GUI of OMNeT++ has been extended so as to fully integrate the capabilities of the COSSIM tool. Specifically, our GUI consists of two Eclipse plugins: (i) the simulation configuration tool and (ii) the execution monitoring tool.

The simulation configuration tool has the form of a wizard which is installed as a plugin in Eclipse/OMNeT++ and guides the user through the GEM5 configuration process for each of the simulated nodes. This process is a very time-consuming one as it is usually performed through the command line and needs a large number of parameters to be set for each of the nodes. Based on our initial measurements within the COSSIM/ACSIM design team, the GUI, by itself, reduces the configuration time of the simulation by 90% in the case of a 10-node system. In addition, our GUI prevents the user from setting wrong parameters and thus minimizing the risk of starting a time-consuming simulation that will latterly be proven wrong or inadequate. Figure 4.16 depicts the 3rd step of the wizard in which the user can define a number of parameters of each cluster depending on the 2nd step configuration.



Cluster 1 Configuration

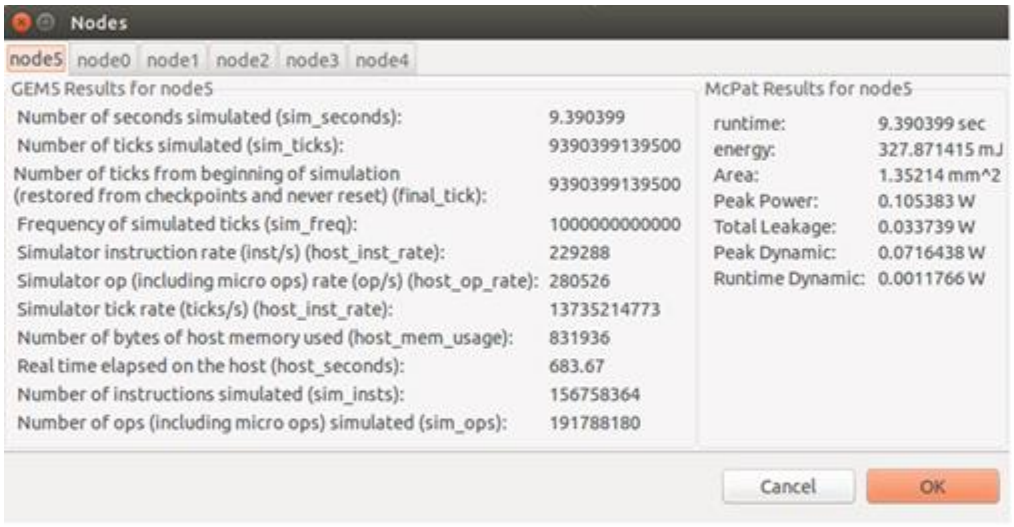
kernel x86_64-vmlinux-3.2.24-smp disk-image x86_64root.img
RxPacketTime 10 ms Cores 1 Cores: 1 SyncTime Synchronization: 10ms
IP •••••••••• Username •••••••••• Password ••••••••••
McPat-xml x86_AtomicSimpleCPU_template.xml mem-size 512MB

Cluster 2 Configuration

kernel vmlinux.aarch64.20140821 disk-image linaro-minimal-aarch64.img
machine-type VExpress_EMM64 dtb (Cores) 4 (vexpress.aarch64.20140821.dtb)
RxPacketTime 20 ms SyncTime Synchronization: 10ms
McPat-xml ARM_AtomicSimpleCPU_template.xml mem-size 2048MB

Figure 4.16. Cluster configuration Parameters

The second plugin that has been developed is the COSSIM/ACSIM execution monitoring tool which is a graphical interface that integrates and visualize the most important cGEM5 and McPat results. The output results that the monitoring tool displays, can be presented either per node or per simulated parallel system. That means that the tool can show the results of each node separately or of all of them together. Figure 4.17 and Figure 4.18 illustrate the cGEM5 & McPaT results for a certain simple application scenario with 6 nodes.



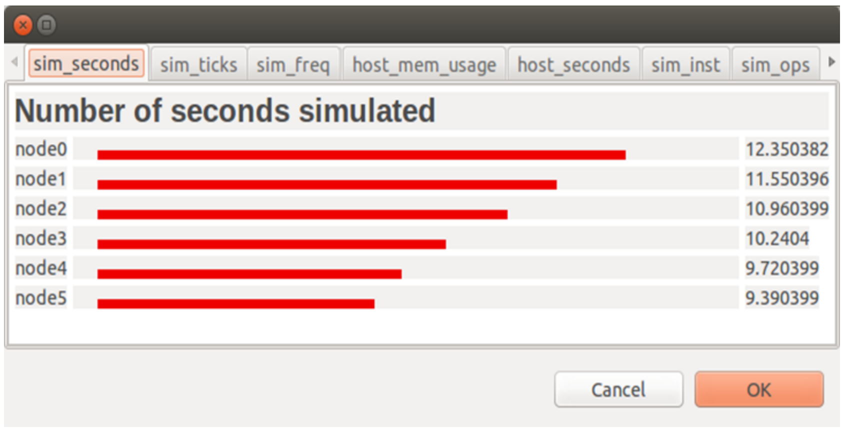
GEMS Results for node5

Number of seconds simulated (sim_seconds):	9.390399
Number of ticks simulated (sim_ticks):	9390399139500
Number of ticks from beginning of simulation (restored from checkpoints and never reset) (final_tick):	9390399139500
Frequency of simulated ticks (sim_freq):	1000000000000
Simulator instruction rate (inst/s) (host_inst_rate):	229288
Simulator op (including micro ops) rate (op/s) (host_op_rate):	280526
Simulator tick rate (ticks/s) (host_inst_rate):	13735214773
Number of bytes of host memory used (host_mem_usage):	831936
Real time elapsed on the host (host_seconds):	683.67
Number of instructions simulated (sim_insts):	156758364
Number of ops (including micro ops) simulated (sim_ops):	191788180

McPat Results for node5

runtime:	9.390399 sec
energy:	327.871415 mJ
Area:	1.35214 mm^2
Peak Power:	0.105383 W
Total Leakage:	0.033739 W
Peak Dynamic:	0.0716438 W
Runtime Dynamic:	0.0011766 W

Figure 4.17. Simulated node results for a specific combination of nodes



Number of seconds simulated

node0	12.350382
node1	11.550396
node2	10.960399
node3	10.2404
node4	9.720399
node5	9.390399

Figure 4.18. Comparison of "Number of seconds simulated" value

5 Novel Intercommunication and Synchronization Mechanism

One of the main novel aspects of COSSIM/ACSIM is the developed intercommunication and synchronization scheme in order to communicate seamlessly the processing with the network components, which is fully compliant with the IEEE HLA standard. In this Chapter, initially, an overview of both HLA and CERTI implementation is presented in Section 5.1. Subsequently the subset of HLA services necessary to allow *cGEM5/cOMNET++* models to participate in an HLA federation are described (Section 5.2), focusing on the time management in HLA simulation. Furthermore, Section 5.3 describes in detail *COSSIMlib* which provides the interface between *cGEM5/cOMNET++* and *HLA/CERTI* environment. Finally, Section 5.4 presents our COSSIM/ACSIM Synchronization scheme, while Section 5.5 presents the *SynchServer* which is responsible to initialize both *cOMNET++* and *cGEM5* federations.

5.1 Overview of HLA and CERTI Implementation

Bringing the processing and the network simulators together requires carefully designed communication interfaces and synchronization schemes. This bidirectional interface will have to pass information of the type and timing of events and to provide a common data representation, since data are represented differently in the processing and the network simulators. Passing actual data between the two simulators is necessary for supporting simulation of real use cases. As referred in Chapter 3, the IEEE standard High-Level Architecture (HLA) is selected for interconnection of the processing and networking simulation sub-systems, while specifically CERTI HLA is configured and extended as the most appropriate HLA implementation for the COSSIM simulator.

HLA is a standard for distributed discrete-event simulations, generally used to support analysis, engineering and training; it has been developed so as to promote reusability and interoperability. In HLA terminology, the logical representation of an interconnection of different simulators is called a *Federation* and includes multiple modules, which are called *Federates*, and which communicate via a *Runtime Infrastructure* (RTI). The RTI provides a number of services like *Federation Management*, *Time Management* and *Object Management* that are utilized in simulation control, synchronization and data exchange [71]. The connection between the simulators, based on the HLA standard, is established by a federate library which encapsulates the HLA interfaces and which can be specialized to a certain simulation scenario; the communication from the RTI to a federate and vice versa is established via the *RTI-Ambassador* and the *Federate-Ambassador* modules which cause a strict segregation of simulation while supporting several communication primitives as illustrated in Figure 5.1. Specifically, Figure 5.1 presents one Federation for one *cGEM5* instance (Federate 1) together with the communication with the *cOMNET++* counterpart node (Federate 2). Finally, the *Federation Object Model* (FOM) is a file that contains a description of the data exchange process within the federation, including for example the objects and interactions that will be exchanged.

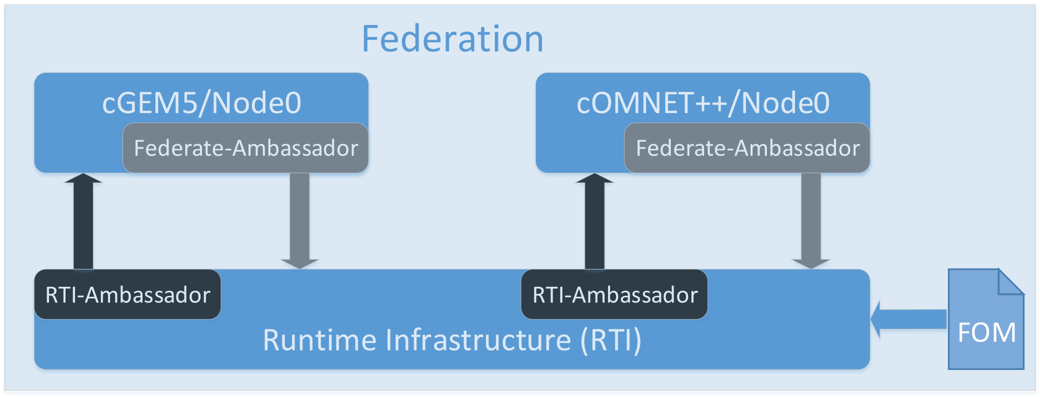


Figure 5.1. The High-Level Architecture (HLA)

CERTI is an HLA RTI implementation which developed since 1996 by ONERA, the French Aerospace Lab. Figure 5.2 illustrates the HLA layered implementation using CERTI. The lower layers consist of two types of processes, local ones called RTI Ambassadors (RTIA) and a central one called RTI Gateway (RTIG). These processes are linked with each other using Unix and TCP sockets. Thereby, the RTIG is of predominant importance since any form of communication between federates, either for data exchange or for synchronization purposes, is done via the RTIG.

Specifically, each federate process interacts locally with an RTI Ambassador process (**RTIA**) through a Unix-domain socket. The RTIA processes exchange messages over the network, in particular with the RTIG process, via TCP (and UDP) sockets, in order to run the various distributed algorithms associated with the RTI services. A specific role of the RTIA is to immediately satisfy some federate requests, while other requests require network message sending or receiving. The RTIA manages memory allocation for the message FIFOs and always listens to both the federate and the network (the RTIG).

On the other hand, the RTI Gateway (**RTIG**) is a centralization point in the architecture in order to simplify the communication implementation. The RTIG manages the creation and destruction of federation executions and the publication/subscription of data. It plays a key role in message broadcasting which has been implemented by an emulated multicast approach. When a message is received from a given RTIA, the RTIG delivers it to the interested RTIAs, avoiding a true broadcasting.

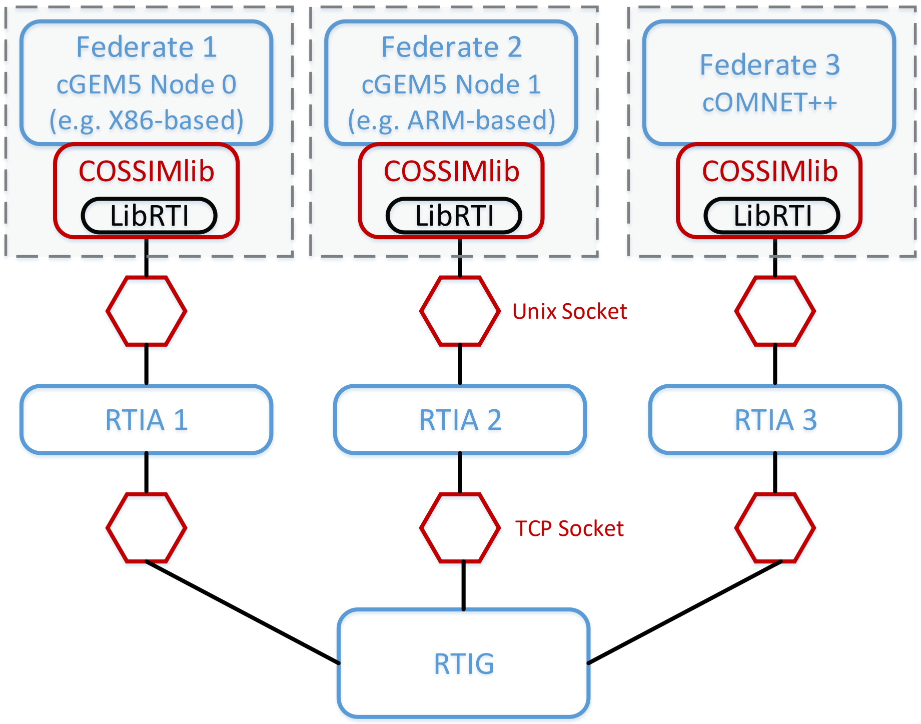


Figure 5.2. The CERTI Architecture

Each federate communicates with RTIA using *libRTI*; a library which links each federate containing a number of functions such as federation creation/destruction, object manipulation, time management etc. as described in the next sections. *COSSIMlib* is implemented to enable the interoperability between *cGEM5/cOMNET++* model and the *HLA/CERTI* federation.

5.2 CERTI HLA Services for COSSIM/ACSIM implementation

HLA services are grouped into four groups based on where they are utilized in the federate life cycle. The services used in COSSIM/ACSIM cover the following aspects:

i. **Federation management** - Defines how federates can connect to the RTI, create, join and manage federations, save and restore federation states and defines a system implementing an accurate synchronization scheme for all the federates.
ii. **Declaration management** - Defines how federates declare their intentions with regard to publication and subscription of classes and interactions.
iii. **Object management** - Defines how federates can utilize objects and interactions.

iv. **Time management** - Defines how time is used in a federation and how it affects object and interaction updates, federate saves as well as all the other implemented services.

The following subsections describe the subset of HLA services used in our framework together with a brief description of each service. The reader is referred to [100], [101] for a complete description of all HLA services. Specifically, Section 5.2.1 describes the *Federation management services* used by the RTI to manage the whole federation; Section 5.2.2 the *Declaration management services*, i.e. which object or objects attributes each federate will publish or subscribe to; Section 5.2.3 the *Object management services*, i.e. the way federates produce attribute updates or receive updated attributes from the federation; and Section 5.2.4 the *Time management services*, i.e. the mechanisms required to implement time management policies and to negotiate time advances.

5.2.1 Federation Management Services for COSSIM/ACSIM implementation

Table 5.1 summarizes the Federation Management Services which are used for the COSSIM/ACSIM implementation.

Services	**Description**
createFederationExecution()	Creates a named federation execution (*FedExecution*) and registers it with the RTIG.
destroyFederationExecution()	Unregisters a named federation and shuts down the associated FedExecution.
joinFederationExecution()	Requests permission to participate in a named federation and initializes the RTI ambassador with federation specific data.
resignFederationExecution()	Terminates the federate's participation in a federation.
registerFederationSynchronizationPoint()	Register a synchronization Point
synchronizationPointRegistrationSucceeded() *	Synchronization point succeeded

announceSynchronizationPoint() *	If the registration succeeds, all federates to which the synchronization point is applicable will receive an announcement in the form of this callback
synchronizationPointAchieved()	Release from a synchronization point
federationSynchronized() *	Announce synchronization to relevant federates
tick()	This service is invoked by the federate to yield processor time to the RTIA. During a tick() invocation, the RTIA will process incoming traffic, deliver callbacks to the federate, and perform various internal RTI maintenance essential to the operation of the federation

Table 5.1. Federation Management Services for COSSIM implementation (services with a * are sent from RTI to Federates (callbacks); all other services are from Federates to RTI)

Figure 5.3 illustrates the primary functions associated with the federation management life cycle. Initially, the local *RTIA* communicates (for the first time) with the RTIG process calling the *createFederationExecution()* function. If the specified federation does not exist, the RTIG process creates a new *FedExececution* process and associates it with the supplied federation name. If the specified federation already exists, a *FederationExecutionAlreadyExists* exception is raised. The same executable may at times be called upon to create a federation and at other times may be asked to participate in an established federation[14]. If the *FederationExecutionAlreadyExists* exception is caught and ignored, then the call to *createFederateExecution()* is robust – creating the federation if required and tolerating the existence of an existing federation execution.

Subsequently, the *joinFederationExecution*() method is called to associate a federate with an existing federation execution. The method provides the name of the calling federate and the name of the federation execution that the federate is attempting to join. If the

[14] This is certainly the case if the same simulation code is executed multiple times to function as multiple federates in a federation.

joinFederationExecution() is called too quickly following a *createFederationExecution()* call, the *FedExecution* process may not yet be initialized to communicate with the federate that successfully created the *FedExecution* (typically the first federate). For this reason *SynchServer* is implemented to ensure the correct order of *createFederationExecution()* and *join FederationExecution()* among different (cGEM5s and cOMNET++) executable instances as described in Section 5.4.

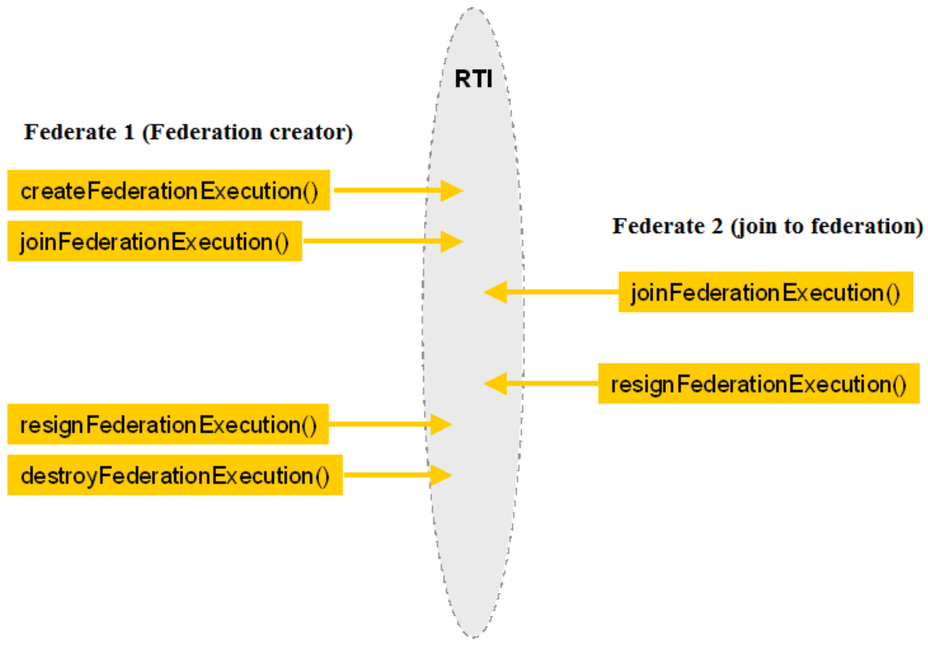

RTI
Federate 1 (Federation creator)
createFederationExecution()
joinFederationExecution()
Federate 2 (join to federation)
joinFederationExecution()
resignFederationExecution()
resignFederationExecution()
destroyFederationExecution()

Figure 5.3. Federation Management Life Cycle [102]

During process termination, *resignFederationExecution()* method is called to terminate a federate's participation in a federation. When one *federate* leaves the *federation*, something must be done with the objects for which the federate has update responsibility. Finally, the *destroyFederationExecution()* method attempts to terminate an executing federation. If successful, the *FedExecution* associated with the federation terminates. If the invoking federate isn't the last federate participating in the targeted federation, a *FederatesCurrentlyJoined* exception is raised.

Moreover, RTI provides functions for **synchronizing activities between federates participating in a federation** using mechanisms for exchanging information between federates. It's possible to associate times with exchanged information and thereby coordinate federate activities. The Federation Management synchronization functions allow federates to communicate explicit synchronization points. Figure 5.4 illustrates the *RTIambassador* service functions as well as *FederateAmbassador* callbacks that together support the synchronization capability.

Specifically, *registerFederationSynchronizationPoint()* initiates the establishment of a named checkpoint that serves to synchronize some or all federates according to federation-defined semantics. Subsequently, the federate is appraised of the success of a synchronization point through *synchronizationPointRegistrationSucceeded()* service. If the registration succeeds, all federates to which the synchronization point is applicable will receive an announcement in the form of a *announceSynchronizationPoint()* callback. *SynchronizationPointAchieved()* informs the *federation* that the *federate* has met the federation-defined criteria associated with a synchronization point that has previously been announced to the federate. When all federates included in the synchronization point (including recently joined federates if a universal synchronization point was registered) have achieved synchronization or resigned, the relevant federates will be informed of synchronization through a *federationSynchronized()* callback.

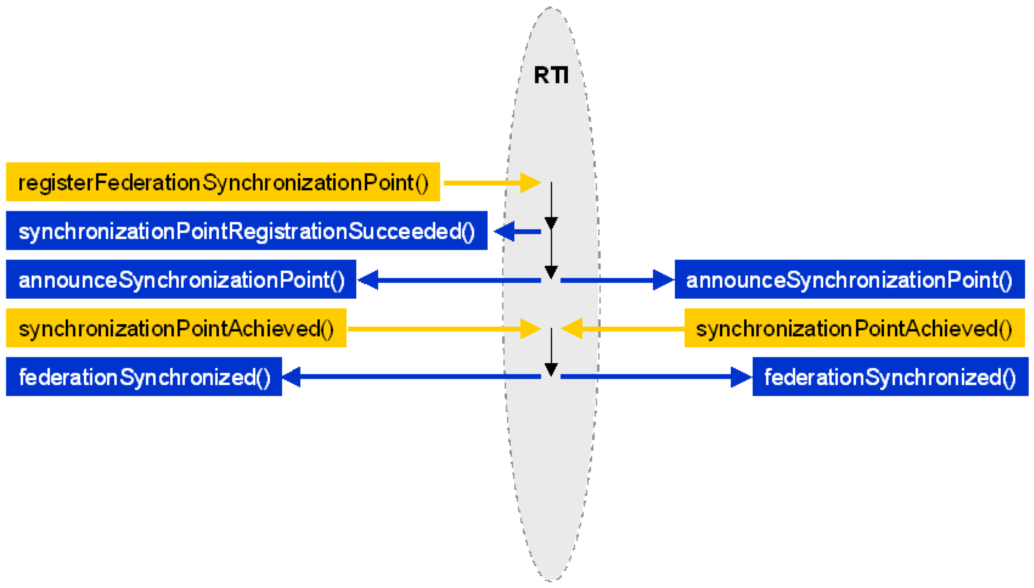

RTI
registerFederationSynchronizationPoint()
synchronizationPointRegistrationSucceeded()
announceSynchronizationPoint()
announceSynchronizationPoint()
synchronizationPointAchieved()
synchronizationPointAchieved()
federationSynchronized()
federationSynchronized()

Figure 5.4. Federate Management Synchronization **(services with *blue* are sent from RTI to Federates (callbacks), while services with *orange* are from Federates to RTI) [102]**

5.2.2 Declaration Management Services for COSSIM implementation

Table 5.2 summarizes the Declaration Management Services which are used in the implementation of our framework.

Services	Description
publishInteractionClass()	Conveys the intention of a federate to begin generating interactions of a specified class
unpublishInteractionClass()	Conveys the intention of a federate to cease generation of interactions of a specified class
subscribeInteractionClass()	Declares a federate's interest in receiving a specified class of interactions
unsubscribeInteractionClass()	Withdraws a federate's interest in receiving a specified class of interactions

Table 5.2. Declaration Management Services for COSSIM implementation

PublishInteractionClass() informs the RTIA that the federate may begin generating interactions of the specified class, while the *subscribeInteractionClass()* service instructs the RTIA to deliver interactions of a specified class to the federate. Subsequent instances of the specified interaction class occurring in the federation will be delivered to the federate in the form of *receiveInteraction()* callbacks as described in the next subsection. Finally, *unsubscribeInteraction Class()* service instructs the RTIA to cease delivering interactions of the specified class to the federate, while *unpublishInteractionClass()* service informs the RTIA that the federate will no longer generate interactions of the specified class.

In other words, the functions mentioned in Table 5.2 are used to announce the *DataPacket* exchange and direction. For example, cGEM5 instance publishes Interaction Class *"GEM5_TO_OMNET"* to send Interactions (*Data Packets*) from cGEM5 to cOMNET++ counterpart node, while it subscribes Interaction Class *"OMNET_TO_GEM5"* to receive Interactions from cOMNET++ to cGEM5 counterpart node.

5.2.3 Object Management Services for COSSIM/ACSIM implementation

Table 5.3 summarizes the Objects Management Services which are used in the implementation of our framework. Specifically, COSSIM uses two services to exchange Data Packets (Header, Payload, Length, etc.) from cGEM5 to cOMNET++ counterpart node; *sendInteraction()* service to send COSSIM Interaction and *receiveInteraction()* callback to deliver the interaction.

Services	Description
sendInteraction()	Generates an interaction event in the federation
receiveInteraction() *	Interaction instances will be delivered, using this callback, to remote federates subscribing to the associated interaction class

Table 5.3. Object Management Services for COSSIM implementation (services with a * are sent from RTI to Federates (callbacks); all other services are from Federates to RTI)

5.2.4 Time Management Services for COSSIM/ACSIM implementation

HLA time management services enable, for the first time in the area of highly parallel system simulation, deterministic, cycle accurate and reproducible distributed simulations. Each federate manages its own logical time and communicates this time to the RTI. The RTI ensures correct coordination of federates by advancing time coherently. Logical time is roughly equivalent to "simulation time" in the classical discrete event simulation terminology and is used to ensure that federates observe events in the same order [103]. However, logical time is not necessarily mapped to real time. The following paragraphs introduces the HLA Federate's time policies and progress, while Table 5.4 summarizes the Time Management Services which are used in the implementation of our framework.

i. **Federate's time policies.** HLA time policies describe the involvement of each federate in the progress of time. It may be necessary to map the progress of one federate to the progress of another. There are two sets of federates: A *regulating federate* participates actively in the decisions for the progress of time while a *constrained federate* follows the time progress imposed by other federates. A combination of both federates is also allowed by HLA. However, since our approach focuses on the synchronization of logical time from different simulation tools, only pure regulating and constrained federates are allowed.

ii. **Time progress.** Time advancement requests by federates are made through the HLA *timeAdvanceRequest* service (TAR) which is used to implement time-stepped federates, while the granted time is provided by *timeAdvanceGrant* (TAG) callback. The time advancement phase of a Federate *F* in HLA is a three-step process: i) F sends a request using the TAR service; ii) *F* can receive interactions (Data Packets) using *receiveInteraction* callback; iii) *F* waits for the granted time tG (TAG). At the TAG(tG) reception, the federate's local time will be advanced to tG according to the data in the TAR request.

Services	**Description**
enableTimeRegulation()	Declare that federate is regulator
timeRegulationEnabled() *	Federate as regulator succeeded
enableTimeConstrained()	Declare that federate is constrained
timeConstrainedEnabled() *	Federate as constrained succeeded
queryFederateTime()	Allows the federate to query the RTI for its current logical time
timeAdvanceRequest(), TAR	Requests an advance of the logical time of the federate to a specified federation time
timeAdvanceGrant(), TAG *	Notify that time Advancement granted

Table 5.4. Time Management Services for COSSIM implementation (services with a * are sent from RTI to Federates (callbacks); all other services are from Federates to RTI)

The time management scheme and its semantics are shown in Figure 5.5. As demonstrated in this example, a federate produces an event at time *t1* sending one *DataPacket* using *sendInteraction()*. The current (logical) time is *t1*; let us consider that the next event is *e2* with timestamp t2, t2 = t1 + Δ, Δ > 0. The federate asks the RTI whether the time should be progressed with the invocation of TAR(t2). Until the reception of the TAG(t2) callback, the logical time of the federate is stalled at *t1*, and it can receive a *ReceiveInteraction(v,t1′)* callback. Timestamp *t1′* is in the interval [*t1*, *t2*]. Subsequently, Federate can execute a computation step with the values received within the *ReceiveInteraction* callback. The federate then receives TAG(t2) and can increase its local time to *t2*. In a nutshell, if a TAR(t2) has been sent, the time granted by the TAG service is *tG* = *t2*.

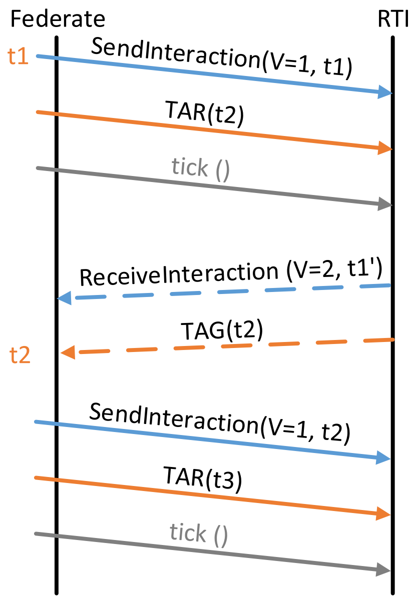


Figure 5.5. Time advancement services

5.3 COSSIMLib Architecture

The implemented *COSSIMlib* enables the interoperability between *cGEM5*/*cOMNET*++ and the *CERTI*/*HLA* Federations. It is built on top of the CERTI API, a C++ binding of CERTI based on HLA version 1.3. Since the HLA communication mechanisms are based on TCP/IP packets, both *RTIG* and *SynchServer* can be executed either in the same physical machine or in a remote server. The same applies for all components of the simulator (i.e. each *cGEM5* instance and *cOMNET*++), thus enabling fully distributed simulation of large parallel systems. Table 5.5 summarizes the implemented methods in *COSSIMlib* and the corresponding HLA services of previous subsections.

	cGEM5/cOMNET++ attribute	CERTI bindings	HLA
pre-initialize functions	join()	createFederationExecution	Federation
		joinFederation	
	publish()	publishInterctionClass	Declaration
	subscribe()	subscribeInteractionClass	
Initialize functions	setTimeRegulation()	enableTimeRegulation	Time
		timeRegulationEnabled (callback)	
		enableTimeConstrained	
		timeConstrainedEnabled (callback)	
	pause()	registerFederationSynchronizationPoint	Federation
		announceSynchronizationPoint (callback)	
	synchronize()	synchronizationPointAchieved	
		federationSynchronized (callback)	
Interaction functions	sendInteraction()	sendInteraction	Object
	step()	queryFederateTime	Time
		timeAdvanceRequest	
		timeAdvanceGrant (callback)	
	getPacket()	receiveInteraction (callback)	Object
finalize function	resign()	resignFederationExecution	Federation
		destroyFederationExecution	

Table 5.5. Overview of *COSSIMlib* Architecture

The *Pre-initialize* methods of the *COSSIMlib* implement certain services of the Federation management (e.g. create and join tasks for a federation) and some services of the Declaration management which have to do with the publication and subscription of object's instances in a federation. Specifically, 3 basic methods are implemented: *Join* which is responsible for creating a Federation and join the Federate to this Federation; *Publish* which is responsible to publish the interaction class so as to allow for the sending of Data Packets; *Subscribe* which is responsible to subscribe to the interaction classes in order to receive Data Packets.

The *Initialize* methods of the *COSSIMlib* implement certain services of the Federation & Time management which are relevant to the HLA synchronization and time policy (e.g. constrained and regulating federates). Specifically, 3 basic methods are implemented:

a. *SetTimeRegulation* to declare the federate policy; Firstly, it calls *queryFederateTime* service to get its local federate time and subsequently, it calls *enableTimeRegulation* service to declare the regulating policy, with its local time plus the next *TIME_STEP*[15] (lookahead). There is a possibility the *local* plus *TIME_STEP* to be less than *FEDERATION_TIME (local + TIME_STEP < FEDERATION_TIME)* as a result *FederationTime AlreadyPassed* exception is raised so that Federate proceeds its *local* time step by step. This task is implemented using *timeAdvanceRequest* HLA service to request to advance the federate's time and *tick()* HLA service to yield processor time to the RTIA.
b. *Pause* to initiate the establishment of a named checkpoint that serves to synchronize all federates according to federation-defined semantics (it is necessary so that RTI pauses the global time until all federates to enter in this federation).
c. *Synchronize* to declare that all other federates (in same federation) have joined.

The Send & Receive Interaction methods of the *COSSIMlib* implement certain *Time management* and *Object management* Services. Specifically, 3 basic methods are implemented which support the send & receive Interactions within the HLA Federations: *SendInteraction*

[15] In COSSIM simulator the TIME_STEP is defined equal to 1 for both simulators.

sends Data Packets from *cGEM5/cOMNET++* Federate to the CERTI server; *getPacket* which inherits the *receiveInteraction* callback decodes and reads the *DataPacket* whenever is triggered (Specifically, one *queue* is created (*packetBuffer*) so as to save the *DataPackets* in the correct order; whenever *receiveInteraction* callback is triggered, an *RTI::MessageBuffer* is created to read the *interactionParameters*); *Step* which requests an advance of the logical time of the federate to a specified federation time using the *timeAdvanceRequest* HLA service. Finally, the *Resign* (finalize) method is implemented which is related to the *Declaration* and *Federation* management services and which is responsible for the correct termination (i.e. resignation) of the federates as well as the destruction of a federation.

5.4 Synchronization between COSSIM/ACSIM Modules

The actual COSSIM/ACSIM synchronization scheme consists of two levels:

1) **Synchronization per node** Each node simulator (i.e. *cGEM5*) 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 5.6 (upper). The user can define the minimum simulated time within which the two simulators can exchange Data Packets as illustrated in Table 5.6.
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 given 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 5.6 (lower). The *SynchNode* is a normal user-space instantiated node (as the rest of the HLA Enabled Nodes) inside OMNeT++ that

follows the standard Node structure and as a result it is 100% compatible with OMNeT++. The user can define the simulated time in which all COSSIM/ACSIM instances are fully synchronized periodically as illustrated in Table 5.6.

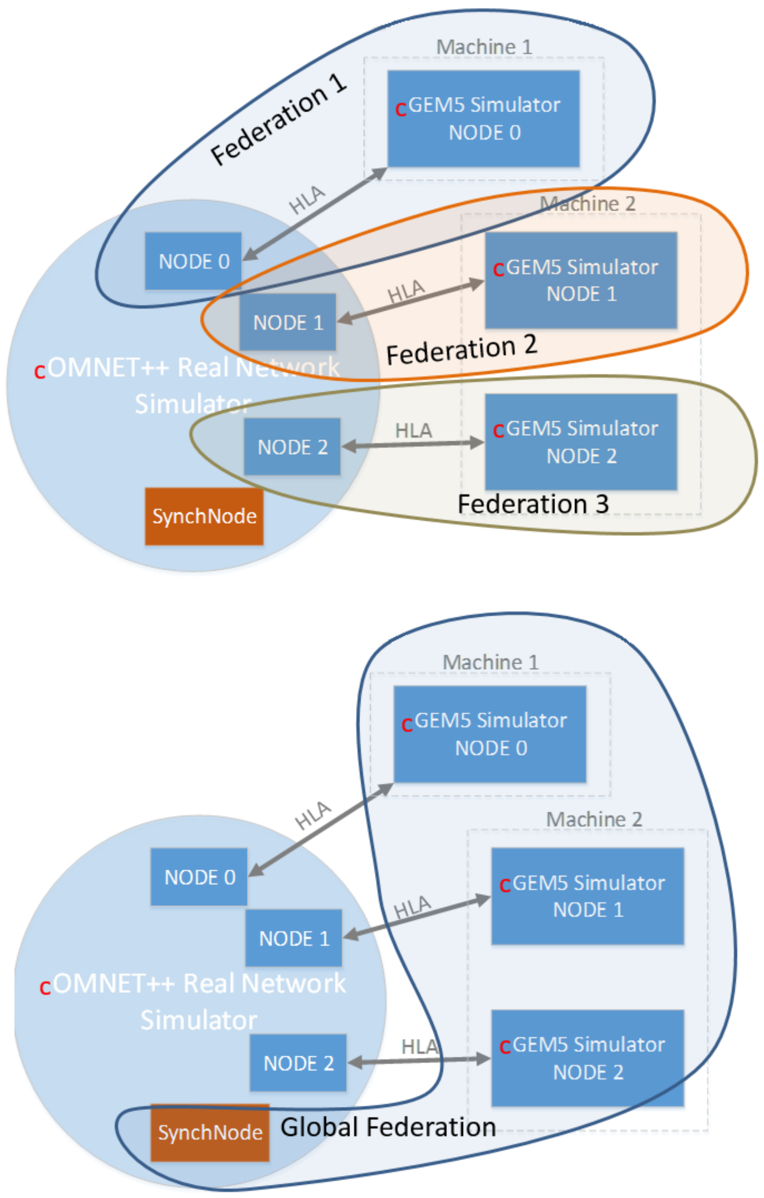


Figure 5.6. COSSIM HLA Federations

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 doesn't constrain 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 (i.e. in terms of clock cycles).

Table 5.6 describes the additional parameters that have to be defined in order to configure the communication between cgem5 and the network simulator as well as among different cgem5 instances. First of all, *--SynchTime* is the *Global Synchronization Time* which is a trade-off between simulation speed and simulation accuracy, while the *--RxPacketTime* (Synchronization per node) is mostly defined by the latency of the network interface and it doesn't constrain the simulation speed. Finally, *--nodeNum* and *--TotalNodes* are the number ID of simulated system and the total number of simulated systems in the network respectively. In all of the above parameters the simulated time is converted automatically to CPU ticks based on CPU frequency because each cgem5 system can simulate different types of CPUs with different clock cycles.

Parameter Name	Usage example	Description
SynchTime	--SynchTime=10ms	Simulated time which all COSSIM components are synchronized periodically
RxPacketTime	--RxPacketTime=2ms	The minimum simulated time which the *cgem5/cOMNET++* system can receive Packet
nodeNum	--nodeNum=0	The number ID of simulated system
TotalNodes	--TotalNodes=2	The total number of simulated systems

Table 5.6. COSSIM Synchronization Parameters

5.5 SynchServer Implementation

A multi-threaded *SynchServer* is implemented to ensure the correct order of *COSSIMlib* functions during **initialization phase** among different (cGEM5s and cOMNET++) instances. Specifically, a TCP Server (*SynchServer*) is implemented to reply in the different simulator instances. *Pthread* library and *pthread mutexes* are used so as to avoid to write two (or more) COSSIM/ACSIM nodes in the same storage element simultaneously (consistency). Both *RTIG* and *SynchServer* can be executed either on localhost or on remote physical machine to extract process parallelization. Figure 5.7 and Figure 5.8 illustrate the abstract view of *SynchServer* requests for synchronization per node and global synchronization respectively.

Specifically, for synchronization per node, two *SynchServer* structures are created to store the requests from c*OMNET++* (c*OMNETtocGEM5*) and c*GEM5* (c*GEM5tocOMNET*). Each of these structures contains *MAX_NODE*[16] elements to support requests from all COSSIM HLA enable nodes. Figure 5.7 illustrates a scenario for the 2nd simulated node. Initially, *Pre-initialize* methods are called from *cOMNET++* to define federation and federate characteristics. Subsequently, it updates the correct position in c*OMNETtocGEM5* structure (step 1), while at the same time cGEM5 reads the corresponding position (step 2). Whenever, c*GEM5* and c*OMNET++* complete the *Pre-initialize* and *Initialize* methods, c*GEM5* updates the c*GEM5tocOMNET* structure (step 3) as acknowledgment and c*OMNET++* waits to read this value (step 4). Finally, both c*GEM5* and c*OMNET++* call *synchronize* function which is used from the federate at the last state of initialization to declare that all other federates (in same federation) have been joined.

[16] MAX_NODE is the total number of COSSIM HLA enable nodes.

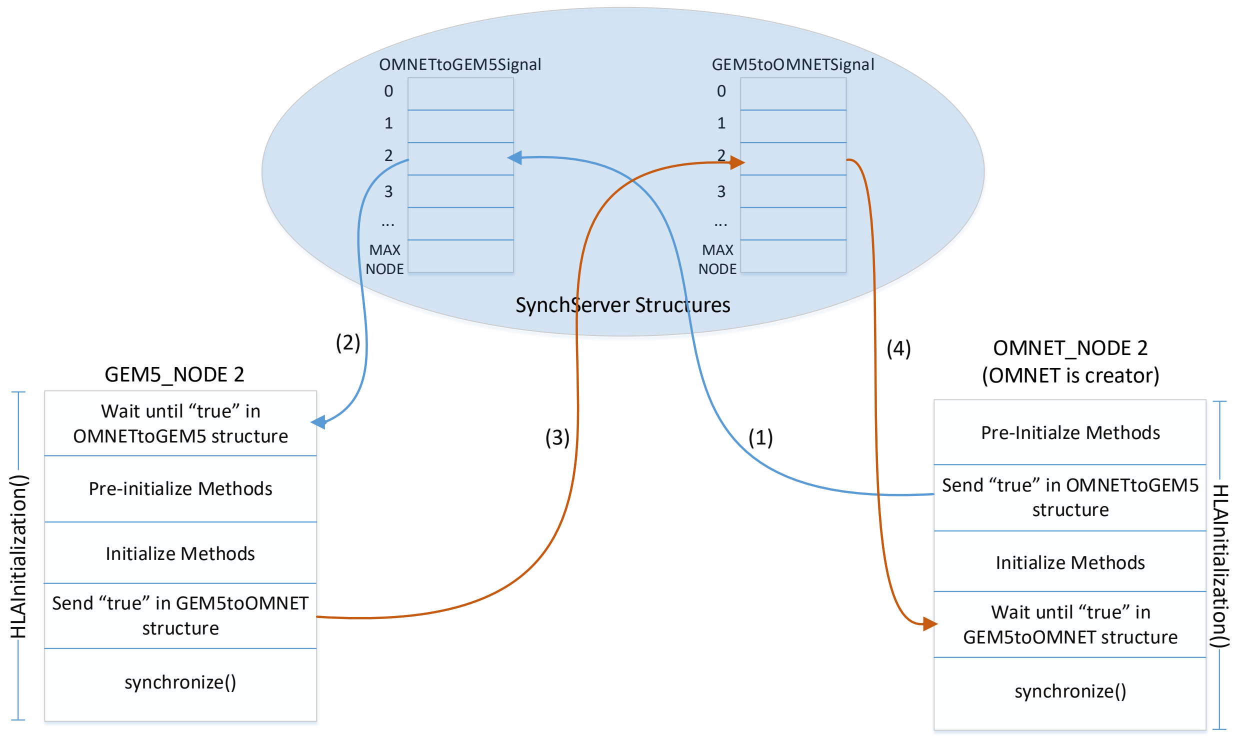


Figure 5.7. Abstract view of *SynchServer* requests for Synchronization per node

Finally, for Global Synchronization, one *SynchServer* structure is created to store the requests from c*GEM5* nodes and reply to c*OMNET++* during **initialization phase**. It contains *MAX_NODE* elements to support requests from all COSSIM HLA enabled nodes as well. In the Global Synchronization Federation, all cGEM5 nodes and one c*OMNET++* transparent helper node (*SynchNode*) have to be joined, as illustrated in Figure 5.8. Specifically, all c*GEM5* nodes call *Pre-initialize* & *Initialize* methods and subsequently they update the corresponding *GlobalSynchSignal* positions and call synchronize functions until all federates to be joined. c*OMNET++* (after *Pre-initialize* and *Initialize* methods) waits all federates to be joined and finally, it calls synchronize function. To be noticed that all **non-creator** federates are paused (through *synchronize* function) until synchronize function from **creator** federate is called. In *COSSIMlib*, c*OMNET++* is selected as **creator** and for this reason it waits for all federates to be joined before calling *synchronize()*.

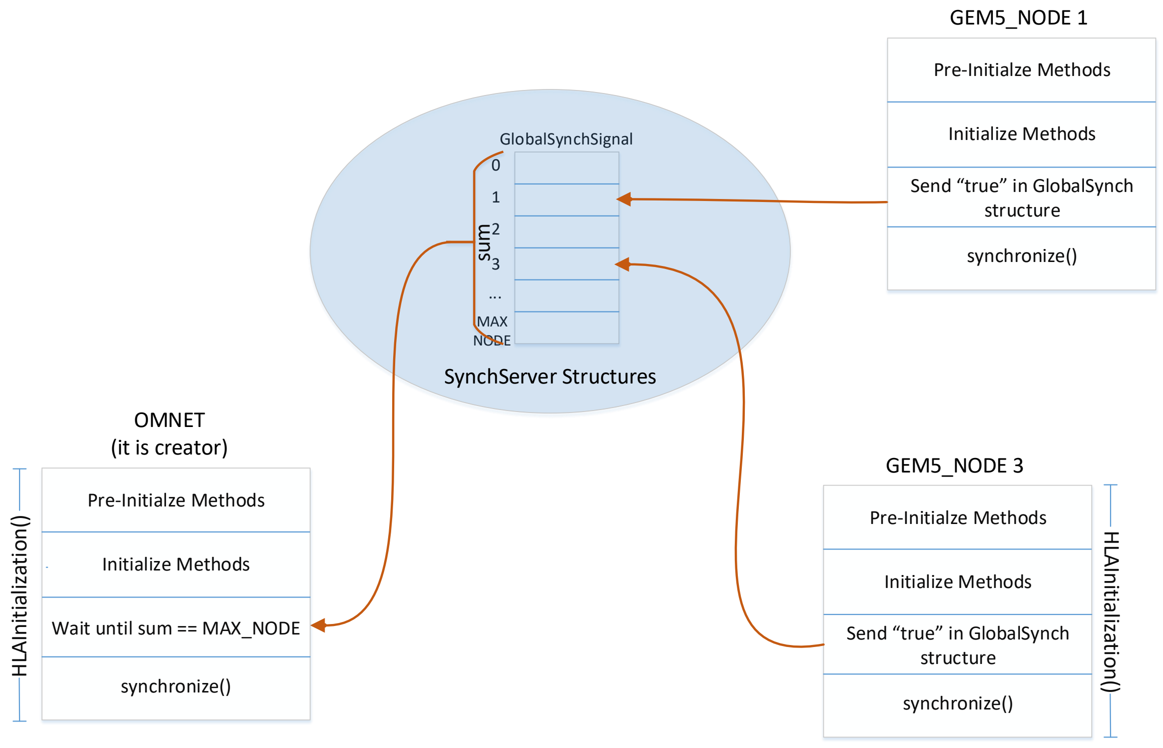


Figure 5.8. Abstract view of *SynchServer* requests for Global Synchronization

6 A novel way to efficiently incorporate Hardware Accelerators

This chapter presents a novel interconnection which is developed in the context of this thesis in order to expand the COSSIM simulator so as to support our novel SystemC accelerator and create ACSIM. Specifically, we introduce a novel flow that enables us to rapidly prototype synthesisable SystemC hardware accelerators in conjunction with cGEM5 simulator without worrying about communication and synchronisation issues. SystemC is selected because of its cycle accurate simulation features, while it is one of the most widely used input languages for the HLS tools. In addition, the official effort for SystemC definition and promotion known as Open SystemC Initiative (OSCI), now known as Accellera [76], provides an open-source proof-of-concept simulator while it has been approved by the IEEE Standards Association [77, pp. 1666–2011]. We use the full-system mode of the simulator so as to be able to simulate a complete system comprised of a number of devices and a full Operating System (OS). As a result, the application can be verified in a cycle-accurate manner via whole system simulation, including memory hierarchy, caches, peripherals, etc, with full operating system interaction (e.g. scheduler, drivers etc.), thus making the simulation more realistic/accurate.

Figure 6.1 presents an abstract view of the GEM5 full-system simulator when coupled with a full operating system. It consists of a central bus onto which several devices can be attached, DRAM memories, caches, CPUs, etc. In order for a new accelerator to be incorporated, the designer must ensure that is connected to the bus as well as that the appropriate OS drivers have been developed, as analytically described in the following sections.

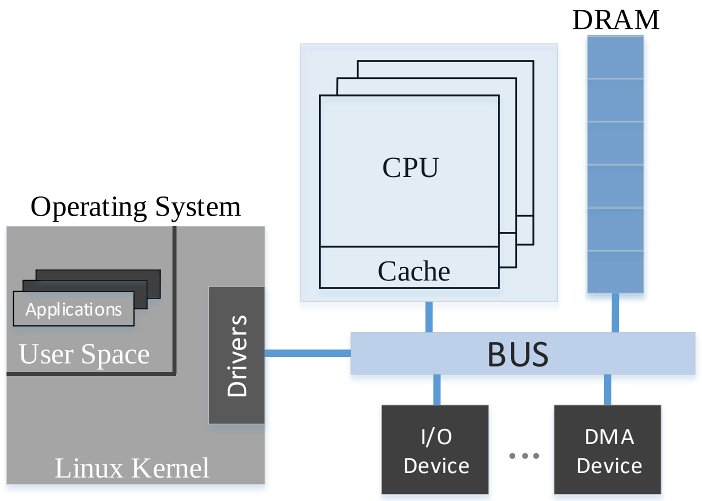


Figure 6.1. Full-system GEM5 Architecture

6.1 Architecture of ACSIM Hardware Accelerator

This section describes the most significant enhancements/modifications developed in order to expand the GEM5 full-system simulator functionality so as to support our novel SystemC accelerator. Figure 6.2 illustrates the integration of our SystemC simulator with the COSSIM; in particular the interconnection is between the SystemC simulator and cGEM5 and then there is an interconnection (which is described in Section 7.5.2) of cGEM5 with cOMNET++ which allows for the complete simulation of a heterogeneous parallel system. The following paragraphs describe the components in detail:

A. **Operating System (OS)** The OS can be represented as a layered structure, as in Figure 6.2. 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 our accelerator module, we have developed a set of device drivers. The accelerator is activated through programmed I/Os that provides 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 was 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 Host (Kernel Space) to Accelerator Wrapper.
  - *QUERY GET DATA* - Initialise the DMA copy transaction from Accelerator Wrapper to Host (Kernel Space).
  - *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 memcpy finish signal, etc.

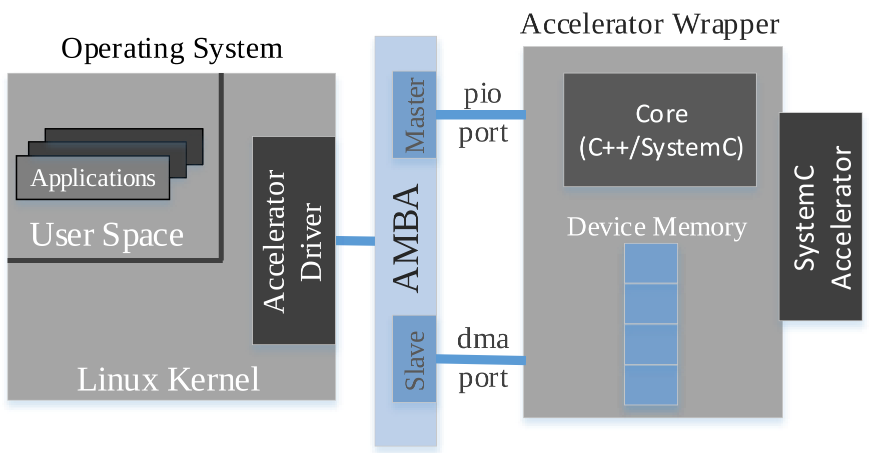


Figure 6.2. Integration of SystemC accelerator with cGEM5 (full-system mode)

B. **Memory Bus (Advanced Microcontroller Bus Architecture - AMBA)**

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 AcceleratorWrapper port in order to write/read large amounts of data to/from the accelerator device memory, as shown in Figure 6.2.

C. **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 memcpy, 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.

Subsequently, a core containing mixed C++ and SystemC code was implemented for the connection of the cGEM5 C++ functions and the accelerator's SystemC threads, as shown in Figure 6.3. Here we must note that the GEM5 and SystemC simulators run concurrently on different threads. Specifically, when the cGEM5 OS requests a SystemC accelerator call, a new thread emerges using the *pthread* library and starts the SystemC simulation by calling function *sc_start*. Consequently, at the end of the SystemC accelerator, the new thread is killed; if cGEM5 requests another SystemC call, a new thread will be created, etc.

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

1) ***Dynamic Memory Allocator C++ Module*** - The Buddy dynamic memory allocation algorithm scheme [104] is implemented in the Accelerator Wrapper to allocate and free Device Memory segments through the cGEM5 operating system, similar to the *cudaMalloc* of NVIDIA GPUs [105].

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 vise versa, similar to the *cudaMemcpy* of NVIDIA GPUs [105]. 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 accelerator[17], it

[17] This can happen when GEM5 has not any processing work to do and SystemC accelerator has to manipulate a lot of threads.

reschedules the synchronisation event function to wait for the SystemC accelerator.

4) ***InitSystemC 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*[18] 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) ***MemCpyToDevice/ToHostSystemC Threads*** - The Two *MemCpy SystemC* threads developed so as to pass the data from the Wrapper Device Memory to the corresponding synthesisable I/O ports, which depend upon the data type, such as *int*, *double*, etc., so that they eventually arrive at the *SystemC* accelerator. The user can define through one parameter the amount of data to be read/written per SystemC cycle.

[18]sc_pause command is selected instead of sc_stop because the latest destroys all SystemC structure and as a result, the same SystemC module cannot be called twice.

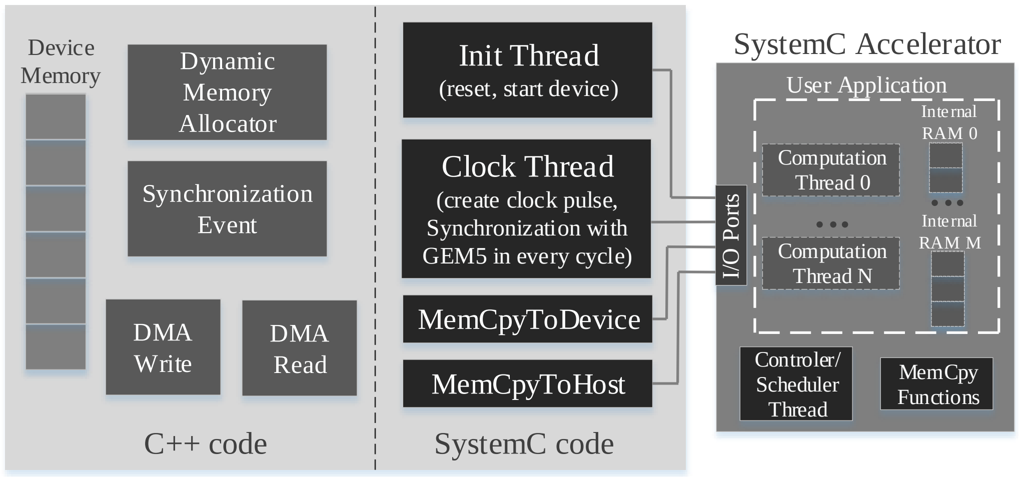


Figure 6.3. Accelerator Wrapper Device

D. **SystemC Accelerator**

In order to simulate the hardware accelerator, the *Accelera* open-source libraries have been incorporated with the cGEM5 *SCons* construction tool [106]; this allows for the compilation and execution of complete system applications running on top of COSSIM 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 synthesisable SystemC, such as the one supported by Xilinx' Vivado HLS or Mentor' Catapult, since the interconnection port modules utilized in the Accelerator Wrapper, are fully synthesizable as well, as illustrated in Figure 6.3.

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, as represented in Figure

6.3 by the dashed line threads. These threads can be scheduled by the Controller/Scheduler SystemC Thread.

- ***MemCpyToDevice/ToHostSystemC 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.

6.2 Accelerator Run-Time Environment

This section describes the interface of our integrated system, and how the users can develop the description of their accelerators in SystemC and their application in C/C++ from within the cGEM5 operating system.

Code segment 1 describes the header functions of the developed drivers library. The user 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.

Code Segment 1. Header of Accelerator Driver (AccelDriver.h)

```
typedef uint64_t DevMemAddr;

void AccelInitialization();
void AccelFinalization();

DevMemAddr AccelMalloc(size_t size, const char label);
void AccelFree(DevMemAddr SWAddr);

void AccelMemcpy(DevMemAddr SWAddr, void * data, size_t size, uint8_t TransferType);
void AccelCallDevice();
```

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 of 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 *DevMemAddr* type which has been returned by *AccelMalloc*.

Function *AccelMemcpy* has been implemented so as to allow for the efficient copying of the data from cGEM5's RAM 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 6.1. 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 6.1.

There are also the *MemCpyToDevice/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 6.1 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.

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

Table 6.1. Type of Elements supported by the custom SystemC application

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 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 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 }
```

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.

7

Validation and Performance Analysis

This chapter provides the experimental results of our work. Evaluation of the COSSIM simulation framework has been initially analysed using light versions of two Linux distributions; Gentoo Base System [107] for x86 processors and BusyBox [108] for ARM ones. Subsequently, full Ubuntu images are configured for both ARM and X86 architectures to evaluate complex real-world applications. In all systems the Linux-minimal package and the JRE7 are installed so as to enable execution of C, C++ and Java applications (thus resembling a realistic deployment scenario). Specifically, Section 7.1 and 7.2 present the evaluation of Processing and Network Simulator parts respectively; Section 7.3 examines the Performance analysis of COSSIM using multiple cores in multiple distributed machines. In addition, COSSIM bottleneck is evaluated examining both network and processing bound applications as well as multiple HLA Servers implementation. Moreover, COSSIM framework is evaluated simulating two real world applications consisting of complex CPS with very different requirements and characteristics as described in Section 7.4. The first case study is a Building Management System (BMS) with relatively simple CPS nodes and relatively low-speed interconnecting networks whereas the second one is a Mobile Visual Search (MVS) framework involving high-end CPS and HPC nodes interconnected with high speed network. Finally, Section 7.5 presents the evaluation and performance analysis of integration with Hardware Accelerators (i.e. ACSIM) using three financial applications.

7.1 Evaluation of Processing Simulator Part

COSSIM utilizes a modified version of the mainstream GEM5 simulator. However, all modifications are mainly related to the Ethernet interface and the developed HLA components. As such, the main computation engine of the GEM5 simulator is left intact and therefore the performance of the GEM5 component of the COSSIM simulator is in line with the reported performance of the publicly available version of the simulator. GEM5's performance varies greatly with the complexity of the system that is being simulated. As referred in the literature, a typical simple CPU (InOrder CPU) in GEM5 can be simulated at a rate of 1 to 3 MIPS (Million Instructions / Sec), while 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 [109], [46].

The key concept of our approach is to execute the cOMNET++ simulator in a typical workstation so as to easily utilize the GUI that facilitates the orchestration and visualization of the simulation, while the cGEM5 instances are run in one or more servers (distributed simulation). For this reason, in the following experiments, the cOMNET++ simulation was executed on a workstation based on an Intel i5-4590 processor (3.3GHz) running Ubuntu Linux 14.04 with 16GB of RAM (Machine 1). On the other hand, the simulation of the CPUs as well as the HLA Server were executed on two servers based on four Intel Xeon E5-2440v2 (8 physical cores/Xeon) running CentOS 7.2 (totally 32 physical cores with 128GB of RAM - Machines 2&3). The main configuration of the simulated processors is described in Table 7.1.

CPU Type	CPU/System Clock	Memory	L1I/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

Table 7.1. The main configuration of the processors simulated

Figure 7.1 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. Those numbers do not include any communication with the Accelerator Simulator. Our results demonstrate 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 2-32 node experiments, while there is a steep performance drop when more than 32 cGEM5 instances are executed on the 32 physical cores due to lack of processing resources. For the specific experiment, the synchronization interval is set at 10ms. Consequently, after every 10ms 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. A short synchronization interval will yield the most accurate results at the cost of lower overall performance, as each simulation instance will be forced to halt very frequently and the time required for synchronizations will become significant compared to the actual computation time. A more relaxed (i.e. longer) synchronization interval will result in higher speeds, with a potential loss of accuracy, as interactions between nodes that occur within this interval are not properly handled.

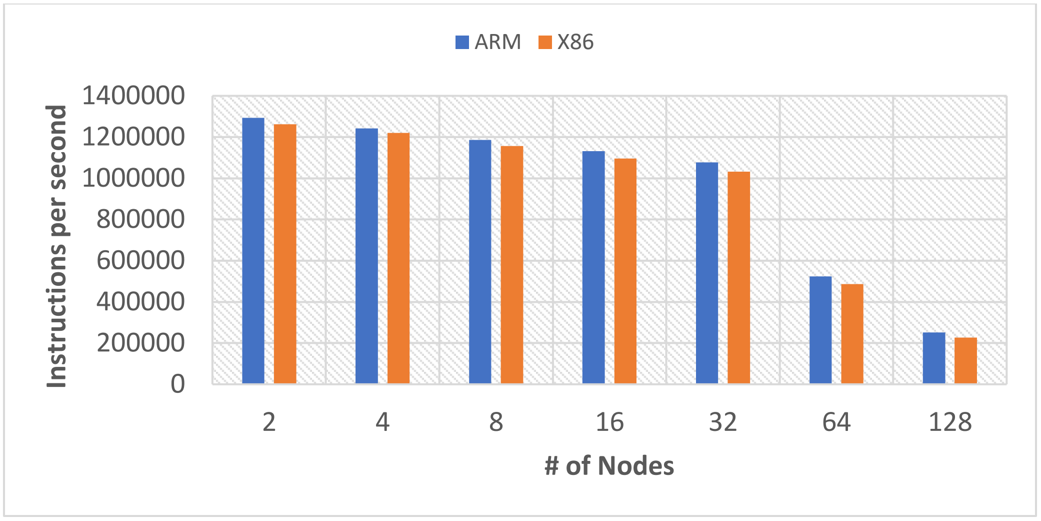


Figure 7.1. Performance results using typical GEM5 configuration

Figure 7.2 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 (all nodes are configured as in Table 7.1 and executed on the above system). There is a dramatic drop in performance for very short interval times (smaller than 100us) mainly due to the increased number of messages that OMNET++ has to manipulate.

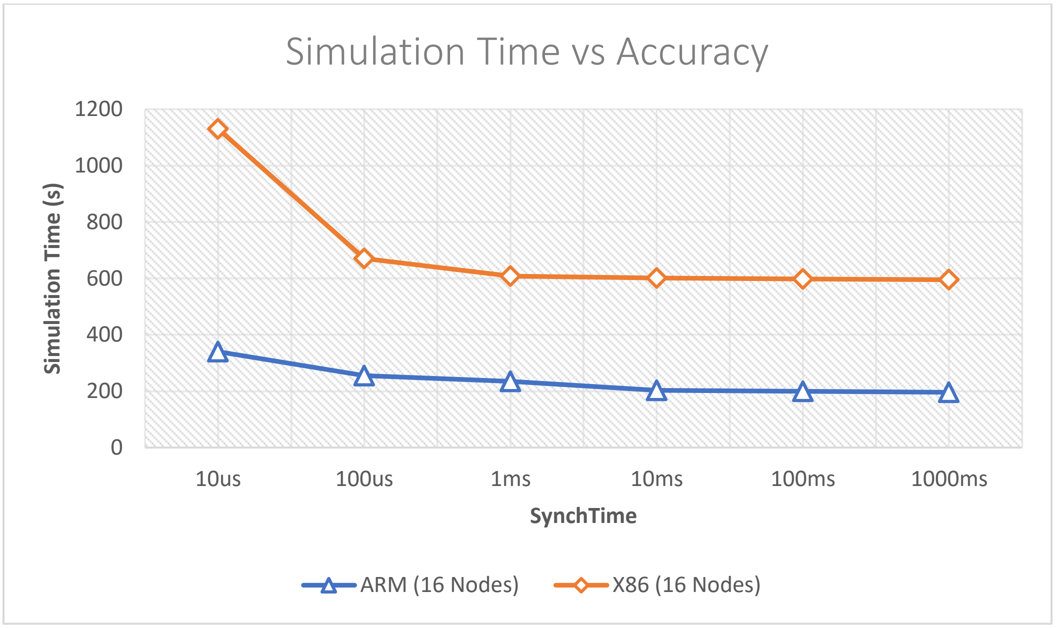


Figure 7.2. Simulation time using different synchronization intervals

7.2 Evaluation of Network Simulator Part

The network performance of COSSIM's novel approach has been evaluated using the widely used Netperf 2.7.0 benchmark suite and specifically the very demanding TCP_STREAM & TCP_RR benchmarks. Netperf [110] 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.

The suite is designed around a basic client-server model which consists of the netserver and netperf executables. When netperf is executed it establishes a "control connection" to the remote system. This connection is used in order to pass the test configuration information and results to and from the remote netserver. Regardless of the type of the test that is executed, the control connection is a TCP one using BSD [111] sockets. The control connection can use either IPv4 or IPv6. After the configuration information has been transmitted, a separate "data" connection is opened which can utilize any network API and/or protocol. When the test is completed, the data connection is terminated and the results from the netserver are passed-back, via the control connection, to netperf and together netperf's results they are displayed to the user.

7.2.1 TCP_STREAM Benchmark

The *TCP_STREAM* test is the most widely-used test in netperf. It transfers certain data from the system where netperf is executed to that running netserver. Figure 7.3 illustrates the throughput (Mbits/sec) achieved on this benchmark in the case of two COSSIM simulated x86 systems and two real x86 systems; in both cases the simulated and real systems are connected through Gigabit Ethernet. Specifically, 5MB[19] of total data are exchanged in each experiment, while different buffer sizes are utilized in the "send" calls of the test. Command 1 shows the netperf TCP_STREAM configuration which is used for 100bytes buffer size, while Table 7.2 summarizes the parameters.

```
netperf -H IP -t TCP_STREAM-c -C -l -5000k -- -m 100     (1)
```

[19] 5MB was selected as adequate quantity of data to reach the maximum speed of experiment.

Parameter	**Description**
-m [bytes]	Set the size of the buffer passed-in to the "send" calls of a STREAM test (in bytes)
-H [IP]	Set the name of the remote system
-l [testlen]	Controls the length of the requested test
-c	Requests CPU Utilization for the local system
-C	Requests CPU Utilization for the remote system

Table 7.2. TCP_STREAM benchmark Parameters

As described in Section 5.4, COSSIM/ACSIM Synchronization is based on two different entities that can be separately defined by the user; for the *TCP_STREAM* experiments, the synchronization per node interval is set at 10us (i.e. each node guarantees that can receive at most one Data Packet every 10us). This interval is selected due to the Gigabit Ethernet interconnection between the nodes. Specifically, since the maximum Ethernet packet is 1536bytes [112], by selecting a synchronization period of 10us, each node can receive up to 1536bytes/10us or 153.600.000bytes/sec or 1.228.800.000bits/sec so it can support full Gigabit Ethernet speed. On the other hand, three different intervals have been tested for the Global Synchronization, from 100us to 10ms. Figure 7.3 demonstrates that as the Global Synchronization decreases, more accurate results are obtained (i.e. much closer to the real systems ones), while using 100us as the Global Synchronization interval, triggers extremely accurate results for all buffer sizes. Furthermore, as the number of buffer messages increases the throughput moves toward the maximum possible speed.

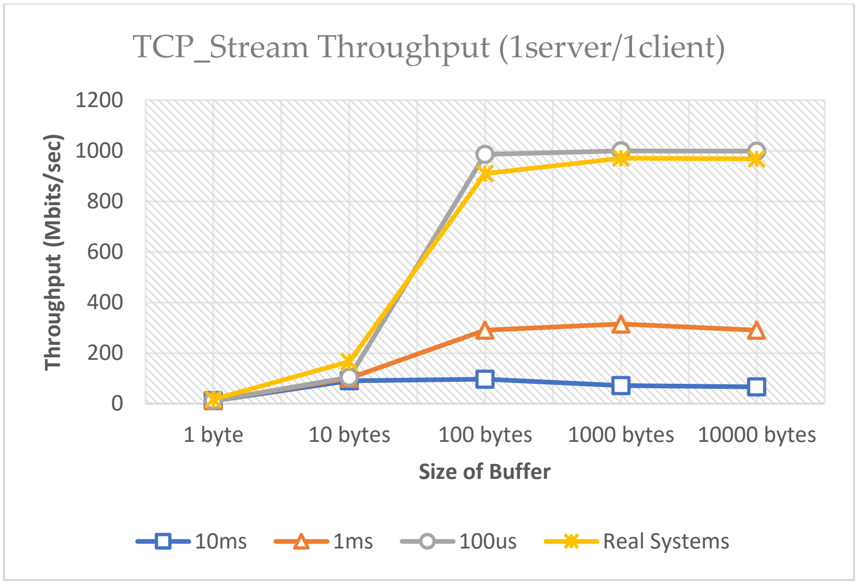


Figure 7.3. TCP_Stream throughput for 1server/1client using different Global Synchronization intervals

CPU utilization is an important, and very often overlooked component of the overall networking performance. Unfortunately, it can be one of the most difficult metrics to measure accurately and portably. Netperf is one of the most accurate benchmarks for measuring CPU utilization [110]. CPU utilization in netperf is reported as a value between 0 and 100% regardless of the number of CPUs involved. In addition to CPU utilization, netperf reports a metric called "service demand". The service demand is the normalization of CPU utilization in terms of the work performed; for a *TCP_STREAM* test it is the microseconds of CPU time consumed to transfer on 1024 Bytes of data.

Figure 7.4 illustrates the CPU utilization (in the simulated node) for 1-server/1-client and 1-server/2-clients experiments using a Global Synchronization interval of 100us, which is adequate for full speed simulation, and Synchronization per node 10us. Figure 7.4 demonstrates that as the buffer-size is less than 10bytes, the CPU utilization is steadily at 100% for both 1 client and 2 clients experiments (i.e. they are CPU-bound). This is reasonable because the clients need to truncate the 5MBs in a lot of small packets, while the throughput achieved is much lower than the maximum one (Figure 7.3). In addition, as the buffer size gets greater to 1000bytes, the CPU utilization is much lower and the experiment becomes network-bound. The 2-client experiment needs less CPU processing in each client CPU because the packets are sent at lower rates while the server CPU utilization is similar to that of the 1-client with smalled differences due to the higher number of total packets the server gets.

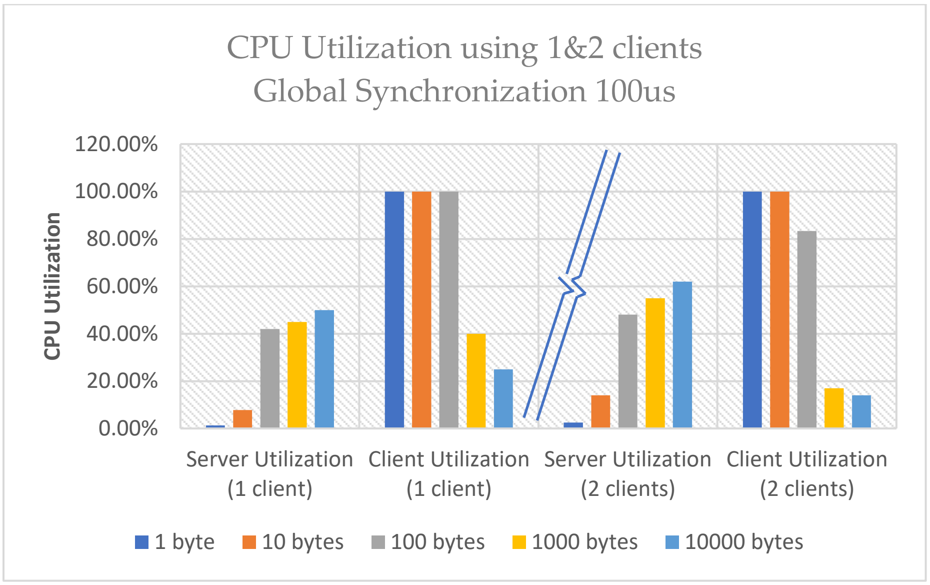


Figure 7.4. CPU Utilization using TCP_Stream for 1server/1client & 1server/2clients experiments

Figure 7.5 demonstrates the throughput for the *TCP_STREAM* experiments using 1 server and 1 up to 32[20] clients. In that configuration a router has been configured with 2-33 Gigabit Ethernet ports to establish a star network topology. Figure 7.5 demonstrates that as the number of clients increases, the throughput decreases because the server has to serve all clients. The maximum throughput achieved by our simulator using 1000bytes in each Ethernet packet is ranging from 998Mbits/sec to 69 Mbits/sec for 1 and 32 clients respectively.

[20] COSSIM tool is tested up to 32 clients, due to physical cores bound; the aim of this experiment was to take measurements assigning one gem5 instance per core to not suffer from performance degradation due to physical cores limit.

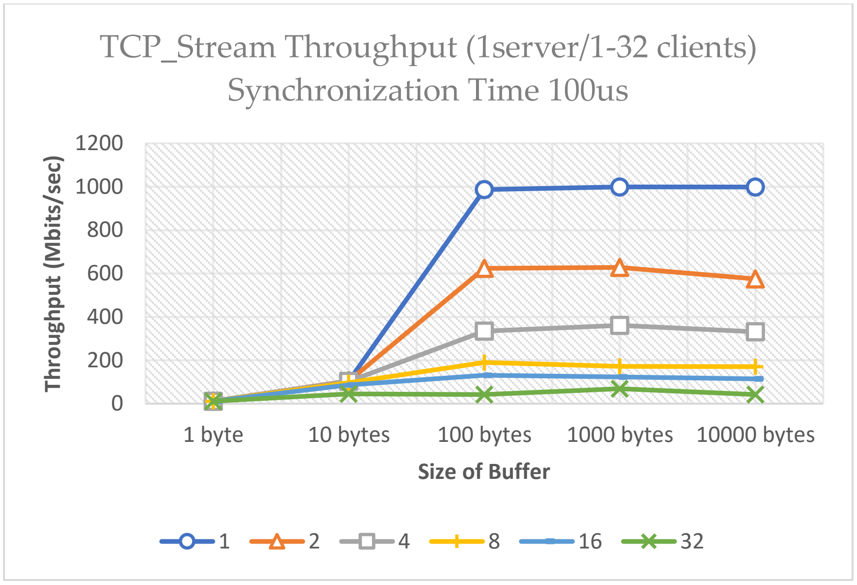


Figure 7.5. TCP_STREAM Throughput using 1server & 1-32 clients

7.2.2 TCP_RR Benchmark

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) [113] 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 7.6 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[21] second of total time. Command 2 shows the netperf TCP_RR configuration which is used for a request/response packet size of 100 bytes, while Table 7.3 summarizes all the parameters of our experiment.

netperf -H IP -t TCP_RR -l 1 -- -r 100,100 (2)

[21] 1second was selected as adequate quantity of time to reach the maximum speed of experiment.

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

Table 7.3. TCP_RR benchmark Parameters

In all the *TCP_RR* experiments, the synchronization per node interval is set at 10us so as to achieve full Gigabit Ethernet rates (as described in the previous paragraph) between the nodes. On the other hand, three different intervals are selected for Global Synchronization from 10us to 1ms. Figure 7.6 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. *TCP_RR* requires smaller Global Synchronization interval (10us) than the *TCP_STREAM* experiment (100us) in order to obtain very accurate results due to smaller length of each packet in *TPC_RR* (i.e. higher number of packets are sent per second). Finally, Table 7.4 summarizes the transmission rate of the above systems; the highest transmission rate for the simulated system is achieved using 1byte request/response packets and a 10us Global Synchronization interval (99,9 microseconds/ packet (Simulated nodes) instead of 98,3 microseconds/packet (Real-nodes)).

Summarizing, both netperf experiments demonstrate that with a relatively low Global Synchronization Interval COSSIM/ACSIM can extremely accurately simulate both network-bound and CPU-bound applications.

	1ms	100us	10us	Real Systems
1 byte	1883.24 us/pkt	395.88 us/pkt	99.90 us/pkt	98.32 us/pkt
10 bytes	1757.47 us/pkt	395.90 us/pkt	99.94 us/pkt	99.60 us/pkt
10^2 bytes	1972.38 us/pkt	396.98 us/pkt	101.83 us/pkt	102.67 us/pkt

10^3 bytes	2012.78 us/pkt	434.78 us/pkt	163.93 us/pkt	151.74 us/pkt
10^4 bytes	6802.72 us/pkt	801.28 us/pkt	431.03 us/pkt	331.89 us/pkt

Table 7.4. Transmission rate of TCP_RR using different Global Synchronization

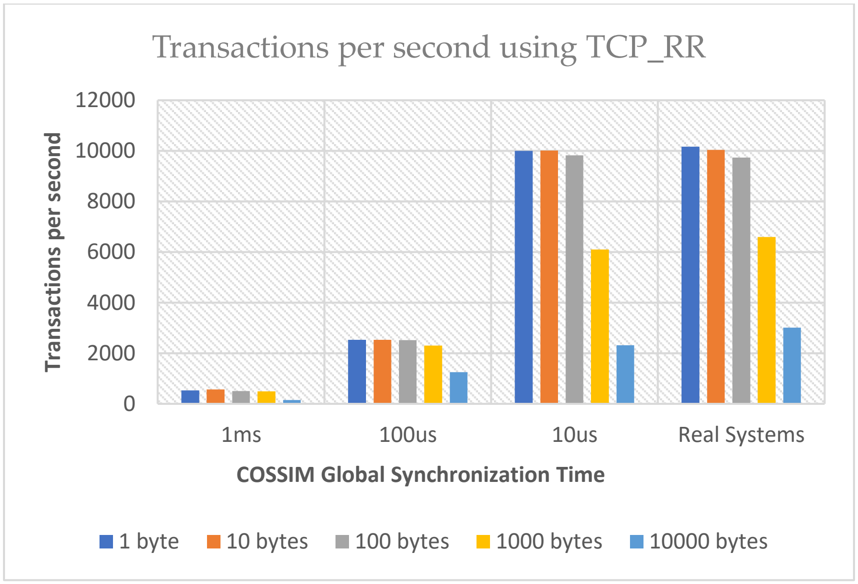


Figure 7.6. Transactions per second using 1server/1client with different Global Synchronization intervals

7.3 Performance Evaluation of COSSIM

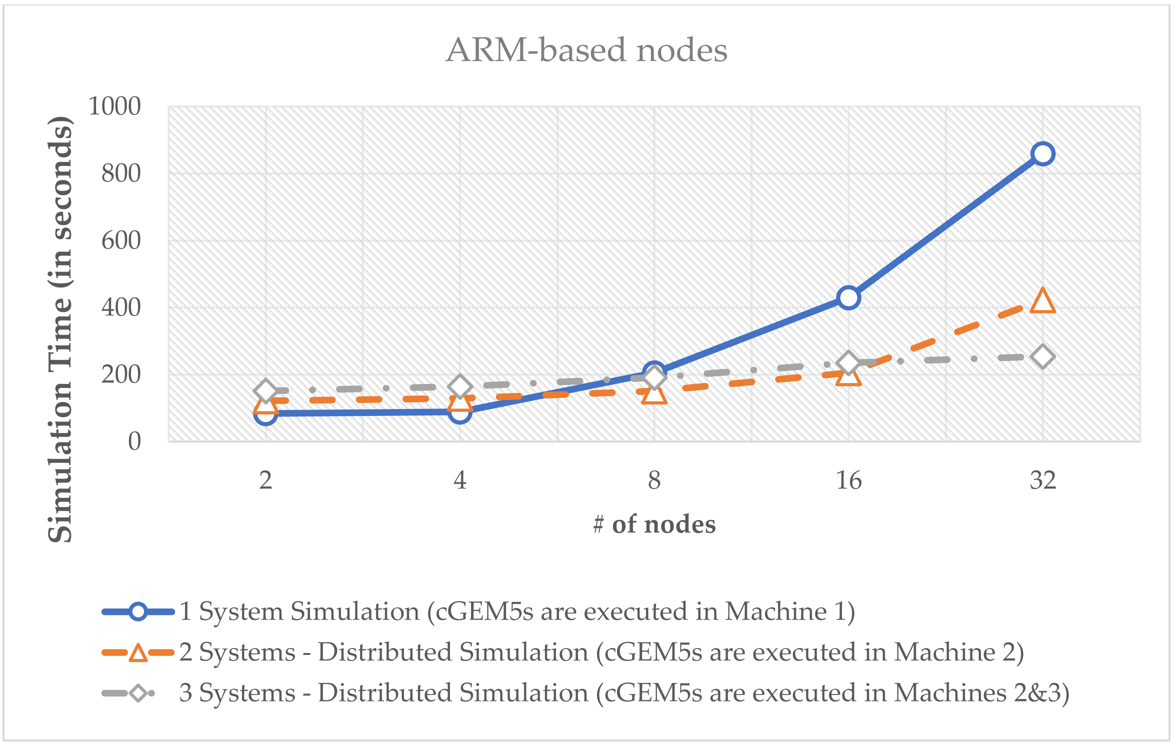
In this Section, the performance of the COSSIM simulator is presented using multiple cores in multiple distributed machines. In addition, the scalability of the COSSIM simulator is presented using multiple cores in a *cluster* of multiple distributed machines. Finally, an analysis is performed about COSSIM bottleneck as well as the speedup achieved through multiple HLA Servers.

7.3.1 Performance Evaluation of Distributed COSSIM

The performance of parallelism COSSIM/ACSIM simulation framework has been analyzed using the distributed scenario as described in Section 7.1 (3 Systems - one workstation and two servers). Specifically, Figure 7.7 demonstrates the performance of the simulator when the main COSSIM components (OMNET++, HLA servers and all GEM5 instances) are executed on the same physical machine (machine 1 - straight line) and when they are executed in the distributed scenario (machine 2 and machines 2&3 - dashed lines). Figure 7.7a illustrates the simulation time required to boot the OSs with their network card configuration for 2-32 ARM-based nodes, while Figure 7.7b for 2-32 X86-based nodes.

As it can be seen from the Figure 7.7, it becomes evident that performance is heavily impacted when the number of GEM5 instances spawned becomes higher than the number of physical processor cores present in the simulation machine (there is a steep performance drop when more than 4 cGEM5 instances are executed on the 4-core machine and more than 16 cGEM5 instances are executed on one of two servers).

On the other hand, the effect of the network has a negligible impact on the overall performance. Specifically, in 2&4 nodes experiments, Machine 1 is faster than distributed scenario due to faster single thread execution on Intel i5-4590 than E5-2440v2. However, when the number of cGEM5 instances increases, 2 System distributed scenario is actually faster comparing with the other two simulation experiments up to 16 cGEM5 instances (1 System simulation & 3 Systems simulation). This happens because there is slightly performance degradation in 3 Systems simulation due to 3 different physical machines instead of 2. However, in 32 nodes experiment, 3 Systems simulation surpasses the 2 System simulation (70% approximately faster). The similar results are obtained in X86-based nodes which requires more processing time to boot the OS (Figure 7.7b). Summarizing, in case of 3 Systems distributed simulation, the simulation time is almost constant for all simulated nodes (each cGEM5 node is executed per physical core), while in 32 cGEM5 instances, COSSIM distributed scenario can achieve up to 337% and 423% speedup for ARM and X86 based architecture respectively.



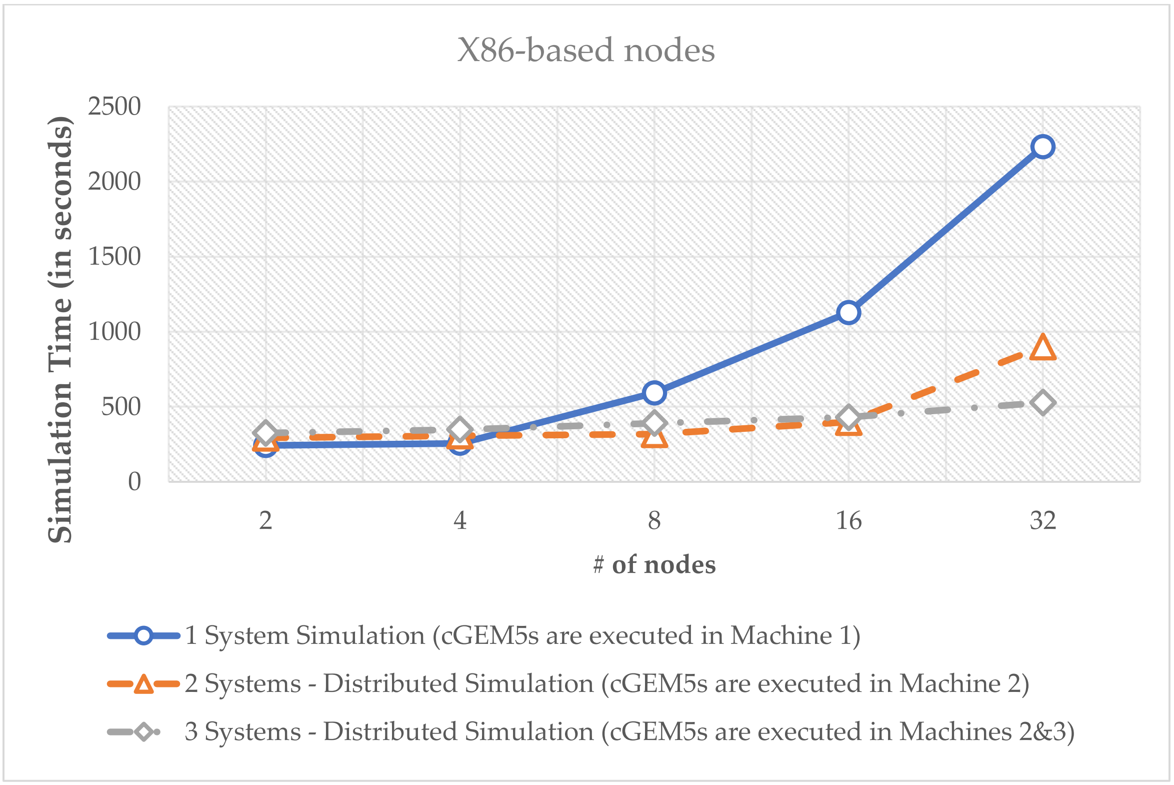


Figure 7.7. COSSIM simulation time using Distributed Machines ((a) Upper: ARM-based nodes (b) lower: X86-based nodes)

7.3.2 COSSIM Scalability

In this section, the scalability of the COSSIM simulator is presented using multiple cores in a *cluster* of multiple distributed machines. Specifically, a *42U HP RACK 10000 G2* Series cabinet 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 in the above cluster).

Specifically, Figure 7.9 demonstrates the performance of the simulator when the network part of COSSIM (cOMNET++) is executed in typical workstation based on an Intel i5-4590 processor (3.3GHz) running Ubuntu Linux 14.04 with 16GB of RAM, while the processing parts (cGEM5 instances) and HLA server are executed in the cluster which is described above running Ubuntu Linux 16.04. Figure 7.9 illustrates the simulation time required to boot the OSs with their network card configuration for 16-1024 ARM-based nodes and X86-based nodes. The network configuration in cOMNET++ are configured automatically using scripts and it contains both micro-routers and switches. Figure 7.8 illustrates a sample configuration for 8 nodes using *Ethernet* from nodes to micro-routers (through switch) and *Wireless* among micro-routers.

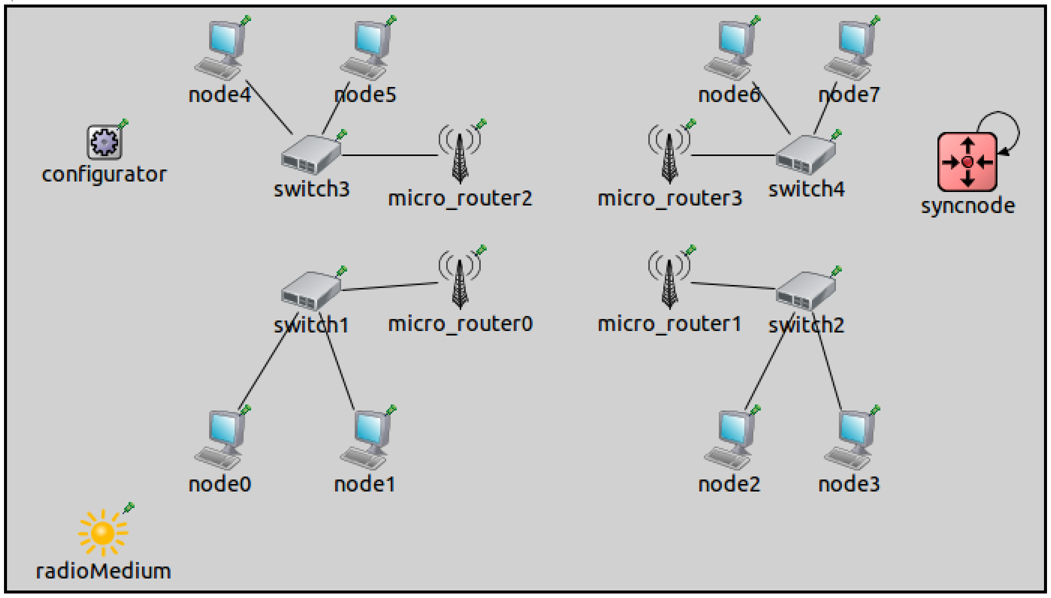


Figure 7.8. Typical network configuration for 8 simulated nodes

As it can be seen from the Figure 7.9, it becomes evident that performance is heavily impacted when the number of cGEM5 instances spawned becomes higher than the number of physical processor cores present in the simulation machine (there is a steep performance drop when more than 176 cGEM5 instances are executed on the 176-core cluster).

Specifically, in 16-128 nodes experiments, the simulation time is almost constant, while doubling the number of cGEM5s instances the simulation time increases by factor of two due to physical cores limitation. The similar results are obtained in X86-based nodes which requires 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 of cGEM5 instances is lower or equal to the number of physical cores.

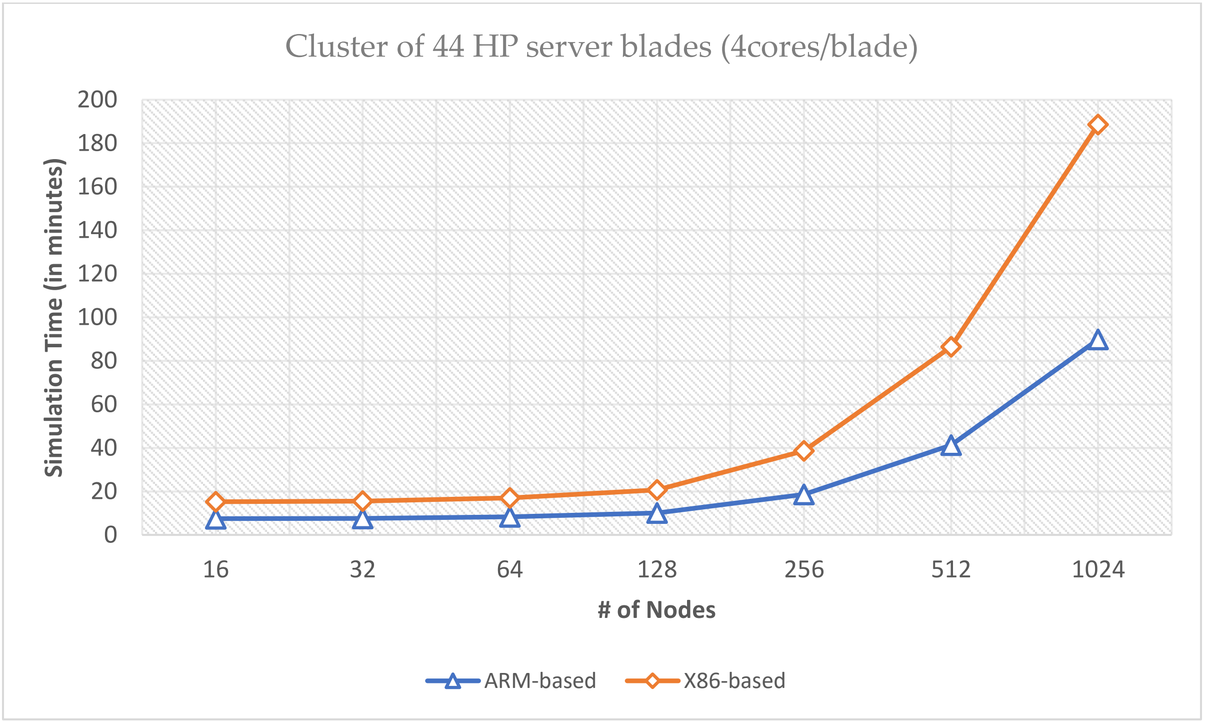


Figure 7.9. COSSIM simulation time using a cluster of 44 HP server blades

7.3.3 COSSIM Bottleneck Analysis

This section presents the bottleneck analysis of COSSIM using *TCP_STREAM* netperf benchmark. Figure 7.10 demonstrates the Simulation time for *TCP_STREAM* experiments using 1 server and 1 up to 32[22] clients. To be noticed that one router has been configured with

[22] COSSIM tool is tested up to 32 clients, due to physical cores bound; the aim of this experiment was to take measurements assigning one gem5 instance per core to not suffer from performance degradation due to physical cores bound.

2-33 Gigabit Ethernet ports to establish a star network topology. Figure 7.10 illustrates that, as the number of clients increases, the simulation time of COSSIM increases by factor of 2 because the cOMNET++ needs to manipulate double Data Packets and double Global Synchronization points in each experiment. This happens because TCP_STREAM is a network-bound application, however, in a common application which uses more the processing unit, the bottleneck is created in the cGEM5 side (Section 7.1). This is evident from the similar simulation time using different Global Synchronization intervals (the time required for the Synchronization is quite less than the time required for the network messages).

Exploring with more insight the simulation time increasing, we end up to measure the cOMNET++ performance behavior (cGEM5 not causes bottleneck; each GEM5 instance is executed per physical core). Figure 7.11 illustrates the CPU and Memory Utilization of cOMNET++ using *TCP_STREAM* benchmark for 16-32 nodes. It is evident that, OMNET++ utilizes the same memory for all Global Synchronization Intervals (this is reasonable because exactly the same number of packets are exchanged in total), while it requires 1/3 more memory doubling the number of simulated nodes. On the other hand, CPU Utilization increases only 4%, increasing the Global Synchronization interval one order of magnitude, while there is slight different from 16 nodes to 32 nodes. It is evident that doubling the simulated nodes, OMNET++ requires more time waiting for all GEM5 nodes to “reach” in the specific synchronization barrier as illustrated in Figure 7.12. As a result, this is a non-CPU intensive process since the CPU is in idle mode waiting for all nodes to “reach” in the synchronization barrier.

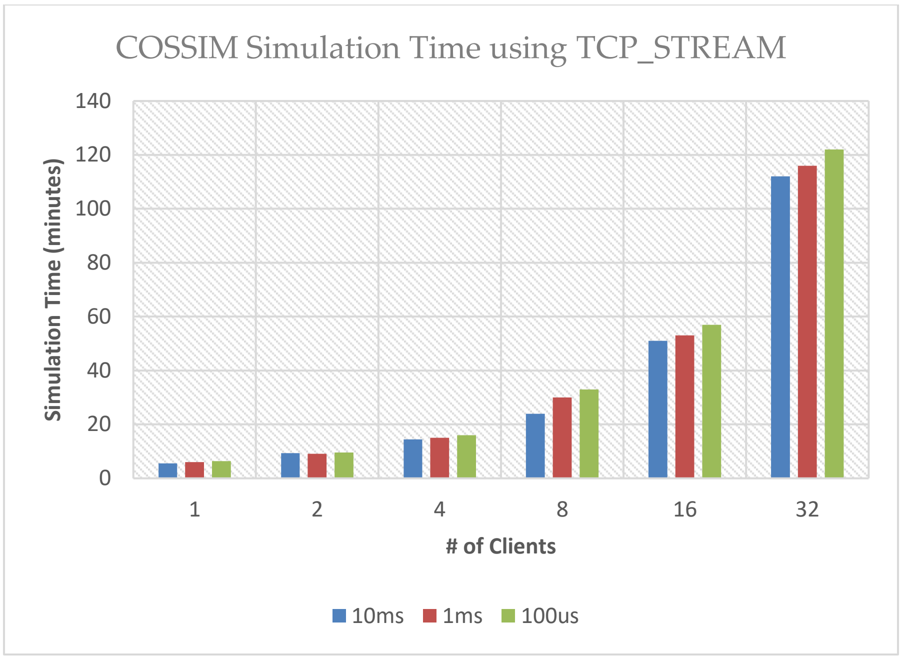


Figure 7.10. COSSIM Simulation Time using different Global Synchronization intervals (TCP_STREAM)

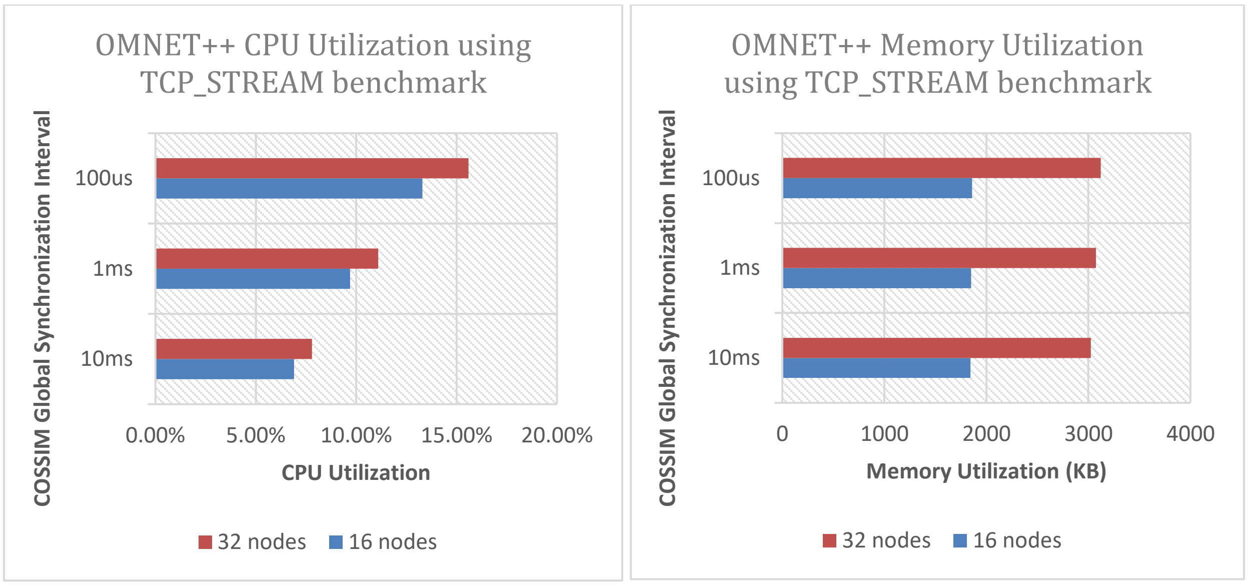


Figure 7.11. OMNET++ CPU and Memory Utilization using different Global Synchronization intervals (TCP_STREAM)

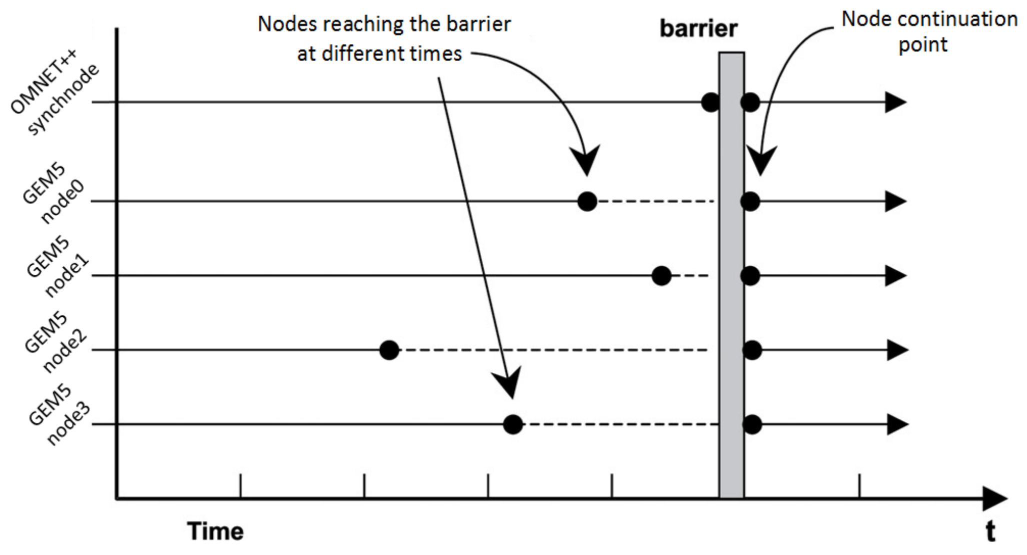


Figure 7.12. Example of COSSIM Synchronization Barrier using 4 nodes

7.3.4 Evaluation of COSSIM using multiple HLA Servers

TCP_RR benchmark is used to evaluate the HLA Server performance due to huge number of messages, and the short Global Synchronization Intervals. Specifically, 1byte request/response packets are used for 1 second of total time as described in the following command (10us Global Synchronization Interval is used).

netperf -H IP -t TCP_RR -l 1 -- -r 1,1 (3)

Figure 7.14a illustrates the CPU utilization of HLA Server (using 1 Global HLA Server for both Global Synchronization & Synchronization per node). The CPU Utilization of HLA Server increases by factor of 2 and 4 doubling the number of simulated nodes due to small Global synchronization intervals and the huge number of packets. For this reason, we decentralize the HLA Server using 2 identical HLA Servers in two scenarios.

In first scenario, 1 HLA Server is used for Global Synchronization (blue federation), and the other for Synchronization per node (orange federations) as described in Figure 7.13 left. In this scenario, the CPU Utilization concentrates on the 1st HLA Server due to small Global Synchronization intervals, while the 2nd HLA Server is under-utilized (Figure 7.14a right).

On the other hand, in second scenario, 1 HLA Server is used to serve 50% of the simulated nodes (both Global & per node Synchronization), and another one to serve the remainder 50% of simulated nodes (Figure 7.13 right). In this case, the CPU Utilization drops dramatically in a quarter, and specifically, it is similar with the nodes/2 CPU utilization of 1 Global HLA Server (Figure 7.14b right). Using this scenario, COSSIM is able to decentralize the HLA bottleneck (if needed) supporting two or more HLA Servers.

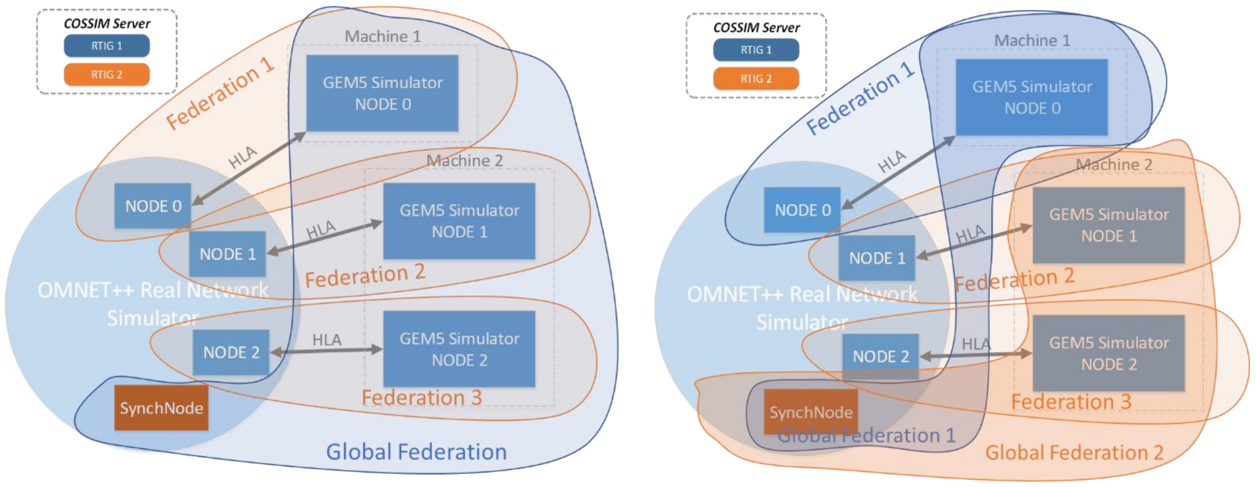
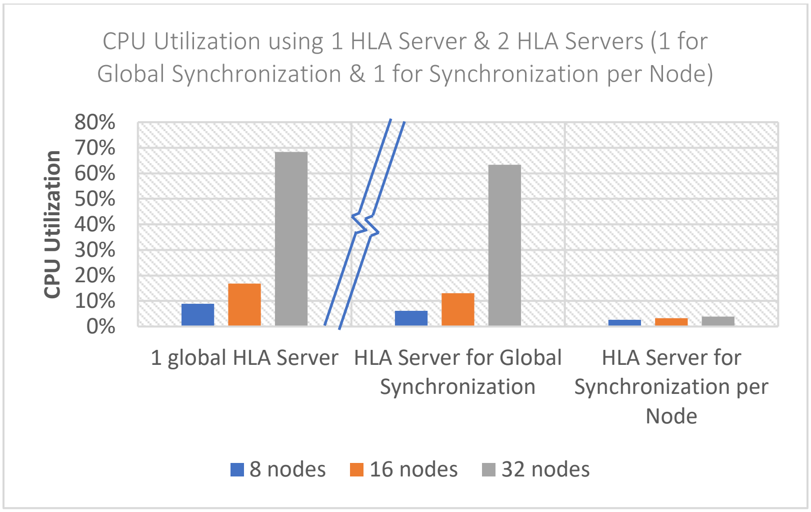


Figure 7.13. COSSIM HLA Federations using 2 HLA Servers ((a) left: 1 HLA Server is used for Global Synchronization & 1 HLA Server is used for Synchronization per node; (b) right: 1 HLA is used for 50% of nodes & 1 HLA Server is used for another 50% of nodes)



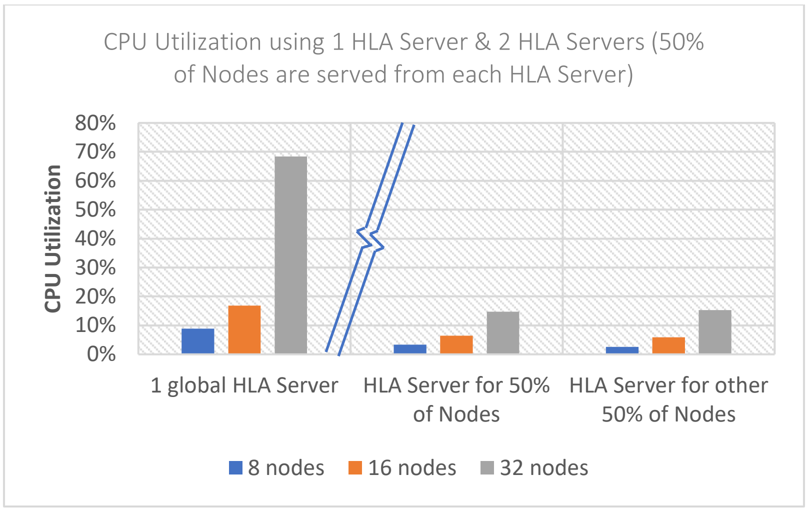


Figure 7.14. CPU Utilization using 1 HLA & 2 HLA Servers ((a) upper: 1 HLA Server for Global Synch & another one for Synch per node; (b) lower: 1 HLA per 50% of nodes)

7.4 COSSIM evaluation using two Real-world Use Cases

This Section presents two complex real-world applications which are used to evaluate the COSSIM framework; Building Management System and Mobile Visual Search developed by Tecnalia [114] and STM [115] respectively. To compare the performance of the COSSIM simulator as applied to the BMS and MVS applications, three different kind of times were defined:

- **Native time** the time recorded in the real native system while executing the native application.
- **Simulated time** the time recorded inside the simulator while executing the simulation.
- **Simulation (Host) time** the time measured outside of the simulator by the host machine while executing the simulation.

7.4.1 BMS (Building Management System)

The environment selected to run the Building Management System (BMS) use case is based on TECNALIA's KUBIK building, which consists of modular and removable structures that allow the installation and monitoring of a wide range of structural elements, devices and energy efficiency control systems. It is divided into three 100 m2 dedicated floors and a larger cellar where all the HVAC (Heat Ventilation and Air Conditioning) equipment is installed.

The main purpose of the BMS deployed in KUBIK is to ensure the boundary conditions for each of the tests running in the "testing cells" or measurement rooms, which involves a single apartment or a specific area in the building. In brief, the BMS could be described as the platform that delivers the appropriate set-points (energy consumption, air flow, temperature and others) to the available HVAC equipment, taking into consideration the boundary conditions of each of the measurement room. The system identified as the most suitable and representative target for the test case in KUBIK building is the Unit For Optimization (UFO) that evaluates the heating/cooling and lighting conditions in order to deliver optimized set-points to the BMS.

The UFO is composed of the Intelligent Control System (ICS), the Building Management Communication Node (BMCN) and an additional set of Remote Sensing Nodes that collect environmental data. The ICS module performs the overall building energy consumption forecasting in the long-term (several days), while the BMCN performs a simplified local forecasting in the short-term (next hours) involving a measurement room or small area. Besides, the BMCN also functions as a gateway for the ICS, enabling the communication with the Remote Sensor Nodes. The Remote Sensing Nodes collect data needed by the forecasting processes. Figure 7.15 illustrates the BMS layout.

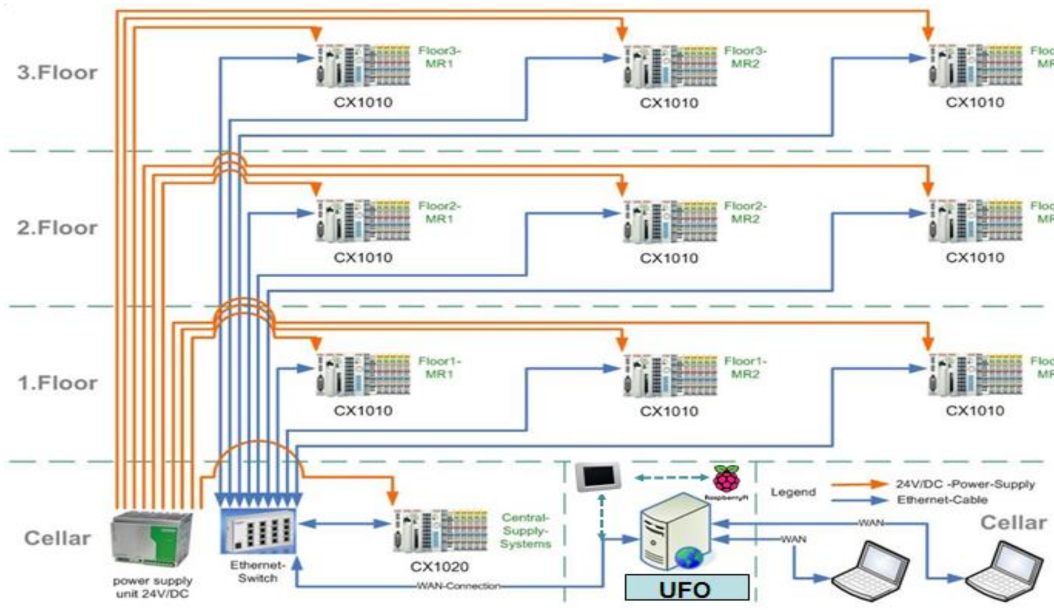


Figure 7.15. KUBIK's BMS + UFO layout

7.4.1.1 Description of BMS simulated scenario

Two use case scenarios are configured in order to evaluate the processing and network performance of the BMS application using the COSSIM simulation framework. As summarized in Table 7.5, simple and complex sensor network is used, while both regular and burst traffic situations were tested. In regular traffic, each node sends one message every 15 seconds (there is no overlap between the messages sent by each node). On the other hand, in burst traffic, each node sends one message every second (messages are sent at the same time).

Scenario	Description	Message frequency
S1.1	BMS simple scenario Evaluate performance of a BMS (forecasting module, message broker and database) with a simple network.	Every 15 s / Regular traffic
S1.2	BMS simple scenario Evaluate impact, in terms of BMS performance, of sending more frequent messages.	Every Second / Burst Traffic
S2.1	BMS complex scenario Evaluate performance of a realistic BMS with a complex sensor network including bi-directional communication and digital signatures.	Every 15 s / Regular traffic
S2.2	BMS complex scenario Evaluate impact, in terms of BMS performance, of sending more frequent messages.	Every Second / Burst Traffic

Table 7.5. BMS Scenario use cases

For both scenarios, the network is composed of a server node (node 0) and three remote sensing nodes (node 1 to 3). The server node has a forecasting module to predict short term energy consumption from historic data and environmental sensor measurements by means of an artificial neural network (ANN). The sensing nodes send measurement to the server node using RabbitMQ message broker as illustrated in Figure 7.16. The simulation starts off by the training of the ANN, and subsequently the remote sensing nodes starts to send data for 2 minutes. The server node receives that data, makes the energy predictions and saves the received and predicted data to a SQLite database.

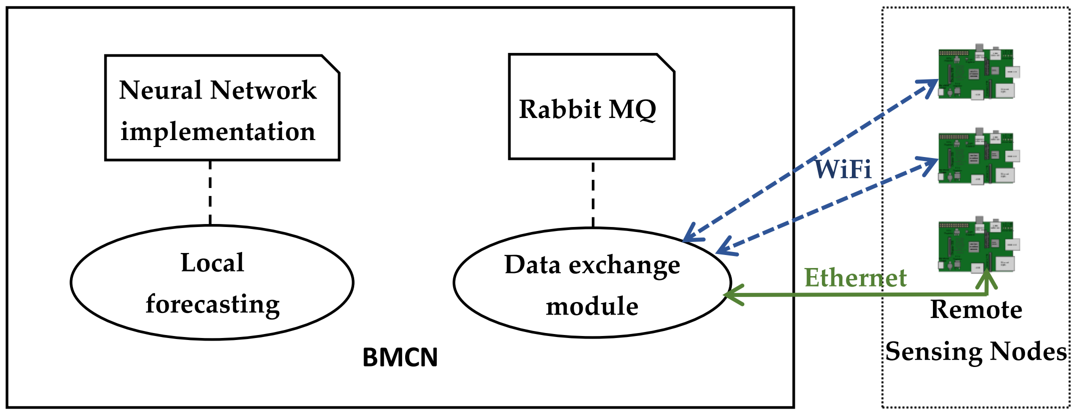


Figure 7.16. COSSIM's test case target involving BMCN and Remote Sensing Nodes

In the first scenario, all the data are sent via wireless connection and the communication is unidirectional (from the nodes to the server). In the second scenario, various levels of complexity are added. First, mixity is included to the topology of the network: node 1 remains wireless while the two other nodes transmit via Ethernet. Bidirectional communication is also introduced, via a reconfiguration of node 1. Lastly, a process of digital signature is used to verify the reliability of the data sent by node 1. Key lengths of up to 2048-bits were used successfully with the SHA256withRSA encryption algorithm.

7.4.1.2 Native and simulated CPS description

From a physical architecture point of view, depending on the complexity of the facilities and the number of remote sensing nodes, the BMCN can be located in an ARM-7 based embedded device or in a separate process located in the same machine as the ICS, an i586 based CPU. An architecture where the BMCN is located in the same machine with the ICS, an i586 based CPU, is chosen for the simulations. On the other hand, the Remote Sensing Nodes are implemented by means of ARM-7 based boards, Raspberry Pi or similar. The CPS descriptions are as following:

- **BMCN:** i586 based CPU, 1 GB RAM, 16GB Non Volatile storage, Full TCP/IP stack
- **Remote Sensing Nodes:** ARM-7 based boards with 512MB RAM, 4GB Non Volatile storage, Full TCP/IP and GPIO (general purpose input output) ports.
- Ethernet and Wi-Fi communications.

The BMS test scenarios have been simulated in the COSSIM framework using the following system:

- **BMCN:** Ubuntu12.04 GEM5 with 4GB of RAM and 4GB of disk space.
- **Remote Sensing Nodes:** ARM-32 GEM5 with 512MB RAM.
- Ethernet and WiFi communications.

In the BMS test case scenarios, an ubuntu12.04 GEM5 image was implemented with the BMCN as a server machine that communicates with ARM GEM5 images as Remote Sensing Nodes.

Concerning the simulations, it should be noted that in an Intel® Xeon(R) CPU E5-2680 v3 Ubuntu machine at 2.50GHz, 16 GB of RAM memory and 70GB of hard disk each simulation for BMS test case lasts approximately 12 to 16 hours. The most significant time-consuming processes are the X86 ubuntu12.04 GEM5 image boot up that lasts near one hour, the RabbitMQ services half an hour, the neural network-based forecasting process several hours and the digital signature application around two hours.

7.4.1.3 Experimental Results

Table 7.6 summarises the processing performance simulation results from both scenarios under regular and burst traffic. The simulated time is coherent with the design of the application. In the second scenario the simulated time is a bit longer because the communication channels stay open a longer to enable the bidirectional communication. Regarding the time elapsed on the host, the simulations took between 10 and 20 hours. The addition message handling of the burst scenario increases the duration by 13.5% for S1 and by 41.5% for S2. The additional complexity of S2 increases the execution time by 32% for regular traffic and by 65% for burst traffic.

Parameter	S1.1	S1.2	S2.1	S2.2
Instructions simulated	39.1G	41.1G	56.0G	97.3G

Seconds simulated	204.22	204.21	239.48	239.49
Time elapsed on host (hh:mm:ss)	10:42:33	12:09:20	14:09:57	20:02:28
Simulated/ Real time	0.00530	0.00467	0.00470	0.00332
Instructions/sec.	1,014,908	938,781	1,098,698	1348,386

Table 7.6. BMS COSSIM Simulator Results from processing component

Figure 7.17 shows how the number of instructions simulated is generally concentrated on node0 (the BMS server). Most of the processing data comes from the ANN training and, to a less extend, from saving the data to the database. This explains the small relative increase in simulation time between the two simulations and suggests that the network handling is not the most resource consuming part of the application. For S2, an increased number of instructions is simulated on node 1. This demonstrates that adding digital signatures to the nodes messages, significantly increases the processing time of the application.

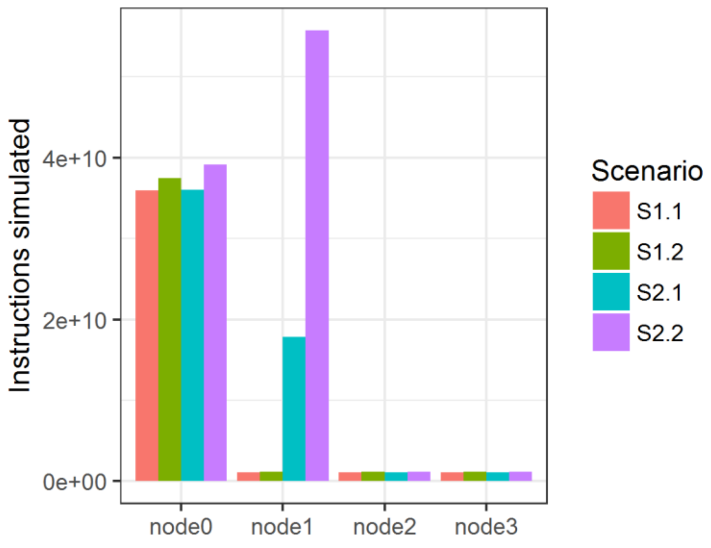


Figure 7.17. Number of instructions simulated on each node

Looking at the network performance metrics from the COSSIM simulator, the bandwidth utilization for regular traffic is noticeably lower than in burst traffic (Figure 7.18). However, even in burst traffic conditions, the bandwidth used is very low compared to the Wi-Fi connection capacity. The information interchanged between the BMS nodes has very low bandwidth requirements. Thus, there are not foreseen issues associated to the scalability of the solution.

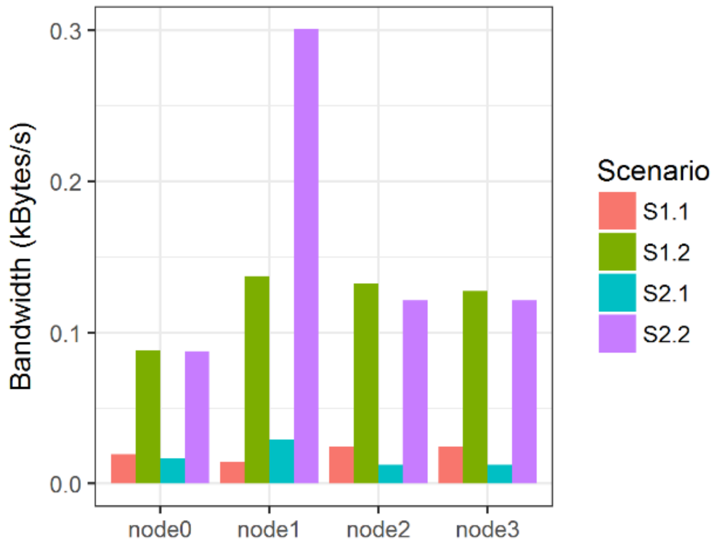


Figure 7.18. BMS Bandwidth for each simulated node

7.4.2 MVS (Mobile Visual Search)

Mobile Visual Search (MVS) [116] is a computer vision application built on the idea of retrieving interesting information about physical objects using only image content; multimedia content can be send back in response to a search or to a user's action. The aim of MVS is to analyze a query image taken by the user and search for similar images inside a large database as illustrated in Figure 7.19. MVS is already used in many different applications such as interactive museum guides, e-commerce mobile applications, localization systems and many more.

The MVS algorithm is composed of two consecutive modules: the Image Analyzer (IA) and the Retrieval Stage (RS). Firstly, in the IA, a series of advanced image processing techniques are applied on the image acquired by the user in order to localize relevant region of it (key-points). Each selected key-point is then analyzed and, for each one of them, a local descriptor is extracted. The last operation of the IA is the aggregation of the computed local descriptors into a compact global descriptor which describes the whole image. These descriptors are concatenated and further compressed in order to reach a size of around 15-20 KB for each VGA image. The whole IA stage can be performed directly by the user's mobile device.

On the other hand, the RS is able to compare the image descriptor created in the IA with a plethora of other descriptors extracted off-line from each image in the database. In the first step of RS, the images in the database are quickly ranked in respect to the similarity of their global descriptor in respect to the one computed from the query image. In this way, only a short list of the images in the database are selected as possible matches. A more precise comparison using local descriptors, including geometry checks and outliers rejection, is performed and the most similar image to the query one is finally selected. The RS requires access to the whole database and for this reason it is performed server-side.

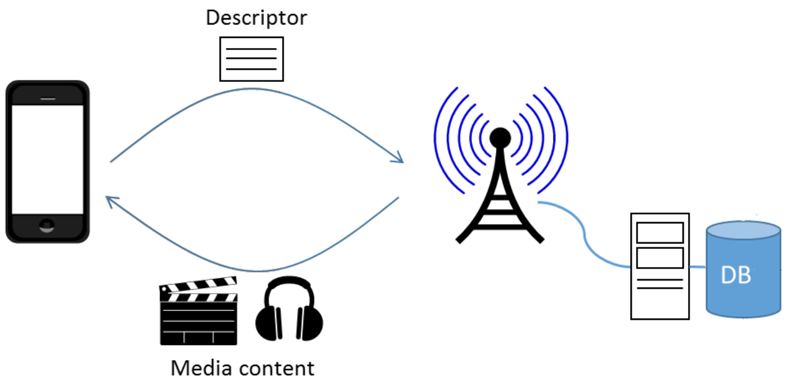


Figure 7.19. Mobile Visual Search topology

7.4.2.2 Native and simulated CPS description

In order to compare the result obtained using COSSIM we used the following native system:

- 1 central node with Intel Core i5-5200 2.2 GHz quad Core CPU and 4 Gbytes of RAM
- 1 ARM based device chosen from the following two: Hardkernel Odroid XU3 (Cortex-A15 2.0 GHz quad core and Cortex-A7 1.4 GHz quad core CPU and 2Gbytes of RAM).
- Wireless Network

Having two different ARM based devices let us compare the ARM machines simulated in COSSIM (based on Atomic Simple and Out Of Order models) with different real ARM processors (A7, A15). Moreover, with these two devices we can assign specific task to be executed by a specific core and we can also set the maximum frequency of each core in order to compare more precisely the results obtained in the simulation. Lastly, the Odroid XU3 comes integrated with the power analysis tool that was modified and used by us in order to collect the data reported in the next sessions.

The MVS cloud application simulated inside COSSIM is composed as follow:

- 1 central node with X86 GEM5 Atomic Simple 2 GHz CPU and 512 Mbyte of RAM
- 1 imaging node with 2 possible different processors: ARM GEM5 Atomic Simple at 1 GHz CPU (In-order), ARM GEM5 Out Of Order at 2 GHz CPU. All of them have 512 Mbytes of RAM.
- Wireless Network (using Microrouters)

7.4.2.3 Quality of Results

In order to test the correctness of the simulation the final output of the application was compared with the output of the native system. In particular, the local and global descriptor scores (defined in Section 7.4.2) extracted from the ordered list of results were compared. Figure 7.20 reports the top three matches for the given query image on the left. As can be observed, all the result images depict the same building as the query one. For each result we report the local (result of local matching) and global (result of global matching) scores in the simulated and real scenarios. The simulation replicates exactly the computations performed by the native system, with a complete agreement between native and simulated search results.

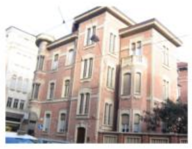
Query	**1st result**	**2nd result**	**3rd result**
			
	Native		
Local Score	93.72	60.21	13.06
Global Score	315.759	63.202	50.172
	Simulated		
Local Score	93.72	60.21	13.06
Global Score	315.759	63.202	50.172

Figure 7.20. Global and Local Matching scores on the Building Database (The result images correctly depict the same object as the query one in both simulated and native system)

7.4.2.4 Experimental Results

The simulation involves the complete MVS application running on a CPS composed by a mobile imaging node and a central one. In this case, the query image is first processed on the mobile device (ARM) in order to extract some descriptors, thus creating a compact representation of it, and then it is sent to the central node to be compared with all the other descriptors extracted from each image in the database, in order to find visually similar images. In this configuration, only compressed information about the image (descriptor) is sent through the network from the mobile device to the central node.

Two different simulations have been run:

- ARM simulated with Atomic Simple CPU model with 1 GHz as frequency (In-Order)
- ARM simulated with Out of Order CPU model with 2 GHz as frequency

The native and simulated systems were tested on the same three VGA images that depict an object included in the database viewed by a different angle.

7.4.2.4.1 Atomic Simple (In-Order)

The results described in this section are related to the following set up:

- Simulated CPS with ARM CPU Atomic Simple at 1 GHZ
- Native CPS Odroid Xu3 using just 1 ARM A7 core limited at 1 GHz

Regarding the power and energy measurements they are collected just from the Odroid device as described in Section 7.4.2.2.

Time measurements

Table 7.7 describes the Imaging node application time measured by the cGEM5, while it was compared with internal timers located inside of the MVS application time. The two-time measurements are almost the same (mean error of 0.19 ms), that means that inside the simulation the time is measured in a consistent fashion. From now on we will simply refer to them as simulated time.

Imaging node application time (ms)	Internal Timers	cGem5 stats	Native (ARM A7)	Host (Simulation Time)
Image 1	14499.3	14499.1	15613.7	10941930
Image 2	21459.3	21459.08	23229.2	16133510
Image 3	14437.3	14437.14	15855.9	10750010
Average	16798.63	16798.44	18232.93	12608483

Table 7.7. MVS Simulated time comparison using ARM A7 model

From Table 7.7 we can also observe that the Imaging node application time (that coincide with the overall simulation time) is close to the time that we get from a native system that includes an ARM A7 device (7.8 % mean error[23]). In addition, the average host time is approximately of 3 hours and 30 minutes. Similar results are obtained for the central X86 node as described in Table 7.8.

Central node execution time(ms)	Simulated Time (X86)	Intel i5-5200
Image 1	645.068	508.8
Image 2	641.135	498.793
Image 3	644.614	502.894
Average	643.6057	501.4957

Table 7.8. MVS Simulated time comparison using ARM A7 model

Energy/Power measurements

Table 7.9 compares the Energy and Peak Power of simulated and native ARM A7. We can observe that in the majority of the case and also on average (less than 400 mJ of difference) the ARM energy consumption estimated in the simulation is quite close to the real one. Furthermore, it could also be noticed that the peak power estimation is satisfactorily accurate (average error of 0.05 W).

[23] This error is due to different clock frequencies (1GHz in simulated time vs 1.4GHz in A7).

	Simulated ARM A7 Energy (mJ)	Native ARM A7 Energy (mJ)	Simulated ARM A7 Peak Power (W)	Native ARM A7 Peak Power (W)
Image 1	889.856348	1381.52	0.172926	0.12
Image 2	1333.858905	1982.53	0.172926	0.14
Image 3	887.620033	899.63	0.172926	0.11
Average	1037.111762	1421.226667	0.172926	0.123

Table 7.9. MVS Energy & Power comparison using ARM A7

7.4.2.4.2 Out of Order

In order to test the Out Of Order ARM CPU model to the following set ups are compared:

- Simulated CPS with ARM CPU Out Of Order at 2 GHz
- Native CPS with Odroid Xu3 using just 1 ARM A15 core limited at 2 GHz

Time measurements

Table 7.10 describes the simulated time of Imaging node application time measured by the COSSIM simulator using ARM A15 Out Of Order model. As can be observed, the simulated time of native ARM 15 device is similar with Out Of Order ARM model available in COSSIM. In addition, the simulation with Out of Order ARM CPU model takes more than 31 hours comparing with the Atomic Simple which takes only 3 hours.

Imaging node application time(ms)	Simulated Time OoO	Native (ARM A15)	Host (Simulation Time) OoO
Image 1	5068793	5824470	98857310
Image 2	6678295	8956860	145087200
Image 3	5078255	6148150	99378850
Average	5608447	6976493	114441120

Table 7.10. MVS Simulated time comparison using ARM A15 model

Energy/Power measurements

Table 7.11 compares the Energy and Peak Power of simulated and native ARM A15. As can be observed, the energy consumption and peak power is closer to the native result.

	Simulated ARM A15 Energy (mJ)	Native ARM A15 Energy (mJ)	Simulated ARM A15 Peak Power (W)	Native ARM A15 Peak Power (W)
Image 1	692.661715	753.71	1.503026	1.54
Image 2	941.130709	1236.84	1.503026	1.55
Image 3	693.604826	771.97	1.503026	1.54
Average	775.7990833	954.173333	1.503026	1.54

Table 7.11. MVS Energy & Power comparison using ARM A15

Network Energy consumption

Figure 7.21 reports the communication energy consumption by COSSIM of the micro router return associated with the ARM device. From the figure we can observe that before 10.2 seconds there is no communication between the devices (before that time the mobile node is executing the descriptor extraction and the retrieval stage on the local database). The energy consumed before that time is 0.0206 J (in 10.3 seconds) which is coherent with the idle power set of 2mW. Between time 10.4s and 10.6s (in which the slope of the curve locally increases) the transmission of the descriptors occurs and at the end the central node sends back the result to the imaging node.

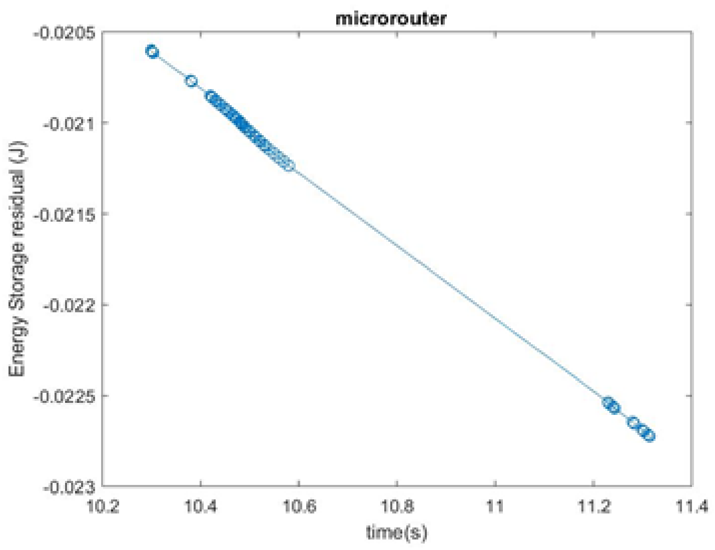


Figure 7.21. MVS Network Energy Consumption for the ARM device

7.4.2.5 COSSIM Co-Simulation

Authors in [117] presents a co-simulation method in COSSIM framework moving some heavy computations outside of COSSIM and executed natively on the machine hosting the simulation (host) or an additional machine, i.e. in a co-simulation configuration. Specifically, the co-simulation uses a Universal Asynchronous Receiver Transmitter (UART) peripheral on the cGEM5 machine to allow it to communicate with its host, which sees the peripheral as a socket. The external component runs on the host machine and connects to this socket, from which it can read what an application running in cGEM5 writes on the UART output channel and the application can read on the UART input what the external component writes on the socket. The application sends the parameters to the external component, that receives them and calls the specific function we want to run externally. Once finished returns to cGEM5 application the results. The user can configure the external component to call on a remote machine. In particular in the MVS simulation, the IA needs to be executed by an ARM machine while the host of the simulation is an x86. In this case, an Ethernet connection between the host and the ARM device is created and the external components on the host call the function on the ARM machine through Ethernet connection as illustrated in Figure 7.22.

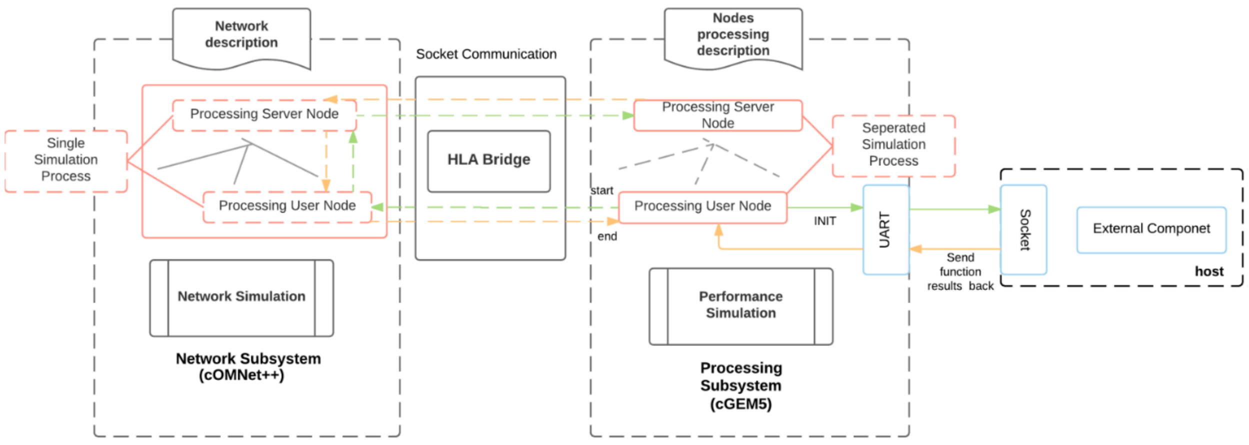


Figure 7.22. COSSIM Co-Simulation using MVS use case

Figure 7.23 reports in hours, minutes and seconds the simulated and host times for the imaging node in both the co-simulation and the simple simulation (COSSIM) scenarios. Two different ARM machines are used for the co-simulation. The first one is the same used in the native system (i.e. Hardkernel Odroid XU3 including ARMcortex-A15), while the second one is a STMicroelectronics B2260 2 using a ST Cannes2-STiH410-EJB SoC including a dual ARMcortex-A9 at 1.5 GHz.

The native time of the IA is only 0.6s on the Odroid XU3. After setting in the Odroid XU3 the number of active cores and their maximum frequencies in line with the simulated device inside COSSIM the native time increases to 5.2s.

The simulation of the descriptor extraction takes 10426.46s (about 2 hours and 53 minutes) of host time on a single VGA image due to the complex computer vision algorithms involved, as described in Section 7.4. When co-simulating takes place using an Odroid XU3, the host time decreases significantly to 78.45s (less than 1 minute and a half), which is the 0.75% of COSSIM simulation time. On the other hand, co-simulating with the ST B2260 takes 82.41s. The difference, in term of host time, between co-simulating using the ST B2260 and ODROID XU3 is due to the slower ST B2260 processor compared to the ODROID one (ARM A9 vs A15).

In the accelerated version we have benefits in term of host time and thus simulator speed, but we get incorrect results in the application and CPU times. Due to the fact that the two systems (simulation and co-simulation component) do not exchange messages to synchronize their time and also run with different real-time clock rates the simulation times measured in the accelerated version of the application are slightly different from the real execution times. After the string of the descriptor is sent to the external component, the time of the two systems runs differently, since the simulator is slower than a real component. Moreover, the descriptor needs to be split into small packets and sent with some delay from one to the other due to channel synchronization issues. In this case, each packet is 200 Bytes and it is sent at one-second intervals (as measured by the host machine).

Summarizing, this solution is a good fit if the user needs to prove the correctness of the application, as could be the case in the first stage of the development process, while it is unfit if the user wants to test the times of the system in a real environment.

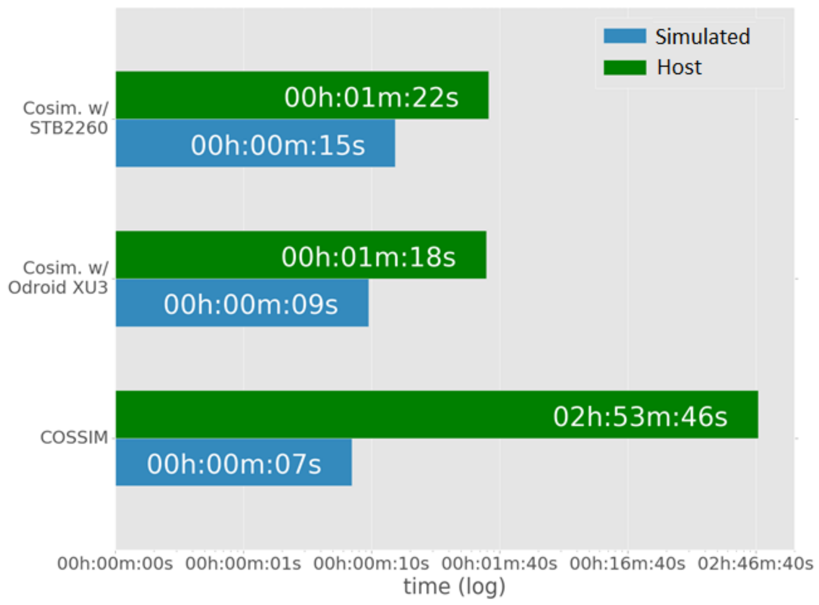


Figure 7.23. COSSIM Co-Simulation using MVS use case

7.5 Validation and Performance Analysis of ACSIM

Initially, the efficiency of SystemC integration with full system simulator (i.e. GEM5) has been evaluated comparing our proposed method (i.e. ACSIM - Chapter 6) with the conventional manual approach. Subsequently, a validation and performance results of SystemC integration with whole COSSIM simulator is presented using a SystemC-described accelerator of a real application.

7.5.1 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 7.24. 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.

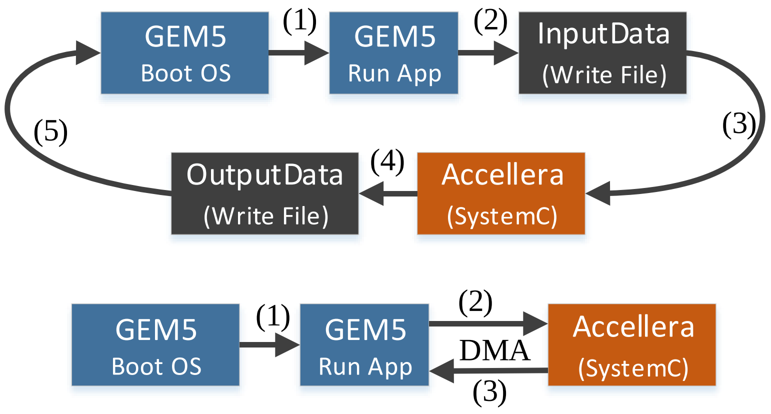


Figure 7.24. (i) Standard (top) & (ii) Our novel method efficiently integrating GEM5 Full System Simulator with Accellera Simulator (bottom)

Based on the existing conventional flow, the application executed in GEM5 writes the data that should be fed to the SystemC simulator in files, denoted as Steps 1 & 2 of Figure 7.24. 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. Subsequently, in the conventional method, the SystemC simulator returns its results to the GEM5 application using output files, denoted as Steps 4 & 5. Another important remark is that normally in every SystemC simulator call, the full-system must boot the OS from scratch, while, and more importantly, no notion of synchronisation exists in such a file-based exchange.

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 7.24.

7.5.1.1 Use Cases

The following paragraphs offer a short but necessary theoretical background to each of the selected use cases. 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. In such a way, in our experiments, we have evaluated the whole range of accelerators, from very simple ones (i.e. implementing just one or very few iterations) to quite complicated ones (i.e. implementing tens of iterations).

1) Mutual Information: Mutual Information (MI) [118] measures the information that two random variables, X and Y, share. Alternatively, it measures the extent to which the uncertainty about one of the two is reduced through the information known on the other (i.e. to what degree the knowledge of one of these variables reduces the uncertainty of the other one). For instance, if X and Y are independent, knowing the value of X does not give any information about Y and thus their Mutual Information is zero. On the other hand, if X is a deterministic function of Y and Y is a deterministic function of X, then all the information conveyed by X is shared with Y. In this case the Mutual Information is the same as the uncertainty contained in X or Y alone.
2) Transfer Entropy: Transfer Entropy (TE) [119] is a nonparametric statistic, which measures the amount of directed (time-asymmetric) transfer of information between two random processes. Moreover, Transfer Entropy is able to distinguish the driving and the responding elements and to detect asymmetry in the interaction of subsystems. Also, it constitutes a conditional Mutual Information, which uses the history of the influenced variable in the condition. Transfer Entropy is very frequently utilized in the analysis of nonlinear signals; however, it usually requires a very large number of samples in order to end up with an accurate estimation. While it was originally defined for bivariate analysis, it has been extended to multivariate forms,

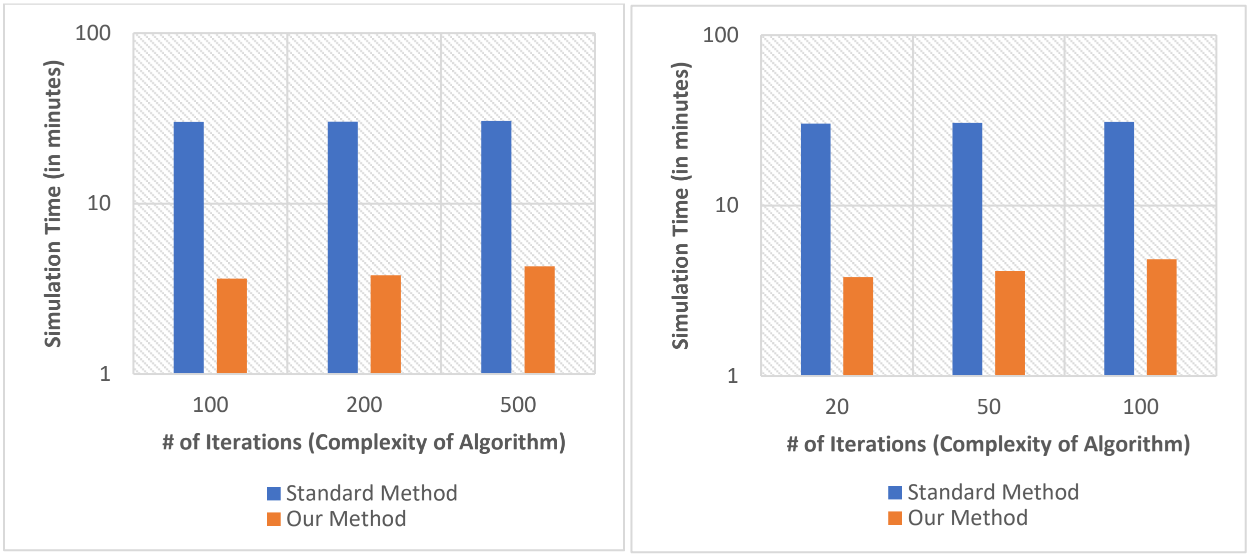
either conditioning on other potential source variables or considering transfer from a collection of sources.

7.5.1.2 Comparative Results

The processing platform used in the comparative analysis is an Intel i5-4590 at 3.3GHz with 16GB of RAM. Furthermore, the MI and TE results are based on a typical data size 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 7.25; 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 7.26 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).

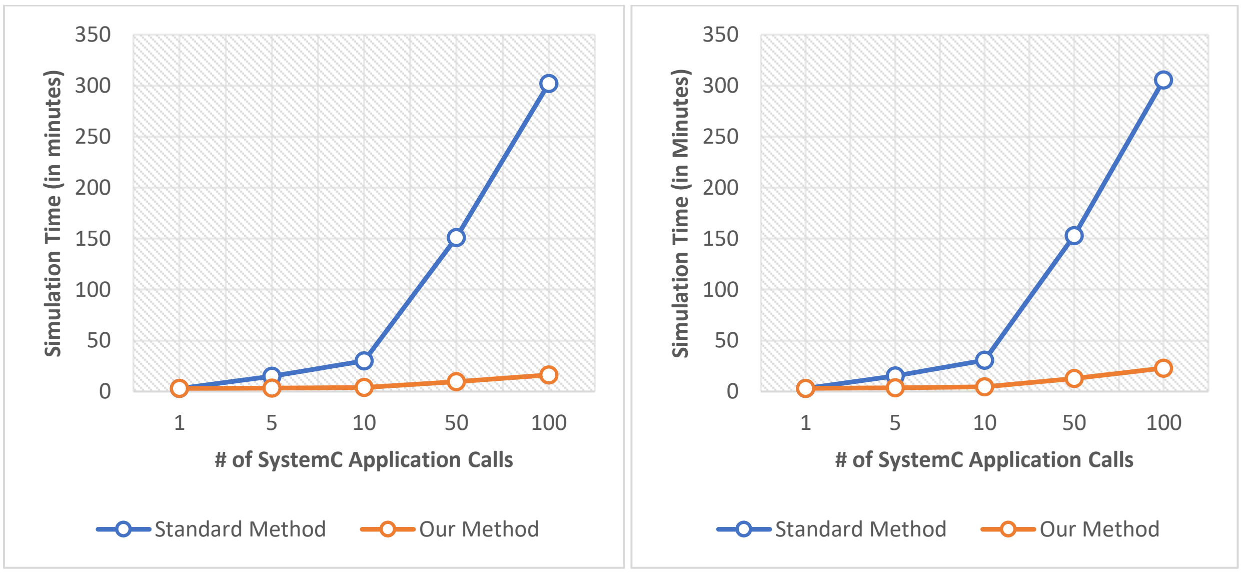
Figure 7.25 first illustrates the overall simulation time when calling 10 times the SystemC accelerator (i.e. our MI and TE SystemC code executed on top of the Accellera Simulator). It becomes apparent that the proposed method is one order of magnitude faster than the conventional file-based method in 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 7.26 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 remains 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.



(a) Mutual Information Algorithm **(b) Transfer Entropy Algorithm**

Figure 7.25. 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**

Figure 7.26. Simulation time of two use cases using different # of Accelerator calls

Finally, in order to evaluate the host/device interactions in ACSIM, we measure both the simulated transfer time from GEM5 to our accelerator as well as the overall ACSIM simulation time as illustrated in Figure 7.27 and Figure 7.28 respectively. Specifically, in our experiments, three different amounts of data are measured from 100MB to 400MB using three transfer rates (from 8Bytes up to 32Bytes per SystemC cycle[24]). In case of 100MB and 8Bytes/cycle for example, ACSIM reports 28ms as simulated time using 500MHz accelerator (100MB/8Bytes = 12,5M cycles or 25ms) plus overhead for the transaction initialization. As a result, the simulated time which reported by ACSIM is quite accurate. In addition, as illustrated in Figure 7.27 the simulated time is decreased approximately by 1,93x doubling the transfer rate due to overhead during initialization of the transaction. Finally we can observe from Figure 7.28 that ACSIM has negligible overhead in overall simulation doubling the amount of transfer data.

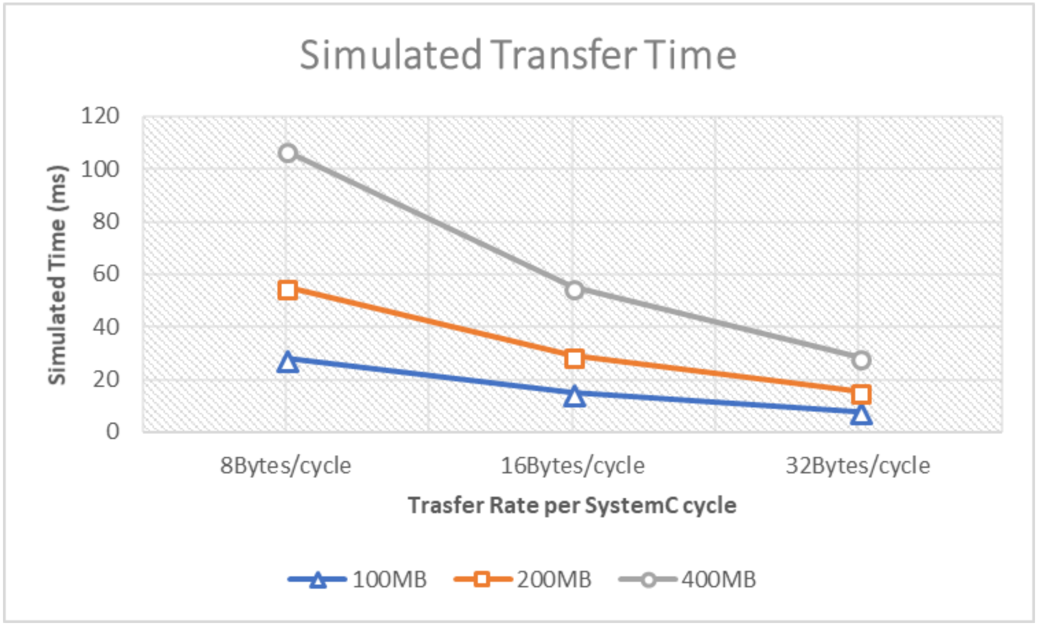


Figure 7.27. ACSIM Simulated Transfer Time using different transfer rates

[24] The user can define the transfer rate through a parameter.

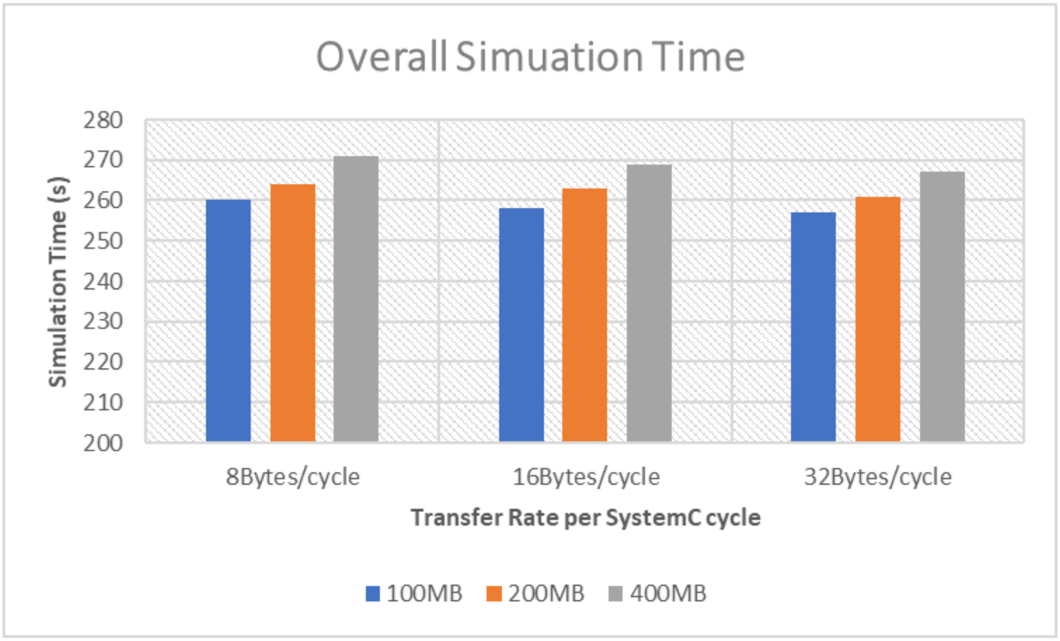


Figure 7.28. Overall ACSIM Simulation Time using different transfer rates

7.5.2 Evaluation of ACSIM within COSSIM framework

Moving to the overall evaluation of ACSIM (in conjunction with COSSIM) it is based on a Reservoir Simulation (RS) application which is the state-of-the-art technology used to predict field performance 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 [120] (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 method [121] that provide results at less iterations and, therefore, in less time.

Figure 7.29 illustrates the ACSIM system when simulating the overall the RS system including the Hyperbolic SystemC cores. Specifically, each processing node represents an ARM-based System running BusyBox OS tightly interconnected with two SystemC instances of the Hyperbolic algorithm. 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.

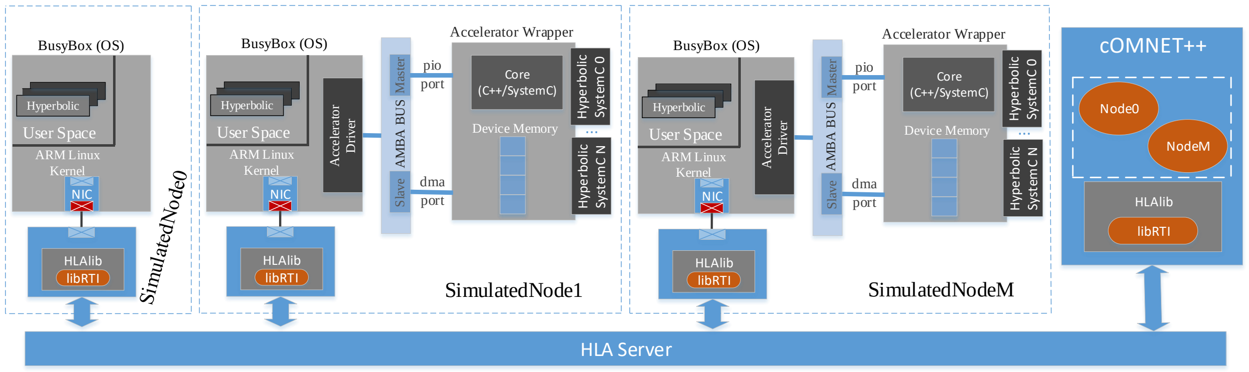


Figure 7.29. A representation of COSSIM simulator with ACSIM integration using Hyperbolic SystemC cores

Figure 7.30 demonstrates the simulated time reported when the whole application is simulated without the accelerators (grey bars) and when ACSIM also simulates the accelerators. Those numbers are from 2-512 consumer nodes using 800K samples. The experiment is executed using multiple physical cores in a *cluster* of multiple distributed machines. Specifically, a *42U HP RACK 10000 G2* Series cabinet is used 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 in the above cluster). The network part of COSSIM (OMNET++) is executed in typical workstation based on an Intel i5-4590 processor (3.3GHz) running Ubuntu Linux 14.04 with 16GB of RAM, while the processing parts (GEM5 instances) and HLA server are executed in the cluster which is described above running Ubuntu Linux 16.04.

Figure 7.30, 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 back[25] 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 [122] and show the efficiency of our approach.

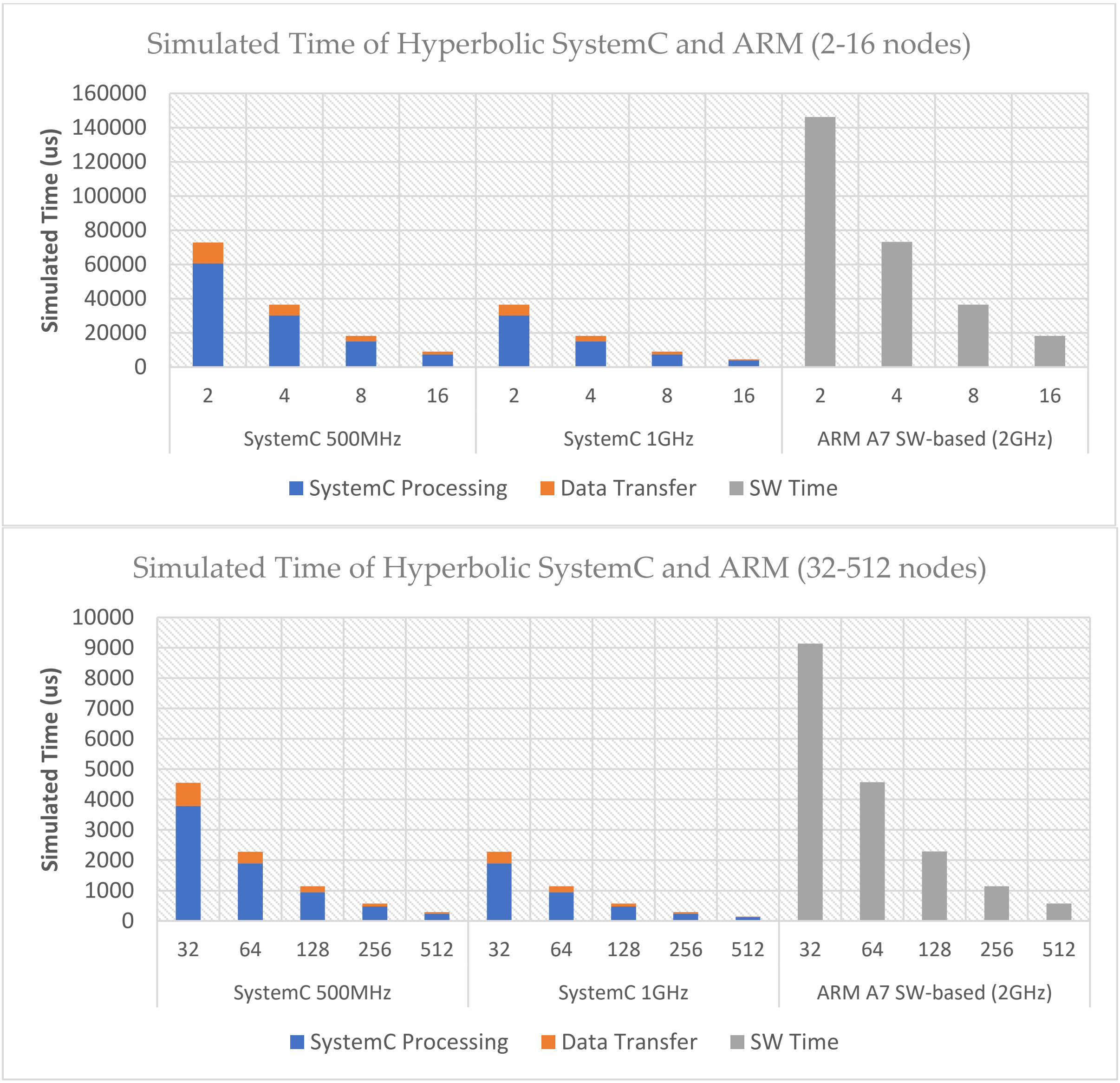
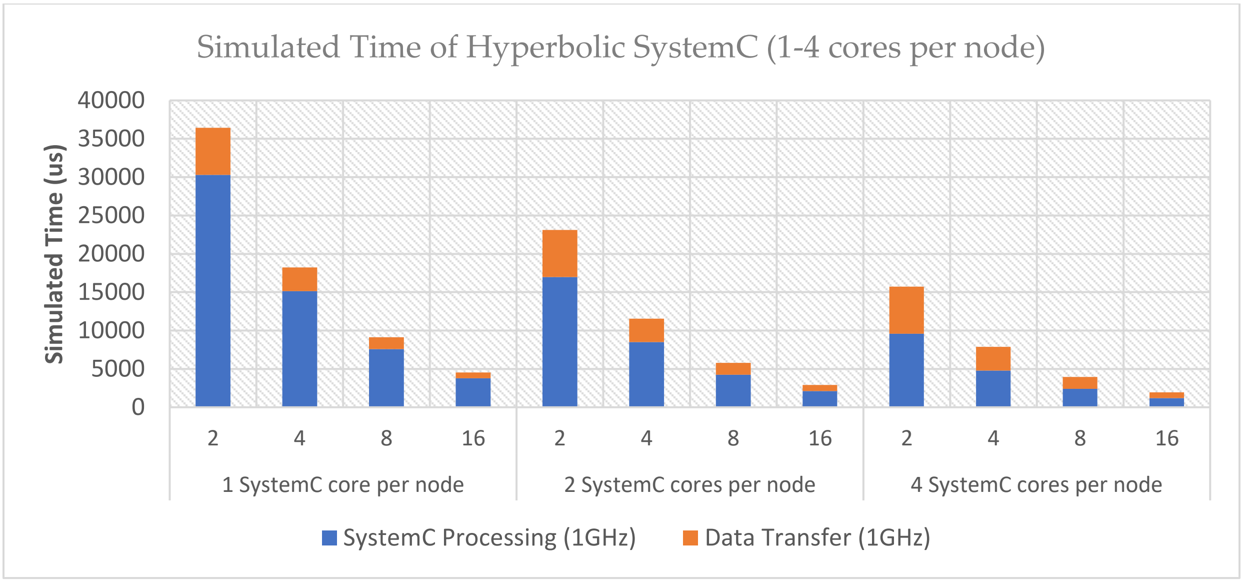


Figure 7.30. Simulated Time of Hyperbolic SystemC Processing and Data Transfer at 500MHz and 1GHz versus ARM SW-based version at 2GHz using 800K samples (upper: 2-16 nodes; lower 32-512 nodes)

[25] 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.

In Figure 7.31 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.



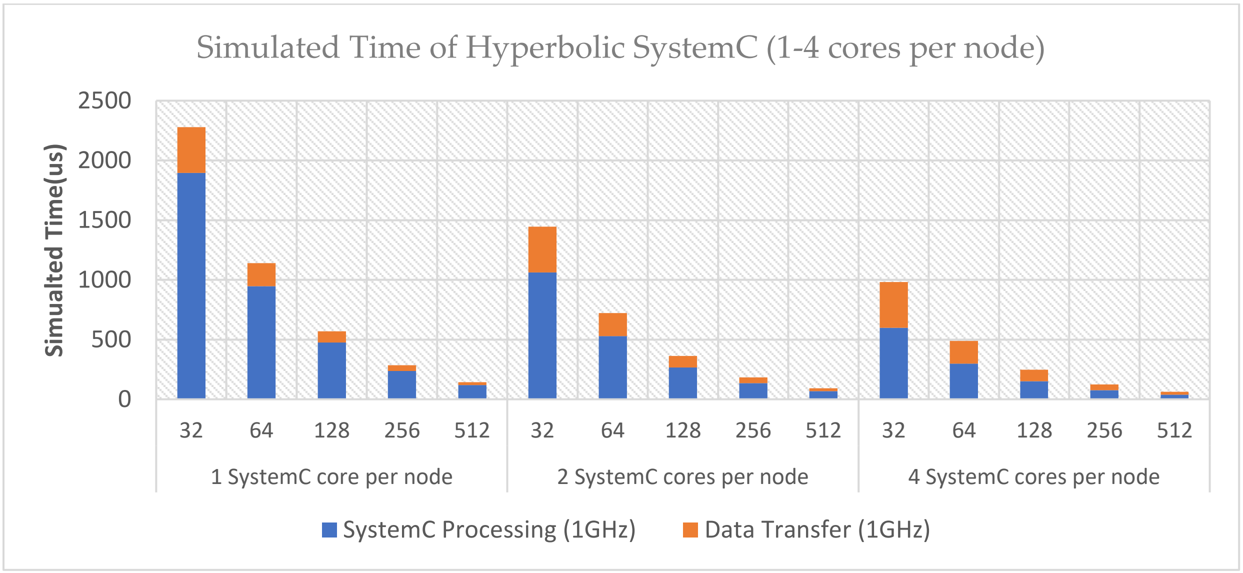


Figure 7.31. Simulated Time of Hyperbolic SystemC Processing and Data Transfer at 1GHz using 1-4 SystemC cores per node (upper: 2-16 nodes; lower 32-512 nodes)

8

Conclusions and Future Work

In this work, we present the COSSIM Simulation Framework, an open-source novel solution that addresses, in a single integrated toolset, the simulation of both the processing and the networking parts of highly parallel systems, while also taking into account the power consumption aspects of them. By using a novel dual-stage synchronization scheme COSSIM is the first known system providing global cycle accurate simulation, highly parallel/distributed execution and high extensibility.

On top of that, 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 together with custom hardware accelerators that are designed in synthesizable SystemC; if needed non-synthesizable SystemC can also be utilized.

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 in highly parallel heterogeneous systems since different number of nodes, different number of accelerators per node and different clock frequencies of the accelerators can all be seamlessly simulated and evaluated, while also accelerate the overall development and verification process.

Based on a number of experiments the COSSIM/ACSIM simulator proved to produce very accurate results (up to 99% accuracy) for both network and processing heavy applications, while it is most probably the only simulator that can efficiently simulate up to a thousand of processing nodes interconnected together in a full distributed manner. 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 and 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 functionality of the Simulation Framework in the CPS domain, we aim to integrate COSSIM with a widely used physical process simulator (i.e. Ptolemy) by using the already developed HLA interface. Additionally, we have already started working on accelerating the framework through the use of FPGA devices. At this stage, we have focused on accelerating the power/energy estimation processes.

Appendix – Lessons Learnt

This appendix describes the most important lessons learnt regarding the implementation issues of the sub-systems developed in the context of the COSSIM/ACSIM simulation frameworks.

- ✓ First of all, in GEM5's publicly available repositories, the only network interface card implemented, tested and verified is the Intel 8254x based gigabit Ethernet adapter. It is provided as a PCI GEM5 network device using the e1000 Linux driver. Unfortunately, the latest version of *GEM5* supports the real-network device only on *ARM-based* architectures. In order to support the *COSSIM* requirements as stated above we had to modify *GEM5* for *x86 ISA* and configure it properly. In the process to achieve this functionality on *x86*, proper drivers for the specific network device were also required. These drivers, however, required a more modern Linux kernel build for *x86* than the one available in the GEM5 repositories (only 2.6.x kernels were available and drivers required 3.x kernels) and as a result we custom-built such a kernel and a custom-made network card of the *x86-based* systems.

- ✓ In addition to the network interface cards, *GEM5* supports networking through a simple *Etherlink* device. *Etherlink* is a virtual dummy link which emulates a cable over which Ethernet packets are sent and received without any delay (no switching or routing functionality is implemented). In the scope of *COSSIM*, this limitation (only a single NIC and a single type of network) are unacceptably restrictive. Therefore, *Etherlink* could not be used in its current form at all and it had to be modified. Since device models are not easy to develop without inside information from their manufacturer (the Intel *NIC* model used in *GEM5* has been contributed by Intel itself), in order to support a varying number of physical networks, we focus on tapping the Ethernet packets from *Etherlink* and send them to a Networking Simulator. To achieve the aforementioned, we employ *CERTI HLA* and specifically a virtual device named *COSSIMlib* is developed and integrated to the main core of the *GEM5* system through

Etherlink to synchronize and interconnect the *GEM5* with *OMNET++* simulator implementing two synchronization levels (*Synchronization per node* and *Global Synchronization*) as described in Chapter 4 & Chapter 5. Specifically, *COSSIMlib* is a wrapper to an RTI Ambassador Class serving to exchange messages over the network with the HLA Server (RTIG process) via TCP (and UDP) sockets. *COSSIMlib* exchanges Ethernet Packets captured from the *Etherlink* Device and sends (and accordingly receives) them to (from) the *HLA* Server (RTIG). The HLA Server forwards these messages to a proper interface in Network Simulator (cOMNET++) that implements all the network related functionality (NIC physical layer and actual network). Finally, *SynchServer* is developed to initialize the HLA connections in both GEM5 and OMNET++.

- ✓ On the other hand, a dedicated network simulator in COSSIM is to be able to support multiple network protocols, topologies and devices through which nodes (represented by cGEM5 instances) can be interconnected. OMNeT++ has been chosen as the most capable and feature-rich network simulator in that context, however there are issues that arise. Initially, OMNeT++ does not support the real protocol stacks (e.g. Linux ones) as GEM5 and therefore to be able to bridge the two packages and use freely all OMNET++ legacy requires a procedure that encapsulates/decapsulates cGEM5 binary packets into OMNET++ - compatible packets. In addition, in order to support different network protocols (generally compatible till a certain level, such as WiFi protocols) within OMNeT++, a micro-router functionality is implemented as described in Chapter 4.

- ✓ Based on the related work which is described in Section 3.4, it is clear that currently there is no framework supporting all the features of COSSIM/ACSIM, namely simulation of hardware accelerators together with CPUs and networks in a fast and fully featured way. Specifically, the existing approaches mainly 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 HPC/CPS/Cloud system design; moreover, they cannot support any kind of networking. In order to eliminate the above restriction, we have developed a set of device drivers as well as Accelerator Wrapper Device to achieve efficient communication between the H/W accelerator and the processing simulator.

- ✓ In the context of security vulnerabilities and specifically for fuzz-testing integration, *pause-resume* functionality is implemented in order to integrate *Fuzz-testing* system with the COSSIM framework. The main concept behind fuzz testing is to send a large number of test vectors to the system under test, making sure the system is in the same exact state before each request; sending even a single test vector to the system will change its internal state and have a possible effect on the processing of subsequent test vectors, thus potentially invalidating the results of the test. In other words, it needs to take a *GEM5* checkpoint and restore *GEM5* simulations from the same checkpoint a lot of times (Chapter 4). Of course, this is possible in standalone *GEM5*, but this is not possible in COSSIM framework because even though GEM5 is able to store and restore its own state (using checkpoints), OMNeT++ on the other hand cannot support this feature (store/restore). Moreover, *GEM5* in the COSSIM environment cannot be stopped but paused in every checkpoint. That's why *HLA* connections will be dropped if some of *GEM5* node stops its execution. On the other hand, *GEM5* restore option can be performed only in a new *GEM5* execution. The proposed solution in COSSIM (to encounter the above limitation) is the *pause/resume* functionality which is developed in the context of this thesis and it is integrated in *cGEM5*. This implementation allows a particular *GEM5* instance to be pause, while sending an appropriate message to the fuzz-testing module and bringing up a separate standalone *GEM5* instance featuring the saved state, in order to perform all kind of tests without interfering with the overall COSSIM simulation.

- ✓ The COSSIM framework can simulate the computation along with the networking aspects of a *CPS* in a holistic approach. Since this creates a bias towards the cyber part rather than the physical part of a *CPS* system, COSSIM need to provide an option to integrate simulation of components of the physical world. For this reason, GEM5 is extended to include sensor devices in the processing system as memory mapped peripherals. This provides an interface to include physical world components (their behavior) to the overall simulation.

- ✓ In GEM5 community, there is not any x86 full Ubuntu image to support apt-get installation packages. It was necessary because one of the COSSIM use cases needs a lot of Ubuntu packages with dependencies. As a result, we create from scratch a x86 GEM5-compatible image with full Ubuntu 12.04.

- ✓ OMNET++ is not support any function to measure the traffic rate of packets. As a result, we implement a function to measure traffic rate in OMNET++ statistics.

- ✓ There is not a standard method to integrate the GEM5 with McPAT. So, we integrate cGEM5 with McPAT execution after COSSIM simulation through our Graphical User Interface.

Bibliography

[1] "NIST | National Institute of Standards and Technology," *NIST*. [Online]. Available: https://www.nist.gov/national-institute-standards-and-technology.

[2] H. Gill and R. Baheti, "'Cyber-physical systems', The Impact of Control Technology," *Wash. C IEEE*, p. 161−166, 2011.

[3] "The Ptolemy Project." [Online]. Available: https://ptolemy.eecs.berkeley.edu/.

[4] "Model-Based Design of cyber-physical systems in MATLAB and Simulink." [Online]. Available: https://www.mathworks.com/discovery/cyber-physical-systems.html.

[5] "TOSSIM." [Online]. Available: http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM.

[6] "ATEMU - Sensor Network Emulator / Simulator / Debugger." [Online]. Available: http://www.hynet.umd.edu/research/atemu/.

[7] "Avrora - The AVR Simulation and Analysis Framework." [Online]. Available: http://compilers.cs.ucla.edu/avrora/.

[8] G. Chelius, É. Fleury, and A. Fraboulet, "Worldsens - Development and Prototyping Tools for Application Specific Wireless Sensors Networks."

[9] "Cooja Simulator - Contiki." [Online]. Available: http://anrg.usc.edu/contiki/index.php/Cooja_Simulator.

[10] H. Sundani *et al.*, "Wireless Sensor Network Simulators A Survey and Comparisons," *Comp. Int. J. Comput. Netwroks*, pp. 249–265, 2010.

[11] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, 2006, pp. 641–648.

[12] "SimulAVR." [Online]. Available: http://www.nongnu.org/simulavr/.

[13] "GEZEL Hardware/Software Codesign Environment." [Online]. Available: http://rijndael.ece.vt.edu/gezel2/.

[14] 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. Exp.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[15] D. Meisner, J. Wu, and T. F. Wenisch, "BigHouse: A simulation infrastructure for data center systems," 2012, pp. 35–45.

[16] 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*, 2010, pp. 1–5.

[17] P.-O. Östberg *et al.*, "The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation," in *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, Washington, DC, USA, 2014, pp. 26–31.

[18] S. K. Garg and R. Buyya, "NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp. 105–113.

[19] 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.

[20] B. Louis, K. Mitra, S. Saguna, and C. Åhlund, "CloudSimDisk: Energy-Aware Storage Simulation in CloudSim," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 11–15.

[21] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 475–484.

[22] M. Shiraz, A. Gani, R. H. Khokhar, and E. Ahmed, "An Extendable Simulation Framework for Modeling Application Processing Potentials of Smart Mobile Devices for Mobile Cloud Computing," in *2012 10th International Conference on Frontiers of Information Technology*, 2012, pp. 331–336.

[23] M. Bux and U. Leser, "DynamicCloudSim: Simulating heterogeneity in computational clouds," *Future Gener. Comput. Syst.*, vol. 46, pp. 85–99, May 2015.

[24] M. Hsieh, K. Pedretti, J. Meng, A. Coskun, M. Levenhagen, and A. Rodrigues, "SST + Gem5 = a Scalable Simulation Infrastructure for High Performance Computing," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2012, pp. 196–201.

[25] C. Minkenberg, W. Denzel, G. Rodriguez, and R. Birke, "End-to-End Modeling and Simulation of High- Performance Computing Systems," in *Use Cases of Discrete Event Simulation*, Springer, Berlin, Heidelberg, 2012, pp. 201–240.

[26] W. Hurst, S. Ramaswamy, R. Lenin, and D. Hoffman, "Development of Generalized HPC Simulator," in *Distributed Computing and Internet Technology*, 2010, pp. 176–179.

[27] A. F. Rodrigues *et al.*, "The Structural Simulation Toolkit," *SIGMETRICS Perform Eval Rev*, vol. 38, no. 4, pp. 37–42, Mar. 2011.

[28] J. Lange *et al.*, "Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[29] G. Zheng, G. Kakulapati, and L. V. Kale, "BigSim: a parallel simulator for performance prediction of extremely large parallel machines," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 78-.

[30] S. Böhm and C. Engelmann, "xSim: The extreme-scale simulator," in *2011 International Conference on High Performance Computing Simulation*, 2011, pp. 280–286.

[31] W. E. Denzel, J. Li, P. Walker, and Y. Jin, "A Framework for End-to-End Simulation of High-performance Computing Systems," *SIMULATION*, vol. 86, no. 5–6, pp. 331–350, May 2010.

[32] J. E. Kim and D. Mosse, "Generic Framework for Design, Modeling and Simulation of Cyber Physical Systems," *SIGBED Rev*, vol. 5, no. 1, pp. 1:1–1:2, Jan. 2008.

[33] "Open Virtual Platforms." [Online]. Available: http://www.ovpworld.org/.

[34] "SimpleScalar LLC." [Online]. Available: http://www.simplescalar.com/.

[35] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 2000, pp. 83–94.

[36] "CPU Sim." [Online]. Available: http://www.cs.colby.edu/djskrien/CPUSim/.

[37] J. Campenhout, P. Verplaetse, and H. Neefs, "ESCAPE: Environment for the simulation of computer architectures for the purpose of education," p. 9, Dec. 1998.

[38] N. Brown, "HASE - a computer architecture simulation environment." [Online]. Available: http://www.icsa.inf.ed.ac.uk/research/groups/hase/.

[39] "MikroSim," *Wikipedia*. 14-Feb-2017.

[40] "SimNow™ Simulator," *AMD*. [Online]. Available: https://developer.amd.com/simnow-simulator/.

[41] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," p. 12.

[42] N. Binkert *et al.*, "The Gem5 Simulator," *SIGARCH Comput Arch. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[43] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[44] M. M. K. Martin *et al.*, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *SIGARCH Comput Arch. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[45] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[46] Trevor Carlson, Andreas Sandberg, David Black-Schaffer, and Erik Hagersten, "Full-System Simulation at Near-Native Speed."

[47] Andreas Sandberg and Ali Saidi, "Gem5 Virtual Machine Acceleration."

[48] "The ns-2 Network Simulator." [Online]. Available: https://www.isi.edu/nsnam/ns/.

[49] M. Stehlık, "Comparison of Simulators for Wireless Sensor Networks," p. 92.

[50] M. Korkalainen, M. Sallinen, N. Kärkkäinen, and P. Tukeva, "Survey of Wireless Sensor Networks Simulation Tools for Demanding Applications," in *2009 Fifth International Conference on Networking and Services*, 2009, pp. 102–106.

[51] "The ns-3 Network Simulator." [Online]. Available: https://www.nsnam.org/.

[52] "The JSim Network Simulator." [Online]. Available: http://www.physiome.org/jsim/.

[53] H. M. Ammari, Ed., *The Art of Wireless Sensor Networks: Volume 1: Fundamentals*. Berlin Heidelberg: Springer-Verlag, 2014.

[54] "NetSim Cisco Network Simulator & Router Simulator." [Online]. Available: http://www.boson.com/netsim-cisco-network-simulator.

[55] "OMNeT++ Discrete Event Simulator." [Online]. Available: https://omnetpp.org/.

[56] "The MiXiM framework." [Online]. Available: http://mixim.sourceforge.net/.

[57] M. D. Ilic, L. Xie, U. A. Khan, and J. M. F. Moura, "Modeling of Future Cyber-Physical Energy Systems for Distributed Sensing and Control," *IEEE Trans. Syst. Man Cybern. - Part Syst. Hum.*, vol. 40, no. 4, pp. 825–838, Jul. 2010.

[58] P. Tandon, J. Chang, R. G. Dreslinski, V. Qazvinian, P. Ranganathan, and T. F. Wenisch, "Hardware Acceleration for Similarity Measurement in Natural Language Processing," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, Piscataway, NJ, USA, 2013, pp. 409–414.

[59] 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)*, 2016, pp. 1–12.

[60] Mingyan Yu, Junjie Song, Fangfa Fu, Siyue Sun, and Bo Liu, "A Fast Timing-Accurate MPSoC HW/SW Co-Simulation Platform based on a Novel Synchronization Scheme," 2010. .

[61] 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)*, 2015, pp. 380–387.

[62] "Functional Mock-up Interface." [Online]. Available: http://fmi-standard.org/.

[63] J. R. Noseworthy, "The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations," in *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, 2008, pp. 259–268.

[64] "IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles," *IEEE Std 12782-1995*, pp. 1–20, Nov. 1996.

[65] A. L. Wilson and R. M. Weatherly, "The aggregate level simulation protocol: an evolving system," in *Proceedings of Winter Simulation Conference*, 1994, pp. 781–787.

[66] "HLA 1516 - Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Framework and Rules." [Online]. Available: https://standards.ieee.org/develop/project/1516.html.

[67] S. P. Griffin, E. H. Page, C. Z. Furness, and M. C. Fischer, "Providing Uninterrupted Training to the Joint Training Confederation (JTC) Audience During Transition to the High Level Architecture (HLA)."

[68] "Run-time infrastructure," *Wikipedia*. 20-Mar-2018.

[69] "Pitch pRTI – A Certified HLA RTI." [Online]. Available: http://www.pitchtechnologies.com/products/prti/.

[70] "The Portico Project." [Online]. Available: http://www.porticoproject.org/comingsoon/.

[71] C. Roth, O. Sander, M. Kühnle, and J. Becker, "HLA-based Simulation Environment for Distributed SystemC Simulation," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2011, pp. 108–114.

[72] "CERTI Project," 2015. [Online]. Available: http://savannah.nongnu.org/projects/certi.

[73] C. Roth, J. Meyer, M. Rückauer, O. Sander, and J. Becker, "Efficient Execution of Networked MPSoC Models by Exploiting Multiple Platform Levels," *Int J Reconfig Comput*, vol. 2012, pp. 6:6–6:6, Jan. 2012.

[74] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, "Distributed Simulation of Heterogeneous and Real-Time Systems," in *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*, 2013, pp. 55–62.

[75] C. Roth, O. Sander, and J. Becker, "Flexible and Efficient Co-simulation of Networked Embedded Devices," in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design*, New York, NY, USA, 2011, pp. 61–66.

[76] "Accellera SystemC wiki website." 05-Mar-2018.

[77] "IEEE 1666-2011 - IEEE Standard for Standard SystemC Language." [Online]. Available: https://standards.ieee.org/findstds/standard/1666-2011.html.

[78] N. Gopalakrishna, "Execution time analysis of audio algorithms," 2014.

[79] Pete Stevenson, "GEM5 mailing List. X86 full system dual," 2014. [Online]. Available: https://www.mail-archive.com/gem5-users@gem5.org/msg09897.html.

[80] Nikolaos Tampouratzis, "GEM5 mailing List. X86 full system dual Solution.," 2016. [Online]. Available: https://www.mail-archive.com/gem5-users@gem5.org/msg12680.html.

[81] Balázs Kiss, Gergő Hosszú, and Attila Szász, "COSSIM Public Deliverable D4.2. 'Initial prototype of security and robustness testing platform.'" .

[82] "GEM5 Checkpoints." [Online]. Available: http://www.m5sim.org/Checkpoints.

[83] "Ioctl - Control Device." [Online]. Available: http://man7.org/linux/man-pages/man2/ioctl.2.html.

[84] E. Vasilakis, I. Sourdis, V. Papaefstathiou, A. Psathakis, and M. G. H. Katevenis, "Modeling energy-performance tradeoffs in ARM big.LITTLE architectures," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[85] S. J. E. Wilton and N. P. Jouppi, "CACTI: an enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.

[86] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[87] A. B. Kahng, B. Li, L. S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Automation Test in Europe Conference Exhibition 2009 Design*, 2009, pp. 423–428.

[88] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, “A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies,” in *2008 International Symposium on Computer Architecture*, 2008, pp. 51–62.

[89] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing,” *ACM Trans Arch. Code Optim*, vol. 10, no. 1, pp. 5:1–5:29, Apr. 2013.

[90] A. S. Leon *et al.*, “A Power-Efficient High-Throughput 32-Thread SPARC Processor,” in *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, 2006, pp. 295–304.

[91] U. G. Nawathe, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, “Implementation of an 8-Core, 64-Thread, Power-Efficient SPARC Server on a Chip,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 6–20, Jan. 2008.

[92] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang, “A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache,” in *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, 2006, pp. 315–324.

[93] A. Jain *et al.*, “A 1.2 GHz Alpha microprocessor with 44.8 GB/s chip pin bandwidth,” in *2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No.01CH37177)*, 2001, pp. 240–241.

[94] “Intel Atom® Processors,” *Intel*. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/atom.html.

[95] A. Ltd, “ARM Cortex-A9,” *ARM Developer*. [Online]. Available: https://developer.arm.com/products/processors/cortex-a/cortex-a9.

[96] *cMcPAT: A modified version of McPAT for the COSSIM framework. The code is based on McPAT v1.3 and integrates with cgem5.* H2020 COSSIM, 2018.

[97] “GEM5ToMcPAT.” [Online]. Available: https://bitbucket.org/dskhudia/gem5tomcpat.

[98] prajithrg, *Modified vanilla McPAT application to perform batch Runtime Dynamic Power computation*. 2017.

[99] "GEM5 - Trace Based Debugging." [Online]. Available: http://www.gem5.org/Trace_Based_Debugging.

[100] "1516.1-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Federate Interface Specification - IEEE Standard." [Online]. Available: https://ieeexplore.ieee.org/document/5557728/.

[101] "IEEE Standard for Modeling and Simulation (M amp;S) High Level Architecture (HLA)– Framework and Rules," *IEEE Std 1516-2010 Revis. IEEE Std 1516-2000*, pp. 1–38, Aug. 2010.

[102] "HLA Federation Managment." [Online]. Available: http://www.cs.cmu.edu/afs/cs/academic/class/15413-s99/www/hla/doc/rti_synopsis/05-Federation_Management/Federation_Management.html.

[103] E. A. Lee, "Heterogeneous Actor Modeling," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, New York, NY, USA, 2011, pp. 3–12.

[104] 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.

[105] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[106] W. Deegan, "SCons: A software construction tool," *SCons*. [Online]. Available: https://scons.org/.

[107] "Gentoo Linux." [Online]. Available: https://www.gentoo.org/get-started/about/.

[108] "BusyBox Operating System," *Wikipedia*. 15-Mar-2018.

[109] A. Saidi and A. Sandberg, "Accelerating Simulation with Virtual Machines."

[110] "Network Performance Benchmark - Netperf," 2015. [Online]. Available: https://hewlettpackard.github.io/netperf/.

[111] "BSD Sockets Interface Programmer's Guide." .

[112] "IEEE Standard for Ethernet," *IEEE Std 8023-2015 Revis. IEEE Std 8023-2012*, pp. 1–4017, Mar. 2016.

[113] Jonathan Corbet, "Large Receive Offload." [Online]. Available: https://lwn.net/Articles/243949/.

[114] "Tecnalia KUBIK: Intelligent Energy Building." .

[115] M. GUARNERA, "Visual Search - Computer Vision @ STMicroelectronics," p. 14, 2012.

[116] M. Paracchini, M. Marcon, E. Plebani, and D. P. Pau, "Visual Search of multiple objects from a single query," in *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2016, pp. 41–45.

[117] D. Martino, Y. Shen, M. Paracchini, M. Marcon, E. Plebani, and D. P. Pau, "Accurate cyber-physical system simulation for distributed visual search applications," in *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, 2017, pp. 1–5.

[118] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York: Wiley, 1991.

[119] T. Schreiber, "Measuring Information Transfer," *Phys. Rev. Lett.*, vol. 85, no. 2, pp. 461–464, Jul. 2000.

[120] C. H. Whitson and M. L. Michelsen, "The negative flash," *Fluid Phase Equilibria*, vol. 53, pp. 51–71, Dec. 1989.

[121] "The Newton-Raphson Method."

[122] P. Malakonakis, K. Georgopoulos, A. Ioannou, L. Lavagno, I. Papaefstathiou, and I. Mavroidis, "HLS Algorithmic Explorations for HPC Execution on Reconfigurable Hardware - ECOSCALE," 2018, pp. 724–736.