
Universidade Estadual de Campinas
Instituto de Computação


Liana Dessandre Duenha Garanhani

# MPSoCBench: a Framework for High-Level Evaluation of Multiprocessor System-on-Chip Tools and Methodologies

# MPSoCBench: um Framework para Avaliação de Ferramentas e Metodologias para Sistemas Multiprocessados em Chip

CAMPINAS
2015

**Liana Dessandre Duenha Garanhani**

# MPSoCBench: a Framework for High-Level Evaluation of Multiprocessor System-on-Chip Tools and Methodologies

# MPSoCBench: um Framework para Avaliação de Ferramentas e Metodologias para Sistemas Multiprocessados em Chip

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

**Supervisor/Orientador: Prof. Dr. Rodolfo Jardim de Azevedo**

Este exemplar corresponde à versão final da Tese defendida por Liana Dessandre Duenha Garanhani e orientada pelo Prof. Dr. Rodolfo Jardim de Azevedo.

CAMPINAS
2015

**Agência(s) de fomento e nº(s) de processo(s):** Não se aplica.

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

G161m

Garanhani, Liana Dessandre Duenha, 1977-
MPSoCBench : a framework for high-level evaluation of Multiprocessor System-on-Chip tools and methodologies / Liana Dessandre Duenha Garanhani. – Campinas, SP : [s.n.], 2015.

Orientador: Rodolfo Jardim de Azevedo.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Simulação (Computadores). 2. Sistemas embarcados (Computadores). 3. Energia - Consumo. 4. Multiprocessadores. 5. Arquitetura de computador. I. Azevedo, Rodolfo Jardim de,1974-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

**Título em outro idioma:** MPSoCBench : um framework para avaliação de ferramentas e metodologias para sistemas multiprocessados em chip
**Palavras-chave em inglês:**
Computer simulation
Embedded computer systems
Energy consumption
Multiprocessors
Computer architecture
**Área de concentração:** Ciência da Computação
**Titulação:** Doutora em Ciência da Computação
**Banca examinadora:**
Rodolfo Jardim de Azevedo [Orientador]
Fernando Gehm Moraes
Alexandro Baldassin
Guido Araújo
Lucas Wanner
**Data de defesa:** 27-11-2015
**Programa de Pós-Graduação:** Ciência da Computação


**Universidade Estadual de Campinas**
**Instituto de Computação**


**Liana Dessandre Duenha Garanhani**

# MPSoCBench: a Framework for High-Level Evaluation of Multiprocessor System-on-Chip Tools and Methodologies

# MPSoCBench: um Framework para Avaliação de Ferramentas e Metodologias para Sistemas Multiprocessados em Chip

**Banca Examinadora:**

- Prof. Dr. Rodolfo Jardim de Azevedo
  Universidade Estadual de Campinas (UNICAMP)
- Prof. Dr. Fernando Gehm Moraes
  Pontifícia Universidade Católica do Rio Grande do SUL (PUCRS)
- Prof. Dr. Alexandro Baldassin
  Universidade Estadual Paulista – Campus de Rio Claro (UNESP)
- Prof. Dr. Guido Araújo
  Universidade Estadual de Campinas (UNICAMP)
- Prof. Dr. Lucas Wanner
  Universidade Estadual de Campinas (UNICAMP)

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 27 de novembro de 2015

*Dedico este trabalho à minha linda família,*
*pela qual tudo vale a pena.*

# Acknowledgements - Agradecimentos

É uma ilusão pensar que os créditos por trabalhos concluídos são somente nossos. Não são! Qualquer que seja o projeto, inevitavelmente nós o realizamos em equipe, e da mesma forma devemos compartilhar os benefícios do seu sucesso. Com este trabalho não foi diferente, cada passo dessa trajetória foi realizada por uma equipe muito esforçada. Entrego aqui meus mais sinceros agradecimentos a estas pessoas tão especiais: às minhas filhas queridas, Iara e Anna, minhas companheiras, que quantas vezes, sem reclamar, devem ter sentido minha falta. Filhas, obrigada pela paciência e compreensão; nada faria sentido sem vocês! Ao Rafael, esposo dedicado e amoroso, que adiou seus próprios planos para me acompanhar e sempre esteve ao meu lado nos bons e maus momentos. Aos meus pais e irmãos por todo amor que me dão e pelos revigorantes e animados domingos com música e boa comida. Ao meu orientador, Prof. Rodolfo, sempre tão presente e motivador, minha admiração e infinitos agradecimentos por conduzir-me nessa jornada. Aos queridos colegas do LSC Tiago, Maxiwell, Gabriel, Raoni e Rafael Auler pelas inúmeras vezes que me ajudaram a entender o GIT (sinto muito, ainda vou precisar de vocês). Ao Diretor da Faculdade de Computação (UFMS), Prof. Nalvo, e ao meu colega de trabalho Prof. Ricardo Santos, por serem sempre tão compreensivos e prontos a ajudar. E aos que me ajudaram sem nem mesmo saber.

Quaisquer que sejam as contribuições técnicas desse trabalho, elas não se comparam à realização pessoal e sensação de dever cumprido que sinto nesse momento. Muito mais do que os resultados profissionais da tese, perdurará em mim e nos mais próximos a mim, o aprendizado obtido em cada etapa vencida. A todos que ajudaram nessa conquista, eterna gratidão!

# Resumo

Recentes metodologias e ferramentas de projetos de sistemas multiprocessados em chip (MPSoC) aumentam a produtividade por meio da utilização de plataformas baseadas em simuladores, antes de definir os últimos detalhes da arquitetura. No entanto, a simulação só é eficiente quando utiliza ferramentas de modelagem que suportem a descrição do comportamento do sistema em um elevado nível de abstração. A escassez de plataformas virtuais de MPSoCs que integrem hardware e software escaláveis nos motivou a desenvolver o MPSoCBench, que consiste de um conjunto escalável de MPSoCs incluindo quatro modelos de processadores (PowerPC, MIPS, SPARC e ARM), organizado em plataformas com 1, 2, 4, 8, 16, 32 e 64 núcleos, cross-compiladores, IPs, interconexões, 17 aplicações paralelas e estimativa de consumo de energia para os principais componentes (processadores, roteadores, memória principal e caches). Uma importante demanda em projetos MPSoC é atender às restrições de consumo de energia o mais cedo possível. Considerando que o desempenho do processador está diretamente relacionado ao consumo, há um crescente interesse em explorar o *trade-off* entre consumo de energia e desempenho, tendo em conta o domínio da aplicação alvo. Técnicas de escalabilidade dinâmica de freqüência e voltagem fundamentam-se em gerenciar o nível de tensão e frequência da CPU, permitindo que o sistema alcance apenas o desempenho suficiente para processar a carga de trabalho, reduzindo, consequentemente, o consumo de energia. Para explorar a eficiência energética e desempenho, foram adicionados recursos ao MPSoCBench, visando explorar escalabilidade dinâmica de voltaegem e frequência (DVFS) e foram validados três mecanismos com base na estimativa dinâmica de energia e taxa de uso de CPU.

# Abstract

Recent design methodologies and tools aim at enhancing the design productivity by providing a software development platform before the definition of the final Multiprocessor System on Chip (MPSoC) architecture details. However, simulation can only be efficiently performed when using a modeling and simulation engine that supports system behavior description at a high abstraction level. The lack of MPSoC virtual platform prototyping integrating both scalable hardware and software in order to create and evaluate new methodologies and tools motivated us to develop the MPSoCBench, a scalable set of MPSoCs including four different ISAs (PowerPC, MIPS, SPARC, and ARM) organized in platforms with 1, 2, 4, 8, 16, 32, and 64 cores, cross-compilers, IPs, interconnections, 17 parallel version of software from well-known benchmarks, and power consumption estimation for main components (processors, routers, memory, and caches). An important demand in MPSoC designs is the addressing of energy consumption constraints as early as possible. Whereas processor performance comes with a high power cost, there is an increasing interest in exploring the trade-off between power and performance, taking into account the target application domain. Dynamic Voltage and Frequency Scaling techniques adaptively scale the voltage and frequency levels of the CPU allowing it to reach just enough performance to process the system workload while meeting throughput constraints, and thereby, reducing the energy consumption. To explore this wide design space for energy efficiency and performance, both for hardware and software components, we provided MPSoCBench features to explore dynamic voltage and frequency scalability (DVFS) and evaluated three mechanisms based on energy estimation and CPU usage rate.

# List of Figures

# List of Tables

# Contents

# Chapter 1

# Introduction

The use of multiprocessors in design of embedded systems introduces new significant challenges for system architects and hardware/software designers. In particular, it is necessary to develop tools based on new paradigms, able to deal with MPSoC complexities [64].

From the design perspective, productivity may be improved by evaluating and optimizing system components through a virtual platform before implementing the final system version. However, simulation can only be efficiently implemented when using a modeling and simulation engine that supports the system behavior description at high abstraction level [21].

By dealing with the disagreement between increasingly complexity and shorter time-to-market, abstract models have been widely used, exhibiting significant gains in simulation speed, allowing fast evaluation and extensive design space exploration. This trade-off essentially allows models at different levels of accuracy and speed. However, it is typically not possible to achieve both high speed and high accuracy at the same time. High-level models fall into the area next to the dashed line in Figure 1.1. Models in the dark area obviously exist, but are practically unusable, because they present low performance and low accuracy, whereas models placed above the blue line are highly desirable although typically not achievable [85].

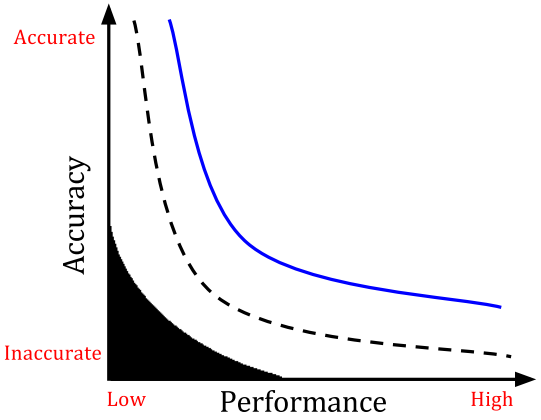



Figure 1.1: Accuracy and Performance Trade-off. Figure reproduced from [85]

Nowadays, there is a massive use of software-based models to verify the accuracy and quantitatively evaluate the system performance. In this scenario, architecture simulators play a significant role in computer architecture design. However, the obvious direction is to explore CMP (chip multiprocessor) designs, which include several new challenges, both in hardware management and development of software that take advantage of the parallelism capabilities.

With the emergence of Multiprocessor Systems on Chip (MPSoCs), power and energy consumption have become the most delicate and limiting issue for the proper functioning of the system. Designers need a global vision of the system power consumption in the earlier design stages to evaluate the effectiveness of the power management strategies beforehand [30].

The first simulators for public use emerged nearly fifteen years ago and have become increasingly robust since then [24]. Recently, some resources have been added to them to achieve power management, mostly through mathematical models for accurate power consumption estimates. The most widely used simulators that include support for power consumption are based on mathematical analysis and estimates rather than based on running applications, as we will see in the literature review presented in Section 2. Therefore, they need to be integrated with data obtained from performance simulators. Although effective, this technique has the disadvantage of having a simulation infrastructure based on two or more tools that do not always have a friendly interface for integration [67, 90].

Summarizing, our hypothesis is that MPSoC virtual platforms integrating high-level description of hardware components, scalable applications, and a target toolchain, focused on architectural exploration, power consumption, and performance estimation have the potential to be used on evaluation of new techniques and methodologies in the early stages of MPSoC and embedded designs.

This thesis presents **MPSoCBench**, a simulation toolset consisting of a scalable set of MPSoCs to enable the development and evaluation of new tools, methodologies, parallel software, and hardware components. The toolset supports four distinct processors (ARM, MIPS, SPARC, and PowerPC) in many configurable and scalable MPSoC platforms with 1, 2, 4, 8, 16, 32, or 64 cores, with different interconnections that define different simulation abstraction levels.

This infrastructure includes 17 parallel benchmarks from SPLASH-2 [96], ParMiBench [58], Mibench [53] benchmarks, and a POSIX PThread emulation library. The tool also provides power consumption estimation for MIPS and SPARC processors, even in multicore environments. A total of 864 distinct configurations are available and automated through an execution script that also allows easy integration with a cluster for parallel execution. The number of possible configurations is even much higher if we consider different cache or power consumption parameters.

## 1.1 Contributions

The main contribution of MPSoCBench is to provide a completely open source simulation infrastructure including scalable hardware and software components, with easy

instrumentation and fast simulation at high abstraction levels to be used on evaluation of new techniques and methodologies in the early stages of MPSoC and embedded designs. We can cite the following specific contributions:

- A System level simulation infrastructure to address requirements in MPSoCs designs;
- Simulation at different abstraction levels, allowing exploring the trade-off between performance and accuracy at the early design stages;
- Easy configuration templates, enabling its use for both research and academic purposes;
- Basic infrastructure for dynamic estimation of power/energy consumption and DVFS techniques exploration;
- A significant set of parallel and scalable applications that use all available models, which characterizes the tool as a benchmark for MPSoCs;

## 1.2 Organization

This Thesis is organized as follows:

**Chapter 2** explores the prior tools or libraries related to this work and exposes how they compare to the MPSoCBench main goals. It introduces basic modeling and simulation concepts and tools, describes prior work related to Dynamic Voltage and Frequency Scaling approaches, and also shows some tools for design space exploration. The main goal of this Chapter is to contextualize the MPSoCBench and compare it with the main tools and approaches that have usually been used.

**Chapter 3** introduces the MPSoCBench toolset, which is a scalable, configurable, and extensible set of MPSoCs, useful to improve development and evaluation of the MPSoC ecosystem, using well-known methodologies and tools. We describe the hardware and power models available in the toolset, and the applications adapted to execute in the MPSoCBench virtual platforms.

**Chapter 4** provides the MPSoCBench characterization, evaluating its main features, applications and issues related to simulators accuracy and performance. The Chapter includes discussions about simulation speed, simulation accuracy, network traffic, memory analysis, power estimation, TLM coding styles, and timing abstraction levels.

**Chapter 5** presents the infrastructure of the MPSoCBench DVFS support, describing power models and evaluating three DVFS techniques based on energy estimation and CPU-usage rate. The main goal of this Chapter is to show that MPSoCBench can be used to perform full-system power efficiency studies in a fast simulation environment;

**Chapter 6** concludes the text and presents our main contributions, published papers and future work. Appendix A presents a tutorial on how to installing necessary tools to use the MPSoCBench. Appendix B presents a tutorial on how to use the MPSoCBench.

# Chapter 2

# Literature Review

This Chapter describes basic concepts and previous work related to our research. To improve clarity, we have divided the review in seven main areas: Section 2.1 introduces some basic concepts, tools and languages related to modeling and simulation; Section 2.2 describes prior work on simulation tools; Section 2.3 reviews basic concepts about Dynamic Voltage and Frequency Scaling (DVFS), Section 2.4 details relevant DVFS approaches, and Section 2.5 presents tools that implement DVFS; Section 2.6 shows tools for design exploration of MPSoCs. Finally, Section 2.7 exposes the benchmark suites that we use to evaluate our tools and methodologies. At the end of each Section we briefly compare the tools previously presented with the MPSoCBench features.

## 2.1 Modeling and Simulation Background

The best choice of the abstraction level in the modeling process is essential to find balance between productivity and modeling level of details. In RTL (Register Transfer Level) design, circuit behavior is described in terms of signals representing data transfer between registers and logic operations conducted over these signals. RTL abstraction is used in hardware description languages like Verilog and VHDL to represent low-level circuits, which are derived from the hardware connections.

Considering the increasing complexity of system designs, the approach adopted by SoC industry is to keep abstraction above RTL during the early stages of the project and reach RTL level in the synthesis stage after successive stages of design refinement. This approach is known as System Level and is characterized by the system specification as a set of software components, hardware components and applications, with a primary focus in the relationship among the subsystems.

**SystemC** [3] is a collection of C++ classes and templates that provides powerful mechanisms to model system architecture with hardware timing, concurrency and reactive behavior, allowing the creation of an executable specification of the system. SystemC is one of the most suitable choices for system design.

In the context of this work, we describe a hardware system as a virtual platform, which is typically implemented as several Transaction-Level (TLM) [2] models, which are a higher level representation of hardware behavior, focusing on discrete events such

as register reading and writing. Although TLM is language independent, SystemC fits perfectly in its representation style, allowing adequate abstraction levels and providing elements to isolate computation and communication.

**ArchC** [13] is an *Architecture Description Language* (ADL) following a SystemC syntax style, which provides enough information in order to allow users to explore and verify a (new or legacy) processor architecture. This ADL automatically generates not only software tools for code generation and inspection (like assemblers, linkers, and debuggers), but also executable processor models for platform representation. **PowerSC** is a SystemC extension designed to collect and process switching activity [60]. As ArchC generates a SystemC processor description, it is an eligible candidate to PowerSC workflow. **acPower** is a library that connects ArchC to PowerSC, thus enabling the power analysis of ArchC processor modules [51].

An important feature of TLM models is their ability to provide data related to the timing and synchronization of devices. Thus, it is possible to establish different abstractions related to the accuracy of the simulator timing information:

- **Untimed:** At this level, the behavior of the components is described without time information associated with the model. Any statistics related to performance can be expressed in terms of the amount and the type of activities performed during the simulation. The CPU models described in this level are called functional models;
- **Approximately timed:** Any time information is added into the model, even if not very accurate, making it possible to couple timed peripherals to the model without difficulty;
- **Cycle-accurate:** At this abstraction level, the microarchitecture is simulated on a cycle-by-cycle basis. Although not always synthesizable, the model behaves exactly like the real implementation of the circuit in regard to the timing. A precise amount of time should be associated with each functionality of the component and the simulated time should take place according to this value.

Figure 2.1 shows that the accuracy increases as the design description progresses from a high level untimed models to cycle-accurate RTL abstraction level. In contrast, the performance decreases when we move from high level to accurate levels.

MPSoCBench uses SystemC 2.3 [3] as simulation infrastructure, TLM2 for standard interconnection, ArchC 2.3 [13] to generate processor models and cross-compilers, and the PowerSC extension for power models. It is open source, and available under GPL license.

In the following sections we describe tools for modeling and simulation in accordance with different abstraction levels.

## 2.2 Modeling and Simulation Tools

A large number of processor simulation tools have been developed in the last decade. In the context of this research, we divide them into roughly two categories: functional simulators and cycle accurate simulators. Functional simulators precisely emulate each

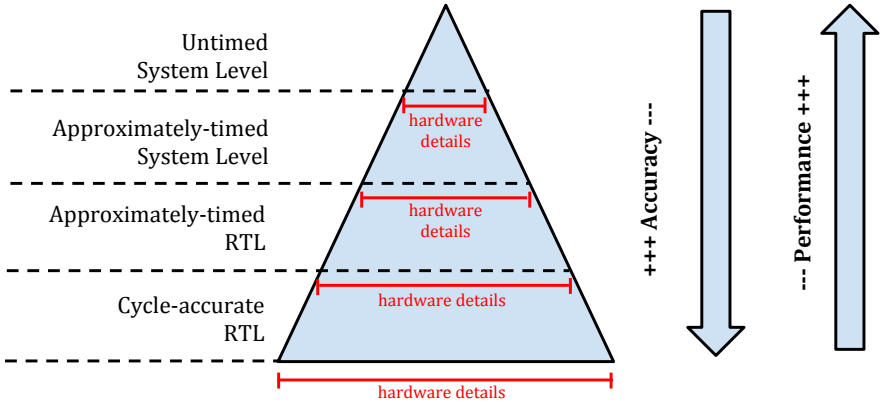



Figure 2.1: Relation between accuracy and performance in system simulators. Figure reproduced from [83].

instruction in the target instruction set, but do not provide any cycle accurate timing information. Cycle accurate simulators, on the other hand, construct a complete microprocessor pipeline in software and simulate the flow of each instruction through this pipeline, to collect much more accurate timing information down to individual cycles. This extra accuracy comes at the cost of longer simulation times.

There are also simulators in lower abstraction level, such as RTL models and circuit level models, but we consider them out of the scope of this work.

### 2.2.1 Functional Simulators

Proceeding from high-level to low-level, the simulations become more accurate, but they also become progressively more complex and take longer to run. Once a behavioral or functional simulation predicts that a system works correctly, it ignores accuracy of timing. Although it may set fixed delays to measure time, this is not the major concern.

A functional simulator is an abstract description of the hardware architecture and its behavior, and it has been widely used in the early stages of hardware design evaluation. The following paragraphs contain a brief description of the existing functional simulators, used both in industry or academic research.

**Bochs** [62] is one of the first well-known open source x86 functional simulator, which supports nearly all its features. Bochs provides complete x86 PC emulation, including the ISA, hardware devices, and memory. This allows users to run OS's and software within the emulator on their workstation. In fact, Bochs is much like a virtual machine that emulates the hardware of interest. The guest OS interacts with the virtual machine, which in turn interacts with the host hardware. The virtual machine collects the instructions the guest is about to run, and emulate them replacing instructions with alternatives as necessary. The disadvantage is that emulation has a substantial computational overhead, and can be slow.

A commercial simulation suite available since 2002 is **Simics** [71], a functional simulation suite for various processor families (including x86) as well as user-designed plug-in

models of real hardware devices. It uses x86-to-x86 binary translation to achieve good performance. However, it does not include cycle-accurate simulation features below the x86 instruction level (just like any other functional simulator). The purpose of simulation in Simics is often to develop software for a particular embedded hardware, using Simics as a virtual platform.

Leveraging the power of Simics as an underlying foundation, Martin et al. presented **GEMS** [73], a full-system multiprocessor simulator that uses Simics as its base and incorporates extensive features to design, specify and validate new cache coherence protocols. The GEMS toolset adds a fairly detailed out-of-order processor simulation model and detailed memory model on top of the functional full-system simulation environment provided by Simics.

Another well known and more robust emulator is **QEMU** [14], that is a fast machine emulator using an original portable dynamic translator, which emulates several CPUs (x86, PowerPC, ARM and SPARC) on several hosts (x86, PowerPC, ARM, SPARC, Alpha and MIPS). It uses dynamic compilation to achieve significantly faster simulation speeds compared to pure interpretation.

One of the first ARM-based platform was presented by Benini et al. [16] as the **MP-ARM**, a complete platform for MPSoC research, including processor models (ARM) modeled in SystemC, SoC bus models (AMBA), memory models, hardware support for parallel programming, a fully operational operating system port (UCLinux) and code development tools (GNU toolchain). The main disadvantages is that the processor cores supported in the MP-ARM framework are relatively simple and the automatic construction of performance models for coprocessors from high-level functional specifications is not supported. Using MP-ARM, Loghi, Poncino and Benini [68] run realistic applications in order to obtain accurate functional behavior and power and performance analysis of the system.

Few years later, Mahadevan et al. [72] proposed a traffic generator model (TG) that is aimed at faithfully replicating traffic patterns generated by a processor running an application. At the same time, this approach allows a straightforward path towards deployment of the TG device on a silicon NoC test chip. To evaluate the concept, the proposed model was integrated into MP-ARM [16].

The first important results about the use of a parallel simulation infrastructure to evaluate techniques for application-to-platform mapping was presented by Paulin et al. in 2004 and 2006 [79,80] and extended by Beltrame, Sciuto and Silvano in 2008 [15]. First of all, Paulin presented the **MultiFlex** [79,80] environment as an *application-to-platform* mapping tool that integrates heterogeneous parallel components (HW and SW) into a platform programming environment; this tool is more focused in network and multimedia applications. Next, in 2008, Beltrame, Sciuto and Silvano [15] showed how it is possible to map applications to the MultiFlex platform. Once the application has been partitioned and mapped to the target platform according to its performance and power constraints, it is possible to further optimize its power consumption with the use of a proper Dynamic Power Management System (DPMS).

In 2007, a single-application simulator named Multi2Sim was proposed by Ubal et al. [94], which intended to simulate final MIPS-32 executable machine. The tool includes cross-compiler and three simulation techniques: functional, detailed and event-driven.

The Michigan **M5** Simulator proposed by Binkert et al. in [18] provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models; It supports the booting of entire operating systems, as well as the execution of unmodified application binaries using system-call emulation. However, only a shared-bus model is supported to model interconnection of multiple processor cores, and only four processor cores can be simulated at a time. In 2011, Nathan et al. [17] presented the merge of the best aspects of the M5 and GEMS [73] simulators, resulting in the **gem5** Simulator. In 2012, the accuracy of this new powerfull tool was evaluated by Butko et al. in [28].

Another extension of the gem5 infrastructure was presented by Power et al. [82] and contains a simulator built on gem5 [17,28] and GPGPU-Sim, a detailed GPGPU simulator. This extension routes most memory accesses through Ruby, which is a highly configurable memory system in gem5. Applications can launch non-blocking kernels, allowing the CPU and GPU to execute simultaneously.

The OVP [57] is a hardware simulator written in C language, instruction-accurate, open-source and able to simulate an entire platform. OVP offers a large model database, supporting several processor families (like MIPS, ARM and PowerPC) besides many peripherals.

In 2014, Endo, Courouss and Charles [43] described the implementation and accuracy evaluation of a micro-architectural simulator of Cortex-A cores, supporting in-order and out-of-order pipelines and based on the open-source gem5 simulator. They showed how to simulate Cortex-A8 and Cortex-A9 cores in gem5, and the execution time of ten benchmarks were compared against real hardware.

In order to provide an open repository of SystemC models, Mello et al. [75] introduced SoCLib, which is an open platform for virtual prototyping of MPSoCs. The core of the platform is a library of SystemC models for virtual components (IP cores). Although there are many components available in SoCLib design repositories, modeling platforms containing multiple devices to evaluate MPSoCs demands a significant effort of joining them together. None of them provides neither a comprehensive set of scalable software to run on platforms nor a framework to simplify new hardware integration.

The **SESC** (SuperESCalar Simulator) [78] is an event driven simulator for the MIPS processor architecture, in which the instructions are executed in an emulation module that emulates the MIPS Instruction Set Architecture (ISA). In 2013, Ardestani et al. [11] presents **ESESC**, an architectural simulator for multicore processors, based on Time-Sampling with virtually no limitation in terms of application type (multiprogrammed or multithreaded), number of cores, homogeneity or heterogeneity of the simulated configuration. They modify SESC and use QEMU as the functional emulator executing ARM instructions.

In the previous paragraphs, we described some of the most popular functional simulators of processors or complete systems, of which some are very elaborate and other pretty simple. Although MPSoCBench contain high level processor models and devices,

its interconnection mechanism allows timed communication using TLM2, placing the MPSoCBench on the edge between functional and timed simulation.

### 2.2.2 Cycle Accurate Simulators

In the following paragraphs we describe some cycle-accurate simulators released in the last decade. Since cycle-accurate models tend to have low performance, most simulators have a hybrid approach, where only part of their components is accurate (for instance: the interconnection, caches, or single processor). Simulation of complete cycle-accurate platforms tend to be very inefficient.

In 1996, Todd Austin et al. [12,26,27] introduced **SimpleScalar**, a simulator for out-of-order superscalar processor pipelines, with the ability to model multiple application workloads or shared-memory parallel applications, as well as the simulation of multiple processor cores. It can model a variety of platforms ranging from simple unpipelined processors to detailed dynamically scheduled microarchitectures with multiple-level memory hierarchies. The tool set's instruction interpreters also support several popular instruction sets, including Alpha, PPC, x86, and ARM. Although robust, SimpleScalar is becoming obsolete due to a dependence to an outdated compiler version. **PTLSim** [98] was the first open source cycle-accurate that that really simulates the x86 instruction set, instead of just interpreting it. PTLsim has models for out of order x86-64 processor cores at a configurable level of detail ranging from RTL-level models of all key pipeline structures, caches and devices up to full-speed native execution on the host CPU. Released two years later by Zeng et al. [99], the **MPTLSim** is a cycle-accurate, full-system simulator for multicore designs based on the x86-64 ISA, a natural multicore extension of PTLSim.

One of the most complete work on MPSoC platforms was proposed by Boukhechem and Bourennane (**StarSoC**, in [21]), which contains an OpenRISC processor as the central part of the system and includes SystemC communication models (Whishbone bus). It made use of an open source ISS of or1Ksim simulator platform, which is designed for uni-processor simulation. In order to use it for the simulation of multiprocessor systems, several ISSs with SystemC communication platform models were connected by using UNIX Inter Process Communication. As an extension of this previous work, Boukhechem, Bourennane and Samahi have compared in [22] three abstraction levels using the StarSoC platform: the traditional register-transfer level modeling (Verilog model) and transaction-level modeling at instruction accurate level and cycle-accurate level. In 2008, Boukhechem et al. [23] defined the methodology to construct the STARSoC TLM simulation environment, which provides a rapid and accurate design space exploration at higher abstraction levels for multiprocessor system on chip architectures.

Ferri et al. [45] provide a cycle-accurate cache-coherent ARM-based cluster similar to ARM's MPCore. This infrastructure is useful for evaluating hardware transactions memory solutions for embedded architectures. Their experiments show that transactional memory can provide clear performance advantages, but it is essential to consider carefully the hardware design in order to conform energy constraints of the system.

Agarwal et al. [5] developed a detailed cycle-accurate interconnection network model (**GARNET**), inside the GEMS full-system simulation framework, which includes models

with microarchitectural details, such as flit-level input buffers, routing logic, allocators and the crossbar switch. GARNET, along with GEMS, provides a detailed and accurate memory system timing model.

Recently, Alali, Assayad, and Sadik [7] introduced an MPSoC composed of MicroBlaze microprocessors, memory, a timer, a VGA and an interrupt handler with two examples of software. The main contribution of this work was to show that simulators using Timed Programmer's View (PV+T) can achieve timing fidelity with high performance.

In the previous paragraphs we briefly discussed about prior cycle-accurate simulation tools and applications, along with their strengths and weaknesses.

## 2.3 Energy Consumption Concepts

The total power consumption $P_{total}$ of the computer system may be separated into two components, as we show in Equation 2.1:

$$P_{total} = P_{system} + P_{CPU} \tag{2.1}$$

$P_{CPU}$ is consumed by the CPU itself and $P_{system}$ by the rest of system. The power consumption $P_{CPU}$ of the CPU can also be split into two parts, as we show in Equation 2.2:

$$P_{CPU} = P_{dyn} + P_{leak} \tag{2.2}$$

$P_{dyn}$ is the power consumed by the CPU during the computation, resulted from the transistors switching activities, and $P_{leak}$ is the power due to leakage effects inherent to silicon-based transistors, originated from currents that flow between differently parts of the transistor. Notwithstanding the existence of multiple sources of leakage in CMOS[1] transistors and their particularities, leakage current models are accurate yet complex since thay depend on the multiple parameters [4].

Given the different sources of power consumption, the total power can be rewritten as we show in Equation 2.3:

$$P_{total} = P_{system} + P_{leak} + P_{dyn} \tag{2.3}$$

The dynamic power consumption in CMOS can be described by Equation 2.4, where $f$ is the switching activity, $C$ is average capacitance loading of the circuit, and $V$ is the supply voltage. To minimize power consumption, we can reduce $f$, $C$ or $V$ [91].

$$P_{dyn} = f.C.V^2 \tag{2.4}$$

Since it was first proposed in 1994 by Weiser et al. [95], *Dynamic Voltage and Frequency Scaling (DVFS)* techniques have proven to be highly effective in achieving low power consumption on a wide range of computing systems.

DVFS deals with the management of a system power consumption and its main idea is to adaptively scale the supply voltage and frequency levels of the CPU so as to reach

[1]CMOS is short for complementary metal oxide semiconductor, a widely used type of semiconductor for constructing integrated circuits.

just enough performance to process the system workload while meeting throughput constraints, and thereby, reduce the energy dissipation [63]. The technique is able to reduce the dynamic power consumption of a CMOS integrated circuit by reducing the voltage and the clock frequency at which it operates.

The relationship between the power $P(t)$ (Watts or Joules/s) and the energy $E(\Delta t)$ (Joules) consumed by an electrical system over a time period $\Delta t$ is given by equation 2.5.

$$E(\Delta t) = \int_0^{\Delta t} P(t) \tag{2.5}$$

In a computing system, the time factor is the execution time, which is inversely proportional to the frequency that the CPU operates. On the other hand, the power $P$ in a CMOS-based circuit is directly proportional to the frequency and the square of the voltage that the system operates (2.6), and therefore, so as the energy(2.7) [89].

$$P \propto f.V^2 \tag{2.6}$$

$$E \propto V^2 \tag{2.7}$$

Voltage and Frequency are determined together based on system requirements. The voltage can be reduced if the frequency is also reduced. This can yield a significant power saving because of the aforementioned $V^2$ relationship [63]. Therefore, DVFS techniques are able to save power and energy of a CMOS integrated circuit by reducing the frequency and/or voltage at which it operates.

One of the main metrics to evaluate the trade-off between energy consumption and the system performance obtained by power saving techniques is the *energy delay product* (EDP), which was initially proposed by Horowitz [55] and has been largely applied since then [25]. The Energy curve is produced similar to the runtime curve using measured CPU energy instead of measured runtime. The EDP is described by the equation 2.8, where $E$ is the energy consumed, $T$ is the execution time and $w = 1, 2$ or 3. The value of $w$ represents how the metric might differ depending on the weight given to time or performance.

$$EDP = E.T^w \tag{2.8}$$

In the next section we describe several DVFS approaches implemented and evaluated in the last few years.

## 2.4 DVFS Approaches

Although relatively recent, DVFS literature is rich and diversified. This section describes relevant applications of DVFS, algorithms and tools that implement them.

In general, most DVFS algorithms work predicting future processing demands, usually from observed past behavior, and use that information to determine the appropriate processor speed and the corresponding frequency and voltage.

Kong et al. [61] propose an energy management technique to explore the trade-off between energy/power consumption and performance. They use a DVFS algorithm which manages energy and power consumption adaptively. Their scheme is based on a previous profiling: they calculate the EDP metric for each application in the workload, in order to characterize the best power-state for each application. The EDP metric is based on the energy consumption (Joules), CPI (clock cycle per instruction) and cycle time.

All information is stored in two tables, such that one is stored in a volatile memory and the other is saved in a non-volatile memory for future use. Next, they consult one of these tables *on-the-fly* to choose the adequate power state for each application. The great advantage of this technique is that considering both energy and delay of the applications, it reduces EDP significantly and effectively.

Although DVFS techniques promise to reduce power consumption while leading to significant reduction in the energy required for a computation, recent developments in processor and memory technology have resulted in the saturation of processor clock frequencies and larger static power consumption, which limit the potential energy savings resulting from DVFS.

Le Sueur et al. [63] exam the potential of DVFS across three platforms with recent generations of AMD processors and they find that while DVFS is effective on the older platforms, it actually increases energy usage on the most recent platform, even for highly memory-bound workloads. They justify this argument because when the first DVFS proposal was published by Weiser [95] in 1994, transistor feature sizes were approximately $0.8\ \mu m$ and typical core voltages were $5V$. Furthermore, the ratio of dynamic power (which DVFS can reduce) to static leakage power was high. Therefore, energy savings resulting from DVFS could be significant.

However, modern CPUs have transistors with feature sizes smaller than $32\eta m$ and core voltages at the highest frequency are around $1V$. According to Le Sueur, "*The small feature sizes result in leakage power reaching or exceeding dynamic power, and the low core voltages reduce the voltage-scaling window (which is limited by the 0.7 V threshold voltage of silicon transistors). Therefore, the potential to save energy via DVFS is dramatically reduced*". This result is strong but limited, since their analysis is simplified by only considering a single memory-bound benchmark, and server-class platforms, which are not very representative for embedded, for example. In their point of view, the industry must adopt ultra-low-power sleep modes to save energy.

Also in this context, Castagnetti et al. [30] discuss that the relation $E \propto P$ that relates energy and power is simplistic and not applicable for complete system designs. So, they propose a power and energy model for a DVFS enabled mobile computing platform, and the results show that the CPU energy saving is far less than when using a model that does not take into account the effect of the static power.

Notwithstanding to all those results about the effect of the static power in modern CPUs, a lot of research has been published proposing DVFS techniques that obtained gains when focused on different approaches. Genser et al. [49] have concentrated on voltage regulators to propose a new DVFS hardware extension to a power emulation approach for modeling the voltage regulator behavior, which allow for performance, power and energy efficiency investigations for embedded.

Lu, Lai and Huang [70] examine many previous parallel processing architetures and DVFS mechanisms. They propose two different orientations of parallel DVFS-enabled H.264/AVC decoders, and implement a multimedia heterogeneous multi-core platform. Halimi et al. [54] propose FoREST, a new runtime DVFS controller able to estimate the energy savings it can achieve from power gains. It determines the potential energy gains from two phases: an offline phase exploiting energy probes embedded in processors, and runtime speedup measurements for the most interesting frequencies.

There is a large amount of work for energy efficient communication via different DVFS scheduling algorithms. Kappiah et al. [59] devised a system that exploits slack arising at synchronization points of MPI programs by reducing inter-node energy gear via DVFS. Li et al. [66] proposed to characterize energy saving opportunities in executions of hybrid MPI/OpenMP applications without performance loss. Predictive models and novel algorithms were presented via statistical analysis of power and time requirements under different configurations. Next, Li et al. [92] extend theses previous works proposing an adaptively DVFS scheduling strategy to achieve energy efficiency for data intensive applications, and further save energy via speculation to mitigate DVFS overhead for imbalanced branches; their A2E scheduling method adaptively schedules an appropriate CPU frequency for the hybrid *energy saving block*, which is defined as a statement block of one specific type of workload such as computation, communication, memory access, etc, where runtime energy savings may be achieved by different means.

Yadav et al. [97] come up with LAURA-NoC, a NoC with distributed approach to dynamic DVFS, in which very simple local DVFS controllers automatically determine the appropriate clock frequency and voltage, eliminating the need for a global controller. 2 voltages and 16 frequencies values are available in each switch.

In Chapter 5 we describe and evaluate three different DVFS techniques in the MP-SoCBench (DVFS-SW, DVFS-ES, DVFS-CPU) confirming the argument that the tool has sufficient infrastructure to explore the trade-off between performance and energy constraints. The DVFS-SW perform frequency and voltage scaling managed by software; DVFS-ES and DVFS-CPU are frequency and voltage auto-selection techniques performed by each core of the platform: the first of them uses the energy consumption as the main metric and the second one uses the CPU workload to choose the best values of frequency and voltage dynamically.

## 2.5 DVFS Simulators

By improving simulators with some DVFS support, it is possible to carefully plan the DVFS mechanisms at design time, optimizing system thermal profile, preventing runtime emergencies, and controlling the trade-off between system performance and power consumption. Although there are several performance simulators, few of them have power modeling and power control features. We describe in this section some relevant tools with DVFS support.

One of the first tools including power features was presented by Brooks [24]. The **Wattch** deal with power-related issues and it was successfully used as an extension for several performance simulators, such as the SimpleScalar simulator.

**McPAT** (*Multicore Power, Area, and Timing*) [67] is a tool for modeling and estimation of physical parameters (power, area, clock frequency) for multiprocessor systems. Its main focus is to model accurately, based on performance parameters and manufacturing technology information, a set of physical parameters, providing real statistics that can be used to support design decisions of MPSoCs.

In combination with performance simulators, McPAT uses input parameters from an XML-based interface that defines the target system design: cache levels, cores, pipelines per core, cores homogeneity/heterogeneity and other parameters. The McPAT output shows results, organized by components (CPU, cache, arithmetic units, buses, and so forth), and for each one of them, it shows the area and power consumed according to the manufacturing technology and clock frequency. McPAT is not a simulator itself and produces estimation data that, although accurate, does not reflect the dynamic behavior of the applications.

One of the aforementioned integration between performance and power tools was obtained joining gem5 and McPAT to get system power estimation. Performance statistics are collected from gem5 and fed through McPAT to provide the final estimation. Voltage and frequency are some of the design parameters expected by the McPAT. So, the designer is not allowed to change these values unless he starts a new system design; this is an important limitation of this infrastructure. Furthermore, the interface of McPAT is hard to feed, due to the great amount of architecture and design parameters, which can not always be achieved.

To improve the aforementioned limitations, Spiliopoulos et al. [90] propose to make the Gem5 suitable for full-systems DVFS studies, including clock and voltage domain declaration, online power-estimation, a DVFS controller, and libraries for DVFS support. As limitations, this infrastructure does not have yet idle-power management or power/thermal monitoring sensors.

Created in the context of joining performance simulators and thermal analysis, the **SST** framework [56] is useful for architecture-level power, area, and thermal simulation, which focus on large-scale systems, but application traces are emulated, rather than collected from cycle-accurate simulation, with higher simulation rate at the cost of lower accuracy. In the same context, Zoni, Corbetta and Fornaciari [100] introduce **HANDS**, a novel framework for joint thermal, performance and power analysis to be used both at early design stages, while the extracted information can be used for further localized platform optimizations and trade-off exploration.

The MPSoCBench already has both power and DVFS features based on high level power estimation, which are described in Chapters 3 (power models) and 5 (DVFS features).

## 2.6 Design Space Exploration Tools (DSE Tools)

It is well known that MPSoCs design complexity induces the use of automatic design space exploration (DSE) tools to explore design alternatives before real implementation.

There are several proposals to characterize a system with multiple processing cores on a single chip. Proposals range from specification models [81], up to the development of tools and frameworks to offer more support to the design space exploration at different abstraction levels [6,34,46,86]; we will describe several of them in the following paragraphs.

Angioline et al. [10] propose a methodology to integrate pre-existing standalone CAD tools in a complete virtual platform, therefore paving the way for faster and more thorough analysis of the available architectural choices. They explored alternative ways to implement such an integration, defining two wrapping policies to give different emphasis to the cache design. Subsequently, they applied the methodology to state-of-the-art CAD tools, such as the commercial LISA [31] suite and the academic MPARM [16] environment.

Maintaining the accuracy by low-level simulation-based tools together with the performance in execution time of analytical-based tools was the goal achieved by Johann Filho et al. in [46]. In this sense, they propose a platform, a design flow and a tool for execution time and energy consumption estimation of homogeneous MPSoCs. The platform is composed of four processors interconnected by a bus system and described in VHDL used as a case study for estimation tool evaluation. After synthesizing the VHDL high-level description of the platform into a gate-level (RTL), time execution and energy consumption estimations are taken from RTL descriptions and organized in a high-level tool, to speedup simulation times.

The focus of the work in [86] is on the Transaction Accurate (TA) level which uses Transaction Level Modeling (TLM) for MPSoC architecture to speed up simulation at the cost of less accurate performance. Instead of using traditional Instruction Set Simulators (ISS), software tasks are annotated with timing information and executed locally with much higher speed. The annotated execution time for each software task is based on statistical processor properties.

The **Sniper** [29] simulator has been used to estimate application performance by employing an analytical model to raise the abstraction level. A similar proposal adopts the M5 simulator [34] to address the trade-off between power and performance by constructing an analytical model based on an extension of Pollack's Rule [20] and Amdahl's Law [9].

In 2011, the **Multicube** was proposed by Silvano et al. [88]. It is an open-source framework for multiprocessors design space exploration. The goal is to drive an architecture designer towards near-optimal parameters, using a set of strategies (multi-objective model formulation and heuristics) to model and solve the problem of architectural exploration.

Petry et al. [81] approach the use of a set of models that describe a whole MPSoC at several abstraction levels. Models of processors, memories, network interfaces, NoCs and software applications can be combined using an integration framework, which is capable of generating a complete executable MPSoC model. Design elaboration, simulation of real

and synthetic applications and debugging can be accessed directly from the integrating environment, in most cases.

We have an ongoing project which includes mechanisms to use a performance-based simulator like MPSoCBench with a low-level power and an area estimation tool like McPAT for design space exploration. We've apply this mechanism in a tool named **Multiexplorer**. Although this tool has been improved to cover other issues (dark silicon, for instance), we published the first version of Multiexplorer in [33]. We describe briefly this project in the conclusion (Chapter 6).

## 2.7 Benchmarks

On the software scenario, there are also benchmarks that we adapted to run on the MPSoCBench simulation infrastructure. The SPLASH-2 [96] and ParMiBench [58] benchmarks have a set of parallel software using POSIX Threads, able to use up to 16 threads. We also joined several Mibench [53] single-threaded benchmarks to create multi-threaded benchmarks that do not share data resources, and we included them in the MPSoCBench software repositories. All applications adapted for the MPSoCBench are detailed in Section 3.2

The MPSoCBench tool fills the gap of a virtual prototyping tool integrating both scalable and configurable hardware and software useful for design and evaluation in the MPSoC modeling and simulation scenario.

## 2.8 Characterizing Simulation and DSE Tools

System simulators have fundamental differences, like the features they contain, detailing level, accuracy, and especially the main purpose of their use. Although this large number of parameters makes difficult to compare them, we present in Table 2.1, in chronological order, several aforementioned simulators and show how they compare with the MPSoCBench, according with requisites like simulation speed, abstraction level, power evaluation and DVFS support. We could not find the simulation speed for all frameworks, so we used a "-" in several of them. Similarly, Table 2.2 shows design space exploration tools and compare them with the MultiExplorer tool.

Table 2.1: Comparison among Simulators and the MPSoCBench. *HL=High level, CA=Cycle accurate*

| **Tool** | **Year** | **Abstr. Level** | **Features** | **Multicore Support** | **Speed** | **Power Support** | **DVFS Support** |
|---|---|---|---|---|---|---|---|
| Bochs [62] | 1996 | HL | Emulation | No | - | No | No |
| Simple-Scalar [12] | 2002 | CA | Emulation | Yes | ≈150 K instr./sec | No | No |
| Simics [71] | 2002 | HL | Simulation (x86-to-x86) | No | ≈7.5 K instr./sec | No No | No No |
| MP-ARM [16] | 2004 | HL | Simulation (ARM) | Yes | 60 K cycles/sec | Yes | Yes |
| SESC [78] | 2004 | HL | Emulation | No | - | No | No |
| GEMS [73] | 2005 | HL | Simulation | Yes | Simics | No | No |
| QEMU [14] | 2005 | HL | Dyn. transl. | No | - | No | No |
| M5 [18] | 2006 | HL | Emulation | 4-core | - | No | No |
| Multi2Sim [94] | 2007 | HL | Simulation | No | - | No | No |
| PTLSim [98] | 2007 | CA | Sim. and VM | No | - | No | No |
| OVP [57] | 2008 | HL | Simulation | Yes | - | Yes | No |
| StarSoc [23] | 2007 | CA | Simulation | Yes | - | No | No |
| MPTLSim [99] | 2009 | CA | Simulation | Yes 16-core | 200-300 cycles/sec | No | No |
| gem5 [17] | 2011 | HL | Emulation | Yes | - | No | No |
| ESESC [11] | 2013 | HL | Emulation | Yes | - | Yes | No |
| MPSoCBench [39] | 2014 | HL | Simulation | Yes | ≈1.5 M instr./sec | Yes | Yes |

Table 2.2: Comparison among Design Space Exploration Tools and the MPSoCBench.

| **Tool** | **Year** | **Features** | **Multicore Support** | **Simulation/Estimation** |
|---|---|---|---|---|
| Wattch [24] | 2000 | Power | Yes | Accurate Estimation (do not simulate) |
| McPAT [67] | 2009 | Power, area and timing | Yes | Accurate Estimation (do not simulate) |
| SST [56] | 2011 | Power, area and thermal | Yes | Emulation with low accuracy |
| Sniper [29] | 2011 | Performance Simulator based on analytical models | No | Simulation and Estimation |
| HANDS [100] | 2012 | Performance, power and thermal | No | Accurate Estimation (do not simulate) |
| MPSoCBench (MultiExplorer [33]) | 2014 | Performance, power, area | Yes | Simulation |

# Chapter 3

# MPSoCBench

In this chapter we will introduce the MPSoCBench, a simulation toolset composed of a scalable set of MPSoCs to enable development and evaluation of new tools, methodologies, parallel software, and hardware components.

In the context of this work, a ***virtual platform*** is a fully functional model of a complete system, containing high level models for each component. Each platform is a scalable set from 1 to 64 processors with data and instructions caches, integrated with a shared memory and several IPs using interconnection mechanisms. We considered 64 as a good limit although this configuration could be easily increased. The MPSoCBench is able to create hundreds of different MPSoC configurations automated through scripts, and the total number of different virtual platforms can be even larger if we consider a range of caches, and frequencies/voltage configurations.

Figure 3.1 illustrates a virtual platform that can be built with the MPSoCBench, using a mesh based NoC as interconnection; this platform has a set of nodes, each one acting as a router, to connect processors, memory, and IPS like DVFS, interrupt controller, and a hardware lock to concurrency management. All available components will be described in this chapter.

This chapter is organized as follows: Section 3.1 describes the main hardware models and software components available in the toolset; Section 3.2 shows the applications adapted to run on the MPSoCBench virtual platforms; Section 3.3 presents the timing model adopted for the processor and the communication devices; Section 3.4 describes the methodology used for power estimation and energy consumption; Section 3.5 briefly describes the MPSoC building and simulation process, and finally, Section 3.6 concludes the chapter.

## 3.1 Components

This section describes the hardware models and the software applications in the MPSoCBench.

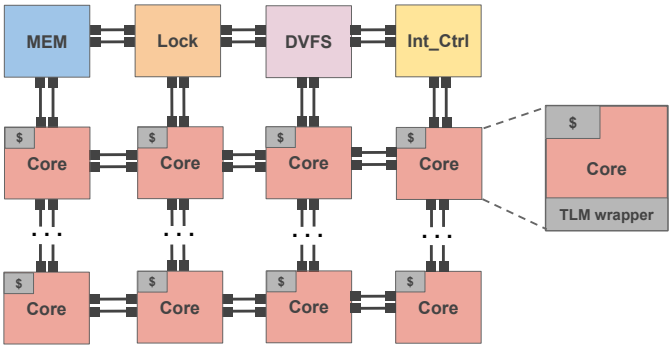



Figure 3.1: A platform designed as a mesh consisting of four processors, memory and IPs

### 3.1.1 Processor Models

MPSoCBench contains four ArchC behavioral processor models: **ARM** is the 32-bit ARMv5e instruction set; **MIPS** is a 32-bit RISC MIPS-I instruction set; **SPARC** is a V8 version of the 32-bit SPARC architecture; **PowerPC** is a 32-bit version of the PowerPC instruction set. All models include operating system call emulation.

The processor models are described using the ArchC Architectural Description Language and they are available in the ArchC repositories. The MPSoCBench installation script only redirect to the official download links.

An architectural description is divided into two parts: Architecture Resources and the instruction set (**AC_ARCH** and **AC_ISA**, respectively). In the **AC_ARCH** the designer tells the storage resource, *pipeline*, and other features. In the **AC_ISA**, the designer must describe the format and behavior of each instruction. Using this information, the ArchC *acsim* tool can generate a processor simulator.

Figure 3.2 shows the ARM architecture description using the ArchC syntax. In this code, we define a memory system including a data cache, an instruction cache and an interface for a 512MB external memory (lines 2,3,4), an interruption port (line 5), register bank and special registers (lines 6 to 11), wordsize and fetchsize of 32 bits (lines 12,13), the external file where the instruction behavior are implemented (line 15), processor endianess (line 16), and finally, how the memory and caches are connected (lines 17,18).

The instruction set description provides all information to automatically create a *decoder*. It is divided into two files, one with instructions and their formats and another with the behavior.

We show in Figure 3.3 a snippet of the file `arm_isa.ac` that contains the instruction formats, for instance: the format of the data processing instructions, branches, and memory access instructions. The behavior of each instruction is described directly in C++ in the `arm_isa.cpp` file. Figure 3.4 shows part of the MUL instruction implementation in the ARM model. Two operands are read from the register bank at lines 6 and 7, and they are multiplied in line 12. At lines 14 to 18 the flags are updated according to the result of the operation; at line 19 the result of the operation is stored in the target register, and finally at line 20 the program counter is updated.

```
AC_ARCH(arm){
  ac_tlm2_port      MEM:512M;
  ac_icache    IC("2w", 128, 32, "wb", "fifo");
  ac_dcache    DC("2w", 512, 32, "wt", "fifo");
  ac_tlm2_intr_port intr_port;
  ac_regbank RB:16;
  ac_reg R14_irq, R14_fiq, R14_svc, R14_abt, R14_und, R13_irq, R13_svc;
  ac_reg R13_abt, R13_und, R13_fiq;
  ac_reg SPSR_irq, SPSR_fiq, SPSR_svc, SPSR_abt, SPSR_und;
  // FIQ private regs
  ac_reg R12_fiq, R11_fiq, R10_fiq, R9_fiq, R8_fiq;
  ac_wordsize 32;
  ac_fetchsize 32;
  ARCH_CTOR(arm) {
    ac_isa("arm_isa.ac");
    set_endian("little");
    IC.bindTo(MEM);
    DC.bindTo(MEM);
  };
};
```

Figure 3.2: ARM architecture description using the ArchC syntax

```
AC_ISA(arm) {

  /* Data processing instructions - ALU */
  ac_format Type_DPI1 = "%cond:4 %op:3 %func1:4 %s:1 %rn:4 %rd:4 %
      shiftamount:5 %shift:2 %subop1:1 %rm:4";
  ac_format Type_DPI2 = "%cond:4 %op:3 %func1:4 %s:1 %rn:4 %rd:4 %rs:4 %
      subop2:1 %shift:2 %subop1:1 %rm:4";
  ac_format Type_DPI3 = "%cond:4 %op:3 %func1:4 %s:1 %rn:4 %rd:4 %rotate:4
      %imm8:8";

  /* Branch instructions */
  ac_format Type_BBL = "%cond:4 %op:3 %h:1 %offset:24";
  ac_format Type_BBLT = "%cond:4 %op:3 %h:1 %offset:24";
  ac_format Type_MBXBLX = "%cond:4 %op:3 %func1:4 %s:1 %one1:4 %one2:4 %
      one3:4 %subop2:1 %func2:2 %subop1:1 %rm:4";

  /* Load/Store */
  ac_format Type_LSI = "%cond:4 %op:3 %p:1 %u:1 %b:1 %w:1 %l:1 %rn:4 %rd:4
      %imm12:12";
  ac_format Type_LSR = "%cond:4 %op:3 %p:1 %u:1 %b:1 %w:1 %l:1 %rn:4 %rd:4
      %shiftamount:5 %shift:2 %subop1:1 %rm:4";
  ac_format Type_LSE = "%cond:4 %op:3 %p:1 %u:1 %i:1 %w:1 %l:1 %rn:4 %rd:4
      %addr1:4 %subop2:1 %ss:1 %hh:1 %subop1:1 %addr2:4";
  ac_format Type_LSM = "%cond:4 %op:3 %p:1 %u:1 %r:1 %w:1 %l:1 %rn:4 %rlist
      :16";
  ...
```

Figure 3.3: ARM instruction set description using the ArchC sintax

```
1  inline void MUL(arm_isa* ref, int rd, int rm, int rs, bool s,
2            ac_regbank<16, arm_parms::ac_word, arm_parms::ac_Dword>& RB,
3            ac_reg<unsigned>& ac_pc) {
4
5    arm_isa::reg_t RD2, RM2, RS2;
6    RM2.entire = RB_read(rm);
7    RS2.entire = RB_read(rs);
8
9    dprintf("Instruction: MUL\n");
10   dprintf("Operands:\n  A = 0x%lX\n  B = 0x%lX\n", RM2.entire, RS2.entire);
11
12   RD2.entire = RM2.entire * RS2.entire;
13
14   if(s == 1) {
15     ref->flags.N = getBit(RD2.entire, 31);
16     ref->flags.Z = ((RD2.entire == 0) ? true : false);
17     // nothing happens with ref->flags.C and ref->flags.V
18   }
19   RB_write(rd, RD2.entire);
20   ac_pc = RB_read(PC);
21 }
```

Figure 3.4: Example of an MUL instruction behavior

The ArchC environment enable generation of assemblers, linkers, debbugers, among other tools.

This thesis is also responsible for some contributions to the ArchC environment, most of them related to the multicore support. We can list here most important of them:

- We added a mechanism for identifying a processor within a platform;
- We introduced support to TLM2 interfaces and the TLM2 base protocol;
- We introduced support to TLM2 loosely-timed and approximately-timed coding style;
- We incorporate the power support to ArchC source code;
- We extended the models with timing based on real implementations of them.

**Timing Model and Frequency Domain:** While functional simulators emulate the behavior of the target system without the need to provide accurate time, timing simulation is used to assess the system performance, and acts as an approximation to the real behavior. Time modeling is needed to measure the fidelity of these simulators with respect to existing systems [65].

In order to introduce a timing model in these processors, we researched the number of cycles required to execute each instruction of the ISA in the following real processors: ARM9E-S [1], MIPS32 M5150 Processor Core Family [69], PowerPC 405TM Core [32], and UT699 LEON 3FT/SPARCTM V8 MicroProcessor [48].

The user can assign the processor frequency using the `set_proc_freq` method. Each object that represents an ISA instruction has a method named `setCycles` that allows

us to enter the number of cycles required to execute instructions in the ISA. These two methods and other methods correlated to them were developed and validated during this project in collaboration with the ArchC group.

The time information is calculated based on the number of cycles and frequency and passed as an argument to the `wait` SystemC function, responsible for the synchronization and advancing the simulation time.

### 3.1.2 Memory

Although memory is becoming more and more of a bottleneck, most CPU simulators have an extremely simple main memory model. In most cases, the memory model is a simple fixed latency, which is problematic because the dynamics of the CPU and memory are highly intertwined. A fixed latency is not realistic, since the complexity of the memory system causes highly variable latencies. On the other hand, including a DRAM accurate simulator in a full system simulator can be costly, and depending of the simulation abstraction level and target demand, it may not be worth it [84].

MPSoCBench uses two approaches. The simple one is an external shared memory pre-configured to 512MB, which is enough to run the biggest configuration available, considering the maximum number of cores and the inputs provided. MPSoCBench allows designers to choose among blocking and non-blocking TLM2 channels to connect the memory to the network.

The second approach aims to provide an accurate memory model, we integrate the MPSoCBench external memory with the DRAMSim2 simulator [84], which is a cycle accurate model of DRAM memory, able to explore memory performance and energy consumption. The use of DRAMSim2 is optional and can be switched easily.

DRAMSim2 can be used in both full systems and trace-based simulations. It provides a repository with multiple configuration files based on reports from memory manufacturers, and even allows new configuration files modifying the voltage, frequency and other parameters related to memory architecture.

Figure 3.5 shows the high level structure of the memory simulator. Each DRAM device is represented as a channel in this abstraction level. These channels are described by instances of the MemorySystem class and contain how many *ranks* it supports, which are also configurable by the user. Each rank contains several memory banks.

The simulator allows evaluation of memory statistics during the simulation and also reports, at the end of execution, a summary containing the bandwidth, latency and power information. Figure 3.6 shows an example of an DRAMSim2 report.

### 3.1.3 Caches

We used the ArchC cache model described in [8]. The cache model has the following configurable parameters that need to be specified in the constructor:

**Block Size**: is the number of units of data in a single cache block. An unit of data is defined previously in the *wordsize* processor parameter. The block size needs to be a power of 2.

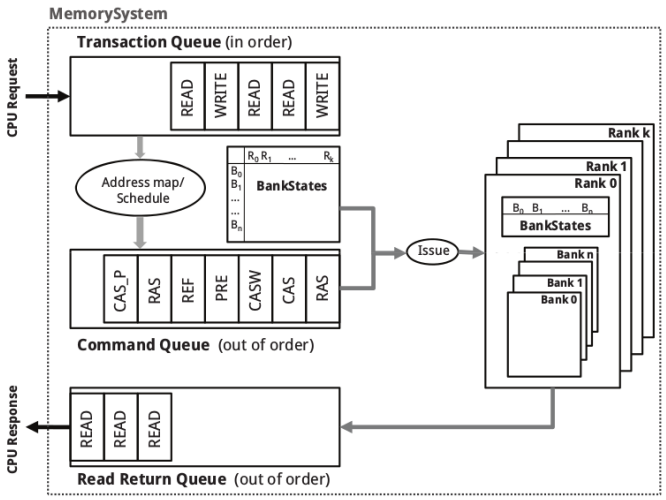



Figure 3.5: View of the components of the DRAM2Sim

```
**************************************************************************
Platform ./platform.router.lt.x with 8 cores running dijkstra.mips.x

Total Time Taken (seconds):     10.974883
Simulation advance (seconds):   0.007739
Lock Access:    11858
Router Access:  4746933
Memory Reads:   4045819
Memory Writes:  689255
Reads and writes per core:
Total reads     Total writes
1: 586052       106026
2: 503987       87488
3: 486510       81719
4: 487853       68335
5: 505233       104075
6: 496535       79090
7: 485153       71190
8: 494364       91302
********************************** Memory report ***********************************
==== Channel [0] ====
 ========================================================
 ============== Printing Statistics [id:0]==============
   Total Return Transactions : 1518887 (48604384 bytes) aggregate average bandwidth 2.796641 GB/s
      -Rank   0:
        -Reads  : 8635 (276320 bytes)
        -Writes : 1239 (39648 bytes)
        -Bandwidth / Latency  (Bank 0): 0.185517 GB/s          409.780701 ns
        -Bandwidth / Latency  (Bank 1): 0.116692 GB/s          384.398987 ns
        -Bandwidth / Latency  (Bank 2): 0.110178 GB/s          364.726257 ns
        -Bandwidth / Latency  (Bank 3): 0.109611 GB/s          345.527954 ns
        -Bandwidth / Latency  (Bank 4): 0.783139 GB/s          683.466309 ns
        -Bandwidth / Latency  (Bank 5): 0.294279 GB/s          468.688416 ns
        -Bandwidth / Latency  (Bank 6): 0.084687 GB/s          360.197815 ns
        -Bandwidth / Latency  (Bank 7): 1.112538 GB/s          519.525635 ns
 == Power Data for Rank 0
   Average Power (watts)     : 2.307477
     -Background (watts)     : 0.778501
     -Act/Pre    (watts)     : 0.840746
     -Burst      (watts)     : 0.648686
     -Refresh    (watts)     : 0.039545
//// Channel [0] ////
```

Figure 3.6: An example of the DRAMSim2 report already integrated with the MPSoCBench

**Number of blocks**: integer that represents the number of cache blocks. The number of blocks also needs to be a power of 2.

**Associativity**: The number of blocks in a set cannot be larger than the number of blocks. If the associativity is equal to the number of blocks, the cache is said to be fully associative. If associativity is equal to 1, the cache is said to be direct mapped (default).

**Write Policy**: In the *Write Through* policy, when a cache-write is executed, the block containing the data is immediately written to memory; in the *Write Back* policy, when a cache-write is executed, the block containing the data will be written only when there is a need to replace it in the cache.

**Replacement Policy:** Once it is necessary to replace a cache block, a replacement policy is applied for selecting which block should be replaced. Three replacement policies were implemented: Random, FIFO (First In, First Out), and LRU (Least Recently Used).

The code in Figure 3.2 (line 2) contains an example of an external memory instantiation with 512MB, data and instruction caches (lines 3,4) and the connection between them in a 32-bit ARM processor (lines 17,18). The instruction cache (IC) is a 2-way set associative write-back cache, with 128 blocks, 32 words per block, using *fifo* replacement policy. The data cache (DC) is a 2-way set associative write-through cache, 512 blocks with 32 words each and *fifo* replacement policy.

**Cache Coherence:** The original ArchC cache implementation was done focusing on standalone simulators; hence, it did not have any cache coherency mechanism. As we are interested in evaluation of multicore simulators and parallel applications, we implemented a simple mechanism for cache coherence.

We connect the last level cache of each core with shared *directory*, which is a device that implements the MSI cache-coherence algorithm, enabling the parallel computing with shared resources.

Currently, the cache coherence protocol is implemented only for write-through caches, because it is necessary a robust mechanism for inter-cache communication to implement the same protocol for write-back caches or more sophisticated protocols form both caches.

### 3.1.4 Interconnections

MPSoCBench contains three interconnection networks: a higher level crossbar to make it easy to check software functionality, and two TLM implementations of a Network on Chip (NoC) modeled after the Hermes RTL NoC [76]: one using loosely-time (NoC-LT) and another using approximately-timed (NoC-AT) coding styles. The interconnection networks are briefly described as follows:

**Crossbar:** Implements a set of blocking TLM2 channels connecting every IP in the platform using loosely-timed coding style. We added timing annotation to each trans-

action and updated the simulation time accordingly. This is the simplest interconnection and is also called Router.

**NoC-LT:** Implements a mesh based NoC in a TLM2 loosely timed (LT) abstraction level, using XY routing protocol, configurable in runtime to connect up to 64 cores, memory, lock and wrappers. This NoC contains a set of nodes, each one connected to the IP wrapper. Each NoC node has 4 ports to connect to adjacent nodes (North, East, South and West) and a Local port to connect to the IP wrapper. All nodes also contain packet buffers.

**NoC-AT:** Implements a mesh based NoC in a TLM2 approximately timed (AT) that uses sockets, forward and backward transport interfaces. Each internal router has North, East, South and West `simple_initiator_socket` and `simple_target_socket` objects to connect to other nodes. Each node has also local sockets to connect to the local IP. Each one of the NoC-AT nodes has one `sc_thread`, and uses two-phases non-blocking transport methods.

Both NoCs were totally developed during this project and they are configured at runtime through user parameters. We extended an existing model or Router to our purposes.

Figure 3.7 illustrates the main components used to implement the communication through the NoC-AT. The forward path is illustrated in Figure 3.7(a) and the backward path is detailed in Figure 3.7(b). The communication starts with a request from the initiator `sc_thread`, which creates the generic payload, sends the request using the forward path and calls `wait()` for an answer event. The wrapper receives the request, creates a payload extension with routing information and puts the packet into the buffer of the first router. Then, the packet goes through several nodes (routers) until it reaches the target.

The backward path is similar. The target (in most of cases, the memory) updates the generic payload with the appropriate response, and sends the request using the backward path. The packet goes through several nodes until it reaches the initiator. The initiator backward transport method (`nb_transport_bw()`) notifies the answer event and unlocks the `sc_thread` that completes the transaction.

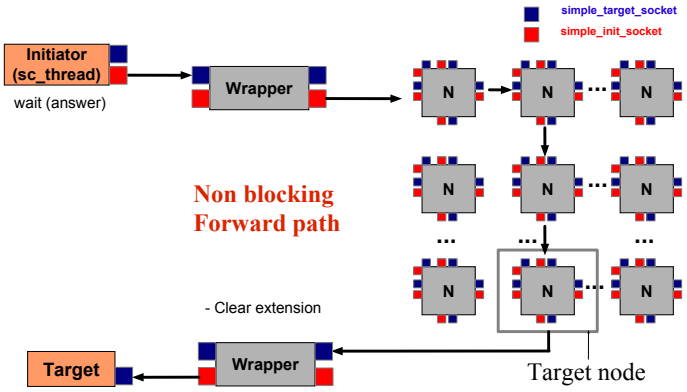
New TLM components can be easily integrated into MPSoCBench by adding all source code into a specific directory.

### 3.1.5 Interrupt Controller

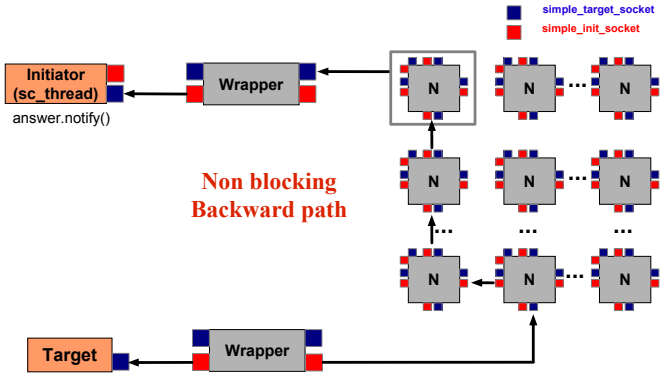
The interrupt controller is connected to the network and the processors interrupt pin. It is in the global system address space, and allows interruption by software or inter-processors interruptions.

At the beginning of the simulation of a multicore platform, only one core starts while the others remain in OFF mode, waiting for an event. This core is called a *master core*. As any core in the platform can actuate as a master core, the SystemC scheduler is the responsible to choose which core will start first.

When the tasks are ready to execute, the master core triggers an event to turn on the other cores. This mechanism is useful to increase the simulator performance, because the




(a) Forwarding path




(b) Backward path

Figure 3.7: Communication through the NoC-AT using Forwarding and Backward Paths

SystemC kernel can remove the thread that represents the core behavior from its queues, reducing the concurrency.

The current ArchC interrupt handling mechanism provides a direct connection between the interrupt controller and processor model interrupt pin. Although we have used this mechanism in MPSoCBench platforms, we agree that this is not an appropriate approach for MPSoCs. So, we added as future work the suitability of the interrupt mechanism to restrictions imposed by multiprocessor systems on chip.

### 3.1.6 DVFS

The global DVFS IP (IP for Dynamic Voltage and Frequency Scaling) is able to manage the voltage and frequency in which each core operates. It is in the system address space and it interacts with the local DFS controllers (per-core) to manage the frequency and power information for each core. The global DFS IP is initialized with the available frequencies for all cores, and it can manage the frequency globally.

The hardware infrastructure to support DFS and the approach adopted in the MPSoCBench are described in the Chapter 5.

### 3.1.7 Lock

We also included a TLM2 IP that acts as a hardware lock, used to implement mutexes, semaphores, conditional variables, and barriers. Before every access to a shared resource, a processor must access the Lock to guarantee atomicity. We consider that the global lock do not compromise the platform scalability because the number of accesses to the IP is too small in contrast with the memory. The next section will describe several functions that use the Lock to implement concurrency management.

## 3.2 Library for PThread Support and Applications

To avoid the burden of porting an operating system, we implemented acPThread, a POSIX PThread emulation library that handles thread management, barriers, mutual exclusions, semaphores, and conditional variables.

Figure 3.8 shows the set of POSIX PThread functions emulated by the acPThread library:

Most parallel programs implemented with PThreads run on MPSoCBench without significant changes. The only modification is the inclusion of a main function to be executed by all platform processors. This function ensures that the application's old main function is executed by one of the processors (the Master), while the others are on standby until the *threads* assigned to them are triggered.

To execute a new PThread-based application in MPSoCBench platforms, it is only necessary to attach the main startup function code as illustrated in Figure 3.9 to the application, adjusting the command line arguments expected by the application, to store all application files and Makefiles in a specific directory inside the platform and compile them with the `acPThread.h` header file instead of the original PThread library (`pthread.h`).

```
1  int pthread_attr_setdetachstate(pthread_attr_t *attr, int
2  detachstate);
3  int pthread_attr_getdetachstate(const pthread_attr_t *attr,
4  int *detachstate);
5  int pthread_attr_init(pthread_attr_t *attr);
6  void pthread_exit(void *);
7  void pthread_exit();
8  void pthread_busywait(int);
9  int pthread_join_init (pthread_join_t *);
10 void pthread_join(pthread_t, const pthread_attr_t *);
11 void pthread_mutex_init(pthread_mutex_t *, const
12 pthread_mutexattr_t *);
13 int pthread_mutex_lock(pthread_mutex_t *);
14 int pthread_mutex_unlock(pthread_mutex_t *);
15 void pthread_mutex_destroy (pthread_mutex_t *m_lock);
16 int pthread_init();
17 int pthread_create(pthread_t *, const pthread_attr_t *,
18                    void (*start_routine)(void *), void *);
19 int pthread_cond_init (pthread_cond_t *, const
20 pthread_condattr_t *);
21 int pthread_cond_wait (pthread_cond_t *,
22 pthread_mutex_t *);
23 int pthread_cond_broadcast(pthread_cond_t *);
24 void pthread_barrier_init (pthread_barrier_t *, int);
25 int pthread_barrier_wait (pthread_barrier_t *);
```

Figure 3.8: The set of PThread functions emulate by the acPThread library

All processors start executing the same code in the main function, where the execution flow is split and the master processor executes the application regular main function (app_main()), while the others are monitoring the thread queue waiting for a task to execute.

The acPThread library can be extended to become a microkernel responsible for tasks such as multithreaded processing and interrupt management, to mitigate the lack of a more robust operating system.

We adapted a total of 17 different applications from well known benchmark suites to be configurable at runtime and execute in a scalable parallel environment.

MPSoCBench has 7 applications from ParMiBench [58] benchmark:

**Basicmath:** For benchmarking mathematical calculation, like cubic function solving, angle conversions from degrees to radians, and integer square root; the parallelization is done by data partitioning.

**Dijkstra:** It calculates the all-pairs shortest paths in a graph represented by an adjacency matrix, using a data decomposition strategy in such a way that one processor handles one vertex to get its single-source shortest paths.

**SHA:** It implements the Secure Hash Algorithm, useful to generate digital signatures used in the secure exchange of cryptographic keys; each processor calculates the digest of a text from a different input file; all processors store the generated data in a unique output file.

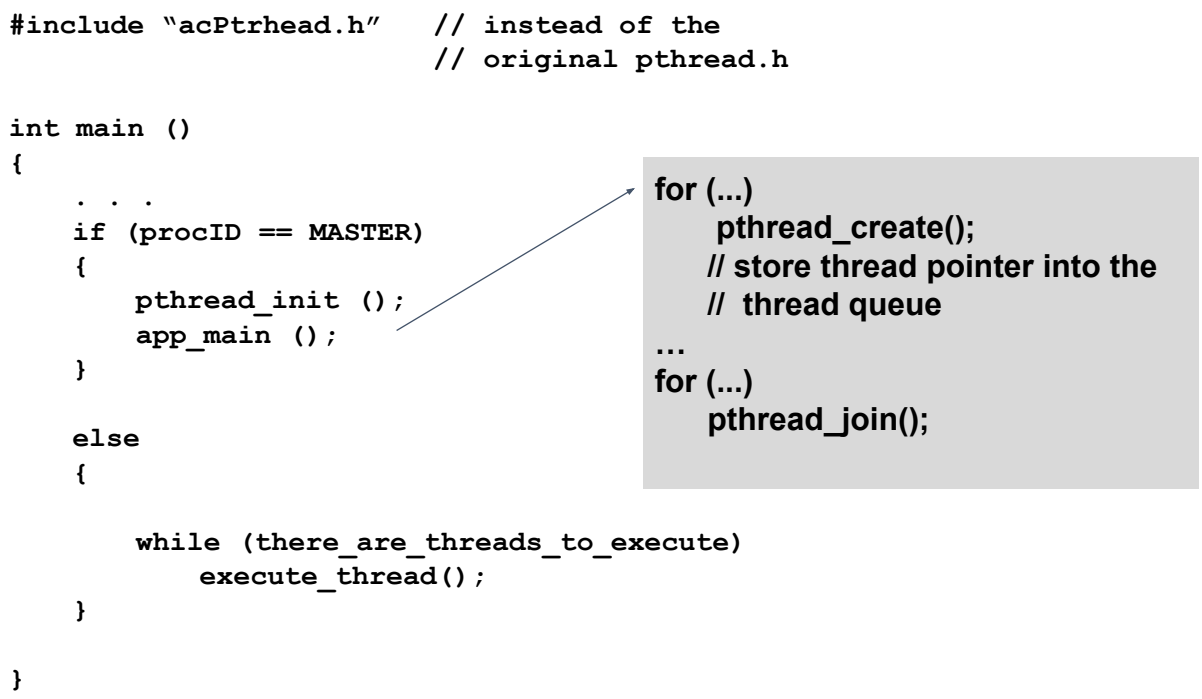



Figure 3.9: Preparing a new parallel application to run in a MPSoCBench simulator

**Stringsearch:** It finds a pattern in a number of given phrases by employing case sensitive or insensitive comparisons algorithms. The strategy is to partition the entire pattern collection into a number of sub-patterns according to the number of workers allocated.

**Susan-corners:** A set of image recognition functionalities, useful for image recognition in Magnetic Resonance Images of the brain; Susan-corners recognizes corners in an input image, it is parallelized by data decomposition and it has the limitation to running on up to 32 cores.

**Susan-edges:** Recognizes edges in an input image, it is parallelized by data decomposition, and it has the limitation to running on up to 32 cores.

**Susan-smoothing:** It performs adjustments for threshold, brightness and image smoothness, it is parallelized by data decomposition, and unlike the others, the original version of susan-smoothing has the limitation to running on up to 8 cores.

**Multi-parallel:** It has four parallel applications from ParMiBench compiled to run in parallel in multi-thread environments (64-core with 16 threads for each application; 32-core with 8 threads for each application and so on). The applications are SHA, Stringsearch, Dijkstra, and Basicmath. This application requires a minimum of 4 cores.

Four applications from SPLASH-2 [96]:

**FFT:** Fast Fourier Transform (FFT); the data set consists of the complex data points to be transformed, and another complex data points referred to as the roots of unity; communication occurs in three matrix transpose steps, which require all-to-all interprocessor communication.

**LU:** It factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The dense $n \times n$ matrix A is divided into arrays of blocks that are allocated locally to processors.

**Water:** It evaluates forces and potentials that occur over time in a system of water molecules. Its original version has the limitation to running on up to 16 cores; however, to run this application in a 16-core platform, it is necessary to update the input size in the source code directly.

**Water-spatial:** It solves the same problem as Water, but uses a more efficient algorithm. Its original version has the limitation to running on up to 16 cores

Five multi-applications software adapted from MiBench [53]:

**Multi-network-automotive:** Four different single-core applications from the Network and Automotive categories, compiled to run on a platform with four processors: Dijkstra, Susan-corners, Basicmath, and Qsort.

**Multi-office-telecomm:** Four different single-core applications from the Office and Telecomm categories compiled to run on a platform with four processors: Stringsearch-PBM, Stringsearch-BMH, ADPCM, and FFT.

**Multi-security:** Four different single-core applications from the Security category compiled to run on a platform with four processors: SHA, Rijndael-encoder, Rijndael-decoder, and Blowfish.

**Multi-8:** Eight applications compiled to run on 8-core platforms. The applications are SHA, Rijndael-encoder, Rijndael-decoder, Blowfish, Stringsearch-PBM, Stringsearch-BMH, ADPCM, and FFT.

**Multi-16:** Sixteen different single-core applications compiled on 16-core platforms: Adpcm-decoder, Adpcm-encoder, Basicmath-angle, Basicmath-Cubic, Basicmath-Sqrt, Bit-Count, Blowfish-decoder, Blowfish-encoder, Dijkstra, FFT, Rijndael-decoder, Rijndael-encoder, SHA, Stringsearch-BMH, Stringsearch-PBM, and Susan-corners.

All MPSoCBench applications can be divided into two groups: the parallel applications in the **Cooperative** group explore cooperative parallelism, solving tasks that can be broken down into subtasks able to run in a multicore cooperative environment; therefore, we may expect a strong concurrency in applications inside this group. The software based on cooperative parallelism are: Basicmath, Dijkstra, SHA, Stringsearch, Susancorners, Susanedges, Susansmoothing, FFT, LU, Water, Water-spatial, and Multi-parallel. Inside

the **Multi** group, we have adapted several different benchmarks that run independently, with no shared resources, which are: Multi-network-automotive, Multi-office-telecomm, Multi-security, Multi-8, and Multi-16.

We released MPSoCBench as a full set of platforms and software and encourage users to enhance and increase it as required. New software design methodologies could be applied such as Transactional Memories, Scratchpad Memories placement, and Real Time software.

## 3.3 Timing Model for Communication

The MPSoCBench system simulators follow two TLM2 timing models depending on the choice of the interconnection mechanism. The Cross-bar and the NoC-LT are consistent with the TLM2 Loosely-Timed coding style and the NoC-AT conform the Approximately-Timed coding style.

**Loosely-timed (LT):** The loosely-timed coding style uses the TLM2 blocking transport interface, which allows only two timing points to be associated with each transaction, corresponding to the call to and to the return from the blocking transport function. The first timing point marks the beginning of the request, and the second marks the beginning of the response. These two timing points could occur at the same simulation time or at different times [2].

The loosely-timed coding style is appropriate for software development using a virtual platform model of an MPSoC, where the accuracy of communication timing is not a crucial issue.

The loosely-timed coding style also supports temporal decoupling, where individual SystemC processes are allowed to run ahead in a local simulation cycle without actually advancing simulation time until they reach the point when they need to synchronize with the rest of the system. Temporal decoupling can result in very fast simulation for certain systems because it increases the data and code locality and reduces the scheduling overhead of the simulator. Each process is allowed to run for a certain time slice or quantum before switching to the next, or instead may yield control when it reaches an explicit synchronization point.

The ArchC uses the TLM2 quantum-keeper [2] mechanism to construct simulators with temporal decoupling. The user can set the amount of time that the the processor simulator can execute without synchronize with the other components (quantum). Once it reached the synchronization point, the *wait()* function is called, returning the simulation control to the SystemC kernel that suspends the simulator, allowing to simulate the next component. Figure 3.10 shows an example of a calling using the blocking transport interface. The Initiator starts a new request at 100ns and passes through the transaction the timing argument at 0ns. The Target responds the request updating the timing parameters, according with its own timing information. The simulation time will be properly updated only at the next synchronization point.

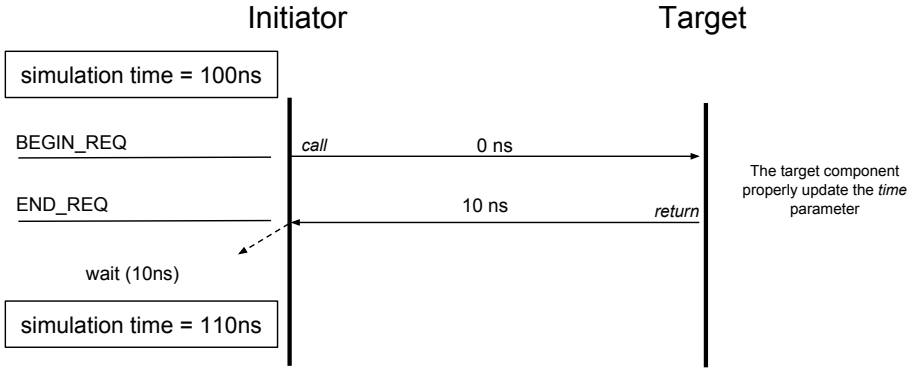



Figure 3.10: An example of calling sequences of the blocking transport

**Approximately-timed (AT):** We used approximately-timed coding style and non-blocking transport interface with timing annotations for each transport phases. The approximately-timed coding style is supported by the non-blocking transport interface, which is appropriate for architectural exploration and performance analysis. The non-blocking transport interface provides opportunities for timing annotation, distinct phases and timing points during the lifetime of a transaction.

In the NoC-AT based protocol, the transaction is broken down into two distinct phases, with an explicit timing point marking the transition between phases. There are timing points marking the beginning and the end of the request and the beginning and the end of the response (BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP). Also, each transport method returns one of these states - TLM_UPDATED, TLM_COMPLETED - which is useful to update the phase properly.

The message sequence chart in Figure 3.11 illustrates various calling sequences to non-blocking transport (`nb_transport_fw` or `nb_transport_bw`, depending on the transaction direction). The arguments and return value passed to and from the non-blocking transport methods are shown using the notation *return, phase, delay*, where return is the value returned from the function call, phase is the value of the phase argument, and delay is the value of the sc_time argument. '-' indicates that the value is unused.

This illustration uses the phases of the base protocol as an example (BEGIN_REQ, END_REQ). With the approximately-timed coding style, a transaction is passed back-and-forth twice between the initiator and target. The recipient immediately calculates the delay to the next timing point, and returns the next state of the transaction.

If the next timing point marks the end of the transaction, the recipient can return either TLM_UPDATED or TLM_COMPLETED. This applies to the initiator and target alike. Because processes are regularly yielding control to the scheduler in order to allow simulation advance, the approximately-timed coding style is expected to have worse performance than the loosely-timed coding style [3].

With TLM_UPDATED, the callee should update the transaction, the phase, and the timing annotation. Figure 3.11 shows the non-zero timing annotation argument passed

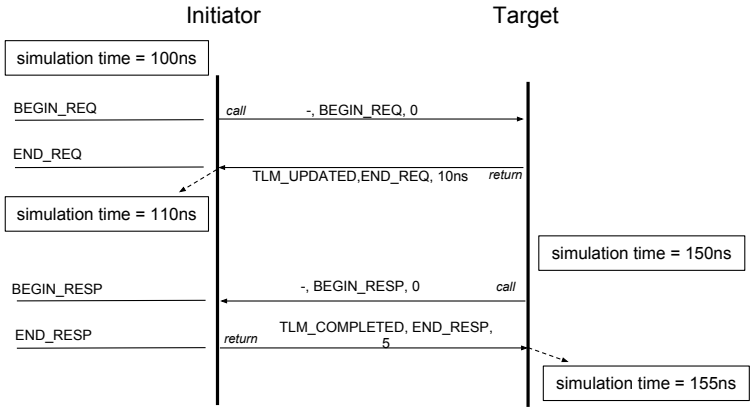



Figure 3.11: An example of calling sequences to non-blocking transport

on return from the function calls indicating to the caller the delay between the phase transition on the hop and the corresponding timing point.

## 3.4 Power Consumption Estimation

ArchC has power models for SPARC and MIPS processors [51], which were implemented based on the open source LEON3 and PLASMA RTL models using PowerSC [60] and acPower [51]. We integrate these power models into the MPSoCBench platforms in order to summarize the power consumption in a multicore environment.

The characterization process is executed only once for each processor and requires an RTL synthesis and simulation tool like Synopsys Design Compiler, Xilinx ISE or Altera Quartus. After synthesis, we simulate the back-annotated version, collect switching activity and use a Power Estimation flow like Xilinx XPower, Altera PowerPlay, or Synopsys Power Compiler. We collect the average power consumption per instruction, using a methodology similar to Tiwari [93], and create one power table for each hardware configuration (technology and operation frequency). Figure 3.12 shows the Power characterization (left) and simulation (right) flows.

From the circuit netlist it is possible to elaborate a back-annotation with detailed gate-level power information. A testbench is structured to receive the processor netlist as a device under test, allowing simulations with characterization software as inputs. Then, the circuit netlist can be simulated using Mentor Graphics Modelsim, generating switching activity files for power analysis tools such as Xilinx Xpower, Altera PowerPlay or Synopsys RTL Power Estimation flow.

This process is based on the Tiwari methodology [93], which shows that the average energy consumption of a characterization program with a loop of several instructions with the same opcode but randomized operands can be used as a power per instruction approximation. In this way, we can obtain a consistent measurement of the average power per instruction. By repeating this process to all different instructions, we conclude the energy characterization per instruction and we are able to estimate power using proces-

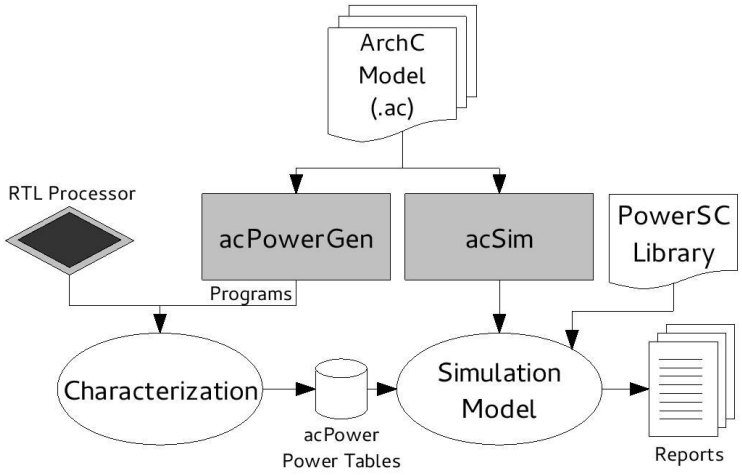



Figure 3.12: The power consumption characterization flow

sor frequency and instruction per cycles (IPC) information. This method provides an instruction-based energy consumption analysis and can be used in order to characterize the entire processor instruction set.

The next step, the simulation process, is executed after the RTL power analysis is done. The instruction-based energy consumption information is automatically integrated into the ArchC instruction set simulator. In this step, the designer uses the power information obtained from the characterization flow into the SystemC simulation. The PowerSC library is responsible for bringing the data into the model. Table 3.1 describes four power profiles available in MPSoCBench and Table 3.2 shows the energy consumed per instruction, both containing power information from the offline characterization process that are used in the MPSoCBench simulators.

Incorporating these power consumption estimation tools to the MPSoCBench allows power evaluation during program execution, detecting power consumption bottlenecks or even comparing different architectures with respect to energy efficiency.

Table 3.1: Power Profiles Description for MIPS model

| **Profile 0:** | **Profile 1:** |
|---|---|
| Technology: FreePDK45nm<br>Frequency: 50 Mhz<br>Energy in Idle: 32.970000 pJ | Technology: FreePDK45nm<br>Frequency: 125 Mhz<br>Energy in Idle: 32.580000 pJ |
| **Profile 2:** | **Profile 3:** |
| Technology: FreePDK45nm<br>Frequency: 250 Mhz<br>Energy in Idle: 32.380000 pJ | Technology: FreePDK45nm<br>Frequency: 400 Mhz<br>Energy in Idle: 32.280000 pJ |

We use ArchC instruction set simulator together with PowerSC [60] and acPower [51] to read the aforementioned power tables and generate a report with each core power consumption. The user can choose among different characterization technologies and

frequency instrumentation for Xilinx (Spartan and Virtex), Altera (Cyclone and Stratix) FPGAs, and OpenPDK 45nm ASIC.

- For MIPS:
  - Xilinx SPARTAN-3 XC3S1200e 100Mhz
  - Xilinx SPARTAN-6 XC6SLX75 100Mhz
  - Xilinx Virtex-4 XC4VLX15 100Mhz
  - Xilinx Virtex-5 XC5VLX50T 100Mhz
  - Altera 25Mhz
  - Altera CycloneV 50Mhz
  - Altera CycloneV 100 Mhz
  - Altera Stratix III EP3SL50 100Mhz
  - FreePDK ASIC 45nm at 50Mhz, 125Mhz, 250Mhz, and 400Mhz
- For SPARC:
  - Xilinx SPARTAN-3 XC3S1000 40Mhz
  - Xilinx SPARTAN-3 XC3S1200e 40Mhz
  - Xilinx SPARTAN-3 50Mhz
  - Xilinx Virtex-5 XC5VLX50T 40mHZ
  - Xilinx SPARTAN-6 XC6SLX75 40Mhz
  - FreePDK ASIC 45nm at 50Mhz, 125Mhz, 250Mhz, and 400Mhz

Table 3.2: Energy characterization per MIPS instructions

| **Instr.Name** | **Energy (pJ)** | | | | **Instr.Name** | **Energy (pJ)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Profile 0 | Profile 1 | Profile 2 | Profile 3 | | Profile 0 | Profile 1 | Profile 2 | Profile 3 |
| lb | 50.480 | 50.050 | 45.400 | 42.200 | lbu | 42.440 | 41.940 | 41.580 | 38.570 |
| lh | 48.900 | 48.510 | 45.760 | 43.200 | lhu | 44.850 | 44.270 | 43.550 | 41.590 |
| lw | 41.630 | 41.420 | 41.210 | 38.960 | lwl | 41.630 | 41.420 | 41.210 | 38.960 |
| lwr | 41.630 | 41.420 | 41.210 | 38.960 | sb | 47.830 | 45.110 | 41.190 | 40.730 |
| sh | 47.520 | 44.910 | 40.980 | 40.350 | sw | 35.530 | 35.090 | 34.770 | 34.690 |
| swl | 35.530 | 35.090 | 34.770 | 34.690 | swr | 35.530 | 35.090 | 34.770 | 34.690 |
| addi | 58.270 | 47.530 | 47.460 | 48.120 | addiu | 58.630 | 48.690 | 47.710 | 48.370 |
| slti | 41.000 | 40.540 | 40.060 | 39.720 | sltiu | 40.780 | 40.390 | 39.830 | 39.600 |
| andi | 37.630 | 37.320 | 36.680 | 36.610 | ori | 48.160 | 50.400 | 51.640 | 45.060 |
| xori | 53.010 | 48.170 | 47.180 | 44.360 | lui | 41.630 | 41.420 | 41.210 | 38.960 |
| add | 62.720 | 43.240 | 38.690 | 43.780 | addu | 61.980 | 43.270 | 38.660 | 43.580 |
| sub | 62.720 | 43.010 | 41.890 | 42.590 | subu | 61.840 | 43.100 | 41.900 | 42.690 |
| slt | 35.620 | 35.120 | 34.780 | 35.580 | sltu | 35.510 | 35.130 | 34.760 | 35.480 |
| and | 35.630 | 35.140 | 34.710 | 34.550 | or | 55.310 | 54.140 | 43.680 | 43.870 |
| xor | 60.880 | 42.200 | 38.930 | 40.640 | nor | 53.770 | 45.130 | 45.450 | 47.860 |
| nop | 32.970 | 32.580 | 32.380 | 32.280 | sll | 35.530 | 35.090 | 34.770 | 34.690 |
| srl | 35.530 | 35.030 | 34.840 | 34.710 | sra | 53.470 | 35.150 | 49.580 | 35.360 |
| sllv | 33.760 | 33.380 | 33.160 | 33.060 | srlv | 36.990 | 43.500 | 38.560 | 40.450 |
| srav | 60.110 | 40.480 | 36.780 | 41.310 | mult | 55.110 | 46.510 | 42.690 | 41.230 |
| multu | 53.540 | 46.120 | 41.980 | 40.220 | div | 51.800 | 50.640 | 47.640 | 44.970 |
| divu | 54.910 | 51.300 | 44.590 | 44.320 | mfhi | 33.760 | 33.380 | 33.160 | 33.060 |
| mthi | 33.770 | 33.390 | 33.160 | 33.060 | mflo | 42.420 | 39.190 | 37.080 | 35.710 |
| mtlo | 41.800 | 38.670 | 36.790 | 35.630 | j | 53.540 | 48.960 | 39.580 | 39.840 |
| jal | 58.980 | 53.760 | 43.170 | 43.210 | jr | 44.250 | 40.490 | 37.900 | 35.160 |
| jalr | 48.200 | 39.250 | 36.710 | 36.620 | beq | 52.210 | 48.470 | 44.880 | 45.750 |
| bne | 52.210 | 48.470 | 44.880 | 45.750 | blez | 52.210 | 48.470 | 44.880 | 45.750 |
| bgtz | 52.210 | 48.470 | 44.880 | 45.750 | bltz | 52.210 | 48.470 | 44.880 | 45.750 |
| bgez | 52.210 | 48.470 | 44.880 | 45.750 | bltzal | 52.210 | 48.470 | 44.880 | 45.750 |
| bgezal | 52.210 | 48.470 | 44.880 | 45.750 | | | | | |

### 3.4.1 Caches Power Estimation Model

The cache power model was defined according to the following methodology. We characterized three possible states of energy consumption: *idle*, *reading* and *writing* states. We use the CACTI tool for characterizing the power in inactive state ($P_{idle}$), the energy consumed in the reading and writing states ($E_r, E_w$) and the access time ($T_{access}$), using 45nm technology. These four values are stored in a file that identifies each cache model.

To calculate the total energy consumed by each cache, we must consider the energy in activity and inactivity states. For every cache operation (read/write), the energy consumed and access time are stored in variables. Consider that the $E_{active}$ is the energy consumed by cache operation (op), so the energy consumed when the cache is active is shown in Equation 3.1:

$$E_{active} = \sum_{op=read} E_r + \sum_{op=write} E_w \tag{3.1}$$

$T_{active}$ represent the time when the cache is active and is calculated summarizing $T_{access}$ of all operations (read/write):

$$T_{active} = \sum_{op} T_{access} \tag{3.2}$$

To calculate the energy consumed in inactivity $E_{idle}$ we multiply the idle power by the time when the cache is in idle (Equation 3.4). The idle time is calculated subtracting the total access time ($T_{active}$) of the simulation time ($T$)(Equation 3.3).

$$T_{idle} = T - T_{active} \tag{3.3}$$

$$E_{idle} = P_{idle} * T_{idle} \tag{3.4}$$

The total energy $E_{cache}$ consumed by the cache is obtained adding the energy consumed when the cache is active and the energy consumed in the idle state (Equation 3.5).

$$E_{cache} = E_{active} + E_{idle} \tag{3.5}$$

When we divide this value by the simulation time, we have the average power of the cache $P_{cache}$ as shown in the Equation 3.6.

$$P_{cache} = E_{cache}/T \tag{3.6}$$

Cache power consumption models take into account active and inactive states (idle state) during the simulation, in which the power estimation is directly dependent on the time estimation model. Current efforts are focused on estimating time as precisely as possible to guarantee accurate cache power estimation.

### 3.4.2 NoC Power Estimation Model

The NoC power characterization was based on the RTL NoC-Hermes description [74], using the ASIC FreePDK 45nm technology. For each router, energy consumption is calculated based on the amount of idle (inactive) and active (transmitting) clock cycles in the router.

Martins et. al [74] shows that the power dissipation of each router is a function of the reception rate of its buffers. Based on this, there was the injection of 1,000 32-flit packets in a NoC 3x3 like that in Figure 3.13 and monitoring of the the reception rate in the buffers.

The number of ports is the main parameter to calculate the energy [74]. Figure 3.13 shows three routers with different amount of ports: (i) central router with 5 ports ($R_{11}$); (ii) border router with 4 ports ($R_{01}$, $R_{10}$, $R_{12}$, $R_{21}$); (iii) corner router with 3 ports ($R_{00}$, $R_{02}$, $R_{20}$, $R_{22}$).

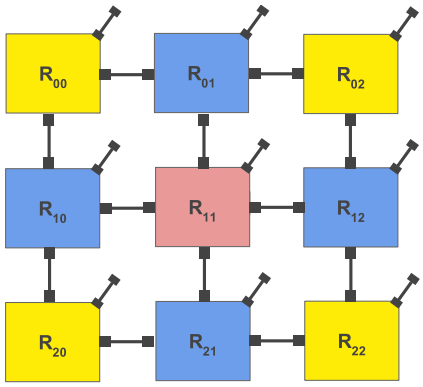



Figure 3.13: A 3x3 NoC to characterize the routers consumption.

MPSoCBench's router uses the same algorithm and arbitration policies of the characterized RTL-NoC. The framework reports an estimate of the clock cycles in active and idle states for each router.

The NoC-AT power characterization was based on the RTL NoC-Hermes description [74], using the ASIC FreePDK 45nm technology. Relevant features of the NoC models include wormhole packet switching, XY routing, input buffering and centralized round-robin arbitration.

Based on this characterization, Figure 3.14 shows the results of NoC energy consumption. Three benchmarks were simulated in five different MPSoC size (4, 8, 16, 32 and 64) to evaluate the NoC energy consumption. All simulations were executed in the frequency set (50, 125, 250 and 400MHz) to characterize the NoC results. The y-axis represents the NoC consumption in pJ, in a logarithmic scale. The x-axis presents the benchmarks simulated and the NoC frequency. The colored columns represent the result of NoC consumption for different MPSoC sizes.

Similarly to cache models, router power consumption models take into account active states and inactive states (idle state) during the simulation, in which the power estimation is directly dependent on the time estimation model. Current efforts are directed to estimate time more precisely.

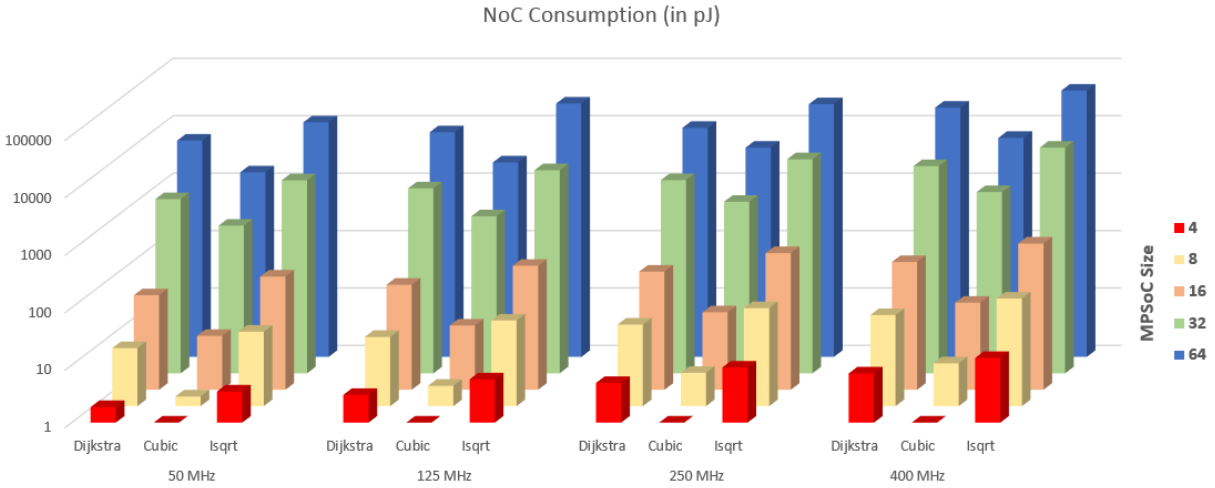



Figure 3.14: NoC power estimation results at 50Mhz, 125Mhz, 250Mhz, and 400Mhz.

## 3.5 The MPSoCBench Simulation flow

To simplify the configuration, compilation and execution process, MPSoCBench comes with a python script that receives configuration parameters, creates a run-directory for each different platform, and starts compilation and/or simulation. We have validated all platform configurations in an Ubuntu Linux environment. Figure 3.15 illustrates a MPSoC simulation process using MPSoCBench.

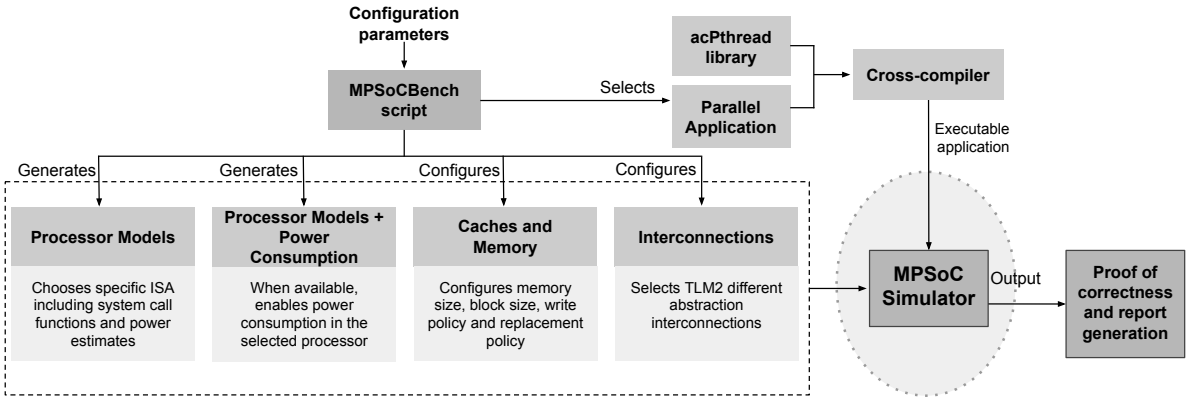



Figure 3.15: Development flow of a MPSoC simulator using MPSoCBench

With the user provided parameters, an automated script generates the appropriate code for processors, interconnections, caches, memories, and IPs. These components are compiled to create the MPSoC simulators required. Applications are compiled with the appropriate cross-compilers, and the executable files are stored together with the simulators.

After each simulation, the output is verified against a golden result. It is remarkably easy to configure and build platforms using the MPSoCBench script. The Appendix A

shows how to install the MPSoCBench and the tools that it uses, and the Appendix B demonstrates how to build and run MPSoC simulator with MPSoCBench.

## 3.6 Conclusion

This chapter introduced the MPSoCBench toolset, which is a configurable and extensible set of MPSoCs, useful to improve development and evaluation of MPSoC platforms, using well known methodologies and tools.

MPSoCBench has a large set of platforms, with three different interconnection devices, easily configurable from 1 to 64 cores of 4 different available processor models, capable of running 17 different parallel applications. The total combined size reaches 864 distinct configurations.

The toolset is released as an open-source license, and it is available in two forms: a virtual machine with all infrastructure ready for use and as source code. Tutorials for installation of all tools are provided in the MPSoCBench website (`http://www.archc.org/benchs/mpsocbench`).

# Chapter 4

# MPSoCBench Characterization

The goal of this Chapter is to characterize the MPSoCBench and to provide enough information so that the user can choose which features are most suitable for his/her purposes. The Chapter is organized as follows:

Section 4.1 compares the four ISAs provided by our tool; Section 4.2 presents network traffic information comparing distinct interconnections available; Section 4.3 evaluates memory and caches; Section 4.4 discusses about the time accuracy in the MPSoCBench simulators; Section 4.5 shows simulation time in hundreds of configurations and discuss how to improve performance of SystemC simulators; Section 4.6 shows how we can use the power estimation infrastructure to capture the processor energy and power behavior; and finally, Section 4.7 describes some usage scenarios in which we expect that MPSoCBench be used in the future.

## 4.1 Processor Model Evaluation

As described in Section 3.1.1, MPSoCBench contains four ArchC behavioral processor models: **ARM** is the 32-bit ARMv5e instruction set; **MIPS** is a 32-bit RISC MIPS-I instruction set; **SPARC** is a V8 version of the 32-bit SPARC architecture; **PowerPC** is a 32-bit version of the Power instruction set. All models include operating system call emulation. ArchC tools generate SystemC/TLM2-based simulators among other tools (compilers, assemblers, debuggers,...).

We present in Table 4.1 general information about the architectures and ISAs available in MPSoCBench. We show the number of general purpose registers in the register bank, the number of specific purpose registers, the amount of instruction formats, total implemented instructions of each ISA, and endianness.

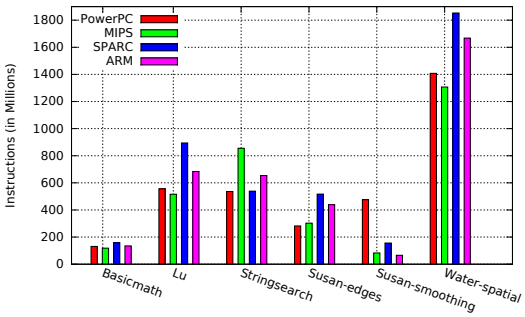
As a basic use case, we provide some comparison among the four available ISAs. As expected, they have fundamental differences and specific optimizations that interfere directly with the number of instructions executed by each of them. For instance, we illustrate in this section the number of executed instructions on a platform based on all four available processors, running all applications.

As there are considerable differences in the magnitude of the numbers shown, we divide the values into four graphs. First, we show the parallel applications based on

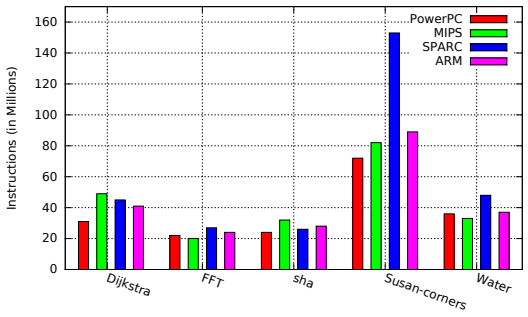
Table 4.1: Architecture and ISA general information

| | ARM | MIPS | PowerPC | SPARC |
|---|---|---|---|---|
| General Purpose Registers | 16 | 32 | 32 | 288 |
| Specific Purpose Registers | 20 | 3 | 13 | 8 |
| Instruction Format | 7 | 3 | 6 | 6 |
| Instruction in the ISA | 95 | 59 | 181 | 118 |
| Endianness | little | big | big | big |

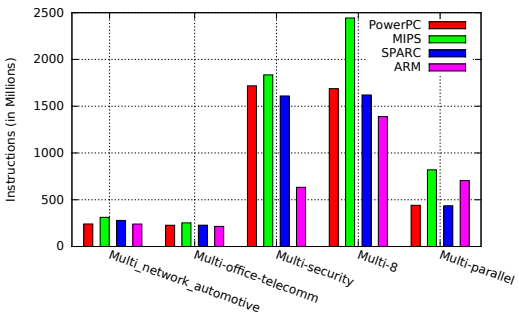
cooperative parallelism in Figure 4.1(a) and Figure 4.1(b). These number refers to the single core platform using the four processor models, with an instruction cache and a data cache per core. The number of instructions executed in applications of the Multi-software group are shown in Figure 4.1(c), which were obtained through simulations of 4-core platforms, except the execution of the `Multi-8` application that requires 8-core platforms. We separated the data obtained through simulations of 16-core platforms running `Multi-16` application, and we show the data per core in Figure 4.1(d).



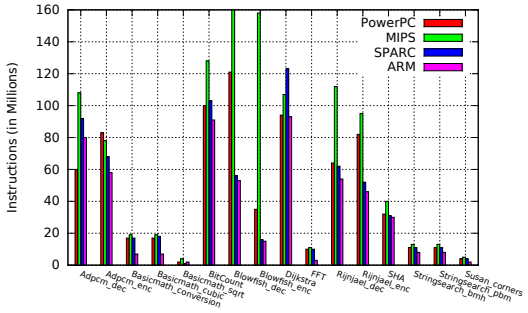

(a) Large applications based on cooperative parallelism




(b) Small applications based on cooperative parallelism




(c) Multi-software applications




(d) Multi-16 application: 16 different applications running in a 16-core platform (The number of instructions executed by the blowfish_dec application in single-core MIPS platform is 544 million.)

Figure 4.1: Instructions executed in all applications (in Millions)

As the applications execute tasks in parallel, there are synchronization points that cause changes in the number of instructions executed. For example, there are synchro-

nization points implemented through *mutex* variables to guarantee that two processors do not access a shared resource. To avoid that a processor does not continuously check if the resource is released for its use, which is called as "busy wait" state, we use a loop to perform some statements without effect before the processor tries to read the lock variable again.

The standard SystemC and TLM models present in the MPSoCBench make it easy to include new processors like OVP [57] or RTL models through wrappers, giving to researchers the opportunity to evaluate their results in bigger platforms without the need to retarget the whole software stack for such platforms.

## 4.2 Network Traffic

MPSoCBench simulators report network traffic data, making it possible to evaluate other routing algorithms, topologies, and optimizations. Figure 4.2 shows an example of the network built automatically by the MPSoCBench script that organize the routers in a 4x3 NoC-AT connecting eight cores, memory, and IPs. In the figure, each Router contains four ports connecting with other routers (North, South, West, and East ports) and Local port for communication with the device. Due to the topology, we deactivated the border ports. For example, the North and West ports of Router $R_{0,0}$ are considered inactive; similarly, the South and East ports of Router $R_{2,3}$ are inactive too. So, an error is reported if the routing algorithm attempts to send data packets through these ports. This figure also illustrates a *hop* that is the distance between two routers.

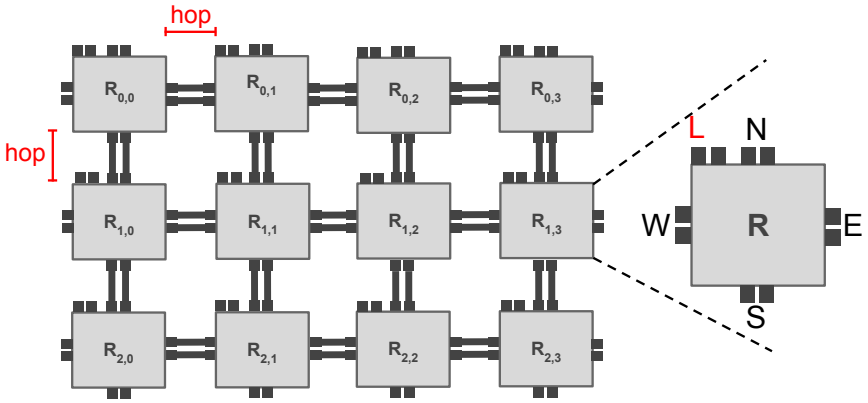
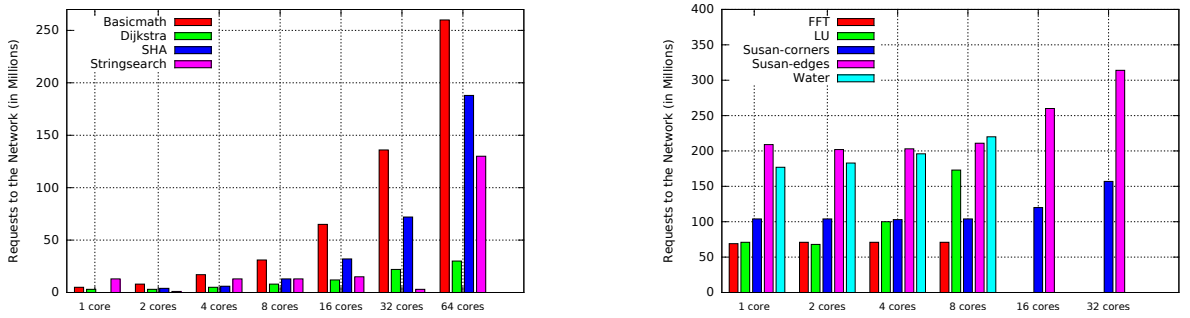



Figure 4.2: Example of a 4x3 NoC-AT

We executed ten applications in multicore MIPS platforms from 1 to 64 cores connected through a NoC-AT and showed at Figure 4.3 the total number of requests to the network performed when we scale from 1 to 64 cores, divided into two graphs to improve clarity. Figure 4.4 shows the number of hops in the same simulations, also divided into two

graphics. We do not use data caches intentionally to stress the network with requests to main memory[1].

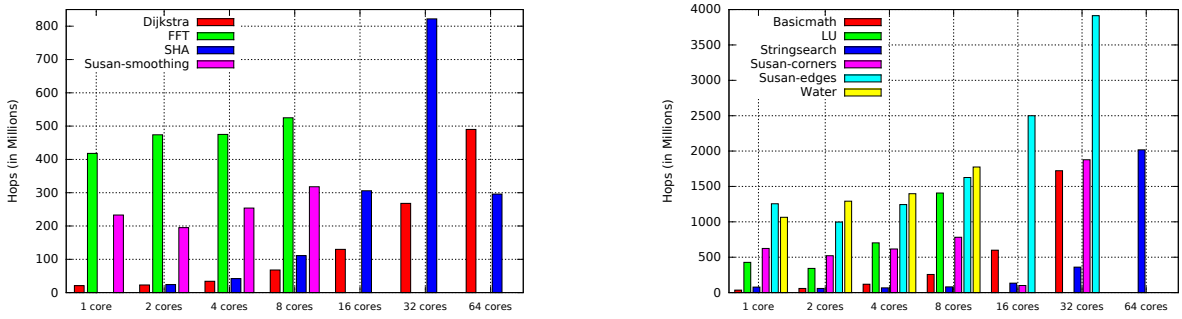
Figures 4.3 and 4.4 show how the software scales. Although all applications perform cooperative parallelism, they have significant differences in scaling. For example, in the case of the SHA, as we increase the number of cores, we also increase the number of input files to the SHA algorithm, so we expect a much larger increase of executed instructions and also on the amount of data through the network. On the other hand, Susan-corners algorithms run over the same input image independently of the number of cores, partitioning the image and dividing it among the available cores. So we note that the number of instructions executed and the amount of data through the network grows slowly as we increase the number of cores.




(a) Total number of requests to the network

(b) Total number of requests to the network

Figure 4.3: Total number of requests to the network (in Millions) performed by parallel scalable benchmarks in a 4x3 NoC-AT




(a) Total number of hops

(b) Total number of hops

Figure 4.4: Total number of hops (in million) performed by parallel scalable benchmarks in a 4x3 NoC-AT

We can also evaluate the network traffic considering a per-router perspective. For instance, we executed the SHA benchmark in a 4x3 NoC-AT with eight cores and monitored

[1]We opted to not show complete bars for the water-spatial application, which are much larger than the others, to not compromise the clarity of the numbers in the graphs: 1163, 1181, 1170, and 1275 million of requests, and 6980, 5872, 6978, and 9565 million of hops in 1-,2-,4-, and 8-core platforms, respectively.

| Router | Node | Ports | | | | |
|---|---|---|---|---|---|---|
| | | North | South | East | West | Local |
| $R_{0,0}$ | MEM | - | 4900135 | 14532508 | - | 19431407 |
| $R_{0,1}$ | Lock | - | 4846811 | 9690069 | 1236 | 5597 |
| $R_{0,2}$ | Intr_Ctrl | - | 4847121 | 4842952 | 11 | 1 |
| $R_{0,3}$ | DVFS-IP | - | 4842956 | - | 14 | 18 |
| $R_{1,0}$ | $P_0$ | 19431407 | 2418446 | 523 | - | 2481689 |
| $R_{1,1}$ | $P_1$ | 5597 | 2419543 | 7 | 7268870 | 2427268 |
| $R_{1,2}$ | $P_2$ | 1 | 2424381 | 8 | 2421209 | 2422740 |
| $R_{1,3}$ | $P_3$ | 18 | 2421745 | - | 2421209 | 2421211 |
| $R_{2,0}$ | $P_4$ | 9681371 | - | 713 | - | 2418446 |
| $R_{2,1}$ | $P_5$ | 2736 | - | 4 | 7263638 | 2419543 |
| $R_{2,2}$ | $P_6$ | 0 | - | 6 | 4846122 | 2424381 |
| $R_{2,3}$ | $P_7$ | 8 | - | - | 2421743 | 2421745 |

Table 4.2: Number of packets transmitted through each Router in a 4x3 NoC-AT with eight cores, memory, lock, interrupt controller and DVFS-IP

the number of packets transmitted by each network router. All these routing values are stored at the *rundir* in a `local_report.txt` file. Table 4.2 shows in the first and second columns, twelve routers and their connections with processors or devices. The following columns show the number of packets transmitted from the router to each one of its ports. For instance, $R_{0,0}$ connects the external memory to the network. All 19431407 packets that passed from the $R_{0,0}$ to the memory device (through the local memory port) refer to requests to read/write from/to the memory, and have come from the South port or the East port for sure (as the memory is at the network top left corner, and we are using XY routing protocol). If we summarize the total traffic through the $R_{0,0}$, we obtain 19432643 packets (4900135 through the South port plus 14532508 packets through the East port). So, part of these packets are requests to the memory (19431407) and the difference (1236 packets) refers to packets that have come from the $R_{0,1}$ (through its west port), and are sent to processors through the southern port (probably backward packets that have come from the LOCK device to the processors).

Table 4.2 clearly shows that there is a considerable increase in traffic over the network coming from memory, compromising scalability. This is an unavoidable problem in a system with a single shared memory. An alternative under development is to use distributed shared memory. The same situation occurs with other shared IPs such as LOCK, DVFS and Intr_Ctrl, but the number of accesses to these IPS is significantly smaller than the number of memory accesses. Therefore, we include in our future work list to distribute the address space of the system in multiple memories to address scalability.

This section presented the network traffic data obtained from multicore platforms simulators running several scalable and parallel applications. We showed the requests to the network and the number of hops to execute this software, and we also presented the number of packets per router to evaluate the network traffic in a per-router perspective. This infrastructure makes possible the development and evaluation of new routing algorithms,

network topologies, packet buffering optimizations, and further specializations to improve the network performance.

## 4.3 Memory and Cache Evaluation

Most modern computer systems have at least four independent caches: an instruction cache to speed up instruction fetch, a data cache to speed up data fetch and store, and translation lookaside buffers (TLBs) for instructions and data used to speed up virtual-to-physical address translation for executable instructions and data, respectively. Since there is no virtual memory system implemented in the ArchC processors, there is no need for TLB.

To show how the memory hierarchy works in our simulators, we performed several simulations and used the reports to produce graphics containing cache and memory information. We executed 7 applications from the Cooperative group on single-core simulators, in which we fixed each core one-level data cache to 64 blocks, 8 words per block, and we varied the associativity at 1-way (or direct mapped), 2-way, 4-way, and 8-way set associativity. Figure 4.5(a) shows the percentage of read hits in the Instruction Cache. Figures 4.5(b) and 4.5(c) show the percentage of read hits and write hits in the Data Cache. As we can see in the graphics, the 2-way set associative cache achieves the best values for all applications.

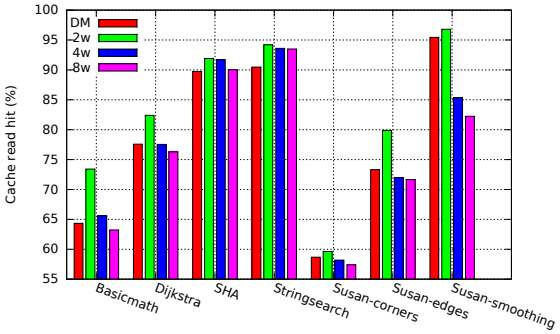
To evaluate replacement policies, we simulated the 11 benchmarks on the same platforms using three algorithms: First In First Out (FIFO), Least Recently Used (LRU), and Random, and we show in Figure 4.5(e) and 4.5(f) the number of blocks replaced. We divided the values into two figures to improve clarity.

To analyze the scalability, we used a data cache with 512 blocks, 32 words per block, 2-way associative, and performed the same simulations using the FIFO, LRU, and RANDOM replacement policies in several multicore platforms running the Stringsearch benchmark. As we can see in the graphics in Figure 4.5(d), the number of block evictions decreases significantly as we increase the number of cores, because the amount of data per core also decreases (while the cache size increases since that each core has its own cache), and the LRU replacement policy causes the fewest number of block evictions.

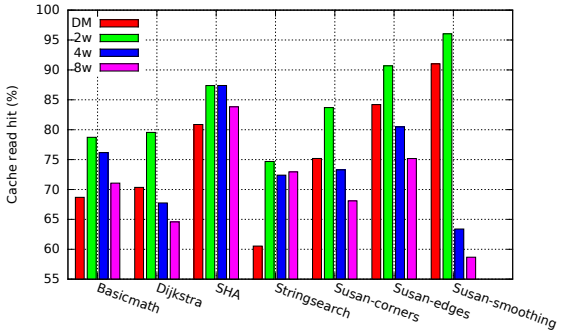
When we use a data cache in multicore simulators, we automatically enable the cache coherence protocol. Last level caches are connected to a shared *directory* (DIR), which is a device that implements the MSI cache coherence algorithm, enabling the parallel computing with shared resources. To evaluate the impact of using caches in the system, we simulated platforms with 1-,4-,16-,64-core MIPS, with 2-way set associative write through instruction and data caches, with FIFO replacement policy, running eleven parallel scalable applications, and we plot the number of access to the DIR device is in Figure 4.6. In these experiments we used only one-level caches.

We leave the implementation of other cache coherence protocols as future work.

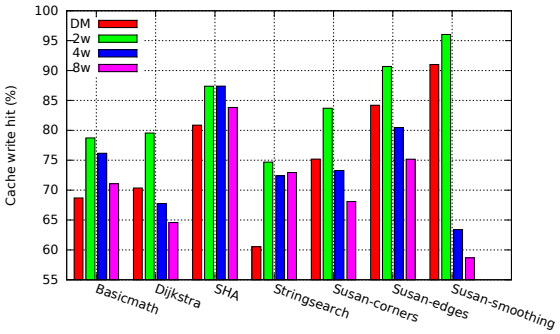
These simulator infrastructure aspects are also relevant for educational use in computer architecture disciplines. Besides being able to assess known techniques, this infrastructure



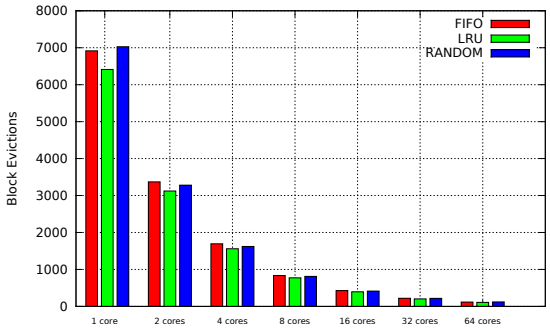

(a) Instruction Cache Read Hit



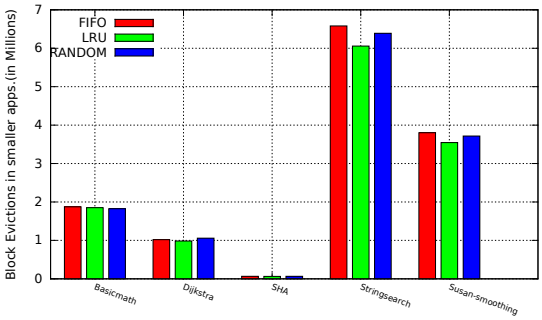

(b) Data Cache Read Hit



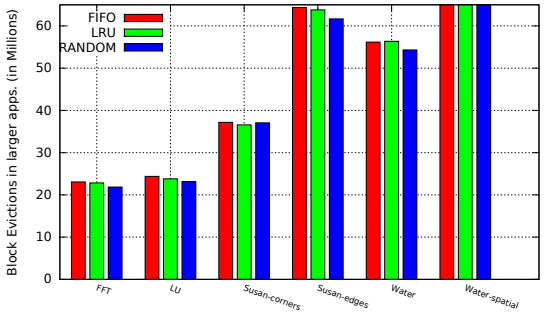

(c) Data Cache Write Hit




(d) Average Block evictions in multicore platforms running Stringsearch




(e) Block evictions for smaller applications

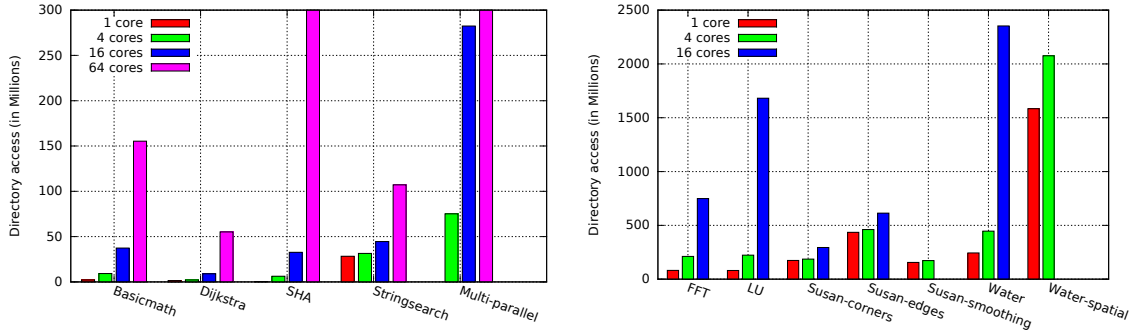



(f) Block evictions for Larger applications (We do not show complete bars for Water-spatial because they are much greater than the others: 373 million (FIFO), 374 (LRU), and 358 million (RANDOM).

Figure 4.5: Cache read/write hit and block evictions

can also be used to evaluate optimizations, new policies and algorithms to improve cache performance.

To evaluate RAM, we executed all 17 applications on platforms with the minimum number of cores as possible for each benchmark without caches, considering 512MB memory-size, and we captured the number of memory operations (reads and writes). Using the graph in Figure 4.7, one can choose among applications that have or not a massive use of memory.




(a) (b)

Figure 4.6: Directory access in multicore MIPS platforms running twelve parallel and scalable applications (To not compromise the graph visibility, we did not show the complete bar reached by the Multi-parallel applications in 64-core platforms (1229.64 million).)

When we use cache, the communication with the main memory is performed in *blocks*. Each block has a configurable number of *words*. The values are shown in Figure 4.7 in words (not blocks), considering that there are no cache memories in the system.

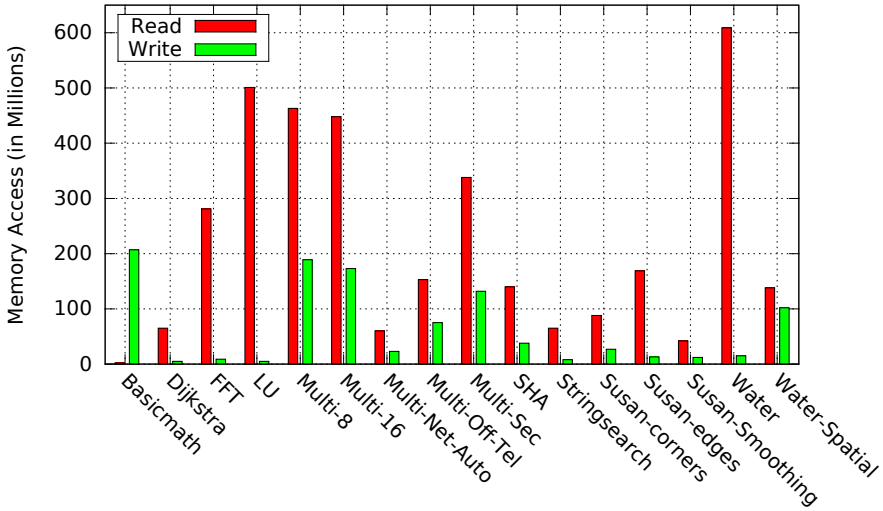
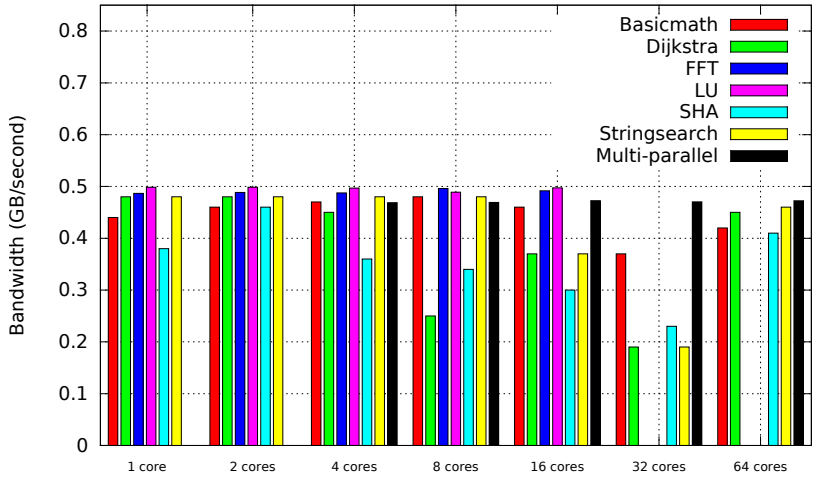


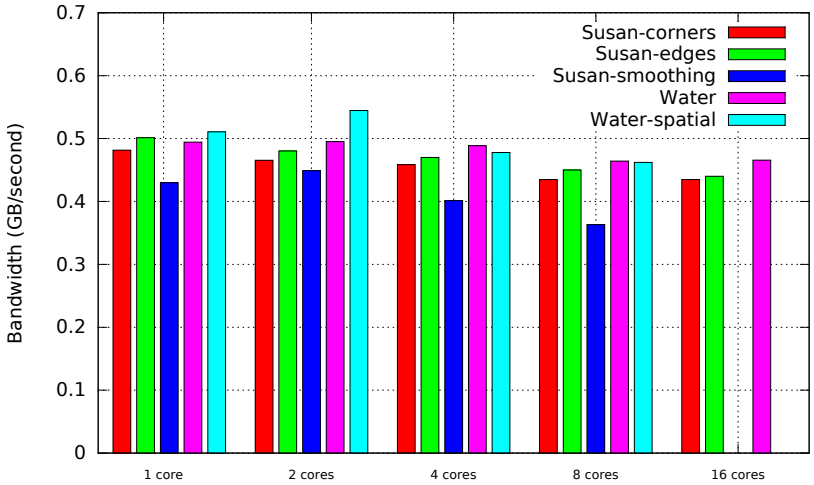

Figure 4.7: Memory accesses (in Millions) for read and write in all Applications

As the Multi-parallel application has a huge number of memory operations (2,215 billions of reads and 139 billions of writes), we do not put its values in the graph.

The DRAMSim memory simulator offers reports with several memory estimates, like latency, bandwidth, average power, and so on. We simulated 12 scalable applications on multicore platforms from 1 to 64 cores with data and instructions caches, 512MB of RAM memory, characterized from a DDR2 Micron memory with a single channel and 8 banks. We plotted the average of the 8 banks bandwidth and latency in Figures 4.8 and 4.9. We divided the data into two graphics in each figure to improve clarity.




(a)




(b)

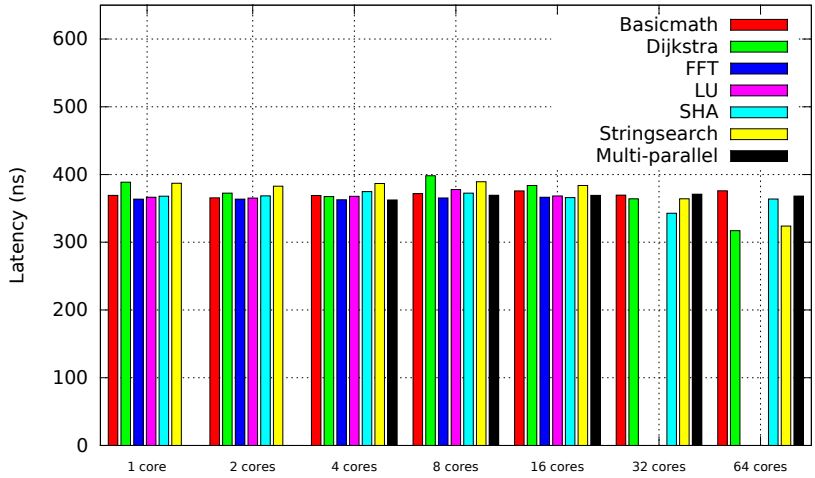
Figure 4.8: Memory Bandwidth running 15 parallel and scalable applications in multicore platforms from 1 to 64 cores

The DRAMSim also exports the average power to refresh the memory, the background power, and the dynamic power. We show in Table 4.3 the average dynamic power consumption obtained through simulations of several multicore platforms running the same applications.

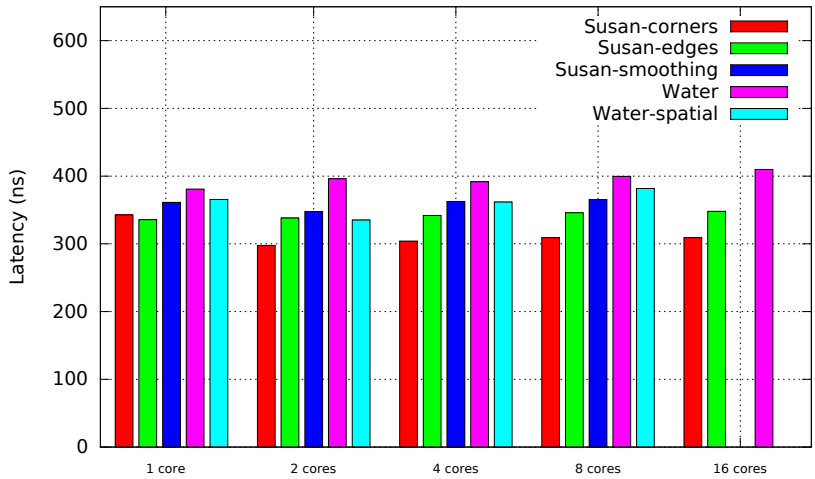
Currently, we are working on fully integrate the data from DRAMSim to the MPSoCBench statistics (at least, power and timing information).

## 4.4 Timing Accuracy

MPSoCBench contains ISAs simulators with timing models based on real implementations previously described in Section 3.1.1. Besides that, each system component (routers, wrappers, and memories) can be parametrized with a specific latency. The NoC-AT timing information was obtained through the RTL NoC-Hermes [74] using the ASIC FreePDK




(a)




(b)

Figure 4.9: Memory Latency running 12 parallel and scalable applications in multicore platforms from 1 to 64 cores

45nm technology. The latency to access caches was obtained through the CACTI tool. The values used for the following experiments are presented in Table 4.4.

Figures 4.10(a) to 4.10(i) show simulated time in multicore platforms with 1 up to 64 MIPS processors, with the low frequency at 50Mhz and the high frequency at 400Mhz, considering processors, caches, and NoC-AT routers, running nine scalable applications. The simulated time is captured by the SystemC `sc_time_stamp` function at the end of the simulation, and can be considered as the system execution time. We used only scalable software to show the simulator time behavior and performance over several multicore platforms and applications.

Applications have different timing behavior, exposing different parallelization techniques and scalability. While applications like Basicmath, LU, and SHA present high increase in simulated runtime, Susan-corners, Susan-smoothing, and Stringsearch applications do not significantly increase the runtime as we increase the number of cores.

Table 4.3: Memory average dynamic power (Watts)

| **Applications** | **Number of Cores** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
| Basicmath | 2.38 | 2.43 | 2.38 | 2.38 | 1.58 | 2.70 | 1.65 |
| Dijkstra | 2.16 | 2.56 | 2.44 | 1.76 | 2.20 | 1.60 | 1.66 |
| FFT | 2.58 | 2.58 | 2.59 | 2.58 | 2.58 | - | - |
| LU | 2.60 | 2.60 | 2.59 | 2.57 | 2.59 | - | - |
| SHA | 2.16 | 2.44 | 2.08 | 2.04 | 1.97 | 1.56 | 1.65 |
| Stringsearch | 2.56 | 2.57 | 2.56 | 2.57 | 2.20 | 1.60 | 2.56 |
| Susan-corners | 2.60 | 2.70 | 2.68 | 2.53 | 2.53 | - | - |
| Susan-edges | 2.71 | 2.56 | 2.67 | 2.58 | 2.56 | - | - |
| Susan-smoothing | 2.35 | 2.40 | 2.25 | 2.13 | - | - | - |
| Water | 2.55 | 2.57 | 2.54 | 2.48 | 2.47 | - | - |
| Water-spatial | 2.60 | 2.76 | 2.56 | 2.51 | - | - | - |
| Multi-parallel | - | - | 2.51 | 2.5 | 2.52 | 2.51 | 1.59 |

| | |
|---|---|
| ARM | ARM9E-S [1] |
| MIPS | MIPS32 M5150 Processor Core Family [69] |
| PowerPC | PowerPC 405TM Core [32] |
| SPARC | UT699 LEON 3FT/SPARCTM V8 MicroProcessor [48] |
| NoC-AT Routers | 5 cycles for arbitration<br>2 cycles per flit |
| Lock | 2 nanoseconds |
| DVFS | 2 nanoseconds |
| Interruption Controller | 5 nanoseconds |
| Caches | 2-way, 64 blocks, 32 words per block<br>Access time: 0. 0.60096 ns<br>Random cycle time: 0.30567 ns |

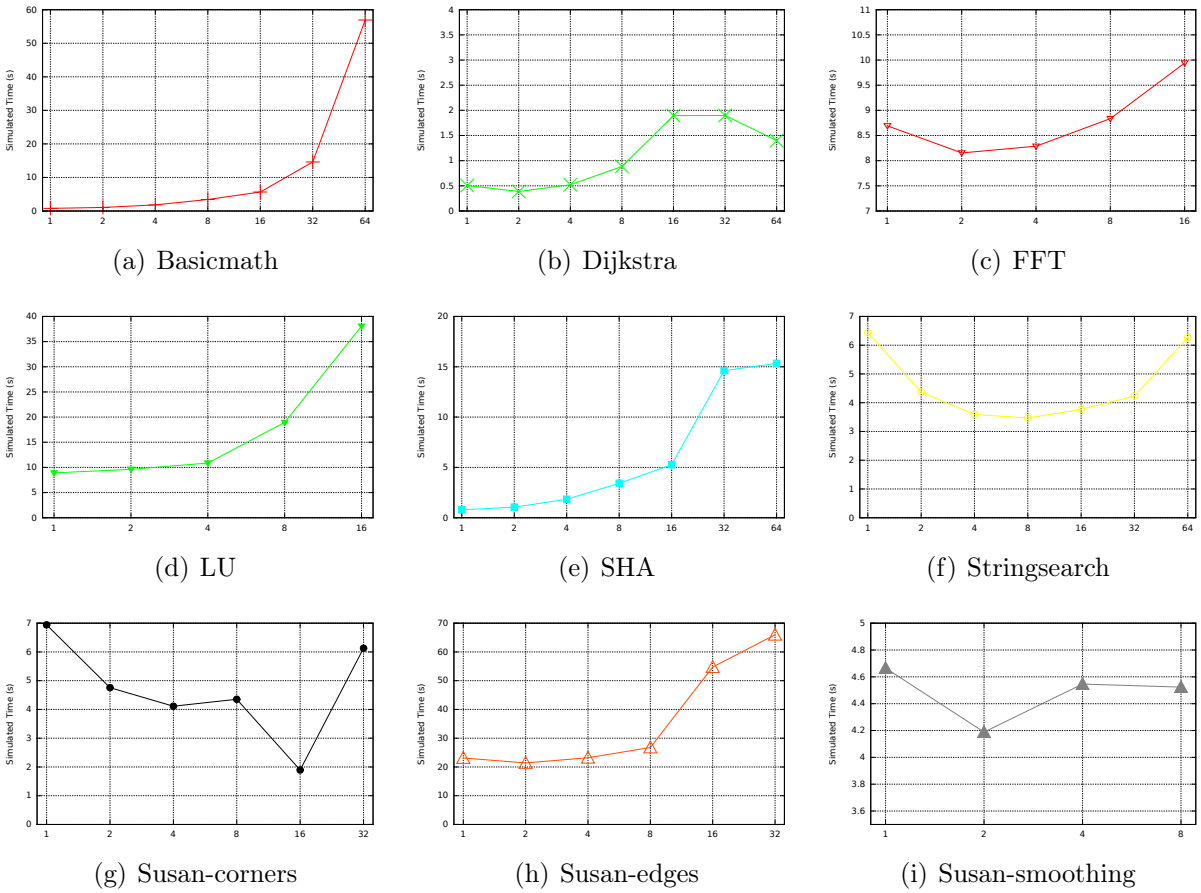
Table 4.4: Timing Information

MPSoCBench has platforms at different abstraction levels, which is useful to explore the trade-off between performance and accuracy. When we connect processor models using the NoC-AT routers, we can expose system timing values combining the advantages of the TLM2 approximately timed coding style (high accuracy) in a system level simulator (high performance).

## 4.5 Simulation Performance

The simulation time directly affects the life cycle of a project; thus, one of the main focus in system design is the study on how to accelerate simulation. SystemC simulation kernel is sequential and can not run tasks in parallel using multiple cores.

Based on this, as we increase the number of components in a virtual platform, we also increase the simulation time. To show that this increase does not make unfeasible the use of the MPSoCBench multicore simulators, we show the simulation time of hundreds of




(a) Basicmath (b) Dijkstra (c) FFT

(d) LU (e) SHA (f) Stringsearch

(g) Susan-corners (h) Susan-edges (i) Susan-smoothing

Figure 4.10: Simulated time in multicore platforms running all parallel scalable applications

configurations, scaling from 1 to 64 cores and using different processors and interconnections, without DVFS and power measurements. Users can use this information to make the best choice of processor model, interconnections, and benchmarks according to their needs and requirements.

Tables 4.5 and 4.6 show the simulation time of each application running on all available multicore platforms, using ARM, MIPS, SPARC, and PowerPC models and the Router as interconnection, without Power Consumption or frequency management. Tables 4.7 and 4.8 show simulation time of simulators using the same processor models and the NoC-LT as interconnection. We also show in Tables 4.9 and 4.10 the same results using the NoC-AT as interconnection.

Simulation times vary from a few seconds to less than 2,000s, making it feasible to explore multiple configurations, different parameters, and multiple runs. Each value in the table is the average of three distinct simulations, and standard deviation varies from 0.4% to 8.6%.

We have omitted values corresponding to configurations that are not applicable, given the particularities of the parallelization technique. All 864 configurations could be run, sequentially, in 85 hours, and in parallel through an HTCondor cluster with eight quad-core machines (32 total cores) in less than 3 hours.

Table 4.5: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device, PowerPC and MIPS processors

| Applications | | PowerPC | | | | | | | MIPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 2 | 3 | 5 | 11 | 21 | 56 | 139 | 2 | 3 | 5 | 10 | 21 | 54 | 121 |
| | **lu** | 13 | 18 | 29 | 57 | 116 | - | - | 11 | 16 | 25 | 21 | 103 | - | - |
| | **sha** | 1 | 2 | 3 | 7 | 14 | 36 | 127 | 1 | 2 | 4 | 8 | 17 | 45 | 146 |
| | **water** | 25 | 38 | 54 | 73 | 303 | - | - | 32 | 33 | 49 | 67 | 275 | - | - |
| | **dijkstra** | 1 | 1 | 2 | 2 | 5 | 13 | 40 | 1 | 1 | 1 | 2 | 5 | 13 | 42 |
| | **fft** | 24 | 32 | 51 | 93 | 173 | - | - | 21 | 29 | 44 | 80 | 152 | - | - |
| | **stringsearch** | 19 | 19 | 20 | 21 | 23 | 28 | 45 | 29 | 29 | 30 | 30 | 31 | 35 | 44 |
| | **susancorners** | 17 | 18 | 19 | 21 | 29 | 66 | - | 19 | 20 | 20 | 21 | 30 | 65 | - |
| | **susanedges** | 33 | 42 | 48 | 53 | 59 | 114 | - | 35 | 44 | 50 | 55 | 58 | 109 | - |
| | **susansmoothing** | 100 | 101 | 107 | 138 | - | - | - | 19 | 20 | 20 | 27 | - | - | - |
| | **water-spatial** | 248 | 250 | 261 | 264 | - | - | - | 217 | 218 | 223 | 234 | - | - | - |
| Multi | **multi_p** | - | - | 22 | 42 | 113 | 641 | 2170 | - | - | 27 | 53 | 110 | 592 | 2046 |
| | **multi_office_telecomm** | - | - | 68 | - | - | - | - | - | - | 202 | - | - | - | - |
| | **multi_net_automotive** | - | - | 115 | - | - | - | - | - | - | 69 | - | - | - | - |
| | **multi_security** | - | - | 381 | - | - | - | - | - | - | 355 | - | - | - | - |
| | **multi_8** | - | - | - | 542 | - | - | - | - | - | - | 512 | - | - | - |
| | **multi_16** | - | - | - | - | 940 | - | - | - | - | - | - | 729 | - | - |

### 4.5.1 Interconnection Evaluation

We have compared the simulation time on multicore platforms using the three different interconnection mechanisms in multicore PowerPC platforms running 11 applications from the cooperative group. Since the Router is the simpler and faster interconnection, we compare the simulation time using the NoC-AT and the Router in Figure 4.11 and the NoC-LT and Router in Figure 4.12. We use the Router simulation time as the baseline in both cases.

As expected, the simulation time on platforms using the approximately timed coding style increases faster than using loosely timed coding style because of a large number of SystemC threads required to simulate the NoC nodes and wrappers, and consequently, the high number of context exchange between them. The Stringsearch benchmark presents the worst slowdown, due to a large number of memory accesses to read its input strings; the simulation time of this benchmark on 64-core platforms with NoC-AT as interconnection is 34× greater than on 64-core platforms with Router.

Considering that there are differences in accuracy and performance in MPSoCBench platforms mostly based on the choice of the interconnection mechanism, we propose different use cases for each of them in Table 4.11:

Table 4.6: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device, SPARC and ARM processors

| Applications | | SPARC | | | | | | | ARM | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 2 | 6 | 10 | 17 | 36 | 78 | 192 | 2 | 3 | 9 | 10 | 21 | 48 | 133 |
| | **lu** | 23 | 29 | 44 | 87 | 174 | - | - | 26 | 33 | 49 | 94 | 188 | - | - |
| | **sha** | 1 | 2 | 3 | 7 | 16 | 42 | 143 | 1 | 2 | 4 | 9 | 20 | 52 | 174 |
| | **water** | 40 | 38 | 52 | 72 | 297 | - | - | 45 | 44 | 62 | 84 | 323 | - | - |
| | **dijkstra** | 1 | 1 | 1 | 2 | 6 | 15 | 46 | 1 | 1 | 1 | 2 | 5 | 12 | 45 |
| | **fft** | 32 | 43 | 66 | 119 | 215 | - | - | 39 | 54 | 86 | 143 | 261 | - | - |
| | **stringsearch** | 21 | 22 | 22 | 23 | 24 | 28 | 38 | 29 | 29 | 29 | 30 | 31 | 35 | 45 |
| | **susancorners** | 28 | 28 | 29 | 30 | 39 | 72 | - | 26 | 26 | 27 | 30 | 39 | 72 | - |
| | **susanedges** | 83 | 104 | 118 | 143 | 136 | 221 | - | 57 | 69 | 78 | 84 | 90 | 160 | - |
| | **susansmoothing** | 33 | 33 | 38 | 52 | - | - | - | 20 | 20 | 22 | 30 | - | - | - |
| | **water-spatial** | 351 | 353 | 369 | 380 | - | - | - | 209 | 198 | 211 | 233 | - | - | - |
| Multi | **multi_p** | - | - | 25 | 54 | 125 | 520 | 1707 | - | - | 30 | 57 | 115 | 227 | 491 |
| | **multi_office_telecomm** | - | - | 199 | - | - | - | - | - | - | 100 | - | - | - | - |
| | **multi_net_automotive** | - | - | 87 | - | - | - | - | - | - | 89 | - | - | - | - |
| | **multi_security** | - | - | 348 | - | - | - | - | - | - | 412 | - | - | - | - |
| | **multi_8** | - | - | - | 438 | - | - | - | - | - | - | 555 | - | - | - |
| | **multi_16** | - | - | - | - | 560 | - | - | - | - | - | - | 562 | - | - |

Table 4.7: Average simulation time, in seconds, for three distinct execution of each software on platforms using the NoC-LT, PowerPC and MIPS processors

| Applications | | PowerPC | | | | | | | MIPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 2 | 3 | 5 | 11 | 27 | 82 | 224 | 2 | 3 | 6 | 12 | 27 | 66 | 194 |
| | **lu** | 22 | 31 | 54 | 121 | 272 | - | - | 20 | 26 | 45 | 100 | 224 | - | - |
| | **sha** | 1 | 3 | 6 | 13 | 28 | 76 | 270 | 1 | 3 | 6 | 12 | 28 | 78 | 269 |
| | **water** | 59 | 62 | 94 | 144 | 702 | - | - | 51 | 52 | 79 | 118 | 623 | - | - |
| | **dijkstra** | 1 | 2 | 2 | 5 | 11 | 32 | 81 | 1 | 2 | 2 | 3 | 8 | 25 | 91 |
| | **fft** | 31 | 48 | 84 | 167 | 1058 | - | - | 16 | 39 | 67 | 134 | 270 | - | - |
| | **stringsearch** | 34 | 35 | 36 | 40 | 47 | 64 | 125 | 51 | 50 | 51 | 57 | 65 | 80 | 128 |
| | **susancorners** | 27 | 28 | 31 | 36 | 56 | 143 | - | 29 | 30 | 36 | 37 | 54 | 129 | - |
| | **susanedges** | 56 | 71 | 90 | 117 | 139 | 314 | - | 59 | 77 | 85 | 111 | 121 | 270 | - |
| | **susansmoothing** | 168 | 168 | 186 | 277 | - | - | - | 33 | 35 | 36 | 53 | - | - | - |
| | **water-spatial** | 398 | 406 | 451 | 480 | - | - | - | 348 | 347 | 361 | 400 | - | - | - |
| Multi | **multi_p** | - | - | 34 | 79 | 193 | 900 | 4041 | - | - | 31 | 86 | 234 | 380 | 3743 |
| | **multi_office_telecomm** | - | - | 97 | - | - | - | - | - | - | 274 | - | - | - | - |
| | **multi_net_automotive** | - | - | 174 | - | - | - | - | - | - | 108 | - | - | - | - |
| | **multi_security** | - | - | 558 | - | - | - | - | - | - | 507 | - | - | - | - |
| | **multi_8** | - | - | - | 828 | - | - | - | - | - | - | 766 | - | - | - |
| | **multi_16** | - | - | - | - | 953 | - | - | - | - | - | - | 1611 | - | - |

Table 4.8: Average simulation time, in seconds, for three distinct execution of each software on each platform using the router interconnection device, SPARC and ARM processors

| Applications | | SPARC | | | | | | | ARM | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 2 | 5 | 9 | 20 | 45 | 101 | 264 | 2 | 3 | 6 | 13 | 30 | 69 | 210 |
| | **lu** | 33 | 48 | 74 | 169 | 380 | - | - | 44 | 56 | 91 | 189 | 402 | - | - |
| | **sha** | 1 | 2 | 5 | 11 | 26 | 73 | 264 | 1 | 3 | 7 | 15 | 33 | 96 | 336 |
| | **water** | 83 | 78 | 119 | 175 | 874 | - | - | 110 | 118 | 173 | 242 | 1040 | - | - |
| | **dijkstra** | 1 | 2 | 2 | 4 | 9 | 29 | 102 | 1 | 2 | 2 | 3 | 8 | 23 | 97 |
| | **fft** | 38 | 57 | 94 | 185 | 375 | - | - | 54 | 80 | 132 | 237 | 458 | - | - |
| | **stringsearch** | 37 | 37 | 39 | 43 | 49 | 67 | 120 | 49 | 48 | 50 | 56 | 61 | 80 | - |
| | **susancorners** | 46 | 47 | 51 | 58 | 81 | 164 | - | 42 | 44 | 49 | 58 | 82 | 166 | - |
| | **susanedges** | 125 | 160 | 203 | 361 | 376 | 522 | - | 91 | 119 | 149 | 185 | 204 | 443 | - |
| | **susansmoothing** | 52 | 53 | 60 | 91 | - | - | - | 35 | 37 | 41 | 59 | - | - | - |
| | **water-spatial** | 502 | 512 | 529 | 591 | - | - | - | 752 | 777 | 835 | 938 | - | - | - |
| Multi | **multi_p** | - | - | 36 | 91 | 185 | 785 | 2279 | - | - | 45 | 90 | 191 | 428 | 1014 |
| | **multi_office_telecomm** | - | - | 250 | - | - | - | - | - | - | 147 | - | - | - | - |
| | **multi_net_automotive** | - | - | 118 | - | - | - | - | - | - | 140 | - | - | - | - |
| | **multi_security** | - | - | 471 | - | - | - | - | - | - | 581 | - | - | - | - |
| | **multi_8** | - | - | - | 994 | - | - | - | - | - | - | 837 | - | - | - |
| | **multi_16** | - | - | - | - | 866 | - | - | - | - | - | - | 938 | - | - |

Table 4.9: Average simulation time, in seconds, for three distinct execution of each software on each platform using the NoC-AT, PowerPC and MIPS processors

| Applications | | PowerPC | | | | | | | MIPS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 6 | 14 | 33 | 78 | 205 | 510 | 1472 | 5 | 11 | 21 | 54 | 138 | 394 | 1531 |
| | **lu** | 69 | 105 | 165 | 404 | 884 | - | - | 50 | 78 | 123 | 276 | 622 | - | - |
| | **sha** | 1 | 15 | 34 | 96 | 287 | 1063 | 1169 | 1 | 15 | 35 | 97 | 296 | 1090 | 4905 |
| | **water** | 166 | 205 | 319 | 483 | 2851 | - | - | 168 | 193 | 299 | 466 | 2059 | - | - |
| | **dijkstra** | 5 | 6 | 11 | 27 | 82 | 244 | 1022 | 5 | 6 | 12 | 25 | 64 | 247 | 1019 |
| | **fft** | 96 | 165 | 265 | 569 | 1176 | - | - | 73 | 130 | 206 | 415 | 871 | - | - |
| | **stringsearch** | 34 | 35 | 36 | 40 | 47 | 64 | 125 | 51 | 50 | 51 | 57 | 65 | 80 | 128 |
| | **susancorners** | 27 | 28 | 31 | 36 | 56 | 143 | - | 29 | 30 | 36 | 37 | 54 | 129 | - |
| | **susanedges** | 56 | 71 | 90 | 117 | 139 | 314 | - | 59 | 77 | 85 | 111 | 121 | 270 | - |
| | **susansmoothing** | 168 | 168 | 186 | 277 | - | - | - | 33 | 35 | 36 | 53 | - | - | - |
| | **water-spatial** | 398 | 406 | 451 | 480 | - | - | - | 348 | 347 | 361 | 400 | - | - | - |
| Multi | **multi_p** | - | - | 34 | 79 | 193 | 900 | 4041 | - | - | 31 | 86 | 234 | 380 | 3743 |
| | **multi_office_telecomm** | - | - | 555 | - | - | - | - | - | - | 1064 | - | - | - | - |
| | **multi_net_automotive** | - | - | 459 | - | - | - | - | - | - | 436 | - | - | - | - |
| | **multi_security** | - | - | 2837 | - | - | - | - | - | - | 2634 | - | - | - | - |
| | **multi_8** | - | - | - | 3990 | - | - | - | - | - | - | 4107 | - | - | - |
| | **multi_16** | - | - | - | - | 10861 | - | - | - | - | - | - | 7533 | - | - |

Table 4.10: Average simulation time, in seconds, for three distinct execution of each software on each platform using the NoC-AT, SPARC and ARM processors

| Applications | | SPARC | | | | | | | ARM | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **01** | **02** | **04** | **08** | **16** | **32** | **64** | **01** | **02** | **04** | **08** | **16** | **32** | **64** |
| Cooperative | **basicmath** | 6 | 15 | 26 | 65 | 133 | 258 | 1082 | 4 | 10 | 20 | 53 | 129 | 345 | 1197 |
| | **lu** | 85 | 126 | 197 | 397 | 641 | - | - | 179 | 262 | 437 | 1001 | 2347 | - | - |
| | **sha** | 1 | 17 | 34 | 93 | 287 | 1080 | 5540 | 2 | 15 | 47 | 98 | 293 | 1070 | 5462 |
| | **water** | 152 | 179 | 265 | 406 | 1590 | - | - | 448 | 542 | 852 | 1352 | 4867 | - | - |
| | **dijkstra** | 5 | 7 | 9 | 21 | 59 | 185 | 805 | 5 | 6 | 11 | 25 | 65 | 207 | 938 |
| | **fft** | 120 | 197 | 320 | 613 | 1279 | - | - | 224 | 371 | 643 | 1338 | 2183 | - | - |
| | **stringsearch** | 9 | 13 | 24 | 50 | 114 | 308 | 1065 | 134 | 163 | 199 | 285 | 430 | 766 | 1773 |
| | **susancorners** | 143 | 152 | 192 | 249 | 458 | 1676 | - | 134 | 152 | 194 | 258 | 444 | 1289 | - |
| | **susanedges** | 315 | 482 | 660 | 906 | 1160 | 2143 | - | 347 | 506 | 825 | 1821 | 2129 | 2743 | - |
| | **susansmoothing** | 162 | 167 | 212 | 375 | - | - | - | 130 | 179 | 188 | 330 | - | - | - |
| | **water-spatial** | 1055 | 1147 | 1301 | 1582 | - | - | - | 2328 | 2902 | 4280 | 5465 | - | - | - |
| Multi | **multi_p** | - | - | 126 | 269 | | 1456 | 3981 | - | - | 194 | 448 | 1060 | 2694 | |
| | **multi_office_telecomm** | - | - | 984 | - | - | - | - | - | - | 682 | - | - | - | - |
| | **multi_net_automotive** | - | - | 590 | - | - | - | - | - | - | 673 | - | - | - | - |
| | **multi_security** | - | - | 1977 | - | - | - | - | - | - | 2977 | - | - | - | - |
| | **multi_8** | - | - | - | 4441 | - | - | - | - | - | - | 4048 | - | - | - |
| | **multi_16** | - | - | - | - | 6334 | - | - | - | - | - | - | 6171 | - | - |

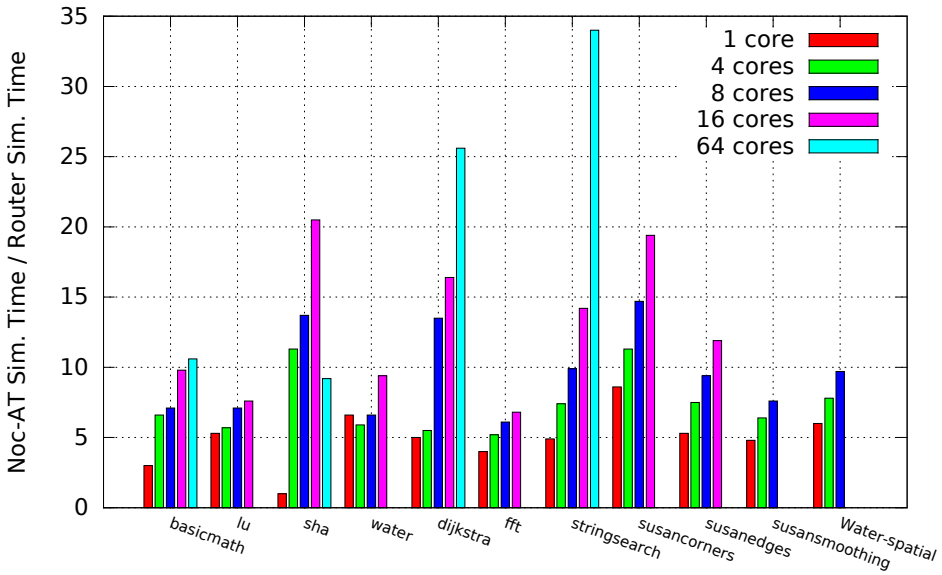



Figure 4.11: Relation between simulation time using NoC-AT and Router

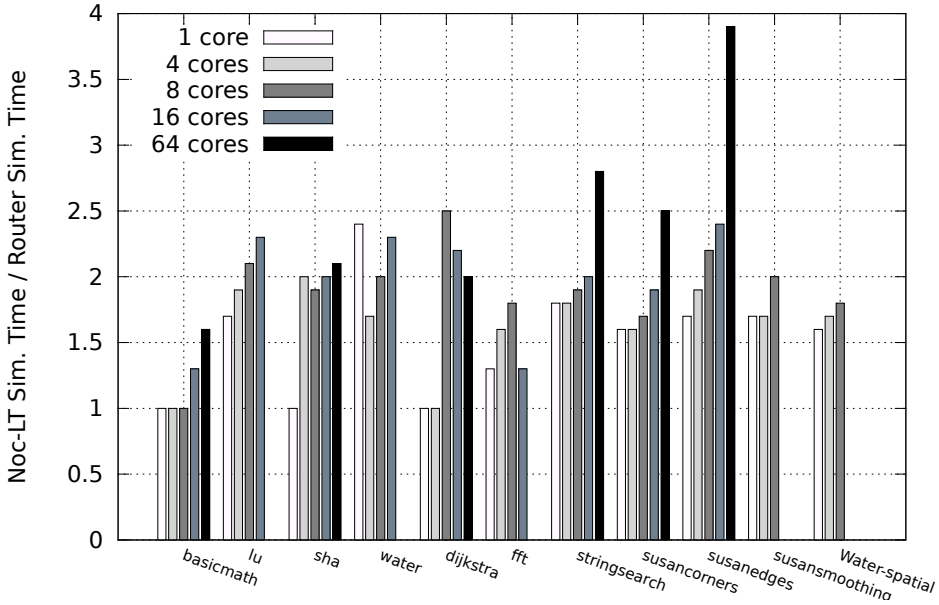



Figure 4.12: Relation between simulation time using NoC-LT and Router

Table 4.11: Most appropriate interconnections for each MPSoCBench use case

| Issues | Router | NoC-LT | NoC-AT |
|---|---|---|---|
| Caches and Memories studies | ✔ | ✔ | |
| DFS studies | ✔ | ✔ | |
| Hardware architectural analysis | | ✔ | ✔ |
| Hardware functional verification | ✔ | ✔ | ✔ |
| Hardware performance verification | | | ✔ |
| Instructions Throughput | ✔ | ✔ | |
| Network routing algorithms | | ✔ | ✔ |
| Network traffic | | | ✔ |
| Processor Optimizations | ✔ | ✔ | |
| Processor Power Evaluation | ✔ | ✔ | |
| Software Development | ✔ | ✔ | |
| Software performance analysis | ✔ | ✔ | |
| System Power Evaluation | | | ✔ |
| Timing Accuracy | | | ✔ |

### 4.5.2 How to improve performance

SystemC has a simulation kernel based on discrete events. It manages the simulation in a single core environment, selecting each process to execute until it finishes or until it gives the control to the kernel (in a non-preemptive management). So, the simulation time increases as we increase the number of modules or system components. Our experiments show that SystemC kernel uses about 50% of the total time scheduling processes, and this fact explains why 64-cores simulators run significantly slower than other simulators with fewer cores.

There are several opportunities to solve this performance issue:

- Running several simulators concurrently in a cluster, which is the alternative that matches perfectly with MPSoCBench.
- Simulating SystemC threads in parallel in a multicore environment, modifying the SystemC kernel to create a parallel simulation based on discrete events. We explored this approach in [35]. Although effective, this technique has a significant inconvenient: when we modify the kernel, there is no guarantee that models implemented according to the regular kernel run over the new one properly. Because of that, the community tends to be conservative and do not accept kernel modifications easily.
- Running simulations in clusters or multicore environment encapsulating components in a Linux process that can be scheduled over distributed cores; this technique is detailed in [44] and has the main advantage is that there is no need to modify the SystemC Kernel or models.

We consider that MPSoCBench is a good simulation infrastructure to evaluate any of these techniques since it has a large and scalable set of SystemC threads and processes, and a significant set of applications to explore them.

## 4.6 Power Estimation Evaluation

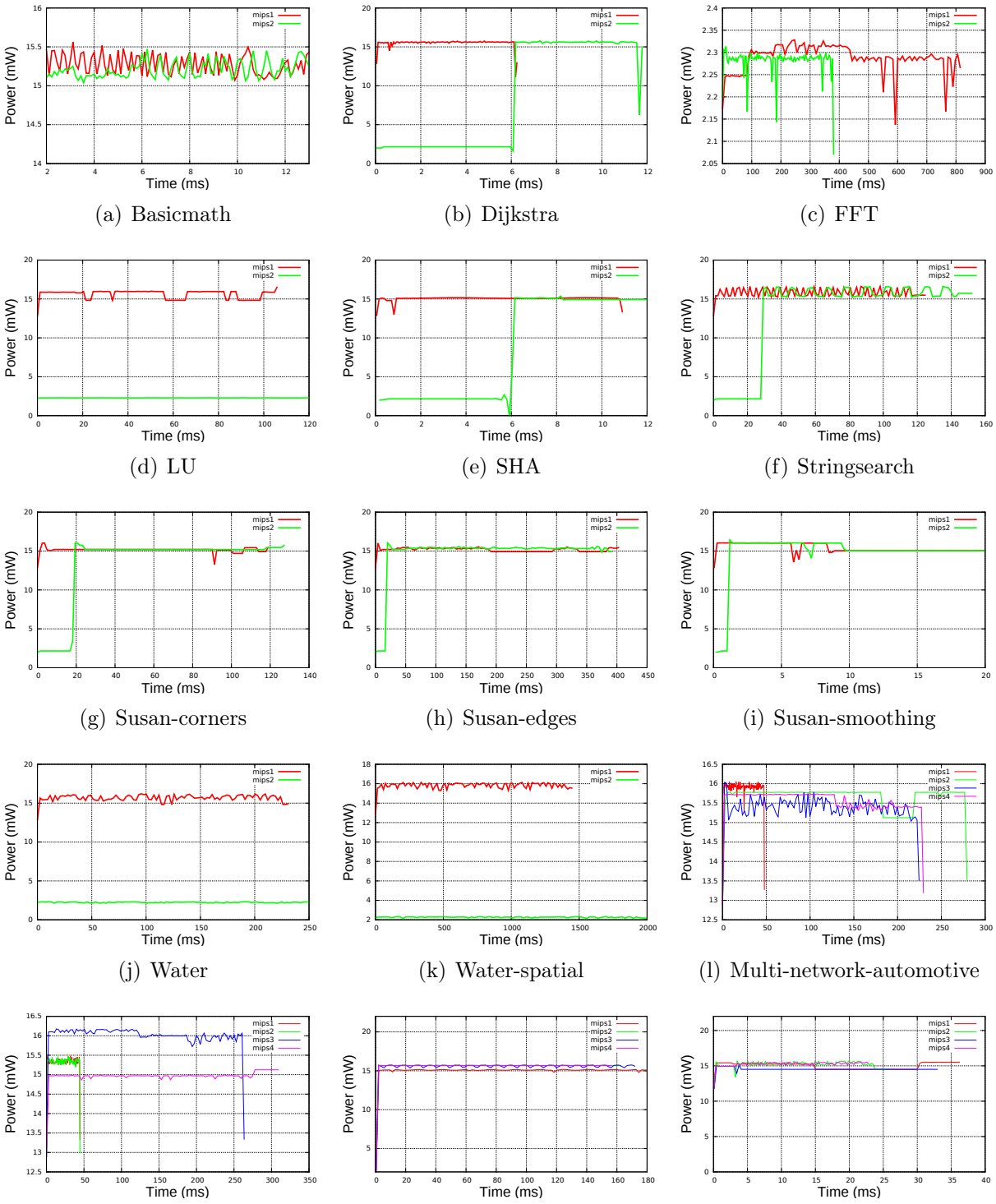
ArchC has power models for SPARC and MIPS processors validated in [51], which were implemented based on the open source LEON3 and Plasma RTL models using PowerSC [60] and acPower [51]. We integrate these power models into the MPSoCBench platforms to summarize the power consumption in a multicore environment.

Figure 4.13 shows power profiles generated by MPSoCBench running 15 applications in dual-core and quad-core platforms. These power profiles allow designers to optimize both software and hardware during early design stages. All graphics show only the processor power consumption, so we can analyze the phases of the processor without considering the network or caches. We set the processor frequency to 400Mhz without DVFS and the parameter START_WINDOW_SIZE with 10000 in the power configuration file, which means that the power is summarized each 10000 instructions. We do not show graphics with more than four cores because there is an overlap of lines, which do not allow good visualization and analysis of results.

It is possible to observe different phases performed by each core. For example, the large number of memory accesses to allocate and initialize the input variables causes the clear difference in power consumption of the two cores in the Figure 4.13(e). This setup phase is performed by one of the cores while the other is waiting in a barrier. The same behavior can be observed in several applications (Figures 4.13(b), 4.13(f), 4.13(g), 4.13(h), among others).

In contrast, the power consumption in Figure 4.13(a), which represents the behavior of the Basicmath application, reflects a large number of mathematical operations carried out over a small statically allocated amount of data.

Besides the average power, we can measure the platform energy consumption, which consist of a important metric for system energy efficiency evaluation. We estimated the




(a) Basicmath (b) Dijkstra (c) FFT

(d) LU (e) SHA (f) Stringsearch

(g) Susan-corners (h) Susan-edges (i) Susan-smoothing

(j) Water (k) Water-spatial (l) Multi-network-automotive

(m) Multi-office-telecomm (n) Multi-security (o) Multi-parallel

Figure 4.13: Power estimation in MIPS dual-core and quad-core platforms running 15 applications

energy consumed by caches, routers, and processors (Section 3.4) in 2-, 8, and 16-core MIPS platforms using the NoC-AT as interconnection running twelve parallel applications based on cooperative parallelism, and we showed the results in Figure 4.14. We used a 2-way write-through instruction cache with 64 blocks, 8 words per block, and a 2-

way write-through data cache with 64 blocks, 8 words per block, both comply with the FIFO replacement policies. We used only the power estimation for processors, caches and routers, omitting the main memory power consumption.

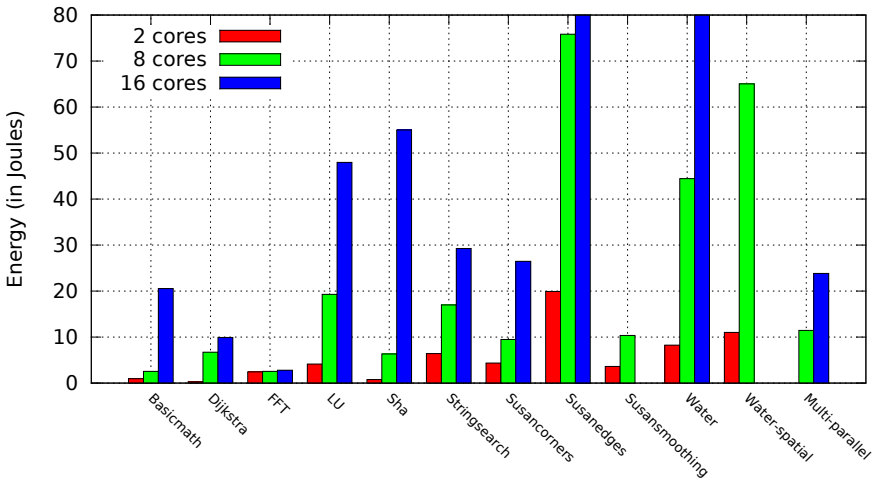



Figure 4.14: Energy consumed (in Joules) in 2- and 8-MIPS platforms connected through NoC-AT, with instruction and data cache per core

By using pre-characterized power tables, MPSoCBench allows designers to change devices and frequency easily, by simply exchange the power table file. Figure 4.6 shows MPSoCBench power estimation data from distinct FPGA technologies (Xilinx SPARTAN-3 1200e XC3S1200, Xilinx SPARTAN-6 XC6SLX75, and Altera CycloneV) running the same software on single MIPS platforms. The power information in this graphic refers only the power consumption of the processor during the simulation using three characterization technologies. We show graphs only to confirm the estimation coherence even when we vary the characterization technology.

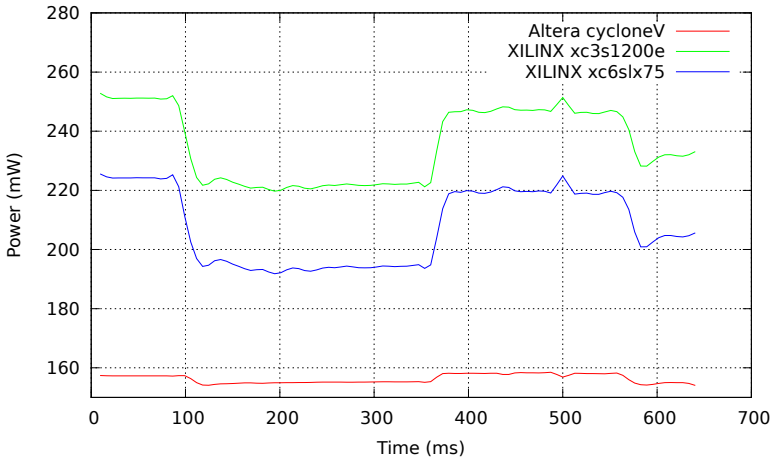



Figure 4.15: Single-MIPS platform running FFT at 100MHz at three distinct FPGAs: Xilinx SPARTAN-3 1200e XC3S1200, Xilinx SPARTAN-6 XC6SLX75 and Altera CycloneV

Introducing power estimation impacted the simulator performance. The average slowdown simulating all applications in multicore platforms is up to 5% in single core platforms to 25% in 64-core platforms.

## 4.7 Other Usage Scenarios

As the MPSoCBench full source code was released, we expect to grow a community of researchers taking advantage of this infrastructure. We list in the next paragraphs possible usage scenarios of MPSoCBench.

**Accelerators Design:** Scalability is always an issue, but new cores may not be enough to provide the necessary speedup for a new system. In such scenario, accelerators could become the solution. By having a configurable network-on-chip, processor models could be exchanged for specific accelerators. Designing and evaluating such accelerators is easier with the MPSoCBench infrastructure.

**Memory Organization:** Non-uniform memory architectures could be evaluated on larger platforms like the 64-core configurations. Memory and last level cache placement could also be evaluated.

**New Interconnection Structures:** Designing a new interconnection requires a big infrastructure to evaluate parameters, dimensions, performance, power, and so on. MPSoCBench scalability to 64-cores makes it easy to evaluate such designs. By changing the software or using one of the multi-software configurations, designers could easily evaluate quality of service. MPSoCBench allows NoC evaluation with real traffic.

**Heterogeneous platforms and DVFS Support:** We designed MPSoCBench to have both small and big instances so that new trends and technologies could be easily explored in the small instances while being validated in the big ones. The execution script also allows using a cluster of computers to spread the simulation so that the total execution time is reduced. This resource opens up opportunities on big-LITTLE architectures, and ESL tools exploration. MPSoCBench allows designers to build platforms with multiple types of processing elements, each able to perform tasks that it is best suited for. It is also possible to use different ISAs in the same MPSoC, with up to 64 cores, in which each core performs an independent application or cluster cores to cooperate in the same application.

# Chapter 5

# Dynamic Voltage and Frequency Scaling

Energy consumption constraints have become a critical issue in MPSoC design. Whereas processor performance comes with a high power cost, there is an increasing interest in exploring the trade-off between power and performance, taking into account the target application domain.

Dynamic Voltage and Frequency Scaling (DVFS) techniques adaptively scale the CPU frequency level or voltage supply allowing it to reach just enough performance to process the system workload while meeting throughput constraints and, thereby, reducing the energy consumption. To explore this wide design space for energy efficiency and performance, both for hardware and software components, a system-level simulation infrastructure must provide features to evaluate power savings mechanisms in early design stages [52].

Unfortunately, summarizing the exact energy consumption of a system depends on the final chip layout. Based on this premise, traditional methods require hardware simulation at the gate level, which is only available at the end of the design flow. Although accurate, this methodology needs long synthesis and simulation time, making it unfeasible for early-stage evaluations. To overcome this limitation, a fast methodology is required to extract the energy consumption of processors and other components on system level simulators and enable designers to explore energy as early as possible.

This chapter presents the DVFS support in the MPSoCBench, including power model for the principal components, and evaluates DVFS mechanisms based on energy estimation and CPU-usage rate. The results show that using the DVFS mechanism based on the energy consumption or the CPU workload metrics, can save energy with insignificant loss of performance. More specifically, the main contributions of this part of the thesis are the following:

- To validate dynamic voltage and frequency scaling techniques, we added the notion of voltage and frequency domains to MPSoCBench, as well as the simulation infrastructure that manage frequency scaling;
- We extended the MPSoCBench platforms with a DVFS IP for global power control and a local DVFS controller per core, capable of switching the core frequency;

- As a consequence of frequency switching, we ensured that the correct power data are used to estimate the consumed energy per-instruction;
- We included power estimation for Data and Instruction Caches, as well as we applied a power model for NoCs based on the RTL-NoC;
- We proposed and evaluated three DVFS techniques;
- We showed that these extensions can be used to run accurate full-system power efficiency studies while not significantly compromising the simulator performance;

This chapter is organized as follows: Section 5.1 shows the overall infrastructure of the MPSoCBench to support DVFS; Sections 5.2, 5.3 and 5.4 describe three DVFS techniques implemented, Section 5.5 evaluates these techniques using the MPSoCBench, and Section 5.6 concludes the chapter.

## 5.1 Overall Energy Infrastructure

DVFS is commonly implemented through *power states* (C-States and P-States). C-states are idle states and P-states are operational states, both including the frequency and voltage that the CPU operates. Although the MPSoCBench DVFS infrastructure supports both frequency and voltage scaling, available input power tables contain only frequency variations using FPGAs and ASICs. Thus, all experiments we present in this text use only frequency changes. It is part of our future work to include power tables considering changes in both power domains and also encourage the community to do so as well.

By extending MPSoCBench with DVFS, we provide the flexibility to assign clock frequency and voltage to the system. From the hardware perspective, we introduce a new global DVFS IP to manage the P-State in which each core operates, and local DVFS controllers (per-core) to manage the power state information for each core. A power state encapsulates energy per instruction considering frequency and voltage domain. Figure 5.1 show the system architecture, including the per-core DVFS controller and the global DVFS IP.

The global DVFS IP is initialized with the available C-States and P-States for all cores, and it can manage them globally. It is in the system address space and allows any core to change the state of another core in the system. To perform this mechanism, the global IP sends messages to the local per-core controller that ultimately changes the core's power state. Applications can interact with the global DVFS IP through the `pthread_changePowerState()` acPThread method.

Each per-core DVFS controller has information about all available power states, and is the responsible for performing the frequency switching activities. Each power table contains:

- Number and description of all available power states (Frequency, Voltage, Scales);
- Description of the platform used in the characterization process;

- The instantaneous power in *idle mode* for each state;
- The instantaneous power for each instruction in the ISA, in every power state;

Once a processor changes its power state, the proper power information in this table is used to compute the average power and also energy consumption of the core.

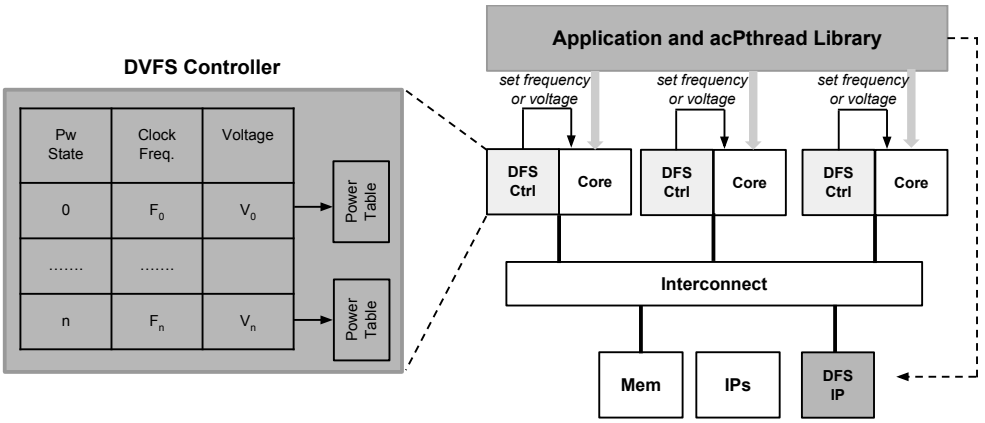



Figure 5.1: The System Architecture including the DVFS controllers

Each processor model inherits from the ArchC `ac_module` class a method to set the frequency. The `set_proc_freq` method receives the frequency in Hertz, calculates the clock period and uses this value to update the simulated time after executing one or more instructions. The amount of instructions executed without synchronization depends on the TLM2 time *quantum* adopted. When we do not offer these parameters, ArchC modules use default values (200Mhz frequency, 5ns clock cycle, and synchronization every 100ns). However, when we use power estimation in the MPSoCBench, the frequency of each core will be initialized using power tables mentioned earlier.

The global IP contains each core power states information (frequency, voltage, energy per instruction). The per-core DVFS controllers include the same information related to the core that are connected. There are two ways to switch the power state of each core: the global IP can change frequency globally, or each core can switch its frequency through an auto-selection mechanism.

## 5.2 DVFS controlled by Software (DVFS-SW)

*Power gating* is a commonly used technique to reduce leakage by putting unused circuits in a low power state. Since this technique incurs energy overhead, the idle state of these circuits needs to be long enough to compensate the overhead [19, 50]. Selecting the most appropriate time for changing frequency is the key point of this technique.

The switching activities can be managed at the appropriate time, producing the end effect of minimizing energy while maintaining the adequate performance of the overall design. Modern operating system engineers are taking advantage of that to supply the OSs with better policies and algorithms for power gating [47]. This section describes a software based power gating technique for parallel applications.

Almost all MPSoCBench available applications are based on cooperative parallelism over a shared memory parallel system, using PThread statements. Therefore, we introduce DVFS methods inside the *acPThread* library to allow power state switching managed by software. The basic execution flow works as follows: one of the processors acts as *master* and send jobs to the other cores (*workers*). While the master is reading input files and preparing arguments, workers stay in a barrier, waiting for jobs; So, in this stage the workers run in low power mode. When there are new jobs, the workers switch to a high power mode immediately before starting the thread code. Figure 5.2 shows the flow from the master and workers perspectives. The red line in the figure correspond to the interaction between master and workers, with the master indicating to the workers that there are jobs to perform.

The average power improvement varies between 0,25% to 65,24%, and 15,9% maximum energy saving in applications based on cooperative parallelism, in which the flow presented in Figure 5.2 matches perfectly. Section 5.5 presents the results of the average power and energy savings for several applications.

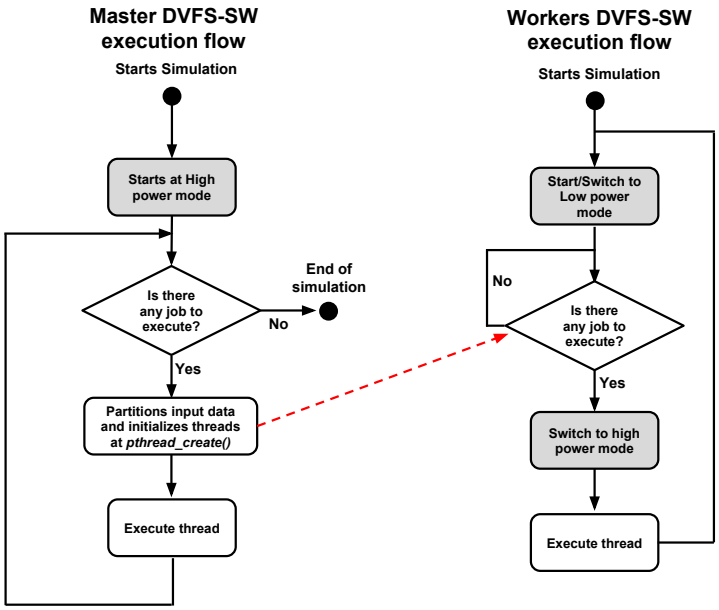



Figure 5.2: Master and workers switching activities

## 5.3 DVFS based on the energy consumption (DVFS-ES)

Excessive energy consumption affects microprocessor properties like battery life, temperature, and performance, especially in mobile/embedded computing environment where the energy/power consumption is a critical factor [61].

Considering both energy and delay of the applications, we propose an energy management technique, which leverages on the trade-off between energy/power consumption and performance.

We introduce the idea of DVFS auto-selection mode in which the best frequency is selected by the per-core DVFS controllers, without software management. The key idea is to create the infrastructure to implement new DVFS mechanisms and techniques, without the need of knowing the ArchC processor details. This mechanism is based on the energy consumption profiling at the beginning of the simulation. We adopted the model described bellow:

- The system has $n$ cores;
- Each core has $m$ power states (or $m$ different power tables obtained in a previous characterization process, one power table per available frequency/voltage in the selected ASIC/FPGA technology);
- We established the SWITCHING_TIME=20 $\mu s$ as the latency for a frequency transition (based on the previous work in [90]);
- Each power table has the EPI (*Energy per Instruction (nJ)*) value for each available instruction of the core instruction set;
- For each processor we define $F = \{f_0, f_1, ..., f_m\}$ as the set of its available frequencies;
- $\Delta_T$ is a time step used in the *evaluation phase* to evaluate the energy consumption behavior of cores;
- $Instr_{\Delta_T}$ is the amount of instructions executed in $\Delta_T$ units of time

**Initialization Phase:** Before starting the simulator, the power information for each profile of each core is stored in the DVFS data structures and the frequency of each processor in the platform is initialized, by default, to the first available frequency.

**Evaluation Phase:** Each core simulator runs for $\Delta_T$ units of time at each frequency $f$, and calculates the *Energy Stamp* ($ES$) metric as follows:

$$ES_f = \frac{(\sum_{\Delta_T} EPI)}{Instr_{\Delta_T}} \tag{5.1}$$

The $\sum_{\Delta_T} EPI$ is the sum of the energy (in Joules) in $\Delta_T$ units of time. The ES metric is based on the Energy Delay Product (EDP), which is an important metric for evaluating energy saving widely used since defined in [55].

At the end of the evaluation phase, the DVFS controller of each core selects the frequency $f$ as the best choice based on the minimum $ES$ as follows:

$$ES_f = min\{ES_j\}, 1 \leq j \leq m \tag{5.2}$$

**Execution Phase:** Energy consumption for each core can change during simulation time based on the executed instructions and memory/IO delays. Therefore, the choice of

$\Delta_T$ is relevant to improve the mechanism efficiency. Most of the MPSoCBench applications have hundreds of millions of instructions, and we can explore a significant part of the application to make the best choice of frequency per core.

Applying DVFS-ES, the average power improvement varies between 1,8% to 85,42%, and 14,4% maximum energy saving in applications based on cooperative parallelism. The experiments are described in detail in Section 5.5.

## 5.4 DVFS based on the CPU usage rate (DVFS-CPU)

To explore DVFS techniques in parallel applications consisting of different workloads, we define an auto-selection and adaptive DVFS mechanism to achieve energy saving based on the CPU workload.

This DVFS technique explores core energy efficiency based on the dynamic evaluation of the CPU usage rate. This mechanism uses the following additional information:

- The system has $n$ cores;
- Each core has $m$ power states (or $m$ different power tables obtained in a previous characterization process, one power table per available frequency/voltage in the selected ASIC/FPGA technology);
- We established the SWITCHING_TIME=20 $\mu s$ as the latency for a frequency transition (based on the previous work in [90]);
- Each power table has the EPI (*Energy per Instruction (nJ)*) value for each available instruction of the core instruction set;
- For each processor we define $F = \{f_0, f_1, ..., f_m\}$ as the set of its available frequencies;
- For each available frequency $f_j$ $(0 \leq j < m)$, we define:
  - $L_j$ as a lower bound for the CPU usage related to the frequency $j$, and
  - $H_j$ as an upper bound for the CPU usage related to the frequency $j$;
- $max\_rate$: maximum CPU usage rate in $\Delta_T$ units of time
- $min\_rate$: minimum CPU usage rate in $\Delta_T$ units of time
- $\Delta_T$ is a time step used in the *evaluation phase* to evaluate the energy consumption behavior of cores;

In this mechanism, the initialization phase is identical to the *energy stamp* technique discussed earlier, adding the initialization of the lower and upper limits for each frequency available for each processor.

The metric that supports this mechanism is the CPU usage rate ($cpu\_rate$) during execution. For every $\Delta_T$ units of time, the DVFS controller calculates the CPU usage

rate based on the time waiting for memory operations (or another IP) ($cpu\_wait\_time$) and the current simulation time.

The waiting time is easily obtained by computing the difference between the time at the beginning of a request to the network ($t_{req}$) and the time that the request was answered ($t_{resp}$). This metric strongly correlates with the processor CPI.

We calculates the CPU wait time for each processor as follows: for each request $r$ done by the processor to the network, at the first $\Delta_T$ units of time, we define:

$$cpu\_wait\_time_{\Delta_T} = \sum_{w} (t\_resp_w - t\_req_w), \tag{5.3}$$

Therefore, the core usage rate ($cpu\_rate_{\Delta_T}$) is obtained based on the $\Delta_T$ and the $cpu\_wait\_time$:

$$cpu\_rate_{\Delta_T} = \frac{\Delta_T - cpu\_wait\_time_{\Delta_T}}{\Delta_T} \tag{5.4}$$

This mechanism avoids the Evaluation Phase. During the execution phase, every $\Delta_T$ units of time, the DVFS controller uses the $cpu\_rate$ to verify if it is necessary switch each processor frequency, following the basic steps:

- Consider that the core runs at frequency $j$;
- If $cpu\_rate$ is in the interval $L_j$ and $H_j$, do not change frequency;
- Otherwise, find the frequency $f$ in which the interval $L_f$ and $H_f$ contains the actual core usage rate ($cpu\_rate$);

Applying DVFS-SPU, the average power improvement varies between 0,45% to 38,04%, and 21,3% maximum energy saving in applications based on cooperative parallelism. The next section presents the experiment details.

## 5.5 DVFS Evaluation

This section shows experiments and results of the DVFS techniques previously presented. For these experiments, we choose power information obtained through the ASIC FreePDK 45nm technology at 50Mhz, 125Mhz, 250Mhz, and 400Mhz. We simulated the platform using different combinations of frequency with and without DVFS mechanisms and compared the average power and energy consumption. DVFS-SW represents the simulation with DVFS managed by software; DVFS-ES represents the simulation with DVFS based on Energy Stamp; the DVFS-CPU represents the simulation with the technique based on the CPU usage-rate. These three DVFS mechanisms were evaluated independently using the following parameters:

- **DVFS-SW**: this mechanism chooses between two different frequencies, which we call LOW (50Mhz) and HIGH (400Mhz). The master starts at 400Mhz, and the workers are maintained at 50Mhz while they do not have jobs to execute. Without

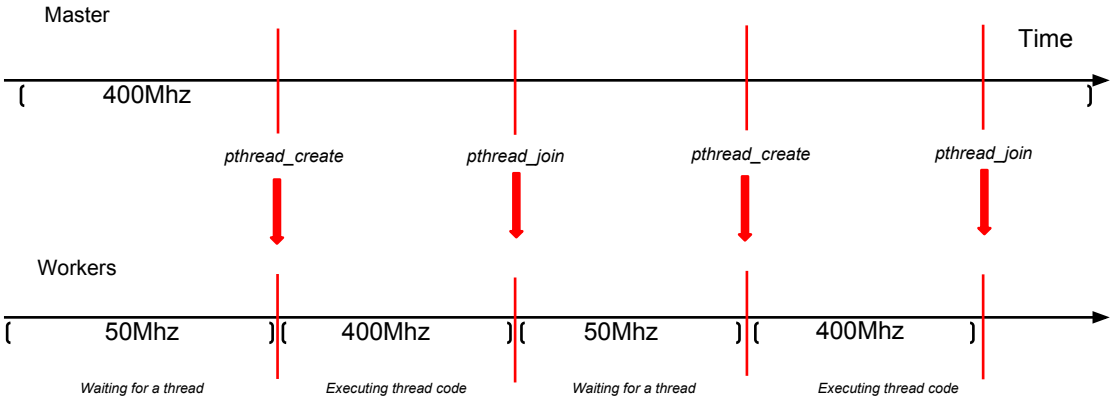



Figure 5.3: DVFS-SW example

DVFS, all cores are kept at 400Mhz all the time. Figure 5.3 illustrates the how the DVFS-SW works.

- **DVFS-ES**: this mechanism uses four different frequencies (50Mhz, 125Mhz, 250Mhz, and 400Mhz) using the energy stamp metric to choose the best of them for each application. We established $\Delta_T = 250\mu s$ to the evaluation phase and SWITCHING_TIME=20 $\mu s$ as the latency for a frequency transition. Figure 5.4 illustrates the how the DVFS-ES works.


Figure 5.4: DVFS-ES example

- **DVFS-CPU**: this mechanism uses four different frequencies (50Mhz, 125Mhz, 250Mhz, and 400Mhz) using the CPU usage rate metric to choose the best of them for each application. We started the lower and higher bounds for each frequency arbitrarily at:
  - 50Mhz: $L_{50} = 0\%$, $H_{50} = 23\%$
  - 125Mhz: $L_{125} = 20\%$, $H_{125} = 35\%$
  - 250Mhz: $L_{250} = 30\%$, $H_{250} = 60\%$
  - 400Mhz: $L_{400} = 50\%$, $H_{400} = 100\%$

In view of the wide variation in application workload, we implement the possibility of recalibrating the lower and upper bounds for each frequency during simulation, giving us more precision in selecting the best frequency for each core. So, we configure the DVFS-CPU to reevaluate the lower and upper bounds for each core every configurable $N$ $\Delta_T$ units of time. The DVFS controller uses the $cpu\_rate$, $max\_rate$ and $min\_rate$ to calibrate the lower ($L_j$) and upper ($H_j$) bounds for each available frequency $j$. Figure 5.5 illustrates the reevaluation method adopted in our experiments.

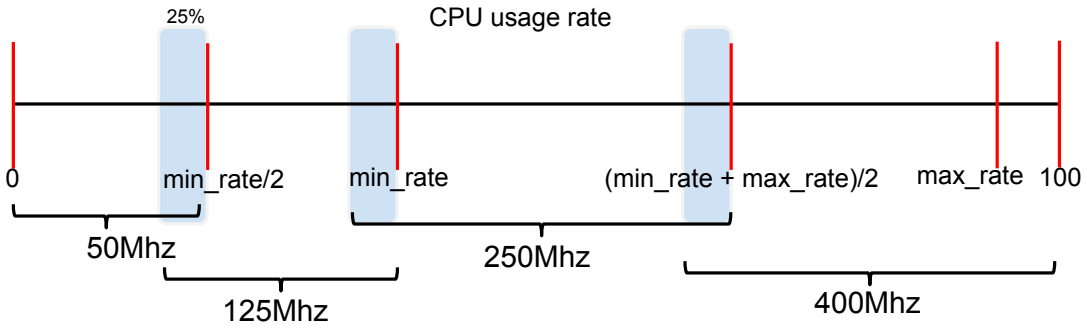



Figure 5.5: Reevaluation method to establish new CPU usage rate bounds

We used $\Delta_T = 250\mu s$ for all simulations, SWITCHING_TIME=20 $\mu s$ as the latency for a frequency transition, and we calibrated the DVFS-CPU to reevaluate the lower and upper bounds for each core every 100x$\Delta_T\mu s$ ($N = 100$). Table 5.1 shows mechanism to establish bounds for each processor and for each frequency available (50Mhz, 125Mhz, 250Mhz, 400Mhz):

Table 5.1: Mechanism to adaptively establish bounds for each power state.

| | |
|---|---|
| $L_{50}$ : | 0 |
| $H_{50}$ : | $min\_rate/2$ |
| $L_{125}$ : | $H_{50} - H_{50} * 25\%$ |
| $H_{125}$ : | $min\_rate$ |
| $L_{250}$ : | $H_{125} - H_{125} * 25\%$ |
| $H_{250}$ : | $(max\_rate + min\_rate)/2$ |
| $L_{400}$ : | $H_{250} - H_{250} * 25\%$ |
| $H_{400}$ : | 100 |

It is important not to vary the processor frequency constantly when the $cpu\_usage$ is close to the edge of two frequencies; so we configure the overlap of 25% between an upper bound of a frequency and the lower bound of the next frequency.

First, we executed all applications based on cooperative parallelism in 8-MIPS platforms based on the NoC-AT with and without these three DVFS techniques and showed in Table 5.2 the average power consumption improvement when we use each of them independently.

Table 5.2: Average power saving with DVFS

| **Applications** | **Average power saving** | | |
|---|---|---|---|
| | **DVFS-SW** | **DVFS-ES** | **DVFS-CPU** |
| Basicmath | 14,8% | 67,5% | 31,7% |
| Dijkstra | 33,3% | 41,7% | 20,4% |
| FFT | 4,8% | 85,4% | 2,0% |
| LU | 0,2% | 43,5% | 4,3% |
| SHA | 65,2% | 75,2% | 35,8% |
| Stringsearch | 45,6% | 63,6% | 17,3% |
| Susan-corners | 37,8% | 82,9% | 12,9% |
| Susan-edges | 52,8% | 65,3% | 3,6% |
| Susan-smoothing | 39,7% | 75,4% | 1,5% |
| Water | 0,1% | 0,3% | 0,4% |
| Water-spatial | 1,6% | 1,8% | 2,5% |
| Multi-parallel | 52,9% | 50,6% | 38,0% |

The improvement achieved is significantly greater in applications that explore cooperative parallelism, in which it is possible to take advantage of the several synchronization points to save energy. The reason for such low gains when we use FFT, LU, Water, and Water-spatial (all from SPLASH-2 [96] applications) is that more than 98% of the time is spent running the thread code in a HIGH power state, which does not take advantage of the power gating technique. We omitted results for Multisoftware applications, in which the processors run threads independently and there are no synchronizing points among them, the improvement is negligible (varying from 0,1% to 2%).

To clarify a possible confusion regarding the classification of applications, we observe that although the Multi-parallel application has four software running independently at the same time, each one of them performs tasks in a multithreaded and scalable environment from a cooperative perspective. Therefore, the Multi-Parallel application is considered in the Cooperative Group.

The graph in Figure 5.6 illustrates different phases using a dual-core MIPS platforms when using DVFS-SW. While the core 0 is preparing the inputs in a HIGH power state (400Mhz) at the beginning of the simulation, core 1 is in a barrier in a LOW power state (50Mhz). When the threads are ready to execute, both cores reach 400MHz while executing the thread code; at the end of the thread code, the core 1 switches frequency to 50Mhz and waits in a barrier for new jobs.

In the second evaluation approach, we choose the multi-parallel applications, which groups four multithreaded applications to assess more carefully how DVFS mechanisms behave when compared with several frequency combinations in 16-cores environments. In the simulations without DVFS, we manually initialized the cores using a combination of frequencies and disabled the DVFS mechanism. The notation MX/Y/Z/W means that we initialize manually (M) the cores using the X,Y,Z, and W frequencies as balanced as possible. For instance, in a 16-core platform, the notation $M50/125/250/400$ means that 4 cores were manually configured at 50Mhz, 4 cores at 125Mhz, 4 cores at 250Mhz and 4 cores at 400Mhz.

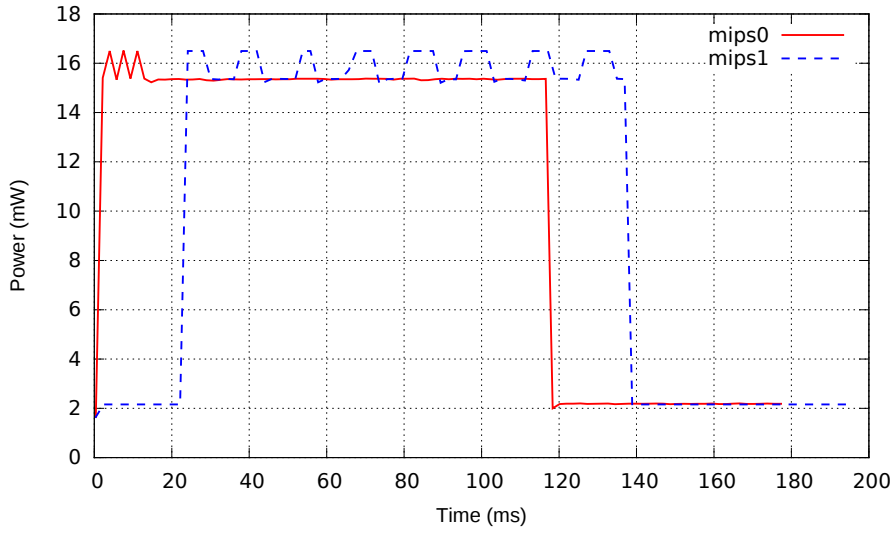



Figure 5.6: Power vs Time in a dual-MIPS platform running Stringsearch

We chose the MPSoCBench Multi-parallel application, which contains four parallel applications (Dijkstra, Basicmath, Sha, and Stringsearch) running simultaneously in a multithreaded environment. As we used a 16-core MIPS platform, each application used 4 threads, 1 thread per core.

We summarize the energy of all cores to execute Multi-parallel with and without DVFS on configuration and calculate the improvement when we use DVFS techniques. Table 5.3 presents our simulation results showing that DVFS-SW and DVFS-ES are better than 14 of 15 manual configurations, and DVFS-CPU is better than all manual configurations. Both DVFS-SW and DVFS-ES techniques could not achieve the manual configuration where all 16 cores run at 125Mhz all the time.

We believe that part of the energy gain are due to the fact that processors can remain in pooling depending on the running conditions (application concurrency, barriers, etc.). Processors at high frequency reach barriers earlier and consume energy while waiting for the processors at low frequency.

Figure 5.7 shows the Energy (J) and Time (seconds) normalized to the M125 energy and time.

As we showed in Section 2.3, the Energy Delay Product (EDP) metric is calculated multiplying the energy (in Joules) by the simulated time (in seconds). Figure 5.8 shows the EDP for simulations with and without DVFS (considering all aforementioned manual configurations), normalized to the M125 EDP.

Introducing power estimation with or without DVFS impacted the simulator performance. Table 5.4 shows the average slowdown comparing execution without power estimation, with power estimation (without DVFS) and with each DVFS mechanisms we have, in 16-core platforms.

In case of DVFS-CPU, simulator's performance decreases as we decrease the $\Delta_T$ and the amount of $\Delta_T$ cycles we reevaluate the bounds for each processor.

Table 5.3: Energy Savings with DVFS-SW

| **Manual frequency configuration** | **Energy Saving** | | |
|---|---|---|---|
| | **DVFS-SW** | **DFVS-ES** | **DVFS-CPU** |
| M125 | -0,2% | -1,9% | 6,2% |
| M50/250/400 | 4,3% | 2,6% | 10,4% |
| M50/M125 | 7,3% | 5,6% | 13,2% |
| M400 | 7,5% | 5,8% | 13,4% |
| M250/400 | 8,4% | 6,8% | 14,2% |
| M125/400 | 9,5% | 8,2% | 15,5% |
| M125/250/400 | 10,2% | 8,6% | 15,9% |
| M250 | 10,2% | 8,6% | 15,9% |
| M50/125/250/400 | 10,6% | 9,0% | 16,3% |
| M125/250 | 10,9% | 9,4% | 16,6% |
| M50/125/400 | 14,4% | 12,9% | 19,8% |
| M50 | 15,1% | 13,5% | 20,4% |
| M50/400 | 15,2% | 13,7% | 20,6% |
| M50/250 | 15,6% | 14,1% | 21% |
| M50/125/250 | 15,9% | 14,4% | 21,3% |

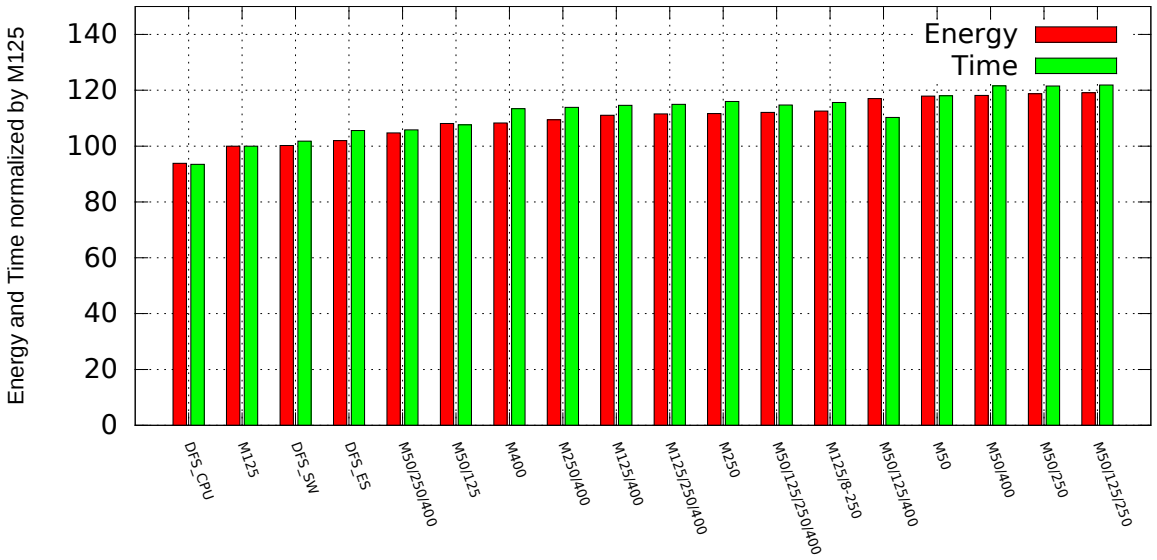



Figure 5.7: Energy and time of all simulated configurations for a 16-cores 4 applications with 4 threads each, normalized to the energy and time of M125.

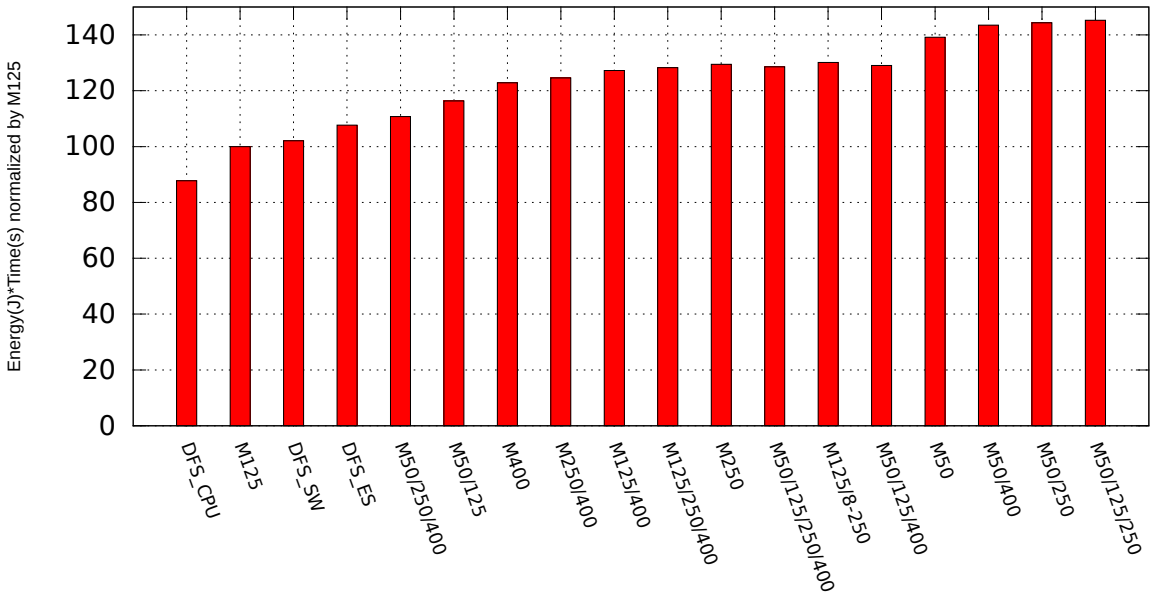



Figure 5.8: Energy Delay Product of all simulated configurations, normalized to the EDP of M125.

| **Simulation Slowdown** | |
|---|---|
| No power estimation *vs* power estimation | 10,2% |
| No power estimation *vs* DVFS-SW | 10,3% |
| No power estimation *vs* DVFS-ES | 10,7% |
| No power estimation *vs* DVFS-CPU | 11,2% |
| Power estimation without DVFS *vs* DVFS-SW | 0% |
| Power estimation without DVFS *vs* DVFS-EW | 3% |
| Power estimation without DVFS *vs* DVFS-CPU | 4% |

Table 5.4: Performance Slowdown

## 5.6 Conclusion

In this chapter, we described how MPSoCBench supports dynamic frequency scaling techniques in its system simulators. We introduced three different DVFS mechanisms, providing the simulation infrastructure to cover three important demands in energy efficiency studies: software managed DVFS, DVFS based on the energy consumption estimation, and DVFS based on the CPU workload.

We have validated the mechanisms using platforms of 8 and 16 cores, according to two metrics (average power and energy) and show that these extensions enable power efficiency evaluation in high-level system simulators at the earlier design stages keeping a small overhead: 3% for DVFS-ES and 4% for DVFS-CPU, negligible slowdown for DVFS-SW, when compared with the same simulations without DVFS; and 10,3% for DVFS-SW,

10,7% for DVFS-ES, and 11,2% for DVFS-CPU when we compared to simulators with no power estimation.

Applying DVFS, we could improve the average power for running all applications, with significant gains in applications that better explore cooperative parallelism. Comparing the proposed approaches with 15 simulations without DVFS configured manually with several frequencies combinations, it was possible to save energy in all of them using DVFS-CPU, and in 14 of them with DVFS-SW and DVFS-ES.

# Chapter 6

# Conclusion

This thesis introduced the MPSoCBench open-source toolset, which is a scalable, configurable, and extensible set of MPSoCs, useful to improve development and evaluation of the MPSoC ecosystem, using well-known methodologies and tools. MPSoCBench has a large set of platforms, each with a different interconnection device, easily configurable from 1 to 64 cores of 4 different available processor models, capable of running 17 different parallel applications. The total combined size reaches 864 distinct configurations, and this number can be even larger if we consider parameters to configure caches, power estimation, and DVFS algorithms.

We characterized the simulation time, timing, network traffic, caches and memory access, and power consumption for different configurations. The results demonstrated that the tool set is a viable alternative to evaluate MPSoC new tools and methodologies and to explore the design space of MPSoCs in different levels, making possible to compare the feasibility of an MPSoC design from the perspective of performance achieved and to estimate the cost considering physical parameters, like area for each component, manufacturing process, and power dissipation. The experiments also showed that the toolset presents architectural scalability while exploring a vast amount of parameters.

We also evaluated power models for hardware components of many-core systems and the support for dynamic frequency scaling techniques in the MPSoCBench. We introduced three different DFS mechanisms into its simulators, providing the simulation infrastructure to cover three important demands in energy efficiency studies: software managed DVFS, DVFS based on the energy consumption estimation, and DVFS based on the CPU workload.

We evaluated these mechanisms and showed that these extensions enable power efficiency evaluation in high-level system simulators at the earlier design stages keeping a small overhead on the simulator performance. Our software-based approach achieved up to 15.9% energy saving in platforms with 16 cores and obtained improvement in 14 of 15 manual configurations. We compared the other two auto-selection techniques using energy, time and EDP parameters. The auto-selection mechanism based on the Energy Stamp metric could save energy up to 14,4% in 14 of 15 manual configurations, decreasing performance by, at most, 3%; the DFS based on CPU usage rate reached the maximum of to 21,3% of energy improvement, and saved energy in all manual configuration with slowdown up to 4%.

We describe various scenarios in which we expect the usage of the MPSoCBench configurable and extensible simulation suite:

- Implementation and evaluation of new tools or methodologies in MPSoC designs, or comparisons between different tools in a similar environment;
- Development and monitoring design refinements for lower abstraction levels, which is an important need in the Electronic System Level designs;
- Evaluation and comparison of parallel applications using various techniques for parallelization and scalability characterization;
- Analysis and optimization of new hardware components, such as routers, buses, NoCs, IPs, and wrappers, in a co-design environment;
- Comparisons among different techniques for power consumption estimation, and dynamic characterization of program power consumption considering different power models;
- Power consumption analysis and energy efficiency optimization at the early project stages;
- Development and evaluation of multithreaded applications, regarding scalability and performance in different platform configurations;
- Evaluation of architectural parameters on variables directly used in the physical design of MPSoCs systems, like area, power dissipation, and clock frequency;

MPSoCBench is released under an open-source license, and they are available in two ways: virtual machines with all infrastructure ready for use and as source code. The tutorials for installation of all tools are provided in the MPSoCBench website (`www.archc.org/benchs/mpsocbench`).

## 6.1 Publications

This thesis resulted in the following publications:

- Duenha, Liana; Madalozzo, Guilherme; Santiago, Thiago; Moraes, Fernando Gehm; Azevedo, Rodolfo. Exploração de Desempenho, Consumo Dinâmico e Eficiência Energética em MPSoCs. XVI WSCAD-Simpósio de Sistemas Computacionais de Alto Desempenho. October, 2015 [42].
- Devigo, Rodrigo; Duenha, Liana; Azevedo, Rodolfo; Santos, Ricardo. MultiExplorer: A Tool Set for MultiCore System-on-Chip Design Exploration. Proceedings of 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP). September, 2015 [33].

- Duenha, Liana; Guedes, Marcelo; Boy, Matheus; Azevedo, Rodolfo. MPSoCBench: A Toolset for MPSoC System Level Evaluation. Proceedings of the Embedded Computer Systems: Architectures, Modeling, and Simulation 2014 IEEE International Conference. p.164-171. July, 2014 [39].

- Duenha, Liana; Guedes, Marcelo; Boy, Matheus; Azevedo, Rodolfo. MPSoCBench: A Toolset for MPSoC System Level Evaluation. Design Automation Conference - Work-in-Progress Session. DAC-WIP. San Francisco, June, 2014 [41].

- Duenha, Liana; Azevedo, Rodolfo. MPSoCBench: A Benchmark Suite for Evaluating Multiprocessor System-on-Chip Tools and Methodologies, Techical Report, Institute of Campinas - Unicamp - IC-13-19. June, 2013 [38].

- Duenha, Liana; Guedes, Marcelo.; Boy, Matheus; Azevedo, Rodolfo. MPSoCBench: A Benchmark Suite for Evaluating Multiprocessor System-on-Chip Tools and Methodologies. North American SystemC User's Group- NASCUG. June, 2013 [40].

We have one paper published in Journal:

- Guedes, Marcelo; Auler, Rafael; Duenha, Liana; Borin, Edson; Azevedo, Rodolfo. An automatic energy consumption characterization of processors using ArchC. Journal of Systems Architecture. June, 2013 [52].

And we have another paper submitted to the Journal of Parallel and Distributed Computing, Special Issue on Energy Efficient Multi-Core and Many-Core Systems (E2MC2).

- Duenha, Liana; Madalozzo, Guilherme; Santiago, Thiago; Moraes, Fernando; Azevedo, Rodolfo. MPSoCBench: a Benchmark for High-Level Evaluation of Multiprocessor System-on-Chip Tools and Methodologies. September, 2015.

In addition, during the development of this thesis, following other side-related papers were published:

- Falcão, Tiago; Duenha, Liana; Azevedo, Rodolfo. How run your Simulation in many cores without change neither the SystemC nor yours modules. XVI WSCAD-Simpósio de Sistemas Computacionais de Alto Desempenho, 2015 [44].

- Oliveira, Helder; Brito, Alisson V.; Melcher, Elmar U. K.; Bucher, Harald; Araujo, Joseana M. F. R.; Duenha, Liana. Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations. Proceedings of the 28th Symposium on Integrated Circuits and System Design (SBCCI). September, 2015 [77].

- Duenha, Liana; Azevedo, Rodolfo; Tendências para Aceleração de Simuladores SystemC de Sistemas Multiprocessados em Alto Nível de Abstração. Publicado nos Anais da II Escola Regional de Alto Desempenho de São Paulo (ERAD). Julho, 2012 [37].

- Duenha, Liana; Azevedo, Rodolfo; Profiling High-Level Abstraction Simulators of Multiprocessor Systems. Workshop of Circuits on System Design-WCAS. July, 2012 [36].

- Duenha, Liana; Azevedo, Rodolfo; Paralelização de Simuladores de Hardware Descritos em SystemC. Publicado nos Anais da II Escola Regional de Alto Desempenho de São Paulo (ERAD). Julho, 2011 [35].

## 6.2 Future Work

We visualize some opportunities for further work:

- **Performance Improvement with PDES:** Experiments show that the 50% of the simulation time is spent by the SystemC kernel, demonstrating that the scheduling is a very expensive task. We aim to improve the SystemC simulators performance through applying temporal decoupling and parallel discrete-event simulation (PDES), reducing the context switches between SystemC processes and taking advantage of the parallel computing resources.

- **Performance Improvement with Distributed Simulation**: There is work in progress to allow SystemC simulations in clusters encapsulating components in a Linux process that can be scheduled over distributed cluster cores. The main advantage of this approach is that it parallelizes SystemC-TLM2 simulators using the original SystemC Kernel and models. A large number of SystemC threads of the MPSoCBench simulators make it an appropriate benchmark to validate both techniques [44].

- **MPSoCBench for Design Space Exploration:** Although the design space exploration is not the main goal of the MPSoCBench, once integrated with McPAT (*Multicore Power, Area, and Timing*) [67], they create a fast high-abstraction simulator and low-level physical estimator (power, area, and timing). This infrastructure enables MPSoCs modeling, experimentation and design exploration, by taking a range of high and low-level parameters to improve accuracy on designing a multiprocessor system on a chip. By joining high-level modeling and fast simulators to the power, area, and timing models, we are trying to meet two primary goals: good design accuracy and simulation performance. The integration of MPSoCBench and McPAT results in a tool for design exploration called MultiExplorer [33], which are an ongoing project developed collaboratively with the research group of the High-Performance Computing Systems Laboratory (LSCAD-UFMS) [33].

  The main contribution of the MultiExplorer is to provide a new toolset for fast simulation of multiprocessor systems modeled in a high-level abstraction, useful for driving design space exploration in the earliest stages of the project, considering architectural parameters, performance, area, timing, and power.

- **Dark Silicon exploration:** The emergence of Dark Silicon introduces several challenges on the architecture, electronic design automation (EDA), and Hardware-Software co-design. Since dark silicon is dependent on the technological process, modern processors designs should identify the dark silicon area to explore architectural resources to mitigate it. There is work in progress using the MultiExplorer tool to dark silicon identification and area estimates. We envisage the implementation of new algorithms to design space exploration, enabling the use of architectural features on the dark silicon area while maintaining the physical constraints and project performance [87].
- **MPSoCBench new features:** We aim to improve MPSoCBench simulators with several new features and resources. We can cite:
  - Power estimation for ARM and PowerPC models;
  - Development of a distributed memory system;
  - MPSoC evaluation using real time constraints;
  - Provision of parallel applications based on distributed memory;
  - Development of mechanisms for inter-processors communication, which permits the use of more efficient cache coherency protocols
  - Development and evaluation of new DVFS techniques;
  - Development of a simple Operating System that encapsulates the acPthread and other system management features;
  - Development of a friendly platform configuration file;

# Bibliography


[1] Arm9e-s technical reference manual - revision 1, author=ARM Limited, year=2000, organization=ARM Limited doi = DDI 0165B, url = http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0165b/I1028171.html.

[2] OSCI TLM-2.0 Language Reference Manual. Software version: TLM 2.0.1 Document version: JA32.
http://www.systemc.org.

[3] IEEE Std 1666$^{TM}$ Standard SystemC Language Reference Manual. IEEE Computer Society, January 2012.

[4] A. Agarwal, S. Mukhopadhyay, C.H. Kim, Raychowdhury A., and K. Roy. Leakage power analysis and reduction: models, estimation and tools. In *IEE Proceedings - Computers and Digital Techniques*, volume 152, pages 353–368. IEE, May,2005.

[5] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42. IEEE, 2009.

[6] Alexandra Aguiar, Sergio Johann Filho, Felipe Gohring de Magalhaes, and Fabiano Hessel. Customizable RTOS to Support Communication Infrastructures and to Improve Design Space Exploration in MPSoCs. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 130–135, 2013.

[7] Abdelhakim Alali, Ismail Assayad, and Mohamed Sadik. Modeling and simulation of multiprocessor systems mpsoc by systemc/tlm2. 2014.

[8] Henrique Dante Almeida and Rodolfo Azevedo. Implementação de cache no projeto archc. In *Masters Thesis*. Unicamp, url=http://www.bibliotecadigital.unicamp.br/document/?code=000867041, 2012.

[9] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of Spring Joint Computer Conference*, pages 483–485. ACM, 1967.

[10] Federico Angiolini, Jianjiang Ceng, Rainer Leupers, Federico Ferrari, Cesare Ferri, and Luca Benini. An integrated open framework for heterogeneous mpsoc design space exploration. In *Proceedings of the conference on Design, automation and*

*test in Europe: Proceedings*, pages 1145–1150. European Design and Automation Association, 2006.

[11] Ehsan K Ardestani and Jose Renau. Esesc: A fast multicore simulator using time-based sampling. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 448–459. IEEE, 2013.

[12] Todd Austin, Eric Larson, and Dan Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[13] Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araujo, Cristiano Araujo, and Edna Barros. The ArchC Architecture Description Language and Tools. In *International Journal of Parallel Programming. Vol. 33, No. 5*, pages 453–484. October 2005.

[14] Fabrice Bellard. Qemu internals. *Jul*, 22:1–7, 2006.

[15] Giovanni Beltrame, Donatella Sciuto, and Christina Silvano. A power-efficient methodology for mapping applications on multi-processor, system-on-chip architectures. In *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pages 177–196. Springer, 2008.

[16] L. Benini, D. Bertozzi, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. In *Journal of VLSI Signal Processing*, volume 41, pages 169–182. 2005.

[17] 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. volume 39, pages 1–7. 2011.

[18] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, Kevin T Lim, Ali G Saidi, and Steven K Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, (4):52–60, 2006.

[19] S. Borkar. Design challenges of technology scaling. In *IEEE Micro*, volume 19, Issue: 4, pages 23–29. IEEE Computer Society, August 1999.

[20] Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.

[21] S. Boukhechem, E. Bourennane, and A. Samahi. Co-simulation Platform Based on SystemC for Multiprocessor System on Chip Architecture Exploration. In *Proceedings of IEEE International Conference on Microelectronics (ICM)*. 2007.

[22] Sami Boukhechem and El-Bay Bourennane. SystemC Transaction-Level Modeling of an MPSoC Platform Based on an Open Source ISS by Using Interprocess Communication. In *International Journal of Reconfigurable Computing*, volume 2008, Article ID 902653. 2008.

[23] Sami Boukhechem and El-Bay Bourennane. Tlm platform based on systemc for starsoc design space exploration. In *Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA Conference on*, pages 354–361. IEEE, 2008.

[24] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, may 2000.

[25] David M Brooks, Pradip Bose, Stanley E Schuster, Hans Jacobson, Prabhakar N Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *Micro, IEEE*, 20(6):26–44, 2000.

[26] Doug Burger and Todd M Austin. The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[27] Doug Burger, Todd M Austin, and Steve Bennett. *Evaluating future microprocessors: The simplescalar tool set.* University of Wisconsin-Madison, Computer Sciences Department, 1996.

[28] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy evaluation of gem5 simulator system. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–7. IEEE, 2012.

[29] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *Procs. of the Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. ACM, November 2011.

[30] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power consumption modeling for dvfs exploitation. In *Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, ISBN:978-0-7695-3407-7, pages 579–586. IEEE, Sept, 2010.

[31] Neil J Cornish, Louis J Rubbo, and Olivier Poujade. The lisa simulator. *Phys. Rev. D*, 69:082003, 2004.

[32] IBM International Business Machines Corporation. Powerpc 405-s embedded processor core - version 1.2. IBM, 2010.

[33] Rodrigo Devigo, Liana Duenha, Rodolfo Azevedo, and Ricardo Santos. Multiexplorer: A tool set for multicore system-on-chip design exploration. In *Proceedings of 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, Scheduled to September, 2015 2015.

[34] Jonathan Dorn, Robbie Hott, and Michelle McDaniel. Exploring Performance and Power Scaling in Multi-Core Processors.

[35] Liana Duenha and Rodolfo Azevedo. Paralelização de simuladores de hardware descritos em systemc. In *II Escola Regional de Alto Desempenho de São Paulo (ERAD)*, 2011.

[36] Liana Duenha and Rodolfo Azevedo. Profiling high-level abstraction simulators of multiprocessor systems. In *Workshop of Circuits on System Design (WCAS) - Chip In*, 2012.

[37] Liana Duenha and Rodolfo Azevedo. Tendências para aceleração de simuladores systemc de sistemas multiprocessados em alto nível de abstração. In *III Escola Regional de Alto Desempenho de São Paulo (ERAD)*, 2012.

[38] Liana Duenha and Rodolfo Azevedo. Mpsocbench: A benchmark suite for evaluating multiprocessor system-on-chip tools and methodologies. In *Technical Report – Institute of Computing (IC-UNICAMP)*, IC-13-19, page 29, 2013.

[39] Liana Duenha, Marcelo Guedes, Henrique Almeida, Matheus Boy, and Rodolfo Azevedo. Mpsocbench: A toolset for mpsoc system level evaluation. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, INSPEC Number: 14564763, pages 164–171. IEEE, July 2014.

[40] Liana Duenha, Marcelo Guedes, Matheus Boy, and Rodolfo Azevedo. Mpsocbench: A benchmark suite for evaluating multiprocessor system-on-chip tools and methodologies. *North American SystemC User's Group- NASCUG*, 2013.

[41] Liana Duenha, Marcelo Guedes, Matheus Boy, and Rodolfo Azevedo. Mpsocbench: A toolset for mpsoc system level evaluation. *Design Automation Conference - Work-in-Progress Session (DAC-WIP)*, June 2014.

[42] Liana Duenha, Guilherme Madalozzo, Thiago Santiago, Fernando G. Moraes, and Rodolfo Azevedo. Exploração de Desempenho, Consumo Dinâmico e Eficiência Energética em MPSoCs. In *Proceedings of the XVI Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD)*. October 2015.

[43] Fernando Endo, Damien Couroussé, Henri-Pierre Charles, et al. 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. IEEE, 2014.

[44] Tiago Falcão, Liana Duenha, and Rodolfo Azevedo. How run your Simulation in many cores without change neither the SystemC nor yours modules. In *Anais do Simpósio de Sistemas Computacionais de Alto Desempenho-WSCAD*. October 2015.

[45] Cesare Ferri, Tali Moreshet, R Bahar, Luca Benini, and Maurice Herlihy. A hardware/software framework for supporting transactional memory in a mpsoc environment. *ACM SIGARCH Computer Architecture News*, 35(1):47–54, 2007.

[46] Sergio Johann Filho, Alexandra Aguiar, César Augusto Missio Marcon, and Fabiano Hessel. High-Level Estimation of Execution Time and Energy Consumption for Fast Homogeneous MPSoCs Prototyping. In *IEEE International Workshop on Rapid System Prototyping*, pages 27–33, 2008.

[47] David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. Integrated Circuits and Systems.

[48] Aeroflex Gaisler. Ut699 leon 3ft/sparctm v8 microprocessor. In *Functional Manual*. Aeroflex Gaisler, url=www.aeroflex.com/LEON, 2014.

[49] A. Genser, C. Bachmann, C. Steger, and R. Weiss. Power emulation based dvfs efficiency investigations for embedded systems. In *Proceedings of the International Symposium on System on Chip (SoC)*, ISBN: 978-1-4244-8279-5, pages 173–178. IEEE, Sept, 2010.

[50] M. Ghosh and Georgia Institute of Technology. *Microarchitectural Techniques to Reduce Energy Consumption in the Memory Hierarchy*. Georgia Institute of Technology, 2009.

[51] M. Guedes, R. Auler, L. Duenha, E. Borin, and R. Azevedo. An automatic energy consumption characterization of processors using ArchC. In *Journal of Systems Architecture*. June 2013.

[52] Marcelo Guedes, Rafael Auler, Liana Duenha, Edson Borin, and Rodolfo Azevedo. An automatic energy consumption characterization of processors using archc. *Journal of Systems Architecture*, 59(8):603 – 614, 2013.

[53] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of IEEE 4th Annual Workshop on Workload Characterization, held in conjunction with The 34th Annual IEEE/ACM*, pages 03–14. December 2001.

[54] J.-P. Halimi, B. Pradelle, A. Guermouche, N. Triquenaux, A. Laurent, J.C. Beyler, and W. Jalby. Reactive dvfs control for multicore processors. In *Proceedings of the Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 102–109. IEEE, Aug, 2013.

[55] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8–11. IEEE, 1994.

[56] Ming-yu Hsieh, Arun Rodrigues, Rolf Riesen, Kevin Thompson, and William Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):63–68, 2011.

[57] Imperas. PlatformsOpen Virtual Platforms - the source of Fast Processor Models and Platforms. `http://http://www.ovpworld.org/`, 2008. [Online; accessed 10-September-2015].

[58] S.M.Z. Iqbal, Yuchen Liang, and H. Grahn. Parmibench - an open-source benchmark for embedded multiprocessor systems. *Computer Architecture Letters*, 9(2):45–48, Feb 2010.

[59] N. Kappiah, Vincent W. Freeh, and D.K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *Proceedings of the ACM/IEEE Supercomputing Conference, 2005*, Print ISBN: 1-59593-061-2, IEEE, Nov,2005.

[60] F. Klein, G. Araujo, R. Azevedo, R. Leao, and L. Santos. An Efficient Framework for High-Level Power Exploration. In *Proceedings of the 50th Midwest Symposium on Circuits and Systems - MWSCAS 2007*, pages 1046–1049. August 2007.

[61] Joonho Kong, Jinhang Choi, L. Choi, and Sung Woo Chung. Low-cost application-aware dvfs for multi-core architecture. In *Proceedings of 3rd International Conference on Convergence and Hybrid Information Technology*, volume 2 of *ISBN:978-0-7695-3407-7*, pages 106–111, Busan, Nov-2008. IEEE.

[62] Kevin P. Lawton. Bochs: A portable pc emulator for unix/x. *Linux J.*, 1996(29es), September 1996.

[63] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8. USENIX Association, 2010.

[64] R. Leupers and O. Temam. *Processor and System-on-Chip Simulation*. Springer New York, 2010.

[65] R. Leupers and O. Temam. *Processor and System-on-Chip Simulation*. Springer US, 2010.

[66] Dong Li, B.R. Supinski, M. Schulz, D.S. Nikolopoulos, and K.W. Cameron. Strategies for energy-efficient resource management of hybrid programming models. In *IEEE Transactions on Parallel and Distributed Systems*, volume 24 of *ISSN: 1045-9219*, pages 144–157, Jan, 2013.

[67] Sheng Li, JH Ahn, and RD Strong. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual International Symposium on Microarchitecture*, pages 469–480, 2009.

[68] Mirko Loghi, Massimo Poncino, and Luca Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 410–406. ACM, 2004.

[69] Imagination Technologies Ltd. and its affiliated group companies. Mips32 m5150 processor core family datasheet. Imagination Technologies Ltd. and its affiliated group companies, 2012.

[70] Yu-Sheng Lu, Chin-Feng Lai, and Yueh-Min Huang. Parallelization of dvfs-enabled h.264/avc decoder on heterogeneous multi-core platform. In *International Symposium on Computer, Consumer and Control (IS3C), 2012*, ISBN:978-1-4673-0767-3, pages 157–160. IEEE, Jun, 2012.

[71] Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[72] Shankar Mahadevan, Federico Angiolini, Jens Sparsø, Michael Storgaard, Jan Madsen, and Rasmus Grøndahl Olsen. A network traffic generator model for fast network-on-chip simulation. In *Design, Automation, and Test in Europe*, pages 173–184. Springer, 2008.

[73] Milo MK 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. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[74] A.L.M. Martins, D.R.G. Silva, G.M. Castilhos, T.M. Monteiro, and F.G. Moraes. A method for noc-based mpsoc energy consumption estimation. In *Electronics, Circuits and Systems (ICECS), 2014 21st IEEE International Conference on*, pages 427–430, Dec 2014.

[75] A. Mello, I. Maia, A. Greiner, and F. Pecheux. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations. In *Proceedings of Design, Automation and Test in Europe - DATE*, pages 606–609. March 2010.

[76] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. volume 38, pages 69–93. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, October 2004.

[77] H.F.A. Oliveira, L. Duenha, A.V. Brito, E.U.K. Melcher, Bucher H., and J.M.F.R. Araujo. Power-aware design of electronic system level using interoperation of hybrid and distributed simulations. In *Proceedings of the 28th Symposium on Integrated Circuits and System Design (SBCCI)*. ACM, September 2015.

[78] Pablo Montesinos Ortego and Paul Sack. Sesc: Superescalar simulator. In *17 th Euro micro conference on real time systems (ECRTS'05)*, pages 1–4, 2004.

[79] P. Paulin, Ch. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagné, and G. Nicolescu. Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and

Multimedia. In *Proceedings of IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 14, no.7. July 2006.

[80] Pierre G Paulin. Automatic mapping of parallel applications onto multi-processor platforms: a multimedia application. In *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pages 2–4. IEEE, 2004.

[81] Carlos A. Petry, Eduardo Wächter, Guilherme M. Castilhos, Fernando Gehm Moraes, and Ney Laert Vilar Calazans. A Spectrum of MPSoC Models for Design Space Exploration and its Use. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 30–35, 2012.

[82] Jonathan Power, Joel Hestness, Martin Orr, Mark Hill, and David Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. 2014.

[83] Lauro Rizzatti. Power trumps performance in today's soc designs. *Electronic Design Europe Magazine*, page 5, May 2013.

[84] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. In *IEEE-Computer Architecture Letters*, ISSN 1556-6056, pages 16–19, Maryland, USA, June 2011. IEEE.

[85] Gunar Schirner and Rainer Dömer. Quantitative analysis of the speed/accuracy trade-off in transaction level modeling. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(1):4, 2008.

[86] Hao Shen, Patrice Gerin, and Frédéric Pétrot. Configurable Heterogeneous MPSoC Architecture Exploration Using Abstraction Levels. In *IEEE International Workshop on Rapid System Prototyping*, pages 51–57, 2008.

[87] Ana Caroline Santos Silva, Tony Bignardi, Edilson Soares de Palma, Rafael Alves, Clara Hayashi, and Ricardo Santos. Identificação automática de Dark Silicon em Processadores Multicore. In *Anais do Simpósio de Sistemas Computacionais de Alto Desempenho-WSCAD*. October 2015.

[88] Cristina Silvano, William Fornaciari, Gianluca Palermo, Vittorio Zaccaria, Fabrizio Castro, Marcos Martinez, Sara Bocchio, Roberto Zafalon, Prabhat Avasare, Geert Vanmeerbeeck, et al. Multicube: Multi-objective design space exploration of multi-core architectures. In *VLSI 2010 Annual Symposium*, pages 47–63. Springer, 2011.

[89] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, New Jersey, USA, Sep 2005.

[90] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras. Introducing dvfs-management in a full-system simulator. *Proceedings of the IEEE 21st International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS-2013)*, pages 535–545, Aug 2013.

[91] Ching-Long Su, Chi-Ying Tsui, and Alvin M Despain. Low power architecture design and compilation techniques for high-performance processors. In *Compcon Spring'94, Digest of Papers.*, pages 489–498. IEEE, 1994.

[92] Li Tan, Zizhong Chen, Ziliang Zong, Rong Ge, and Dong Li. A2e: Adaptively aggressive energy efficient dvfs scheduling for data intensive applications. In *IEEE 32nd International Performance Computing and Communications Conference (IPCCC)*, ISBN: 978-1-4799-3213-9, pages 1–10, San Diego, CA, Dez,2013. IEEE.

[93] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In *IEEE Transactions on VLSI Systems, vol. 2*, page 437–445. 1994.

[94] Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro 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, 2007.

[95] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[96] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The Splash-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture - ISCA'95*, pages 24–36. ACM, 1995.

[97] M.K. Yadav, M.R. Casu, and M. Zamboni. Laura-noc: Local automatic rate adjustment in network-on-chips with a simple dvfs. In *IEEE Transactions on Symposium on Circuits and Systems II: Express Briefs*, volume 60 of *ISSN:1549-7747*, pages 647 – 651. IEEE, Oct, 2013.

[98] Matt 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. IEEE, 2007.

[99] Hui Zeng, Matt Yourst, Kanad Ghose, and Dmitry Ponomarev. Mptlsim: a cycle-accurate, full-system simulator for x86-64 multicore architectures with coherent caches. *ACM SIGARCH Computer Architecture News*, 37(2):2–9, 2009.

[100] Davide Zoni, Simone Corbetta, and William Fornaciari. Hands: Heterogeneous architectures and networks-on-chip design and simulation. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 261–266. ACM, 2012.

# Appendix A

# How To Install the MPSoCBench

This Appendix shows how to install the MPSoCBench and its requirements. The benchmark is released as open-source, and it is available in two ways: a virtual machine with all infrastructure ready for use, and as source code. The two following sections describe these two processes.

## A.1 Source Code - Installing all tools manually

This tutorial aims to facilitate the configuration of MPSoCBench and all tools involved.

### A.1.1 Requirements

The following tools must be installed and properly configured. If the user does not have any of the tools on its system, the next section can help on necessary settings:

- GCC 4.8
- Python 2.6.x
- SystemC 2.3.1 including TLM 2.0
- Archc and compilers for target applications

### A.1.2 Installing SystemC

SystemC is a collection of C++ classes and templates that provides powerful mechanisms to model system architecture with hardware timing, concurrency, and reactive behavior, allowing the creation of an executable specification of the system. To install SystemC 2.3.1, perform the following steps:

We recommend installing SystemC and the other tools in the `opt` directory, but if the user wants to install in another directory, replace the `opt` directory by the new path in all the following commands.

- Download the systemc-2.3.0.tgz (that includes TLM) (`http://www.accellera.org/downloads/standards/systemc`)

- Unzip in the opt directory;

```
$ tar xvf systemc-2.3.1.tgz
```

- It is recommended that you rename the directory:

```
$ mv systemc-2.3.1 systemc
```

- Configure and install:

```
$ make uninstall #if yout want to remove old versions
$ ./configure --prefix=/opt/systemc
make
make install
```

Note that the user uses the SystemC paths to configure and compile the tool. Make sure that use correct paths.

### A.1.3 Installing ArchC and Cross-Compilers

ArchC is an Architecture Description Language (ADL) following a SystemC syntax style, which provides enough information in order to allow users to explore and verify a (new or legacy) processor's architecture by automatically generating not only software tools for code generation and inspection (like assemblers, linkers, and debuggers), but also executable processor models for platform representation. To install ArchC with TLM 2.0, perform the following steps:

We recommend installing ArchC and the other tools in the `opt` directory, but if the user wants to install in another directory, must replace the `opt` directory by the new path in all the following commands.

- Download the ArchC (`http://www.archc.org/`)
- Unzip in the /opt directory;

```
$ tar xvf archc.tar.gz
```

- Execute the `autogen.sh` script to prepare the environment;

```
$ sh autogen.sh
```

- Configure and install. Note that you use the SystemC and TLM paths to configure ArchC. Make sure that you use correct paths:

```
$ make uninstall #if yout want to remove old versions
$ ./configure --prefix=/opt/archc --with-systemc=/opt/systemc
$ make
$ make install
```

- Download the ArchC Cross-compilers (`http://www.archc.org/`-Downloads-Cross Compilers and Tools). The user needs to download the cross-compiler (compilers for target applications) for PowerPC, MIPS, SPARC, and ARM;
- Unzip each one of them in the `/opt/compilers`:

  ```
  $ tar xvf <compiler_name>.tar.gz
  ```

- If the user has problems to execute some cross-compiler in a 64-bit Linux environment, it must try to use `apt-get install libc6-i386`.
- Please edit the `/.profile` file with the compiler paths. Example, if the compilers are in `/opt/compilers/` folder:

  ```
  export PATH=/opt/compilers/mips-newlib-elf/bin:$PATH
  export PATH=/opt/compilers/arm-newlib-eabi/bin:$PATH
  export PATH=/opt/compilers/powerpc-newlib-elf/bin:$PATH
  export PATH=/opt/compilers/sparc-newlib-elf/bin:$PATH
  ```

### A.1.4 MPSoCBench

The MPSoCBench suite benchmark is a scalable set of MPSoC platforms composed of four different processor models (ARM, PowerPC, MIPS, and SPARC), IPs, communication devices and a representative set of application adapted from Splash-2 and ParMibench benchmarks, which contains 300 different multi-core configurations with up to 64 cores.

We recommend installing MPSoCBench and the other tools in the `opt` directory, but if the user wants to install in another directory, it must replace the `opt` directory by the new path in all the following commands.

- Download the MPSoCBench (`https://github.com/ArchC/MPSoCBench/releases`)
- Unzip in the opt/ directory:

  ```
  $ tar xvf MPSoCBench-2.0.tar.gz
  ```

- Use the `setup.sh` script to download automatically the newest version of the ArchC processor models:

  ```
  $ sh setup.sh
  ```

- Use the `env.sh` script to configure your system environment automatically:

  ```
  $ source env.sh
  $ bash -l
  ```

## A.2 Virtual Machine

If the user is looking for an easy way to prepare the environment to use MPSoCBench, we suggest using a Virtual Machine. So, the user can test and use this benchmark easily without mush effort to install and configure Linux packages and tools. However, if the goal of using the MPSoCBench requires simulator performance, is better to install all tools on its system. The source code will be available in this virtual machine as well. Currently, only the MPSoCBench 1.2 is available as a Virtual Machine. We will prepare a MPSoCBench 2.0 Virtual Machine as soon as possible.

### A.2.1 Requirements

The following tools must be installed and properly configured:

- VirtualBox (`https://www.virtualbox.org/wiki/Downloads`)
- Vagrant (`http://www.vagrantup.com/downloads.html`)
- SystemC: The SystemC License does not allow distribution of the source code. So, the user needs to download its last version on the official website and put the file `systemc-2.3.1.tgz` in the same folder that the Vagrant files. After this, our Vagrant script will build and install SystemC and other tools properly (`http://www.accellera.org/downloads/standards/systemc`).
- Vagrant Script (`http://www.archc.org/benchs/mpsocbench/script.tar.gz`)

### A.2.2 Preparing the Virtual Environment

- Installing Vagrant and other tools:

  ```
  $ dpkg -i vagrant_1.5.4_x86_64.deb
  ```

  Then go to the Vagrant directory and execute:

  ```
  $ cd vagrant
  $ vagrant up
  ```

  All tools will be downloaded and installed correctly. The file `systemc-2.3.1.tgz` needs to be in the same folder that Vagrant script. So, the user can start now the virtual machine and go to the correct folder:

  ```
  $ vagrant ssh
  $ cd /opt/MPSoCBench
  ```

- Running your first simulation:

  After installing all tools, start you virtual machine and go to the MPSoCBench folder:

```
$ vagrant ssh
$ cd /opt/MPSoCBench
```

Build the first multicore platform (example: quad-core MIPS with power estimation, connected by a router, running the Dijkstra application)

```
$ cd /opt/MPSoCBench
$ ./MPSoCBench -p=mips -pw -s=dijkstra -i=router.lt -n=4 -b
```

After build, you can simulate your first multicore platform:

```
$ ./MPSoCBench -p=mips -pw -s=dijkstra -i=router.lt -n=4 -r
Go to How To Use to see other examples of this benchmark.
```

After using, you can stop the vagrant machine with:

```
$ vagrant halt
```

# Appendix B

# How To Use the MPSoCBench

This Appendix shows how to use the MPSoCBench. Section B.1 describes the MPSoCBench script parameters and options. Section B.2 shows output report examples.

## B.1 The MPSoCBench script

MPSoCBench execution can be controlled by the MPSoCBench command line tool, which controls the execution of a hierarchical structure of Makefiles. Several platform components are configured using this script, providing the following parameters:

```
-b  or --build: to build simulators
-r  or --run: to run simulators
-nb or --nobuild: to run without recompiling the models
-l  or --clean: delete object files and the simulators previously created
-d  or --distclean: delete processor model files previously created \
         by ArchC
-h  or --help: help
-p  or --processor: to choose processor models
-pw or --power: to enable power consumption for SPARC and MIPS platforms
-p  or --processor: to choose processor models
-n  or --numcores: to choose the number of cores (1,2,4,8,16,32,or 64)
-s  or --software: to choose the application
-i  or --interconnection: to choose the interconnection device
-pw or --power: to enable power consumption for SPARC and MIPS platforms
-c  or --condor: to enable execution on HTCondor
-h  or --help: help
```

Four of these parameters are mandatory and must be filled with the following available options.

- `-p or -processor`: powerpc, mips, sparc, or arm
- `-n or -numcores`: 1,2,4,8,16,32, or 64

- `-i or -interconnection`: router.lt, noc.lt, noc.at
- `-s or -software`: basicmath, dijkstra, fft, lu, sha, stringsearch, susancorners, susanedges, susansmoothing, water, water_spatial, multi_8, multi_16, multi_office_telecomm, multi_network_automotive, multi_security, multi_parallel.

One of the `-b` or `-r` parameters are also mandatory, but do not have arguments. If the user chooses the `-b` option, the platform will be build and all executable files will be stored in a proper run-directory. In case of using the `-r` option and the run directory exists, the simulator starts immediately. However, if the simulator does not exist, all components are build and the simulator starts after.

Not all software scales from 1 to 64 cores. TableB.1 shows the limitation in the number of cores for each application.

| **Applications** | **Number of cores** | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| basicmath | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| dijkstra | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| fft | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| lu | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| sha | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| stringsearch | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Susancorners | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | |
| Susanedges | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | |
| Susansmoothing | ✔ | ✔ | ✔ | ✔ | | | |
| water | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| water_spatial | ✔ | ✔ | ✔ | ✔ | | | |
| multi_parallel | | | ✔ | ✔ | ✔ | ✔ | ✔ |
| multi_8 | | | | ✔ | | | |
| multi_16 | | | | | ✔ | | |
| multi_network_automotive | | | ✔ | | | | |
| multi_office_telecomm | | | ✔ | | | | |
| multi_security | | | ✔ | | | | |

Some of these configuration parameters allow all option to make it easy to execute the benchmark fully.

**Usage:**

```
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -b
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -r
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -r \
              -nb
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -b
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -r
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -r \
              -nb
```

```
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -c=queue -b
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -c=queue -r
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -c=queue  \
              -r -nb
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -c=queue \
              -b
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -c=queue \
              -b
./MPSoCBench  -p=processor -n=n_cores -s=software -i=device -pw -c=queue \
              -r -nb
./MPSoCBench  -n=n_cores -i=device -ht -c=queue -pw -b
./MPSocBench  -n=n_cores -i=device -ht -c=queue -pw -r
./MPSoCBench  -n=n_cores -i=device -ht -c=queue -pw -r -nb
./MPSoCBench -l
./MPSoCBench -d
```

This script creates a run-directory for each different platform and configure all Makefiles required. We show some command line examples:

- To build and run (`-r`) all programs (`-s=all`) in the 64-MIPS platform (`-n=64 -p=mips`) including power consumption (`-pw`), using a NoC-LT as interconnection device (`-i=noc.lt`):

```
$./MPSoCBench -s=all -p=mips -n=64 -pw -i=noc.lt -r
```

- To build without running (`-b`) the dijkstra benchmark (`-s=dijkstra`) in the 64-core platforms (`-n=64`), for all processors (`-p=all`), using a simple router as interconnection device (`-i=router.lt`):

```
$./MPSoCBench -s=all -p=all -n=64 -i=router.lt -b
```

- To compile all available configurations for future execution:

```
$./MPSoCBench -s=all -p=all -pw -n=all -i=all -b
```

The script edits the Makefile properly according with the arguments. For example, the following command line configures a platform with 8-mips connected by the NoC-AT interconnection, with power estimation to run the susan-edges application.

```
$./MPSoCBench -s=susanedges -p=mips -n=8 -i=noc.at -pw -b
```

After use the command above, all components will be compiled:

```
[['mips'], ['8'], ['susanedges'], ['noc.at']]
Making Processor mips ...
--- No simulator found, using acsim to generate one.
ArchC: Parsing AC_ARCH declaration file: mips_nonblock.ac
ArchC: Parsing AC_ISA declaration file: mips_isa.ac
ArchC: mips model files generated.
Making IP tlm_memory_at ...
Making IP tlm_lock_at ...
Making IP tlm_dfs_at ...
Making IP tlm_intr_ctrl_at ...
Making IS tlm_noc_at ...
Making Software susanedges ...
Making Platform platform.noc.at
Creating rundir for mips.noc.at.pw.8.susanedges...
---copy platform.noc.at.x to the appropriate rundir
---copy susanedges.mips.x to the appropriate rundir
```

The Makefile created after using the script with these arguments is stored at the root of the benchmark. If necessary, the user can edit this file directly.

```
export PROCESSOR := mips
export NUMPROCESSORS := 8
export SOFTWARE := susanedges
export PLATFORM := platform.noc.at
export CROSS := mips-newlib-elf-gcc
export POWER_SIM_FLAG := -DPOWER_SIM
export WAIT_TRANSPORT_FLAG := -DWAIT_TRANSPORT
export TRANSPORT := nonblock
export MEM_SIZE_DEFAULT := -DMEM_SIZE=536870912
export RUNDIRNAME := mips.noc.at.pw.8.susanedges
export ENDIANESS := -DAC_GUEST_BIG_ENDIAN
```

Other important parameters that a user can edit to configure details of its simulator are in the file `defines.h` at the root of the benchmark. Next we list the main resources that can be configure using this file:

- Enable/disable debug for each component
- Define the latency of each component
- Define the address of each component in the network
- Enable/disable DVFS-ES, DVFS-CPU (DVFS-SW is enabled by default)
- Define DVFS parameters and interruption codes
- Enable/disable DRAMSim estimations

## B.2 Outputs and Reports

For a purpose of illustrating the MPSoCBench output information, Figure B.2 shows an example of a piece of an output report after running the susan-edges application in an 8-core MIPS platform. For the sake of clarity, we show only information about one of the eight cores. Details of each core power consumption are stored in separated output files.

All files produced by the simulator are stored at the appropriate run-directory. The MPSoCBench produces the following reports:

- **Processors and cache statistics:** only at the standard output;
- **Power statistics:** a summary at the standard output and power details per core and per cache at the run-directory (CSV files)
- **Cache power debug:** at per cache files in the rundir, when the CACHE_-POWER_DEBUG flag is defined at the `ac_cache_power.H` ArchC file.
- **Platform statistics:** at the standard output, at the local_report.txt and at the global_report.txt files;
- **Network Traffic:** at the local_report.txt and at the global_report.txt files (only for NoC-AT);
- **DRAMSim2 statistics:** at the standard output, at the local_report.txt, and at the global_report.txt files;

At the end of the simulator, the application output is compared with a gold output and the result is printed at the end of the main report. Only to illustrate, we show in Figure B.2 the input file for the susan-edges application and in FigureB.3 the file after performing the parallel algorithm for recognizing edges.

```
SystemC 2.3.1-Accellera --- Jun  1 2015 09:15:26
Copyright (c) 1996-2014 by all Contributors,
------------------------------------------------------------------
------------------------  MPSoCBench  ----------------------------
--------------------- Running: susanedges -------------------------
--------------- The results will be available in ------------------
--------------------- the output.pgm file -------------------------
------------------------------------------------------------------
Total Time Taken (seconds):    171.905920
Simulation advance (seconds):    0.100569
MPSoCBench: Ending the time simulation measurement.
ArchC: Simulation statistics
    Times: 172.79 user, 0.26 system, 171.91 real
    Number of instructions executed: 50284615
    Simulation speed: 291.01 K instr/s
cache: IC
Cache statistics:
Read:   miss: 1232316 (2.4507%) hit: 49052012 (97.5493%)
Write:  miss: 0 (0%) hit: 0 (0%)
Number of block evictions: 1232188
cache: DC
Cache statistics:
Read:   miss: 13774 (0.102513%) hit: 13422627 (99.8975%)
Write:  miss: 18695 (0.447946%) hit: 4154799 (99.5521%)
Number of block evictions: 31860
...
------------------------ POWER REPORT -------------------------------
  Switching power :    0.000000e+00W
  Internal power  :    0.000000e+00W
  Leakage power   :    0.000000e+00W
  Aggregate power :    7.147999e-01W
                     ----------------
  TOTALS          :    7.147999e-01W
------------------------------------------------------------------
DVFS Access:    37
Lock Access:    22752
Router Access:    262467374
Memory Reads:    64951928
Memory Writes:    197492656

TESTING RESULTS: Test Passed.
```

Figure B.1: MPSoCBench report after running the susan-edges application in a 8-MIPS platform

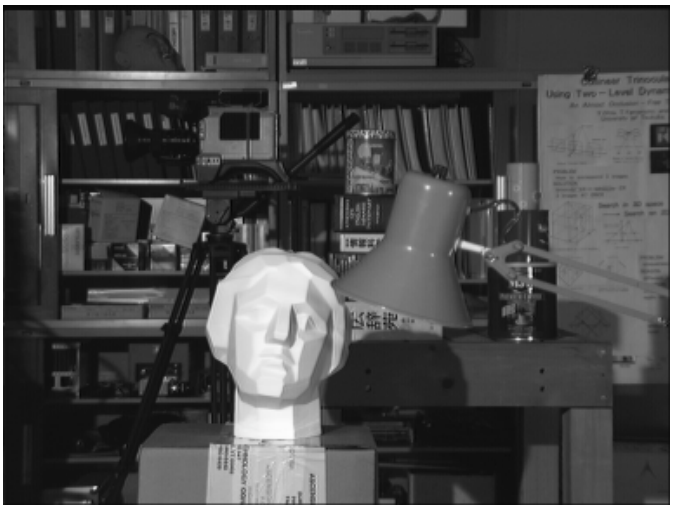


Figure B.2: The susan-edges input file

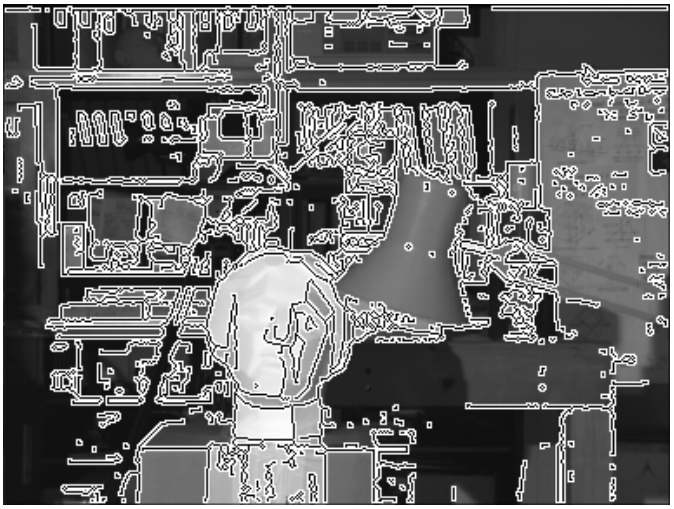


Figure B.3: The susan-edges output file