Evaluating the Impact of Future Memory Technologies in the Design of Multicore Processors

Guillem López Paradís

Thesis supervisor: Prof. Teresa Monreal

Co-Supervisor: Dr. Lluc Alvarez

Internal examiner: Dr. Miquel Moretó

Specialization in Computer Engineering
Polytechnic University of Catalonia

This dissertation is submitted for the degree of
Bachelor Degree in Informatics Engineering

Barcelona School of Informatics January 2017

Acknowledgements

First of all, I would like to thank all of my advisors: Teresa, Lluc and Miquel. They have been very patient with me and I appreciate this effort for guiding me during the last months. They also have made corrections to this document and I am pleased for this amazing experience.

In addition, I would like to thank all the RoMoL team for the feedback given during the project and its acceptance. Specifically I would like to thank the main developer of TaskSim, Francesc, which has helped me with TaskSim.

Finally, I would like to thank my friends and my family for their loving and support.

Abstract

"It's the Memory, Stupid!"

In 1996, Richard Sites, one of the fathers of Computer Architecture and lead designer of the DEC alpha, wrote a paper [36] with the title above. In that paper he realized that the only important design issue for microprocessors in the next decade would be the memory subsystems design.

After more than a decade later, the community of researchers started to digest and internalize this quote. Now, after more than two decades, it can be said that a lot of progress has been done since 1996 but the expectations of the enormous data sets that software is going to handle in the followings years tells that more aggressive designs are needed.

Another reason new memory technologies are needed is because of the multicore architecture which has increased the required memory bandwidth. This architecture completely extended across the main computer sectors was the result of continuing the Moore's law in exchange of adding more difficulties for software and hardware developers.

All of this has promoted this project. First, it has been decided to create a bridge of the state of the art DRAM simulator Ramulator [30] with the micro-architecture simulator TaskSim [34]. Once the bridge has been completed, the second goal of this project has been to make an evaluation of the impact of the current and future memory technologies in multicore architectures.

As a first approach, this new infrastructure has been used to evaluate the behavior of several parallel applications concluding that the execution time of the applications varies significantly across different memory technologies which even increase the differences while simulating different processors. The doubtless winner among all the memory technologies evaluated has been HBM which in some cases has achieved the best expected memory cycle response time.

Abstract (Catalan Version)

"És la Memòria, Estúpid!"

El 1996, Richard Sites, un dels pares de l'Arquitectura de Computadors i dissenyador principal del DEC Alpha, va escriure un article [36] amb el títol anterior. En aquest article es va adonar que l'única qüestió important per al disseny de microprocessadors en la dècada següent seria el disseny de subsistemes de memòria.

Després de més d'una dècada, la comunitat d'investigadors va començar a digerir i assimilar aquesta cita. Ara, després de més de dues dècades, es pot dir que un gran progrés s'ha fet des de 1996, però les expectatives dels enormes conjunts de dades que el software utilitzarà en els següents anys indica que es necessiten dissenys més agressius.

Una altra raó per el qual es necessiten noves tecnologies de memòria és degut a les arquitectures multinucli que augmenten l'ample de banda de memòria requerit. Aquesta arquitectura completament estesa a través dels sectors principals va ser el resultat de seguir la llei de Moore a canvi d'afegir més dificultats per als desenvolupadors de software i hardware.

Tot això ha promogut aquest projecte. En primer lloc, s'ha decidit crear un pont sobre el capdavanter simulador de DRAM Ramulator [30] amb un de micro-arquitectura TaskSim [34]. Una vegada que s'ha completat el pont, el segon objectiu d'aquest projecte ha estat fer una avaluació de l'impacte de les tecnologies de memòria actuals i futures en les arquitectures multinucli.

Com a primera aproximació, s'ha utilitzat aquesta nova infraestructura per avaluar el comportament de diverses aplicacions paral·leles arribant a la conclusió que el temps d'execució de les aplicacions varia significativament entre diferents tecnologies de memòria, que fins i tot augmenten les diferències amb la simulació de diferents processadors. El guanyador, sens dubte, entre totes les tecnologies de memòria ha estat HBM que en alguns casos ha aconseguit el millor temps de cicle de memòria esperat.

Abstract (Spanish Version)

"Es la Memoria, Estupido!"

En 1996, Richard Sites, uno de los padres de la Arquitectura de Computadores y diseñador principal del DEC alpha, escribió un artículo [36] con el título anterior. En ese artículo se dio cuenta de que el único problema de diseño importante para los microprocesadores en la próxima década sería el diseño del subsistema de memoria.

Una década más tarde, la comunidad de investigadores comenzó a digerir e interiorizar esta cita. Ahora, después de más de dos décadas, se puede decir que se han hecho muchos progresos desde 1996, pero las expectativas de los enormes conjuntos de datos que el software va a manejar en los próximos años indican que se necesitan diseños más agresivos.

Otra razón por la que se necesitan nuevas tecnologías de memoria es debido a la arquitectura multinúcleo que aumenta el ancho de banda de memoria requerido. Esta arquitectura completamente extendida a través de los principales sectores fue el resultado de continuar la ley de Moore a cambio de añadir más dificultades para los desarrolladores de software y hardware.

Todo esto ha promovido este proyecto. En primer lugar, se ha decidido crear un puente entre el potente simulador de DRAM Ramulator [30] con uno de micro-arquitecura TaskSim [34]. Una vez que el puente se ha completado, el segundo objetivo de este proyecto ha sido hacer una evaluación del impacto de las tecnologías de memoria actuales y futuras en arquitecturas multinúcleo.

Como primera aproximación, se ha utilizado esta nueva infraestructura para evaluar varias aplicaciones paralelas llegando a la conclusión de que el tiempo de ejecución de las aplicaciones varía significativamente entre las diferentes tecnologías de memoria y que incluso aumentan las diferencias al simular diferentes procesadores. El ganador sin duda entre todas las tecnologías de memoria evaluadas ha sido HBM que en algunos casos ha logrado el mejor tiempo de ciclo de memoria de respuesta esperado.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ACM Association for Computer Machinery

BSC Barcelona Supercomputing Center

CPU Computer Processing Unit

DAC Departament d'Arquitectura de Computadors(Computer Archirtecture Department)

DDR Double Data Rate

DRAM Dynamic Random Access Memory

GDDR Graphics Double Data Rate

HBM High Bandwidth Memory

HMC Hybrid Memory Cub

ISA Instruction Set Architecture

JEDEC Joint Electron Device Engineering Council

LPDDR Low Power Double Data Rate

RoMoL Riding on the Moore's Law Project

SDR Single Data Rate

SDRAM Synchronous Dynamic Random Access Memory

WIO Wide I/O

Chapter 1

Introduction

During the last decades, it has been given all the relevance to the Moore's law which has been predicting the decreasing size of transistors every two years. This law was an observation made by Gordon Moore[1] which has completely direct the semiconductor industry until now having every two years a more powerful computer.

Unfortunately, last year was fateful to the Moore's law and finally Intel could not continue shrinking the size of the transistor at the same speed. Intel had to change the manufacturer process to three years. This also confirmed and defensed the bet of going for multicore processors.

Some years ago it was decided to go for processors with multiple cores because it was no longer profitable as before to increase the performance of a single core increasingly complex. This change resulted in not increasing as much the single core performance in exchange of having more cores. However, this change also increased the complexity of the software needed to take full advantage of the new design.

At the same time as the Moore's law was predicting the nearly exponential growth of CPU performance, the memory performance was not growing at that rate. This phenomena commonly known as the memory wall problem, visualized in figure 1.1, has also been very important for computer architects for not being blocked the CPU activity by the memory.

In the search to obtain an ideal performance in the execution of the programs, it has been the memory subsystem that presented the most obstacles. When running multiple applications on a chip multiprocessor it occurs that a large percentage of the execution cycles is waiting the memory.

One of the first approaches to solve this problem was to manage a cache hierarchy inside the CPU chip, in this manner the data access latency was extremely reduced to nearly one cycle in the best case scenario. While the integration technology has advanced, the logical

[1] Co-Founder of Intel.

approach has been to put more caches inside the chip until the current three levels of cache. In addition, a simultaneous goal of increasing the with of instruction issue and reducing the average latency of memory access, requires to have a off-chip memory technologies able to support several concurrent accesses and high rates of data transfers (high bandwidth) [26].

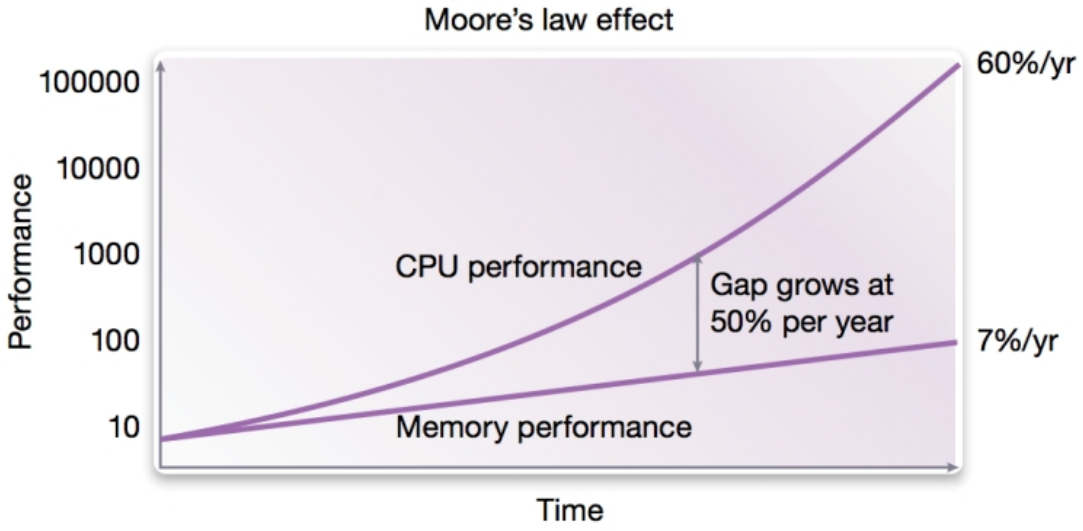

Fig. 1.1 Memory Wall and Moore's Law. Figure taken from [7].

In a bigger perspective, this can be viewed as a memory hierarchy where at the top are the faster technologies: all the CPU registers and on-chip caches, but the lower levels are the slowest: off-chip DRAM memory and then the hard drive. This project focuses on the study and evaluation of memory technologies located on the low side of this hierarchy and more specifically off-chip DRAM memories.

On this side, several design decisions have been taken in the aim of shortening the now called bandwidth wall[2] between the CPU and the off-chip memory. One of the first advances was the transition from *Single Data Rate* (SDR) to *Double Data Rate* (DDR) in the firsts designs of SDRAM. Currently, DDR is the most widespread design.

This architecture allows to send in one cycle two words[3] which increased by two the performance. This prolific standard is nowadays in its fourth revision DDR4 (2012) [5] and has also been expanded to different sectors like video cards (GDDR) and low power memories for the mobile sector (LPDDR).

Although this architecture has made enormous achievements, the new era of multicore processors with high frequency and bigger data sets imposed by the future software has aimed

[2]Limited communication bandwidth beyond chip boundaries.
[3]Particular unit of data for processor typically 32 or 64 bits.

to investigate new technologies able to achieve higher bandwidths. One of the promising technologies is the stacking of memory over the processor chip, reducing drastically the latency of a memory request. For this technology under development has already been made some standards like *High Bandwidth Memory* (HBM) [4] and its rival *Hybrid Memory Cube* (HMC).

On the other hand, advances in both industries (processors and memories) have been taken advantage of simulators in order to fully determine step by step whether one new proposal is useful or not. While CPU simulators are abundant and very powerful, simulating DRAM has not been very easy in the last decades until 2015 when Ramulator [11, 30] was published by a group of researchers at *Carnegie Mellon University* (CMU). This promising simulator has a clear advantage over the others: supporting several standards in contrast to a few offered by the others which could be interesting for furthers research.

Moreover, Ramulator comes with the possibility of being simulated with one of the most extended CPU simulators, Gem5 [18]. These features expand its uses with simulating a full memory hierarchy including the usually forgotten RAM in depth. Gem5 can simulate in different levels of abstraction and detail, which had enabled during the last years prolific research. However, as its main purpose is not simulating a big number of processors, it is slow compared to others simulators.

Another interesting simulator proposed by BSC is TaskSim [16, 34], a simulator not as powerful as Gem5 in terms of simulating at a low level of abstraction in exchange of simulating bigger amounts of processors. This simulator is under development and very used through the researchers at BSC offering great opportunities for evaluating the capability of parallel applications to scale up to high number of processors [24, 37, 27, 31, 32]. However, this kind of simulation needs a better memory simulation detail because of the increased memory bandwidth requirements of future applications.

The first approach taken by TaskSim to simulate the memory in a great level of detail was using DramSim2 [35] only able to simulate old standards like DDR2 (2003) and DDR3 (2007), which makes it obsolete for future memory technologies. This creates an opportunity for this project to integrate a new memory simulator in TaskSim and extend this infrastructure to make further studies.

This is one of the reasons why this project is interesting because making the bridge with Ramulator will add a needed realism to TaskSim in order to permit the evaluation of the impact of future memory technologies on parallel applications. This kind of evaluations are rarely found in the literature, which creates an opportunity for this project to make a study that is useful for the community.

1.1 Project Objectives

The project has two main objectives: integrating Ramulator in TaskSim and using these simulation infrastructure to study the impact of different memory technologies on parallel applications.

1.1.1 Integration of Ramulator in TaskSim

The first objective of this project will be the integration of a memory simulator inside a micro-architecture simulator. This will add a needed realism when simulating the memory in the selected micro-architecture simulator: TaskSim, a trace-driven cycle-accurate simulator that is being developed at BSC.

On the other hand, the memory simulator that will be used is Ramulator, a fast and extensible DRAM simulator that is being developed at the Carnegie Mellon University (CMU).

In order to achieve this objective, first of all, a stand-alone study of each simulator will be necessary. This first study will allow understanding in depth each simulator finding its drawbacks or limitations. This study will consist on reading the documentation provided by each simulator, experimenting with simulations and reading its code to understand how work internally.

Having completed this first study, it will be the time to create the actual bridge of the simulators. This bridge will consist in a piece of software inside TaskSim that will send memory requests to Ramulator and eventually receive messages from Ramulator for the purpose of ending these memories requests.

After the integration is done a validation will be necessary. The validation will determine whether it works as expected or there is something wrong. Although this may not seem very important, it is the key for further uses of this bridge. It has to be verified that the integration simulates the workloads correctly and does not break the execution.

1.1.2 Impact of Memory Technologies on Parallel Applications

Once the integration has been done , researches that use TaskSim will be able to evaluate better the performance of parallel programs with different real standards of memory. It also would allow to analyze the impact of memories on parallel applications.

This second objective is as ambitious as the first and several steps will be needed. First, it will be required a study of different DRAM memory technologies in the interest of finding

their limitations and advantages. This will allow to decide which standards will be used for the analysis in pursuance of a fair competition and for not obtaining a bad comparison.

One of the best features of Ramulator is its diverse range of standards that accepts by default. This feature is ideal for the integration because it will allow to simulate parallel programs with different DRAM standards in furtherance of an interesting and new analysis.

Finally, the study will provide several results conductive to an evaluation of the bottlenecks and advantages of the DRAM standards in parallel programs. Adding to this, using different CPU configurations in TaskSim will add richer results to this study.

1.2 Document Structure

This document presents a logical order through the chapters starting from the introduction and state of the art until an expected conclusion. The first part of this document is the first two chapters which presents the problem and explains the history until nowadays in order to easily entry in the topic of this project.

The second part of the document consists of the detailed explanation of the project from chapter 3 until chapter 7. Within this part it can be divided into three main topics:

- Chapters 3 and 4 explain the simulators and the bridge.
- Chapters 5 explains the experimental setup and the validation process of the bridge.
- Chapter 6 is where the analysis of memory standards in parallel applications is detailed giving several charts to easily see the results.

Having reviewed all these parts, the conclusions are given in chapter 6 and finally some project management considerations are summarized in the last chapter. After this, there are the references of all the chapters numbered in ACM[4] style. Finally, Appendices A and B show two configurations files: one of TaskSim and the other of Ramulator.

[4] Association for Computer Machinery.

Chapter 2

State of the art

The state of the art of memory technologies is going on the direction of on-chip memories reducing the big latency of off-chip standards and at the same time, increasing the bandwidth. This new memories are possible using an old technique called *Through-Silicon Via* (TSV) which enables to interconnect through the package of the chip. The next figure 2.1 shows how are interconnected.

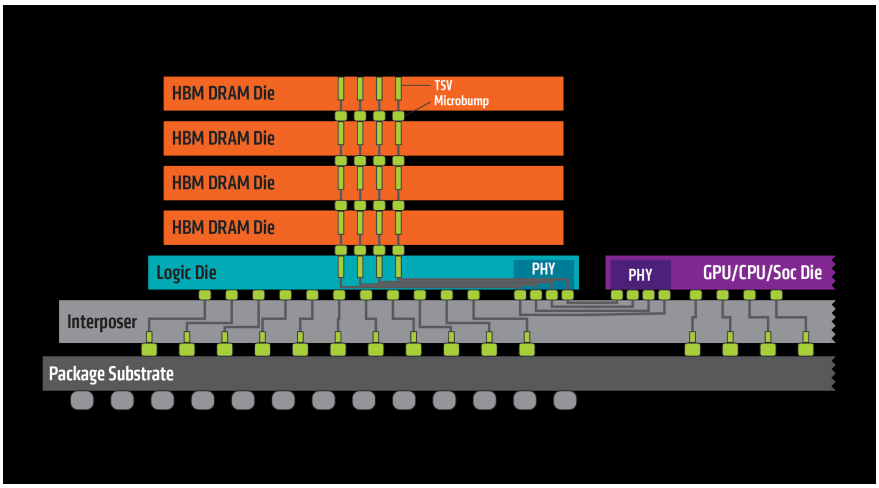

Fig. 2.1 AMD's HBM implementation. Figure taken from [1].

Following this technique, there are three standards already published by the *Joint Electron Device Engineering Council* (JEDEC) which is the Solid State Technology Association in charge of publishing new memory technologies. The three standards corresponds to different implementations using TSV: the most famous ones are HBM and HMC: HBM intended for video cards and HMC targeting supercomputers. There is also another standard called Wide I/O which aims to be the low power solution for memories requiring high bandwidth.

Moreover, TSV is a good technology for the 3D stacking of the chips, packaging the memory over the processor in order to reduce the latency. Having the memory over the processor has the clear advantage of proximity to the CPU but they also have some temperature problems. Up to date, this kind of packages have already been fabricated in some commercial graphics cards and industry research oriented products.

Furthermore, these new standards are intended to compete with the current established standards like DDR4 [5] for desktops and servers, GDDR5 [3] for video cards and LPDDR4 [2] for the mobile sector. This competition have to be validated and further studies are needed. It will be done in simulation and one of the current best memory simulators is the already mentioned Ramulator, which can simulate all these new standards except HMC.

Ramulator is the state of the art simulator offering a great range of standards to simulate, a fast simulation time and the possibility to be integrated with others simulators or to be altered in order to make studies of new memories. Some studies have already been made modifying Ramulator to simulate two levels of heterogeneous memory composed by HBM and DDR3 and to make an evaluation of reliability versus performance of this heterogeneous computers [25].

Moreover, another study [21] using Ramulator makes for the first time a detailed experimental characterization and analysis of latency variation. Once again it is used a modified Ramulator to do the study confirming its adaptability. One of the best competitors of Ramulator, DramSim2 [35], only accepts DDR2/3, so it is very limited to evaluate new proposals.

Nevertheless, Ramulator also offers an integration with one of the state of the art CPU simulators: Gem5. Gem5 is a modular execution driven architectural simulator that is developed by the community of researchers. A lot of studies have been made with this tool but not focusing in the memory. In contrast to this simulator, there are others simulators that are more lightweight so, even though they lose some accuracy, they are better suited to simulate big machines. For example TaskSim is a computer architecture simulator perfect for analyzing programming models and evaluate architecture with multiple cores.

As being the integration of simulators a must, TaskSim has some bridges and the one which is interesting to this project is with DramSim2. Although having said DramSim2 is obsolete for the one of the purposes of this project, it is very useful to have a related work inside TaskSim allowing a future bridge to be easier.

Nevertheless, related works about memories studies exist, for example a study [38] focused on new memory technologies using TSV defends its use comparing to the off-chip solutions. This study is interesting and related to this project in the manner that for future parallel applications this might be the key for achieving great performance.

Apart from the above paper, there is a good revision of the standards in the most important sectors like desktop, mobile and video cards [23]. From this work it can be obtained several interesting observations of memory technologies like the correlation between the memory performance and the number of *Dual In-line Memory Module* (DIMM)[1] of a channel. More DPC[2] reduce the overall performance of the memory system.

Although, these articles can be very useful they are not exactly the kind of study aimed for this project and no further related work have been found. The evaluation of memory architectures for parallel applications it is kind of new.

Finally, new architectures have been proposed specifically designed for multicore architectures. Udipi et al. [39] proposed a new design and organization of DRAM to improve the performance in multicore architectures.

[1] Memory module with both sides of the PCB with DRAM chips.
[2] DIMM Per Channel.

Chapter 3

Simulators

In this section, the simulators used for the project are presented. First, the selected CPU simulator is TaskSim and a brief discussion about this simulator is given as for one of the best alternatives, Gem5. Finally, for the part of Ramulator the same approach is followed.

3.1 TaskSim

TaskSim is a trace-driven simulator developed at Barcelona Supercomputing Center (BSC) by the team of Alejandro Rico [34] and nowadays, the RoMoL team. Being trace-driven means that TaskSim requires a trace[1] of the application to be simulated and a configuration file. This configuration file[2] specifies to TaskSim which kind of simulation will be done and the characteristics of the processors and the memory hierarchy. Lastly, TaskSim targets the simulation of parallel applications abstracting the detailed computation of each core for systems with lots of them in order to view the scalability of a large computational system.

TaskSim works at the level of tasks and this allows to model issues of parallel applications like inter-task synchronization, overlapping of data transfer with task computation, cache sizes, coherency protocols and data migration, interconnection bandwidth and topology and memory latencies. It has two main modes of operation, the one that will be used in this project as it offers a great level of detail and a *Burst Mode* which is faster in exchange of not modeling the accesses to the memory hierarchy. The original aim of TaskSim was to scale up to 1000 processors which was very ambitious and it is still under development.

However, there are several kinds of simulators that have its advantages and drawbacks in contrast to trace-driven. In the last years, execution-driven simulators have been made very famous which in exchange of bigger simulation time they can dynamically change the

[1] Pre-recorded stream of instructions with a fixed input.

[2] An example of the default configuration file can be seen in Appendix A.

instructions to be executed depending on different inputs data. This kind of simulators work at the level of instructions allowing to be very specific in timings of the micro-architecture and see related issues to this kind of abstraction. The drawback with this kind of approach is the scalability for enormous systems.

TaskSim accepts different kind of traces that affects its performance (simulation time). However, since 2014 there is a default configuration which is binary, instead of plain text. This change enabled to decrease the size of the traces. They can also be different depending on the desired type of simulation. The most recent ones are *uop* and *rle*: both stores the memory access and operations separated but in *rle*, the memory addresses are expressed in a relative value to the last memory address reducing the total trace size. Finally, there was also an old type of traces called *mem* in plain text.

Furthermore, traces are generated by an external instrumentation tool called PIN provided by Intel and with the help of Nanos++[3]. PIN enables to record which instructions are executed and the addresses of all memory accesses for a binary program execution at run-time. Moreover, it also detects the Nanos++ library calls and is able to catch the Task creation, deletion, pausing, resuming, etc. Merging all of this information, TaskSim is able to generate a trace.

For the study of scalability of large systems it is not required a detailed level of instruction because it will be inviable in simulation time. The approach that follows TaskSim is the commented above task-execution which scales very good with so many cores. With the help of native execution, off-line simulation and analytical-models it is obtained a burst length for the task not being that detailed. Combining this with the inter-task synchronization and data-transfers allows to model the memory and interconnection systems in cycle-accurate detail.

Although there are several good simulators in the market, one of the most famous is the Gem5 simulation infrastructure. It is the result of the merge between M5 [19] and GEMS [33]. M5 provides a a highly configurable framework, multiple *Instruction Set Architecture's* (ISA) and different CPU models. On the other hand, GEMS complements these features with a detailed and flexible memory system, including support for multiple cache coherence and interconnect protocols.

Gem5 is an execution-driven simulator with an excellent detailed CPU and hierarchy model that permits to simulate in different levels of abstraction easily specifying which level is required. Two of the main features of the simulator are the Python Integration that permits to easily configure the system while being coded in C++ (around 85%) and its Object-Oriented design that allows flexibility for multiple configurations.

[3]Run-time library developed in C++ that takes care of dependency management in task-parallel programs.

However, one of the most common problems in the commented simulators is the lack of a good memory simulation. For this reason, TaskSim has a bridge with DramSim2 that enables simulating with a great level of accuracy systems with DDR2/DDR3. This bridge has some drawbacks like high simulation time overhead and a limited number of memory standards.

3.2 Ramulator

Memory simulation has received little attention during the last decade compared to CPU simulation. Simulators have been emerging but never been designed to accept differents standards. For example, two of the most famous simulators a few years ago were DramSim2 and USIMM [20] but only support two standards: DDR2/DDR3. This is a big limitation for the rapid changes of DRAM standards. Moreover, in the recent years new academic proposals have been proposed.

One of the newest memory simulators is Ramulator, a fast, extensible and cycle-accurate DRAM simulator developed at CMU that offers a good performance and multiple DRAM industry and academic standards configurations. This allows to evaluate the strengths and the weaknesses of present and future standards. Also, the extensibility of Ramulator is kind of new in the world of DRAM simulators.

Ramulator is an open source simulator with a *MIT* license that enables to the community to modify and help in the development of the project if the authors authorize it. Actually, the project is on Github [11] which enables anybody to view, download and use the simulator.

Having a *MIT* license means that anybody can use, publish, modify, merge, etc with the little restriction of including the copyright. This kind of license created at MIT is one of the most famous one in the free software community. This license also enables to improve the simulator by the community finding new bugs or proposing new features.

Ramulator is based on an important observation: DRAM can be abstracted as a hierarchy of states-machines where the behavior of each state machine is dictacted by the standard. For this reason, Ramulator offers a standard-agnostic state-machine which is fitted in the class DRAM.h and then all the standards are built from this class. As a result, it offers a wide range of academic and industrial standards: DDR3/4 (desktop), GDDR5 (video cards), LPDDR3/4 (mobile), HBM (server, high bandwidth), WIO1/2 (mobile, high bandwidth), SALP (academic) ...

Any standard class that comes with Ramulator inherits from DRAM.h class. They offer a great level of detail and every timing reference can be checked with the JEDEC standards. Actually, in this project, it will be checked as a validation procedure for ensuring the validity of the standards.

By default, it supports three different types of usage:

- **Memory Trace Driven:** In this method Ramulator reads memory traces from a regular file and simulates the DRAM behavior. It assumes that the trace files contains at each line an address followed by a 'R' or 'W' that first indicates the address and then the operation. An example could be: 0xFF233245 R which performs a read of the address 0xFF233245.

- **CPU Trace Driven:** There is a simplified model of a "core" that generates memory requests to the DRAM subsystem built in Ramulator. In this manner, Ramulator expects in each line of a trace file a memory request and must have the amount of CPU instructions followed by the address and optionally a writeback[4] request, which is the dirty cache-line[5] that is evicted by the read request. An example could be: <num-cpuinst> <addr-read> <addr-writeback> .

- **Gem5 Driven:** In this case, Ramulator works as an independent simulator that receives memory requests as they are generated by the Gem5 simulator. This has a lots of possibilities for studying the performance using different configurations.

I order to execute Ramulator, a configuration file must be provided. At appendix B there is an example although the configuration consists in the following key-data:

- **standard** = One of the all standards that supports (Ex: DDR3/4, HBM, etc...)

- **channels** = The number of channels of the configuration. Being a channel the physical connection between a memory controller and DRAMs [28].

- **ranks** = The number of ranks (set of DRAM chips) per channel

- **speed** = The speed regarding the type of organization selected between the standard.

- **org** = The organization selected (standard-size-width of bank: DDR3_2Gb_x8). Being a bank the basic unit considered in Ramulator which is a group of several memory arrays of storage cells [28].

This organization will be explained in the analysis chapter where further explanations will be given.

[4] A writeback request occurs in caches with a writing policy of write-back where the only copy of the memory even modified is stored in the cache until a request to the same position in the cache force the copy back to the memory.

[5] Cacheline is the container of several blocks of memory in a cache and a dirty cacheline is the common way to refer to a modified cacheline in write-back caches.

Internally the organization of memories in Ramulator is translated in this manner:

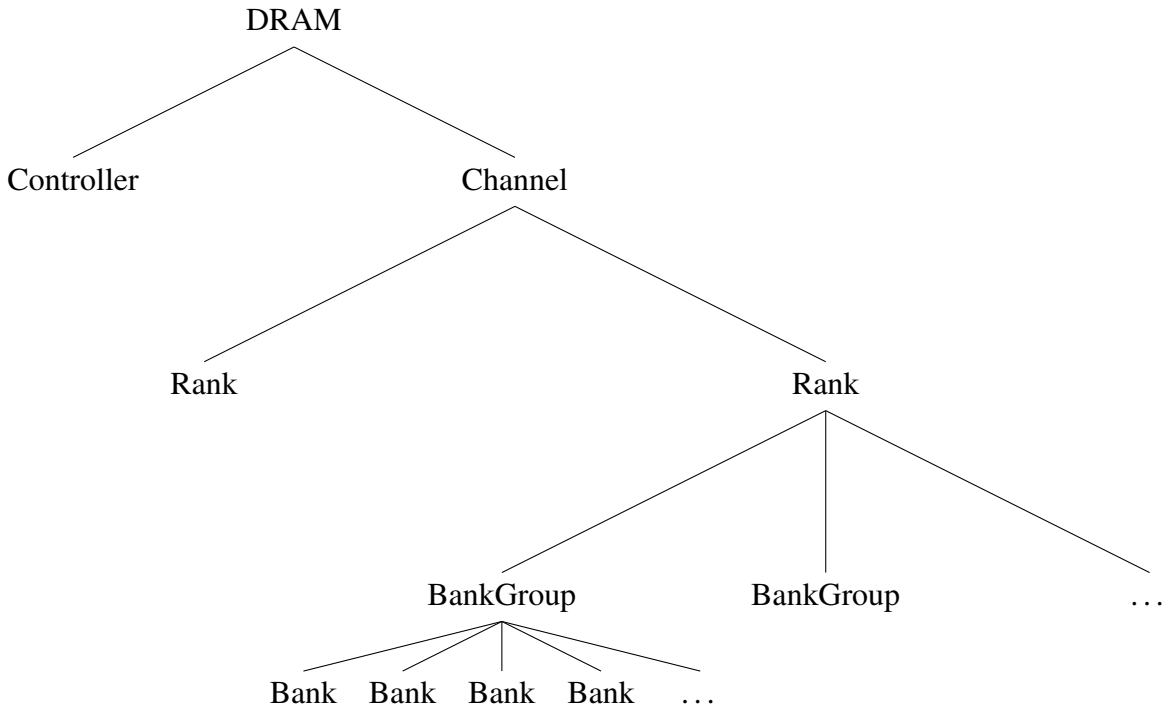

Fig. 3.1 Internal Memory Organization in tree.

It also allows to record all the internal commands and timings from the memory controller to the banks of the simulation in order to see exactly what happens inside. As expected, this feature increases dramatically the simulation time and it is only useful for debug and learning purposes.

```
PRE      90959:
ACT      90960:
PRE      90960:
 RD      90961:
```

This snippet of the command trace shows the command at the first column and the clock at the second. Ramulator also permits to print more internal parameters but they are not useful for this project. Acronyms *PRE, ACT and RD* are some of the commands taking part in the process of accessing the memory (Precharge, Activate, Read) [28].

Comparing to its competitors Ramulator's creators state that Ramulator is 2.5X faster than its best competitor, USIMM. This is due to the fact that it uses look-up tables to store and update the relevant timings of the queries. The look-up table is a two-dimensional array of

lambdas functions[6], which is indexed using the level in the hierarchy at which the command is being decoded for in this manner store relevant data with a rapid access.

[6] A C++11 construct.

Chapter 4

Bridge

In this chapter it is explained the process of making the bridge between TaskSim and Ramulator. It starts talking about the first get in touch with both simulators and its tools. Then it is explained the DramSim2 bridge with TaskSim and then the Gem5 with Ramulator. Following the bridges it is explained the bridge with Ramulator and finally, the results and tests.

4.1 First use of Simulators

The first simulator that has been used is Ramulator because of its size. It is less complex and for starting in the world of simulators it is a better choice than a more complex one as TaskSim.

4.1.1 Ramulator

Its installation is very straight forward and it only requires to download it from the Github page and then follow the instructions explained in the same page. The only requirement for the installation is a C++11 compiler like g++-5 or clang++[1].

Once it has been downloaded, executing the usual command: "make -j " will install it. By default it will execute clang++, so if the preferred compiler is g++-5 a little change in the Makefile is needed but for the initial execution stand-alone is fine clang++.

However, if the desired usage is the third one with gem5, then it is necessary to follow some others instructions that also requires to apply a patch that Ramulator offers to Gem5. Although it does not seem very difficult, it does not suit into the goal of this TFG so this has not been tried.

[1] g++-5 is the fifth version of G++ and clang is an alternative developed by Apple.

For testing purposes and because the simple test provided by Ramulator is very easy, some different tests have been created to observe the performance. Memory trace driven is the perfect candidate for this kind of test, hence both random and sequential memory access traces have been simulated.

These initial sequential and random traces were created using a C++ program. In the sequential trace, the memory positions of a vector traversal were printed. In the case of the random trace, several numbers were generated using a random library.

In addition, it has also been reviewed the different configuration files that Ramulator offers in order to interact a little bit with the standards and its configurations.

The results of this preliminary tests are very promising as running a 10 million memory access do not last more than 20 minutes in a laptop environment, which is incredibly fast.

Furthermore, the second task of this part is more important because for making the bridge it is required a good understanding of how the simulator works. So further reading of the simulator code and tests have been useful to understand better the simulator.

4.1.2 TaskSim

Comparing to Ramulator, TaskSim is a bigger software and has a more complex installation. First, it has a copyright which means that the software is not publicly available and for this reason, a request for permissions is needed. TaskSim is maintained using a tool called RedMine [12], a flexible project management web application that in the case of TaskSim helps to group a wiki and the code repository in the same web.

For the installation of TaskSim there is a fully detailed page that explains step by step what is needed but for the first time, it is very easy to get lost. TaskSim is organized by different modules and each of them has a Makefile and finally a huge Makefile for every part. For this part, the help of the development team of TaskSim, in special mention to Francesc, has been crucial for correctly installing TaskSim.

Furthermore, TaskSim depends on other software for working like: Nanos++ [9], a runtime library designed at BSC, the compiler Mercurium [8], a source-to-source compilation infrastructure also from BSC and finally the Pin Instrumentation tools from Intel for making traces.

Therefore, the first installation is more difficult than Ramulator requiring other software to be installed and multiple configurations to manage. Also, comparing to Ramulator, first the sources are built and finally installed enabling multiple configurations but increasing its complexity.

This last feature is very useful in the usual case of having different versions of TaskSim: one for tracing, one using DramSim2, one for a special kind of trace... In this project,

five different TaskSim's have been installed: two enabling Ramulator or not, two enabling different traces (rle and uop) and finally one for tracing.

For testing purposes, it has been used traces already generated like the parsec benchmark suite [10] ported to Ompss[2] [22]. The first execution was the blackscholes[3] benchmark due to its simulation time which only takes about 45 minutes. This simulation time is much bigger compared to Ramulator and hence the laptop may suffer. For this part is just enough although for further experiments a cluster will be needed.

Executing this test has been very useful to understand how works the configuration file of TaskSim and although very simple modifications have been tested it offers multiple options.

Finally, the classes of the bridge with DramSim2 have been reviewed in order to understand the approach it has been followed and to start thinking how to do the bridge with Ramulator.

4.2 Other Bridges

TaskSim bridge with DramSim2 is done by three classes: Dramsim2Controller.h, Dramsim2-Controller-impl.h and Dramsim2ControllerDefs.h. The code resides in Dramsim2Controller-impl.h file which contains the implementation of the bridge. It is quite straight forward because Dramsim2 offers the possibility to be compiled as a library.

The connection is quite simple although some difficulties may come with the callbacks of the reads and writes which must be initialized properly in order to permit DramSim2 notify when it has finished doing a read or write request. Then it is also important to deal with the types of the variables and not loose accuracy. Finally, the Dramsim2ControllerDefs.h exists in order to have a configuration by default but enabling to modify at the configuration file of TaskSim.

After some time of debugging this bridge, an improvement was made by the development team of TaskSim because some users claimed its simulation time overhead was important. The improvement consisted on making in bursts all the cycles until Dramsim2 was up to date with TaskSim. This enabled to reduce the simulation time overhead significantly.

On the other hand, the **Gem5 bridge with Ramulator** is not that easy and needs further installation steps. To begin with, the bridge with Ramulator needs to apply to Gem5 a special patch[4] that is given with Ramulator code. This may not be the ideal solution due to Gem5

[2]Extension of OpenMp by BSC.

[3]Program consisting in option pricing with Black-Scholes Partial Differential Equation (PDE).

[4]Update of the code using a diff highly efficient.

users tend to modify the code an this patch may break some parts. Another solution could be to insert this patch inside the repository of Gem5 and have Ramulator as module.

Once the patch is applied, as ideally only creates two classes and modifies few lines, actually the size of the patch is 500 lines. Then it can be executed like # Run gem5 with –mem-type=ramulator and –ramulator-config=configs/DDR3-config.cfg. Finally, it can be seen that a configuration file must be specified and a special mem-typed is needed.

In the following figure 4.1 it is shown the software architecture of this bridge:

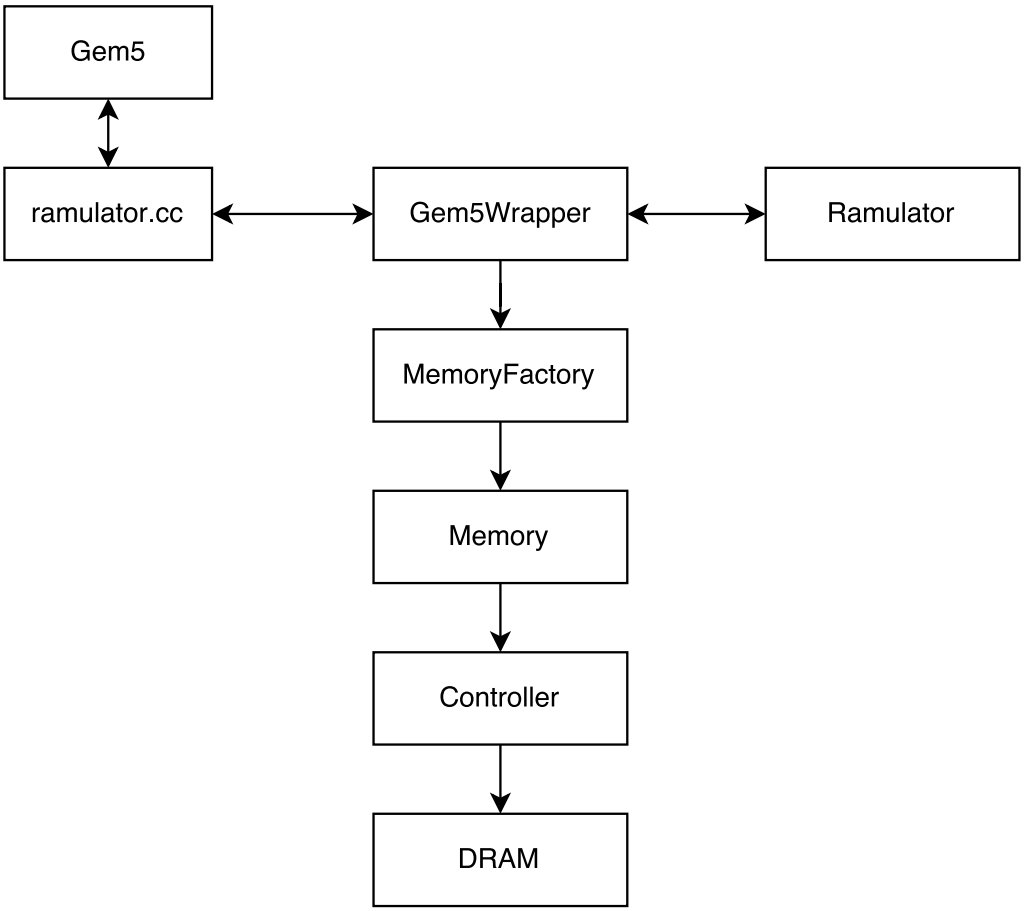

Fig. 4.1 Gem5 bridge with Ramulator.

Gem5Wrapper is the class that serves as middleware to use the statistics from Gem5. It also serves as a *application programming interface* (API) to use it in Gem5 in order to

make memory requests. It communicates with Gem5 through a class called *Ramulator* inside Gem5 that its only function is to call Gem5wrapper.

Then, inside the Ramulator code base there is MemoryFactory which helps in the task of further bridges and it is used by Gem5Wrapper to create the Memory. It also checks the parameters of the Ramulator configuration file that passes Gem5 makes sense before proceeding.

MemoryFactory creates an instance of Memory which only needs a configuration file and a vector of Controllers of the selected DRAM already created. This class is incredible useful for the bridge with TaskSim and it will be used because it allows to create automatically the memory.

As it can be seen, this bridge is different than the one of TaskSim with Dramsim2 because Ramulator team has made the bridge but does not participate in the design of Gem5. This is the opposite approach as in TaskSim with DramSim2 although it is very useful as it offers the special class MemoryFactory which facilitates any future bridge.

4.3 Bridge TaskSim-Ramulator

The first problem found was the version of g++ needed by Ramulator. The version used in TaskSim by default is 4.7 whereas the minimum needed in Ramulator is at 5.0 which created a conflict. The only possible solution was to fixed the problems that may have TaskSim to compile with the version 5 and hopefully were not much. The only errors found were related to syntax changes of some C++ libraries like String.

Having seen both bridges, the selected software architecture was inspired by both and the next figure 4.2 shows the general idea of the bridge.

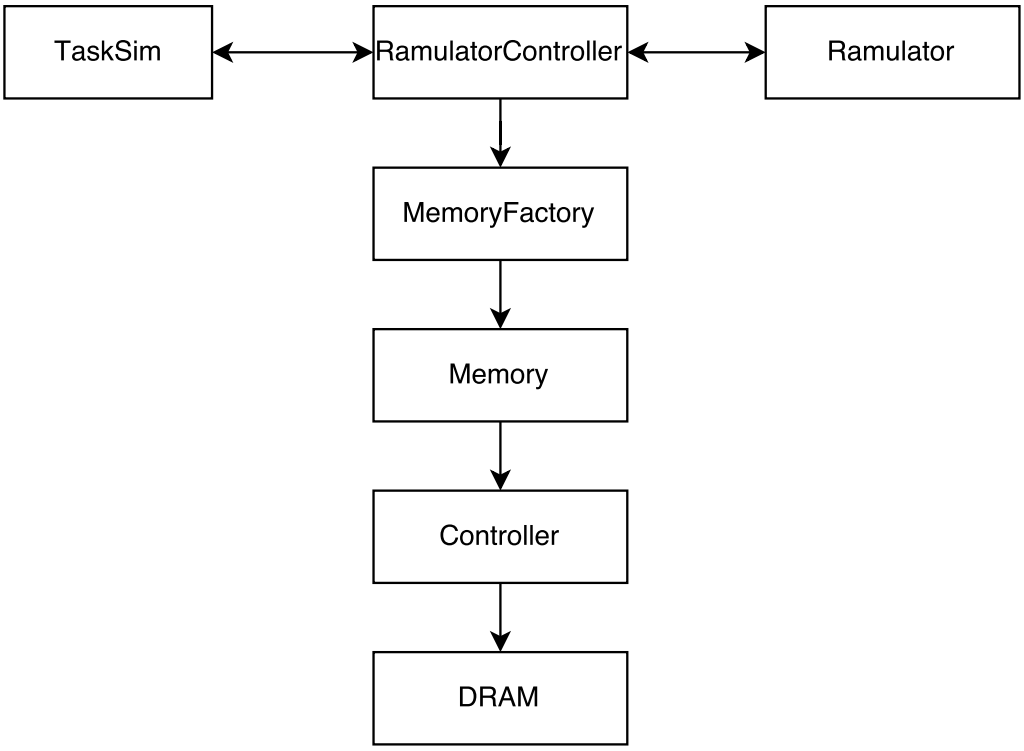

Fig. 4.2 TaskSim bridge with Ramulator.

The main difference between the bridge of Ramulator with Gem5 is that although RamulatorController is inside TaskSim it does not use Gem5Wrapper and it uses directly the MemoryFactory class. This is necessary due to the dependencies of Gem5Wrapper and also because it allows to not depend on an API that may change.

Another big difference is the decision of not using a separate configuration file than the one that uses TaskSim. This is for simplicity for the user and because it is not difficult to do.

The main idea is that the user will put inside the TaskSim configuration file a module called RAMULATOR and then specify the standard, channels, ranks that characterize the simulated technology.

The only difference between this configuration and the one that uses Ramulator is that in this case the user must specify the path where Ramulator statistics will be saved because they are not integrated inside TaskSim ones. This is another decision and it is done in this way because depending on the configuration the statistics file from Ramulator can be bigger than 200 lines and it is not viable to print it in the terminal.

```
[RAMULATOR]
full-path = /path/to/write/statistics
standard = HBM
channels = 8
ranks = 1
speed = HBM_1Gbps
org = HBM_4Gb
```

On the other hand, this architecture is similar to the bridge with Dramsim2 because some parts can be applied similarly. In the opposite side, the main difference is in the building: Dramsim2 offers a library which facilitates the process of linking and building whereas Ramulator does not have this feature which has made to include all the possible files in the corresponding Makefile of the memory module which ends in a slightly more time of compilation.

Furthermore, some Makefiles have been modified in order to enable some new flags to compile TaskSim with Ramulator. For enable Ramulator, the flag –enable-ramulator must be indicated and the source where is installed Ramulator: –with-ramulator-src-path=path.

Finally the bridge consists in three new classes:

- **RamulatorController.h**: This class contains the headers of the private functions and is mandatory for the connection with TaskSim.
- **RamulatorController-impl.h** : Here is where the code resides and the definition of each function is given. Here is the part that depends on Ramulator code as includes functions of MemoryFactory.h class.
- **RamulatorControllerDefs.h** : This class is a default configuration class in order to have some default in case the user only wants to try if it works.

The selected configuration by default is summarized in the following table 4.1:

Standard	Channels	Ranks	Speed	Org
DDR4	1	1	DDR4_2400R	DDR4_4Gb_x8

Table 4.1 Default configuration.

The same improvement done in Dramsim2 to reach a better performance has been preserved in this bridge. When Ramulator is needed, it does not wake up for only one cycle as it tries to do as much as it can in one burst of cycles.

In Dramsim2 some statistics were included in TaskSim but in this case, the decision was not to include them as it did not provide any more information than Ramulator's statistics.

4.4 Debug and firsts tests

In order to try for the first time the bridge some synthetics simple tests were required. The first synthetic was a for loop of 10 iterations that traverse a vector for making only reads. This test was rapid enough for making the simulation and see the firsts errors. The configuration was no relevant so the default was good enough.

Moreover, the next logically test was making writes and several approaches were tried. The difficulty came with default configuration of TaskSim which have a last level cache of 20MB and filtered all the memory requests. The intrinsic problem was having a cache with a write-back policy which required to have more request than 20MB. The useful approach was to reduce the capacity of the last level cache which enabled to find a severe bug explained in the next section.

Then, the next step was to increase the number of iterations of both tests to see if any error occurs. It did not happen so we conclude this part in order to continue to the validation of the bridge.

4.5 Bugs found

During the development of the bridge some bugs have been found. First of all, in TaskSim documentation there were several little mistakes that were fixed in order to be more friendly for new users or to clarify some miss-interpretations.

TaskSim also had a problem with sending the CPU ID[5] of a request. This CPU ID is very useful for Ramulator statistics. Some changes were done by the development team

[5]This ID corresponds to the core in a multicore-processor that issue the memory request.

of TaskSim in order to enable the delivery CPU ID of the issuer of a memory request to Ramulator. However, when the type of Cache is Writeback the actual issuer of a write request to the memory is the cache itself and the solution proposed was to send a -1 in the issuer to mark as write from cache. Then in the statistics one core will be added in order to track these requests.

On the other hand, a major bug was found in Ramulator when the writes were tried. Any write to Ramulator did not send its correct callback although it had finished properly. The little mistake created a middle bottleneck in the development of the project because in the best case scenario it finished in a stack overflow but in the worst case it halted the simulation.

At the time it was found in this project, this bug had already been reported by another person and the solution proposed was indeed similar to the one proposed in this project. In addition to this problem, which affected nearly all the standards of memory, it was noticed that this bug will affect on some academics standards like SALP [29] so it was reported on GitHub.

Finally, the developers respond to both pull requests[6] but no fixing has been made to this date which means that error stills in the public code on GitHub.

[6]Request to modify the code on GitHub with the supervision of the authors, very common in open source projects.

Chapter 5

Experimental Setup and Validation of the Bridge

In this chapter, the hardware and software setup used for the following experiments is explained. Then, the applications that will be used are introduced and finally the configuration of the processors and DRAM used in the simulations.

5.1 Hardware and Software

First of all, the experiments will be executed on three different machines: personal laptop, cluster Arvei and MareNostrum, each of them for a purpose explained below.

- A personal laptop will be used for the first testing and validation. It is a Dell XPS 13 (year 2013) sufficiently powerful to execute little tests, code and make charts:
  - **CPU:** Intel(R) Core(TM) i7-4500U 1.80GHz 2 cores
  - **RAM:** DDR3 8 GB 1600 MHz
- Arvei is a cluster of the Department of Computer Architecture (DAC) at UPC focused on research. It is composed of different configurations starting from 2006 until 2016 and the newer server that can be used has:
  - **CPU:** 2x Intel(R) Xeon E5-2630L v3 1.80 GHz 8 cores
  - **RAM:** 128 GB

- MareNostrum is the most powerful supercomputer in Spain and one of the most powerful in Europe. Actually, during 2017 it is planned to have its 4th upgrade and achieve a better position in the TOP 500[1]. The current nodes are composed by:
  - **CPU:** Intel Xeon CPU E5-2670 2.60GHz 8 cores
  - **RAM:** 32 GB

Arvei cluster is specifically oriented to the collaborative work of the research groups at DAC. Hence, the waiting time for running a job at this cluster is not high which permits doing the experiments here. Nonetheless, this cluster does not allow to lock a node for only one user and execute different scripts in furtherance of giving the maximum resources for a simulation.

In the following chapters it will be seen that an interesting metric of the bridge done in the project will be a study of simulation time overhead that is added using the bridge versus not. For this kind of study, it will be required to lock the node for only the simulation to not being affected by others jobs.

Conversely to Arvei, MareNostrum allows to reserve the node for only one user and hence permits the analysis of simulation time overhead. However, this machine is highly used compared to Arvei by a lot of national and international researchers which may add some waiting time to this analysis.

On the other hand, the strictly required software that will be used for the experiments will consist of:

- TaskSim: CPU simulator.
- Ramulator: DRAM simulator.
- G++: v5.1 in personal laptop and MareNostrum but v6.0 in Arvei.
- Matplotlib: library in Python for making charts.

5.2 Applications

In order to make an interesting analysis and validate the bridge two different sets of programs (benchmarks) have been used. First of all, a benchmark suit has been used, the PARSEC suite which is ideal for parallel simulations because it has been designed for today's and future real world applications.

[1]TOP 500 is the list of 500 most powerful supercomputers in the world.

However, this kind of suites have some limitations like being quite huge and difficult to change, so for this reason it also has been used three synthetics benchmarks for testing some rare cases and for debugging. Finally, it has also been added the benchmark SpecFem3D.

5.2.1 PARSEC Benchmarks and SpecFem3D

This suite was created as a reference for parallel simulations because it offers 13 programs from many different areas like computer vision, video encoding, financial analytics, animation physics and image processing.

One of the common features of benchmarks suites is having different inputs sets and Parsec suite offers four inputs for simulation: simdev, simsmall, simmedium and simlarge. It also offers a native input designed for native execution. The selected input used in this project will be simlarge.

Luckily eight out of thirteen applications have already been ported to OmpSs for enabling its simulation in TaskSim by the research group of RoMoL: blackscholes, body-track, canneal, dedup, ferret, fluidanimate, swaptions and x264. For this project, although all applications have been simulated, only three of them are sufficiently memory intensive for being useful for the analysis.

The most interesting applications have been facesim, canneal and ferret. They are quite diverse because while facesim simulates the motions of a human face, canneal simulates cache-aware annealing to optimize routing cost of a chip design and ferret calculates the similarity of generic data specially useful for search servers.

On the other hand, SpecFem3D [14], as they authors state, simulates seismic wave propagation in any type of conforming mesh of hexahedra (structured or not). An example of this could be simulating earthquakes or ocean acoustics. This benchmark has also been ported to OmpSs by the research group of RoMoL.

5.2.2 Synthetic Benchmarks

The PARSEC suite has showed some limitations like a big simulation time and low MPKI[2] for the next analysis. Therefore, two synthetics benchmarks were created to debug: stores and loads. As their name show, each program consists on a loop doing loads or stores. This program has been very useful for debugging purposes and the loads benchmark also has been used for the analysis and validation of the bridge.

Furthermore, a well-known memory intensive synthetic benchmark has also been used: the Stream [15]. Stream is a simple synthetic benchmark that measures memory bandwidth

[2]Misses per Kilo Instruction, a good metric for memory intensive applications.

(in MB/s) and it is usually used to measure the bandwidth of a system. In order to be included in the study we have ported it to OMPSS. In contrast to the original, it has also been vectorized to increase its MPKI.

5.3 Simulation configurations

5.3.1 CPU configuration

In furtherance of the realism of the simulations, three different CPU configurations have been used in TaskSim. They fit to the server, the laptop and the mobile sectors showing good variations in the end. These three configurations have resulted in three CPU configurations files for TaskSim similar to the one in appendix A.

The configurations are explained and table 5.1[3] summarizes its main characteristics.

- **marenostrum** CPU: similar to an Intel Xeon with three-level cache hierarchy with the last level reduced in order to beneficially get more misses in pursuance of putting more pressure to the DRAM subsystem.
- **i7** CPU: similar to an Intel i7 that can be found in laptops with also a three-level cache hierarchy.
- **arm** CPU: similar to an ARM Cortex-A57 included in the Qualcomm Snapdragon 810 with a two-level cache hierarchy.

	marenostrum	i7	arm
Cores	8	4	2
L1	32KB	32KB	32KB
L2	256KB	256KB	1MB
L3	10MB	8MB	

Table 5.1 CPU configurations.

The three sectors are investing in multicore architecture and the trend is going to continue in the near future. Moreover, one of the most promising is the mobile sector with its manycore architecture. Manycore architecture helps to reduce the energy consumption providing different cores designed at a different frequency changing dynamically depending on the load.

[3]For marenostrum and i7 configurations, L1 and L2 are private for each core and L3 is shared whereas in arm configuration L1 is private to each core and L2 shared.

5.3.2 DRAM configuration

This is one of the critical parts of the study. Without properly equally configurations, the results could be misleading. Although the last statement is true, depending on the type of analysis different configurations can be great at the same time. This is why it has selected a basic configuration similar to the one used in the paper of Ramulator [30] which for a start is good enough.

The selected standards for the study are the most used in the industry nowadays: DDR4 [5], LPDDR4 [2] and GDDR5 [3]. It has also been added HBM [4] to the study due to its promising performance.

Standard	Rate (MT/s)	Timing (CL-RCD-RP)	Channels	Width Channel	BW (GB/s)	# Bank width
DDR4	2400	16-16-16	2/4	64 bits	38.4	x16
LPDDR4	2400	22-22-22	2/4	32 bits	19.2	x16
GDDR5	6000	18-18-18	2/4	64 bits	96.0	x16
HBM	1000	7-7-7	8	128 bits	128.0	x1024

Table 5.2 DRAM relevant characteristics.

This table shows the most relevant characteristics that are looked when comparing DRAM's [28]:

- Rate is the usual metric corresponding to the number of operations transferring data per second. In this case, MegaTransfers per second which in DDR memories is the double of the frequency.
- Timing englobes three of the most important DRAM timing parameters:
  - CL: It is also referred as CAS and corresponds to the time interval between sending a column address to the memory and the start of data return by the DRAM device.
  - RCD: Time interval from the row access until the data is ready at the sense amplifiers before issuing a column delay.
  - RP: Time interval of precharging a DRAM array for another row access.
- Despite HBM that only allows 8 channels, the other configurations permit to modify and hence two sets of channels have been tried: 2 and 4.

- Channel width is a characteristic defined by JEDEC and this can not be modified for the selected standards. It represents the amount of bits that are read each from the bank each time.

- Bandwidth (BW) is the maximum theoretical data rate that can be achieved with the selected configuration. Is has been calculated with:

$$\text{BW(GB/s)} = \text{Rate (MT/s)} * 1 \times 10^6 * \text{channel width} * \#\text{channels} * \frac{1 \text{ byte}}{8 \text{ bits}}$$

- The bank width is also determined by the JEDEC standard and depends on the organization selected.

Standard	Channels	Ranks	BankGroups	Banks	Rows	Columns	Page size	Capacity
DDR4	2	1	4	16	65536	2^10	2 KB	8 GB
DDR4	4	1	8	32	131072	2^10	2 KB	16 GB
LPDDR4	2	1	None	16	32768	2^10	2 KB	4 GB
LPDDR4	4	1	None	32	65536	2^10	2 KB	8 GB
GDDR5	2	1	8	32	32768	2^7	2 KB	8 GB
GDDR5	4	1	16	64	65536	2^7	2 KB	16 GB
HBM	8	1	32	128	131072	2^7	2 KB	4 GB

Table 5.3 DRAM technical characteristics.

Conversely, this table 5.3 shows the technical characteristics of each configuration. All of these characteristics can be checked at each corresponding JEDEC and Ramulator uses the same.

Although the ranks could be a great study, for starters it has been set to one. The rest of the characteristics can not be changed without changing the organization.

Eventually we have used a total of nine DRAM configurations: seven of them are used with Ramulator and the last two using a memory model inside TaskSim:

- TaskSim memory by default which has a constant latency of 200 ns for each memory request useful for measuring the actual improvement of realism with Ramulator.

- Perfect memory configuration consisting on the ideal of 1 cycle per request useful for an upper bound of the benefits that memory technologies can provide.

5.4 Validation

In this section, the process of validation of the bridge is described. First of all, it is described the initial tests for validating the number of misses that arrived to Ramulator. Finally, a timing overhead analysis of the bridge is done.

5.4.1 Verification of the requests

The main fear in a bridge is breaking one of both simulators. This is why it has been payed attention to this case. To start with, a test has been created to check if all the requests sent by TaskSim arrives to Ramulator. This test is a Python script which makes a comparison of two greps[4]: one for the TaskSim statistics file and the other for the Ramulator statistics file.

This script is, on one side, very useful to check if any simulation has not ended correctly which eventually may happen and on the other hand, to check whether the bridge is not working properly.

Furthermore, in order to track all the misses through the memory hierarchy and finally expect a determined amount of memory requests has been impossible in the Parsec benchmark suite. Adding to this, the simulation time of these benchmarks is not acceptable for debugging and validation purposes in early stages.

All of this has encouraged creating special benchmarks for testing little cases. The first and most useful test has been a simple loop that only makes loads which results in a sum of a vector. If the size of the vector is known and can be modified, a study of different sizes can be sufficient to determine if the bridge is obtaining the expected amount of memory requests.

The following code shows the above mentioned loop.

```
#define SIZE 1024*1024*16
...
double *v = malloc(sizeof(double)*SIZE);

for(i=0; i<SIZE; i+=BSIZE) {
    suma += v[i+ii];
}
```

[4]Grep is a program in linux environments very useful to find strings in files.

Although by hand we can calculate the amount of misses that should happen, some more misses could be produced on account of run-time behavior. However, TaskSim offers the feature of only simulating the main thread which results in not having a big deviation of the calculated misses with the simulated misses. In the next table 5.4 it can be summarized.

Vector Elements	64K	128K	256K	512K	1M	2M	4M
Simulation misses	10020	18213	34596	67363	132899	263973	526749
Theoretical misses	8192	16384	34768	65536	131072	262144	524288

Table 5.4 Misses tables comparing theoretical and in simulation.

The calculation is quite simple as we know the cacheline size: 64 bytes, and the size of a double in C is 8 bytes resulting in one miss every 8 elements of the vector. Therefore, the theoretical misses are calculated dividing by 8 the size of the vector which K means 1024 and M 1024*1024. It can be seen that there is a constant deviation about 2K misses that it can be attributed to run-time behavior for creating the threads.

As a result, it has been proved that the misses can be perfectly tracked across the memory hierarchy. However, another procedure for validating the standards given by Ramulator was also done. It was checked the internals parameters of the DRAM configurations in Ramulator with the standards by JEDEC and they were the same in nearly every case. With these good results, it was finished the process of validation.

5.4.2 Time overhead analysis

Having validated the bridge, an interesting and necessary analysis is the study of simulating time overhead that it has been added on result of the bridge. This study has resulted in a chart normalized to the TaskSim execution without Ramulator to see the overhead added in a percentage.

This study is not trivial because of the environment that is needed to not deviate any simulation. For this reason, an entire node of MareNostrum was locked to do the study and then was executed sequentially each configuration. In this case, only the marenostrum CPU configuration was used.

This kind of study using the simulation time is extremely affected by the operating system and the load of the machine. This is the reason for locking the whole node only for one benchmark and executed sequentially. Nevertheless, some dynamic behavior can not be predicted and several executions are needed.

The following chart 5.1 shows the overhead time normalized to the simulation without Ramulator. For each of the benchmarks, a bar corresponding to the overhead percent is

plotted and Chart 5.1 indicates as well the overhead added using Ramulator to the simulations but differentiating when the simulation is in Ramulator and when is in TaskSim. The grey part indicates when it is being used Ramulator.

Finally, it has been added another set of percentages at the right equivalent to the geometric mean of each DRAM configuration for all the benchmarks. It is very promising to see that the simulation time overhead is not high enough to discard or affect hugely the use of bridge being about 20% of overhead. In the Stream benchmark, the overhead is huge because it is extremely memory intensive.

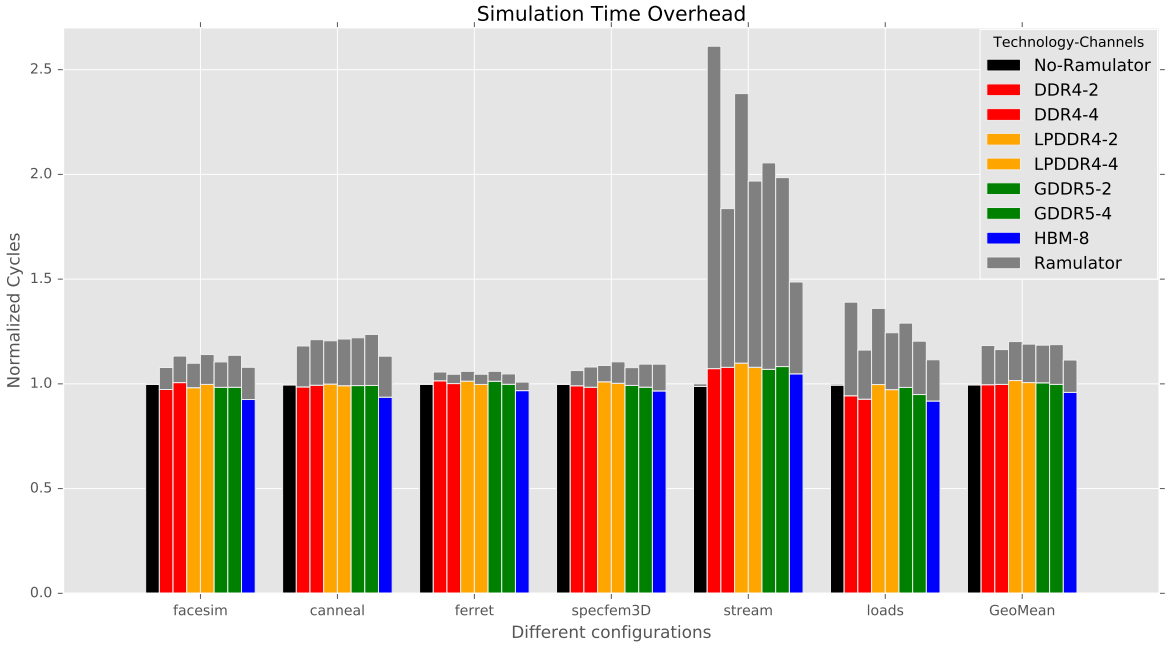

Fig. 5.1 Simulation time overhead measured as normalized time in cycles.

Chapter 6

Analysis

In this chapter the analysis of different DRAM configurations is done for the selected benchmarks. Several charts are included in order to help visualize the figures. First, it is evaluated the statistics given by TaskSim and finally the Ramulator's ones.

6.1 MPKI

One of the most useful metrics in memory intensive applications is the MPKI which is defined as "Miss Per Kilo Instruction". It is a good metric to determine whether the benchmark is memory intensive or not. It has to be remarked that the misses of this metric comes from the last level cache, L2 or L3 in this study.

$$\text{MPKI} = \frac{\text{Last Level Cache Misses}}{\text{Instructions}} * 1000$$

As a reference number, a MPKI bigger than 20-30 generally means an application is memory intensive. However, in this study applications with less MPKI than 20 have given great and interesting results. For example, the memory access pattern can affect significantly the behavior of DRAM.

The following chart 6.1 shows the MPKI of each benchmark obtained for each CPU configuration. Three colors determine the CPU configuration. The y-axis corresponds to the MPKI obtained and the x-axis to the benchmarks.

The different results between the CPU configurations can be easily seen in the chart as in the worst case can increase up to 2x the MPKI between marenostrum and arm. The origin of this difference is clearly the on-chip memory hierarchy and as it can be seen, having more levels of cache with bigger sizes results in filtering better the misses and not arriving to the memory.

However, in some benchmarks it is not that clear and could be because of its memory access pattern. The doubtless winner is the Stream benchmark. It is a synthetic benchmark and it has been defined to be more memory intensive with vectorization techniques obtaining a MPKI of 78 for marenostrum and 109 for i7 and arm configurations.

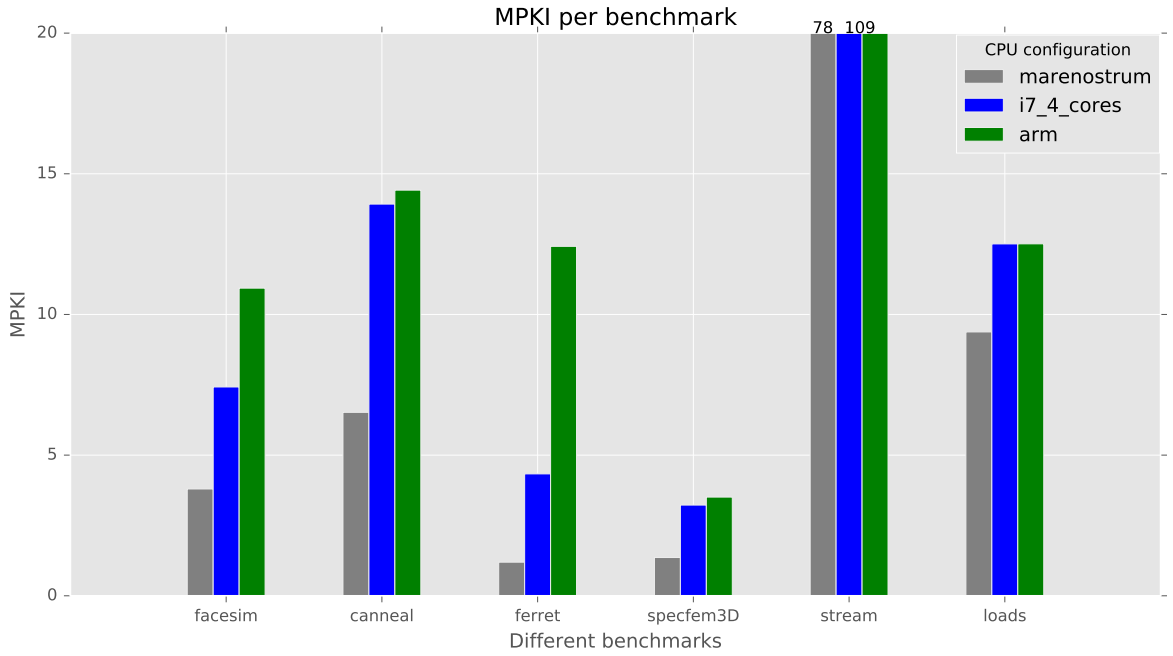

Fig. 6.1 Misses per Kilo Instruction per benchmark.

For this analysis, it has only been used TaskSim standalone without Ramulator. Although having different memories could slightly vary the total amount of memory requests, it did not change this chart.

6.2 Execution cycles

Being revised the MPKI of the selected benchmarks, the next metric studied is the total execution cycles. This metric is the amount of cycles that the benchmark would take in a real machine determined by the simulation.

This metric is very useful to state for example the realism of the simulation. In this project, the Ramulator is intended to add realism to the simulation of an overall system and this would require to see differences comparing to the simulation without Ramulator.

However, the interesting and expected result of this metric is the variations between different DRAM configurations and inside a DRAM configuration but different channels. This behavior has been obtained as the above extra realism added by Ramulator.

The next charts 6.2, 6.3 and 6.4 show the result of the execution cycles obtained by the simulation for each CPU and DRAM configuration. It has also been added a perfect memory very useful to see the lower bound cycles. To be more practical, the different results have been normalized to the simulation without Ramulator.

The y-axis represents the normalized execution cycles reached by every benchmark and DRAM configuration named in the x-axis.

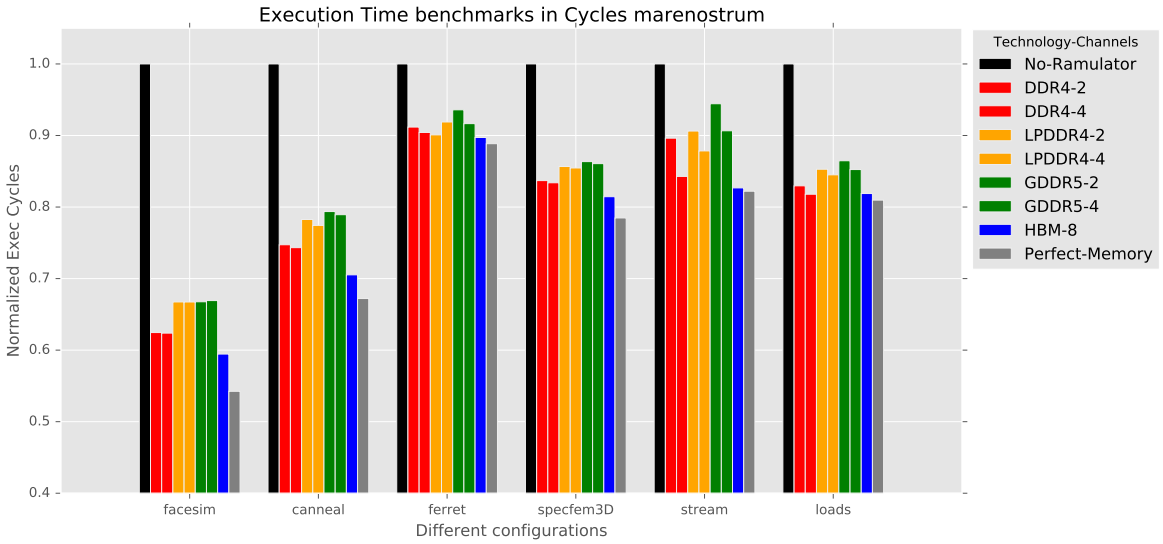

Fig. 6.2 Execution time of benchmarks on **marenostrum** configuration with different memory technologies.

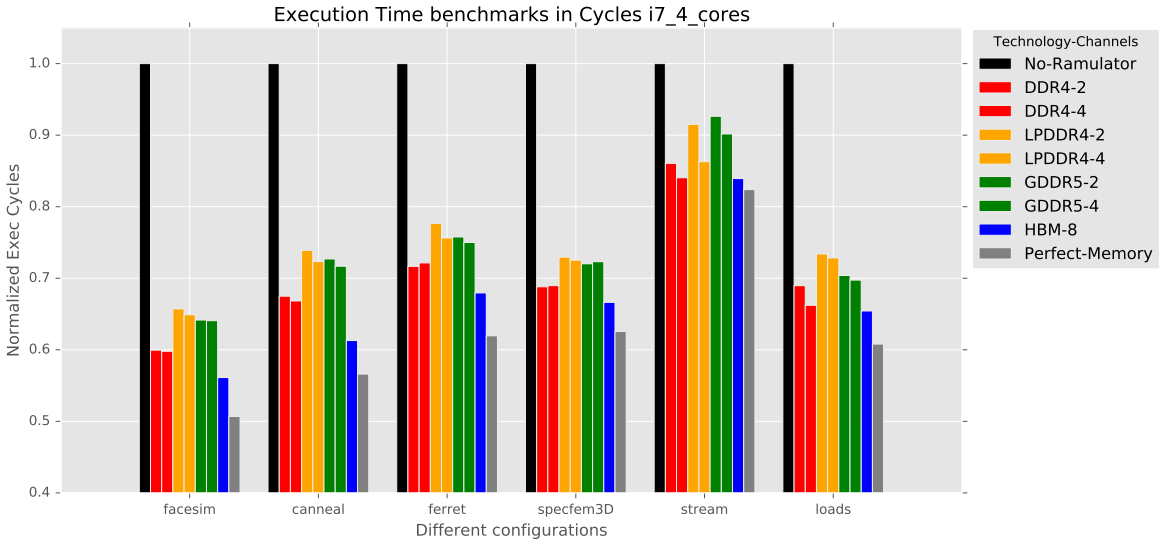

Fig. 6.3 Execution time of benchmarks on **i7** configuration with different memory technologies.

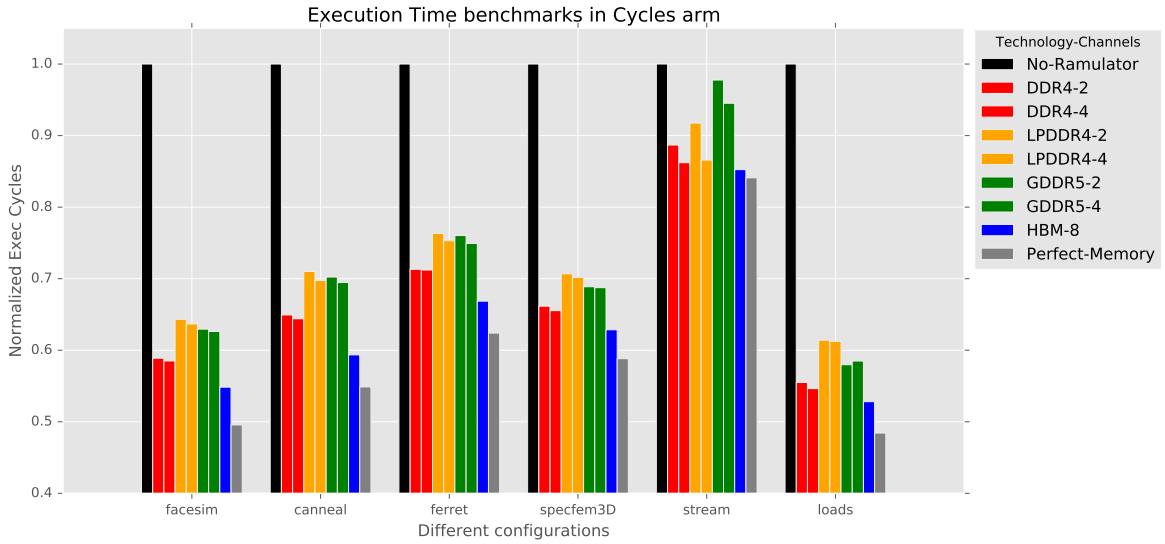

Fig. 6.4 Execution time of benchmarks on **arm** configuration with different memory technologies.

The first thing realized is the lack of implicit correlation between MPKI and having great variations in execution cycles. This can be unquestionably seen in the stream benchmark. Nevertheless, this is a special case and it has been very disappointing to see this bad results.

We believe that the reason for this behavior is the limit of TaskSim of not having different ports to the DRAM. This benchmark is so much memory intensive than for not being blocked at the last level cache, would need to have several ports in order to parallelize multiple requests. In this case, TaskSim accepts different requests but they are blocked on a buffer between last level cache and DRAM. This could be an interesting improvement on the next versions of TaskSim.

On the other hand, a promising result of this chart is the commented above realism of DRAM simulation in TaskSim. Using the DRAM configuration by default (200 cycles) can result in an execution time up to 40% more than using for example the DDR4 which could be the usual DRAM nowadays.

This improvement of TaskSim DRAM simulation realism shows that the DRAM configuration by default is far from being real. Although this realism has a cost of between 20 and 30% in simulation time, seen in the timing overhead analysis, it is low enough to firmly consider it.

Having reviewed all the CPU configurations it can be seen that the execution time is affected as expected. While comparing marenostrum and i7 CPU configurations shows a good difference in execution time using different memories, in the case of marenostrum and arm this already seen difference is increased.

For all the benchmarks there is a clear winner of the fastest DRAM memory technology: HBM. It is fastest enough to be near to the perfect memory model (Perfect-Memory bar).

Finally, another interesting result could have been the variations inside a standard when changing the channels but in this case there is not enough variation. The only benchmark which shows a good enough variation between channels is stream.

6.3 Row Hits, Misses and Conflicts

Having reviewed the most relevant metrics given by TaskSim, now it is the time to delve into the DRAM statistics provided by Ramulator.

First of all, when a memory access is issued and arrives to the memory controller, it has to be decomposed in order to get the requested data from the location through the memory. This decomposition depends on the addressing mapping policy[1] and the memory configuration. The next figure 6.5 shows a simplified access to the row and column of a given address memory.

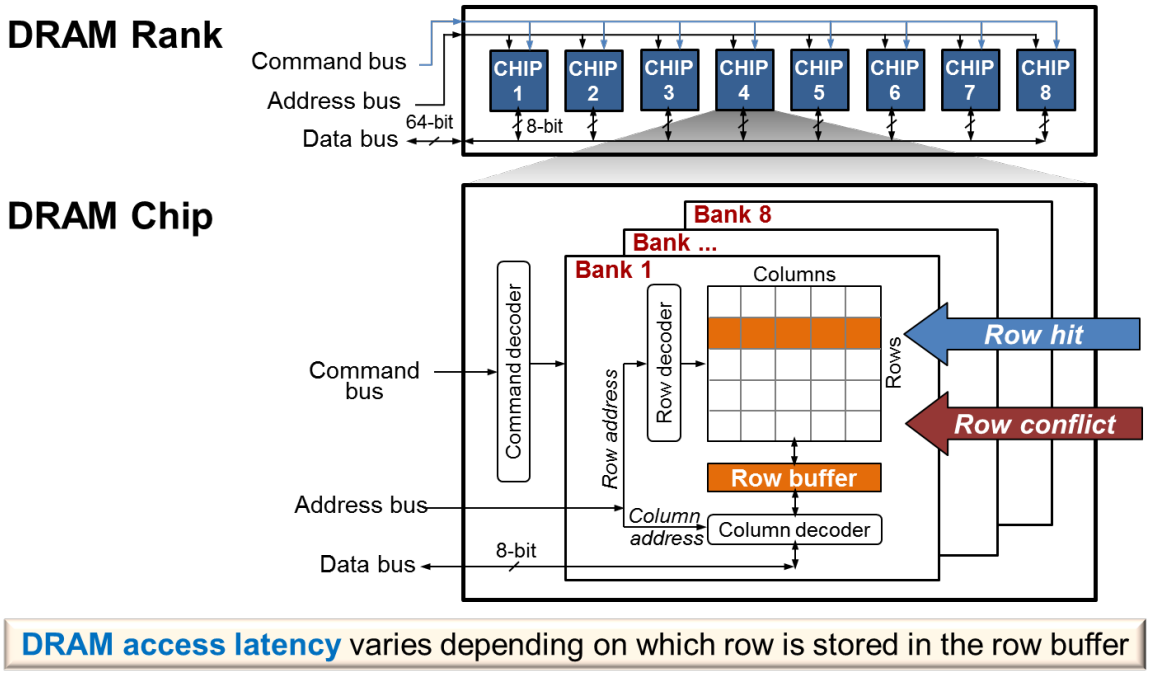

Fig. 6.5 Simplified address mapping. Figure taken from [6].

The addressing mapping used in Ramulator is RoBaRaCoCh meaning Row-Bank-Rank-Column-Channel. This means that for an address X bits are consecutively reserved, starting

[1]The address mapping policy a process that maps the physical address bits provided by processors to the internal structure of DRAMs [28].

from the channel until the row. Having the row in a higher position helps to the Open row page[2] policy, which tries to minimize changing the row in order to have less conflicts.

The figure also shows the meaning of a Row hit and conflict. The **row hit** occurs when two consecutive memory requests access the same row and usually different column. This is the ideal case because it is not required to close an old row, open the needed one and then get the column.

However, another situation can happen: the **row conflict**, that occurs when a row is accessed when there is already a different row open in a given bank. Then it has to be closed the opened row, get the new row and finally access the column. This increase the latency of a request as expected.

Furthermore, the last common situation is a **row miss** which happens when at the time of the request there is no open row and hence it has to be opened for the reading or writing. This situation is better than a conflict in latency time.

In order to determine the percentage of each situation a bar chart has been created for each CPU configuration. It has been normalized to the total amount of memory access to visualize the rate. Different styles of bars have been required for improving the differentiation of the events. The next charts 6.6 , 6.7 and 6.8 show these events. Y-axis represents the rate of row hits, misses or conflicts and x-axis indicates the simulation of the standards.

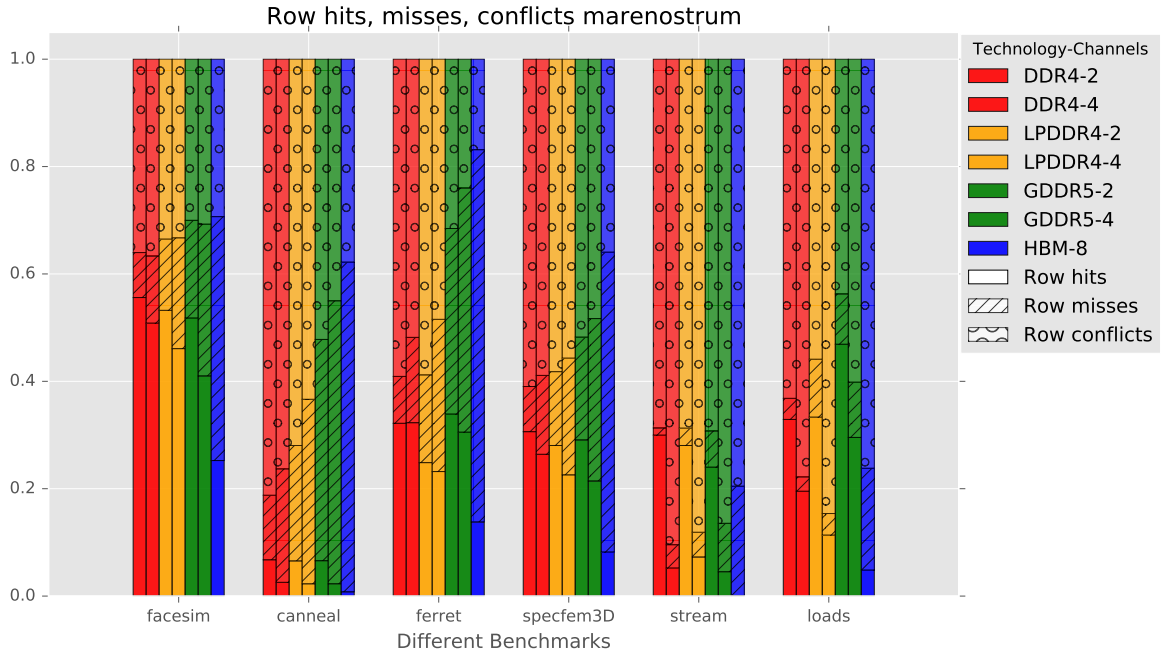

Fig. 6.6 Row hits, misses and conflicts of benchmarks on **marenostrum** with different memory technologies.

[2]Another way of saying row.

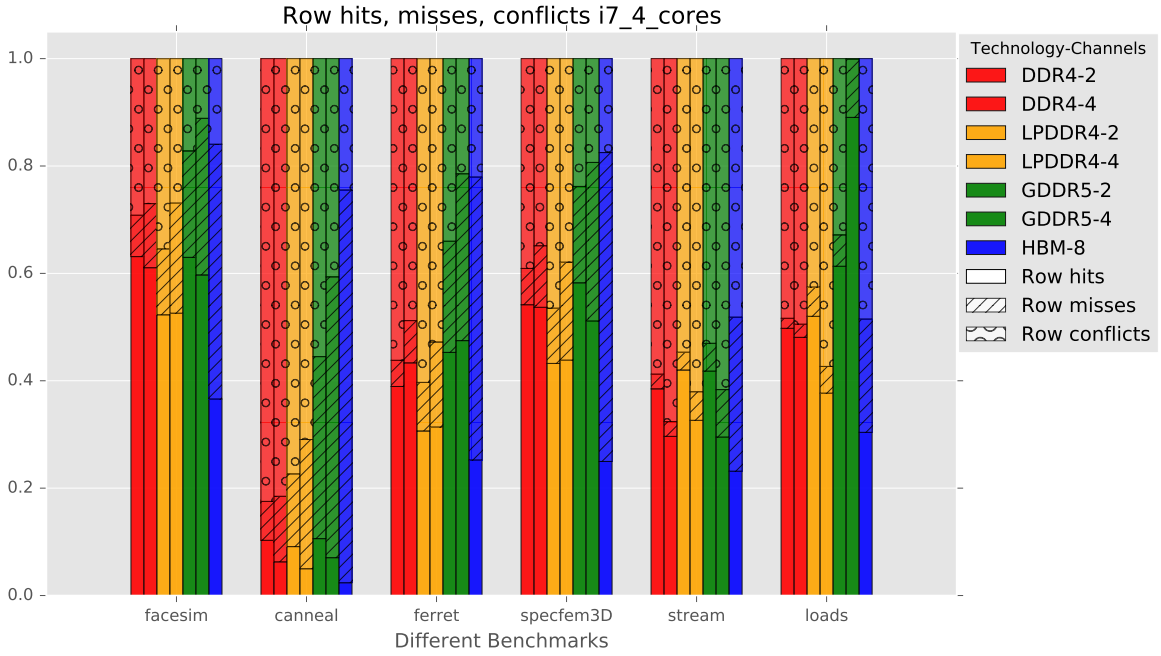

Fig. 6.7 Row hits, misses and conflicts of benchmarks on **i7** with different memory technologies.

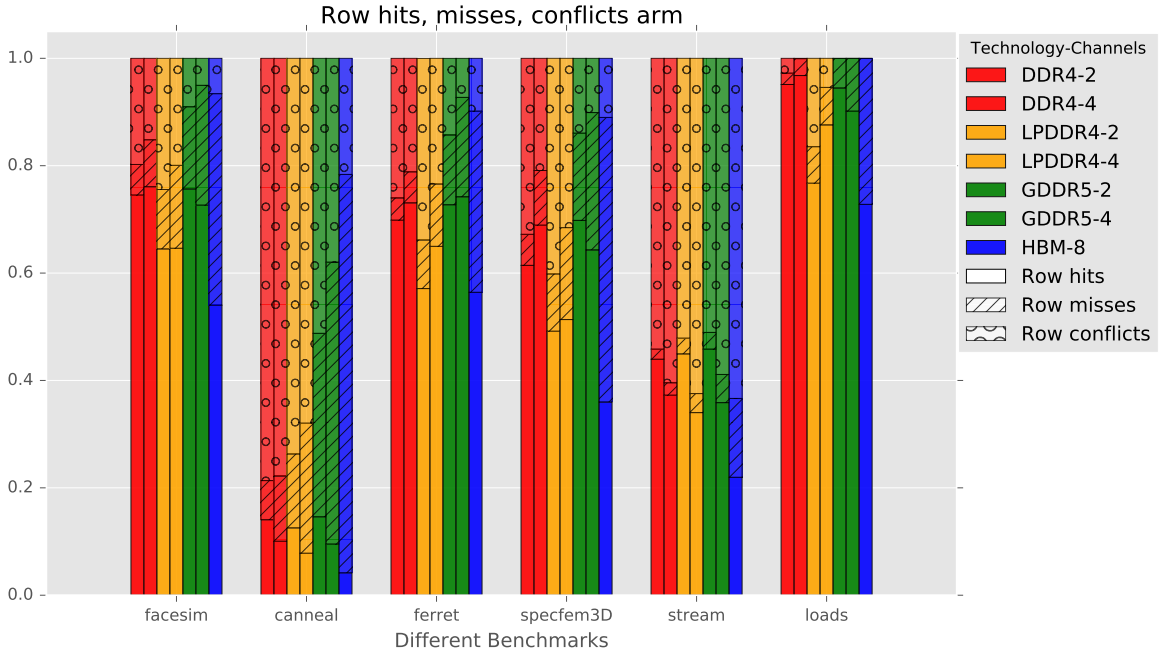

Fig. 6.8 Row hits, misses and conflicts of benchmarks on **arm** with different memory technologies.

Chart 6.6 differentiates the row events for the marenostrum configuration. It can be seen a common pattern through all the standards that the row hits decreases when more channels are added and by the memory access pattern.

Adding more channels in an ideal situation would mean that there are more rows to be opened and hence it is quite prone to get more misses and fewer conflicts. This tendency is clearly seen in all the benchmark except in stream and loads. Stream and loads have a good sequential memory pattern access which in contrast results in more conflicts.

When comparing different memory technologies the result is that DDR4 and GDDR5 get more row hits which is the best situation comparing to the miss or conflict. However, the latencies of this standards can not compete to HBM which although having the worst rate of hits it is the fastest memory saw at the execution cycles charts. In addition to that, HBM has a fewer rate of conflicts having more misses which helps having less latency.

On the other hand, when comparing different CPU configurations, there is an interesting inclination of having less row hits in marenostrum, more in i7 and the best result in arm. This inclination could be due to having less last level cache size and hence a good temporal locality behavior could be helping getting this rate.

Finally, for i7 and arm configurations when more channels are added, the benchmark ferret gets more row hits which contradicts the tendency. This fact has aimed another study to try to determine the origin of this result which is explained in the following section. This tendency is also broken by some simulations with LPDDR4 that do not perform as well as expected.

6.4 Distribution of Commands Across Banks

The interesting results in ferret have encouraged a deep study about it. The given assumption to ferret was a strange memory access pattern which with the help of a distribution memory access chart will confirm this assumption. This will also help visualize if there is a tendency in distribution or not across the different DRAM standards.

After trying different metrics given by Ramulator, the best for understanding ferret is the amount of commands issued by the memory controller to the banks of the DRAM. In a fancy way, this number tells the interaction of the memory controller with the banks and if the memory requests are equally distributed they should have an equally distributed amount of commands.

An usual chart for this kind of study, the violin plot, shows the minimum, maximum and median. It has been normalized in order to see relevant differences due to huge variations

across the benchmarks. Y-axis shows the amount of commands of a given bank normalized to the median and x-axis shows the benchmarks. The resulted charts are the following:

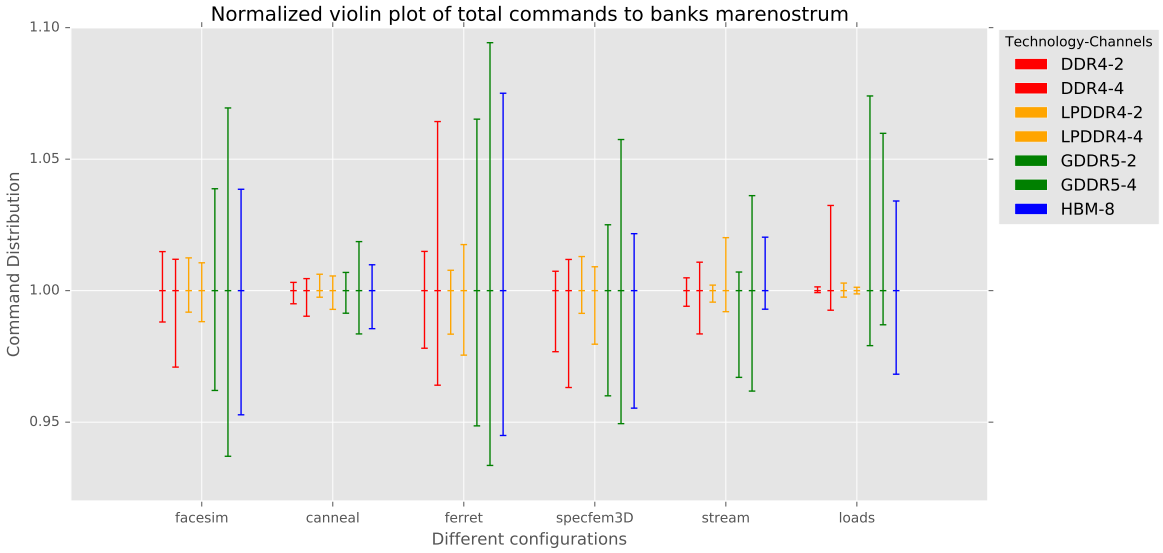

Fig. 6.9 Command distribution violin plot of benchmarks on **marenostrum** with different memory technologies.

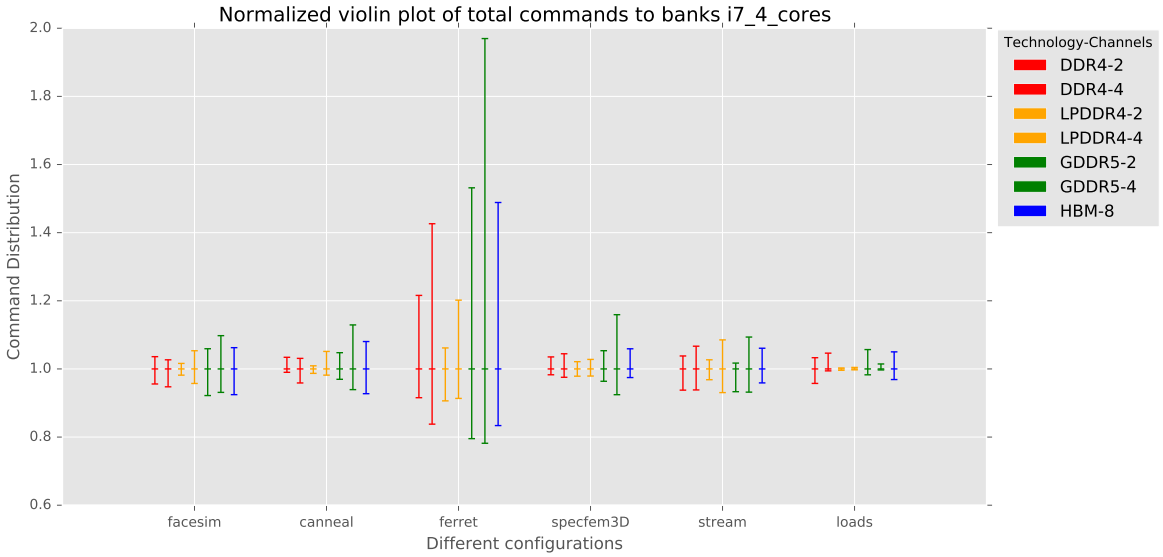

Fig. 6.10 Command distribution violin plot of benchmarks on **i7** with different memory technologies.

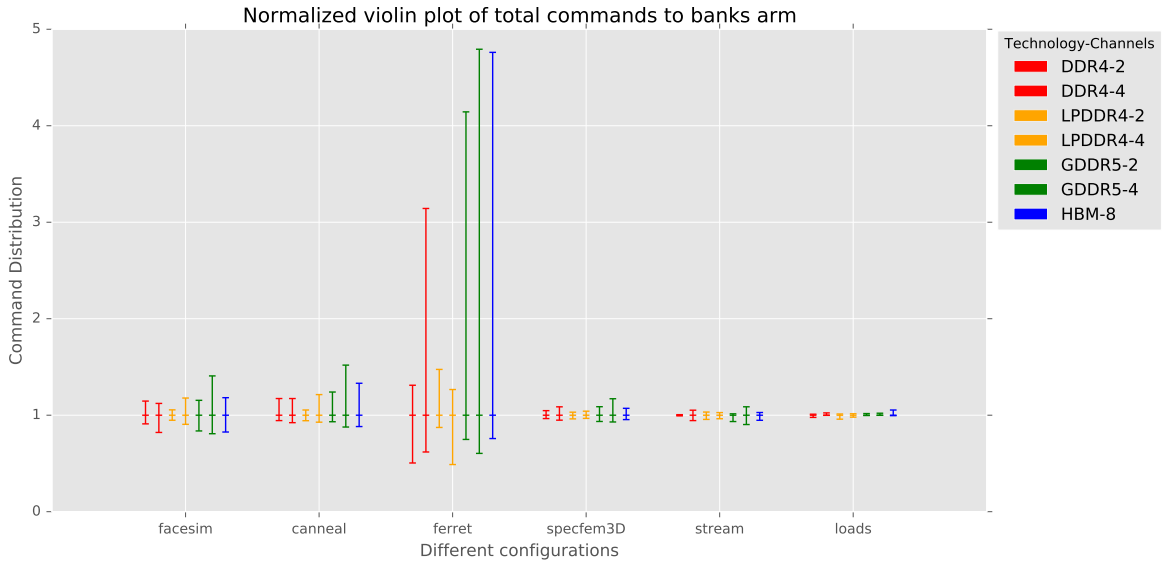

Fig. 6.11 Command distribution violin plot of benchmarks on **arm** with different memory technologies.

It has to be observed very cautiously because marenostrum chart shows big variations although in a little range comparing to the others. The maximum amount of commands is below a 10% comparing to the median. Following with the charts, i7 configuration shows a huge difference in ferret with nearly a 2x in the worst case.

Furthermore, arm chart shows nearly a 5x variation which follows the already observed tendency of arm working worse than i7 and marenostrum. This is a great metric to confirm that in i7 and arm configuration due to the memory access pattern and the cache memory hierarchy ends having a bad distribution of the memory accesses across the banks.

Comparing all standards, GDDR5 performs very badly in distributing among the benchmarks and even worse when adding more channels. On the other hand, the best result is obtained by LPDDR4 which performs very good for all benchmarks.

In all benchmarks except ferret the commands are distributed across banks in a very uniform manner, with differences of less than 20%. This indicates that the amount of commands that has to be served by each bank is very similar, so the banks can work in parallel and there is no bottleneck in any of the banks.

However, this situation does not happen in ferret, where some banks have to serve many more commands than the rest of the banks, causing important performance penalties and limiting the parallelism in the DRAM module. This is caused by the memory access pattern of ferret.

Chapter 7

Conclusions

The memory wall problem has been present during all these years without getting the needed attention. While Moore's law has been predicted correctly the increase of CPU performance, DRAM performance has been not growing at the same rate. This has created the above problem and it is expected to be worse in the next years. We need better and more powerful memory technologies.

As already been seen, the most promising memory technologies right now are the ones taking advantage of 3D-stacking above the chip like HBM, HMC and WIO. However, to study their performance in today's multicore architectures there are not enough tools but being Ramulator one of the best at the moment. In fact, it can be used to simulate HBM and WIO which make possible this mentioned study.

This is the reason why in this project, a bridge of Ramulator with TaskSim has been done, then it has been validated to be sure it was working correctly in order to be used to study the impact of memory technologies in parallel applications. The bridge has resulted in an increase of the simulation time in about 20%, but the extra realism added in the simulations are very satisfactory.

Moreover, this project has also analyzed how different memory technologies affects parallel programs. This analysis shows that some benchmarks are more sensitive to memory than others, and the MPKI is a good metric to identify them. In addition, the memory access pattern of the applications also plays an important role.

One the one hand, the access pattern of most benchmarks causes an uniform distribution of commands across the banks of the DRAM, so they can all serve requests in parallel. In these cases, the speed at which memory requests are served determines performance, so the row hit/miss/conflict rates and the latency of the memory commands specified in the DRAM standard by JEDEC are very important. HBM is the doubtless winner due to its very low latencies.

On the other hand, in some cases like ferret the memory access pattern puts a lot of pressure on only a subset of the banks, creating a clear bottleneck. In this cases the memory performance is far from optimal.

Finally, it can be concluded that the projects objectives have been achieved successfully. The bridge has been done and validated and it will be of great help for the researchers that use TaskSim and further research that may be done in the future. Furthermore, it has been done a first study of the impact of different memory technologies in multicore architectures and some promising results have been obtained.

7.1 Contributions

The main contribution of the *Evaluating the Impact of Future Memory Technologies in the Design of Multicore Processors* project is the bridge of TaskSim with Ramulator. This bridge will enable to researchers that use TaskSim to simulate the DRAM memory with a bigger level of realism. Furthermore, this bridge could be modified in order to create new memories and hence further studies could be done.

Having created a necessary bridge for the TaskSim simulator has been sufficiently satisfying. However, the second goal of the project, making a study of the impact of the current and future memory technologies in multicore processors has been very learning rewarding. This project has allowed me to study the different DRAM memory technologies as also to understand why it is needed further research in this topic.

From one side, it has introduced me in the world of research while looking papers, reading them, making plots, simulating and for the most important, to apply a method. On the other side, it has also introduced me in the world of open source software, Ramulator is open source like many others simulators.

7.2 Future Work

First of all, due to limitations of time, the study of the impact of different memory technologies on multicore architectures have not been extended in more CPU configurations and more memory intensive benchmarks. Although it has resulted in a good first study, more configurations and more benchmarks can be added in furtherance of a richer study.

Another different approach could be modifying Ramulator which is possible and hence create for example an heterogeneous memory system. This could be an interesting future work, modify the bridge in order to simulate an hybrid memory hierarchy similar to the

one in Intel Knight's Landing. This kind of machines are actually installed in the KNL[1] section in MareNostrum III which could make an even better study of comparison between the simulation and the actual machine, which has never been done yet.

Furthermore, Ramulator offers the feature of being quite simple making new standards. This could aim the proposal of a new memory design specifically considered for multicore architectures. This kind of study although it has already been proposed is more ambitious one and could be for a longer project.

Finally, being Ramulator open source, it could be possible to make some contributions as already been done in this project but for creating a new revision of a standard already been published by JEDEC. For example, up to date Ramulator does not offer the second version of HBM, HBM2. This could be relatively easy to be done as well as an interesting comparison of both standards.

[1] Intel Knights Landing processors.

Chapter 8

Project Management

This chapter is a summary of the initial project management course done in October 2016. In the next three sections ares described the methodology, stakeholders, project planning and finally the budget and sustainability.

8.1 Stakeholders, Obstacles and Methodology

8.1.1 Stakeholders

In this part, the stakeholders of the project are described which can be defined as people who are interested or direct beneficiaries of the project. Due to the scope of the project, the main interest of this project is for researchers using TaskSim.

Developer

In this case, I have not been only developer of Tasksim although for the bridge yes. Therefore, it is clear that I am interested in this project, not only for the thesis itself but also for the interest in learning how works in full depth the RAM.

Project Advisors

Both directors and the internal examiner are interested in further investigations that could result from this project including a future modeling of a hybrid memory hierarchy similar to Intel Knights Landing using this bridge.

Researchers that use Tasksim

Tasksim is a private software widely used in different European projects. It was the result of the PhD by Alejandro Rico and since then it has been used and developed at BSC. This project improves an existing feature using Ramulator as the DRAM cycle-accurate simulator instead of DramSim2.

8.1.2 Obstacles

During the development of the project, different obstacles may could have been occurred. They could have been both hardware or software which may could have been possible to avoid or not.

Hardware

In order to develop the project, different machines have been used. In the case that the MareNostrum III or the Arvei cluster stopped working I could have used a laptop and continue the project as a temporal solution.

On the other hand, if the laptop stopped working it would be required to find a solution like using the computers at UPC while the laptop is being repaired but no machine stopped working during the project.

Another problem that may could have occurred is a big waiting time in the queue before executing a job. This is a common problem of the clusters but using Arvei has alleviated it.

Software

However, the most problematic obstacles that may could have occurred are the software ones:

- Integrate two different software is difficult.
- In order to do the bridge, first, a learning process is required which has consumed time.
- Tasksim project compared to the Ramulator is enormous. It has been more difficult to deal with this simulator.
- The big clusters like Arvei or Marenostrum have a lot of problems with incompatibilities and installing the software may be in the worst case impossible. Fortunately, this has not been the case.

8.1.3 Methodology

For doing this project it has been followed the Scrum methodology. As it can be seen on the Scrum web page[13], it is an agile framework to complete complex projects, not only for software ones.

It has been chosen this methodology because it fits the requirements of the project advisors and because it is a set of rules and practices that enables to have a fast feedback and continual improvement.

Development cycle

As a big interesting requirement of the advisors, a weekly meeting with the advisors has been done in order to track the results of the week and plan new objectives for the next. This fits with the scrum practices because it allows to change or rapidly see any difficulty that may occur.

Validation

Although the project by itself would require little software generation, because of the fact that both simulators were already developed and only few classes were required for the bridge, the validation process was a big tedious labor. The validation of the bridge was done simulating and tracking an already known test to compare the results.

Tools

In order to follow this methodology and for being more transparent with the advisors a web Board Scrum called Trello [17] has been used.

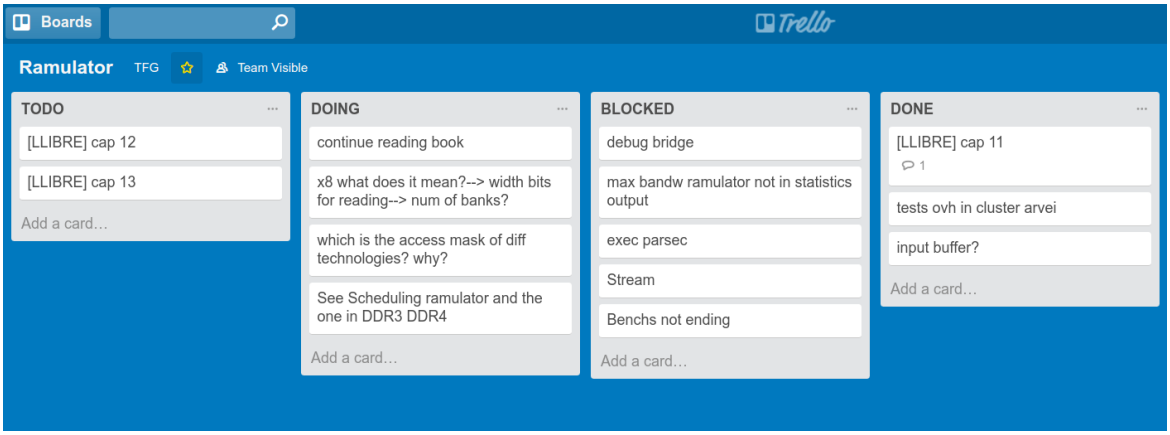

Fig. 8.1 Example of my Scrum board in Trello.

This has allowed me and my advisors to track the tasks over the week and also see the progress or problems that have occurred.

8.2 Project Planning

In this section the temporal planning of the project is given. It includes a brief description of all the tasks and a Gantt chart. It also contains the critical resources of each task and for some tasks, the possible alternatives in order to not miss the deadline.

The estimated duration of this project was from 1st September 2016 until one week before the date that final presentation is given, which is the 24th January 2017. So the final day of the project has been the 17th January.

8.2.1 Project Tasks

In this section, a full description of each task is given with the resources and possible alternatives. It has been assumed 20h of work per week. Another annotation is that for this project, there are three roles of human resources: advisor, developer and senior engineer.

Project management course

This task is mandatory and it has been done during the firsts weeks. The main goal was to learn the basics of project management including how to define the scope of a project, its time management, economic viability and basic presentation hints.

This task was divided in six deliverables that required a total of 75h summarized in five weeks. Although this was not a hard task, the time constrains increased its priority during these weeks.

For this task, a camera was the only unique resource from other tasks. The resources were as basic as a computer for doing the deliverables, LaTeXfor editing the document, a camera to record a presentation and a supervisor to provide some feedback.

Study of simulators

This first get in touch with the simulators was crucial for the correct development of the project. Each simulator was studied standalone to understand how work internally.

The expected time of this task was 20 hours per simulator making a total of 40 hours. In this task, was needed: a personal computer, a cluster like MareNostrum III or Arvei, both simulators and as a human resource, the engineer in charged of the development of TaskSim, Francesc.

Bridge

Making a bridge of two simulators might work or can be impossible but in this case, both simulator had already bridges which stated that a bridge was possible.

The expected time for this task was no longer than 40 hours because the main problem was the validation of the bridge. The resources were the same as the task of "Study of simulators".

If the bridge has not been possible to make, it would have required to think in another DRAM simulator to fulfill this gap or use the old bridge to did the study. Fortunately, it has not been the case.

Validation

This, by far, has been the most difficult task because it could have changed the future of the work or invalidate the previous work. We have been had to be sure that the bridge worked correctly by doing synthetics programs and also using benchmarks already known.

The estimated time has been wrong and in the end it has been used 80 hours. The resources were the usuals but it had required more attention from the engineer of TaskSim and from the advisors.

Evaluate different memory technologies for multicore processors

The study of the different memory technologies was one of the main goals of this project. It has mainly depended in what it has been learnt from the books and papers to fully understand and detect the bottlenecks of each technology.

This task has lasted 80 hours. The resources were mainly the same although it was needed some software to make the plots like some libraries in Python.

Study DRAM technologies

This task had no dependency although it was very important for the study when the bridge was done. This means that it has been done in parallel and in fact, during the study of Ramulator it has been very interesting to understand what was happening with different technologies.

This task has lasted until the end of the work because of the incremental development, although at some point it has been necessary to learn more about X technology. However, the expected time has been 60 hours.

The resources needed in this case were mainly material: a book of reference in Memory systems, any paper interesting that may appear and class material from subject like Computer Architecture.

Writing the memory

This task was nearly completely parallel to every task and had only the dependency of doing first the work that then it was needed to be written about. There were no additional or special resources needed to this. The length has been 60 hours distributed in four months.

Final presentation

It is expected to prepare the defense of this thesis in less than 25h.

8.2.2 Time Table

Tasks	**Hours**
1. Project management course	75
2. Study of simulators	40
3. Bridge	40
4. Validation	80
5. Evaluate memory technologies	80
6. Study DRAM technologies	60
7. Writing the Memory	60
8. Oral presentation	25
Total	**460**

Table 8.1 Hours per task.

8.2.3 Planning

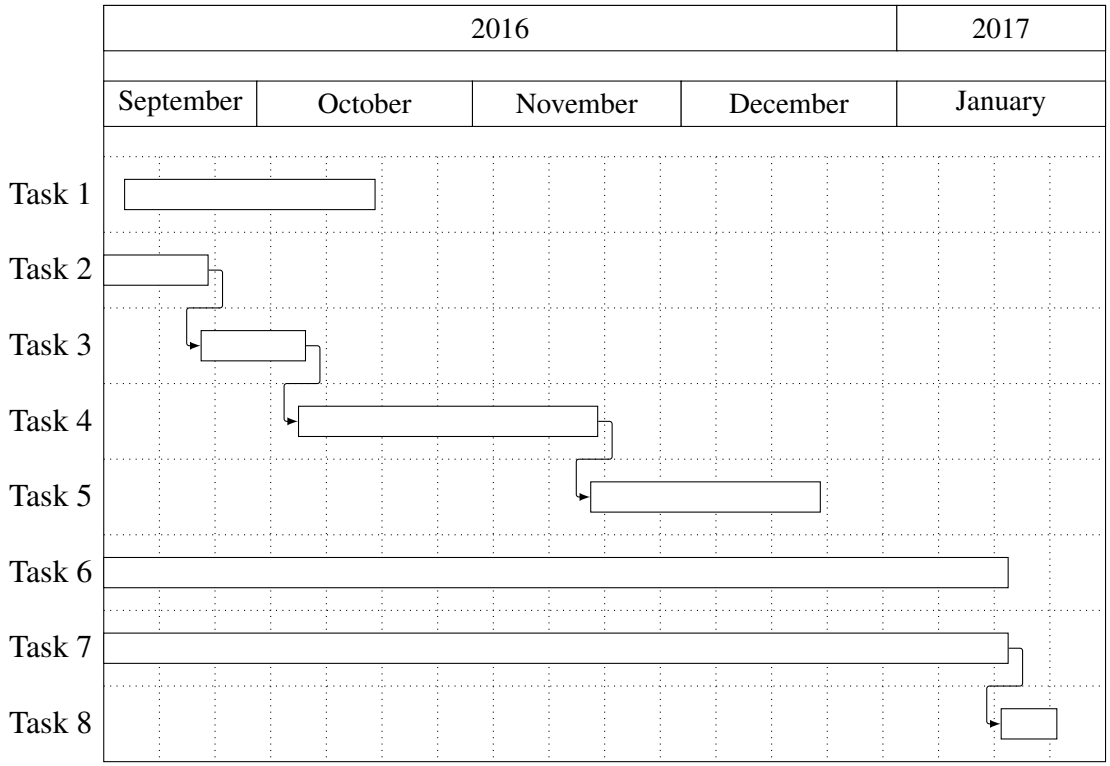

Fig. 8.2 Gantt chart.

8.2.4 Alternatives and Action Plan

During the development of the project it could have been normal that some deviations occurred. It's quite difficult to predict in term of hours the tasks. This is one of the reason that I have used the Scrum methodology in order to manage the possible problems that may could have occurred. Scrum is a dynamic methodology that permits the adaptation of the planning as the project advance.

Some rules were defined in order to do not miss any deadline:

- If a task takes less time than predicted, I start the following or I use this "free" time for writing the memory.
- If a task takes more time than predicted and it last a little bit more it does not matter and the following task will start later, I have reserved more time at the end for this reason.

Nevertheless, these deviations have not affected the assigned resources and the weekly meetings have helped to deal with these possible delays.

8.3 Budget and sustainability

In this chapter I analyze the estimated budget for this project and study its sustainability and viability. First, I present the cost evaluation and then, the sustainability and viability of the project.

8.3.1 Cost Evaluation

I have divided the costs in four groups: Human Resources, Hardware, Software and General Expenses.

Table 6.1 shows the costs of each part and the total. The contingency has been calculated as the 7.5% of the total costs (human resources, hardware, software and general expenses). I have chosen that percentage for any deviation of the budget that may could have occurred.

Category	**Price (€)**
Human Resources	8180,00
Hardware	174,66
Software	0,00
General Expenses	75,00
Contingency	656,22
Total	**9061.88**

Table 8.2 Global Cost.

Human Resources

Table 8.3 shows the costs of the human resources needed in this project. It also shows the cost per hour. Cost per hour is an estimation taken from others TFG's.

I have given the same amount of time to all of the advisors and the total estimation time comes from the weekly meetings(sept-dec) and have been 12 in total. Also, the price of the engineer comes from the fact that has helped me in the integration of the simulator inside TaskSim.

Human resources	**Price euros per hour**	**Hours**	**Cost (€)**
Advisor	30	100	3000
Senior Engineer	25	60	1500
Research assistant BSC	8	460	3680
Total		**620**	**8180**

Table 8.3 Human Resources price.

Hardware

Table 8.4 shows the actual expenses of all hardware components that this project have required. Every component was bought before the start of the project and I have omitted office-related objects, like a desk, a chair, pens because they have not been bought specifically for this project and other people can them after the project.

As there is no cost for the researchers at BSC to access to MareNostrum III and Arvei it has been omitted this part although for a private company it would have been necessary. This machines are running non-stop during nearly all the year and can be accessed remotely so I have made an estimation of hours per task. The depreciation is calculated with the following equation. The other number of hours are extracted from the planning. I have considered the lifetime of the hardware as 3 years.

$$\text{Depreciation}(€) = \frac{\text{cost}(€)}{3 \text{ years}} * \frac{1 \text{ year}}{365 \text{ days}} * \frac{1 \text{ day}}{4 \text{ hours}} * \#\text{ hours}$$

Hardware Resources	**Power (W)**	**Cost (€)**	**Time (h)**	**Total Cost (€)**
Dell XPS 13 (personal laptop)	45	1300	500	148,40
Keyboard	1	20	500	2,28
Mouse	1	10	500	1,14
Screen	20	200	500	22,84
Mare Nostrum III	1MW	22700000	1000*	0
Cluster Arvei	-	-	1000*	0
Total				**174,66**

Table 8.4 Hardware resources.

In table 8.4: (*) indicates an estimation.

Software

For this project, the required software has been free. So, it has not been any cost related to software. This is the list of software that I have used:

- Operating System: Ubuntu 16.04 LTS, SuSe Distribution 11 SP3 and Ubuntu 14.04.5 LTS
- For writing the memory: Mendeley and LaTeX.
- Text editor: Sublime Text3, CLion(free for students) and Vim

- Compiler: Mercurial(mcc), gcc 4.9, gcc 5.1, gcc 6.1
- Simulators: TaskSim and Ramulator.
- Runtime System: Nanox
- Pin Instrumentation tool
- Cloud storage system: Google Drive
- Pdf visualizer: Document Viewer
- Web Searcher: Google Chrome

General expenses

The main place of developing of this project has been inside BSC and the regular price of kWh in Spain has been between 0.10 € and 0.20 € so I have chosen 0.15: 500h*0.15/kWh=75 euros. I have not counted the amount energy consumed by the machines as I do not have to pay anything.

8.3.2 Budget Monitoring

In table 8.5 it is showed the cost of each task for the resources of human and hardware because they are the most important resources. For the human cost, task 2 to task 6 are the only tasks that have required the help of the senior engineer. On the other hand, all the tasks have required the researcher assistant and the advisors, hence it has been calculated proportionally. Finally, Material cost has also been calculated proportionally.

Task	Human cost (€)	Material Cost (€)	Total (€)
Task 1	1333,70	28,47	1362,17
Task 2	711,30	15,19	726,49
Task 3	711,30	15,19	726,49
Task 4	1422,61	30,38	1452.99
Task 5	1422,61	30,38	1452.99
Task 6	1066,96	22,78	1089,74
Task 7	1066,96	22,78	1089,74
Task 8	444,56	9,49	454,05
Total	**8180**	**174,66**	**8354.66**

Table 8.5 Cost per task.

8.3.3 Sustainability and Viability

Economic

I have given a rating of 8 for this part. The estimated cost of the project is 9061,88€ and it is very adjusted and proportional for the work of each task. I will reuse for the first part of the project two simulators so the only cost is the bridge. For the second part, it does not really apply although I have used some papers and benchmarks that had already been traced.

Social

I have given a rating of 9 because one of the objectives of the project has been to evaluate the DRAM technologies for the design of the next generation of multicores processors so in the end, if a new memory technology is discovered it is going to be beneficial for the response time of every device. This means that nearly each device in the world could possibly consume less.

Furthermore, the community of researchers that use TaskSim will be more than pleased for the bridge. Finally, this project has helped me to understand in depth DRAM memory technologies, the world of research and simulators, so by my point of view it has been very beneficial.

Environmental

I have given a rating of 8.5 because after the ending of this project the software is going to be available to the people that use TaskSim so they may use the software of this project. Furthermore, two simulators have been reused and one of them is more environmentally-friendly than its competitors.

For the part of power consumption, two clusters have been used which have been shared with a lot of people. In the end I may have added some tasks to the queue but I have used responsibly.

Viability

This project is part of a bigger one which is called RoMoL. As product it is very viable because the increase of realism of a DRAM simulation could make a point in the following years.

References

[1] HBM Image Web: https://www.amd.com/es-xl/innovations/software-technologies/hbm. Last accessed: January 2017.

[2] JEDEC. JESD209-4 Low Power Double Data Rate 3 (LPDDR4), Aug. 2014.

[3] JEDEC. JESD212 GDDR5 SGRAM, Dec. 2009.

[4] JEDEC. JESD235 High Bandwidth Memory (HBM) DRAM, Oct. 2013.

[5] JEDEC. JESD79-4 DDR4 SDRAM, Sept. 2012.

[6] Memory Bank Partitions Image Web: https://www.sei.cmu.edu/cyber-physical/research/timing-verification/multicore-scheduling-cont.cfm. Last accessed: January 2017.

[7] Memory Wall Problem and Moore's law Image Web: https://www.extremetech.com/computing/185797-forget-moores-law-hot-and-slow-dram-is-a-major-roadblock-to-exascale-and-beyond. Last accessed: January 2017.

[8] Mercurium Web: https://pm.bsc.es/projects/mcxx Last accessed: January 2017.

[9] Nanos++ Web: https://pm.bsc.es/nanox Last accessed: January 2017.

[10] Parsec Benchmark Suite Web: http://parsec.cs.princeton.edu/overview.htm Last accessed: January 2017.

[11] Ramulator Github Page: https://github.com/cmu-safari/ramulator. Last accessed: January 2017.

[12] Redmine Web: https://www.redmine.org/ Last accessed: January 2017.

[13] Scrum Alliance Web: https://www.scrumalliance.org/why-scrum. Last accessed: October 2016.

[14] Specfem3d Benchmark Web: https://geodynamics.org/cig/software/specfem3d/ Last accessed: January 2017.

[15] Stream Benchmark Web: https://www.cs.virginia.edu/stream/ref.html Last accessed: January 2017.

[16] Tasksim Web: http://www.bsc.es/computer-sciences/computer-architecture/tasksim. Last accessed: January 2017.

[17] Trello Web: https://trello.com/. Last accessed: January 2016.

[18] BINKERT, N., SARDASHTI, S., SEN, R., SEWELL, K., SHOAIB, M., VAISH, N., HILL, M. D. M. D., WOOD, D. A. D. A., BECKMANN, B., BLACK, G., REINHARDT, S. K. S. K., SAIDI, A., BASU, A., HESTNESS, J., HOWER, D. R., KRISHNA, T., BASIL, A., HESTNESS, J., HOWER, D. R., KRISHNA, T., SARDASHTI, S., SEN, R., SEWELL, K., SHOAIB, M., VAISH, N., HILL, M. D. M. D., AND WOOD, D. A. D. A. The gem5 Simulator. *Computer Architecture News 39*, 2 (2011), 1.

[19] BINKERT, N. L., DRESLINSKI, R. G., HSU, L. R., LIM, K. T., SAIDI, A. G., AND REINHARDT, S. K. The m5 simulator: Modeling networked systems. *IEEE Micro 26*, 4 (July 2006), 52–60.

[20] CHAMPIONSHIP, S., CHATTERJEE, N., BALASUBRAMONIAN, R., SHEVGOOR, M., PUGSLEY, S. H., UDIPI, A. N., SHAFIEE, A., AWASTHI, M., AND CHISHTI, Z. USIMM: the Utah Simulated Memory Module (February 2012).

[21] CHANG, K. K., KASHYAP, A., HASSAN, H., GHOSE, S., HSIEH, K., LEE, D., LI, T., PEKHIMENKO, G., KHAN, S., AND MUTLU, O. Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization. *SIGMETRICS Perform. Eval. Rev. 44*, 1 (June 2016), 323–336.

[22] CHASAPIS, D., CASAS, M., MORETÓ, M., VIDAL, R., AYGUADÉ, E., LABARTA, J., AND VALERO, M. Parsecss: Evaluating the impact of task parallelism in the parsec benchmark suite. *ACM Trans. Archit. Code Optim. 12*, 4 (Dec. 2015), 41:1–41:22.

[23] CHULWOO KIM, H.-W. L., AND SONG, J. Memory Interfaces. *IEEE Solid-State Circuits Magazine*, 1 (2006), 23–34.

[24] GRASS, T., ALLANDE, C., ARMEJACH, A., RICO, A., AYGUADÉ, E., LABARTA, J., VALERO, M., CASAS, M., AND MORETO, M. Musa: A multi-level simulation approach for next-generation hpc machines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2016), SC '16, IEEE Press, pp. 45:1–45:12.

[25] GUPTA, M., ROBERTS, D., MESWANI, M., SRIDHARAN, V., TULLSEN, D., AND GUPTA, R. Reliability and performance trade-off study of heterogeneous memories. In *Proceedings of the Second International Symposium on Memory Systems* (New York, NY, USA, 2016), MEMSYS '16, ACM, pp. 395–401.

[26] HENNESSY, J. L., AND PATTERSON, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[27] J. GONZALEZ, J. GIMENEZ, M. C. M. M. A. R. J. L., AND VALERO., M. Simulating Whole Supercomputer Applications. IEEE Micro. Vol. 31, no. 3, May/June 2011 .

[28] JACOB, B., NG, S., AND WANG, D. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[29] KIM, Y., SESHADRI, V., LEE, D., LIU, J., AND MUTLU, O. A case for exploiting subarray-level parallelism (salp) in dram. *SIGARCH Comput. Archit. News 40*, 3 (June 2012), 368–379.

[30] KIM, Y., YANG, W., AND MUTLU, O. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters PP*, 99 (2015), 1–1.

[31] M. CASAS, M. MORETÓ, L. A. E. C. D. C. T. H. L. J. O. P. O. S. U. A. C. E. A. J. L., AND VALERO, M. Runtime-Aware Architectures. Euro-Par 2015, Vienna, Austria, pages 16-27.

[32] M. VALERO, M. MORETÓ, M. C. E. A., AND LABARTA., J. Runtime-Aware Architectures: A First Approach. International Journal on Supercomputing Frontiers and Innovations (SuperFRI), vol. 1, issue 1, pages 29-44, June 2014.

[33] MARTIN, M. M. K., SORIN, D. J., BECKMANN, B. M., MARTY, M. R., XU, M., ALAMELDEEN, A. R., MOORE, K. E., HILL, M. D., AND WOOD, D. A. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News 33*, 4 (Nov. 2005), 92–99.

[34] RICO, A., DURAN, A., CABARCAS, F., ETSION, Y., RAMIREZ, A., AND VALERO, M. Trace-driven simulation of multithreaded applications. *ISPASS 2011 - IEEE International Symposium on Performance Analysis of Systems and Software* (2011), 87–96.

[35] ROSENFELD, P., COOPER-BALIS, E., AND JACOB, B. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters 10*, 1 (2011), 16–19.

[36] SITES, R. R. sites. it's the memory, stupid! microprocessor report, 10(10):2–3, aug. 1996.

[37] T. GRASS, A. RICO, M. C. M. M., AND AYGUADE., E. TaskPoint: Sampled Simulation of Task-Based Programs. International Symposium on Performance Analysis of Systems and Software (ISPASS), Uppsala, Norway, April 2016.

[38] TRAN, K. Scaling the Next-Generation System Architecture. *IEEE Solid-State Circuits Magazine 8*, 2 (June 2016), 35–42.

[39] UDIPI, A. N., MURALIMANOHAR, N., CHATTERJEE, N., BALASUBRAMONIAN, R., DAVIS, A., AND JOUPPI, N. P. Rethinking dram design and organization for energy-constrained multi-cores. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 175–186.

Appendix A

Configuration File TaskSim

```
ncpus = 8
threads_per_cpu = 1 #ignored (not supported yet)
mode_selector = MEMORY
idle_cycles = 10000
forward_task_in_out_to_cpu = false
measure = FULL_APPLICATION # PARALLEL_SECTIONS
master_speedup_ratio = 20.0
cpu_freq_mhz = 1000

[Paraver]
sampling_policy = PERIODIC
hardware_sampling_interval = 100000
trace_base_name = test_instrumentation
pcf_filename = ./example.pcf
modules = CPU RAM TLB CACHE RAM
# Posible modules CPU CACHE NC IN MC DMA TLB RAM

[Burst]
perf_ratio = 1

[MemCPU]
out_buff_size = 100
rob_size = 168
issue_rate = 4
commit_rate = 4
```

```
[DL1Cache]
mshr = DL1MSHR
num-ports = 1
latency = 4
size = 32768
line-size = 64
word-size = 4
assoc = 8
level = 1
victim-lines = 0

[DL1MSHR]
size = 32

[L2Cache]
mshr = L2MSHR
num-ports = 1
latency = 11
size = 262144
line-size = 64
word-size = 4
assoc = 8
level = 2
victim-lines = 0

[L2MSHR]
size = 32

[L3Cache]
mshr = L3MSHR
num-ports = 1
latency = 28
size = 20971520
line-size = 64
word-size = 4
```

```
assoc = 20
level = 3
victim-lines = 0

[L3MSHR]
size = 128

[Memory]
bandwidth = 10250000000 # [bytes/s]
request-size = 64
latency = 200  # [ns]
input-buffer = 128

[L3Bus]
latency = 10
width = 8

[DRAM]
frequency-divider = 5
cas = 11
ras = 11
precharge = 11
access-mask = rrrrrrrrrrrrrRRRRBBBBhhhhhhhhhhbbb
megs_of_ram = 16384
NUM_CHANS = 4
tCK = 5
input-buffer = 4

[MMU]
page-size = 8192
#input-access-profile = access_profile_NEMO_4p.dat
access-priority = total
#output-access-profile = access_profile_NEMO_4p.dat
allocation-policy = dynamic_2
empty-page-threshold = 16
backward-migration-factor = 0.4
```

```
[ModeSelector]
fast_forward_limit = 1000000000
num_samples_history = 4
sampling_cut_off = 15
num_warmup_instances = 1
num_warmup_instances_start = 2
sample_replacement_policy = REPLACE_ALL
serialization_file = NULL
```

Appendix B

Configuration File Ramulator

```
########################
# Example config file
# Comments start with #
# There are restrictions for valid channel/rank numbers
 standard = DDR4
 channels = 4
 ranks = 1
 speed = DDR4_2400R
 org = DDR4_4Gb_x16
# record_cmd_trace: (default is off): on, off
 record_cmd_trace = off
# print_cmd_trace: (default is off): on, off
 print_cmd_trace = off

### Below are parameters only for CPU trace
 cpu_tick = 8
 mem_tick = 3
### Below are parameters only for multicore mode
# When early_exit is on, all cores will be terminated when the earliest
 early_exit = on
# early_exit = on, off (default value is on)
# If expected_limit_insts is set, some per-core statistics will be reco
 expected_limit_insts = 200000000
 cache = no
# cache = no, L1L2, L3, all (default value is no)
```

```
  translation = None
# translation = None, Random (default value is None)
#
#########################
```