Fast Cycle-approximate Simulation Techniques for Manycore Architecture Exploration

Anastasiia Butko

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par **Université Montpellier**

Préparée au sein de l'école doctorale **Information, Structures, Systèmes (I2S)**

Et de l'unité de recherche **Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)**

Spécialité : **Microélectronique**

Présentée par **BUTKO Anastasiia**

Fast Cycle-approximate Simulation Techniques for Manycore Architecture Exploration

Soutenue le 1 decembre 2015 devant le jury composé de

M. Leonardo SOARES INDRUSIAK, Senior Lecturer, University of York, UK	Rapporteur
M. Jean-François MEHAUT, Professeur, Laboratoire d'Informatique de Grenoble, FRANCE	Rapporteur
M. Yann THOMA, Professeur, Institut REDS at HEIG-VD, Geneva, SWITZERLAND	Examinateur
M. Chris ADENIYI-JONES, Principal Engineer, ARM Ltd., Cambridge, UK	Examinateur
M. Michel ROBERT, Professeur, Université de Montpellier, FRANCE	Examinateur
M. Gilles SASSATELLI, Directeur de Recherche au CNRS, LIRMM/Université de Montpellier, FRANCE	Directeur de Thèse
M. Abdoulaye GAMATE, Directeur de Recherche au CNRS, LIRMM/Université de Montpellier, FRANCE	Co-encadrant

UNIVERSITÉ MONTPELLIER

Abstract

Since the computational needs precipitously grow each year, HPC technology becomes a driving force for numerous scientific and consumer areas. The most powerful supercomputer has been progressing from TFLOPS to PFLOPS throughout the last ten years. However, the extremely high power consumption and therefore the high cost pushed researchers to explore more energy-efficient technologies, such as the use of low-power embedded SoCs.

The evolution of emerging manycore systems, forecasted to feature hundreds of cores by the end of the decade calls for efficient solutions for the design space exploration and debugging. Available industrial and academic simulators differ in terms of simulation speed/accuracy trade-offs. Cycle-approximate simulators are popular and attractive for architectural exploration. Even though enabling flexible and detailed architecture evaluation, cycle-approximate simulators entail slow simulation speeds, thereby limiting their scope of applicability for systems with hundreds of cores. This calls for alternative approaches capable of providing high simulation speed while preserving accuracy that is crucial to architectural exploration.

In this thesis, we evaluate cycle-approximate simulation techniques for fast and accurate exploration of multi- and manycore architecture exploration. Expecting to significantly reduce simulation time still preserving the accuracy at the cycle-approximate level, we propose a hybrid trace-oriented approach to enable flexible manycore architecture simulation. We design a set of simulation techniques to overcome the main weaknesses of the trace-oriented approach. The trace synchronization technique aims to manage control and data dependencies arising from the abstraction of processor cores. The trace replication technique is proposed to simulate manycore architectures using a finite set of pre-collected traces. The computation phase scaling technique is designed to enable flexible switching between multiple processor models without considering microarchitectural difference but taking into account the computation speed ratio.

Based on the proposed simulation environment, we explore several manycore architectures in terms of performance and energy-efficiency trade-offs.

Keywords: High-performance computing, energy-efficiency, manycore, heterogeneous, big.LITTLE, modeling, gem5, trace-driven.

Résumé

Le calcul intensif joue un rôle moteur de premier plan pour de nombreux domaines scientifiques. La croissance en puissance crôte des supercalculateurs a évolué du téraflops au pétaflops en l'espace d'une décennie. Toutefois, la consommation d'énergie associée extrômement élevée ainsi le coût associé ont motivé des recherches vers des technologies plus efficaces énergétiquement comme l'utilisation de processeurs issus du domaine des systèmes embarqués à faible puissance. Selon les prévisions, les systèmes multicoeurs émergents seront constitués de centaines de coeurs d'ici la fin de la décennie. Cette évolution nécessite des solutions efficaces pour l'exploration de l'espace de conception et le débogage. Les simulateurs industriels et académiques disponibles à ce jour diffèrent en termes de compromis entre vitesse de simulation et précision. Les simulateurs quasi cycle-précis sont populaires et attrayants pour l'exploration architecturale. En outre, bien que permettant une évaluation flexible et détaillée de l'architecture, les simulateurs quasi cycle-précis entraînent des vitesses de simulation lentes ce qui limite leur champ d'application pour des systèmes avec des centaines de cœurs. Cela exige des approches alternatives capables de fournir des simulations rapides tout en préservant une précision élevée ce qui est cruciale pour l'exploration architecturale.

Dans cette thèse, des modèles d'architectures multicœurs complexes ont été développés et évalués en utilisant des systèmes de simulation quasi cycle-précis pour l'exploration de la performance et de la puissance. Sur cette base, une approche hybride orientée traces d'exécution a été proposée pour permettre une exploration rapide, flexible et précise des architectures multicoeurs à grande échelle. Sur la base de l'environnement de simulation proposé, plusieurs configurations de systèmes manycoeurs ont été construites et estimées en évaluant le passage à l'échelle des performances. Enfin, des configurations alternatives d'architectures multicoeurs hétérogènes ont été proposées et ont montré des améliorations significatives en termes d'efficacité énergétique.

Mots-clés: calcul à haute performance, efficacité énergétique, multicoeurs, hétérogène, big.LITTLE, modélisation, gem5, simulation orientée trace.

Acknowledgements

The acknowledgements will be present in the final version of this thesis.

Contents

List of Figures

List of Tables

Abbreviations

ALU	**A**rithmetic **L**ogic **U**nit
API	**A**pplication **P**rogramming **I**nterface
ASV	**A**daptive **S**upply **V**oltage
BSD	**B**erkeley **S**oftware **D**istribution license
CCI	**C**ache **C**oherence **I**nterface
CMP	**C**hip **M**ulti**P**rocessor
CPU	**C**entral **P**rocessing **U**nit
CSM	**C**entralized **S**hared **M**emory
DDR	**D**ouble **D**ata **R**ate memory
DES	**D**iscrete-**E**vent **S**imulation
DSM	**D**istributed **S**hared **M**emory
DVFS	**D**ynamic **V**oltage and **F**requency **S**caling
EtoS	**E**nergy-to-**S**olution
FFT	**F**ast **F**ourier **T**ransform
FPU	**F**loating-**P**oint **U**nit
FS	**F**ull **S**ystem
GPU	**G**raphics **P**rocessing **U**nit
GTS	**G**lobal **T**ask **S**cheduling
HMP	**H**eterogeneous **M**ulti**P**rocessing
HPC	**H**igh-**P**erformance **C**omputing
HPL	**H**igh-**P**erformance **L**inpack
ILP	**I**nstruction **L**evel **P**arallelism
IPC	**I**nstructions **P**er **C**ycle
IPS	**I**nstructions **P**er **S**econd
ISA	**I**nstruction **S**et **A**rchitecture

ISS	**I**nstruction **S**et **S**imulator
JIT	**J**ust-**I**n-**T**ime
LPDDR	**L**ow **P**ower **D**ouble **D**ata **R**ate memory
MESI	**M**odified **E**xclusive **S**hared **I**nvalid protocol
MJPEG	**M**otion **J**oint **P**hotographic **E**xperts **G**roup
MPI	**M**essage **P**assing **I**nterface
MPSoC	**M**ulti**P**rocessor **S**ystem-on-**C**hip
NA	**N**ot **A**vailable
NoC	**N**etwork-on-**C**hip
OpenMP	**O**pen **M**ulti-**P**rocessing
OS	**O**perating **S**ystem
PE	**P**rocessing **E**lement
PGAS	**P**artitioned **G**lobal **A**ddress **S**pace
PoP	**P**ackage on **P**ackage
Pthread	**P**OSIX thread
RAM	**R**andom **A**ccess **M**emory
RTL	**R**egister-**T**ransfer **L**evel
SCU	**S**noop **C**ontrol **U**nit
SMP	**S**ymmetric **M**ulti**P**rocessor
SMT	**S**imultaneous **M**ulti**T**hreading
SoC	**S**ystem-on-**C**hip
SPALSH-2	**S**tanford **P**aralle**L** **A**pplications of **SH**ared memory
SPM	**S**cratch**P**ad **M**emory
TA	**T**race **A**rbiter
TBS	**T**ime-**B**ased **S**ampling
TCI	**T**race **C**ollection **I**nterface
TD	**T**race-**D**riven
TI	**T**race **I**njector
TLB	**T**ranslation **L**ook aside **B**uffers
TLM	**T**ransaction-**L**evel **M**odeling
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
XML	e**X**tensible **M**arkup **L**anguage

Chapter 1

Introduction

1.1 Context and Motivation

High-performance computing is a field of computational science intended to perform particularly difficult tasks in the shortest time. Supercomputer is the principal component of an high-performance computing system and has been designed to solve complex high-level computational issues in various scientific domains, e.g. climate research, quantum mechanics, chemical and biological modeling, airplane and spacecraft dynamics simulations, etc. Since the computational needs quickly grow each year, HPC technology becomes a driving force for numerous scientific and consumer areas.

The first supercomputer was built in 1960s by Seymour Cray and since then it undergone a lot of changes and improvements. While at the beginning, supercomputers contained only a few specific purpose processors, current massively parallel samples consist of tens of thousands processors. The TOP500 is a list which ranks the most powerful supercomputers in the world since 1993 [1]. Figure 1.1 illustrates performance development reported by the TOP500 list [2]. The most powerful supercomputer, characterized by the dotted-curve labeled "N = 1", has been progressing from GFLOPS to PFLOPS throughout last twenty years. The current leader of the list is the Tianhe-2 with 54.9 PFLOPS peak performance.

The next frontier in high-performance computing refers to computing systems performing at least one exaFLOPS, i.e. a quintillion floating point operations per second. According to projections, such a system is expected by 2018 with a 20MW power budget

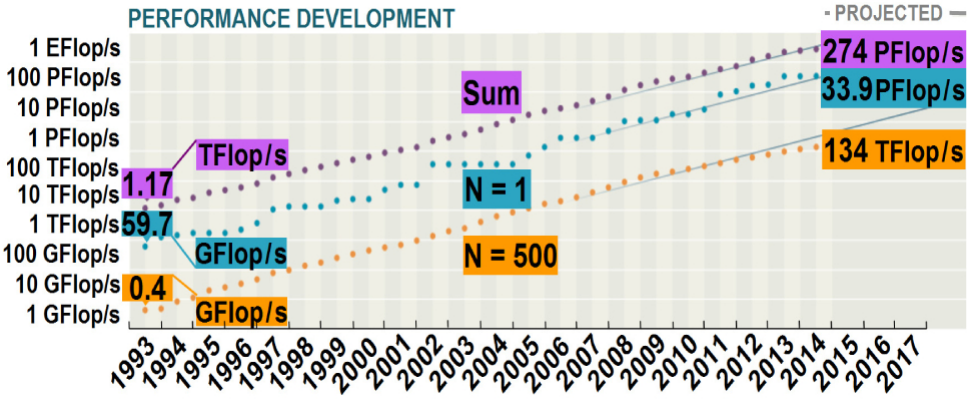

FIGURE 1.1: Performance development in the TOP500 list [2].

[3]. Future exascale computing systems face a number of challenges, such as system reliability, energy-efficiency, scalability of software, etc. Energy consumption of HPC systems is an important research field that calls for intense exploration of alternative 'green' technologies.

The Green500 list contains the most energy-efficient supercomputers [4]. Unlike the TOP500 list, it shows the computation rate delivered by a supercomputer per watt. Figure 1.2 shows the list of supercomputers ranked from 1 to 5 in June 2015 [4]. The current leader of the list is the Shoubu supercomputer located at the RIKEN Advanced Institute for Computational Science in Japan. This supercomputer as well as other top representatives is based on the Intel Xeon E5 processor and the InfinityBand communication technology.

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	7,031.58	RIKEN	Shoubu - ExaScaler-1.4 80Brick, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SC	50.32
2	6,842.31	High Energy Accelerator Research Organization /KEK	Suiren Blue - ExaScaler-1.4 16Brick, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband, PEZY-SC	28.25
3	6,217.04	High Energy Accelerator Research Organization /KEK	Suiren - ExaScaler 32U256SC Cluster, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, PEZY-SC	32.59
4	5,271.81	GSI Helmholtz Center	ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150	57.15
5	4,257.88	GSIC Center, Tokyo Institute of Technology	TSUBAME-KFC - LX 1U-4GPU/104Re-1G Cluster, Intel Xeon E5-2620v2 6C 2.100GHz, Infiniband FDR, NVIDIA K20x	39.83

FIGURE 1.2: The Green500's energy-efficient supercomputers ranked from 1 to 5 in June 2015 [4].

Nevertheless, to achieve the expected one ExaFLOPS per 20MW a future green supercomputer must provide the 50 GFLOPS/W performance rate. That is seven times

higher than the current most energy-efficient supercomputer is able to produce. Considering the magnitude of the task, researchers have turned to explore low-power embedded system-on-chips as an attractive solution to apply for energy-efficient supercomputing [5].

Being designed to minimize the power consumption, embedded processors feature poor performance versus supercomputer's nodes. The idea to combine multiple SoCs in a large-scale clustered system looks unrealistic due to the potentially vast interconnect traffic. Therefore, embedded processor performance is a key factor to make a substantial progress in that direction.

Looking for ways to improve the performance, embedded systems switched from the operating frequency scaling towards the increasing of on-chip parallelism. The amount of parallelism on a single processor is forecasted to reach hundreds of cores by the end of the decade. Moreover, several alternative architecture configurations arouse the researcher interest revealing a number of budding opportunities. Among them, single-ISA heterogeneous multicore technology [6], which allows operating system to explicitly manage processor load balancing for fine control over performance and power consumption. In the mobile market, several SoC platforms operating on that principle already exist, such as Nvidia Tegra 3/4 SoC [7] and Samsung Exynos 5/7 Octa SoC [8] based on ARM big.LITTLE technology. Despite some important contributions to heterogeneous multiprocessing [6] [9] [10] [11] [12], an adequate solution has not been proposed yet and remains a wide area for further exploration.

Exploration of computer architectures is mainly carried out by means of simulation. Architectural simulators are widely used in academic and industry research. They allow avoiding costly prototyping of real hardware and provide usable environment for design trade-off evaluation. There are three key aspects, which define a computer architecture simulator: accuracy, speed and flexibility.

The accuracy level is determined by the error that the given simulation model produces compared to the reference system. The validation process aims to identify the error value and is a substantiation that the model within its domain of applicability generates results with a satisfactory level of veracity [13]. As high simulation error may lead to wrong conclusions, exploration model validation is strongly required [14].

The simulation speed depends on the abstraction level, which extends between particularly slow cycle-accurate models and imprecise analytical models with simulation time close to real execution. Simulation time remains the major obstacle for efficient architecture exploration forcing researchers to raise the level of abstraction.

In recent years, the computer architecture complexity has grown rapidly, caused by the increasing number of cores, advanced interconnect and memory hierarchy. The desired architectural simulator is expected to provide the environment flexible enough to explore a complete set of design configurations. Debugging capabilities also plays an important role in architecture simulator usability.

Among the existing architecture simulators, cycle-approximate models are of particular interest providing suitable level of accuracy and a wide modeling scope. However, in the context of large-scale manycore architecture exploration, traditional cycle-approximate simulators yet entail slow simulation speed. This calls for alternative approaches capable of providing high simulation speed while preserving the sufficient level of accuracy.

There are various techniques designed to reduce simulation time. They can be classified into two fundamental groups. The first group focuses on increasing computational power [15], i.e. increasing the number of simulated events per second. It is usually achieved by distributing the simulation across multiple host machines. Because of its concurrent nature, distributed simulation faces synchronization issues. To insure simulation consistency, a process running on one host machine might take into consideration the past of other processes. This issue can be addressed either by using looking mechanisms or by allowing synchronization error and applying a rollback procedure [16].

The second group includes approaches designed to reduce the number of simulation events required for accurate results [15]. Trace-driven approach is one of the commonly used solutions. Depending on the target exploration level, it allows abstracting a large number of unnecessary events yet providing relevant results. Namely, for multicore architecture simulation an approach of processor computation abstraction is usually applied. However, such a trace-driven approach has several considerable limitations.

Due to the abstraction of operating system and application execution, synchronization mechanisms dedicated to guarantee data and control flow consistency are left out of

consideration. Meanwhile these mechanisms are crucial for manycore architecture exploration.

One more limitation concerns data-dependent applications, i.e. applications in which control flow and memory access pattern are determined by the input datasets. Data-dependent conditionals of such applications cannot be reproduced in the correct order by a trace-driven simulation.

Generally, trace-driven simulation is easy to implement. Nevertheless, the need to deal with the inherent limitations affect its usability. In addition, when computation phase abstraction is used, simulation speedup largely depends on the computation-communication ratio, which is determined by the application nature.

1.2 Thesis objectives and contributions

This thesis is conducted within the Mont-Blanc European project [17]. The main goal of the project is to design a new energy-efficient supercomputer based on the low-power embedded technologies.

This thesis focuses on two interconnected research directions. The first direction aims to develop a fast, flexible and accurate environment for efficient architecture simulation. The second direction is dedicated to multi- and manycore architectures exploration using the proposed simulation environment.

Objective 1 The first objective of this thesis is to answer the following questions:

> **Q:** How accurate are performance and power models implemented in cycle-approximate full system simulation frameworks? What are the main sources of error in these models? Can these models be used to realistically predict important exploration metrics?

For this purpose, we implement performance models of two multicore SoCs in gem5 cycle-approximate simulation framework [18] and a power model in McPAT simulation framework [19]. These models are then validated against the real hardware. Based on the simulation results we analyze sources of error in the proposed models and summarize whether these models can be used in architecture exploration and under what conditions.

The main contributions of this work are the first-published evaluation and detailed analysis of multicore performance/power models implemented in gem5/McPAT simulation frameworks. Moreover, the evaluated models are made to be freely available online for the research community. This work is supported by the following publications:

- [20] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli. Accuracy evaluation of gem5 simulator system. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7, July 2012
- [21] A. Butko, A. Gamatié, G. Sassatelli, L. Torres, and M. Robert. Design exploration for next generation high-performance manycore on-chip systems: Application to big.little architectures. In *VLSI (ISVLSI), 2015 IEEE Computer Society Annual Symposium on*, July 2015

Objective 2 The second objective of this thesis is to develop a flexible framework enabling fast and yet accurate simulation for manycore architecture exploration. Under this study, we address the following questions:

> **Q:** How can simulation time can be reduced while preserving the accuracy level? What are the limitations of the existing approaches and how can they be avoided? Is the proposed approach efficient enough to enable future manycore architecture exploration?

Our contribution differs from all previous works by proposing a novel hybrid trace-oriented simulation approach suitable for an efficient exploration of entire manycore system. It includes fully simulated memory infrastructure, trace synchronization technique, trace replication technique and computation phase scaling technique. Altogether, the proposed approach demonstrates significantly reduced simulation time and high accuracy level compared to reference full system simulation. Also, the source code of the proposed approach implementation is freely available on the project webpage [22]. This work is supported by the following publication:

- [23] A. Butko, R. Garibotti, L. Ost, V. Lapotre, A. Gamatié, G. Sassatelli, and C. Adeniyi-Jones. A trace-driven approach for fast and accurate simulation of manycore architectures. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 707–712, Jan 2015

In addition, the proposed approach has been applied to investigate the performance scalability of alternative memory mappings techniques running on compute accelerators. This work has been presented in the PhD thesis of R. Garibotti [24].

Objective 3

The third objective of this thesis is to explore multi- and manycore architectures in terms of performance and energy-efficiency trade-offs. At this stage, the following questions are raised:

> **Q:** What are the promising directions in computer architecture design? How can the existing programming models be used to benefit from heterogeneous multi- and manycore architectures? Which configurations of single-ISA heterogeneous architectures can significantly enhance the system energy-efficiency?

In this context, we consider single-ISA heterogeneous multicore architectures as a promising solution. Based on the detailed analysis and using the described simulation environment, we propose alternative architecture configurations, which demonstrate significant enhancements in energy-efficiency. The publications related to this work are currently under submission.

1.3 Thesis organization

This thesis is organized as follows: Chapter 2 presents the state-of-the-art concerning computer architecture simulation techniques, existing simulation frameworks, accuracy evaluations of performance and power models and existing solutions for large-scale many-core architecture simulations. In Chapter 3 the implementation, validation and analysis of performance and power models are described. The hybrid trace-driven approach, the synchronization and replication techniques and also the computation phase scaling technique are described in Chapter 4. The detailed analysis of existing ARM big.LITTLE SoC and exploration of relevant alternative configurations are presented in Chapter 5. Chapter 6 draws conclusions and discusses possible directions for future work.

Chapter 2

State-of-the-art

This chapter presents a survey of the areas relevant for the thesis. Section 2.1 provides a general background of computer architecture simulation techniques. Section 2.2 describes the existing simulation frameworks. The accuracy evaluations of performance and power models are presented in Section 2.3. In Section 2.4 we discuss the existing solutions for large-scale manycore architecture simulations.

2.1 Computer architecture simulation

The computer architecture simulator or architectural simulator is a software tool that models a computer architecture in order to predict a required set of output metrics, e.g. performance, power, architectural statistics, etc. Architectural simulators have been widely used since the beginning of the computer era for many reasons. It allows evaluating various architecture designs without building expensive real hardware systems. Unlimited access to architecture component description provides the possibility to evaluate non-existing implementations. Moreover, a large set of output metrics can be easily obtained by static or dynamic collection. This also greatly simplifies the debugging of the evaluated system.

Time progress management. Talking about simulation, a notion of *simulation time* appears and it inherently differs from the continuous time of reality. There are three conceptual time advancement strategies: activity scanning, event scheduling and process interaction [25].

Under the activity scanning, time is divided into tiny increments and a simulator continuously tracks system dynamics over time. Naturally, for most time increments, no changes in system state will happen and activity tracking will needlessly waste simulation time. For this reason, the activity scanning strategy is very time consuming. Several works however implement this time simulation strategy [26] [27].

In event scheduling which is also known as *event-driven simulation*, time advances discontinuously from one event to another. This time management strategy is more efficient because it does not require computations to be performed during inactive periods. The credibility of the algorithm of next-event scheduling determines the simulation accuracy.

For process interaction paradigm each simulation activity is modeled by a process. Often the process-oriented view is internally implemented on the top of an event-oriented framework [28]. Compared to the event scheduling simulation, it is much slower, and more difficult to implement and debug [27].

Simulation scope. The other way to classify computer architecture simulators is based on simulation scope. Simulation scope refers to the architecture exploration level and can cover either microarchitecture or entire system simulation.

An *instruction set* or *instruction set architecture* specifies a set of basic instructions that a processor supports. It also includes the size of main memory, number of registers and instruction format.

An *instruction set simulator* is used to emulate the processor behavior by using the instruction set and maintaining internal variables which represent processor registers [29].

A *full system simulation* models the entire system, making target software, e.g. OS and application, believe that it is running on a real physical hardware.

Abstraction level. Depending on the level of details, architectural simulators are also classified on different abstraction levels. The golden point design is an approach to specify architectures in a very detailed level using hardware description languages like VHDL or Verilog. This limited RTL abstraction gives high timing accuracy, but in turn poses severe limitations on design space exploration, extremely time consuming and difficult to debug.

Cycle-accurate simulator is a software model, usually coded in a high-level programming language. It reproduces the cycle-by-cycle system behavior. Furthermore, cycle-accurate models facilitate analysis iterations around various architectural options as well as software execution, which gives flexibility to explore more features than in a low-level abstraction model. Such simulators also may contain components with different accuracy levels. In this case, it is referred as *qausi cycle-accurate* or *cycle-approximate* simulator.

Following this direction, there are *function-accurate* models, *instruction-accurate* models and others high-level abstraction models, which result in faster simulation at the cost of a loss of accuracy.

An *analytical model* is a mathematical model which contains a set of equations describing the performance of a computer system. Despite the low level of accuracy, analytical modeling remains a highly practical method of analysis because of its relative simplicity and high simulation speed [30].

2.2 Simulation frameworks

This section provides a survey of the most popular computer architecture simulators, according to different criteria: accuracy, simulation speed, supported processor architectures, licensing, development activity and other simulation features.

PTLsim [31] is a cycle-accurate full system x86 microprocessor simulator that has an out-of-order pipelined model. PTLsim also supports modeling of multi-processor or simultaneous multithreading machines. The error of PTLsim compared to the reference silicon (AMD Athlon 64 at 2.2 GHz) is within 5% across major parameters. Micro-operation (uops) metric differs in 31%. Simulation speed is around 270 KIPS. PTLsim presents two main drawbacks, only x86 architectures are supported and the tool suite is not actively maintained anymore.

MARSSx86 [32] is based on PTLsim with extensive enhancements for improved simulation accuracy and performance. It complements PTLsim key features with unmodified operating system running, detailed models for coherent caches, on-chip interconnections

and advanced multiprocessing simulation. Simulation speed is around 200 KIPS. Accuracy validation results are not available. As PTLsim, MARSS focuses only on x86 architecture.

Simics is a functionally-accurate full system simulator that enables unmodified target software (e.g. operating system, applications) to run on the virtual platform similar to the physical hardware [33]. Simics supports a wide range of processor architectures (e.g. Alpha, ARM, MIPS, PowerPC, SPARC, x86), as well as operating systems (e.g. Linux, VxWorks, Solaris, FreeBSD, QNX, RTEMS). Simics is composed of an instruction-set simulator, memory-management units models, as well as all memories and devices found in the memory map of the processors. Simics has two main disadvantages, it is not claimed to be cycle-accurate and a commercial license is required (marketed by Wind River Systems [34]). Besides the general functional-accurate class, other information about accuracy validation is not available. Simulation speed is around 300 KIPS for a detailed out-of-order architecture simulation and 6.6 MIPS for a fast simulation.

Flexus [35] is a computer architecture simulator based on Virtutech Simics. In contrast to Simics, it is open source but supports only SPARC ISA. Simulation speed varies from 20-25 KIPS for a detailed simulation to 30-90 MIPS for a fast simulation. Accuracy validation results are not available.

SimpleScalar is an open source infrastructure for simulation and architectural modeling. It supports several processor architectures including Alpha, ARM, PowerPC and x86. Moreover, it features a large range of CPU models, which varies from simple unpipelined processors to detailed dynamically scheduled microarchitectures with multiple-level memory hierarchies [36]. The IPC accuracy had been validated against the SA-1110 platform and showed 3% of error. It provides two simulation modes, which differ in details and speed. Sim-OutOrder detailed mode runs at 350 KIPS, Sim-Fast mode runs at 10 MIPS simulation speed. SimpleScalar features were widely improved in the past, but it seems that both development and support have slowed down significantly. Indeed, the last update is more than four years ago at the time of this writing. Multiple SimpleScalar extensions have been implemented to support multiprocessing/multithreading and thermal models, e.g. SSMT [37], M-Sim [38], SMTSim [39], HotSpot [40].

Multi2Sim is a CPU-GPU heterogeneous computing simulation framework [41]. It has been developed integrating models for superscalar, multithreaded, multicore CPUs/GPU

architectures, as well as cache coherence, multi-level cache hierarchy and interconnection networks. ARM, MIPS, NVIDIA Fermi and AMD GPU models are supported by Multi2Sim. It is classified as application-only emulator that focuses on user-level application execution, removing operating system and device drivers. Thereby, simulation speed is around tens of MIPS. However, neither accuracy validation nor simulation speed results are available.

ESESC, i.e. enhanced SESC [42], is an open source implementation of time-based sampling. Authors claim that the proposed framework is the first to enable sampling in simulation of multicore processors with virtually no limitation in terms of application type, number of cores, homogeneity or heterogeneity of the simulated configuration. It is also the first TBS to enable integrated power and temperature evaluation in statistically sampled simulation of multicore systems [43]. The reported error of performance evaluation is within 5% compared to full system simulation. The power and maximum temperature model errors are 5.5% and 2.4% respectively. The sampling technique allows up to 9 MIPS of simulation speed to be reached. Actually, only ARM architecture is available in the ESESC simulator.

gem5 [18] is a modular discrete event-driven full system simulator, under BSD license. This simulator supports different instruction set architectures, such as Alpha, ARM, x86, SPARC, PowerPC and MIPS. The simulator provides a flexible, modular simulation system that makes it possible exploring multiprocessor architecture features by offering a diverse set of CPU models, system execution modes, and memory system models. Moreover, this simulator has an active development and support team. Simulation speed is classified into three classes depending on the simulation mode: 1 KIPS for detailed simulation, 25 KIPS for simplified timing simulation and up to 5 MIPS for fast simulation, e.g. emulation. At the beginning of this thesis, no material has been published that reports gem5 accuracy in terms of performance estimation.

Table 2.1 summarizes the presented simulation frameworks according to such criteria as ISAs support, accuracy level, simulation speed and key simulation and exploration features.

TABLE 2.1: Simulation frameworks comparison.

Simulator	**ISAs**	**Accuracy** (Error)	**Speed**	**Key Features**						
				FS	**OS**	**L**	**MA**	**MP**	**C**	**M**
PTLsim	x86	CA Cycles -4.3% Uops - 30.99%	270 KIPS	✓	*	✓	✓	*	*	*
MARSS b.o. PTLsim	x86	CA NA	200 KIPS	✓	✓	✓	✓	✓	✓	✓
Simics	Alpha ARM MIPS PowerPC SPARC x86	FA N/A	F: 6.6 MIPS D: 300 KIPS	✓	✓	✗	✓	✓	*	*
Flexus b.o. Simics	SPARC	FA N/A	F: 30-60 MIPS D: 20-25 KIPS	✓	✓	✓	✓	✓	✓	✓
SimpleScalar	Alpha ARM PowerPC x86	CA IPC - 3%	F: 10 MIPS D: 350 KIPS	✗	✗	✓	*	✓	✓	*
Multi2Sim	ARM MIPS x86 GPUs	CA N/A	N/A	✗	✗	✓	✓	✓	✓	✓
ESESC b.o. SESC	ARM	CA FS ± 11%	E: 90 MIPS F: 9 MIPS D: 500 KIPS	✗	✗	✓	✓	✓	✓	*
gem5	Alpha ARM MIPS PowerPC SPARC x86 GPUs	CA N/A	E: 5 MIPS F: 25 KIPS D: 1 KIPS	✓	✓	✓	✓	✓	✓	✓

CA - cycle-accurate, FA - function-accurate, N/A - not available

b.o. - based on, E - emulation, F - fast, D - detailed

'✓' - fully supported, '*' - partially supported, '✗' - not supported

FS - full system, **OS** - operating system, **L** - licensing, **MA** - microarchitecture,

MP - multiprocessing, **C** - cache hierarchy, **M** - memory infrastructure

Analyzing the summary table, we observe a considerable advantage of the gem5 simulation framework. Fully supporting all key features, it provides seven ISAs and three simulation speed modes. The only characteristic that has been unclear at the beginning of this thesis was the accuracy evaluation.

2.3 Accuracy evaluation of gem5 and McPAT simulation frameworks

The accuracy of the simulation framework is a critical aspect for design space exploration. The lack of accuracy knowledge may lead to wrong conclusions. Not only the total runtime error is important, but also the detailed analysis of error sources [14]. In this section, we present an overview of the previously published works as well as works that are based on the proposed accuracy evaluation and have been published throughout the advancement of this thesis.

Authors in [44] evaluate the accuracy of the M5 full system simulator for TCP/IP based network-intensive workloads, using only two benchmarks that were executed on a single Alpha CPU model. By using a relatively imprecise model they achieve reasonable accuracy - the network bandwidth mismatch against real system is reported within 15%.

Authors in [45] design a gem5 model of CoreTile Express system-on-chip and estimate the accuracy of Cortex-A15 core, memory system and interconnect. They deeply explore the microarchitectural simulation for the homogeneous dual-core system. This work has been done in collaboration with ARM Research in Austin, TX. By thoroughly tuning the ARM CPU model, authors achieve a mean absolute percentage runtime error of 15% (SPEC CPU2006 and PARSEC benchmarks) and an error of 20% on average for several key microarchitectural metrics. Authors conclude that these errors are acceptable and are not a hindrance to evaluating research ideas.

The work presented in [46] deals with the calibration and simulation of Cortex-A8 and Cortex-A9 cores in gem5. A comparison in terms of execution time is achieved against a real hardware execution and based on ten benchmarks. Authors claim that their core models are more accurate than similar microarchitectural simulators. For both models, the average absolute error is within 17%.

A similar study has been achieved for Cortex-A7 and Cortex-A15 cores in [47] by focusing on the microarchitectural simulation of these cores. The gem5 and McPAT frameworks have been combined to validate area and energy/performance trade-offs against the published datasheet information. However, this work does not aim to evaluate multicore configurations. It only demonstrates the difference between Cortex-A7 and Cortex-A15 cores running single-threaded applications.

Xi et al. in [48] present the first assessment of McPAT's core power and area models. The results show that McPAT's models can provide significant error due to abstraction and modeling errors. They also discuss ways to avoid such errors and improve the architectural power models.

To summarize the related work on accuracy evaluation of gem5 and McPAT, Figure 2.1 depicts the publications appeared throughout the time. This thesis works are also presented in the time scale. In [20], we evaluated the accuracy of the performance model for ARM multicore processors exploration. This work is the first that reports and discusses the accuracy of gem5 framework in terms of performance estimation. The work published in [21] addresses performance model validation of single-ISA heterogeneous big.LITTLE architecture. It covers the comparison of multiple Cortex-A7 in-order implementations as well as multicore simulations in heterogeneous multiprocessing mode. The future publication, which is currently under submission complements the ARM big.LITTLE exploration with power modeling.

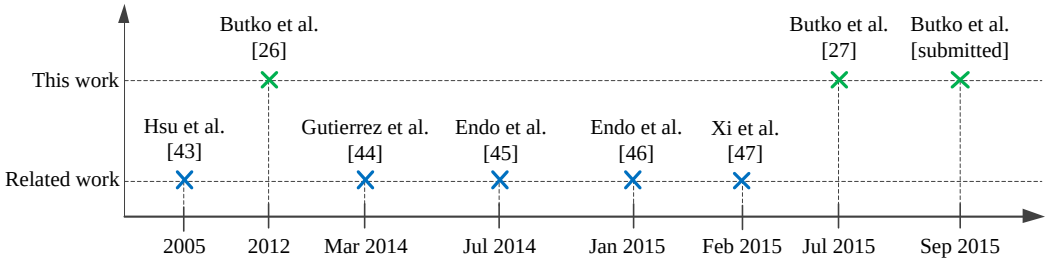

FIGURE 2.1: The related work on accuracy evaluation.

2.4 Approaches to accelerate the simulation

Traditional cycle-approximate simulation frameworks entail slow simulation speed, therefore limiting its scope of applicability for systems with hundreds of cores. This calls for alternative approaches capable of providing high simulation speed while preserving accuracy. Over the past decades, various simulation speedup approaches have been presented by researches. All existing techniques can be classified into two fundamental groups depending on the main principle of simulation time reduction approach [15].

The first group focuses on the increasing of computational power, e.g. increasing the number of simulated events per second. Usually it is achieved by running the simulation distributed across multiple host machines [35], [49], [50].

Distributed simulation. Graphite [51] is a distributed simulator using dynamic binary translation to deal with functional behavior. It minimizes synchronization overhead by abstracting away events ordering along the simulation. Therefore, while decreasing simulation costs, such a relaxed synchronization vision limits architectural explorations such as communication bottlenecks.

ZSIM [52] improves simulation speed by parallelizing the simulation on x86 multicore hosts performing CPU work using instruction-driven timing models that rely on dynamic binary translation. Authors claim about 2/3 and 4 orders of magnitude speedup than respectively Graphite and gem5. ZSIM supports only x86 ISA-compatible microprocessors.

More generally, the use of distributed simulators is delicate in the sense that users have to carefully deal with simulation partitioning and synchronization among available CPUs, which limits simulation speedup.

JIT dynamic binary translation. The other approach to accelerate the simulation is just-in-time dynamic binary translation, e.g. OVP [53] and QEMU [54]. JIT-based simulators are instrumented with timing models so that basic architecture block models and their inter-operations can be driven according to the annotated timing information. For instance, in the pipeline model included in QEMU [55], authors propose a two-phase approach to estimate the application performance. In the *offline* phase, a cycle pre-estimation of the application execution time is performed. It is then exploited adaptively in the *online* phase according to the CPU status and execution time of critical

instructions, improving the approach accuracy with a mismatch around 10%. A similar approach [56] combines worst-case execution time analysis and QEMU. Here, the offline phase is composed of four steps producing a timing database used online. These approaches miss expressive modeling supports such as those related to cache hierarchies and coherency protocols. Such simulators can achieve speeds close to thousands MIPS [57] at the cost of a limited accuracy. They often focus on functional validation rather than those of architectural exploration.

The second group includes approaches designed to reduce the number of simulation events required for accurate results. It concentrates on optimizing component descriptions (e.g. CPUs, interconnect infrastructure) following the transaction-level modeling strategy [58] or by using trace-driven simulation [59].

Transaction-level modeling. In TLM, details of communication are separated from the details of computation components. Communication is modeled as channels using SystemC interface classes. Communication requests are performed by calling interface functions.

A trace-based simulation method using transaction level SystemC modeling technique has been proposed in [60]. The proposed simulator is used for system performance evaluation on a high abstraction level. It provides high simulation efficiency at the cost of accuracy. Nevertheless, simulation speed comparison and accuracy validation results are not available.

Another trace-based SystemC TLM simulator is described in [61]. The work focuses on scheduling policies and mappings of the target applications. Simulation accuracy results show 5% execution time mismatch compared to a PPC e200z6 processor model on VaST CoMET tool [62].

While reducing the number of simulation events, the use of TLM for architecture exploration is strongly penalized by the poor performance of SystemC kernel and the lack of accurate microarchitecture modeling capabilities [63].

Trace-driven modeling. Trace-driven modeling on the other hand, is a relevant approach in high-performance and embedded computing for reducing simulation cost. A

various number of works have been proposed over the past decades. They target different system components evaluation, e.g. processor microarchitecture, cache hierarchy, network interconnect, multi-/manycore platforms.

Processor simulation: ReSim is a trace-driven reconfigurable ILP processor simulator [64]. It achieves simulation speed up to 28 MIPS and demonstrates an enhancement by at least a factor of 5 enhancement over the reported ILP processor hardware simulators [65] [66].

Cache simulation: An uniprocessor cache simulator based on trace-driven approach has been proposed in [67]. It has been developed to simulate a memory hierarchy consisting of various caches. The simulator produces a set of performance metrics, such as traffic to and from memory. In [68] Dinero IV has been used to predict the performance of a 3D processor-memory chip stack.

The work in [69] presents a trace-driven tool for cache simulation and memory performance studies. Authors report performance improvements of MetaSim Tracer [70] by using techniques developed in SimPoint [71]. The work reduces the cache simulation overhead by decreasing the number of instructions that must be reproduced during the simulation.

NoC simulation: In [72], authors propose a trace model called self-related traces for network-on-chip simulation. The self-related traces are independent on the timing parameter of the network. Each trace includes a dependency field, thus it depends only on the previous message. They demonstrate a simulation time reduction about 75% by comparing a trace-driven model with a relevant GEMS full system simulation. The accuracy of the proposed model is validated by using a statistical hypothesis test. The reported test shows that their trace-driven model is valid.

A trace-based network simulation methodology is presented in [73]. It captures dependencies between network messages and focuses on multithreaded applications. The methodology is implemented in Netrace simulation library for efficient NoC evaluation. Authors demonstrate that Netrace can be orders of magnitude faster than the corresponding full system simulation. The accuracy aspect is outside the scope of their work.

Authors in [74] propose a traffic generation model for fast and effective NoC evaluation. They show a factor of two improvement in simulation time and almost 100% accuracy compared to a complete system simulation.

Platform simulation: In [75], a combination of trace-driven simulation and virtual synchronization techniques is proposed to improve simulation performance and allow exploring the dynamic behavior of application and OS execution. Authors focus on the multi-task software execution and the resolution of communication conflicts. The proposed trace-driven co-simulation performs trace generation and co-simulation simultaneously. Thus, a small amount of storage space is required. The simulation performance and the accuracy level have been evaluated in [76]. The results show up to 38% performance gain and an error within 7.3% compared to the time-accurate instruction set simulator.

ManySim is a trace-based simulation framework for the performance and scalability evaluation of large-scale CMP architectures [77]. It contains four general modules: the core, the cache, the interconnect and the memory. Instruction traces are collected on a single-core real system and fed into a cycle-accurate simulator. To enable the simulation of manycore architectures, authors propose to replicate the traces collected on a single-core system. Data sharing and trace synchronization issues are not discussed in this work. The accuracy level is not reported, but authors mention that a simplistic queuing model of memory module is used.

In [78], the ManySim simulator is used to evaluate memory models for 16-cores architectures. Authors demonstrate that the use of simplistic memory models may lead to wrong conclusions both in absolute performance numbers and in relative performance comparison. The reported results show a difference in performance of up to 65% between the simplistic models and the accurate model of the memory controller.

Tsim is a multi-/manycore processor event-driven simulator which is based on the two-phase trace-driven framework [79]. Authors address two essential issues using trace-driven simulation: the out-of-order processor modeling and the multithreaded workload synchronization. It achieves 150x simulation speed w.r.t the reference Simics-based simulator. CPI error is around 3.2% in average. Tsim demonstrates a simulation speed of 146 MIPS.

A trace-based simulation framework presented in [80] targets the performance analysis of stream-oriented applications on complex MPSoC architectures. Collected application traces are forwarded to subsequently connected virtual machines, which model target system architecture. The proposed framework considers such important aspects as resource sharing, memory allocation and data dependencies. Compared to the CoWare VPA (a commercial instruction-accurate simulator, now part of Synopsys [62]) framework, the evaluation demonstrates a gain in simulation time of 900x times and 97% accuracy in average.

TaskSim is a trace-based event-driven simulator targeting large-scale accelerator-based architectures [81]. The simulator abstracts the computation phase and focuses on the data transfer simulation. A validation against Cell B.E. demonstrates that the execution time mismatch ranges from 1% to 65%. The simulation slowdown of TaskSim compared to the execution of the real Cell B.E. varies from 5-10x to 400-600x depending on the computation-communication ratio of the application. In [59], authors concentrate on exploring multithreaded application behaviors using TaskSim. The application traces, which contain sections of sequential code and parallel work management operations, are distributed diversely among multiple cores to predict application scalability.

The problem of simulating parallel multithreaded applications using the trace-driven approach is also discussed in [82]. Additional synchronization traces are inserted among memory traces. Authors demonstrate a potential error of up to 10.22% by enabling and disabling the implemented synchronization mechanism.

MacSim is a trace-driven and cycle-level architecture simulator [83]. It supports x86 and NVIDIA PTX instruction set architectures. MacSim simulates detailed a pipeline microarchitecture, a memory infrastructure including multi-level caches, a NoC, memory controllers and homogeneous as well as heterogeneous multicore architectures. No accuracy validation or simulation speed results have been published yet.

In a recent work [84], authors present SynchroTrace, a trace-based multithreaded simulation methodology. The tool for capturing computation, communication and synchronization traces is based on the Valgrind Dynamic Binary Instrumentation framework [85]. They also present a trace compression algorithm that reduces the trace file sizes by 63% on average but produces around 10% difference in execution cycles. The trace

filtering technique based on the non-shared data hits filtering is demonstrated. Simulation speed evaluation shows a peak speedup of up to 18.4x over simulation in Gem5 full system mode. Such poor simulation time gain is caused by the fact that not only communication but also computation events are simulated.

In Table 2.2 we compare the existing trace-driven implementations, which target multi-/manycore platform exploration with the proposed trace-driven approach. The implementation field reports the corresponding trace-driven simulator and the environment it is based on. The error percentage is a relative value as in each case it is compared to different bases, e.g. cycle-accurate or instruction-accurate simulations. Speed values are absolute and speedup values are relative. The key features field summarizes such capabilities of trace-driven simulations as core microarchitecture (in-order, out-of-order execution), system architecture (e.g. cache hierarchy, interconnect, memory), multithreading or multitasking, trace synchronization, trace reduction and trace replication.

TABLE 2.2: Comparison of trace-driven implementations.

Reference	**Implementation**	**Error**	**Speed(-up)**	**Key Features**					
				MA	**A**	**MT**	**S**	**RD**	**RP**
[75], 2005	ARMulator, ModelSim	7.3%	38%	✗	*	✓	✓	*	✗
[77], 2007	ManySim, ASPEN	65%	25 KIPS	✗	✓	*	✗	✗	✓
[79], 2008	Tsim (TPTS)	5-10%	150x	✓	✓	✓	✓	✗	✗
[80], 2009	TSim, SystemC, Virtual machines	3%	900x	✗	*	*	✓	✗	✗
[81], 2010	TaskSim	1-65%	100 KIPS 2 MIPS	✗	*	✓	✓	✗	✗
[82], 2012	Pin	10%	N/A	✗	✗	✓	✓	✗	✗
[83], 2012	MacSim (x86, NVidia)	N/A	N/A	✓	✓	*	✗	✗	✗
[84], 2015	gem5, Valgrind	10%	18x	✗	✓	✓	✓	✓	✗
[23], 2015	gem5	14%	800x	*	✓	✓	✓	✓	✓

'✓' - fully supported, '*' - partially supported, '✗' - not supported

MA - microarchitecture, **A** - architecture, **MT** - multithreading/multitasking

S - synchronization, **RD** - reduction, **RP** - replication

Our contribution differs from all previous works by proposing a novel hybrid trace-driven simulation approach suitable for an efficient exploration of an entire manycore system. The great advantage of the proposed simulation is related to its hybrid nature. That means that the trace-driven simulation is embedded into an event-driven environment so that the system architecture including cache, interconnect and memory is dynamically simulated at runtime. The proposed approach includes a trace synchronization technique, which allows managing control and data dependencies as well as a trace reduction technique related to event filtering. Among its strengths is the ability to accurately simulate a computer architecture made of M cores based on traces captured in a reference simulation on a system comprising N cores, with $M \geq N$, thanks to the trace replication. The proposed implementation demonstrates a simulation speedup compared to the reference full system simulation of up to 800x. The average performance metric error is 14% in the worst case architecture exploration scenario.

Chapter 3

Evaluation of multicore architecture models in cycle-approximate simulation frameworks

3.1 Introduction

There is a wide range of approaches from the use of high-level models to hardware prototyping each of which entails different simulation speed/accuracy trade-offs. Some simulation frameworks devoted to CPU-centric systems have been developed over the past decade, that either feature a near real-time simulation speed or moderate to high speed with quasi-cycle level accuracy, often by means of instruction-set simulators or binary translation techniques.

To maintain a reasonable balance between simulation time and accuracy, we put focus on the gem5 simulation framework, which is an event-driven full system simulator. It provides a flexible, modular simulation system that makes it possible exploring multicore architecture features by offering a diverse set of ISAs, CPU models, system execution modes, and memory system models. Furthermore, gem5 has an open source license, a good object-oriented infrastructure and an active mailing list. While the simulation speed is trivially observed, the claimed level of accuracy of the system remains unclear.

Authors in [86] show that the error magnitude in common simulators is often larger than the performance gains yielded by new architecture ideas. A fine tuning of the evaluated model and a detailed analysis of the simulator accuracy are strongly required. The lack of knowledge about the simulation error magnitude may lead to wrong conclusions.

At the moment the proposed accuracy evaluation has been published, there existed no published material that reported and discussed gem5 error magnitude in terms of performance estimation. The following works [45] [46] [47] refer to the proposed accuracy evaluation. The work presented in Section 3.4.2 and 3.4.3 advances the state-of-the-art by addressing the complex performance and power simulation of the heterogeneous ARM big.LITTLE processor.

This chapter presents the accuracy evaluation of two simulation frameworks, gem5 for performance and McPAT for power estimation. It is organized as follow: Section 3.2 presents gem5 and McPAT frameworks and describes their key features. Section 3.3 describes the validation and the accuracy assessment methodology. Section 3.4 demonstrates the accuracy evaluation of performance and power models in gem5/McPAT simulation frameworks by comparing them with real hardware. Section 3.5 summarizes the results of the chapter.

3.2 Background

3.2.1 gem5 for performance modeling

gem5 was created with the best features of two projects, one is focused on a full system simulation (M5 [87]) and another in memory systems (GEMS [88]). It is a modular discrete event-driven simulator running under BSD license. gem5 provides various simulation modes characterized in accuracy and speed namely *full system* simulation and *system call* emulation. The system call mode emulates most operating system-level services through stubs on the simulation workstation, which include the operating system services and devices, resulting in a significant simulation speedup at the cost of limited support for some functionalities, such as multithreading. On the other hand, the full system mode performs complete system simulation, including the OS, thread scheduler

and peripheral devices that run on both user-level and kernel-level instructions, providing high simulation accuracy and penalizing the simulation time. The key features of gem5 framework in terms of architecture simulation are:

- **Multiple CPU models:** gem5 provides four interchangeable CPU models. The simplest model is called *AtomicSimpleCPU*. It is a functional in-order model that uses atomic memory accesses. The *TimingSimpleCPU* is a variation of the SimpleCPU model, which however uses timed memory accesses. The *MinorCPU* and *InOrderCPU* models are developed for detailed in-order microachitecture simulation. And the *DerivO3CPU* model presents full out-of-order microarchitecture simulation.

- **Multiple ISAs:** gem5 supports a variety of common platforms. ALPHA is the most used ISA on gem5. This architecture based on a DEC Tsunami system boots unmodified Linux 2.4/2.6 kernels and FreeBSD, and can be extended for up to 64 cores. ARM models an ARMv7 A-profile ISA, including support for Thumb, Thumb-2, VFPv3 and NEON instruction set extensions. x86 models a generic x86 CPU (64-bit) that boots unmodified Linux kernel in a SMP configuration. SPARC models an UltraSPARC T1 processor (single core) that boots Solaris. PowerPC models a 32-bit processor based on the POWER ISA v2.06 B. MIPS models a 32-bit processor.

- **Memory subsystem:** gem5 provides two different memory systems which are inherited from the M5 (Classic model) and GEMS (Ruby model) projects. The simplest is the Classic model, which provides a fast and easily configurable memory system, whereas the Ruby model focuses on accuracy, interconnect and supports various cache coherency protocols.

- **Multiprocessor/multi-system:** gem5 supports both symmetric and asymmetric multiprocessor systems within a single simulation process.

Moreover, gem5 provides strong debugging capabilities, runs on most operating systems and architectures and has active development/support activity.

3.2.2 McPAT for power modeling

McPAT is a power, area and timing modeling framework for multithreaded, multicore, and manycore architectures. It is developed as integrated framework that works with a variety of performance and thermal simulators via an XML-based interface. This interface describes the microarchitecture specification and is used to communicate dynamic activity statistics generated from the performance/thermal simulator side [19].

The McPAT hierarchy includes three simulation levels [89]:

- **Architectural level.** It represents the multicore processor configuration decomposed into major architectural components as cores, networks-on-chips, caches, memory controllers, and clocking. A core can be in-order or out-of-order and is composed of multiple units such as instruction fetch, execution, load and store units. It also supports multithreading.
- **Circuit level.** In this level, the architectural components are mapped into four basic circuit structures as hierarchical repeated wires, arrays, complex logic, and clock distribution network.
- **Technology level.** This level uses data from the ITRS roadmap to calculate the physical parameters of devices and wires, such as unit resistance, capacitance, and current densities. The current implementation of McPAT includes data for the 90nm, 65nm, 45nm, 32nm, and 22nm technology nodes, which covers the ITRS roadmap through 2016.

The output simulation file contains two groups of values per each architectural component: (i) area, peak dynamic power, sub-threshold leakage power and gate leakage power, which are calculated based on the chosen architecture and (ii) technology parameters and runtime dynamic power that depend on the activity statistics.

3.3 Methodology

3.3.1 Validation and accuracy assessment

gem5 and McPAT simulators together propose a promising exploration environment which allows predicting with a cycle-approximate accuracy architecture performance and power characteristics. However, depending on the complexity and on the modeling details the accuracy level can vary from one system architecture to another. The best solution to validate an implemented architecture model is to compare it with the real hardware and identify the simulation error.

Simulation of advanced complex systems expectedly comprises multiple sources of error. Black and Shen [14] distinguished three separate categories of error sources:

1. *Modeling errors* occur when the simulator functionality is implemented erroneously due to the developer fault.

2. *Specification errors* occur when the simulator developer has untruthful information or has no access to the relevant information. Therefore, the desired functionality shows wrong behavior.

3. *Abstraction errors* occur when developer decides to raise the abstraction level of component implementation. It can significantly reduce simulation time but brings the additional mismatch.

Validation and accuracy assessment flow is illustrated in Figure 3.1. The flow is composed of two phases: *calibration phase* and *experimental phase*.

The calibration phase is intended to analyze the specification and behavior of the target platform in order to most closely implement it on the top of the simulation environment. Generally, it includes three steps, which are marked in the figure with numbers.

In the first step, we study the hardware system and capture component configurations. In this step, the specification error may be found if the hardware specification is not described in detail. It is often the case when academics have no access to commercial architecture information. The second step (highlighted by a dotted line) occurs when the desired component or functionality is not implemented on the simulation environment.

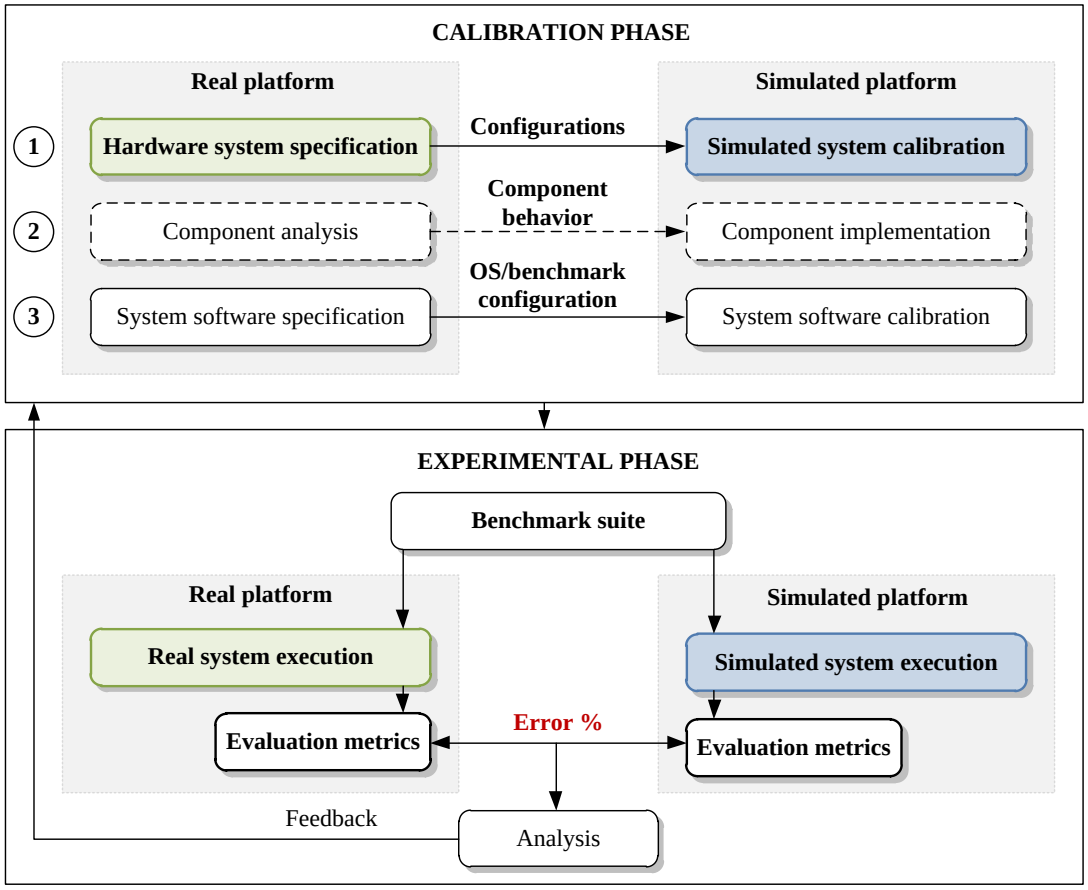

FIGURE 3.1: Validation and accuracy assessments flow.

Thus, we have to analyze the component behavior and implement it properly. Usually, the modeling error relates to this calibration step. The third step concerns the system software specification, namely operating system and benchmark suite choices. The OS features can be used to turn-off some specific components or functionality, which are not implemented in the simulator and thereby prevent their impact on the final error. A significant error may be caused by non-representative benchmark input sets.

The experimental phase consists in benchmark suite execution on both the real and the simulated platforms. The resulting evaluation metrics are compared and the final error percentage is calculated. The number of measurements should be defined in order to obtain statistical significance [90].

Additional error analysis may be done to produce a useful feedback for enhancing the calibration phase.

3.3.2 Evaluation metrics

For our evaluation we used a set of measurement metrics which can be classified into the following groups: (i) *performance*, (ii) *power*, (iii) *accuracy* and (iv)*speed*. *Accuracy* and *speed* metrics are related to the simulation framework properties. We used *performance* and *power* metrics, which are usually used to evaluate the architecture itself for accuracy assessment.

Performance. To evaluate the performance of simulated systems we chose two commonly used metrics: *execution time* and *speedup*.

Execution time of a given task is defined as the time spent by the system executing that task from the beginning to the end.

Speedup of a given task is defined as the relative performance improvement by executing that task. The notion of speedup is commonly used in the context of parallel computing and is instituted by Amdahl's law [91]. For execution time values, the speedup is defined by the formula 3.1:

$$Speedup = \frac{Execution\ time_{old}}{Execution\ time_{new}} \tag{3.1}$$

Power. To analyze the system power consumption and energy-efficiency we used the following metrics: *power consumption* (Watt) and *energy-to-solution* (Joule).

Energy-to-solution of a given task is defined as the amount of energy spent to execute that task. This metric is used to describe the optimization criteria of investigated systems and is calculated by the formula 3.2.

$$EtoS = \int_0^{Execution\ time} P(t)dt \tag{3.2}$$

Accuracy. The accuracy is expressed in *error percentage*. To assess the accuracy of the proposed performance and power models, we calculate the error for performance and power metrics described above namely *execution time*, *power* and *energy-to-solution*. The absolute error percentage is defined by the formula 3.3.

$$Error = \frac{|Evaluated\ value\ - Reference\ value|}{Reference\ value} * 100\% \tag{3.3}$$

Speed The last metric is *instruction per second* that is commonly used by the computer architecture research community and shows the simulation speed. There are several numerical reduction developed to simplify the perception: thousand instructions per second (KIPS), million instructions per second (MIPS) and others.

3.4 Accuracy assessments of gem5 and McPAT versus real platforms

In this section, we validate our performance and power models implemented in gem5 and McPAT simulation frameworks against the real hardware. Two evaluation scenarios are considered.

In the first scenario, we model a simple dual-core ARM based architecture using already implemented features to validate the gem5 simulator itself. In the second scenario, we focus on the complex heterogeneous ARM big.LITTLE architecture evaluation, which requires the implementation of additional functionalities. Both, performance and power models are considered.

3.4.1 Performance modeling: Dual-core SMP architecture

Figure 3.2 illustrates the accuracy assessment flow for the ARM dual-core performance model validation. Chosen benchmark sets, e.g. SPLASH-2, ALPBench and STREAM, are executed on the real reference platform and on the simulated platform. The execution time error percentage is calculated. To identify the source of error we correlate the error with the statistics generated by gem5, namely the number of cache misses.

3.4.1.1 Experimental setup

Reference platform. The Snowball SKY-S9500-ULP-C01 board [92] is used as reference platform. It is equipped with the ST-Ericsson Nova A9500 SoC. As illustrated in

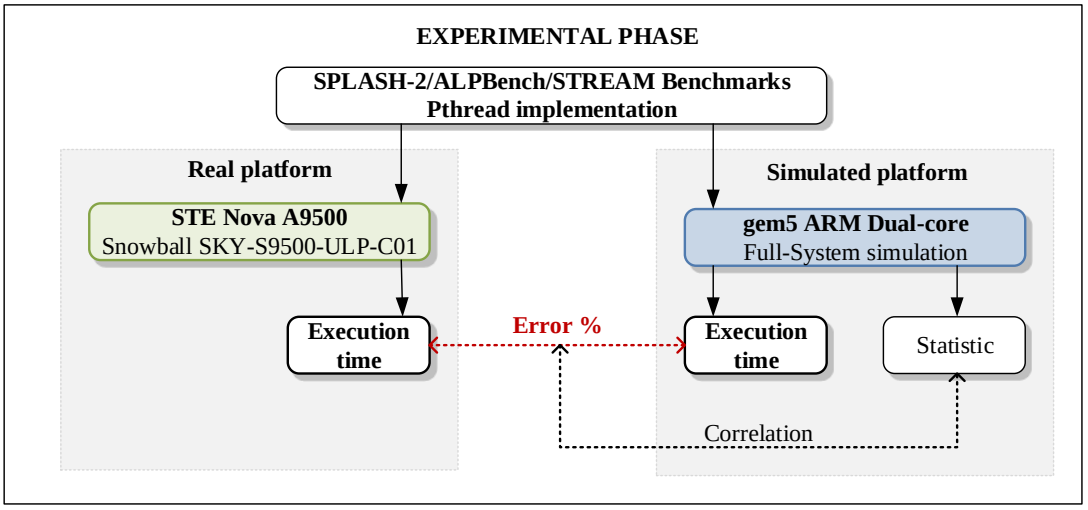

FIGURE 3.2: Accuracy assessments flow for ARM Dual-core architecture.

Figure 3.3, the ST-Ericsson Nova A9500 SoC is built around a dual-core ARM Cortex-A9 processor. It also features a number of DSP and ASIP cores along with a Mali-400 GPU, which will not be used in the experiments.

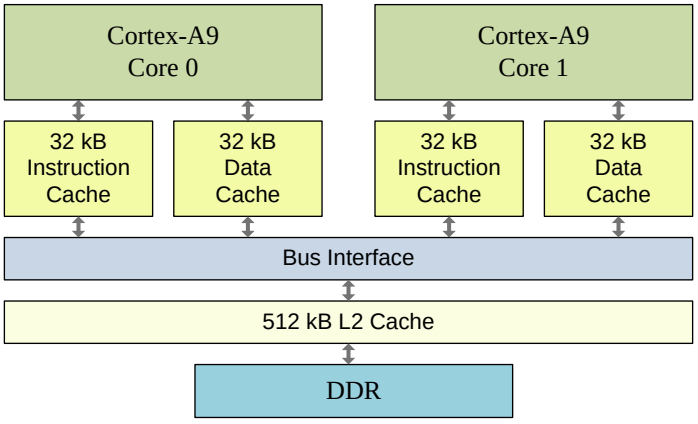

FIGURE 3.3: ARM Cortex-A9 Dual-core block diagram.

This device as most in its class support DVFS, the CPU frequency can be scaled from 200MHz up to 1GHz according to a given policy implemented at kernel-level, and a given set of parameters (e.g. board temperature, battery level and drain, etc.). Because of the difficulty of modeling physical sensor information this feature was disabled, this will ensure the system runs at constant frequency (1GHz).

Performance model. The hardware characteristics are extracted and used for configuring the gem5 model, as follows: (i) ARM dual-core model, (ii) CPU core running at 1 GHz, (iii) Linux Kernel 2.6.38, (iv) 32-kB private L1 data and instruction caches,

(v) 512-kB shared L2 cache, (vi) 64 bits channel width and (vii) DDR physical memory running at 400MHz.

The reference platform features a dual-core ARM Cortex-A9 out-of-order processor. During the calibration phase, our preliminary experiments have demonstrated that *DerivO3CPU*, the out-of-order model delivered by gem5 developers together with ARM society, provides an essential discrepancy compared to the reference platform. According to ARM developers, the microarchitectural configuration of the model has been changed for the commercial security reason. In the early stages of our study, the detailed description of the corresponding microarchitecture, which has been published recently in [93] and [46], was not available. Therefore, we decided to use the more simple *TimingSimpleCPU* model that has showed better accuracy results.

Benchmarks. A set of scientific and multimedia benchmarks are chosen to evaluate the performance model accuracy. The first benchmark is *ALPBench* [94]. This image- and video-centric media benchmark suite is composed of five complex media applications: speech recognition (*CMU Sphinx 3*), face recognition (*CSU*), race tracing (*Tachyon*), MPEG-2 encode (*MPG2enc*), and MPEG-2 decode (*MPG2dec*). Among these, were used the two most frequently used applications: MPEG-2 encoder and decoder.

The SPLASH-2 [95] is the second benchmark suite used from the area of scientific and engineering computing, targeted at cache coherent shared address space machines. Eight complete applications kernels were selected: *barnes*, *fmm*, *ocean* (contiguous/noncontiguous partitions), *radiosity*, *water-spatial*, *fft*, *lu* and *radix*. The methodology and the considered features of each SPLASH-2 workload are described by Woo in [95].

These two benchmarks are implemented using POSIX Threads Programming [96]. Therefore, each benchmark is executed with two parallel threads.

The third used benchmark suite is the STREAM benchmark, which is a simple synthetic program that measures the memory bandwidth (in MB/s) and calculates the corresponding rate for simple vector kernels [97]. This benchmark is a serial implementation.

The complete set of used benchmarks, their domain and detailed description are presented in Table 3.1. The benchmarks source code was compiled using a cross-compiler tool-chain for ARM Linux.

TABLE 3.1: Benchmark set description.

Benchmark	Application	Domain	Description
SPLASH-2	Barnes	Scientific	Simulates the interaction of a system of bodies using the Barnes-Hut hierarchical N-body method.
	Fmm		Simulates a system of bodies over a number of timesteps.
	Ocean		Studies large-scale ocean movements based on eddy and boundary currents.
	Radiosity		Computes the equilibrium distribution of light in a scene using the iterative hierarchical diffuse radiosity method.
	Water-Spatial		Evaluates forces and potentials that occur over time in a system of water molecules.
	FFT		A complex 1-D version of the radix- $\sqrt{n}$ six-step FFT algorithm.
	LU		A kernel that factors a dense matrix into the product of a lower triangular and an upper triangular matrix.
	Radix		An integer radix sort kernel.
ALPBench	MPG2dec	Media	Decompresses a compressed MPEG-2 bit-stream.
	MPG2enc		Converts video frames into a compressed MPEG-2 bit-stream.
STREAM	Stream	Engineering	Measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernels.

3.4.1.2 Accuracy assessments

Benchmark execution. Selected benchmark input sets provide significant difference in execution time, which ranges from milliseconds to tens of seconds. Therefore, the impact of the workload duration on the simulation error can be observed. The operating system `TIME` library is used as the mechanism to measure the execution time of the target benchmark. The `settimeofday()` system call is inserted into the application source code to accurately determine the measuring points. In order to obtain statistically significant results each workload is executed five times. Preliminary experiments showed that such quantity covers the execution time discrepancy and is sufficient for accuracy evaluation. The results are then calculated as average values.

The execution time measured on the reference platform and simulated on gem5 as well as the calculated error percentage are presented in Figure 3.4.

The results show that the mismatch between the real platform and the simulated system varies between 1.39% and 17.94%. Besides, we observe that the error percentage does not depend on the benchmark duration. Simulation of the most time consuming workload,

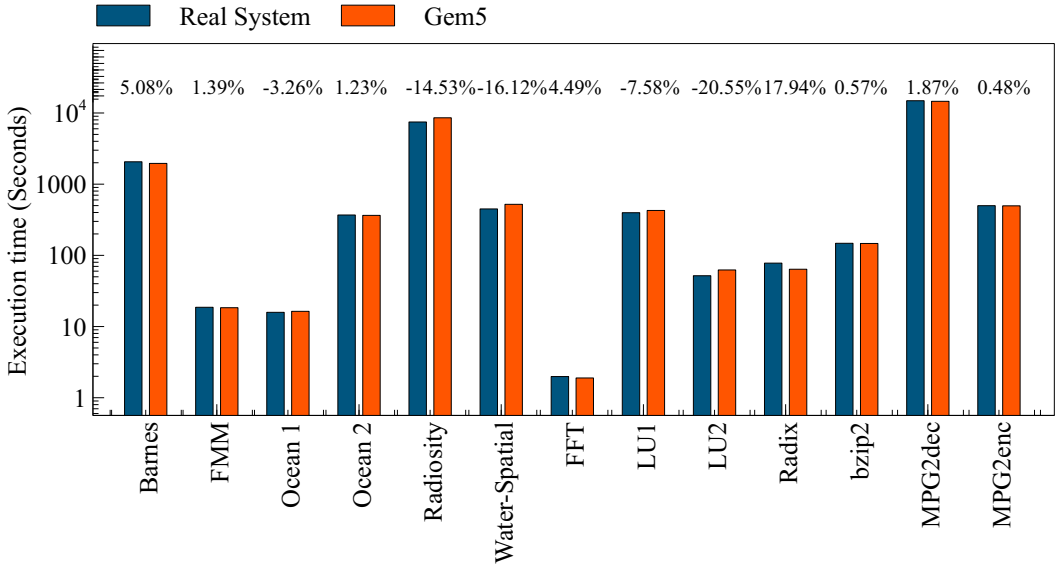

FIGURE 3.4: Benchmarks execution time comparison between the real system and gem5 model.

e.g. *MPG2dec*, provides only 1.87% of error. While the similar in terms of execution time *radiosity* workload shows 14.53% of error.

To investigate this effect and identify the source of error, we chose two representative applications which produce different errors (i) *lu* factorization (noncontiguous blocks) and (ii) *radix* sort kernel. The implementation of these two applications allows easily changing the input parameters, e.g. processed data size, thus the memory communication impact can be analyzed.

The *LU* application behavior exploration. *lu* factorization of a dense matrix is a part of SPLASH-2 benchmark. It can be performed efficiently if the dense $n \times n$ matrix A is divided into an $N \times N$ array of $B \times B$ blocks. Blocking is performed to explore the temporal locality on sub-matrix elements [95].

The default *lu* configurations which were used in the previous experiment are: matrix size is 512×512, the block size is 16. We vary the matrix size between 8×8 and 512×512 in increments of 2, then we compare the execution on both real and simulated systems. The results are presented in Table 3.2 and are shown in Figure 3.5.

In Figure 3.5 a) we observe that as the input matrix size varies from 8×8 to 256×256, the application execution mismatch remains around 6%. Small fluctuations could be explained by the scheduling of various processes in the Linux OS during program

TABLE 3.2: Analysis of the LU factorization execution.

Matrix size		8x8	16x16	32x32	64x64	128x128	256x256	512x512
Execution time (s)	RP	0.99	1.02	1.28	2.26	8.17	54.05	527.6
	gem5	0.95	0.95	1.23	2.30	8.38	54.75	397.4
Error (%)		3.73	6.21	4.05	1.81	2.57	1.29	23.38
# L2 cache misses		23617	23784	23891	24399	26205	55027	789451

execution. However, the error percentage rapidly rises to 23% when the matrix size is increased up to 512×512.

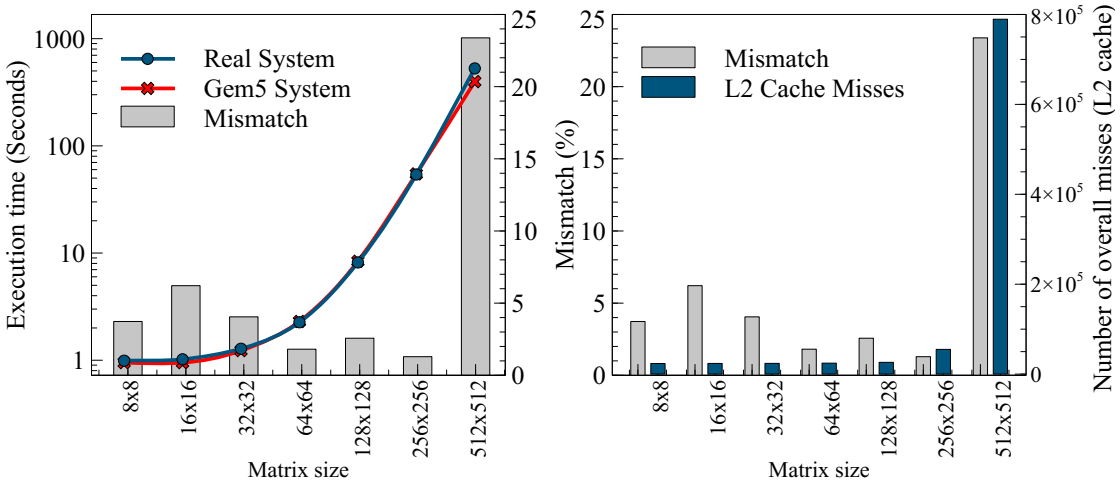

FIGURE 3.5: Analysis of the LU factorization execution.

According to the developer's description, the block size B must be large enough to keep the cache miss rate low, and small enough to maintain good load balancing. Fairly small block sizes (B=16) strike a good balance in practice [95]. When the matrix size changes, it produces a critical condition thus the gem5 model provides the error that is more significant. One reason lies in the increased size of the processed data that makes external memory access latencies prominent. Figure 3.5 b) confirms this hypothesis, showing the number of overall misses into L2 cache memory.

Thus, a change of matrix size from 256×256 to 512×512 results into an increased L2 cache miss rate. At the same time, there is a significant mismatch increased in the reported performance metric between the real board and gem5 simulator. This originates from a somewhat inaccurate model of the external DDR memory used in these experiments. It models latency for each access and an optional random spread factor, therefore abstracting the actual DDR complex access patterns.

***Radix* application behavior exploration.** Another benchmark chosen to explore the mismatch effect is the integer *radix* sort kernel that exhibits the highest mismatch (17.94%). Table 3.3 presents the execution results by varying the number of keys to sort as the input application parameter.

TABLE 3.3: Analysis of the *Radix* sort kernel execution.

Number of keys		512	1024	2048	4096	8192	16384	32768
Execution time (s)	RP	1.587	1.74	2.014	2.502	3.632	5.616	10.10
	gem5	1.434	1.497	1.647	1.951	2.607	3.926	6.53
Error (%)		9.64	10.79	18.22	22.02	28.22	30.09	35.31
# L2 cache misses		26103	26547	26802	27109	28264	31229	37414

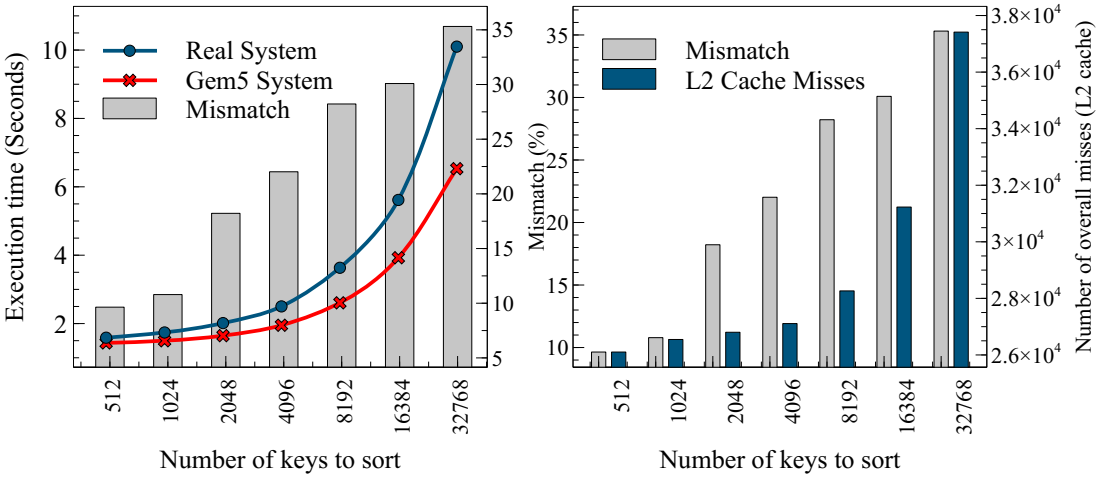

FIGURE 3.6: Analysis of the *Radix* sort kernel execution.

This originates from the integer radix sort kernel that requires the movement of bulk data (the keys being sorted) from the memory, where each core is assigned with an equal fraction of the N sorted keys [98]. Thus, a large number of keys produces a greater communication volume and increases the mismatch.

Figure 3.6 b) shows the L2 cache miss rate, which again confirms the assumption concerning an inaccurate modeling of the DDR memory.

STREAM benchmark exploration. In order to further explore the memory model issue discussed above, we use the *STREAM* benchmark that makes it possible to measure the memory bandwidth.

Each of the four tests adds independent information to the results:

- "Copy" measures transfer rates in the absence of arithmetic operations.

- "Scale" adds a simple arithmetic operation.
- "Sum" adds a third operand to allow multiple load/store ports on vector machines to be tested.
- "Triad" allows chained/overlapped/fused multiply/add operations [97].

The results presented in Table 3.4 show that the memory rate of the *copy* function for both systems is quite similar, the *scale* function reports faster real board transfers, *add* and *triad* functions demonstrate that the gem5 memory rate is twice the rate on the board.

TABLE 3.4: Memory bandwidth when executing STREAM benchmark.

Function	**System**	**Rate (MB/s)**	**Average time (s)**	**Minimum time (s)**	**Maximum time (s)**
Copy	Board	1054.9	0.0304	0.0303	0.0307
	gem5	1058.4	0.0303	0.0302	1.3
	Error %	0.3	0.3	0.3	0.0303
Scale	Board	937.9	0.0341	0.0341	0.0367
	gem5	835.8	0.0383	0.0383	0.0384
	Error %	10.9	12.3	12.3	4.6
Sum	Board	555.6	0.0868	0.0864	0.0873
	gem5	916.2	0.0524	0.0524	0.0524
	Error %	64.9	39.6	39.4	40
Triad	Board	468.7	0.1025	0.1024	0.1027
	gem5	882.4	0.0544	0.0544	0.0545
	Error %	88.3	47	46.9	46.9

We explain these results by the fact that the two last benchmarks manipulate larger data sets (three arrays instead of two) that are likely stored in different pages of the DDR memory. This is confirmed by a lower bandwidth on the physical system (DDR penalty for opening/closing pages, etc.) compared to the gem5 model, which does not account for DDR behavior.

3.4.2 Performance modeling: Heterogeneous multicore architecture

Heterogeneous multicore architectures usually consist of different cores that differ from each other in their instruction set architectures, their execution paradigms, e.g. in-order and out-of-order, their cache size and other fundamental characteristics. Single-ISA heterogeneous multicore [6] processors are of particular interest because of being software-agnostic i.e. a unique standard SMP operating system may be used, taking advantage

of load-balancing features for fine control over performance and power consumption. In the mobile market, several SoC platforms operating on that principle exist, such as Nvidia Tegra 3/4 SoC [7] and ARM big.LITTLE technology integrated to Samsung Exynos 5/7 Octa SoC [8]. ARM big.LITTLE processor features two clusters, "big" and "LITTLE", each of which consists of advanced high-performance cores and low-power cores respectively.

In particular, ARM big.LITTLE processors have three main software execution models [99]. The first and simplest model is called cluster migration. A single cluster is active at a time, and migration is triggered on a given workload threshold. The second mode named CPU migration relies on pairing every "big" core with a "LITTLE" core. Each pair of cores acts as a virtual core in which only one actual core among the combined two is powered up and running at a time. Only four physical cores at most are active. The main difference between clustered migration and CPU migration models is that the four actual cores running at a time are identical in the former while they can be different in the latter. The heterogeneous multiprocessing mode, also known as global task scheduling allows using all of the cores together. A strong argument in favor of HMP is that it provides a fine-grained control of workloads and consequently opens a promising direction for additional performance/energy trade-offs.

Figure 3.7 illustrates the accuracy assessment flow for the ARM big.LITTLE performance model validation. The OpenMP implementation of the Rodinia benchmark suite is used throughout this study. In contrast with previous work, the present study covers several particular points. The target big.LITTLE architecture is evaluated in two operation modes. In the first mode, either big or LITTLE cluster is used at the same time. Thereby, the system represents a symmetric multiprocessing architecture. The second mode involves that both clusters are running simultaneously. Thus, the system becomes an asymmetric or heterogeneous multiprocessing architecture. These two modes are considered in the present validation and accuracy assessment study.

Due to the lack of information, previous work has not introduced the microarchitecture modeling aspects. In contrast, this study is focused on different execution models, e.g. in-order and out-of-order, as well as their impact on the accuracy level.

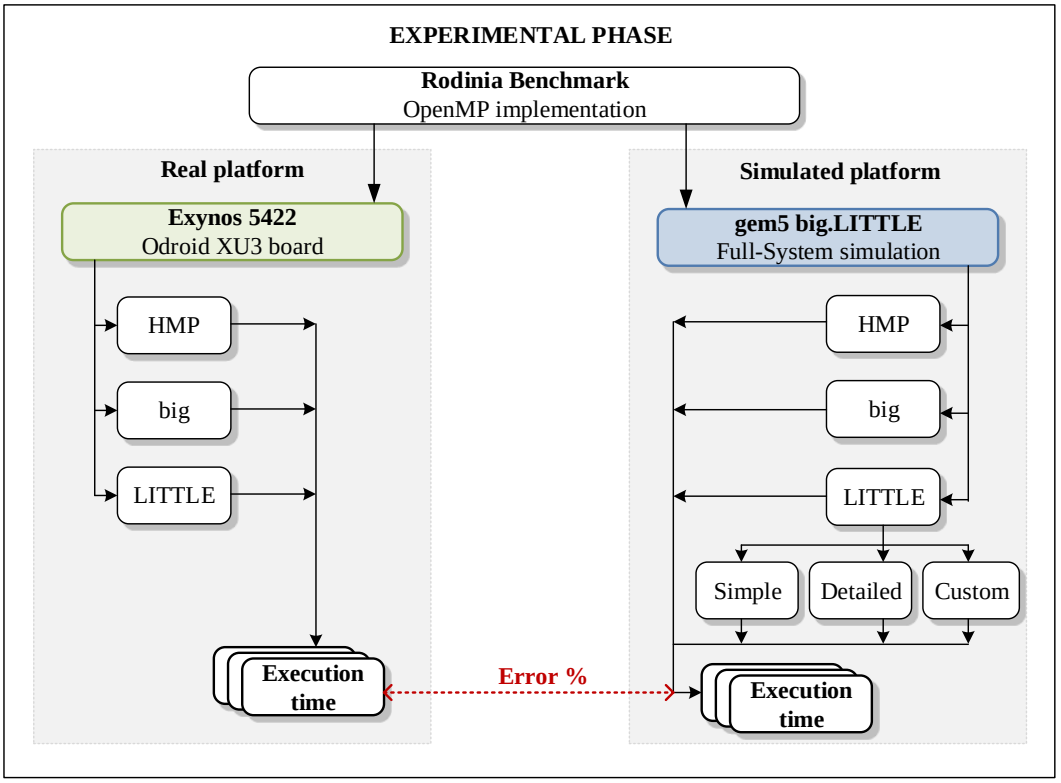

FIGURE 3.7: Accuracy assessments flow for ARM big.LITTLE architecture.

TABLE 3.5: Exynos Octa 5422 chip specification.

Parameters		Cortex-A7 (in-order)	Cortex-A15 (out-of-order)
Number of cores		4	4
Core clocks		200 MHz - 1.4 GHz	200 MHz - 2 GHz
L1 D/I	Size	32 kB	32 kB
	Assoc.	2-way	2-way
	Latency	4 ns	4 ns
L2	Size	512 kB	2 MB
	Assoc.	8-way	16-way
	Latency	21 ns	21 ns
Coherent Interconnect		CCI-400 64-bit	
Memory		2 GB LPDDR3 RAM 933 MHz, 14.9 GB/s, 32-bit, 2 channels	

3.4.2.1 Experimental setup

Reference platform. As a real reference platform, we use the Odroid XU3 board. It is equipped with the Exynos Octa 5422 chip. The general architecture parameters are taken from the product web page [8] and are presented in Table 3.5.

The block diagram of Exynos Octa 5422 chip is shown in Figure 3.8. It features two

clusters, big and LITTLE, each of which consists of quad Cortex-A15 and quad Cortex-A7 cores respectively. Each cluster operates at different frequencies, from 200MHz up to 1.4GHz for the LITTLE and up to 2GHz for the big. Each core has its private instruction (I) and data (D) caches. Both clusters own private L2 cache that is shared among their cores. Cache sizes differ: the Cortex-A7 cluster has a smaller 512kB L2 cache and the Cortex-A15 has a 2MB L2 cache. L1 as well as L2 sizes, associativity and latency are taken from the recently published works [93] [47]. L2 caches are connected to the DRAM memory via the 64-bit cache coherent interconnect CCI-400 [100]. As a system memory, Exynos Octa 5422 chip contains 2GB LPDDR3 package on package (PoP) RAM. It runs at 933MHz frequency and with 2x32 bit bus achieves a 14.9GB/s memory bandwidth.

It is important to note, that the configurations listed above are further inconsistent depending on the sources. Furthermore, Samsung has an ARM architecture license, thus the actual chip architecture (including cache coherent interconnect and microarchitecture) may significantly deviate from ARM 'reference design' and therefore participates in the specification error.

To avoid the error caused by the lack of DVFS and thermal throttling implementation on the simulated system, we disable these features on the reference board. The GPU component is not used in the experiments.

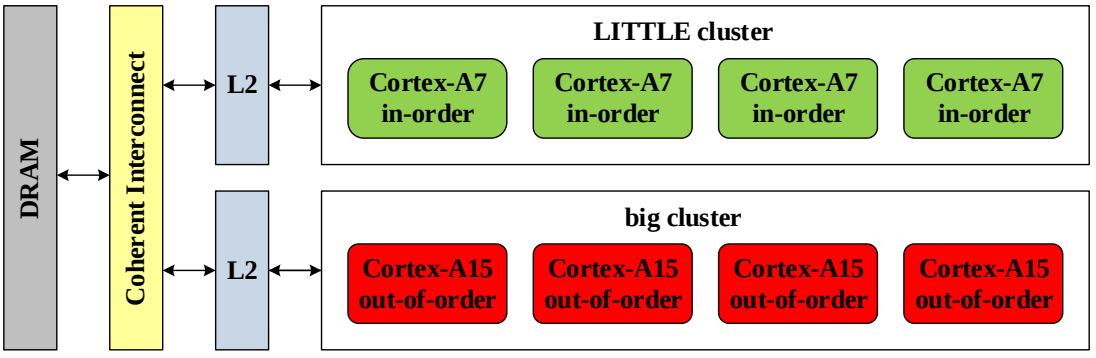

FIGURE 3.8: ARM big.LITTLE heterogeneous multicore processor architecture.

Performance model. We configure our simulation system following the reference platform specification. The gem5 framework provides a set of CPU models, among which are in-order and out-of-order models. The in-order ARM ISA CPU model currently is not supported. This issue is often discussed in the research community and according to gem5 developers there are three solutions: (i) *TimingSimpleCPU* model, (ii) *MinorCPU*

model and (iii) *DerivO3CPU* model, which can be modified to produce quasi-in-order execution behavior [46] [47].

The actual gem5 full system mode has a number of limitations. To overcome these limitations, we realize a set of enhancements:

- **Support of eight ARM cores**: the first limitation is related to the available ARM MPCore processor model that supports only four ARM v7 cores. To run eight ARM-cores system we modify the description of the snoop control unit register. Thus, the SCU count contains no masked number of cores. The `diff` patch is illustrated below.

```
diff -u a/src/dev/arm//a9scu.cc b/src/dev/arm//a9scu.cc
--- a/src/dev/arm//a9scu.cc
+++ b/src/dev/arm//a9scu.cc
@@ -63,11 +63,12 @@
          pkt->set(1); // SCU already enabled
          break;
        case Config:
-         assert(sys->numContexts() <= 4);
+         assert(sys->numContexts() <= 8);
          int smp_bits, core_cnt;
          smp_bits = power(2,sys->numContexts()) - 1;
          core_cnt = sys->numContexts() - 1;
-         pkt->set(smp_bits << 4 | core_cnt);
+         pkt->set(core_cnt);
          break;
        default:
```

- **Heterogeneous multicore**: to build the cluster-based quad ARM Cortex-A15 core together with the quad ARM Cortex-A7 core system, the creation script is enriched by the possibility to merge various CPU models, e.g. in-order and out-of-order, throughout all full system simulation modes.

- **Multiple frequency domains**: to get the big and the LITTLE clusters operate at different frequencies we supply the full system simulation mode with the ability to separately assign distinct clocks to cores.

- **Multiple shared L2 caches**: another gem5 limitation concerns multiple L2 caches that are individually shared among clusters. We add the option to identify the L2 cache number to full system simulation mode. The big.LITTLE technology

assumes cache coherency even when all eight cores are working simultaneously. This sophisticated task is performed at the hardware level by means of a coherent interconnect. Because the particular ARM CCI-400 is not implemented in gem5, we use the CoherentXBar component. This coherent crossbar connects a number of (potentially) snooping masters and slaves, and routes the requests and responses based on the address, also forwards all requests to the snoopers and deals with the snoop responses. It can be used as a template for modeling QPI, HyperTransport, ACE and coherent OCP buses, and is typically used for the L1-to-L2 buses and as the main system interconnect [101].

System Software. The reference Odroid XU3 board runs the latest Ubuntu 14.04 OS on Linux kernel LTS 3.10, which supports global task scheduling. Note that throughout all our experiments we do not use the embedded graphical processing unit. Several modifications are performed in the kernel source code to enable gem5 full system support. These modifications are summarized as follows:

- **Ability to boot eight cores simultaneously**. This modification relates to that described previously and aims at enabling a higher core count in the hardware model, at the SCU-level. The corresponding function fetching the number of available cores from the hardware register has been modified accordingly. The corresponding `diff` patch is presented below.

```
diff --git arch/arm/kernel/smp_scu.c arch/arm/kernel/smp_scu.c
--- a/linux-2.8.38/arch/arm/kernel/smp_scu.c
+++ b/linux-2.8.38/arch/arm/kernel/smp_scu.c
@@ -26,7 +26,8 @@
 unsigned int __init scu_get_core_count(void __iomem *scu_base)
 {
         unsigned int ncores = __raw_readl(scu_base + SCU_CONFIG);
-        return (ncores & 0x03) + 1;
+        return ncores + 1;
 }
```

- **Global Interrupt Controller support**: The *cpu_logical_map* function presented in Linux kernel 3.10 is posing problem, the former implementation (Linux kernel 3.7) is used here.

Benchmark. The Rodinia benchmark suite [102] is used throughout the rest of the section. It is composed of applications and kernels from different domains such as bioinformatics, image processing, data mining, medical imaging and physics simulation. It also includes simpler compute-intensive kernels such as LU decomposition and graph traversal. Rodinia is designed for heterogeneous computing, for that reason CUDA, OpenMP and OpenCL implementations are available. The OpenMP implementation is here chosen, with four threads per cluster, i.e. one thread per core. Also, the `GOMP_CPU_AFFINITY` variable is used to ensure identical thread scheduling on the board and on the gem5 system. The following subset of applications is selected for the investigations: *backprop*, *bfs*, *heartwall*, *hotspot*, *kmeans openmp/serial*, *lud*, *nn*, *nw* and *srad v1/v2*. The problem size for each application is presented in Table 3.6.

TABLE 3.6: Rodinia benchmark description.

Application/Kernel	Abbreviation	Domain	Problem size
Back Propagation	backprop	Pattern Recognition	65536
Breadth-First Search	bfs	Graph Algorithms	4096
Heart Wall	heartwall	Medical Imaging	test.avi, 1 frame
HotSpot	hotspot	Physics Simulation	64 x 64
K-means openmp/serial	kmeans	Data Mining	100
Lower Upper Decomposition	lud	Linear Algebra	256
k-Nearest Neighbors	nn	Data Mining	42760
Needleman-Wunsch	nw	Bioinformatics	1024
Speckle Reducing Anisotropic	srad v1	Image Processing	1 x 502 x 458
Diffusion	srad v2		512 x 512

3.4.2.2 Accuracy assessments

In-order Cortex-A7 model. We explore three available options to model the ARM in-order processor and to identify the accuracy of each one:

1. *TimingSimpleCPU* is the simplest purely functional in-order model, which uses timing memory accesses.

2. *MinorCPU* is an in-order processor model with a fixed pipeline but configurable data structures and execution behavior. It supports the Fetch (1,2), Decode and Execute pipeline stages.

3. *DerivO3CPU* (modified) is the most complex out-of-order model which has Fetch, Decode, Rename, Issue/Execute/Writeback and Commit pipeline stages.

The comparative results are presented in Figure 3.9. Note that the scale for the execution time is logarithmic. The figure shows the execution time for eleven Rodinia applications and kernels executed on the Cortex-A7 cluster running at 200MHz on: (i) reference board, (ii) gem5 *TimingSimpleCPU* model, (iii) gem5 *MinorCPU* and (iv) modified gem5 *DerivO3CPU* model.

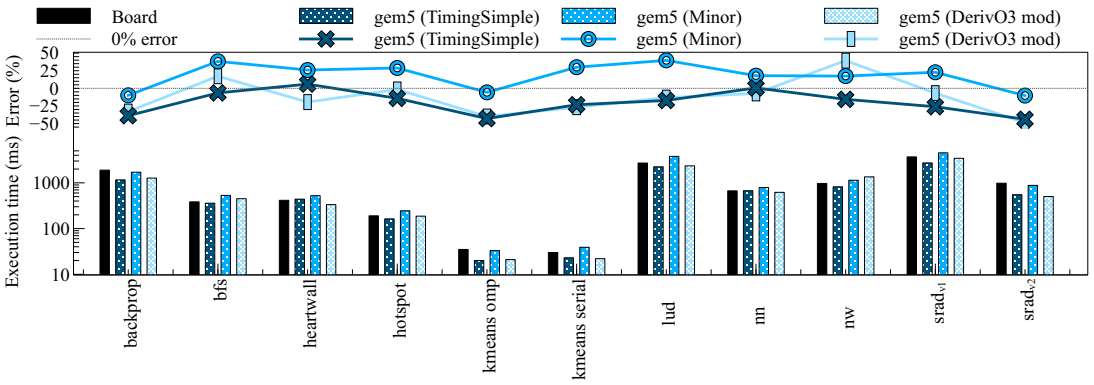

FIGURE 3.9: Execution time comparison for LITTLE Cortex-A7 cluster running at 200MHz.

The absolute error percentage varies between 1% and 50%. The minimum and maximum errors as well as the absolute average error for each scenario are listed in Table 3.7. The results show that the execution time absolute error for all three models is around 22%. Thus, we conclude that for performance evaluation it is enough to use the *TimingSimpleCPU* model. However, for microarchitectural and power consumption explorations we suggest to use detailed CPU models.

TABLE 3.7: Cortex-A7 in-order model execution time error summary.

CPU model	Minimum error	Maximum error	Absolute average error
TimingSimpleCPU	0.6%	43.9%	21.4%
MinorCPU	5.7%	39.7%	22.5%
DerivO3CPU (mod)	2.2%	48.7%	21.6%

SMP and HMP modes. We compare the execution time observed on the board to that given by our gem5 model executing the same workloads. The reported values are averaged over five subsequent runs for ensuring consistency. In order to assess the impact of core frequency, the following configurations are considered:

1. SMP mode: LITTLE Cortex-A7 cluster running at 200MHz, 800MHz and 1.4GHz with 4 threads,

2. SMP mode: big Cortex-A15 cluster running at 200 MHz, 1.1GHz and 2GHz with 4 threads,

3. HMP mode: big.LITTLE Cortex-A7 and Cortex-A15 clusters running at 200/200MHz, 200MHz/2GHz, 1.4 GHz/200MHz and 1.4/2GHz with 8 threads.

The comparison results are presented in Appendix A Table A.1. For each configuration, three rows provide the execution time obtained on the board (indicated as 'B') and with gem5 simulator (indicated as 'S'), and the corresponding absolute error (indicated as '%'). The comparison results are visualized in Figure 3.10. Each scenario has eleven points that correspond to the chosen Rodinia kernels and applications. Their execution time varies between milliseconds and seconds thus the scale is logarithmic. Two large-dotted lines show the -50% and 50% error edges. The absolute error varies significantly depending on the configuration, ranging from 1% to 57%.

To analyze the variability of the measurement the box-plot graphic representation is proposed. Box-plot is a visualization tool, which allows summarizing of the distribution of a dataset [103]. It provides a 5-number summary consisting of the minimum and maximum range values, the upper and lower quartiles, and the median. There are several modifications of the box-plot, for example 2/98 percentile, which provides seven values, e.g. 2/10/25/50/75/90/98 [104].

Figure 3.11 shows a box-plot with 2/98 percentile reporting the observed mean error between the model and the board. The extremes correspond to the *bfs* as a minimum and the *lud* as a maximum outliers for the Cortex-A7 cluster, and to the *backprop* as a minimum and the *lud* as a maximum for the Cortex-A15 cluster. We summarize the observed errors:

- The average absolute error of the Cortex-A7 cluster running at 200MHz, 800MHz and 1.4GHz ranges from 17% to 20%.

- The average error of the Cortex-A15 cluster running at 200MHz, 1.1GHz and 2GHz ranges from 18% to 21%, worsening at higher frequencies.

- The average error of the system in HMP mode with Cortex-A7/A15 clusters running at 200/200MHz, 1.4/2GHz, 200 MHz/2GHz and 1.4 GHz/200MHz are 22.7%, 19.5%, 26.0% and 23.3% respectively.

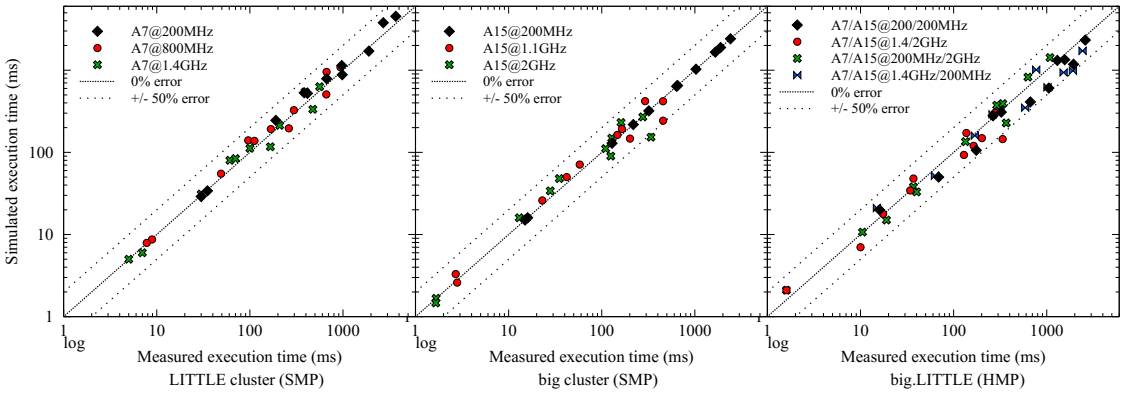

FIGURE 3.10: Execution time comparison for big.LITTLE performance model.

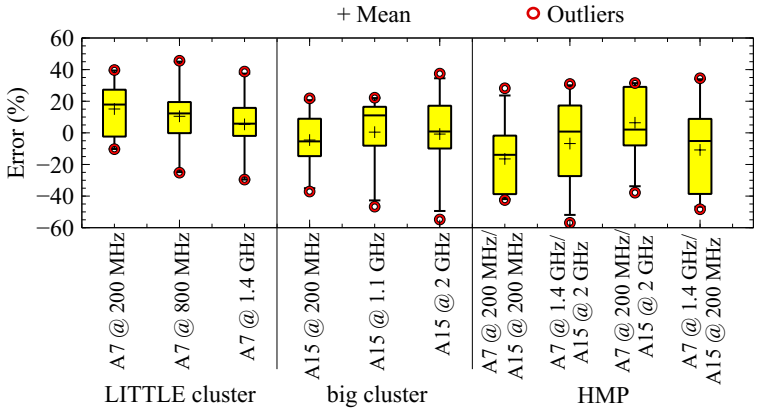

FIGURE 3.11: Execution time error distribution of simulated Rodinia benchmark.

According to the previous accuracy investigations published in [45] and presented in the previous section, the observed errors have multiple sources. The first and foremost source relates to a non-fully cycle-accurate modeling of the in-order and out-of-order execution pipelines of the Cortex-A7 and Cortex 15 respectively. The memory subsystem model also plays an important role, especially regarding the memory controller and the specific timings of the used DDR memory.

Sources of error investigation. For a more detailed analysis of sources of error, we consider the application output information over the time spent in different stages, e.g. initial setup, I/O, kernel execution, etc. We chose three applications: *bfs*, *lud* and *srad v1*. The comparative results between the measured and the modeled Cortex-A15 cluster running at 1.1GHz are shown in Table 3.8. We notice that the execution time error varies dramatically between 5% and 90% depending on the execution stage. The total execution time is compensated. In the presented examples, we observe that the

TABLE 3.8: Application different stage comparison.

Stage	Execution time (ms) Board	gem5	Error
	bfs		
Read graph	0.58	0.04	-93.7%
Allocate memory	3.7	4.5	21.5%
Kernel	33.7	41.5	23%
Store results	5.1	3.3	-35.1%
Total	44.0	49.6	12.7%
	lud		
Kernel	79.476	72.384	-8.9%
Verify	216.514	347.586	60.5%
Total	295.99	419.976	41.9%
	srad v1		
Initial setup	0.31	0.06	-79.4%
Read image from file	140.5	259.4	84.6%
Resize image	3.5	3.3	-5.4%
Allocate memory	0.12	0.09	-29.5%
Extract image	90.3	8.8	-90.3%
Compute	14.4	11.1	-22.9%
Compress image	38.8	23.3	-40%
Save image into file	170.8	111.1	-35%
Free memory	2.4	0.9	-62.3%
Total	461.1	418.5	-9.2%

error of the computation stage is low and amounts to around 20%. At the same time, stages related to memory operations, e.g. *Store results*, *Read image from file*, *Save image into file*, etc., produce high error percentage. Thus, we conclude that the main source of error in the proposed model is the memory system. The actual modeling of the cache coherency in gem5 is probably not fully accounting for all advanced features of the ARM cache coherent interconnect [105]. This observation can also explain the slight error increase when switching to the HMP mode. In this mode, the memory communications are more complex and inaccurate cache coherency protocol provides a noticeable discrepancy.

3.4.3 Power modeling: Heterogeneous multicore architecture

Figure 3.12 depicts the accuracy assessment flow for the validation of the ARM big.LITTLE power model. This study is related to the performance model evaluation in Section 3.4.2. The power model includes the architecture description obtained from the architecture simulator, i.e the gem5 performance model, the statistics file generated by the full system simulation and the technology specification described in McPAT environment.

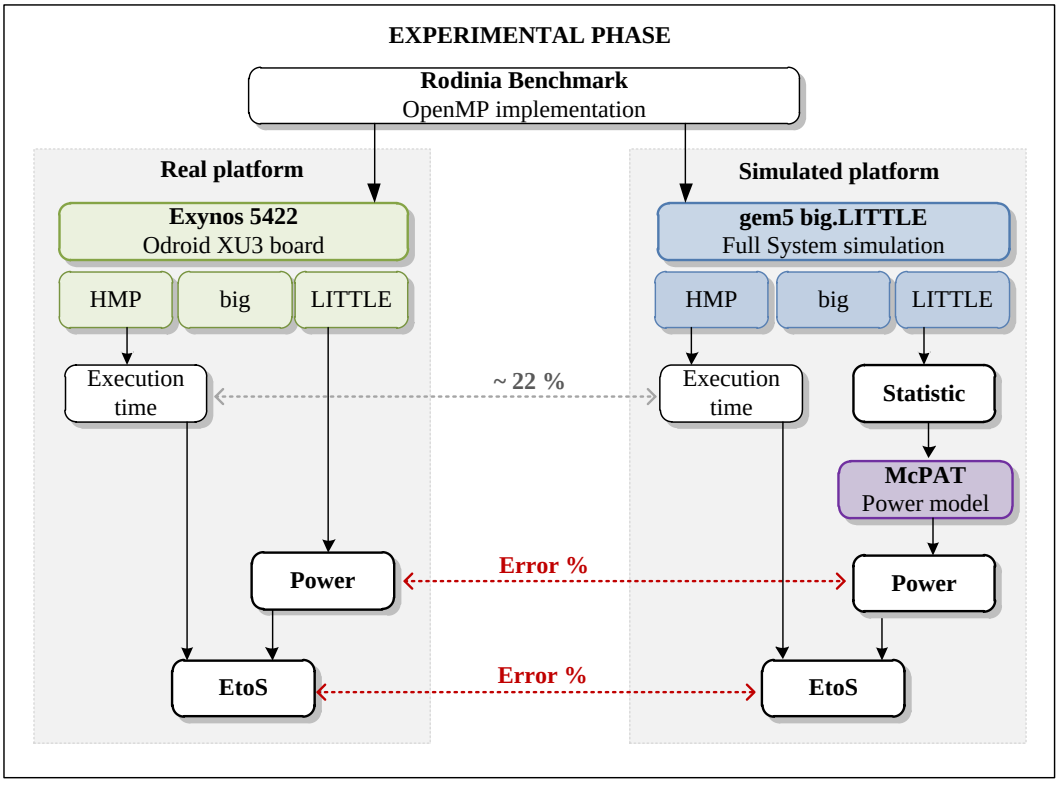

FIGURE 3.12: Accuracy assessments flow for ARM big.LITTLE power model.

As well as in the performance model study, we consider three scenarios, that are big cluster only in SMP mode, LITTLE cluster only in SMP mode and big.LITTLE architecture in HMP mode. Besides the power metric comparison, we also calculate the energy-to-solution mismatch by using the execution time results which have been reported in Section 3.4.2 and presented in Appendix B Table A.1.

3.4.3.1 Experimental setup

The McPAT framework allows accurately calculating power consumption based on the statistics collected through the gem5 simulation. The general architecture parameters are configured according to Table 3.5. The additional McPAT parameters, which are related to the manufacturing technology are presented in Table 3.9.

The Exynos Octa 5422 SoC is built using a 28nm manufacturing process. The Vdd and temperatures are experimentally measured on the Odoid XU3 board by means of the internal sensors. The Vdd values depend on the Linux kernel configuration and are related to the adaptive supply voltage technique used in Samsung SoCs. The operating

TABLE 3.9: big.LITTLE McPAT parameters.

Parameters	Cortex-A7 (in-order)	Cortex-A15 (out-of-order)
Technology	28 nm	
Vdd @ 200/200 MHz	0.91 V	0.91 V
Vdd @ 1.4/2 GHz	1.24 V	1.3 V
Temperature @ 200/200 MHz	310-320 K	310-320 K
Temperature @ 1.4/2 GHz	310-320 K	320-330 K

temperature strongly depends on the cluster architecture and application nature. For the Cortex-A7 cluster, the temperature is always below 323K and the board fan is never used to cool down the chip. For the Cortex-A15 cluster, the temperature strongly rises above 323K and the board fan is configured to cool down the chip.

3.4.3.2 Accuracy assessments

The McPAT framework [19] provides the following results: area, peak dynamic, sub-threshold leakage, gate leakage and runtime dynamic power estimations. To evaluate the accuracy of the McPAT big.LITTLE model we compare the average power consumption measured on the Odroid XU3 board to the average runtime power estimated in the McPAT. The considered runtime power equals the sum of runtime dynamic power and leakage power.

The following three scenarios are investigated: (i) Cortex-A7 cluster only running at 200 MHz and 1.4 GHz, (ii) Cortex-A15 cluster only running at 200MHz and 2GHz, (iii) Cortex-A7 and Cortex-A15 running at 200/200MHz, 200 MHz/2GHz, 1.4 GHz/200MHz and 1.4/2GHz respectively. The comparison results are presented in Table B.1. The reported total power consumption equals the sum of the Cortex-A7 cluster power, Cortex-A15 cluster power and the DDR memory power. Other peripherals such as storage, network and cooling are therefore here not accounted for.

The error percentage distribution including the negative values is shown in Figure 3.13 in the form of box-plot. The highest error percentage is provided by the memory component. The external memory model is the most influenced by the cache and interconnect inaccuracy. These results allow estimating the power consumption that ranges between tens and thousands of mW.

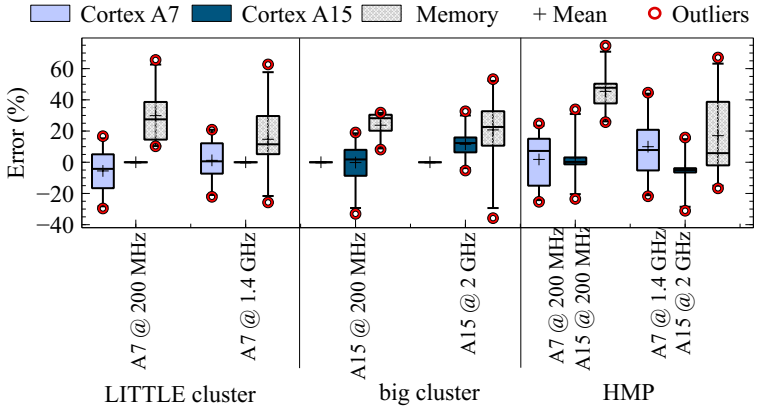

FIGURE 3.13: Power consumption error percentage summary.

The corresponding total power error percentage is 12.7%, 11.7% and 10.8% for the LITTLE cluster, for the big cluster and for the HMP big.LITTLE respectively.

Based on the simulated execution time presented in Section 3.4.2 and the above total power results we calculate the EtoS and compare it with the values measured on the Exynos Octa 5422 chip. The comparison results among all applications are shown in Figure 3.14. The average absolute error percentage is 21.9%, 27.9% and 22.1% for the LITTLE cluster, the big cluster and HMP big.LITTLE respectively.

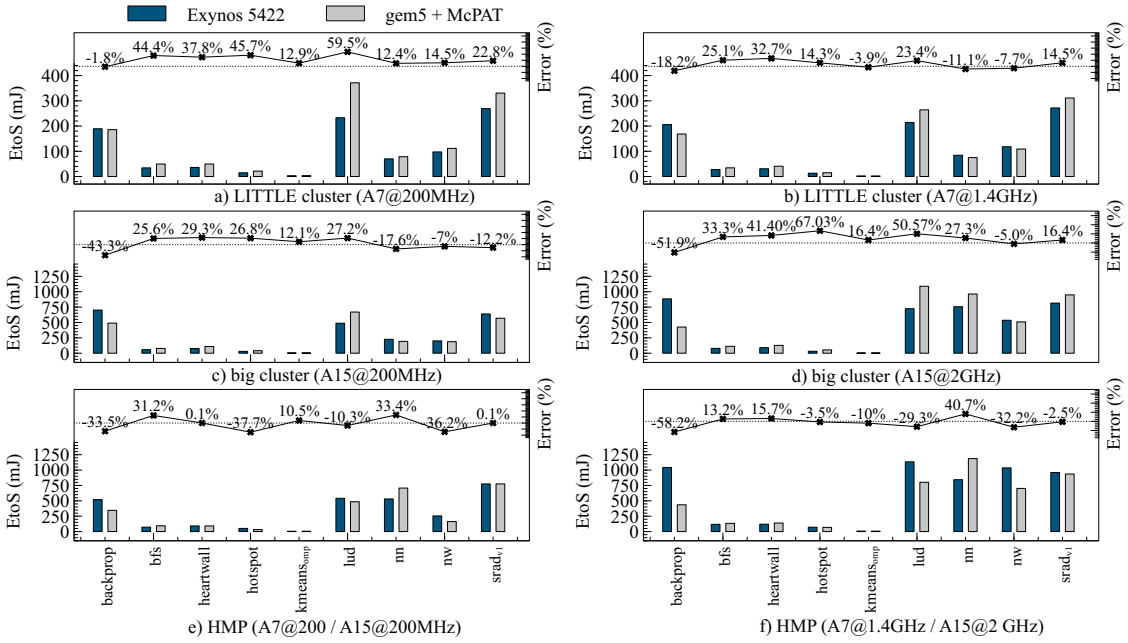

FIGURE 3.14: Energy-to-solution comparison for ARM big.LITTLE architecture model.

The summary of error percentage for the execution time, power and EtoS is shown in Figure 3.15. The EtoS error percentage includes both, the gem5 execution time error and the McPAT power consumption error. Therefore, such scenarios as Cortex-A7 cluster running at 200MHz and Cortex-A15 cluster running at 2GHz cumulates the error and show a higher mismatch. On the other hand, the HMP scenarios have negative execution time error and as a result a compensated EtoS.

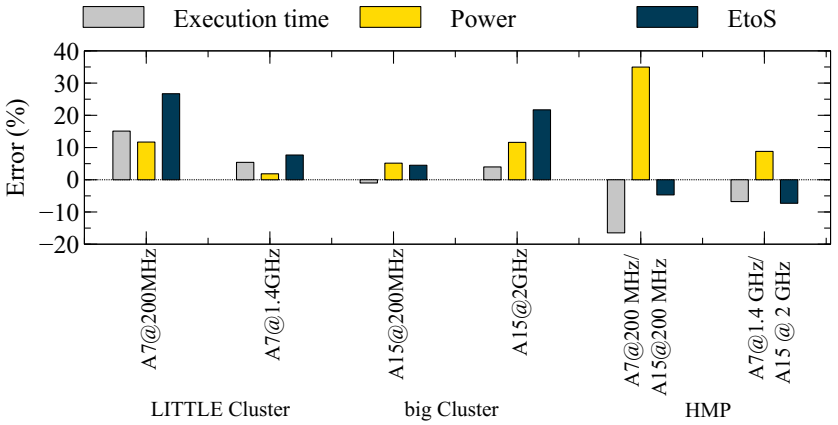

FIGURE 3.15: Error percentage summary for performance, power and energy-to-solution.

3.5 Discussion

In this chapter, we evaluated the accuracy of multicore architecture performance and power models implemented in gem5 and McPAT simulation frameworks. Within this study, three scenarios were investigated: (i) the performance model of the ARM dual-core symmetric multiprocessor architecture, (ii) the performance model and (iii) the power model of the ARM big.LITTLE heterogeneous multiprocessor architecture.

First scenario. This study focused on the performance metric evaluation, e.g. execution time. The microarchitecture as well as the power consumption aspects were out of the scope. We used an extensive set of multithreaded and serial benchmarks, which represents a variety of scientific workloads (SPLASH-2 benchmark suite), media applications (ALPBench) and memory bandwidth benchmark (STREAM). The accuracy evaluation showed that the error varies from 1.39% to 17.94%. No visible relation between error and workload durations had been observed.

Two applications, namely *lu* factorization and *radix* sort kernel, with different error percentages were chosen to explore the accuracy dependency on the input problem size. In both cases, the execution time mismatch has been growing with the application problem size. These observations were also correlated with the number of L2 cache misses which showed similar in direct ratio behavior.

We assume that these results are caused by an inaccurate model of the external DDR memory together with the unrealistic model of the cache coherency protocol [105]. To conclude, due to the lack of available architecture description information and overly simple memory model, our performance model provides specification and abstraction errors between 1% and 35%.

Second scenario. This study is related to the ARM big.LITTLE heterogeneous multiprocessor architecture performance and power models evaluation. The models have been calibrated and validated w.r.t. the Exynos 5 Octa 5422 chip.

Within the performance model validation, the in-order Cortex-A7 model evaluated. The results demonstrated the averaged error around 22% for all three considered scenarios, i.e. *TimingSimpleCPU*, *MinorCPU* and *DerivO3CPU* (modified) models. As the authors in [45] concluded, architectural simulators are not microarchitectural simulators. If the main goal of the study is performance exploration and not the microarchitecture components study, the choice of the model does not affect the results. Thus for faster, yet accurate execution time analysis, the use of *TimingSimpleCPU* is sufficient. For more detailed exploration of microarchitecture the *MinorCPU* may be used. Finally, for power consumption or are estimation, it is important to model each processor component in detail thus the *DerivO3CPU* would be the best solution.

For this reason, the rest of the experiments were performed with *DerivO3CPU* (modified) Cortex-A7 model. The big.LITTLE performance model evaluation results demonstrated around 20% of absolute error percentage in average. We observed that the switching from SMP (either big or LITTLE cluster) to HMP mode does not affect the error.

The additional analysis of the outputs of several applications showed, that the execution stages related to the memory operations, e.g. image reading or results store, produce high errors. While the kernel computation stage errors are less than 20%.

Based on these observations we conclude that the ARM big.LITTLE performance model also contains an inaccurate memory system. The first reason is the CoherentXBar component that replaces the CCI-400 cache coherent interconnect. The second point is the LPDDR3 memory controller, which timing settings are different from the real board PoP RAM.

The last part of this chapter concerns the ARM big.LITTLE power model assessment. The McPAT/gem5 simulation showed 13% of the total power consumption error in average and 24% of the combined energy-to-solution error.

The accuracy evaluation results confirmed that the performance and power models implemented in cycle-approximate simulation frameworks can be used to realistically predict desired exploration metrics. Nevertheless, these models are too slow for large scale manycore architecture exploration. In this context, we followed to the approaches that allow accelerating the simulation still preserving a suitable level of accuracy.

Chapter 4

Hybrid trace-oriented approach for fast and accurate simulation of manycore architectures

4.1 Introduction

Efficient exploration of complex manycore systems requires fast, flexible and yet accurate simulators. Available industrial and academic simulators differ in terms of simulation speed and accuracy trade-offs, and their adoption is usually defined by desired exploration level. Cycle-accurate simulators are popular and attractive for computer architecture exploration. gem5 environment is a popular event-driven full system simulator that provides a large number of simulation capabilities, such as a rich set of ISAs, CPU models, system execution modes and memory system models. In Chapter 3 we evaluated the accuracy level of gem5 simulator by modeling homogeneous and heterogeneous multicore architectures. According to the results, the performance model provides around 20% of error in average throughout all considered scenarios. Even though enabling flexible and detailed multicore architecture exploration, gem5 entails slow simulation speed, thereby limiting its scope of applicability for large-scale manycore systems. This calls for alternative approaches capable of providing high simulation speed while preserving accuracy that is crucial to architectural exploration.

Within event-driven simulation paradigm there are two fundamental groups of simulation time reduction approaches. The first group approaches allow increasing the number of simulated events per second. It can be achieved by running the simulation distributed across multiple host machines [35], [49], [50]. Just-in-time dynamic binary translation, e.g. OVP [53] and QEMU [54], also refers to that category. JIT-based simulators are instrumented with timing models so that basic architecture block models and their interoperations can be driven according to annotated timing information. Such simulators can achieve speeds close to thousands MIPS at the cost of limited accuracy. They often focus on functional validation rather than those of architectural exploration.

The second group includes approaches designed to reduce the number of simulation events required for accurate results. It concentrates on optimizing component descriptions following the transaction-level modeling strategy [58] or by using trace-driven simulation [59]. While reducing the number of simulation events, the use of TLM for architecture exploration is strongly penalized by the poor performance of SystemC kernel and the lack of accurate microarchitecture modeling capabilities citeAutomaticTLM. A various number of works that implement trace-driven approach have been proposed over the past decades. Depending on the exploration target these approaches focus on simulation of processor microarchitecture [65] [66], cache memory [67] [69], network-on-chip [72] [73] or entire multi-/manycore architecture [75] [77] [79] [80] [81] [82] [83] [84].

Single-core architecture exploration has been successfully maintained by trace-driven simulation over the last decades. However, its applicability to multicore architecture exploration strongly requires complementary mechanisms to manage resource sharing, memory allocation and data dependencies. In [75] and [81] authors focus on task distribution exploration among multiple cores. Virtual synchronization techniques are therefore proposed. In [83] multithreading programming is considered but trace synchronization mechanisms are not discussed. Focusing on manycore architecture exploration authors in [77] propose to collect traces on single-core system and then replicate a single trace on multiple cores. Trace synchronization is also out of the published work scope. The existing multicore simulation works [79] [80] [82] [84] do not cover large-scale manycore architecture exploration.

The presented work aims at proposing a novel hybrid trace-oriented simulation approach

for event-driven computer architecture simulators such as gem5. The fundamental principle of the approach lies in decreasing simulation complexity by abstracting away core execution into traces, as follows:

1. core execution traces, i.e. incoming/outgoing memory transactions, are captured in a full system simulation;
2. these traces are augmented with synchronization semantics, then replicated into so-called augmented vector traces to simulate systems made of a higher core count;
3. augmented vector traces are replayed into a final event-driven simulation through traffic injectors; only interconnect and memory subsystem are actually simulated thereby resulting in significant performance boost.

This approach is implemented in gem5 simulation framework and is validated on ARM ISA (it operates however on other ISAs). Known limitations lie in the trace-driven nature of the approach: threads are pinned to cores and runtime features such as load balancing are not modeled any further. Our solution advances state-of-the-art in trace-driven simulation through its ability to fast and yet accurately simulate a computer architecture made of M cores based on traces captured in a reference simulation on a system comprising N cores, with $M \geq N$, thanks to trace replication and synchronization.

This chapter is organized as follow: Section 4.2 introduces the general concepts of the trace-driven simulation approach. In Section 4.3 we describe the proposed methodology. Section 4.4 presents the implementation of the proposed trace-driven approach in gem5 simulator. In Section 4.5 we evaluate the speedup and accuracy comparing to the full system mode. An alternative implementation of trace simulation phase in RTL framework and memory mapping investigations are presented in Section 4.6. Section 5.7 summarizes the results of the chapter.

4.2 Background

To define the trace-driven simulation concept the notion of trace is used. Trace is a time ordered record of events on real system. We distinguish tree fundamental phases of the trace-driven approach: *trace collection*, *trace reduction* and *trace simulation*.

Trace collection. The first phase, which is devoted to trace collection may be divided into two basic classes. The class determines what kind of reference system is used to collect the traces: hardware-based or software-based. The hardware-based class is an older approach and is complicated by the runtime perturbation and severely restricted trace size issues. At the same time, the active development of software tools has pushed researches toward software-based trace collection. Namely, the use of diverse emulation and simulation frameworks breaks the wall of trace collection complexity and makes it much easier. Nonetheless, the trace representativeness and completeness are still under developer's responsibility. It is the well-known fact that the output of any model is only as good as the input to that model. Therefore, the accuracy issue is directly related to the trace quality.

Trace reduction. Here we highlight the second common issue, which concerns the size and interoperability of the collected traces. Depending on the assigned tasks the trace size may reach hundreds of gigabytes per one workload. Even with the rapidly growing disk space such amounts of data are unacceptable. As a result yet another intermediate trace reduction phase occurs and is intended to reduce the size of the collected traces. There are three essential groups of reduction techniques [106]: (i) *compression*, (ii) *sampling* and (iii) *filtering*. Each of them is estimated according to such effectiveness indicators as reduction factor and accuracy loss.

The compression technique involves the use of some compression algorithm [107] [106]. The effectiveness of compression depends not only on the reduction factor, but also on the compression/decompression slowdown, which can wipe the achieved simulation speedup. In general, this technique yields to no loss of accuracy.

The sampling technique focuses on the experiments with a large data set and is intended to obtain a smaller representative set of traces [108] [109]. It has a number of restrictions concerning cache simulation. Due to the prediction methods used in the sampling technique, the accuracy loss can be significant [110].

The filtering technique stores only the significant-event traces and filters the other. One of the example of this technique is cache filtering [111] that stores only miss traces ignoring the hits. The restrictions in this case concern cache size, lines and associativity, which should be fixed. According to the trace-driven memory simulation survey [110], the error of the filtering techniques is less than 5%.

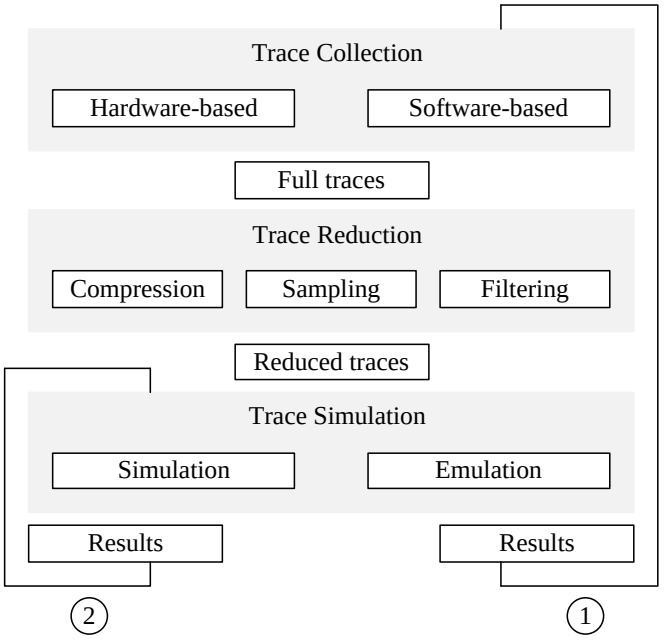

FIGURE 4.1: Trace-driven exploration flow.

Trace simulation. When the traces are collected and ready to be used the final phase, e.g. trace simulation, begins. This is the key exploration stage, which should produce essential results. Indeed, the trace collection and simulation phases can be performed in the different environments. For example, traces collected on detailed full system simulator then are injected into RTL simulator. The environment is often dictated by the design space and expected accuracy.

Figure 4.1 illustrates the general concepts of trace-driven exploration flow. Notice, that there are two exploration flow loops. The first loop (highlighted as '1' in the figure) presents the main path, which is traversed at least once during an exploration. The short loop (highlighted as '2' in the figure) addresses the design space exploration and is the main source of benefits from the trace-driven approach. Depending on the chosen implementation, the design space exploration may require multiple main loop iterations that, obviously, reduce the efficiency of the trace-driven approach.

4.2.1 Abstraction levels of computer architecture exploration

The trace-driven approach have been widely used since the first publication in 1969 [112]. With the increasing complexity of explored architectures, the variety of trace-driven approach application became more important. We classified them based on the

system abstraction levels.

Figure 4.2 schematically illustrates the various levels at which exploration can be conducted in a typical computer architecture.

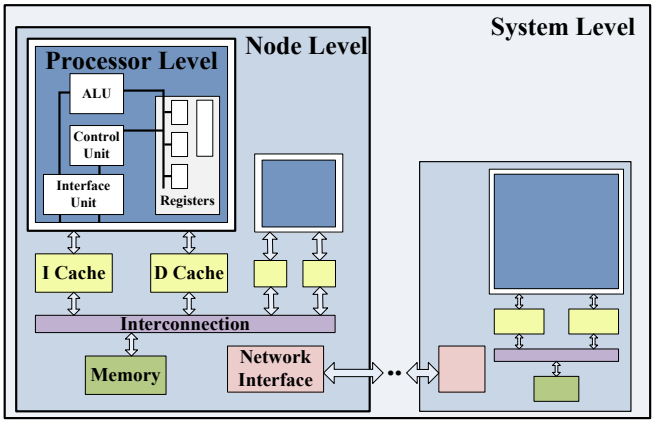

FIGURE 4.2: Computer architecture exploration levels.

The first level relates to processor microarchitecture study. Usually, it focuses on the internal pipeline organization and instruction set architecture. At the processor level simulation uses the *instruction traces* which may contain instruction opcodes, interrupts, memory reference addresses. Authors in [113] [114] [64] investigate processor microarchitecture using trace-driven simulation.

Node level explorations focus on memory hierarchy organization. As the basic performance scaling issue in today computer architecture is memory bottleneck, this abstraction level is commonly used. The explored architecture may include multi-level cache memory, interconnect, main memory. *Memory* or *address traces* are typically used on the node-level exploration. They may contain virtual/physical address, memory reference type, etc. Works [84] [82] [59] [61] focus on node-level exploration.

At system level, explorations relate to cluster behavior in which participating nodes exchange messages via an arbitrary communication network. These messages are introduced than by the *I/O traces*. This is the highest exploration level abstraction. It targets machines with a very large number of processors, which communicate their parallel activities via MPI. Examples of system-level exploration have been published in [35] [115] [116].

4.2.2 From in-order to out-of-order processor

Processor microarchitecture has undergone major changes. A variety of implementation concepts are used nowadays to achieve better performance at lower cost. Most of them aim at instruction parallelization in order to hide usually much slower memory communication latency. The following techniques for increasing the program execution speed have been developed over the past decades: instruction pipelining, cache memory, branch prediction, superscalar, out-of-order execution, register renaming. Each of them brings more challenges into the trace-driven simulation.

In Figure 4.3 the in-order and out-of-order processor execution difference is demonstrated. In both cases, the program executes three instructions. In the first line when loading the value a cache miss occurs. In the case of the in-order execution illustrated in Figure 4.3 a), the processor waits until the miss is satisfied to continue program execution. Instead, if the out-of-order execution is allowed, processor executes the following independent instruction during the cache miss waiting. Thus, it achieves a significant program execution speedup.

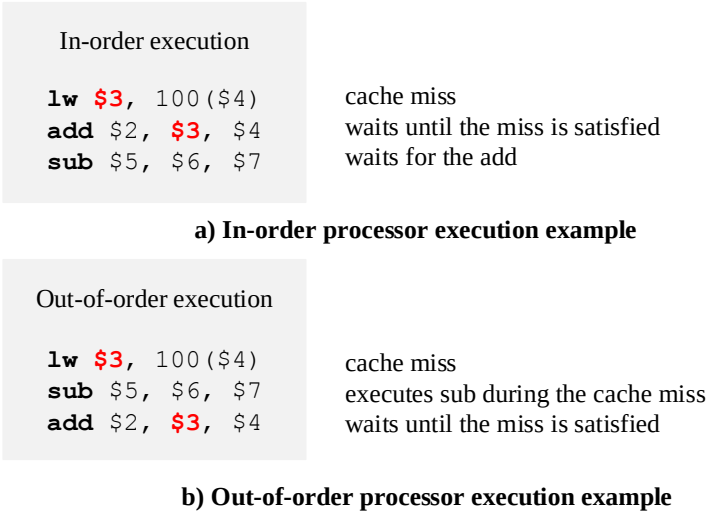

FIGURE 4.3: In-order versus out-of-order processor execution.

While the in-order execution is easily reproducible by trace-driven simulation, the dependency between *load* and *add* instructions in out-of-order processor requires a deeper analysis. It is called the *microdependency* issue. In the node-level exploration when the processor is abstracted away, the microdependency analysis plays an important role.

The lack of accuracy in instruction order execution may lead into false memory accesses and consequently may lead to significant runtime error.

4.2.3 From single-core to multicores processors

The crucial landmark in computer architecture, which is related to the switching from single-core to multi-core processing sparked a wave of challenges throughout all relevant community. Since, a field of parallel programming keeps pace with the rapidly evolving multiprocessing architecture design. Two fundamental categories of architecture/programming model couple are characterized by the memory organization. Figure 4.4 illustrates these principally different memory organizations: *shared memory* and *distributed memory* [117].

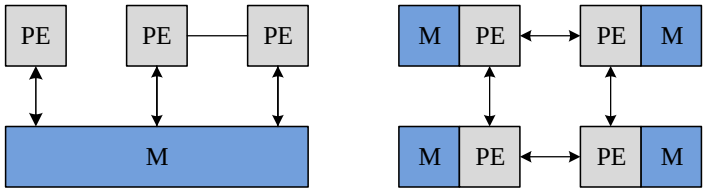

FIGURE 4.4: Memory organization: how processing elements (PEs) communicate and access memory (M).

Shared memory architecture presented in Figure 4.4 a) implies that all processing elements share the same address space and communicate through global memory reading and writing operations. The main problem of shared memory architecture is performance scaling degradation which is due to two factors:

1. The contention problem occurs when several processing elements try to access the same memory location. The use of complex cache memory hierarchy became the typical solution for this issue.

2. The presence of multiple copies of the same data in caches may cause coherence problems. To ensure data consistency a variety of cache coherency protocols have been developed.

A great advantage of shared memory architecture is a simple programming model. It is built on three main constructs: (i) task creation, (ii) communication and (iii) synchronization. There are several most commonly used shared memory programming.

The `POSIX Threads` [96] is a standardized C language threads programming interface for UNIX systems. Pthreads are defined as a set of programming types and procedure calls. The library is supported by the most hardware vendors in addition to their proprietary API's.

The `Open Multi-Processing` API specification [118] defines a portable and scalable shared-memory parallel programming model. It supports multi-platform programming in C, C++ and FORTRAN on most processor architectures and operating systems from the standard desktop computer to the supercomputer. OpenMP consists of a set of compiler directives, library routines and environment variables for developing parallel applications.

Another programming language for multithreaded parallel computing based on the C and C++ is `Cilk` with its later commercialized versions `Cilk++` and `Cilk Plus` [119] [120]. Design in Cilk language assumes that the programmer is responsible to identify parallel regions. While the decision how to actually divide the work between processors is done by the runtime environment.

Distributed memory architecture presented in Figure 4.4 b) refers to a computer system in which each processing element has its own private memory. Unlike the shared memory architecture, there is no global memory that eliminates the race condition issue. However, to access remote memory locations processing elements have to communicate data to each other. Communication is performed through the interconnection network, which is a key factor of the multi-processor architecture scaling.

Distributed memory programming model is based on the parallel task execution which typically communicates by send/receive message passing. The key issue in distributed memory programming is the complexity of data distribution schemes. It forces programmers to explicitly manage data location.

The `Message Passing Interface` (MPI) is a standardized communications protocol that consists of a set of routines for implementing message-passing programs in different

programming languages such as FORTRAN, C, C++. MPI remains the dominant model used in high-performance computing.

In addition to described basic memory organizations there are also adjacent scenarios. In **distributed shared memory** the physically distributed memories are logically shared and can be addressed as a global address space.

The multi-core/processor programming complexity which includes thread synchronization, control and data dependencies, memory allocation brings new challenges to trace-driven simulation.

In the case of node-level exploration when cores are abstracted away, extra mechanisms that allow managing resource sharing and application control flow are required.

4.3 Methodology

In this section, we present the methodology based on the previously described general concepts of the proposed hybrid trace-oriented approach. Here we define the key features of the proposed approach:

- **Abstraction level:** We focus on the node-level exploration thus processor cores are replaced by trace injectors and memory traces are used in the proposed trace-driven simulation.
- **Target system:** Multi-/manycore architectures exploration is the primary goal of this research.
- **Trace reduction:** Cache miss filtering technique is applied in order to reduce trace file sizes and speedup the simulation.
- **Synchronization:** To ensure control and data dependencies a trace synchronization technique is proposed. It is based on the complementary synchronization traces collected from the annotations at application source code.
- **Trace replication:** To enable simulation of manycore architecture made of M cores based on traces captured in a reference simulation on a system comprising N cores, with $M \geq N$ using trace replication technique.

- **Microarchitecture:** Microarchitecture details are hidden inside the trace injector architecture. The out-of-order instruction execution is not the target of the current study. Computation phase acceleration technique is proposed to emulate different processor models.

4.3.1 From collection to simulation

We define the target system as consisting of the *hardware architecture* and *executed software*. In Figure 4.5 (a), the hardware architecture comprises N cores, each one having its private instruction (I) and data (D) caches. Communications between cores and external memory is achieved via an arbitrary communication infrastructure that may comprise caches (L2, L3).

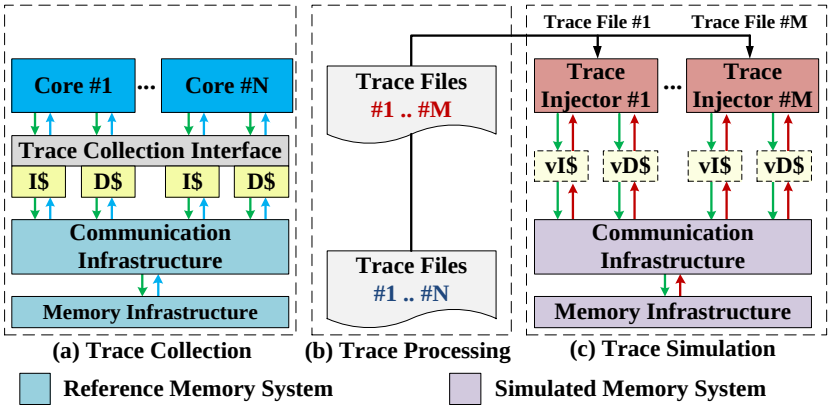

FIGURE 4.5: Three phases of trace-driven approach.

Figure 4.5 depicts communications as a stream of *request* and *response* events via the communication and memory infrastructures. The structure of a trace associated with a request event is a record of the time at which the event occurs, the memory access type (read or write), destination address and data value. The structure of a trace associated with a response event is a record of the time when the associated request event is satisfied. The *trace collection interface* shown in Figure 4.5 (a) tracks and stores all request and response traces. It is located at the interface of each private cache memory. This allows capturing information on the core side memory requests and corresponding memory side responses. As a global output of collection phase a trace-set consisting of one trace-file for each core is obtained.

The trace-driven reduction phase is intended to prepare the collected traces for further use. To reduce the trace file size our approach applies trace filtering technique therefore only traces related to cache miss event are considered. This technique allows significantly improving simulation speed as not only processor computation events are abstracted but also these related to cache hits. Simulation speedup obtained by cache miss filtering inversely depends on the cache miss rate. According to [110], filtering techniques show good accuracy. However, such aspects of cache memory behavior as cold start bias, replacement policies, writebacks should be taken into account.

The trace-driven simulation phase depicted in Figure 4.5 (c) takes the augmented vector traces as inputs, and per-core traces are injected in the communication and memory subsystem by *trace injectors*. I$ and D$ are replaced by their virtual implementation (vI$ and vD$) to manage cache misses only and therefore anticipate false hits caused by empty registers.

4.3.1.1 Synchronization traces

Software system includes applications and an operating system. Collected memory traces contain computation-to-communication application behavior but does not provide synchronization mechanisms supported by operating system.

In multithreading programming model which is dedicated to shared memory multicore architecture, multiple threads exist within the same process and share resources. Each core executes a separate thread simultaneously. As a trace set of one trace-file per each core is obtained after the collection phase we assume that each trace-file corresponds to one thread. Multithreading programs contains synchronization points therefore making execution behavior non-deterministic. That is, between two synchronization points task execution depends on the multiple runtime factors, e.g. cache misses, interconnect traffic, memory bandwidth. Once traces are collected, nothing can guarantee that tasks will be executed in the same way.

Figure 4.6 illustrates the basic points of multithreading program execution. The synchronization points (barriers and join) introduce control-flow dependencies, which are typically managed by programming APIs. We propose the *trace synchronization* technique which allows to control the trace simulation flow according to the runtime factors.

The additional *synchronization traces* are collected at the moments of APIs call and then are inserted into memory traces.

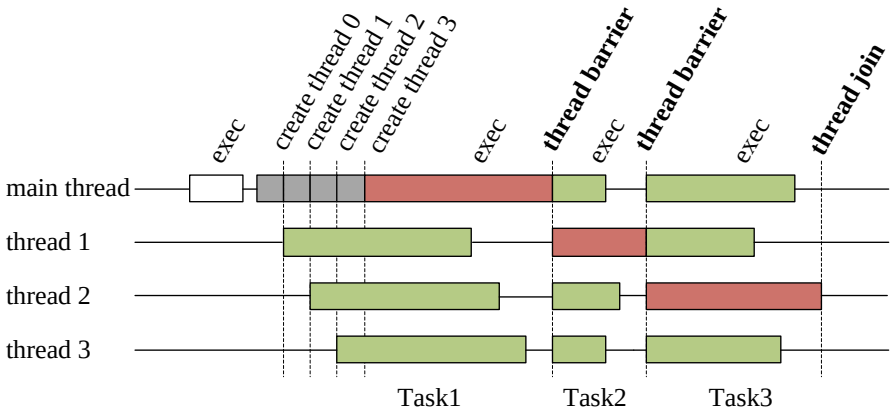

FIGURE 4.6: Shared memory programming.

For the target system with distributed memory organization, the same mechanism of synchronization traces collection can be applied. Figure 4.7 illustrates the basic points of MPI program execution. In this case synchronization traces are collected at the moment of message passing.

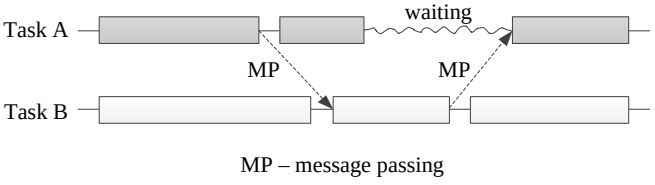

FIGURE 4.7: Distributed memory programming.

4.3.1.2 Trace replication

To enable efficient manycore architecture exploration the proposed approach features the trace replication technique. It implies collection of traces on a reference system comprising N cores and simulation of replicated traces on a target system comprising M cores, with $M \geq N$.

Figure 4.8 illustrates how the trace pattern looks like when a multithreaded application is executed. In regular applications, computation and communication phases alternate orderly within each thread. In the case of *data-dependent* application, computation phases depend on the input-data size therefore trace pattern cannot be correctly replicated.

If the application is *not data-dependent*, computation phases are deterministic and communication phases are not-deterministic, e.g. depend on the interconnect traffic. Replicating such trace pattern, we multiply the virtual size of processed data. During communication phase multiple memory operations, e.g. read or write, are performed. Accesses may be either to shared or to private memory blocks. Distribution of memory operations is related to application nature. This aspect must be taken into account when replicating the trace pattern.

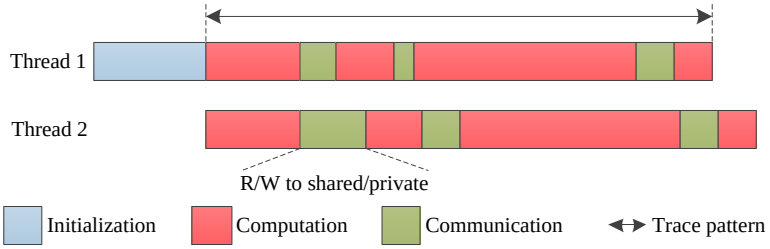

FIGURE 4.8: Replication trace pattern.

4.3.1.3 Computation phase scaling

In some cases, traces collection or replay for a specific CPU model can be challenging. In order to emulate different CPU model behavior using traces collected on reference system we propose the technique for computation phase scaling. The proposed approach implies application of an approximately estimated coefficient to computation phase duration as shown in Figure 4.9. 'CPU model 1' represents the reference trace simulation and 'CPU model 2' is the target trace simulation where an acceleration coefficient $k = 1.8$ is applied to reference computation phases.

Such technique does not guarantee high accuracy level because a computation phase usually includes various operations (e.g. floating point, integer) which execution ratio may severely differ. Therefore, it requires detailed analysis for each target application.

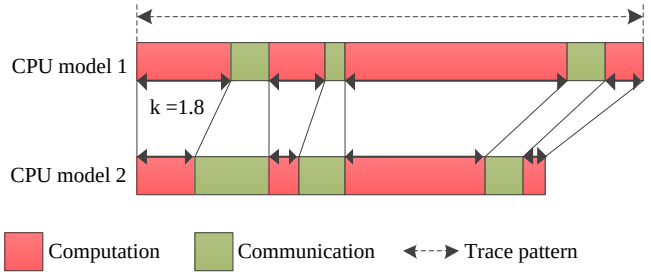

FIGURE 4.9: Computation phase acceleration.

4.3.2 Case studies

In this section, we present the design space exploration which the proposed trace-driven simulation covers. Figure 4.10 illustrates how the collected on the reference system traces can be used. In the left side of the figure, two collection cases are presented: reference system which is combined on four processing elements executes four-threaded **Application A** and **Appication B**. Figure 4.10 b) shows exploration scenarios, which are based on two trace sets. We divided them into four groups. They also can be combined in a different way.

1. First group focuses on interconnect and memory system exploration. Processing elements are replaced by trace injectors, their number and type remain the same as in reference system. Only interconnect/memory configurations are modified.

2. Second group addresses multitasking execution exploration where two trace sets are merged under one simulation. This group targets application and OS studies.

3. Third group concerns heterogeneous multicore exploration. Heterogeneity in this case is achieved by applying an acceleration coefficient at trace injection mechanism. This approach is used in order to emulate different core architectures.

FIGURE 4.10: Trace-driven simulation case studies.

4. Last simulation group aims at manycore architecture exploration through trace replication technique. This technique lies in actual duplication of collected traces in order to emulate larger number of cores as the reference system contains. This exploration scenario allows to extend the collection system capabilities and to switch from multicore to manycore architectures.

4.3.3 Evaluation metrics

We evaluate the proposed trace-driven approach on two criteria: simulation speedup and accuracy.

Simulation speedup. To identify the gain in terms of simulation speed of the trace-driven approach, we consider simulation process from two perspectives: full event-driven simulation and corresponding trace-driven simulation. These perspectives are illustrated in Figure 4.11, where a target architecture consists of four communicating layers corresponding to processing elements/trace injectors, private cache memory, interconnect and external memory. The time intervals T1 and T2 respectively represent the induced durations in case of cache miss and cache hit.

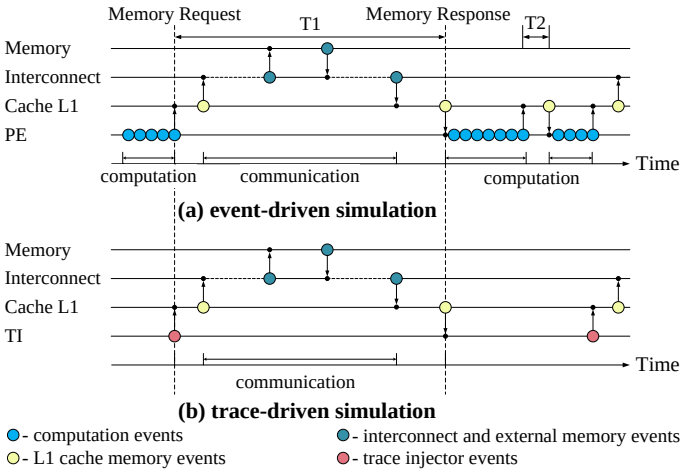

FIGURE 4.11: Comparing event-driven and trace–driven simulations. Events represent inter-communication activity between components.

In order to quantify the speedup of the trace-driven approach compared to a full event-driven simulation, the following event sets are considered: (set E_{PE}) for all events related to PE computation, E_{cache} same for cache memory access events composed of cache hit events (set E_{hit}) and cache miss events (set E_{miss}), E_{comm} same for communication events (covering the interconnection and external memory) and E_{TI} same for trace injector events. In addition, we denote by ξ_{ED} the set of all possible events occurring during the full event-driven simulation. This set is defined as the union of (E_{PE}), E_{cache} and E_{comm}. Given a set E, the denotation $|E|$ represents the cardinality of E.

The trace-driven simulation which is shown in Figure 4.11 (b) replaces PEs of Figure 4.11 (a) by TIs. The PE events are therefore replaced by TI events. Due to trace reduction, the set E_{TI} only includes cache miss events.

Then, the set ξ_{TD} of events composing trace-driven simulation is defined as the union of E_{TI}, E_{miss} and E_{comm}.

To identify the gain in terms of speedup obtained with the proposed trace-driven simulation, we first observe that the full event-driven simulation time T_{ED} equals to $T_{PE} + T_{cache} + T_{comm}$ (i.e., the simulation time corresponding to PE, cache and communication), while the trace-driven simulation time T_{TD} is $T_{TI} + T_{miss} + T_{comm}$ (i.e., the simulation time corresponding to TI, cache miss and communication). Then, we need to establish a relation between the simulation time and the number of simulated events. For a set E of events, where t_E is the simulation time for each event in E, the total simulation time T_E of all events of E is $t_E * |E|$. Here, we consider that this simulation time is proportional to the number of events according to event-driven paradigm.

Hence, the gain G in terms of speedup is given by the ratio between the simulation durations in the two simulation modes, as follows:

$$G = \frac{T_{ED}}{T_{TD}} \approx \frac{t_{E_{PE}} * |E_{PE}| + t_{E_{cache}} * |E_{cache}| + t_{E_{comm}} * |E_{comm}|}{t_{E_{TI}} * |E_{TI}| + t_{E_{miss}} * |E_{miss}| + t_{E_{comm}} * |E_{comm}|} \tag{4.1}$$

The above gain is implicitly related to numbers of events in the two simulation modes. For computation-intensive applications, $|E_{PE}|$ is significantly greater than $|E_{cache}| + |E_{comm}|$. Then, the gain provided by the trace-driven approach is observed by the fact that the events related to PEs are abstracted away via $|E_{TI}|$, which accelerates the simulation. For communication-intensive applications, the gain is less important since we only keep the events related to cache miss, which reduces the simulation time in a smaller proportion.

Simulation accuracy The main metric that we use to evaluate an architecture performance is execution time (see Chapter 3). For the trace-driven simulation, execution time is calculated as time between the first trace record and the last simulated event that is last response arrival.

Let us consider a reference trace collected from the trace collection phase in Figure 4.5. We replay this trace according to the trace processing phase. First, we configure the memory and/or communication infrastructures in a different way compared to the initial system in trace collection phase. The resulting trace injector has to deal with the fact that memory responses may arrive at different moments compared to the timeline recorded in the reference simulation. This is illustrated in Figure 4.12. Trace injector sends a request at time t_1, which reaches the memory at time t'_2, while in the reference

simulation this request was expected at t_2 (represented by bullets with dashed borders). This implies a shifting delay that will postpone the response to the injector at time t'_3 instead of t_3. As a result, observed temporal behavior changes. The shifting delay is determined by the trace injector during the simulation. It is used online to dynamically adjust the occurrence times of subsequent events.

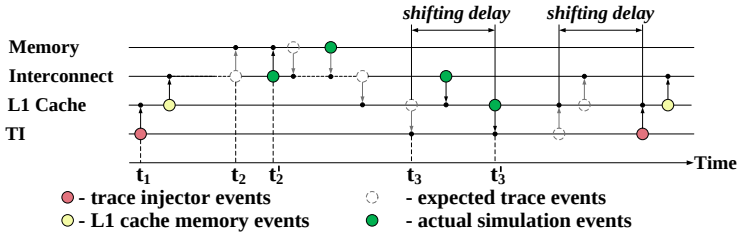

FIGURE 4.12: Dynamic allocation of the trace-drive simulation events in case of memory system modification.

The accuracy of the proposed trace-driven simulation is reflected by the observed variations of the temporal behaviors of a system upon different memory and communication configuration settings. To assess this accuracy, we consider the full event-driven simulation as the reference result. Given the execution time x_{ED} captured from event-driven simulation and the corresponding execution time x_{TD} obtained from trace-driven simulation, we define accuracy as follows:

$$\begin{cases} x_{TD}/x_{ED} * 100\%, & x_{ED} > 0 \\ 100\%, & x_{ED} = x_{TD} = 0 \end{cases} \tag{4.2}$$

The *error* percentage of the event-driven simulation regarding the reference result is:

$$\begin{cases} (x_{ED} - x_{TD})/x_{ED} * 100\%, & x_{ED} > 0 \\ 0\%, & x_{ED} = x_{TD} = 0 \end{cases} \tag{4.3}$$

Now, to define the *relative error*, we consider the number of events within a time slot δ of execution, which results from an arbitrary partitioning of the total execution time. Assuming $E_{miss} = E_{miss\_req} \bigcup E_{miss\_resp}$, the number of cache miss request events from a set E_{miss_req}, observed during δ is noted as $|E_{miss\_req|\delta}|$.

Restricting the simulation to cache miss requests, the relative error percentage of the event-driven simulation regarding the reference result, is:

$$\begin{cases} \frac{\left|E^{ED}_{miss\_req|\delta}\right| - \left|E^{TD}_{miss\_req|\delta}\right|}{\left|E^{ED}_{miss\_req|\delta}\right|} * 100\%, & \left|E^{ED}_{miss\_req|\delta}\right| > 0 \\ 0\%, & \left|E^{ED}_{miss\_req|\delta}\right| = \left|E^{TD}_{miss\_req|\delta}\right| = 0 \end{cases} \tag{4.4}$$

where $\left|E^{ED}_{miss\_req|\delta}\right|$ and $\left|E^{TD}_{miss\_req|\delta}\right|$ respectively denote the number of cache miss request events in full event-driven and trace-driven simulations.

4.4 Trace-driven implementation in gem5

We present the implementation of the proposed trace-driven simulation approach in gem5 according to the three main phases already distinguished in the previous section.

4.4.1 Trace collection and reduction

The first two phases, i.e. trace collection and reduction are treated at the same time. To implement the trace collection interface gem5 already has the necessary functionality that is part of its trace-based debugging which contains `DPRINTF` statements. Each `DPRINTF` is associated with a debug flag and refers to architecture components such as a bus, a cache, Ethernet controller, etc. To obtain specific information regarding memory requests/responses, it is enough to slightly modify the gem5 simulator code by adding new debug flags. It provides a flexible text-format file, which can be in some cases sub-optimal in terms of space and processing time.

The gem5 memory subsystem is based on the notions of port, packet and request/response. Ports connect memory components to each other. They support three types of accesses: (*i*) timing access, which is designed for the realistic timing modeling; (*ii*) atomic access, which is used for fast forwarding and is processed instantaneously, and (*iii*) functional access, which is leveraged for a specific purpose. A packet is used to encapsulate a transfer between two memory objects. Requests and responses are used to encapsulate CPU or I/O device messages. Our implementation of trace collection interface relies on these notions. Figure 4.13 (a) shows an extract of the collected trace file. Each memory access is introduced by the pair of events: request send and response

receipt. The request event record contains the following set of information: *time* when the event occurred, request *source*, i.e. core number and instruction/data cache port, memory access *type* (Read or Write), cache coherency *flags*, target physical *address*, packet size (8, 16, 32 or 64 bits) and *value*. The response event record contains only time and destination information.

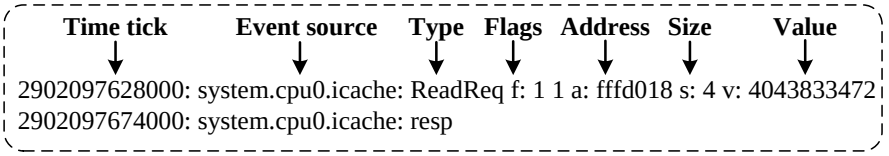

FIGURE 4.13: An example of collected trace file extract.

In this implementation, we consider a multithreading programming model and the dependencies management is maintained by synchronization traces together with trace arbiter. The first step of our synchronization mechanism consists in collecting additional synchronization traces. For that purpose, we add macros to input application source code to capture synchronization points. The gem5 specific operation `rpns()` returns the time at which it has been invoked in the format of internal time count, e.g. ticks. We create three macros, `CREATE`, `BARRIER` and `JOIN`, based on POSIX thread API and the `rpns()` operation of gem5 as shown in Figure 4.14 (a). The execution output of an application annotated with such macros is presented in Figure 4.14 (b). This makes it possible to automatically append in the execution trace synchronization information that are later used.

4.4.2 Trace simulation

For the simulation of the augmented vector traces, two main components are implemented in gem5 simulator: (i) *Trace Injector* and (ii) *Trace Arbiter*.

The trace injector consists of two fundamental modules: the first is dedicated to parse a trace file and to provide request/response structures, whereas the second triggers request injection and time interval management, i.e. time shifting of upcoming requests according to response times.

The second important component is trace arbiter. It is connected to each injector and has a global view of the entire system. Arbiter deals with dependency traces collected

```
#define PTHREAD_CREATE(...)   pthread_create(&threads[i], 0,func,(void *) &params);
                              printf("Thread id %i create tick: %lu\n", ID, rpns());
#define PTHREAD_BARRIER(...)  pthread_barrier_wait(barrier);
                              printf("Thread id %i barrier tick: %lu\n", ID, rpns());
#define PTHREAD_JOIN(...)     pthread_join(threads[i], 0);
                              printf("Thread id %i join tick: %lu\n", ID, rpns());
```

(a) Application macros for dependency management

```
Thread id 0 create tick: 1627449500
Thread id 1 create tick: 1627690378

Thread id 1 barrier tick: 1628594414
Thread id 0 barrier tick: 1628632384

Thread id 0 join tick: 1727194499
Thread id 1 join tick: 1727322559
```

(b) Dependency traces collected via macros

FIGURE 4.14: An example of collected synchronization traces.

via the macros. Communications between the arbiter and injectors are achieved via a simple protocol. Trace injector has two basic states: LOCK and UNLOCK. As soon as the trace injector reaches a synchronization entry in the trace, it sends a corresponding signal to the arbiter. The arbiter sets the injector to LOCK state and waits until all the other injectors reach the same synchronization point. Once it happens, the locked arbiter unlocks all injectors and the simulation continues. Exchange of messages between the arbiter and the injectors is carried by atomic packets and does not affect the runtime. Arbiter and injector connections are shown in Figure 4.15.

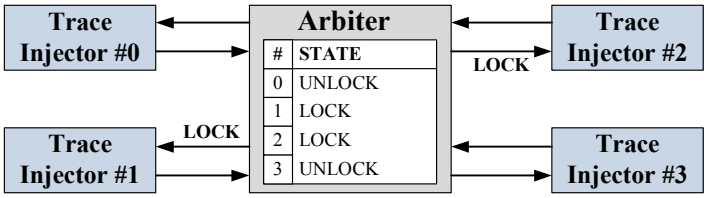

FIGURE 4.15: Trace arbiter.

Figure 4.16 shows the block diagram of trace injector internal logic.

After initialization, trace injector parse the trace file and identifies the trace type. If the trace corresponds to the memory access, trace injector inserts the *Send* event into global event queue. The request will be sent automatically by the framework. Then the trace injector is waiting until the memory request arrives. This time is not deterministic and depends on the runtime factors.

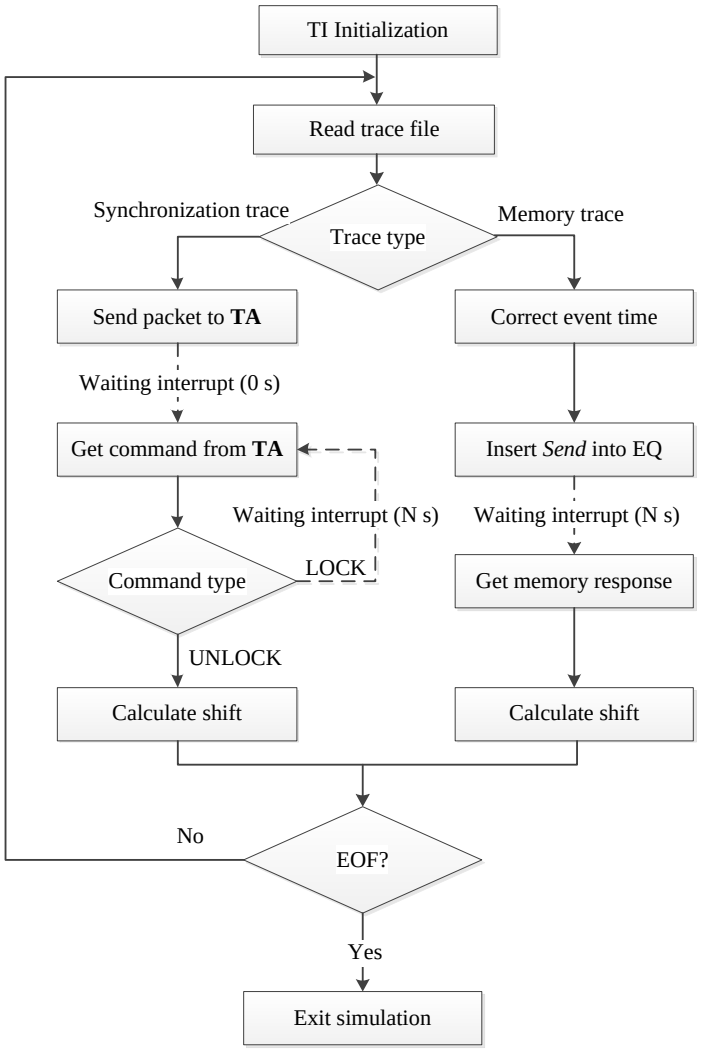

FIGURE 4.16: Trace injector block diagram.

In the case of *synchronization traces* trace injector sends an *Atomic* packet to trace arbiter and at the same cycle receives the command. If the command is LOCK, it will wait until the UNLOCK arrives. This time is also not deterministic.

The *calculate shift* blocks indicate where the runtime shift is calculated. It is then used to correct the next event time.

Yet another important modification concerns the cache. Since the trace-driven simulation starts with a cold (empty) memory, it causes a form of bias. This problem is usually referred to as cold-start bias [121]. For private caches, we exploit the fact that

all memory requests are cache misses and are declared as such for the trace-driven simulation. This solution eliminates cold-start error, but brings a new restriction: each individual trace file collected through one cache size cannot be simulated with other cache sizes. For the other levels of cache, e.g. L2, there are a few methods, discussed in [121], reducing the problem.

Limitations. The proposed approach has three main constraints. The first is related to the core and L1 cache abstraction. It requires recollecting traces and changing their configuration, e.g. core frequency, L1 cache size, etc. The second constraint concerns the nature of considered applications. The proposed trace replication approach does not support applications in which the computation phase cannot be statically bounded, e.g. data-dependent applications for which this phase depends on input data. The third constraint concerns simulation host machine capabilities. Indeed some traces could take tens of gigabytes disk space and tens of gigabytes operating memory for 512 cores simulation. Thus, considered machines should provide enough storage space.

4.5 Evaluation

In this section we evaluate the following aspects of the proposed approach and its implementation:

- simulation accuracy assessment throughout several case studies compared to the reference full system simulation;
- evaluation of achieved simulation speed gain in comparison to the corresponding full system simulation;
- simulation cost evaluation in terms of hard disk space required for traces, simulation runtime distribution among trace-driven system components and host machine memory occupancy depending on the number of components.

4.5.1 Experimental setup

We validate the proposed trace-driven approach by providing a detailed analysis of the simulation process. The reference platform is built on the top of gem5 framework in

full system simulation mode. In Chapter 3 we provided a detailed analysis of the gem5 simulator by evaluating the accuracy level against the real hardware. We assume that in addition to the previous validation, the comparison to the full system simulation provides an acceptable vision of the proposed approach accuracy.

Our reference platform is characterized by a set of architecture parameters: (i) 1-,2-,4-,8-cores processor running at 500 MHz, (ii) 4-kB private L1 D/I caches, (iii) 64 bits channel width, (iv) 30 ns DDR memory latency and (v) Linux Kernel 2.6.38. The fastest ARM ISA *TimingSimpleCPU* (in-order) model is used. The processors are connected to the DDR memory via a coherent bus.

Benchmark. We consider a set of applications from scientific and multimedia computing domain, implemented in POSIX Threads. Some of them come from the SPLASH-2 benchmark suite [95]: *radix*, *barnes*, *lu* and *ocean*, which are relevant due to the presence of multiple dependencies in the corresponding algorithms. In addition, we adopt further applications: Motion JPEG (*MJPEG*), encoding video streams *H.264*, Finite Impulse Filter (*FIR*), Smith Waterman (*SW*), histogram for histogram graph computing (*hist*), *Merge Sort*, *N-body* for simulating a dynamical system of particles, Reduction of vectors (*reduct*), Vector Operations (*VO*) and Fast Fourier Transform (*FFT*).

The list of applications, their detailed description, problem size per thread and collected trace size are presented in Table 4.1.

TABLE 4.1: Applications description.

Application	**Domain**	**Problem size**	**Trace size (Gb)**	**Execution time (ms)**	
				FS	TD
SPLASH-2					
Barnes	Scientific	512	1.04	161.7	161.3
LU	Scientific	64	2.51	479	484
Ocean	Scientific	16	2.48	602.7	602.5
Radix	Scientific	16 384	0.99	18.8	19.1
Mont-Blanc					
Hist	Scientific	2 * 1024 * 1024	0.06	267.2	264.9
Merge Sort	Scientific	512	5.48	48169	48150
N-body	Scientific	1024	0.66	881.1	880.9
Reduction	Scientific	4 * 1024 * 1024	2.49	371.6	371.5
VO	Scientific	128 * 1024	0.92	301	300.9
FFT	Scientific	1 * 512 * 1024	11.6	1745	1852
Adopted					
MJPEG	Multimedia	203 kB	0.04	26.4	26.2
FIR	Scientific	1024	0.87	454.9	454.6
SW	Scientific	6 x 2	0.05	18.6	17.8

4.5.2 Simulation accuracy

Since the proposed approach provides multiple case studies that are described in Section 4.3, we propose to consider several accuracy evaluation scenarios:

- The first scenario is intended to validate the trace injection functionality and identify the possible modeling and abstraction errors.
- The second scenario evaluates the accuracy of interconnect and memory system exploration.
- The third scenario assesses the trace replication technique.

Functionality validation. The trace files are collected from the above reference platform with eight cores, and then are executed in the proposed trace-driven simulator without any architectural changes. The comparison results are shown in Figure 4.17. These results include the execution time obtained on the reference platform and on the evaluated trace-driven system, as well as the comparative error percentage. Note that the execution time scale is logarithmic. The execution time error of the proposed trace-driven simulation compared to the corresponding full system simulation is within 0.1% and 5.8%. The execution time of chosen applications varies from tens of milliseconds to thousands of seconds thus we can guarantee that the error does not depend on the duration and is not accumulated. The evaluated applications combine these without dependency issues and applications which contains multiple synchronization points, namely *radix*, *lu*, *ocean* and *barnes*.

The mechanism that we defined for addressing the trace synchronization issue has been presented in Sections 4.3. The dynamic behavior of the trace execution is illustrated on the integer *radix* sort kernel.

In the trace collection phase, the reference platform contains four cores and *radix* kernel is executed with four threads. For each core, the numbers of cache misses observed during a given time slot and the induced error are illustrated in Figure 4.18. The moments at which synchronization barriers occur are highlighted by dashed vertical lines. We observe a fluctuation of the error on barriers occurrences. However, the observed peaks in the error are compensated during the entire simulation, so that the cumulated error is only 1.39%.

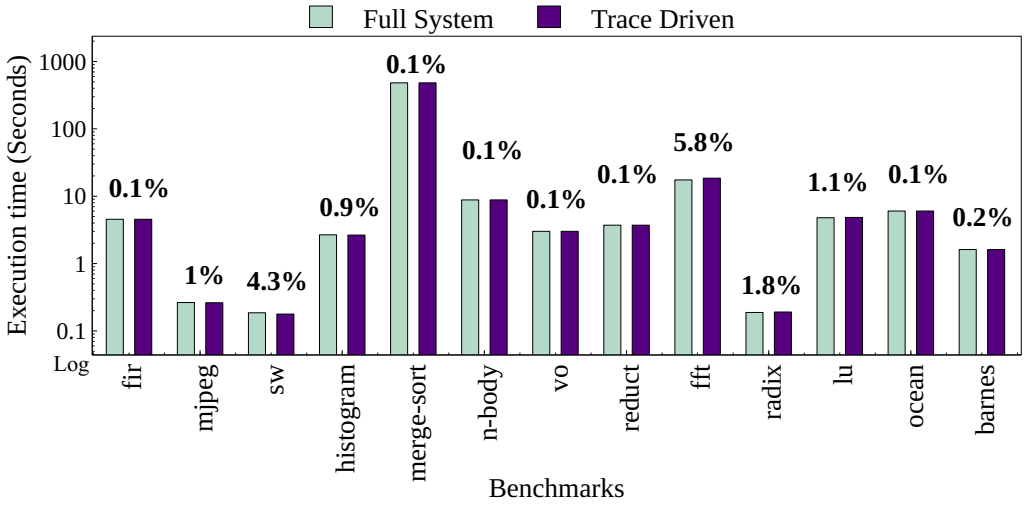

FIGURE 4.17: Execution time comparison between full system and trace-driven modes among all applications.

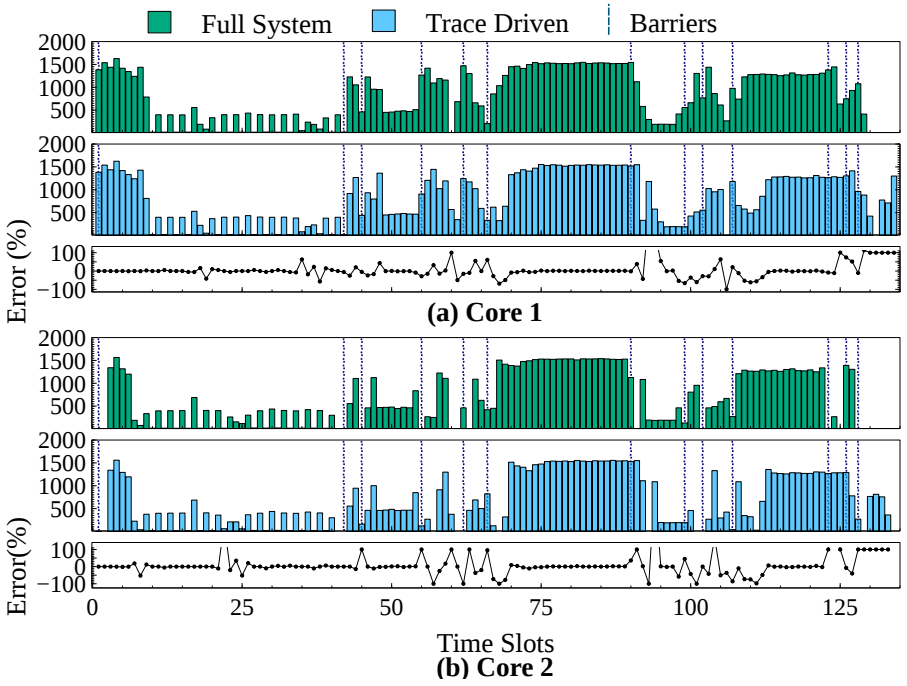

FIGURE 4.18: Cache miss pattern comparison between full system and trace-driven execution for *Radix* kernel.

Memory exploration validation. We evaluate the trace-driven simulation consistency by varying the internal architectural parameters just after the trace collection phase. Five memory latency values are used: 5ns, 15ns, 30ns, 45ns and 55ns. Their impact on two applications (*MJPEG* and *fft*) execution time is evaluated, as illustrated in Figure 4.19. The results show that the error in terms of execution time is around 6%. Thus, the proposed trace-driven simulation reproduces the application behavior properly even when architectural parameters change.

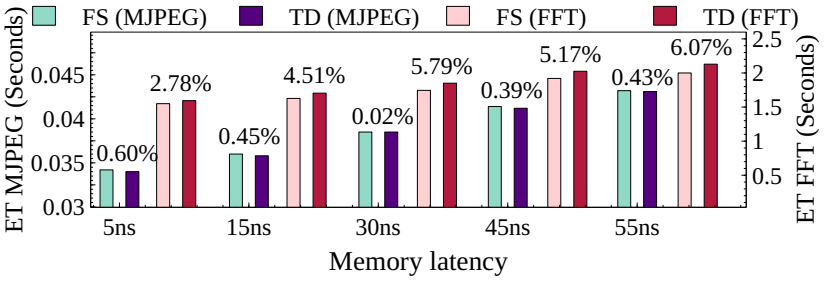

FIGURE 4.19: Execution time comparison between full system and trace-driven modes by varying the internal architectural parameters.

In another experiment, we used the captured traces to evaluate system configurations including components that were not present during the original trace collection phase. To illustrate this evaluation, traces are collected from our reference platform with eight cores running *mjpeg*, and then transferred to a trace-driven system, which includes an L2 cache shared between all trace-injectors.

Here, in order to minimize the error in terms of execution time, we must address the cold-start bias issue as discussed in Section 4.4. We propose to warm up L2 cache by considering traces captured before application execution phase. We collect and compare three traces: (i) execution time traces (ET), (ii) execution time with initialization phase traces (ET + init) and (iii) execution time with initialization and OS boot phases. The results are presented in Figure 4.20. The traces in (i) lead to 14.01% of absolute error on average. By using the traces in (ii), we obtain an absolute error of 7.88% on average, which is two times less than provided by (i). The traces in (iii) give 6.60% of absolute error on average, which is the best.

Trace replication validation. We evaluate the opportunity of simulating the replication of a given trace set on more processing elements than those used to collect this trace set. A preliminary observation about cache miss behaviors is that they are similar.

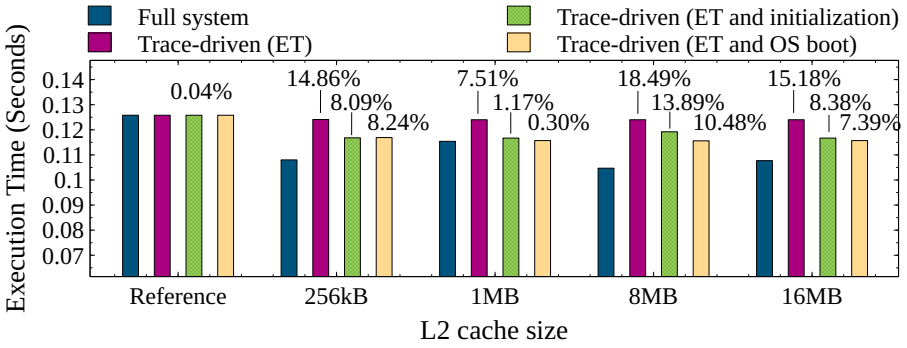

FIGURE 4.20: Execution time comparison between full system and trace-driven modes by including L2 cache memory.

The cache misses of all cores follow the same memory access time pattern. Then, such a pattern can be used as a possible trace template to be replicated among more cores. We explore different scenarios for identifying such a pattern by considering the *mjpeg* application, where traces are collected from: (*i*) 1-core reference platform, (*ii*) 2-cores reference platform, (*iii*) 4-cores reference platform and (*iv*) 8-cores reference platform. The traces obtained from these scenarios are replicated on a platform with eight trace injectors. We use normalized correlation function to produce an accurate estimation. Correlation is made between the number of cache misses obtained through the 8-cores gem5 full system simulation and the number of cache misses obtained via the above four scenarios. The obtained correlation coefficients for each scenario are shown in Figure 4.21.

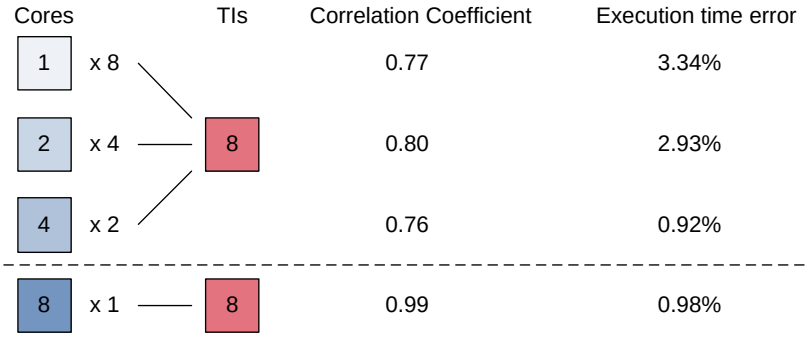

FIGURE 4.21: Replication technique: correlation coefficient and execution time error analysis.

These results show that the three scenarios provide very similar behaviors regardless of the number of cores used. The produced execution time error comparing to the full

TABLE 4.2: Application problem size impact on correlation coefficients.

Application	Radix	LU	Ocean	Barnes
Problem size 1	0.82	0.65	0.57	0.59
Problem size 2	0.87	0.75	0.66	0.73
Problem size 3	0.90	0.90	0.80	0.80

system simulation is under 4%. Thus, the 2-cores scenario is relevant enough for a meaningful replication targeting up to hundreds of injectors.

In order to improve the correlation coefficients, we study the impact of application problem size on these coefficients. We focus on four applications: *radix*, *lu*, *ocean* and *barnes*. For each of these applications, we chose three problem sizes, where Size 1 $<$ Size 2 $<$ Size 3. Here, size means the amount of processed data. Then, we collected their corresponding traces on the reference platform with four cores and four threads. Table 4.2 shows the average correlation coefficients calculated for each application trace according to the three problem sizes. Note the significant increase in correlation with larger problem sizes, originating from the increased pressure to memory subsystem: a higher dynamics in the cache miss rate over time results in more prominent correlation.

The correlation studies concern only time component of the trace pattern. However, the second important aspect of trace replication technique is memory access address distribution. As the physical address field is directly accessed by the trace injector it provides a powerful tool to memory mapping exploration.

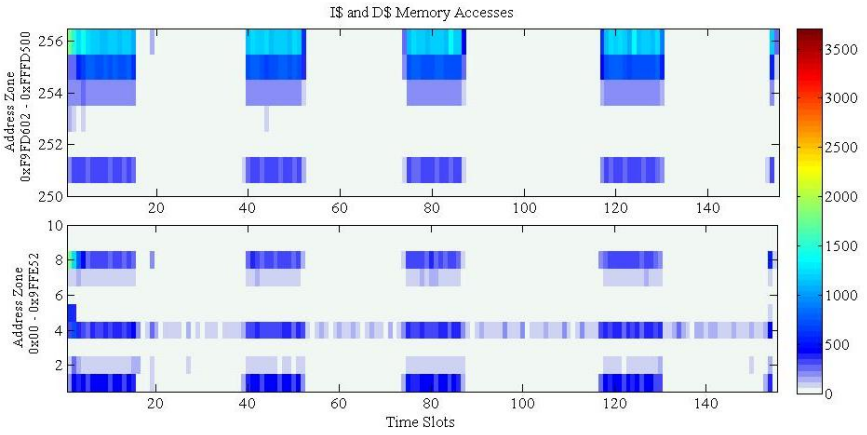

FIGURE 4.22: Replication technique: address map of MJPEG 1 core 1 thread.

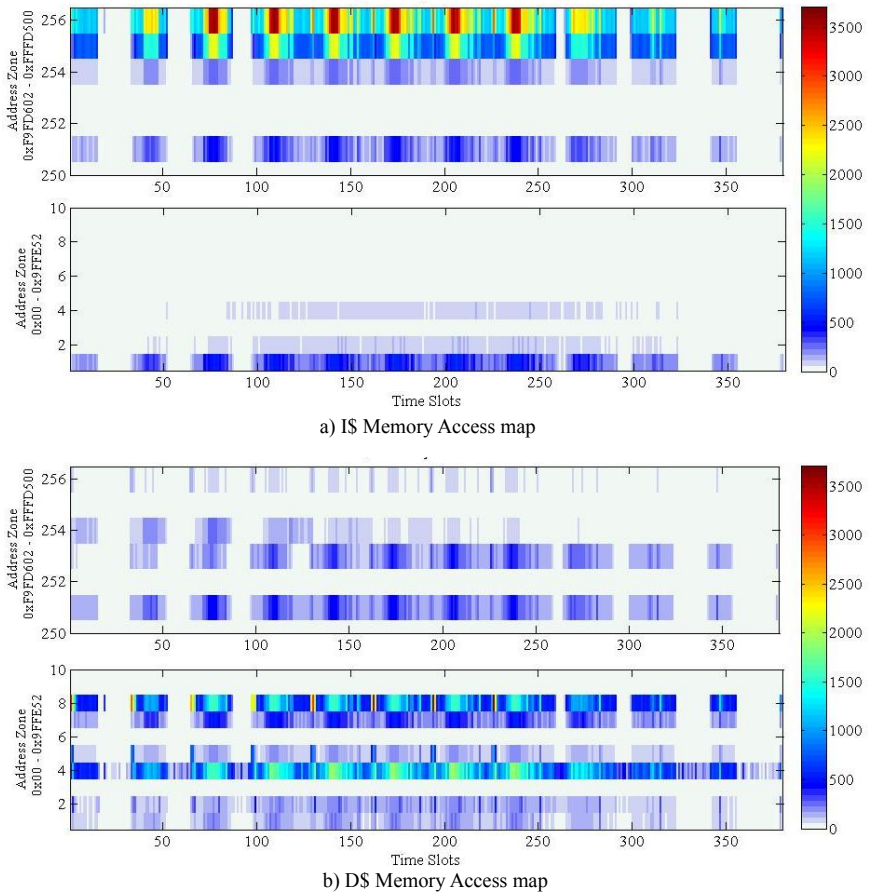

FIGURE 4.23: Replication technique: address map of MJPEG 8 cores replication.

In Figure 4.22 we presented the timing memory access map for a single-core one-threaded *mjpeg* execution. By replicating that trace we applied the following principle: instruction cache accesses are replicated with the same address map, data cache addresses are randomly shifted for each trace injector. The resulting timing memory access maps for instruction and data caches are demonstrated in Figure 4.23 a) and b) respectively.

Such distribution allows emulating the shared and private thread data distribution. More complex algorithms can be applied for data mapping exploration.

4.5.3 Simulation speedup

To demonstrate the achieved simulation speed gain we run a set of applications on gem5 full system and trace-driven systems with eight cores/trace injectors. The simulation time comparison as well as relative speedup are shown in Figure 5.5.

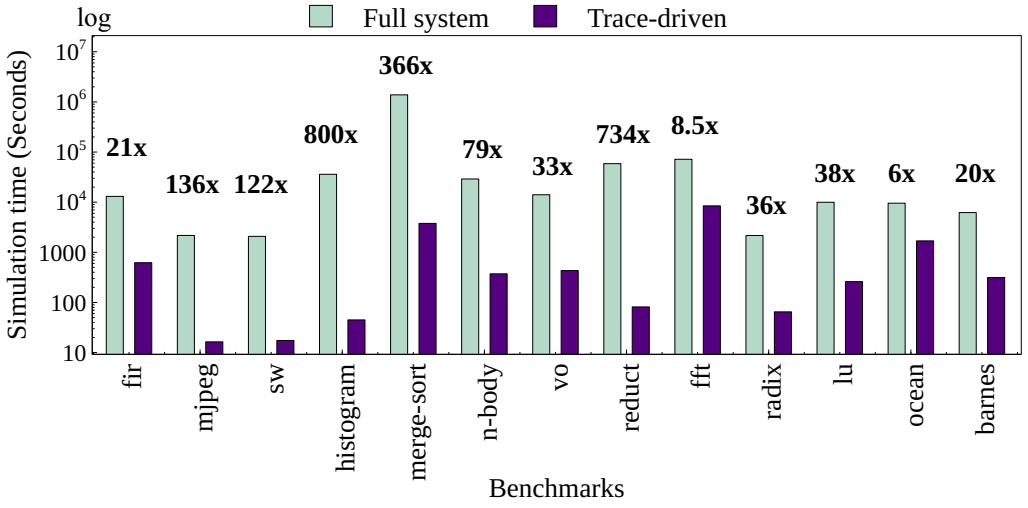

FIGURE 4.24: Simulation time comparison between full system and trace-driven modes.

The results provided by the trace-driven simulation show that the gain in terms of speedup is in a fairly wide range of $6x - 800x$. Such discrepancy is due to the application nature. That is computation-intensive applications provide higher simulation time gain than communication-intensive applications.

We chose two applications, e.g. *mjpeg* and *h.264*, in order to illustrate the simulation speed dynamics when changing the number of cores in system architecture. The simulation time comparison among three scenarios are shown in Figure 4.25: full system simulation of an application with Linux kernel boot, only application full system simulation and corresponding trace-driven simulation.

With the increasing number of cores, the simulation time in full system modes rises significantly, while in trace-driven mode it varies slightly.

Simulation scaling issue required further analysis. For that reason, we apply our trace replication technique to demonstrate the simulation time behavior for up to 256 injectors. The results are shown in Figure 4.26. Note that the scale for the simulation time is logarithmic. We tested *mjpeg* application with 1 kB, 4kB and 16 kB L1 cache sizes, *histogram*, *vo* and *sw* applications with 4 kB L1 cache size. While on a small scale trace-driven mode provides an imperceptible simulation time growth on a large scale we observe a significant simulation slowdown. Such behavior is caused by the architecture component simulation. As the number of injectors increases the accompanying memory

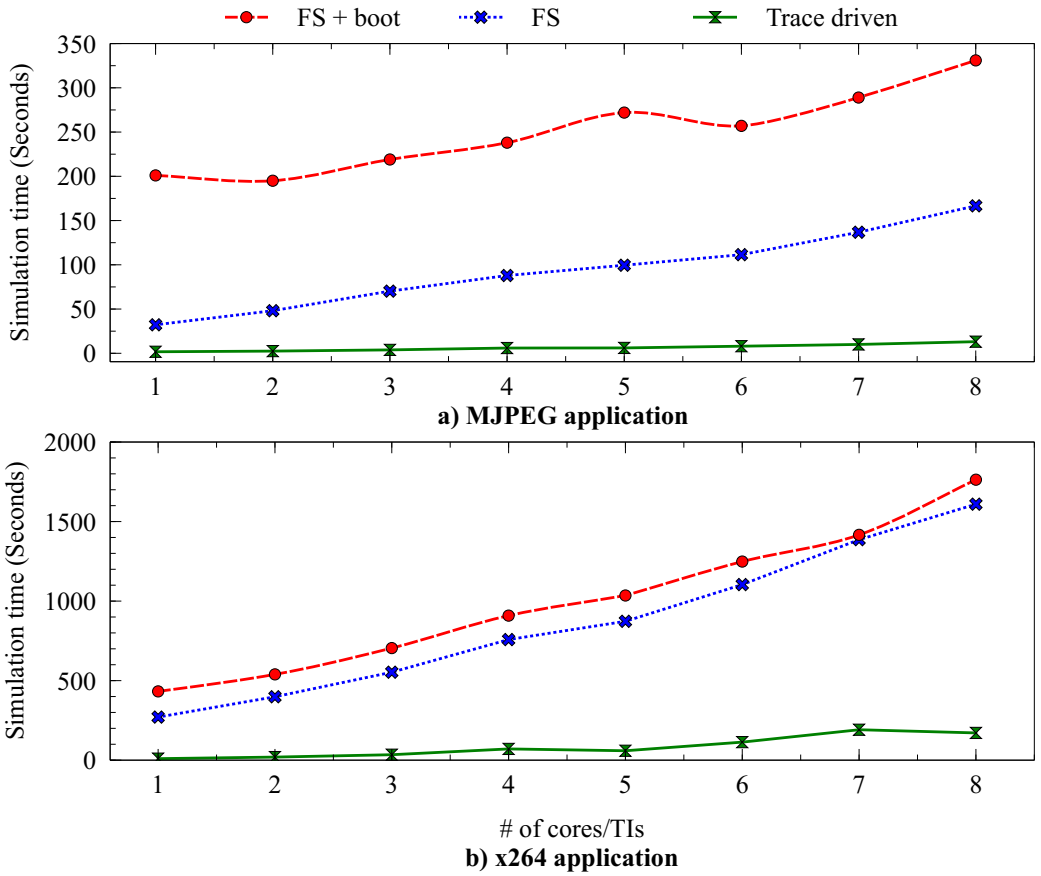

FIGURE 4.25: Simulation time scaling: comparison between full system and trace-driven modes.

traffic that includes cache coherency protocol messages also increases. We expect a similar behavior for full system mode simulation as well. However, the most time consuming simulation for 256 injectors takes 6 hours that is still acceptable.

4.5.4 Simulation cost

We here analyze the breakdown of the simulation effort on the host machine for the following components: trace injector, cache, bus, memory and gem5 simulator itself. A single core full system simulation is also given for reference. From the achieved experiments, we observed similar distributions that can be instantiated as in Figure 4.27 for the *mjpeg* decoder. We replicate the gem5 full systam simulation trace of this application into up to 256 injectors. The analysis is performed with the standard *gprof* profiling tool [122]. For comparison, the simulation percentage of a single CPU is 70% of simulation runtime while for a single trace injector it only takes 25%. The percentage

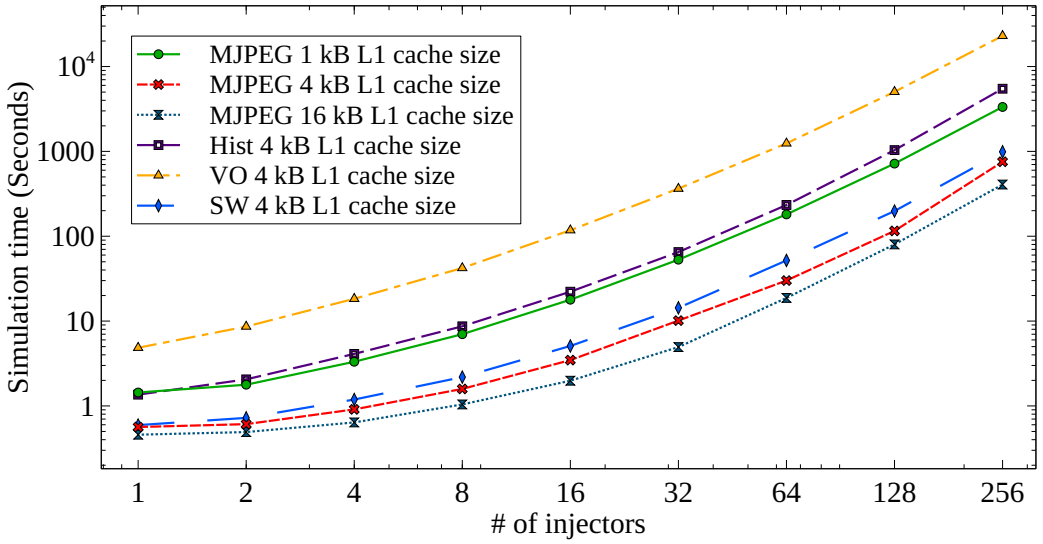

FIGURE 4.26: Simulation time scaling up to 256 trace injectors system.

of the cache simulation time increases with the number of injectors. For 256 injectors it amounts to 90%. The most part of simulation time is spent on cache coherency snooping protocol simulation. Since the target system is bus-based, the memory traffic congestion and performance degradation is induced by increasing processors and injectors.

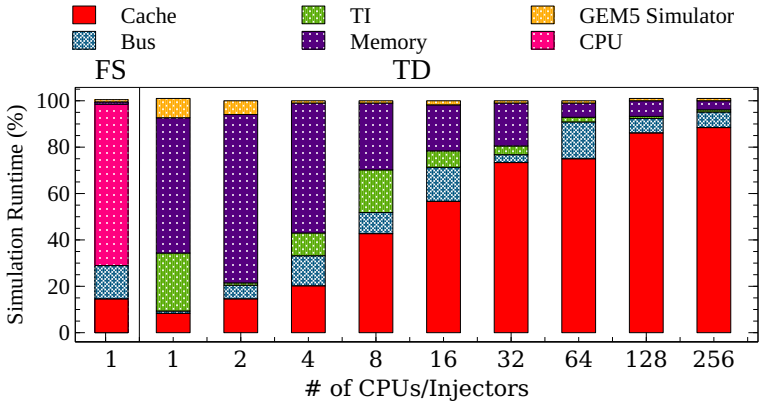

FIGURE 4.27: Simulation time distribution among cache, bus, memory and gem5 simulator for trace-driven simulation.

The last simulation characteristic that we explore is host machine memory occupancy. The main goal is to identify the impact of trace file size on host memory occupancy. For this experiment, we selected two applications with strongly different trace file sizes: *mjpeg* with 40Mb and *vo* with 1 GB. To analyze the simulation we used the Valgrind tool for memory debugging and profiling [85]. By replicating and running these applications,

the memory occupancy is same for both. The functions that consume the memory belong to the gem5 simulator itself. This is illustrated by Figure 4.28, where the memory occupancy is evaluated w.r.t. the number of injectors.

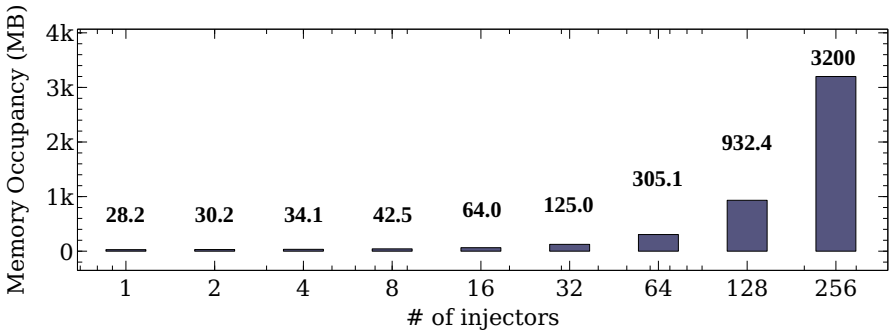

FIGURE 4.28: Memory consumption variation for trace-driven simulation.

4.6 Application to compute accelerators exploration

In this section, we demonstrate how the proposed trace-driven approach is applied to compute accelerators exploration. This work has been done in collaboration with R. Garibotti and has been presented in [24]. In [24], author aims to discover a compute accelerator configuration that achieves the best trade-off performance scalability versus power consumption. He propose a scalable and memory-efficient solution for compute accelerators explored through centralized shared memory and then distributed shared memory approaches.

The proposed trace-driven simulation approach has been adapted for this purpose so that collected traces are not injected in gem5 simulation framework but rather in a cycle-accurate simulation that contains SystemC NoC and memory models. Trace injectors read trace files and, according to the given memory mapping, construct the memory request transactions that are then fed into the NoC.

Figure 4.29 shows the five evaluated memory mappings. Two mappings are considered for CSM, denoted *Long Hops*. The other mapping, referred to as *Short Hops* consists in assigning all of the shared memory in a central PE for better reachability. Three mappings are evaluated for DSM. *Contiguous* mapping is the default mapping used in the initial DSM investigations. *Interleaved* mapping consists in assigning every other

memory position (32 bit word size) to another PE in a cyclic fashion. *Distributed* mapping assigns memory blocks on the sole basis of their nature: microkernel and applications. Within a category (i.e. OS or APP), contiguous mapping is used.

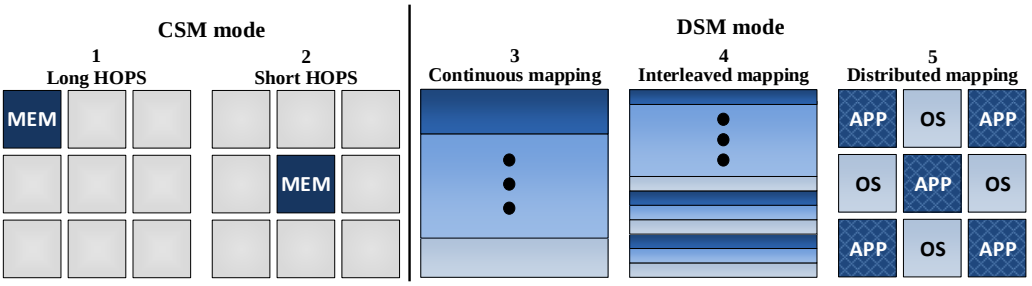

FIGURE 4.29: Different evaluated memory mappings.

A set of twelve application kernels has been used for benchmarking the presented mappings. Figure 4.30 shows the CSM long hops-normalized execution time for different vSMP cluster sizes ranging from 8 to 64 processing elements. CSM short-hops mapping leads to no visible improvement performance-wise. This originates from the memory bottleneck not addressed by this mapping, which only relocates all of the shared data to a central processing element (Figure 4.29). Conversely, interleaved and distributed mappings yield to significant speedup for most applications, especially for large array sizes.

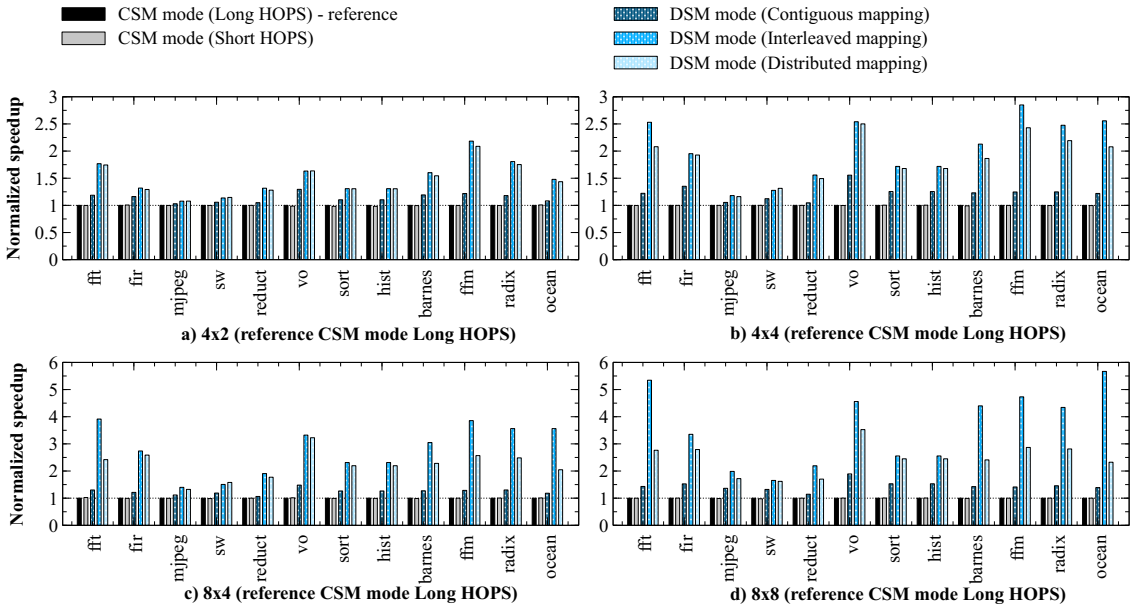

FIGURE 4.30: Execution time for different memory mappings and vSMP cluster sizes.

Figure 4.31 depicts the same information averaged over the entire set of benchmarks. For 8x8 PE array size, up to 3-fold reduction in execution time is observed which shows the criticality of memory access for such workloads.

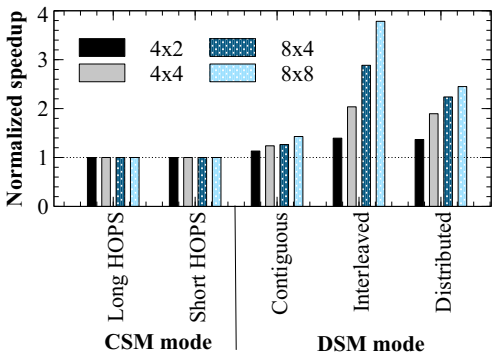

FIGURE 4.31: Normalized execution time averaged for all benchmarks.

4.7 Discussion

In this chapter we presented the hybrid trace-oriented approach for fast and accurate manycore simulation. We provided a detailed description of the approach implementation on the top of gem5 simulator. The comparison to the reference full system simulation showed the expected speed gain in a wide range of $6x - 800x$ depending on the application nature.

The great advantage of the proposed approach is related to its hybrid nature. That means that the trace-driven simulation is embedded into an event-driven environment so that the system architecture including cache, interconnect and memory is dynamically simulated at runtime. The trace-driven implementation was validated versus the cycle-approximate full system simulation. The results showed a 14% of average execution time error in the worst case architecture exploration scenario.

The proposed approach includes a set of techniques designed to enable accurate and flexible simulation. They includes: the trace synchronization technique, which allows managing control and data dependencies, the trace replication technique, which in tandem with the trace synchronization allows us to emulate a parallel application behavior for up to hundreds cores architecture and the computation phase scaling technique dedicated to fast and flexible switching between different CPU models.

Chapter 5

Single-ISA heterogeneous architecture exploration

5.1 Introduction

Among the popular initiatives, the use of low-power embedded SoCs to build energy-efficient supercomputers is regarded as a promising solution [5]. In this context, the use of mobile heterogeneous processor technology for further benefits in term of energy-efficiency appears attractive. Single-ISA heterogeneous multicore [6] processors are of particular interest because of being software-agnostic i.e. a unique standard SMP operating system may be used, taking advantage of load-balancing features for fine control over performance and power consumption.

In this thesis we explore performance and power trade-offs of popular single-ISA heterogeneous multicore ARM big.LITTLE processor. Detailed analysis of the existing Exynos 5 Octa SoC provides useful insight allowing us to infer alternative ARM big.LITTLE inspired architecture configurations that perform better energy-wise. For doing this the validated gem5 models in Chapter 3 and the proposed in Chapter 4 trace-driven simulation approach are used.

This chapter presents the ARM big.LITTLE architecture exploration. Section 5.2 introduces the background in heterogeneous single-ISA multicore architectures. In Section 5.3 we analyze performance and energy-efficiency of the existing Exynos 5 Octa SoC. Alternative big.LITTLE configurations are proposed in Section 5.4. Their performance and

power evaluation by using previously validated gem5/McPAT models are also shown in this section. Section 5.5 demonstrates the big.LITTLE architecture scaling by means of the proposed trace-driven simulation. Section 5.7 summarizes the results of the chapter.

5.2 Background

5.2.1 Single-ISA Heterogeneous multicore architecture

Heterogeneous multicore architectures usually consist of multiple cores that differ from each other from their instruction set architectures, their execution paradigms, e.g. in-order and out-of-order, their cache size and other fundamental characteristics. In single-ISA heterogeneous multicore architecture [6] all cores execute the same instruction set but differ in terms of performance and energy consumption.

In the mobile market, several SoC platforms operating on that principle exist, such as Nvidia Tegra 3/4 system-on-chip [7] and the ARM big.LITTLE technology integrated to Samsung Exynos 5/7 Octa SoC [8] as well as other SoCs from other vendors.

Possessing non-uniform computing environment under a unique standard SMP operating system, single-ISA heterogeneous multicores enable various adaptive software implementations. A large part of studies on single-ISA heterogeneous multicores focuses on efficient task scheduling for performance and power trade-offs. In [10] the authors propose a hierarchical power management framework for heterogeneous multicores, in particular for ARM big.LITTLE architecture, in order to minimize energy consumption within the thermal design power constraint. Yu et al. in [11] evaluates ARM big.LITTLE power-aware task scheduling, via power saving techniques such as dynamic voltage and frequency scaling and dynamic hot plug. Tan et al. in [12] implement a computation approximation-aware scheduling framework in order to minimize energy consumption and maximize quality of service, while preserving performance and thermal design power constraints.

5.2.2 OpenMP programming model

In order to take advantage of a single-ISA heterogeneous multicore architecture, an appropriate strategy to manage computation task distribution is required. Within a multithreading programming model, it refers to efficient thread scheduling.

The OpenMP [118] is a popular shared memory parallel programming interface. OpenMP features thread-based fork-join task allocation model. It consists of a set of compiler directives, library routines and environment variables for developing parallel applications. The OpenMP loop scheduling allows determining the way in which iterations of a parallel loop are assigned to threads. Iterations can be assigned in *chunks*, e.g. the number of contiguous iterations. There are three general loop scheduling types:

- **Static:** it is a default loop scheduling algorithm, which divides the loop into equal or almost equal chunks. Chunk size therefore is calculated as $number\_of\_interations/number\_of\_threads$. Simple static scheduling provides the lowest overhead, but the potential load imbalance can cause significant synchronization delays.
- **Dynamic:** it allows assigning chunks at runtime once threads complete previously assigned iterations. The internal work queue of chunk-sized blocks is used. By default, the chunk size is '1' and can be explicitly specified by a programmer at compile time. Dynamic scheduling allows better balancing the load among threads, but provides higher overhead costs.
- **Guided:** it is similar to dynamic scheduling, but the chunk size exponentially decreases from the value calculated as $number\_of\_interations/number\_of\_threads$ to '1' by default or to value explicitly specified by a programmer at compile time.

There are two loop scheduling types, which combine the general algorithms: *auto* delegates the decision to the compiler and *runtime* uses the $OMP\_schedule$ environment variable.

5.2.3 ARM big.LITTLE technology

The ARM big.LITTLE technology implements two sets of cores: low power energy-efficient cluster that is called 'LITTLE' and power hungry high performance cluster which is called 'big'. Having a large number of suitable architectures, ARM provides multiple combinations. In current work, we focus on the Cortex-A7/Cortex-A15 core cluster pair. As demonstrated in Figure 5.1, chosen combination covers significant performance/power area. Manufacturers promise to achieve ∼50% energy saving together with better performance [123]. Heterogeneity causes tough technical challenges, which concern hardware cache coherency and programming.

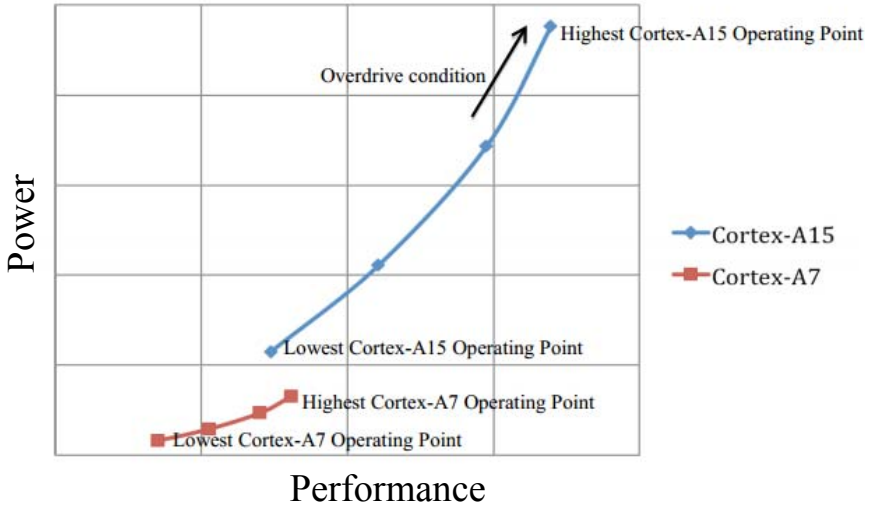

FIGURE 5.1: ARM big.LITTLE technology [123].

5.2.3.1 Hardware support

In general, big.LITTLE architecture contains the following set of components:

- **CPU** includes two clusters each of which has a set of cores and L2 cache memory.
- **Interconnect component** manages hardware cache coherency.
- **Memory and periphery interfaces**.

The interconnect component plays an essential role in architecture performance. To demonstrate how the cache coherence interconnect works we consider two implementations which are shown in Figure 5.2. Figure a) illustrates the timing diagram of cluster

switching in the case of AXI (AMBA3) interconnect. In this case, the cache clean operation that writes back the up-to-date data into DRAM is required. For example, for 2 MB L2 cache this operation takes around 1.4 ms. ARM also released the CCI-400 interconnect, which is shown in Figure b). It allows reducing core switching time to around 30 us [123].

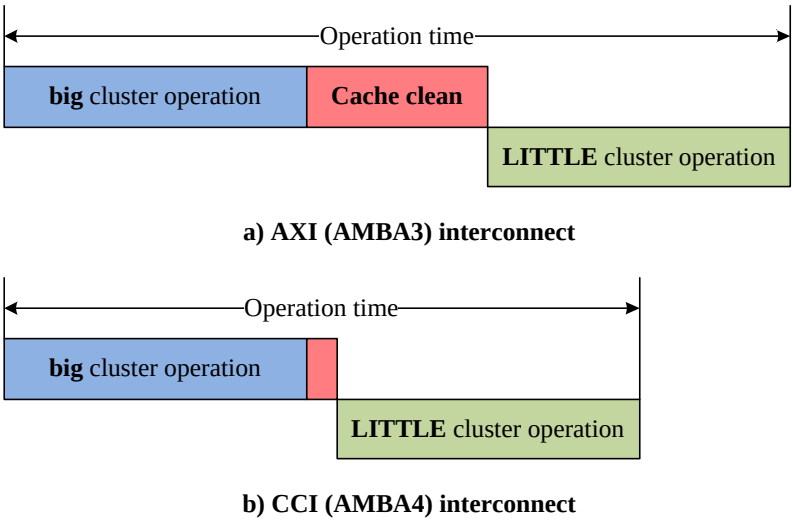

FIGURE 5.2: ARM big.LITTLE Cache Coherent Interconnect.

5.2.3.2 Software support

In particular, ARM big.LITTLE processors have three main software execution models [99] which are illustrated in Figure 5.3.

The first and simplest model is called cluster migration. A single cluster is active at a time, and migration is triggered on a given workload threshold.

The second mode named CPU migration relies on pairing every "big" core with a "LITTLE" core. Each pair of cores acts as a virtual core in which only one actual core among the combined two is powered up and running at a time. Only four physical cores at most are active. The main difference between clustered migration and CPU migration models is that the four actual cores running at a time are identical in the former while they can be different in the latter.

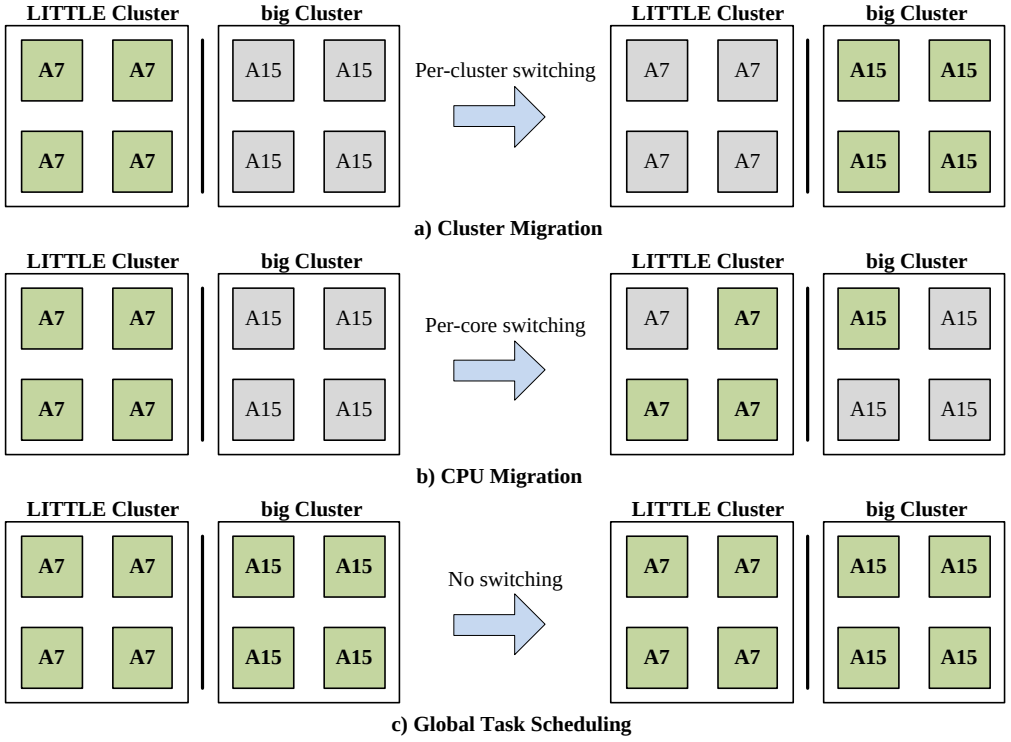

FIGURE 5.3: Software execution models for ARM big.LITTLE architecture.

The heterogeneous multi-processing mode, also known as global task scheduling allows using all of the cores together. A strong argument in favor of HMP is that it provides a fine-grained control of workloads and consequently opens a promising direction for additional performance/energy trade-offs.

We summarize the key features of presented ARM big.LITTLE execution models in Table 5.1. We assume that both, performance and power saving depend on the switching granularity. Thus the most promising and yet little studied HMP execution mode is our target research in this chapter.

TABLE 5.1: ARM big.LITTLE execution model comparison.

	Cluster migration	Core migration	HMP
Switching granularity	Low flexibility	Medium flexibility	High flexibility
Maximum performance	Medium: N 'big' cores	Medium: N 'big' cores	High: N 'big' + M 'LITTLE' cores
Power saving	Low: coarse granularity	Medium	High: fine granularity

5.3 Evaluation of the Exynos 5 Octa SoC

5.3.1 Experimental setup

In this section, we present a detailed analysis of the heterogeneous big.LITTLE architecture. Figure 5.4 pictures the used exploration flow. As shown on the right-hand side of the figure, we first design a gem5 big.LITTLE processor model for performance evaluation and a McPAT model for power consumption estimation. The accuracy in both performance and power estimations was assessed in Section 3.4 by comparing to that measured on the reference Exynos Octa 5422 SoC system, a Odroid XU3 computer board as represented in the left-hand side of the figure. This study is conducted using the Rodinia benchmark suite through its OpenMP implementation [124] executed on both the board and in the gem5 full system environment. Furthermore, a detailed analysis of the Rodinia kernels and applications runtime behavior is conducted using the Scalasca/Score-P instrumentation [125] and Vampir event trace data visualization tool [126]. This analysis provides useful insight allowing us to infer new ARM big.LITTLE inspired architecture configurations that perform better energy-wise.

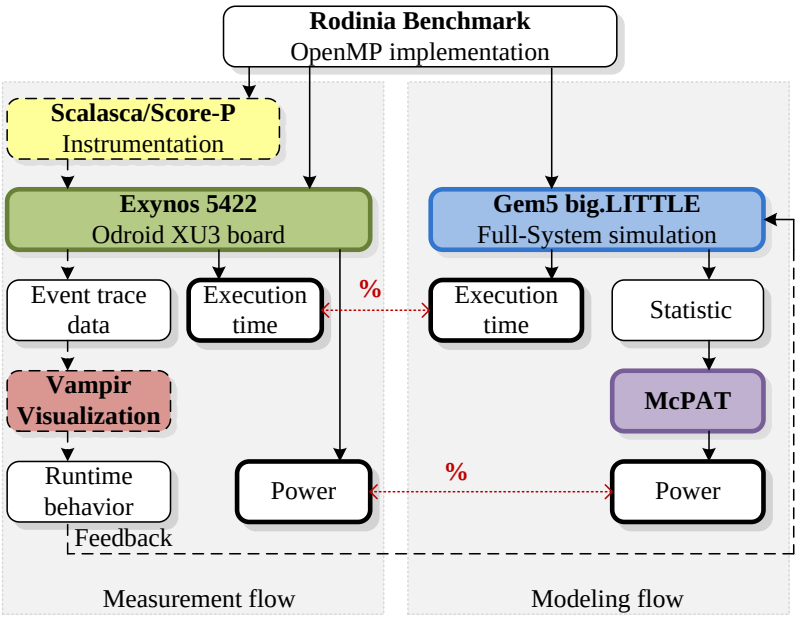

FIGURE 5.4: ARM big.LITTLE exploration flow.

System software. The reference Odroid XU3 board runs the latest Ubuntu 14.04 OS on Linux kernel LTS 3.10, which supports global task scheduling. Note that throughout all experiments we did not use the embedded GPU.

The Rodinia benchmark is used throughout the rest of the chapter. The OpenMP implementation is here chosen, with four threads per cluster, i.e. one thread per core. The following subset of applications is selected: *backprop*, *bfs*, *heartwall*, *hotspot*, *kmeans openmp/serial*, *lud*, *nn*, *nw* and *srad v1/v2*. Detailed description of selected applications and kernels including the problem size have been presented in Table 3.6.

We consider the following scenarios:

- Cortex-A7 cluster running at 200MHz, 800MHz and 1.4GHz;
- Cortex-A15 cluster running at 200MHz, 800MHz and 2GHz;
- HMP Cortex-A7/A15 running at 200/200MHz, 800/800MHz, 1.4/2GHz, 200MHz/2GHz and 1.4GHz/200MHz.

5.3.2 Performance analysis

Reported execution times are normalized against the slowest configuration, i.e. Cortex-A7 running at 200MHz. Observed speedup is shown in Figure 5.5 using logarithmic scale.

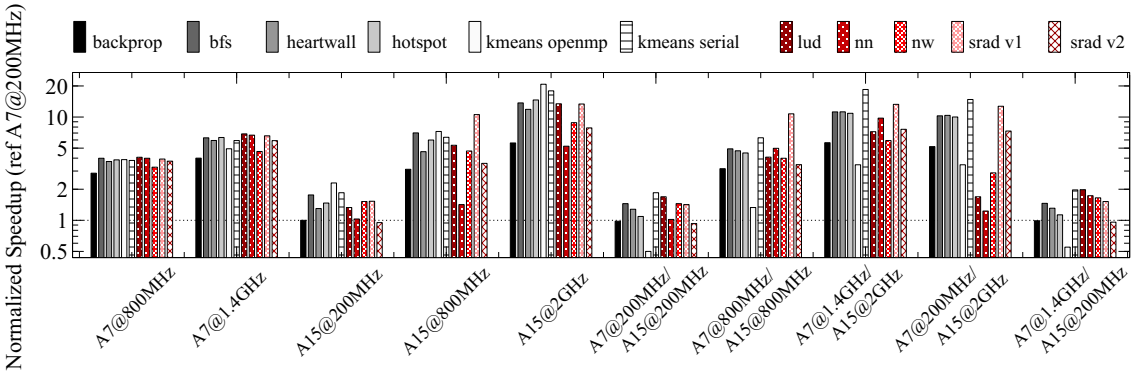

FIGURE 5.5: Normalized measured speedup.

Based on these results the following observations can be made:

- Speedup strongly depends on application nature.
- Best performance is in most cases achieved by Cortex-A15 running at 2GHz. A speedup of 21x is observed for *kmeans openmp* in big cluster.

- HMP mode exhibits similar performance compared to A15 cluster, despite four additional actives cores. For some applications (*kmeans openmp*, *lud*, *nw*) worse performance is obtained. A significant penalty is further observed in this mode when operating the LITTLE cluster at low frequency, notably for *lud*, *nn* and *nw* applications.

- The Cortex-A15 cluster shows speedups ranging from 1.4 to 10, possibly originated from the out-of-order microarchitecture and larger L2 cache size.

Further investigations were carried out with Scalasca [125] and Vampir [126] software tools that permit instrumenting the code and then make it possible to visualize low-level behavior based on collected execution traces. The original Rodinia source code was instrumented and executed on the Odroid XU3 board. We selected three representative configurations: (i) the reference LITTLE cluster running at 200MHz, (ii) the most efficient big cluster running at 2GHz and (iii) the HMP big.LITTLE running at 200MHz/2GHz previously observed as underperforming.

Figure 5.6 depicts the runtime breakdown between master thread, parallel regions and OMP barriers for the selected Rodinia benchmarks. Figure 5.6.a) shows that information for the LITTLE cluster, observed behaviors may be categorized as follows:

1. **Rather serial implementations** in which over 90% of execution time is spent in the master thread: *bfs*, *heartwall*, *hotspot* and *kmeans serial*.

2. **Moderate parallel implementations** in which 20-50% of execution time is spent in the parallel regions: *backprop*, *srad v1* and *srad v2*. An example of *srad v1* runtime behavior is demonstrated in Figure 5.7.a).

3. **Parallel implementations** in which over 50% of time is spent in the parallel regions. This group includes *kmeans openmp*, *lud*, *nn* and *nw*. An example of *nn* runtime behavior is demonstrated in Figure 5.7.b).

Note that the above classification applies to the implementations related to problem sizes listed in Table 3.6: though chosen problem sizes were deemed adequate for this study, different problem sizes would possibly move members across identified categories. The observed runtime behavior is correlated to the published Rodinia analysis [102].

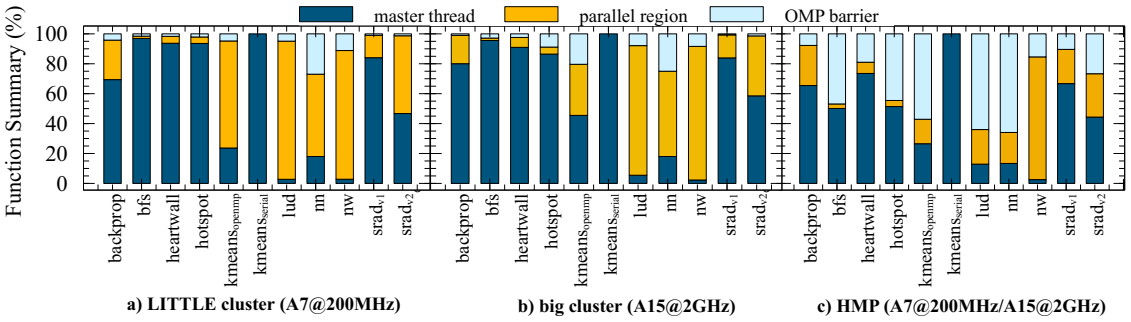

FIGURE 5.6: Runtime breakdown between master thread, parallel regions and OMP barriers for the Rodinia benchmark.

Figure 5.6.b) shows the execution time breakdown for the big cluster running at 2GHz. Similar distributions are logically observed in that configuration.

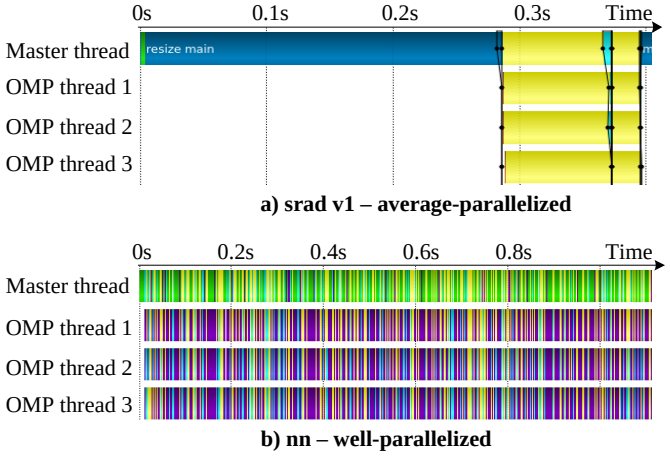

FIGURE 5.7: Runtime behavior: a) *srad v1* executed on Cortex-A7 cluster running at 200MHz, b) *nn* executed on Cortex-A7 cluster running at 200MHz.

When operating in HMP mode with the previously discussed configuration (LITTLE and big clusters running at 200MHz and 2GHz), rather different behavior is observed as depicted in Figure 5.6.c). A substantial increase of the OMP barrier is reported, often accounting for over 50% of the execution time. Figure 5.8 shows a snapshot of the execution trace of the *lud* application alongside a zoom on two consecutive *parallel-for* loop constructs. It is clearly visible the OpenMP runtime spawned eight threads that were assigned to all eight cores. Those four threads assigned to the Cortex-A15 cores completed execution in far less time compared to the other four executed on the Cortex-A7 cores. This results in significant time wasted idling: the slowest cores are on the execution critical path and thereby bridle system performance. These experiments were performed with the default static schedules for OpenMP. Using dynamic schedules by means of

modifying OpenMP pragmas in the application source code led to some improvements varying from application to application. Overall, the OMP barrier time remained in the order of 30%. OpenMP API therefore poses some performance issues when using HMP mode, because of the thread-oriented synchronization mode that requires well-balanced workload across cores.

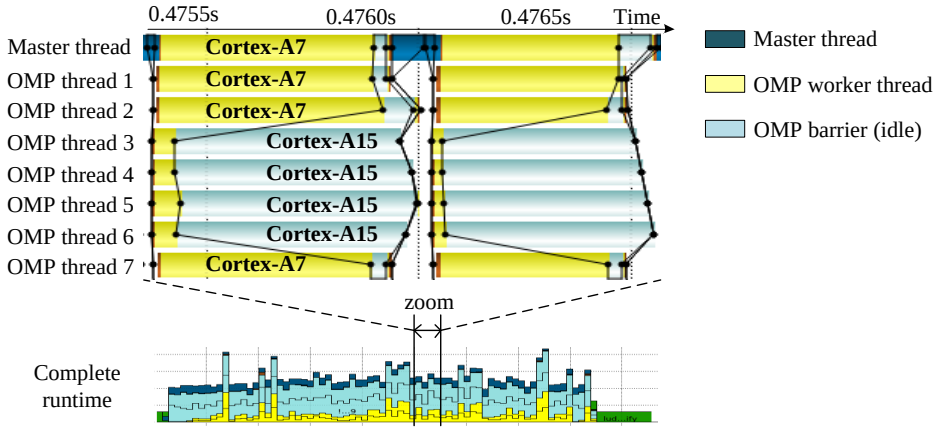

FIGURE 5.8: Runtime behavior: *lud* executed on HMP big.LITTLE Cortex-A7/A15 running at 200MHz/2GHz.

5.3.3 Energy-to-solution analysis

Figure 5.9 shows the normalized EtoS for all configurations, measured on the Odroid XU3 board. Results are again normalized against the reference Cortex-A7 running at 200MHz. The following observations can be made:

1. The Cortex-A7 cluster is overall more energy-efficient than the Cortex-A15. Best energy-efficiency is achieved when operating at 800MHz.

2. For certain applications (*bfs*, *kmeans serial*, *srad v1*) the Cortex-A15 running at 800MHz provides better EtoS than the reference Cortex-A7 cluster. These applications benefit the most from the A15 out-of-order architecture with the largest speedups, thereby resulting in energy savings.

3. In HMP mode, some outliers exhibiting much elevated EtoS are observed: this concerns the configuration Cortex-A7/A15 running at 200MHz/2GHz, when running *lud* and *nn* applications. This finds roots in the behavior identified in previous

section where slower cores are observed to create a critical path in the application execution. This is particularly prominent in that configuration for which the A7 cluster is clocked at the slowest frequency whereas power hungry A15 cluster spends over 60% of execution time at synchronization barriers.

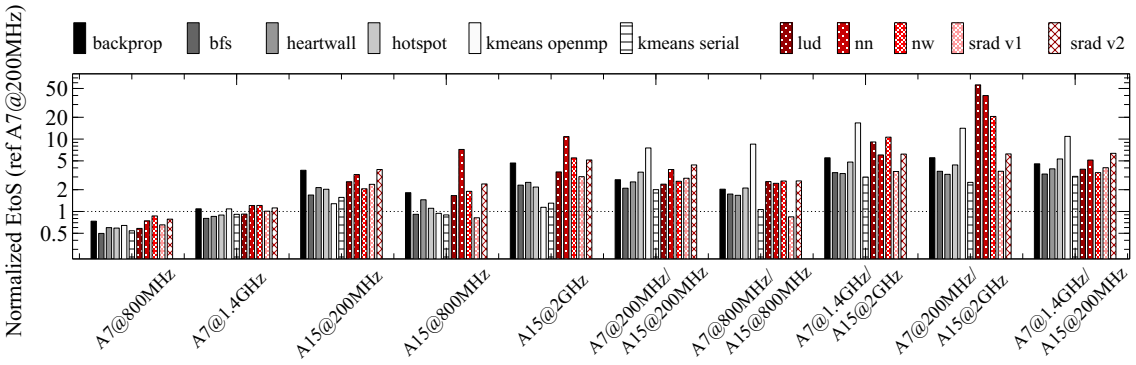

FIGURE 5.9: Normalized measured energy-to-solution.

5.4 Alternative big.LITTLE architectures exploration

5.4.1 Experimental setup

The previous analysis of Rodinia kernels shows a significant penalty in terms of energy-efficiency caused by the presence of critical sequential code section in kernels. Given the fact the Cortex-A7 core proves more energy-efficient for most workloads, it can be beneficial to increase Cortex-A7 core count while maintaining a single Cortex-A15 for efficient handling of serial code section (namely OpenMP master thread). Three kernels are chosen for exploring the gain of such new architecture configuration: *backprop*, *srad v1* and *srad v2*. These three kernels exhibit a balanced distribution between the assumed serial and parallel sections.

Table 5.2 summarizes the five proposed asymmetric configurations (C1-C5) in which we varied the ratio of cores between the two clusters, core clocks as well as the L2 cache size. Execution time and EtoS are shown in Figure 5.10.

TABLE 5.2: big.LITTLE proposed configurations.

	Cortex-A7 cluster		Cortex-A15 cluster		
	Count	Clock	Count	Clock	L2
C1	4	800 MHz	1	800 MHz	2 MB
C2	4	800 MHz	1	2 GHz	2 MB
C3	7	800 MHz	1	800 MHz	2 MB
C4	7	800 MHz	1	2 GHz	2 MB
C5	7	800 MHz	1	800 MHz	512 kB

5.4.2 Exploration results

Reported results show that configurations C3 and C5 provide up to 27% of improvement in energy-efficiency compared to the best existing configuration used as reference (Cortex-A7 cluster running at 800MHz). Furthermore, configuration C5 with smaller L2 cache demonstrates better execution time and EtoS than the similar configuration C3. According to the collected statistics, configuration C5 smaller cache size leads to an average miss latency reduction of 30%. This finds root in rather cache-unfriendly access patterns for these applications as well as the L2 cache architecture specifics: transaction processing time increases with cache size [127] and is not counterbalanced by lower cache miss rate for such workloads.

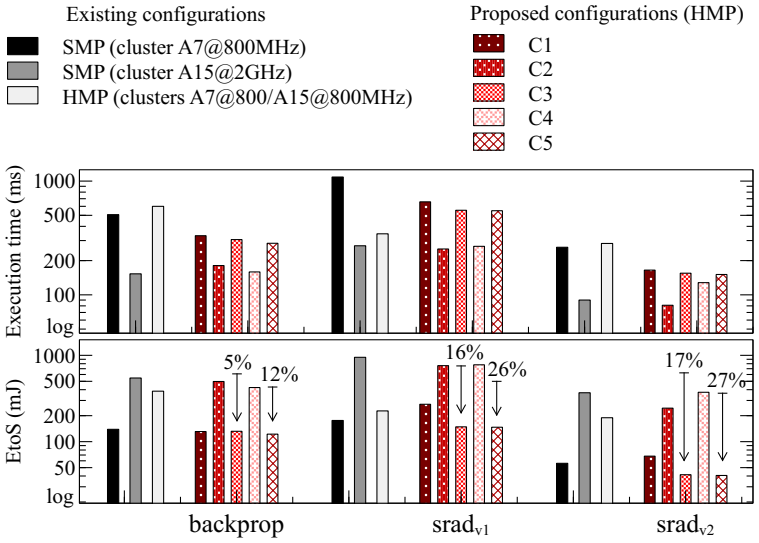

FIGURE 5.10: Execution time and energy-to-solution comparison between existing and proposed configurations.

The exploration of alternative configurations of heterogeneous architecture considered above indicates that best performance and energy-efficiency trade-off is obtained with few big cores and many LITTLE cores. Indeed, the sequential code section (mainly

the "main" thread in OpenMP program) in the considered kernels benefits from the single Cortex-A15 core whereas the parallel data-intensive region (i.e. OpenMP parallel threads) has been executed across the multiple Cortex-A7 cores. Reducing the number of power-hungry big cores that severely degrade processor energy-efficiency and by increasing the number of LITTLE cores, we observe up to 27% of improvement in energy-efficiency compared to the best existing configuration used as reference. Moreover, according to [47] and [128], typical silicon footprint for Cortex-A7 and Cortex-A15 clusters are respectively $3.8mm^2$ and $19mm^2$. This 1:5 ratio further suggests configurations such as 1 Cortex-A15/16 Cortex-A7 cores would likely be implementable on a die area similar to that of existing big.LITTLE SoCs. The improved performance and energy trade-off observed in those new hetererogeous architecture configurations suggests that the equal core count found in the two clusters of the considered Exynos 5422 chip is not necessarily the most efficient configuration for OpenMP scientific workloads. Such configurations are often suggested in the litterature as in [**?**], in which authors promote a sequential accelerator associated with several simpler cores (for parallel code regions) as a very attractive design solution. Beyond the assessment of energy efficiency gains, this study further demonstrate that software-friendly single-ISA heterogeneous systems achieve signifiant gains, despite their shared-memory architecture.

5.5 big.LITTLE architecture scaling via trace-driven simulation

The presented big.LITTLE gem5 model allows us exploring such important parameters as cache size, interconnect width, memory infrastructure. However, due to current limitation of gem5 to simulate more than eight ARM cores, exploring large-scale ARM-based system models is not feasible. Thus to evaluate the scalability of the Rodinia benchmark running on the big.LITTLE heterogeneous manycore we used the proposed trace-driven approach presented in Chapter 4. To demonstrate the exploration flow we chose *hotspot* application described in Table 3.6 with the problem size equal to 1024.

5.5.1 Experimental setup

To collect the Cortex-A7 traces we use the *TimingSimpleCPU*. As we showed in Chapter 3 Section 3.4 this model is suitable for performance evaluation and in addition it provides well-organized traces where each request is always followed by a response. The Cortex-A15 trace-driven simulation is a tedious task. The out-of-order nature at times complicates trace injections and requires extra micro-dependency analysis. More in detail this challenge is discussed in Chapter 4 Section 4.2.2. To emulate the Cortex-A15 processor behavior we use the computation phase scaling technique described in Section 4.3.1.3.

In Figure 5.11 we illustrate the *hotspot* kernel runtime behavior captured on the Odroid XU3 board with Scalasca/Score-p instrumentation [125] and analyzed with Vampir tool [126]. The figure represents execution of four threads under two Cortex-A7 and two Cortex-A15 cores running at the same frequency. The Cortex-A15 duration is less than the Cortex-A7 corresponding to 0.16s and 0.23s respectively. Based on these values we calculate an acceleration factor as 1.45x and applied it to the trace-driven simulation of the Cortex-A15 cluster.

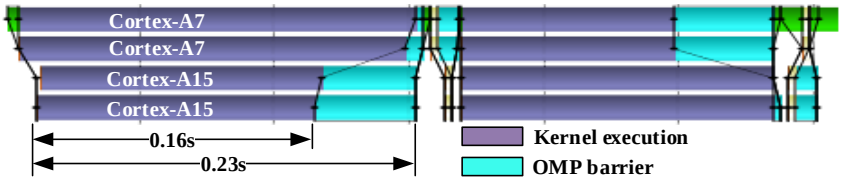

FIGURE 5.11: Hotspot parallel region runtime behavior running on the Odroid XU3 board.

Trace replication technique (see Chapter 4) relies on overlapping trace patterns with the increasing number of TIs. To obtain the replication pattern we capture the parallel region traces presented in Figure 5.12. The initialization phase of the application is not considered in these experiments. We illustrate the trace pattern collected at the core#0 (Figure 5.12 a)) and at the core#1 (Figure 5.12 b)) on the system with four cores and four threads. Each kernel iteration is composed on two `pragma omp parallel for`: (i) compute temperature and (ii) store results. We observe the results storage region provide a high number of cache misses comparing to the compute temperature region.

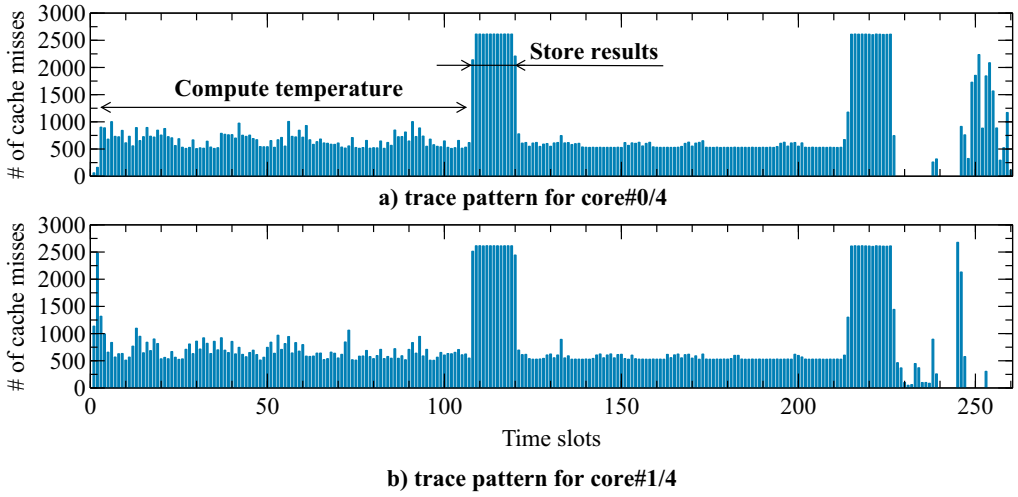

FIGURE 5.12: Hotspot parallel region trace pattern.

5.5.2 Exploration results

We evaluated three scenarios:

- LITTLE cluster with 4, 8, 16, 32, 64 and 128 cores (injectors),
- big cluster with 4, 8, 16, 32, 64 and 128 cores (injectors),
- big.LITTLE in HMP mode with 4/4, 8/8, 16/16, 32/32 and 64/64 cores (injectors).

The execution time and speedup for each scenario are presented in Figure 5.13. The best execution time, as well as the speedup shows big cluster. The LITTLE cluster provides the worst execution time. The big.LITTLE speedup is normalized by the faster big cluster. We observe that the execution time in HMP mode is worse than in the big cluster and slightly better than in the LITTLE cluster. It explained by the OpenMP programming nature that we observed in Figure 5.11: the slower Cortex-A7 cores slow down the execution. For all three scenarios, the speedup reaches the plateau around 64 cores (injectors). It explained by the memory/interconnect saturation. The figure a) also contains three values measured on the board. The comparison shows the high level of simulation accuracy: the error percentage is around 15%.

To address this common issue we propose to explore the big.LITTLE architecture with alternative network-based *Ruby* memory subsystem [105]. System includes two-level cache hierarchy. The consistency of the memory is maintained by the MESI coherence

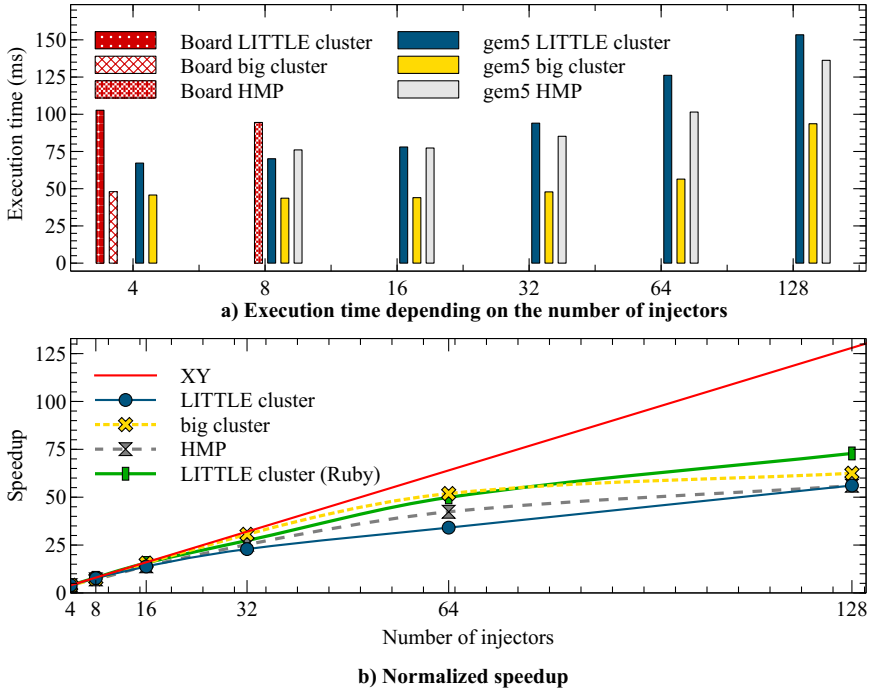

FIGURE 5.13: Execution time and speedup evaluation using trace-driven simulation.

protocol. This protocol models inclusion between the L1 and L2 caches and has four stable states, M, E, S and I, hence the name. The interconnection network has the following features: Mesh topology, XY routing algorithm and detailed GARNET network microarchitecture model (16-byte links, 10 virtual networks, 4 virtual channels per virtual network, 4 buffers per virtual channel, 1 cycle on-chip link latency). The block diagram of the proposed architecture and detailed description of their parameters are presented in Figure 5.14 and in Table 5.3.

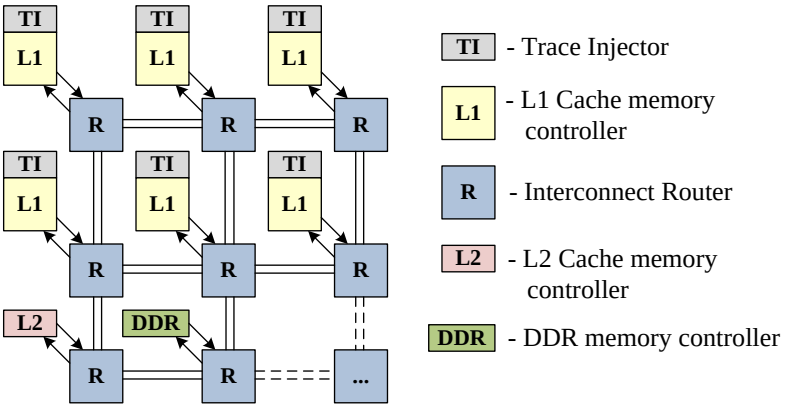

FIGURE 5.14: Alternative big.LITTLE-based network-on-chip manycore architecture.

TABLE 5.3: Architecture Configuration

Collection	
Number of CPUs	4
CPU ISA	ARMv7-a
CPU frequency	2 GHz
L1 D/I Caches	32 kB, 4-way associativity, 32 B/line
Interconnect	Bus
Benchmark/Programming Model	Rodinia/OpenMP
Replication	
Number of traces	4
Number of injectors	8, 16, 32, 64, 128, 256
Address distribution	Random
Processing	
L2 Cache	512 kB
Cache Coherent Protocol	MESI
Interconnect	Mesh, XY routing, GARNET, 16-byte links, 10 virtual networks, 4 virtual channels per virtual network, 4 buffers per virtual channel, 1 cycle on-chip link latency
Memory	double channel, DDR3

Figure 5.13 b) shows the achieved speedup for the LITTLE cluster (Ruby) up to 128 cores. Application shows a plateau, which originates from saturation of the external memory bandwidth that according to the gem5 statistic file is about 200 million DDR accesses per second. *Hotspot* parallel region investigation shows that system scalability can be improved by efficient network interconnect on around 30% of execution time speedup.

5.6 Single-ISA heterogeneous multicore granularity evaluation

The evaluated ARM big.LITTLE architecture features two types of clusters, e.g. Cortex-A7 and Cortex-A15. There are also chips which contain Cortex-A17/Cortex-A7 pair (MediaTek MT6595 series [129]) and Cortex-A57/Cortex-A53 pair (Qualcomm Snapdragon 808 and 810 [130]). The important strength of big/LITTLE clusters combining is high granularity provided by HMP mode as shown in Table 5.1. However, the level of

granularity can be further raised. To the best of our knowledge, this aspect of single-ISA heterogeneous architectures has not been investigated.

We propose to explore the heterogeneity of ARM single-ISA architecture. In Figure 5.15 a set of most commonly used processors of ARM Cortex-A series are compared. The comparison is averaged and based on the published information [123] [46] [47]. The performance and power ratios are normalized by Cortex-A9.

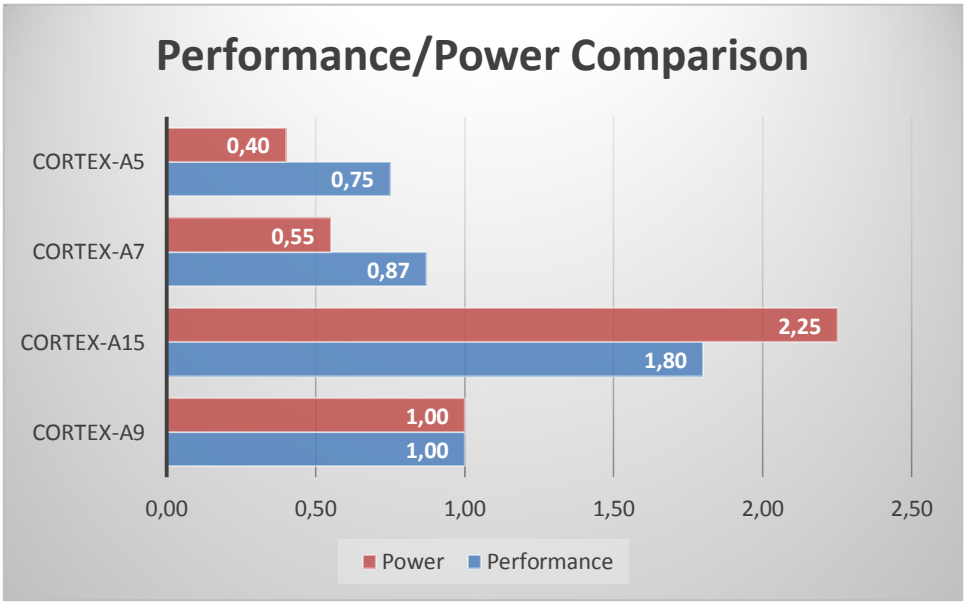

FIGURE 5.15: ARM Cortex-A series performance/power ratios.

Obviously, the ratios may vary for different operations (e.g. integer or floating point computation), frequencies and technologies. Actual comparison includes Cortex-A5, Cortex-A7 and Cortex-A15 at 28 nm technology and Cortex-A9 at 40 nm technology. All processors run at the same 1 GHz frequency.

Our goal is to analyze the possible combination of the presented set of cores. As we demonstrated in Section 5.4, changing the ratio between different types of cluster cores may bring significant energy improvements.

In order to predict approximate performance/power results we implement an analytical model in MATLAB tool [131]. The model is based on the illustrated in Figure 5.15 performance and power ratios.

The model executes an abstract application in parallel. Application contains a set of tasks, which are dynamically distributed among the cores. The algorithm of distribution reproduces the one implemented in OpenMP *dynamic* scheduler. That is each core grabs a task from the common queue one by one. The execution time and power consumption of each task is calculated according to the performance and power ratios.

The model functioning is illustrated in Figure 5.16. The input of the model contains a set of cores, S_{cores}, which represents architecture configuration and a set of tasks, S_{tasks}, as an application to execute. The output results of the model are total energy, E_{total}, and application execution delay D_{time}.

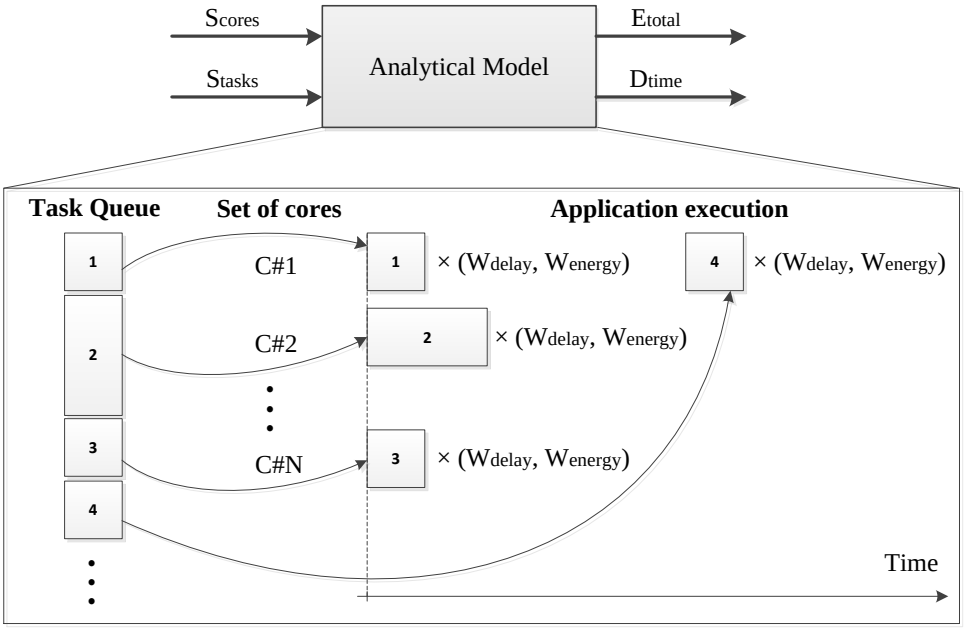

FIGURE 5.16: Analytical model functioning.

Set of cores is an array of $Core(W_{delay}, W_{energy})$ where W_{delay} is a weight which corresponds to task execution delay and W_{energy} is a weight which corresponds to task energy spent. Set of tasks is an array of tasks duration.

Inside the model, S_{tasks} is transformed into a task queue. The model captures tasks one by one and assigns them to cores according to the implemented algorithm. Cores delay and energy weights are then applied to those tasks. The proposed model calculates the potential load imbalance, but does not consider inter-core dependencies and communications.

An example of 10-tasks application modeling on 3-cores system is illustrated in Figure 5.17. Example demonstrate perfectly parallelized application execution. As Cortex-A7 core is the slowest one, it executes only two tasks. Cortex-A9 manages to perform three tasks. The most powerful Cortex-A15 performs five tasks.

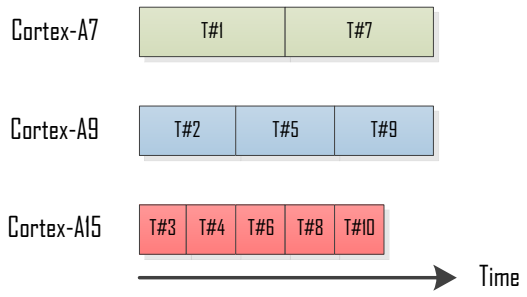

FIGURE 5.17: Example of abstract application execution with 10 tasks distribution.

Using the proposed model we explore a set of alternative configurations. Evaluated system always contains eight cores, only the types of cores and their number change. We consider five scenarios:

- Existing big.LITTLE configuration with Cortex-A7 and Cortex-A15 clusters. The number of cores in both clusters varies from 8A7/0A15 to 0A7/8A15.
- Alternative two-cluster configuration with Cortex-A9 and Cortex-A15 pair. The number of cores in both clusters varies from 8A9/0A15 to 0A9/8A15.
- Another alternative two-cluster configuration with Cortex-A5 and Cortex-A15 pair. The number of cores in both clusters varies from 8A5/0A15 to 0A5/8A15.
- Three-cluster configuration that we called big.M.LITTLE (big.Medium.LITTLE) contains Cortex-A7, Cortex-A9 and Cortex-A15. We believe that yet another medium cluster which further raises the level of granularity may bring essential improvements. We iterate through all core number combinations from 6A7/1A9/1A15 to 1A7/1A9/6A15.
- Four-cluster configuration with Cortex-A5, Cortex-A7, Cortex-A9 and Cortex-A15 cores.

Figure 5.18 presents the results comparison. For this experiments we chose 100 equivalent tasks application. We highlighted several interesting points. The point *4A7/4A15*

demonstrates where the actual ARM big.LITTLE architecture is. The point *7A7/1A15* shows the proposed in Section 5.4 alternative configuration. According to the analytical model results, *7A7/1A15* configuration provides 17% energy improvement. The performance degradation is around 8%. We are also interested by the point *6A7/1A9/1A15* as it shows both energy and performance improvements.

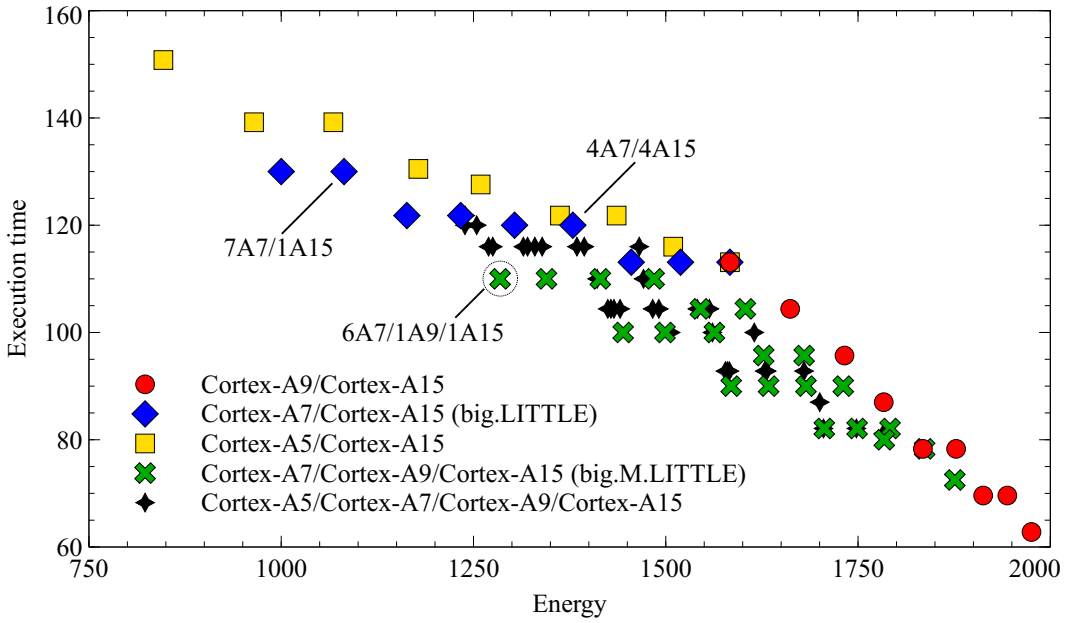

FIGURE 5.18: Heterogeneous architectures Energy/Delay comparison for equivalent tasks application.

In the next experiment we demonstrate the 100-tasks application with non-equivalent duration but with random distribution. Such application introduces the class of non-regular benchmarks. The results are shown in Figure 5.19. We observe the similar behavior. However, several points changed their positions relative to each other. For example, the points *4A7/4A15* and *6A7/1A9/1A15* now are closer. Thus the expected performance and energy enhancements became 6% and 1% respectively. We assume that the gain for different applications significantly varies.

The accuracy of such analytical model estimation does not allow drawing final conclusions. Nonetheless, it demonstrates the high interest for further explorations. Especially, with three- and more-cluster heterogeneous multicores.

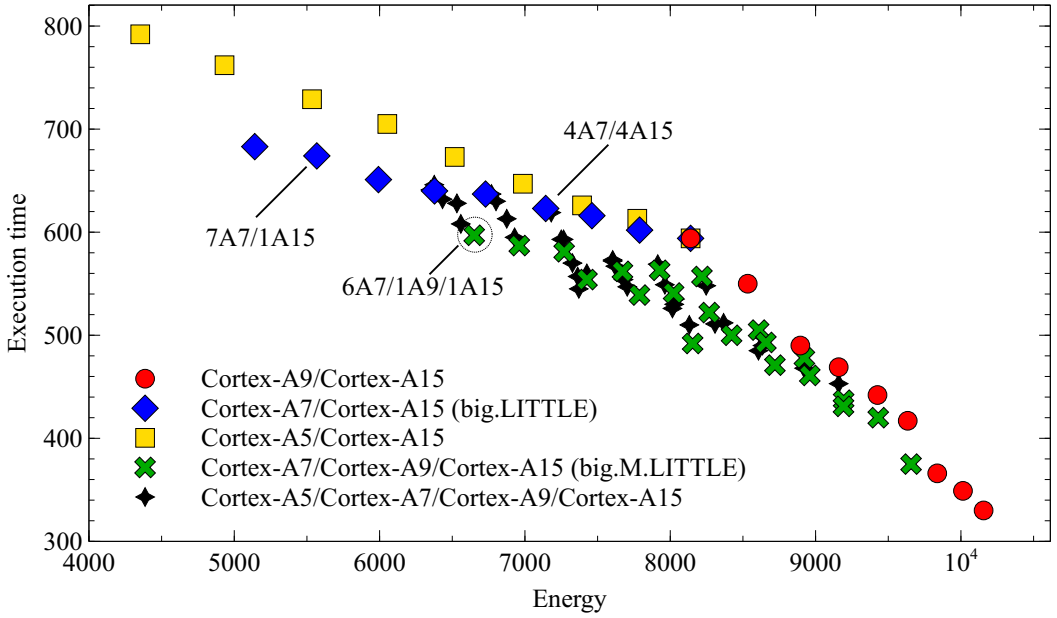

FIGURE 5.19: Heterogeneous architectures Energy/Delay comparison for random tasks application.

5.7 Discussion

In this chapter, we explored the design of single-ISA heterogeneous multiprocessor architecture. The investigations were conducted on the ARM big.LITTLE technology.

Beyond the actual configuration of the chosen SoC, further configurations were evaluated by using its performance and power models implemented in gem5 and McPAT simulation frameworks. We proposed alternative heterogeneous configurations with one high-performance core dedicated to sequential code sections and multiple low-power cores for parallel code section execution. The results demonstrated up to 27% of energy-efficiency improvement compared to the best existing configuration used as reference.

Due to the limitation of the gem5 full system mode, we applied the proposed trace-driven approach for explore the heterogeneous nature of big.LITTLE systems including more than one hundred of cores. The scalability of such systems was addressed and compared to that of homogeneous systems. *Hotspot* parallel region investigation showed that system scalability can be improved by efficient network interconnect on around 30% of execution time speedup.

To investigate the ARM big.LITTLE heterogeneity and granularity, an analytical model was implemented in MATLAB tool. The model takes architecture configurations, i,e. number of clusters and cores, as input and generates the execution time and the total energy as output. Evaluating 140 architecture configurations, the proposed analytical model demonstrated high interest for exploration of three- and more clusters heterogeneous multicore architectures.

Chapter 6

Conclusions

The next step in high-performance computing evolution refers to computing systems performing at least one exaFLOPS, i.e. a quintillion floating point operations per second. According to projections, such a system is expected by 2018 with a 20MW power budget [3]. A number of projects, which target to deliver exascale computing systems, have been announced. This thesis was conducted within the European Mont-Blanc project [17]. The aim of the project is to design a new energy-efficient exascale computing system using low-power embedded technologies.

The research was set out to explore multi- and manycore architectures for future scalable and energy-efficient supercomputers. For efficient design space exploration a fast and accurate simulation environment is mandatory. This thesis therefore was looking to answer several groups of questions:

> **Q:** How accurate are performance and power models implemented in cycle-approximate full system simulation frameworks? What are the main sources of error in these models? Can these models be used to realistically predict important exploration metrics?
>
> **Q:** How can simulation time can be reduced while preserving the accuracy level? What are the limitations of the existing approaches and how can they be avoided? Is the proposed approach efficient enough to enable future manycore architecture exploration?

Q: What are the promising directions in computer architecture design? How can the existing programming models be used to benefit from heterogeneous multi- and manycore architectures? Which configurations of single-ISA heterogeneous architectures can significantly enhance the system energy-efficiency?

This chapter is organized as follow: Section 6.1 discusses the scientific contributions of this thesis. Section 6.2 proposes several future research directions. In Section 6.3 the list of publications is presented.

6.1 Contributions

Contribution 1 The first objective of this thesis was to evaluate the accuracy of the performance and power models implemented in cycle-approximate simulation frameworks. Two multicore processors were chosen for model calibration and validation: (i) ARM dual-core STE Nova A9500 processor and (ii) ARM big.LITTLE Exynos 5422 processor. gem5 event-driven simulator together with McPAT framework were combined into a suitable simulation environment for model implementation and evaluation. To obtain a high model confidence, a large set of parameters including architecture configurations and representative benchmark suites were considered.

The results of the comparison between the proposed models and real hardware execution demonstrated that the absolute error varies significantly depending on the configuration, ranging from 1% to 57%. The proposed performance and power models have been made freely available online for research community [22].

Based on the detailed analysis of the sources of error we conclude:

- Detailed implementation of core microarchitecture is not mandatory if the aim is to explore the performance of the entire system. Memory infrastructure, especially the main memory component, can provide a harmful discrepancy in the case of inaccurate or unrealistic implementation. Thus if a researcher is looking for a way to simplify the performance model, we suggest to use a less detailed core model but keep a detailed model of the memory infrastructure.

- If the target is power or area exploration, each system component must be as detailed as possible. Indeed, in that case, the mismatch between a simplified and a detailed CPU power models is significant. Moreover, the execution time error, which is already present in performance model simulation directly affects the power model accuracy. Therefore, we suggest using of both models being implemented in detail for power consumption exploration.

Contribution 2 The second objective of this thesis was to study the existing approaches for simulation acceleration. The main challenge was to significantly reduce the simulation speed in order to enable manycore architecture exploration and preserve a suitable accuracy level. The proposed hybrid trace-oriented approach was implemented in the event-driven simulation framework. It included a fully simulated memory system, the trace synchronization mechanism for dependency management, the trace replication technique allowing manycore architecture simulation and the technique of computation phase scaling. The comparison of the proposed approach with the reference full system simulation demonstrated a performance error around 14% in the worst case scenario and a speedup greatly varied between tens and hundreds times depending on the computation-communication nature of the applications. The implementation of the proposed approach has been made freely available online for research community [22].

The proposed approach advanced the state-of-the-art by achieving a suitable trade-off between simulation speed and accuracy level, as well as enabling trace replication and synchronization (see Table 2.2). However it demonstrated a number of vulnerabilities:

- The chosen cache miss filtering technique is very sensitive to cache coherency protocol. The cache outbound traffic may contain not only missed memory accesses, but also writebacks and snooping packets. Without taking into account these communications, the simulator risks to miss a significant share of interconnect traffic. Thus making the simulation results unreliable. This issue particularly affects manycore architecture simulations where the interconnect traffic is much more important.

- The trace replication technique has two key points: it is not applicable to data-dependent applications and it requires an address mapping algorithm to manage private and shared data accesses.

- The technique of computation phase scaling allows flexible switching between multiple core models. Nonetheless, a detailed analysis of application computations is mandatory to accurately reproduce the behavior of target processor.

Contribution 3 The third objective of this thesis was to explore performance and energy-efficiency of future multi- and manycore architectures. Single-ISA heterogeneous multicore architecture was chosen as a promising direction to achieve a suitable balance between performance and energy-efficiency. The proposed performance and power models, as well as the proposed trace-driven simulation were used for architecture exploration. Among other, we considered the efficiency of the OpenMP task scheduling running on ARM big.LITTLE heterogeneous architecture. We also explored the multicore architecture heterogeneity by implementing an analytical model and analyzing potential load imbalance.

Based on the architecture exploration results we observed:

- On ARM big.LITTLE-like heterogeneous multicore processors the task scheduling is a key factor for system performance and energy-efficiency. The traditional equal distribution of tasks among cores is inefficient. This is what OpenMP static loop scheduling demonstrated. The cursory analysis of dynamic and guided scheduling did not demonstrate significant improvements as well. We assume that more research and experiments are required to achieve satisfactory results.

- The exploration of alternative configurations of heterogeneous architecture indicated that best performance and energy-efficiency trade-off is obtained with few high-performance cores and many low-power cores. Indeed, the sequential code section (mainly the "main" thread in OpenMP program) benefits from the single big core whereas the parallel data-intensive region (i.e. OpenMP parallel threads) has been executed across the multiple LITTLE cores. Alternative configurations with one power-hungry big cores and multiple low-power LITTLE cores showed up to 27% of improvement in energy-efficiency compared to the best existing configuration used as reference. This approach is also beneficial in terms of area due to the significant ratio between the silicon footprints of big and LITTLE cores.

- The exploration of multicore heterogeneity with three and more core types demonstrated the high prospects of this research direction. Raising the level of system granularity provides more flexibility for task scheduling.

6.2 Future work

According to the work presented in this thesis, the following research directions are proposed as future work:

- **Trace-oriented approach for data-dependent applications.** Many algorithms from various scientific domains employ input data-dependent behavior [132], e.g. data mining, optimization theory, meshing, etc. This fact points to high importance of resolving the appropriate limitation in the trace-oriented simulation. We believe that the problem can be solved by using the annotations in the source code similarly to the synchronization traces. Identifying the points of branching, a decision to replay different trace segments can be taken. To predict the correct application behavior we propose to use a methodology for application arterial structure detection [133].

- **Alternative memory organization exploration.** The exploration results demonstrated the performance scalability limitation of the considered shared memory manycore architectures (see Section 5.5.2). The coherency protocol underlying hardware cache functionality generates additional interconnect traffic to ensure the single-writer, multiple-reader invariant [134]. Using current technologies, cache coherence cannot scale to the large number of cores. One of the solution is to use a software cache, i.e. a scratchpad memory explicitly managed by a programmer. While the main advantage of the hardware cache that it is transparent to the application software, the software cache requires great efforts from a programmer to efficiently manage data coherency. This task is often delegated to the compiler and the runtime library [135] [136] [137] [138]. In this context, we propose to explore the manycore architecture that employ scratchpad memory organization by using the proposed trace-oriented approach. It will allow us to study different data placement algorithms avoiding complex and time-consuming implementation of appropriate software mechanisms.

The explicitly memory management also opens up the possibility of using alternative technologies, such as non-volatile memories. Recent studies have demonstrated the low leakage power consumption and high density of NVM that allow achieving a significant energy saving [139] [140] [141]. A hybrid on-chip SPM combining both NVM and SRAM [142] [143] [144] became a promising research direction. Using the proposed trace-oriented approach we expect to explore smart data placement algorithms allow moving the most-written data into SRAM and the most-read data into NVM.

- **Exploration of single-ISA heterogeneous manycore architectures.** In Section 5 we determined several promising directions for future research. The analytical model results demonstrated the high interest in evaluating heterogeneous multicore architectures with more than two core types. To confirm the potential gain we plan to explore the performance and power models of these architectures in cycle-approximate simulation frameworks. It will also allow us to analyze how different applications behave on heterogeneous multicore architectures with high level of granularity.

  Efficient task scheduling algorithm is crucial for heterogeneous multicore architecture. The preliminary analysis of OpenMP loop scheduling algorithms showed the necessity of further explorations. We also expect to investigate different task-oriented programming models, such as OmpsS [145] whose semantics enable smarter dynamic scheduling.

6.3 Publications

The list of publications includes:

Journals

1. Luciano Ost, Rafael Garibotti, Gilles Sassatelli, Gabriel Marchesan Almeida, Remi Busseuil, **Anastasiia Butko**, Michel Robert, and Jurgen Becker. Novel Techniques for Smart Adaptive Multiprocessor SoCs. In IEEE Transactions on Computers, March 2013.

International Conferences

1. **Anastasiia Butko**, Abdoulaye Gamatié, Gilles Sassatelli, Lionel Torres and Michel Robert. Design Exploration For Next Generation High-Performance Manycore On-chip Systems: Application To big.LITTLE Architectures. In 2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Montpellier, France, July 2015.

2. **Anastasiia Butko**, Rafael Garibotti, Luciano Ost, Vianney Lapotre, Abdoulaye Gamatié, Gilles Sassatelli, and Chris Adeniyi-Jones. A trace-driven approach for fast and accurate simulation of manycore architectures. In 2015 20th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, pages 707–712, January 2015.

3. Sophiane Senni, Lionel Torres, Gilles Sassatelli, **Anastasiia Butko** and Bruno Mussard. Exploration of Magnetic RAM Based Memory Hierarchy for Multi-core Architecture. In 2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, United States, July 2014.

4. Sophiane Senni, Lionel Torres, Gilles Sassatelli, **Anastasiia Butko** and Bruno Mussard. Power efficient Thermally Assisted Switching Magnetic memory based memory systems. In 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), Montpellier, France, May 2014.

5. **Anastasiia Butko**, Rafael Garibotti, Luciano Ost and Gilles Sassatelli. Accuracy Evaluation of GEM5 Simulator System. In 2012 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), York, UK, July 2012.

National Workshops

1. **Anastasiia Butko**, Rafael Garibotti, Luciano Ost and Gilles Sassatelli. Accuracy Evaluation of GEM5 Simulator System. In Colloque National 2012 du GDR SOC-SIP (GDR SOC-SIP'12), Paris, France, June 2012.

Under submission

1. **Anastasiia Butko**, Abdoulaye Gamatié, Gilles Sassatelli, Lionel Torres and Michel Robert. Exploration of Performance and Energy Trade-offs for Heterogeneous Multicore Architectures. [Under submission].

2. Rafael Garibotti, **Anastasiia Butko**, Luciano Ost, Abdoulaye Gamatié, Gilles Sassatelli, and Chris Adeniyi-Jones. Efficient Embedded Software Migration towards Clusterized Distributed-Memory Architectures. In IEEE Transactions on Computers, 2015. [Accepted].

Appendix A

Performance accuracy evaluation results

Table A.1: Execution time comparison (gem5 versus Exynos Octa 5422).

	backprop	bfs	heartwall	hotspot	kmeans openmp	kmeans serial	lud	nn	nw	srad v1	srad v2
Cortex-A7 @ 200 MHz											
B	1895	384	415	190	35	30	2708	668	970	3689	981
S	1708	531	523	245	34	29	3784	788	1139	4531	880
%	9.9	38.3	26	28.6	2.2	2.5	39.7	17.9	17.4	22.8	10.3
Cortex-A7 @ 800 MHz											
B	662	96	112	49	8.9	7.8	666	168	297	940	262
S	507	140	138	55	8.7	7.9	952	192	326	1086	196
%	23.3	45.6	23.5	12.3	2.4	2	42.9	14.6	9.7	15.5	25.1
Cortex-A7 @ 1.4 GHz											
B	474	61	70	30	7	5	392	100	209	561	166
S	334	80	84	31	6	5	544	112	213	630	117
%	29.6	32.3	19.4	5.8	4.9	1	38.7	11.6	1.8	12.1	29.5
Cortex-A15 @ 200 MHz											
B	1889	218	320	129	15	16	1658	651	637	2412	1028
S	1188	265	371	145	16	14	2315	669	603	1977	754
%	37.1	21.8	16.2	11.9	5.9	11.4	39.6	2.7	5.4	18	25.6
Cortex-A15 @ 1.1 GHz											
B	456	42	58	23	2.7	2.8	292	165	147	455	201

Continued on next page

Table A.1 – *Continued from previous page*

	backprop	bfs	heartwall	hotspot	kmeans openmp	kmeans serial	lud	nn	nw	srad v1	srad v2
S	243	50	71	26	3.3	2.6	420	192	163	419	147
%	46.7	16.7	22.2	13.3	21.2	5.8	43.6	16.3	11.1	8	26.9
Cortex-A15 @ 2 GHz											
B	338	28	35	13	1.66	1.65	161	128	110	276	125
S	153	34	48	16	1.67	1.47	231	149	111	270	90
%	57.8	17.9	37.5	22.7	0.9	10.8	43.2	16.4	0.9	2.2	27.8
Cortex-A7 @ 200 MHz / Cortex-A15 @ 200 MHz											
B	1938	265	325	175	69	16	1559	1298	667	2594	1053
S	1182	279	309	106	50	20	1342	1316	411	2327	606
%	39	5.4	4.9	39.6	27.7	28.2	13.9	1.4	38.5	10.3	42.5
Cortex-A7 @ 1.4 GHz / Cortex-A15 @ 2 GHz											
B	336	34.2	37	17.5	10	1.6	201	137	164	278	129
S	145	34.4	48	17.8	7	2.1	149	172	120	303	93
%	56.9	0.8	30.9	1.7	31.9	26.8	25.8	25.6	27	9	27.7
Cortex-A7 @ 200 MHz / Cortex-A15 @ 2 GHz											
B	367	37	40	19	10.5	1.6	628	1084	337	290	134
S	228	38	33	15	10.7	2.1	824	1425	392	375	135
%	37.8	2.1	16.7	17.5	2	28.7	31.3	31.5	16.5	29.5	0.9
Cortex-A7 @ 1.4 GHz / Cortex-A15 @ 200 MHz											
B	1910	262	318	169	63	15	1517	771	587	2425	1019
S	987	290	339	160	62	21	938	1013	350	1720	620
%	48.3	10.9	6.8	5.2	2.4	34.5	38.1	31.4	40.4	29.1	39.1

Appendix B

Power accuracy evaluation results

Table B.1: Power consumption comparison (gem5/McPAT versus Exynos Octa 5422).

	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem
	backprop			bfs			heartwall			hotspot			kmeans		
Cortex-A7 @ 200 MHz															
B	35	-	65	36	-	53	33	-	83	37	-	41	41	-	48
S	37	-	72	42	-	68	36	-	58	38	-	50	37	-	66
%	5.1	-	11	16.7	-	27.5	8	-	10.2	2.8	-	22.7	11.2	-	38.6
Cortex-A7 @ 1.4 GHz															
B	341	-	93	409	-	44	367	-	70	400	-	44	425	-	63
S	411	-	93	494	-	72	412	-	74	423	-	58	394	-	70
%	20.6	-	0	20.8	-	62.7	12.2	-	5.6	5.6	-	29.6	7.3	-	11.5
Cortex-A15 @ 200 MHz															
B	-	227	145	-	224	41	-	191	49	-	195	35	-	223	39
S	-	231	181	-	233	60	-	236	56	-	233	49	-	225	56
%	-	1.8	20.2	-	4	30.4	-	19	12.6	-	16	28.2	-	0.9	30.8
Cortex-A15 @ 2 GHz															
B	-	2537	81	-	2737	41	-	2495	77	-	2397	37	-	2065	47
S	-	2698	90	-	3256	76	-	2844	81	-	3559	54	-	2374	61
%	-	6.3	10.6	-	15.9	46.2	-	12.3	4.7	-	32.7	30.2	-	13	22.6
Cortex-A7 @ 200 MHz / Cortex-A15 @ 200 MHz															
B	19	184	64	18	211	42	18	213	52	19	233	43	26	264	45
S	21	188	83	20	243	73	19	214	65	22	223	59	19	202	67
%	10.4	2	28.8	15	15.3	74.7	7.3	0.2	25.7	16	4.2	37.7	25.4	23.5	50.3

Continued on next page

Table B.1 – *Continued from previous page*

	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem	A7	A15	Mem
Cortex-A7 @ 1.4 GHz / Cortex-A15 @ 2 GHz															
B	198	2822	79	210	210	46	209	3188	77	214	2995	45	224	3792	49
S	240	2680	84	303	3489	77	294	2542	65	230	3551	64	232	3254	61
%	20.8	5	5.8	44.6	9.4	67.1	40.7	15.1	16.7	7.8	6.4	42.6	3.7	31.1	24.4
	lud			nn			nw			srad			Average		
Cortex-A7 @ 200 MHz															
B	45	-	41	62	-	43	48	-	52	39	-	34			
S	37	-	61	44	-	56	38	-	60	38	-	56			
%	16.8	-	47.2	29.6	-	31.6	21	-	14.5	4.2	-	65.6	12.8	-	29.9
Cortex-A7 @ 1.4 GHz															
B	503	-	43	788	-	54	441	-	123	421	-	64			
S	429	-	56	613	-	57	420	-	92	423	-	71			
%	14.8	-	31.6	22.2	-	5.2	4.8	-	25.7	0.6	-	12	12.1	-	20.5
Cortex-A15 @ 200 MHz															
B	-	256	38	-	308	39	-	254	61	-	211	53			
S	-	234	55	-	232	56	-	234	77	-	230	58			
%	-	9.6	31.8	-	33.1	30.1	-	8.7	21.3	-	8	8.1	-	11.2	23.7
Cortex-A15 @ 2 GHz															
B	-	4451	31	-	5853	45	-	4701	183	-	2889	66			
S	-	4648	66	-	6385	66	-	4465	135	-	3435	85			
%	-	4.2	53.3	-	8.3	32.7	-	5.3	35.8	-	15.9	22.7	-	12.7	28.7
Cortex-A7 @ 200 MHz / Cortex-A15 @ 200 MHz															
B	34	271	42	50	316	42	38	293	50	19	228	51			
S	29	270	62	51	424	63	31	289	76	24	235	74			
%	15	0.1	47.7	1.7	33.9	49.1	18.6	1.4	50.7	24.8	3.1	11.1	14.9	9.3	45.4
Cortex-A7 @ 1.4 GHz / Cortex-A15 @ 2 GHz															
B	365	5246	42	589	5490	61	385	5812	108	206	3177	67			
S	346	4985	58	461	6360	60	317	5431	101	241	3056	67			
%	5.2	5	38.7	21.8	15.8	2	17.7	6.5	6	16.9	3.8	0.6	19.9	10.9	22.7

Bibliography

[1] TOP500 Supercomputing Sites, 2015. URL http://top500.org/.

[2] Roger Dangel, Jens Hofrichter, Folkert Horst, Daniel Jubin, Antonio La Porta, Norbert Meier, Ibrahim Murat Soganci, Jonas Weiss, and Bert Jan Offrein. Polymer waveguides for electro-optical integration in data centers and high-performance computers. *Opt. Express*, 23(4):4736–4750, Feb 2015. doi: 10.1364/OE.23.004736. URL http://www.opticsexpress.org/abstract.cfm?URI=oe-23-4-4736.

[3] Patrick Thibodeau. Scientists, it community await exascale computers. December 2009.

[4] Ranking the World's Most energy-efficient supercomputers, 2015. URL http://www.green500.org/.

[5] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with commodity cpus: Are mobile socs ready for hpc? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 40:1–40:12, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2378-9. doi: 10.1145/2503210.2503281. URL http://doi.acm.org/10.1145/2503210.2503281.

[6] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, pages 64–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2143-6. URL http://dl.acm.org/citation.cfm?id=998680.1006707.

[7] NVIDIA Corporation. Nvidia tegra mobile processors, 2015. URL `http://www.nvidia.com`.

[8] Samsung. Exynos Octa 5422 SoC, 2015. URL `https://http://www.samsung.com/`.

[9] Tong Li, Dan Baumberger, David A. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–11, Nov 2007. doi: 10.1145/1362622.1362694.

[10] T.S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–9, May 2013.

[11] Kisoo Yu, Donghee Han, Changhwan Youn, Seungkon Hwang, and Jaechul Lee. Power-aware task scheduling for big.little mobile processor. In *SoC Design Conference (ISOCC), 2013 International*, pages 208–212, Nov 2013. doi: 10.1109/ISOCC.2013.6864009.

[12] Cheng Tan, T.S. Muthukaruppan, T. Mitra, and Lei Ju. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 618–623, Jan 2015. doi: 10.1109/ASPDAC.2015.7059077.

[13] R.G. Sargent. Verification and validation of simulation models. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 166–183, Dec 2010. doi: 10.1109/WSC.2010.5679166.

[14] B. Black and J.P. Shen. Calibration of microprocessor performance models. *Computer*, 31(5):59–65, May 1998. ISSN 0018-9162. doi: 10.1109/2.675637.

[15] Poul E. Heegaard. Speed-up techniques for simulation. *TELEKTRONIKK*, 91: 85–7130, 1995.

[16] Richard Fujimoto. Distributed simulation challenges in sensor networks and the cloud. Presented as the NSF Workshop on Simulation Methodology, 2012.

[17] Mont-Blanc project, 2015. URL `www.montblanc-project.eu`.

[18] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, aug 2011. ISSN 0163-5964. doi: 10.1145/2024716.2024718. URL `http://doi.acm.org/10.1145/2024716.2024718`.

[19] L.P. Hewlett-Packard Development Company. Mcpat, 2008. URL `http://www.hpl.hp.com/research/mcpat/`.

[20] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli. Accuracy evaluation of gem5 simulator system. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7, July 2012.

[21] A. Butko, A. Gamatié, G. Sassatelli, L. Torres, and M. Robert. Design exploration for next generation high-performance manycore on-chip systems: Application to big.little architectures. In *VLSI (ISVLSI), 2015 IEEE Computer Society Annual Symposium on*, July 2015.

[22] Anastasiia Butko. ADAptive Computing group, 2015. URL `http://www.lirmm.fr/ADAC/`.

[23] A. Butko, R. Garibotti, L. Ost, V. Lapotre, A. Gamatié, G. Sassatelli, and C. Adeniyi-Jones. A trace-driven approach for fast and accurate simulation of manycore architectures. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 707–712, Jan 2015.

[24] Rafael Garibotti. *Exploration of Compute Accelerators for High Performance Computing*. PhD thesis, University Montpellier II, 2014.

[25] Osman Balci. The implementation of four conceptual frameworks for simulation modeling in high-level languages. In *Proceedings of the 20th Conference on Winter Simulation*, WSC '88, pages 287–295, New York, NY, USA, 1988. ACM. ISBN 0-911801-42-1. doi: 10.1145/318123.318204. URL `http://doi.acm.org/10.1145/318123.318204`.

[26] A. Muzy, J.J. Nutaro, B.P. Zeigler, and P. Coquillard. Modeling and simulation of fire spreading through the activity tracking paradigm. *Ecological Modelling*, 219(1–2):212 – 225, 2008. ISSN 0304-3800. doi: http://dx.doi.org/10.

1016/j.ecolmodel.2008.08.017. URL `http://www.sciencedirect.com/science/article/pii/S0304380008004134`.

[27] N. Matloff. *Introduction to discrete-event simulation and the simpy language*. 2008. URL `http://heather.cs.ucdavis.edu/~{}matloff/156/PLN/`.

[28] Antonio Cuomo, Massimiliano Rak, and Umberto Villano. Process-oriented discrete-event simulation in java with continuations:quantitative performance evaluation. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH) - Rome, 28-31 July 2012*, volume 2nd International Conference on Simulation and Modeling Methodologies,Technologies and Applications, pages 87–96. SciTePress, 2012. doi: 10.5220/0004014500870096. URL `http://deal.ing.unisannio.it/perflab/assets/papers/simultech2012.pdf`.

[29] I. Almasri, G. Abandah, A. Shhadeh, and A. Shahrour. Universal isa simulator with soft processor fpga implementation. In *Applied Electrical Engineering and Computing Technologies (AEECT), 2011 IEEE Jordan Conference on*, pages 1–6, Dec 2011. doi: 10.1109/AEECT.2011.6132512.

[30] Gregory V. Caliri. Introduction to analytical modeling. In *Int. CMG Conference*, pages 31–36. Computer Measurement Group, 2000. URL `http://dblp.uni-trier.de/db/conf/cmg/cmg2000.html`.

[31] M.T. Yourst. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 23–34, April 2007. doi: 10.1109/ISPASS.2007.363733.

[32] Computer Architecture and Power Aware Systems Research Group. State University of New York at Binghamton. Micro-ARchitectural and System Simulator for x86-based Systems, 2015. URL `http://marss86.org/~marss86/index.php/Home`.

[33] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, Feb 2002. ISSN 0018-9162. doi: 10.1109/2.982916.

[34] WIND An Intel Company. WIND RIVER SIMICS. Simulate Anything, Chip to System, 2015. URL `http://www.windriver.com/products/simics/`.

[35] Gengbin Zheng, Gunavardhan Kakulapati, and L.V. Kale. Bigsim: a parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 78–, April 2004. doi: 10.1109/IPDPS.2004.1303013.

[36] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, Feb 2002. ISSN 0018-9162. doi: 10.1109/2.982917.

[37] Dominik Madon, Eduardo Sanchez, and Stefan Monnier. A study of a simultaneous multithreaded processor implementation. In *Euro-Par '99: Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, pages 716–726, London, UK, 1999. Springer-Verlag. ISBN 3-540-66443-2.

[38] Joseph J. Sharkey, Dmitry Ponomarev, and Kanad Ghose. Abstract m-sim: A flexible, multithreaded architectural simulation environment. Technical report, Department of Computer Science, State University of New York at Binghamton, 2005.

[39] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, pages 392–403, June 1995.

[40] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004. ISSN 1544-3566. doi: 10.1145/980152.980157. URL `http://doi.acm.org/10.1145/980152.980157`.

[41] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez. Multi2sim: A simulation framework to evaluate multicore-multithreaded processors. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 62–68, Oct 2007. doi: 10.1109/SBAC-PAD.2007.17.

[42] Pablo Montesinos Ortego, Paul Sack. SESC: SuperESCalar Simulator, 2004. URL `http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/`.

[43] Ehsan K. Ardestani and Jose Renau. Esesc: A fast multicore simulator using time-based sampling. In *in International Symposium on High Performance Computer Architecture*, page 19, 2013.

[44] Lisa R. Hsu Ali G. Saidi, Nathan L. Binkert and Steven K. Reinhardt. Performance validation of network-intensive workloads on a full-system simulator. In *Proceedings of the First Annual Workshop on Interaction between Operating System and Computer Architecture (IOSCA)*, pages 33–38, 2005.

[45] A. Gutierrez, J. Pusdesris, R.G. Dreslinski, T. Mudge, C. Sudanthi, C.D. Emmons, M. Hayenga, and N. Paver. Sources of error in full-system simulation. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 13–22, March 2014. doi: 10.1109/ISPASS.2014.6844457.

[46] F.A. Endo, D. Courousse, and H.-P. Charles. Micro-architectural simulation of in-order and out-of-order arm microprocessors with gem5. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 266–273, July 2014.

[47] Fernando A. Endo, Damien Couroussé, and Henri-Pierre Charles. Micro-architectural simulation of embedded core heterogeneity with gem5 and mcpat. In *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM. ISBN 978-1-60558-699-1. doi: 10.1145/2693433.2693440. URL `http://doi.acm.org/10.1145/2693433.2693440`.

[48] Sam Likun Xi, H. Jacobson, P. Bose, Gu-Yeon Wei, and D. Brooks. Quantifying sources of error in mcpat and potential impacts on architectural studies. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 577–589, Feb 2015. doi: 10.1109/HPCA.2015.7056064.

[49] Jianwei Chen, Murali Annavaram, and Michel Dubois. Slacksim: a platform for parallel simulations of cmps on cmps. *SIGARCH Comput. Archit. News*, 37(2): 20–29, July 2009. ISSN 0163-5964. doi: 10.1145/1577129.1577134. URL `http://doi.acm.org/10.1145/1577129.1577134`.

[50] Mieszko Lis, Pengju Ren, Myong Hyon Cho, Keun Sup Shim, Christopher W. Fletcher, Omer Khan, and Srinivas Devadas. Scalable, accurate multicore simulation in the 1000-core era., 2011. URL `http://dblp.uni-trier.de/db/conf/ispass/ispass2011.html`.

[51] J.E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, 2010. doi: 10.1109/HPCA.2010.5416635.

[52] Daniel Sanchez and Christos Kozyrakis. Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 475–486, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2079-5. doi: 10.1145/2485922.2485963. URL `http://doi.acm.org/10.1145/2485922.2485963`.

[53] OVP. Open virtual platforms, 2013. URL `http://www.ovpworld.org/`.

[54] QEMU. Qemu open source processor emulator, 2013. URL `http://wiki.qemu.org/Main_Page`.

[55] D. Thach, Y. Tamiya, S. Kuwamura, and A. Ike. Fast cycle estimation methodology for instruction-level emulator. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 248–251, 2012. doi: 10.1109/DATE.2012.6176470.

[56] S. Stattelmann, S. Ottlik, A. Viehl, O. Bringmann, and W. Rosenstiel. Combining instruction set simulation and wcet analysis for embedded software performance estimation. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 295–298, 2012. doi: 10.1109/SIES.2012.6356600.

[57] Imperas. Quantumleap simulation synchronization, 2013. URL `http://www.ovpworld.org/`.

[58] L. Cai and D. Gajski. Transaction level modeling: an overview. In *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, pages 19–24, Oct 2003. doi: 10.1109/CODESS.2003.1275250.

[59] A. Rico, A. Duran, F. Cabarcas, Y. Etsion, A. Ramirez, and M. Valero. Trace-driven simulation of multithreaded applications. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 87–96, April 2011. doi: 10.1109/ISPASS.2011.5762718.

[60] T. Wild, A. Herkersdorf, and R. Ohlendorf. Performance evaluation for system-on-chip architectures using trace-based transaction level simulation. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1–6, March 2006. doi: 10.1109/DATE.2006.244111.

[61] R. Plyaskin, A. Masrur, M. Geier, S. Chakraborty, and A. Herkersdorf. High-level timing analysis of concurrent applications on mpsoc platforms using memory-aware trace-driven simulations. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 229–234, Sept 2010. doi: 10.1109/VLSISOC.2010.5642665.

[62] Synopsys, Inc. Synopsys Silicon to Software, 2015. URL `http://www.synopsys.com`.

[63] S. Abdi, G. Schirner, Yonghyun Hwang, D.D. Gajski, and Lochi Yu. Automatic tlm generation for early validation of multicore systems. *Design Test of Computers, IEEE*, 28(3):10–19, 2011. ISSN 0740-7475. doi: 10.1109/MDT.2010.117.

[64] S. Fytraki and D. Pnevmatikatos. Resim, a trace-driven, reconfigurable ilp processor simulator. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 536–541, April 2009. doi: 10.1109/DATE.2009.5090722.

[65] Michael Pellauer, Muralidaran Vijayaraghavan, Michael Adler, Arvind, and Joel Emer. A-ports: An efficient abstraction for cycle-accurate performance models on fpgas. In *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, FPGA '08, pages 87–96, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-934-0. doi: 10.1145/1344671.1344685. URL `http://doi.acm.org/10.1145/1344671.1344685`.

[66] D. Chiou, Dam Sunwoo, Joonsoo Kim, N. Patil, W.H. Reinhart, D.E. Johnson, and Zheng Xu. The fast methodology for high-speed soc/computer simulation. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 295–302, Nov 2007. doi: 10.1109/ICCAD.2007.4397280.

[67] Jan Edler and Mark D. Hill. Dinero IV Trace-Driven Uniprocessor Cache Simulator, 2015. URL `http://pages.cs.wisc.edu/~markhill/DineroIV/`.

[68] P. Jacob, O. Erdogan, A. Zia, P.M. Belemjian, R.P. Kraft, and J.F. McDonald. Predicting the performance of a 3d processor-memory chip stack. *Design Test of Computers, IEEE*, 22(6):540–547, Nov 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.151.

[69] Michael Laurenzano, Beth Simon, Allan Snavely, and Meghan Gunn. Low cost trace-driven memory simulation using simpoint. *SIGARCH Comput. Archit. News*, 33(5):81–86, December 2005. ISSN 0163-5964. doi: 10.1145/1127577.1127593. URL `http://doi.acm.org/10.1145/1127577.1127593`.

[70] A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 149–156, Dec 2001. doi: 10.1109/WWC.2001.990754.

[71] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. *SIGPLAN Not.*, 37(10):45–57, October 2002. ISSN 0362-1340. doi: 10.1145/605432.605403. URL `http://doi.acm.org/10.1145/605432.605403`.

[72] F. Trivino, F.J. Andujar, F.J. Alfaro, J.L. Sanchez, and A. Ros. Self-related traces: An alternative to full-system simulation for nocs. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 819–824, July 2011. doi: 10.1109/HPCSim.2011.5999914.

[73] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, NoCArc '10, pages 31–36, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0397-2. doi: 10.1145/1921249.1921258. URL `http://doi.acm.org/10.1145/1921249.1921258`.

[74] Shankar Mahadevan, Federico Angiolini, Michael Storgaard, and Rasmus Grøndahl Olsen. A network traffic generator model for fast network-on-chip simulation. In *In Proceedings of the conference on Design, Automation and Test in Europe*, pages 780–785. IEEE Computer Society, 2005.

[75] Dohyung Kim, Youngmin Yi, and Soonhoi Ha. Trace-driven hw/sw cosimulation using virtual synchronization technique. In *Design Automation Conference, 2005. Proceedings. 42nd*, pages 345–348, June 2005. doi: 10.1109/DAC.2005.193830.

[76] Youngmin Yi, Dohyung Kim, and Soonhoi Ha. Fast and time-accurate cosimulation with os scheduler modeling. *Design Autom. for Emb. Sys.*, 8(2-3):211–228, 2003. URL `http://dblp.uni-trier.de/db/journals/dafes/dafes8.html`.

[77] Li Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell. Exploring large-scale cmp architectures using manysim. *Micro, IEEE*, 27(4):21–33, July 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.66.

[78] Sadagopan Srinivasan, Li Zhao, Brinda Ganesh, Bruce Jacob, Mike Espig, and Ravi Iyer. Cmp memory modeling: How much does accuracy matter? In *Fifth Annual Workshop on Modeling, Benchmarking and Simulation*, pages 24–33, June 2009.

[79] Sangyeun Cho, Socrates Demetriades, Shayne Evans, Lei Jin, Hyunjin Lee, Kiyeon Lee, and Michael Moeng. Tpts: A novel framework for very fast manycore processor architecture simulation. In *Int'l Conf. on Parallel Processing (ICPP)*, pages 446–453, 2008.

[80] Kai Huang, I. Bacivarov, Jun Liu, and W. Haid. A modular fast simulation framework for stream-oriented mpsoc. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 74–81, July 2009. doi: 10.1109/SIES.2009.5196198.

[81] A. Quesada M. Pavlovic A. J. Vega Y. Etsion A. Rico, F. Cabarcas and A. Ramirez. Scalable simulation of decoupled accelerator architectures. Technical report, Universitat Politecnica de Catalunya, 2010.

[82] Pengfei Zhu, Mingyu Chen, Yungang Bao, Licheng Chen, and Yongbing Huang. Trace-driven simulation of memory system scheduling in multithread application. In *Proceedings of the 2012 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, MSPC '12, pages 30–37, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1219-6. doi: 10.1145/2247684.2247691. URL `http://doi.acm.org/10.1145/2247684.2247691`.

[83] Nagesh B. Lakshminarayana Jaewoong Sim Jieun Lim Tri Pho Hyesoon Kim, Jaekyu Lee. *MacSim: A CPU-GPU Heterogeneous Simulation Framework.* HPArch research group.

[84] S. Nilakantan, K. Sangaiah, A. More, G. Salvadory, B. Taskin, and M. Hempstead. Synchrotrace: synchronization-aware architecture-agnostic traces for light-weight multicore simulation. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pages 278–287, March 2015. doi: 10.1109/ISPASS.2015.7095813.

[85] Valgrind^{TM} Developers. Valgrind, 2013. URL `http://valgrind.org/`.

[86] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, pages 266–277, New York, NY, USA, 2001. ACM. ISBN 0-7695-1162-7. doi: 10.1145/379240.565338. URL `http://doi.acm.org/10.1145/379240.565338`.

[87] 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. *Micro, IEEE*, 26(4): 52–60, July 2006. ISSN 0272-1732. doi: 10.1109/MM.2006.82.

[88] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, nov 2005. ISSN 0163-5964. doi: 10.1145/1105734.1105747. URL `http://doi.acm.org/10.1145/1105734.1105747`.

[89] Sheng Li, Jung Ho 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 *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480, Dec 2009.

[90] R.A. Fisher. *Statistical methods for research workers.* Edinburgh Oliver & Boyd, 1925.

[91] David P. Rodgers. Improvements in multiprocessor system design. *SIGARCH Comput. Archit. News*, 13(3):225–231, June 1985. ISSN 0163-5964. doi: 10.1145/327070.327215. URL `http://doi.acm.org/10.1145/327070.327215`.

[92] *SKY-S9500-ULP-CXX (aka Snowball PDK-SDK). Hardware Reference Manual.* Calao-Systems, July 1 2011. Revision 1.0.

[93] Igor Pavlov. 7-zip lzma benchmark samsung exynos 5250 arm cortex-a15, 2015. URL `http://7-cpu.com/`.

[94] Man-Lap Li, R. Sasanka, S.V. Adve, Yen-Kuang Chen, and E. Debes. The alpbench benchmark suite for complex multimedia applications. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 34–45, Oct 2005. doi: 10.1109/IISWC.2005.1525999.

[95] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, pages 24–36, June 1995.

[96] Posix threads programming, 2015. URL `https://computing.llnl.gov/tutorials/pthreads/`.

[97] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[98] M.A. Richards. The rapid prototyping of application specific signal processors (rassp) program: overview and status. In *Rapid System Prototyping, 1994. Shortening the Path from Specification to Prototype. Proceedings., Fifth International Workshop on*, pages 1–6, Jun 1994. doi: 10.1109/IWRSP.1994.315915.

[99] Brian Jeff. big.little technology moves towards fully heterogeneous global task scheduling. November 2013. URL `http://www.arm.com/files/pdf/`.

[100] *CoreLink CCI-400 Cache Coherent Interconnect Technical Reference Manual.* ARM, November 16 2012. Revision r1p1.

[101] CoherentXBar Class Reference, 2015. URL `http://www.gem5.org/docs/`.

[102] Shuai Che, M. Boyer, Jiayuan Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54, Oct 2009. doi: 10.1109/IISWC.2009.5306797.

[103] Martin Krzywinski and Naomi Altman. Points of Significance: Visualizing samples with box plots. *Nature Methods*, 11(2):119–120, January 2014. ISSN 1548-7091. doi: 10.1038/nmeth.2813. URL `http://dx.doi.org/10.1038/nmeth.2813`.

[104] Kristin Potter. Methods for presenting statistical information: The box plot. *Hans Hagen, Andreas Kerren, and Peter Dannenmann (Eds.), Visualization of Large and Unstructured Data Sets, GI-Edition Lecture Notes in Informatics (LNI)*, S-4: 97–106, 2006.

[105] gem5. Classic Memory System, 2015. URL `http://www.m5sim.org`.

[106] E.E. Johnson, Jiheng Ha, and M. Baqar Zaidi. Lossless trace compression. *Computers, IEEE Transactions on*, 50(2):158–173, Feb 2001. ISSN 0018-9340. doi: 10.1109/12.908991.

[107] E.E. Johnson and Jiheng Ha. Pdats lossless address trace compression for reducing file size and access time. In *Computers and Communications, 1994., IEEE 13th Annual International Phoenix Conference on*, pages 213–, Apr 1994. doi: 10.1109/PCCC.1994.504117.

[108] N.C. Thornock and J.K. Flanagan. Facilitating level three cache studies using set sampling. In *Simulation Conference, 2000. Proceedings. Winter*, volume 1, pages 471–479 vol.1, 2000. doi: 10.1109/WSC.2000.899754.

[109] John W.C. Fu and J.H. Patel. Trace driven simulation using sampled traces. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 1, pages 211–220, Jan 1994. doi: 10.1109/HICSS.1994.323170.

[110] Richard A. Uhlig and Trevor N. Mudge. Trace-driven memory simulation: A survey. *ACM Comput. Surv.*, 29(2):128–170, June 1997. ISSN 0360-0300. doi: 10.1145/254180.254184. URL `http://doi.acm.org/10.1145/254180.254184`.

[111] Wen-Hann Wang and Jean-Loup Baer. Efficient trace-driven simulation methods for cache performance analysis. *ACM Trans. Comput. Syst.*, 9(3):222–241, August 1991. ISSN 0734-2071. doi: 10.1145/128738.128740. URL `http://doi.acm.org/10.1145/128738.128740`.

[112] P.S. Cheng. Trace-driven system modeling. *IBM Systems Journal*, 8(4):280–289, 1969. ISSN 0018-8670. doi: 10.1147/sj.84.0280.

[113] Bryan Black, Andrew S. Huang, Mikko H. Lipasti, and John Paul Shen. Can trace-driven simulators accurately predict superscalar performance? pages 478–485, 1996.

[114] Eric Larson and Saugata Chatterjee. Mase: A novel infrastructure for detailed microarchitectural modeling. In *in Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, pages 1–9, 2001.

[115] M.M. Tikir, M.A. Laurenzano, L. Carrington, and A. Snavely. Psins: An open source event tracer and execution simulator. In *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, pages 444–449, June 2009. doi: 10.1109/HPCMP-UGC.2009.73.

[116] Sergi Girona, Jesús Labarta, and Rosa M. Badia. Validation of dimemas communication model for MPI collective operations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 7th European PVM/MPI Users' Group Meeting, Balatonfüred, Hungary, September 2000, Proceedings*, pages 39–46, 2000. doi: 10.1007/3-540-45255-9_9. URL `http://dx.doi.org/10.1007/3-540-45255-9_9`.

[117] Hesham El-Rewini and Mostafa Abd-El-Barr. *Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2005. ISBN 0471467405.

[118] OpenMP Architecture Review Board. The openmp api specification for parallel programming, 2015. URL `http://openmp.org/wp/`.

[119] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *SIGPLAN Not.*, 30(8):207–216, August 1995. ISSN 0362-1340. doi: 10.1145/209937.209958. URL `http://doi.acm.org/10.1145/209937.209958`.

[120] Intel. Intel cilk plus, 2013. URL `https://www.cilkplus.org/`.

[121] Richard A Uhlig. *Trap-driven Memory Simulation*. PhD thesis, University of Michigan, 1995.

[122] gprof. Gnu gprof, 2013. URL `http://sourceware.org/binutils/docs/gprof`.

[123] ARM. big.little technology, 2015. URL `http://www.arm.com/products/processors/technologies`.

[124] Kevin Skadron and Ke Wang. Rodinia:accelerating compute-intensive applications with accelerators, 2014. URL `http://lava.cs.virginia.edu/Rodinia/`.

[125] Scalasca, 2015. URL `http://www.scalasca.org/`.

[126] Vampir - performance optimization, 2015. URL `https://www.vampir.eu/`.

[127] Ching-Long Su and Alvin M. Despain. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of the 1995 International Symposium on Low Power Design*, ISLPED '95, pages 63–68, New York, NY, USA, 1995. ACM. ISBN 0-89791-744-8. doi: 10.1145/224081.224093. URL `http://doi.acm.org/10.1145/224081.224093`.

[128] S. Sarma and N. Dutt. Cross-layer exploration of heterogeneous multicore processor configurations. In *VLSI Design (VLSID), 2015 28th International Conference on*, pages 147–152, Jan 2015. doi: 10.1109/VLSID.2015.30.

[129] MediaTek Inc. MT6595 Octa-core LTE platform, 2015. URL `http://www.mediatek.com/en/products/mobile-communications/smartphone1/mt6595/`.

[130] Qualcomm Technologies, Inc. Qualcomm Snapdragon Processors, 2015. URL `https://www.qualcomm.com/products/snapdragon/processors`.

[131] The MathWorks, Inc. MATLAB The Language of Technical Computing, 2015. URL `www.mathworks.com/products/matlab/`.

[132] M.A. O'Neil and M. Burtscher. Microarchitectural performance characterization of irregular gpu kernels. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 130–139, Oct 2014. doi: 10.1109/IISWC.2014.6983052.

[133] P. Fritzsche, C. Roig, A. Ripoll, E. Luque, and A. Hernandez. A performance prediction methodology for data-dependent parallel applications. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–8, Sept 2006. doi: 10.1109/CLUSTR.2006.311879.

[134] Daniel J. Sorin, Mark D. Hill, and David A. Wood. *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers, 1st edition, 2011. ISBN 1608455645, 9781608455645.

[135] Robert Pyka, Christoph Fassbach, Manish Verma, Heiko Falk, and Peter Marwedel. Operating system integrated energy aware scratchpad allocation strategies for multiprocess applications. In *Proceedingsof the 10th International Workshop on Software &Amp; Compilers for Embedded Systems*, SCOPES '07, pages 41–50, New York, NY, USA, 2007. ACM. doi: 10.1145/1269843.1269850. URL `http://doi.acm.org/10.1145/1269843.1269850`.

[136] Carlos Villavieja, Yoav Etsion, Alex Ramirez, and Nacho Navarro. Feli: Hw/sw support for on-chip distributed shared memory in multicores. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6852 of *Lecture Notes in Computer Science*, pages 282–294. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23399-9. doi: 10.1007/978-3-642-23400-2_27. URL `http://dx.doi.org/10.1007/978-3-642-23400-2_27`.

[137] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Juan Yi, Meikang Qiu, and E.H.-M. Sha. Data placement and duplication for embedded multicore systems with scratch pad memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(6):809–817, June 2013. ISSN 0278-0070. doi: 10.1109/TCAD.2013.2238990.

[138] Janmartin Jahn Joerg Henkel Nikil Dutt Hossein Tajik, Bryan Donyanavard. Spmpool: Runtime spm management for embedded many-cores. Technical report, Center for Embedded Computer Systems University of California, Irvine, USA, May 2014.

[139] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. *SIGARCH Comput. Archit.*

News, 37(3):14–23, June 2009. ISSN 0163-5964. doi: 10.1145/1555815.1555759. URL http://doi.acm.org/10.1145/1555815.1555759.

[140] Jingtong Hu, C.J. Xue, Wei-Che Tseng, Y. He, Meikang Qiu, and E.H.M. Sha. Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 350–355, June 2010.

[141] Wei-Che Tseng, C.J. Xue, Qingfeng Zhuge, Jingtong Hu, and E.H.M. Sha. Optimal scheduling to minimize non-volatile memory access time with hardware cache. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 131–136, Sept 2010. doi: 10.1109/VLSISOC.2010.5642609.

[142] Jingtong Hu, C.J. Xue, Qingfeng Zhuge, Wei-Che Tseng, and E.H.-M. Sha. Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011. doi: 10.1109/DATE.2011.5763127.

[143] Luis Angel Bathen and Nikil Dutt. Havoc: A hybrid memory-aware virtualization layer for on-chip distributed scratchpad and non-volatile memories. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 447–452, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1199-1. doi: 10.1145/2228360.2228438. URL http://doi.acm.org/10.1145/2228360.2228438.

[144] Jingtong Hu, Qingfeng Zhuge, Chun Jason Xue, Wei-Che Tseng, and Edwin H.-M. Sha. Management and optimization for nonvolatile memory-based hybrid scratchpad memory on multicore embedded processors. *ACM Trans. Embed. Comput. Syst.*, 13(4):79:1–79:25, March 2014. ISSN 1539-9087. doi: 10.1145/2560019. URL http://doi.acm.org/10.1145/2560019.

[145] Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Processing Letters*, 21(2):173–193, 2011. doi: 10.1142/S0129626411000151. URL http://dx.doi.org/10.1142/S0129626411000151.