UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Posgrados

Architecture to System Level Analysis of DMTJ-based Cache Memory

José Félix Chávez Jácome

Marco Lanuzza, Ph.D.
Director de Trabajo de Titulación

Trabajo de titulación de posgrado presentado como requisito
para la obtención del título de: Magister en Nanoelectrónica,

Quito, 29 de diciembre de 2019

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Posgrados

HOJA DE APROBACIÓN DE TRABAJO DE TITULACIÓN

Architecture to System Level Analysis of DMTJ-based Cache Memory

José Félix Chávez Jácome

	Signature
Prof. Marco Lanuzza, Ph.D. Director del Trabajo de Titulación	[illegible]
Omar Aguirre, Ph.D. Director de la Maestría en Nanotecnología	______________________
César Zambrano, Ph.D. Decano del Colegio de Ciencias e Ingenierías	______________________
Hugo Burgos, Ph.D. Decano del Colegio de Posgrados	______________________

Quito, 29 de diciembre de 2019

© Derechos de Autor

Firma del estudiante:	
Nombre:	José Félix Chávez Jácome
Código:	00140255
C. I.:	1722625199
Fecha	Quito, 29 de diciembre de 2019

DEDICATION

This research work is mainly dedicated to God who through his hand gives me wisdom, courage and strength to reach the desired goal.

My parents, to their love and unconditional support that were the engine that inspired me to continue every day, to their confidence placed in me that motivated me to achieve this triumph, it is a pride to have parents like them.

My brothers and their words of encouragement, I leave a grain of guidance that when we set out we can achieve our dreams.

My dear friends and girlfriend Alexandra who with their presence and their knowledge made this journey a pleasant and enriching experience.

I dedicate this triumph to all of you

José Félix Chávez Jácome

Gratitude

A special thanks to my friend Esteban Garzón, your guidance and patience made this research possible.

I thank to 2 great friends, Cinthia and Daniela, their advice in difficult times helped me make the right decisions.

Total thanks to the professors of the universities San Francisco (Quito - Ecuador) and Università della Calabria (Calabria - Italy) especially to Prof. Marco Lanuzza, tutor of this project, for giving me the opportunity to work in his research area.

Università della Calabria, for the opportunities offered to foreign students to specialize in a European university, to live in a multicultural environment and grow professionally.

To each and every one of you, thank you very much.

José Félix Chávez Jácome

RESUMEN

El presente trabajo detalla la construcción de una plantilla de simulación desde el nivel de arquitectura hasta el nivel de sistema considerando una tecnología FinFET de 0.8V con unión de túnel magnético de barrera simple y doble (MTJ). Para investigar su comportamiento dentro de una configuración de simulación real, se consideran características eléctricas extraídas en un entorno Cadence Virtuoso para la configuración de los parámetros de entrada que se refieren a un conjunto de memorias híbridas basadas en FinFET / MTJ que toman como característica la latencia, la corriente de leakage y la energía. Para este propósito, se diseñaron dos plantillas con características de single-core y multi-core utilizando un cierto grupo de beanchmarks (archivos binarios) para cada plantilla. De estas simulaciones se obtuvieron los siguientes datos: número de instrucciones simuladas, tiempo de simulación del host, número de datos leídos y datos escritos. Finalmente, determinamos el tiempo real de simulación, el número de instrucciones ejecutadas por segundo y la energía por instrucción. Por lo tanto, al final mostramos una evaluación del rendimiento y la energía presentada por las diferentes configuraciones de memoria.

Descriptores: plantilla de simulación, nivel de arquitectura, nivel de sistema, FinFET, MTJ, single-core, multi-core

ABSTRACT

The present work details the construction of a simulation framework from architecture to system level by considering a 0.8V FinFET technology and the single- and double-barrier magnetic tunnel junction (MTJ). In order to investigate their behavior within a realistic simulation setup, it is considered electrical characteristics extracted in Cadence Virtuoso environment to the configuration of the input parameters referring to a set of hybrid FinFET/MTJ-based memories taking as a characteristic the latency, leakage and energy. For this purpose, two frameworks were designed with single- and multi-core features using a certain set of benchmarks (binary files) for each framework. From these simulations were obtained the following data: number of simulated instructions, time host simulation, number of read data and write data. Finally, we determine the real simulation time, number of instructions executed per second and energy per instruction. Thus, in the end we show an evaluation of the performance and energy presented by the different memory configurations.

Descriptors: simulation framework, architecture to system level, FinFET, MTJ, single-core, multi-core.

Tabla de Contenido

FIGURES CONTENTS

TABLES CONTENTS

CHAPTER 1. INTRODUCTION AND LITERARY REVIEW

The study of nonvolatile memories (NVM) is a key point to face the actual challenges of power and speed presented in memories such as SRAM and DRAM due to the reduction of the semiconductor size in the industry. In order to investigate their behavior within a real time simulation system. An attractive solution is the development of caches by using NVMs, which allows the system to operate in a sleep mode without losing information, thus reducing the leakage current and static power consumption. Within NVMs, spin-transfer torque magnetic random access memory (STT-MRAM) are excellent candidates for their potential to offer low power consumption, high densities and high speeds. The main block of STT-MRAM is the Magnetic tunnel junction (MTJ), which is mainly composed of three layers, one layer as an oxide barrier and other ferromagnetic layers. The analysis of this work is based on two types of MTJs, single-barrier MTJ (SMTJ) and double-barrier MTJ (DMTJ). The latter has two oxide layers in order to reduce the critical current I_{c0} and so the write energy (Zhang, y otros, 2017).

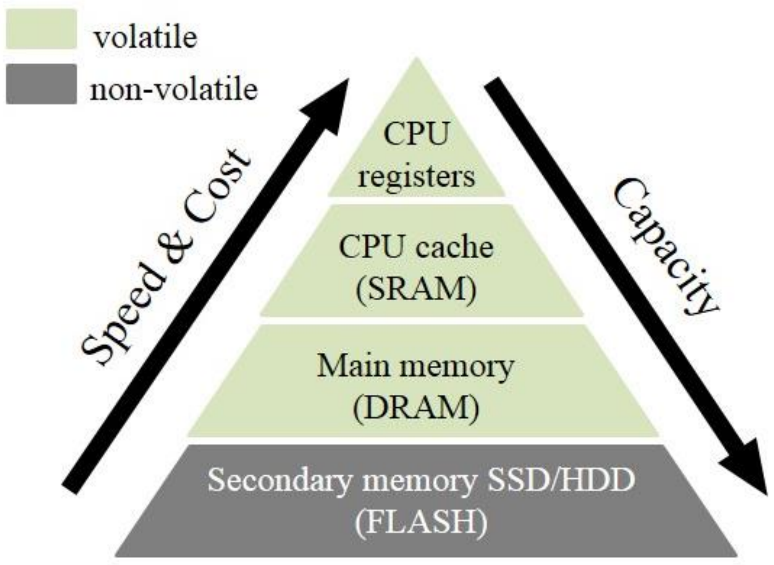


Figure 1: System level memory hierarchy (Turmero, s.f.)

The system level memory hierarchy is shown in Figure 1. It is seen that as we are closer to the highest hierarchical level, the greater the need to develop memories that cover these limits, the hybrid FinFET/MTJ-based memories that we will analyze in this work are good candidates to take into consideration. A complete analysis regarding the design of devices (from the circuit design to implementation at the system level) is made up of three stages which demand an exhaustive investigation as well as the support of computer tools (softwares) that facilitate the flow of simulation and analysis. Figure 2 shows this hierarchy (Bishnoi, Ebrahimi, Oboril, & Tahoori), and in the following is briefly described its levels.

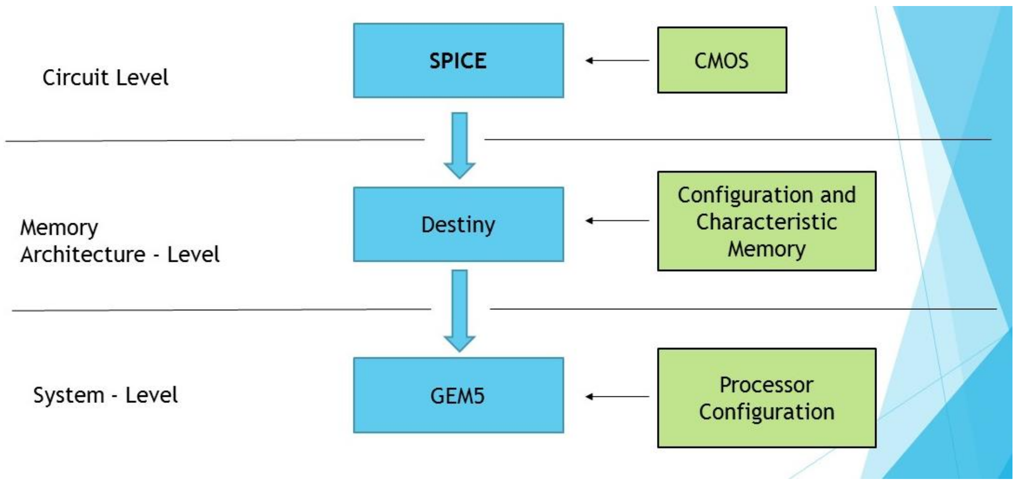


Figure 2: Simulation Flow Tools

Circuit Level or Device Level: This stage of development corresponds to the design of the device as an independent circuit taking into account its individual behavior establishing its static and dynamic parameters (sizing, leakage current, V threshold). This stage is strongly complemented with design and simulation software such as LTspice, Cadence, Sentaurus.

Architecture Level or Memory Level: This stage includes the set of many devices forming matrices. Then these matrices together with support circuits (decoders, pre-decoders, pre-charger, multiplexers, sens. Amplifiers) make up one more circuit complex capable of performing processes such as reading, writing and data retention within memory. The parameters highlighted in this level that we will use in our research are latency, energy and leakage current, among the simulation software for this level we have NV-Sim and Destiny. The latter is used in this work.

System Level: Regarding the last level of study, this part includes the memory within a real-time simulation system, the data delivered by the previous stage are loaded into the cache levels of our system. Thus configuring the different memory systems that we want to investigate. Finally, we obtain data such as simulation time, number of instructions executed per second, and energy per instruction. For such level, it is used the help of the GEM5 tool.

Simulators for architecture and system levels.

Among the levels mentioned above, this work focuses on the architecture and system levels, making further study in the latter.

Tools at architecture level.

There are many softwares used for the modeling of memories at the architecture level, tools such as CACTI or NVSIM are widely used for these purposes. However, they have limitations with respect to the types of memories that they are able to simulate such as modeling and research of kind 3D or multi-level-cell (MLC). For this work, we used another tool that was used in the last 10 years, it has shown an acceptable accuracy and flexibility for research purposes, it is known as DESTINY.

NVSim: This tool has its CACTI-based framework, so its libraries referring to memory technologies are similar. However, researchers use CACTI and the new CACTI (3DD) for technologies like SRAM / DRAM, while NVSim is mostly used for technologies of type NVM (non-volatile memory) (Syu, Shao, & Lin, 2–3 May 2013.). Nevertheless, both tools have different frameworks, so they use different types of data in the input of their frameworks and show different types of results. Such simulation is shown in the Table 1.

Table 1: Tools output parameters

File	**CACTI**	**NVSim**
output_formats	access_time/random_cycle_time	hit/miss/latency/energy/leakage_power

An advantage of NVSim is that it allows the configuration of its framework to prioritize simulation based on desired parameters such as area or leakage unlike CACTI that does not allow it (Syu, Shao, & Lin, 2–3 May 2013.). Another peculiarity of these tools are the different results that show regardless of whether they use the same configuration as input (See Table 2).

Table 2: Results of CACTI and NVSim for the same input: 32nm, 64 Block, 16 ways 4 MB (Mittal, Wang, & Vetter, 11/09/2017)

CACTI	**NVSim**
Superfice: 14 mm2	Superfice: 6 mm2
Leakage: 0,570 W	Leakage: 0,390 W
Access_time: 0,63 ns	Hit: 2000, miss: 0,31
Random_cycle_time: 3,1 ns	Write_latency: 1,08 ns
Read_energy: 0,18 nJ	Write_energy: 0,39 nJ

The results shown in Table 2 clearly indicate the strong influence of the configurations of the frameworks. Therefore, it is necessary to have much more comprehensive tools, which allow the researcher to use different options when modeling, as well as the use of new memory technologies in order to avoid incorrect conclusions.

Destiny: It is a modeling and design tool capable of working in 2D and 3D spaces, specialized in technologies such as: SRAM, eDRAM and NVMs. It has great advantages such as modeling of single-level cells (SLC) and multi-level cells (MLC) since it combines strongly the advantages of CACTI and NVSim tools.

Within the 2D SLC modeling uses the NVSim framework for SRAM, STTRAM, PCM, ReRAM and Flash technologies additionally, MLC modeling uses frameworks of type eDRAM, SOT-RAM, DWM, due to the combination this set of models allows DESTINY to perform a combination of 2D / 3D modeling as shown in the following Table 3:

Table 3: Models use open source tools (Mittal, Wang, & Vetter, 11/09/2017)

Tools/ Technologies	**SRAM**	**eDRAM**	**PCM**	**STT-RAM**	**ReRAM**	**SOT-RAM**	**Flash**	**DWM**
CACTI (3DD)	2D	2D/3D	X	X	X	X	X	X
NVSim	2D	X	2D, SLC					X
DESTINY	2D/3D		2D/3D, SLC/MLC			2D, SLC/MLC		

As we can see in Table 3, the DESTINY tool is much more complete and versatile, taking advantage of the tools above, also allowing the modeling of 2D/3D type architecture and together with the combination of increasing the density of the circuit through configurations. SLC and MLC.

DESTINY also presents a percentage of error with respect to the design and simulation, this has had an observation of up to 25% as in the worst-case reaching up to 10% for the best case, which makes DESTINY as an acceptable option by researchers (Dong, Xu, Jouppi, & Xie, 2012).

Tools at the system level.

Within this work, the part referring to the system level was worked through the GEM5 tool. It is a very complete tool in reference to other tools specifically oriented for the investigation of computer systems and architectures. Moreover, Gem5 is a simulation platform very used in the last 10 years.

Gem5: The Gem5 simulator is a simulation platform that results from the fusion of 2 tools: GEMS and M5. Gem5 uses their main advantages, the M5 simulator uses a highly configurable simulation framework which allows to use various types of machines such as ISAs as different CPUS models, while the GEMS simulator complements these advantages with a highly detailed and flexible control memory framework including cache parameter manipulation, bus communication protocols and interconnection models (Black, y otros, 2/05/2011).

These characteristics allow researchers to face new research goals, new dimensions or system configurations related to the use of multi-core processors or new cache hierarchies demanding a tool with a flexible simulation framework. Gem5 has responded satisfactorily to this need (Black, y otros, 2/05/2011).

Another advantage of GEM5 is that it allows researchers to collaborate with each other through its modular flexible simulation system. Many simulation tools have communication problems between researchers due to its complex open source code system, so GEM5 allows Researchers focus on a particular code depending on the desired configurations without understanding the entire base code. This facilitates communication between researchers in order to share information (Black, y otros, 2/05/2011).

The GEM5 simulator bases its success on 3 key aspects: Flexible modeling that allows many users to be manipulated, high availability for the research community and a high level of collaboration

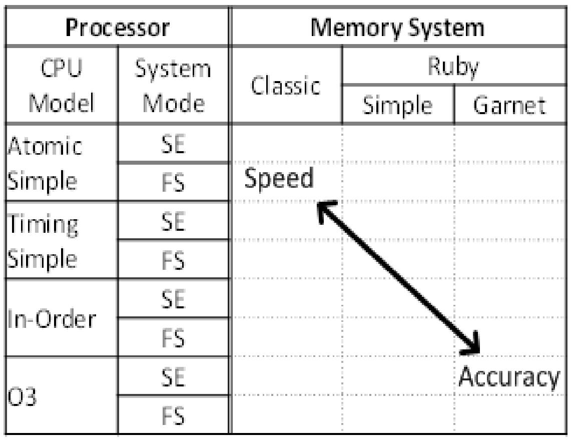
- ***Flexibility:*** It is a tool that allows you to evaluate several levels of systems in detail while maintaining a tread off between speed and accuracy, the tool provides a wide variety of options and components that make this wide flexibility possible. The capacity for flexibility depends on 3 key parameters:

<u>*CPU Model*</u>: The GEM5 tool provides 4 different types of CPU: *Atomic Simple* (Simple CPU Model) *Simple Timing* (Simple model but capable of simulating real-time memories) *In Order* (uses a pipeline configuration of CPU execution models in execution) *O3* (uses an out order configuration)

<u>*System Model:*</u> It has 2 simulation modes related to the system, *System-Call Emulation (SE)* a simpler system model that avoids the need to configure an operating system or input and output devices since it emulates most levels of system service. The second one is the *Full System (FS)*, a complete system that involves the configuration of an operating system, as well as input and output devices.

<u>*Memory System:*</u> the tool allows the use of two types of memory related to the system: Classic and Ruby. The Classic model is a reference of M5 and allows an easy and fast configuration of the system memory while Ruby belongs to GEMS provides a more infrastructure accurate and flexible.

- ***Availability:*** One of the advantages of GEM5 is its availability for different types of users, from researchers, industry engineers to students are able to manipulate GEM5 for their design purposes.
- ***High level of collaboration:*** Thanks to the platform presented by hundreds of researchers, they make great efforts every year to maintain flexibility in designs that require complex simulation systems such as full systems as well as provide high capacity in the configuration of system components (Black, y otros, 2/05/2011). Below we show a graph (Figure 3) showing the tread off that Gem5 maintains between its framework and its different configuration parameters that allow its flexibility.



Processor		Memory System		
CPU Model	System Mode	Classic	Ruby	
			Simple	Garnet
Atomic Simple	SE			
	FS	Speed		
Timing Simple	SE			
	FS			
In-Order	SE			
	FS			
O3	SE			Accuracy
	FS			

Figure 3: Speed vs. Accuracy Trend (Syu, Shao, & Lin, 2–3 May 2013.)

Multi-core simulation systems

In terms of electronics, multi-core indicates the presence of two or more cores within the same system or processor, multi-core systems are used for the simulation of heavy benchmark files that have an extremely high number of instructions than when are simulated in a system of single core the simulation would take hours to days, for this work we will take advantage of the multi-core system architecture that GEM5 provides us, additionally it should be mentioned that the benchmark files for simulation are special files, created for this type of multi-core systems. In this way a multi-core simulation can gain speed while maintaining precision levels. Thus multi-core systems being one of the most attractive options when it comes to design and research.

Benchmarks File

From a computer concept a benchmark file is an application developed to determine the performance of a device, architecture or computer system. The benchmark is made up of an executable file and when it is initiated the execution of the benchmark, which executes a large number of instructions, will be performed by the computer system, thus measuring the performance of the system.

For our analysis we use 2 types of benchmarks: a group of benchmarks (MiBench - University of Michigan) used for the single-core system, while for the multi-core system we use Rodinia benchmarks.

CHAPTER 2: METHODOLOGY

Architecture level simulation framework

The research part begins with the analysis of five systems composed of 3 types of memories: SRAM, STT-MRAM SB (Single Barrier), STT-MRAM DB (Double Barrier). The five configurations are depicted in Table 4 and are analyzed in three different technologies (28nm, 24nm and 20nm).

Table 4: Memory systems for simulation

Tags	Cache Configuration. (L1+L2)
Sys_A	SRAM+SRAM
Sys_B	SRAM+SB
Sys_C	SRAM+DB
Sys_D	SB+SB
Sys_E	DB+DB

Additionally, the memory size corresponding to cache of level 2 or L2-Cache will vary in 5 sizes: 128kB, 256kB, 512kB, 1MB and 2MB while for cache of level 1 or L1Cache it will be only one size: 32kB.

Once the initial configuration part is established, we need to start determining the main characteristics of these memories taking into account parameters such as: size and type of memory. In this stage of architecture, we need to find the main characteristics of a cache memory (timing, energy, leakage power) remembering that a memory is nothing more than a matrix array of subgroups that in turn these subgroups are made up of matrix arrays of bit-cells or as we best know the FinFET device. We have used the DESTINY tool which allows us to enter parameters such as Banks, Mat, Sub -arrays, in the figure 4 we show the framework used by the DESTINY tool.

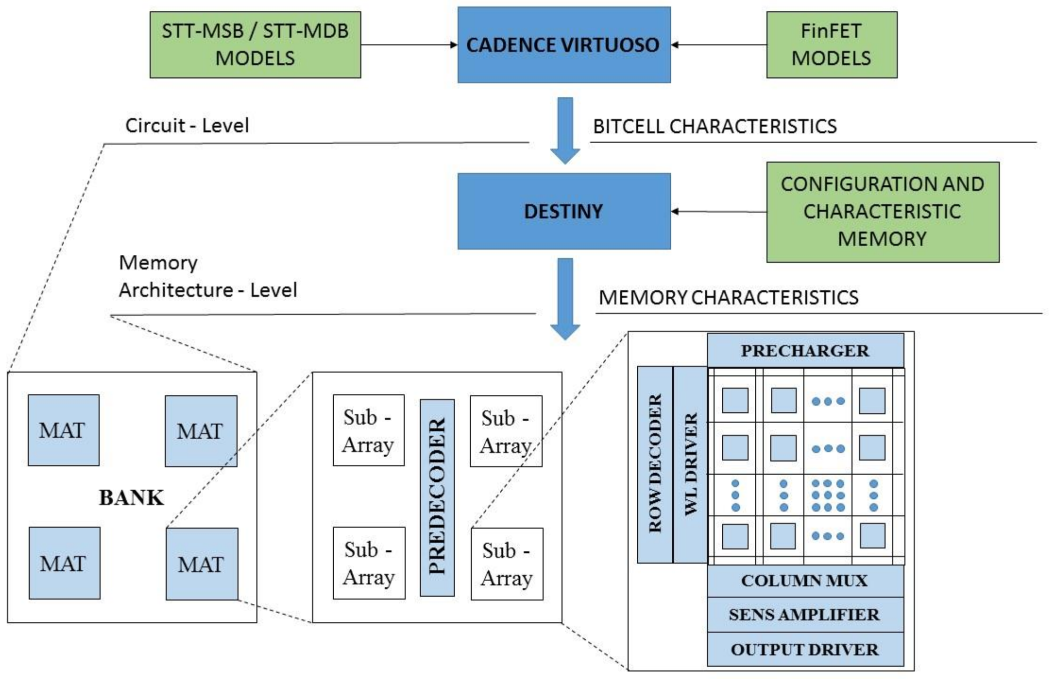


Figure 4: Simulation framework for STT-MRAMs in DESTINY

Bank: It is the highest level of architecture modeling. An NVM chip can have multiple layers of bank, the bank is a multifunctional memory unit that can operate independently, in each bank there are multiple mats that are connected to each other.

MAT: Multiple mats in a bank operate simultaneously to fulfill a memory operation, a mat consists of multiple sub-arrays and a pre-decoder block.

Subarrays: it is the most elementary structure in architecture modeling, each subarray consists of peripheral circuits, row of decoders, columns of multiplexers and driver output (Dong, Xu, Jouppi, & Xie, 2012).

Next, we show the input templates (Tables 5, 6, 7, 8) corresponding to the different types of memories, each of these templates was simulated for each of the 3 corresponding technologies (28nm, 24nm, 20nm), the present study does not focus on determining what is the process used by the DESTINY tool to arrive at the simulation results, but in explaining the data entered and the data extracted which will be used at the system level where the operation of the simulation is explained in more detail system.

Table 5: Main Template for RAM memory

Temperature (K)	300
Type	RAM
Word Width (bit)	512
Vdd	1V

Table 6: Number of Banks, Mats, Sub-Arrays for RAM memories

Size unit	**Size**	**Array**	**Bank**	**Mat**	**MuxSA**	**MuxLevel1**	**MuxLevel2**
kB	32	128x256	1x2, 1x2	2x2, 2x2	1	1	4
kB	128	256x256	1x4, 1x4	2x2, 2x2	1	1	8
kB	256	256x256	2x4, 1x4	2x2, 2x2	1	1	8
kB	512	256x256	4x4, 1x4	2x2, 2x2	1	1	8
MB	1	256x256	8x4, 1x4	2x2, 2x2	1	1	8
MB	2	256x256	16x4, 1x4	2x2, 2x2	1	1	8

Table 7: Main Template for STT-MRAM memory

Temperature (K)	300
Capacity (MB)	1
Word Width (bit)	8
Vdd	1V

Table 8: Number of Banks, Mats, Sub-Arrays for STT-MRAM memories

Bank	Forced	2x2 2x2
Mat	Forced	2x2 2x2
MuxSA	Forced	16

Simulation data are presented in the annexes section (Annexes 1), which are ordered respectively according to: the five systems to be tested, the technological nodes and the cache sizes (L1-Cache and L2-Cache).

The latency parameters obtained are expressed in nano-seconds, due to the input characteristics for the simulation framework at the system level detailed below. The latency parameter can be entered in two ways: in nano-seconds or cycles of simulation (cycle is the representation how many times a latency value has been repeated). The simulation framework designed for the system accepts cycle latency so to make the conversion we take as reference the latency of the SRAM memory with the smallest size (32kB) as shown in the table 9

Table 9: Reference latency in the L2_Cache 1MB template for conversion to cycles.

		L1-Cache (32kB)		L2-Cache	
Nodes	**Configuration**	**Lat. Read (ns)**	**Lat. Write (ns)**	**Lat. Read (ns)**	**Lat. Write (ns)**
28	Sys_A	0,373	0,349	0,508	0,417
24	Sys_A	0,314	0,296	0,413	0,345
20	Sys_A	0,264	0,247	0,334	0,287

The reference latency in the 1MB template is the same for the rest of the templates. The SRAM memory is taken as a basis because our analysis will always be comparing the STT-MRAM memory systems with respect to the SRAM, so for each of the configurations the latency values are divided for the reference latency. In this way, the latency in cycles will be expressed as shown in the table 10.

Table 10: Latency cycles for the L2_Cache template of 1MB

				GEM5 Simulated Data			
				32KB		1MB	
Nodes	**System**			**L1-Cache-Exact**		**L2-Cache-Exact**	
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**
nm	**-**	**GHz**	**GHz**	**cycles**	**cycles**	**cycles**	**cycles**
28	Sys_A	2,68	2,68	1	0,934	1,361	1,116
	Sys_B	2,68	2,68	1	0,934	2,415	10,342
	Sys_C	2,68	2,68	1	0,934	3,825	4,051
	Sys_D	2,68	2,68	2,056	10,152	2,415	10,342
	Sys_E	2,68	2,68	3,622	3,938	3,825	4,051
24	Sys_A	3,18	3,18	1	0,943	1,314	1,099
	Sys_B	3,18	3,18	1	0,943	2,856	10,948
	Sys_C	3,18	3,18	1	0,943	4,739	4,116
	Sys_D	3,18	3,18	2,533	10,795	2,856	10,948
	Sys_E	3,18	3,18	4,556	4,028	4,739	4,116
20	Sys_A	3,79	3,79	1	0,937	1,264	1,086
	Sys_B	3,79	3,79	1	0,937	3,383	11,714
	Sys_C	3,79	3,79	1	0,937	6,191	4,192
	Sys_D	3,79	3,79	3,112	11,520	3,383	11,714
	Sys_E	3,79	3,79	6,011	4,088	6,191	4,192

The value of latency cycles must be an integer, so we make an approximation to an integer value. The latency cycles for the other templates can be found in Annexes 2. Moreover, determining the templates (latency in cycles, energy for reading and writing instruction, leakage power) for each of the sizes of L2-Cache (See in the Annexes 2). Now, we will continue with the simulation system.

System level simulation framework

This part referring to the Gem5 tool describes the development and configuration process for the single-core and multi-core systems responsible for simulating the different memory systems determined above.

Single-core system framework

This system is the main basis of this project, the simulations of the five memory systems are performed in this framework. Its development is based on the scheme proposed by the Gem5 tutorial (Jason, s.f.). Its scheme can be seen in the figure 5:

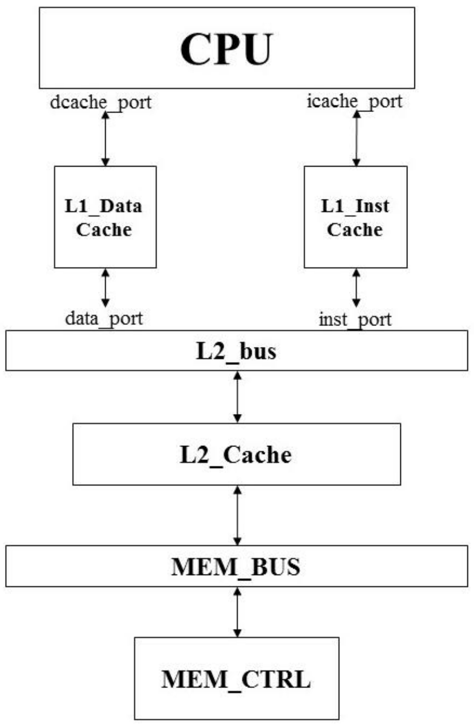


Figure 5: Single core system framework in Gem5

For our research purposes it was necessary to make modifications to the parameters of this framework, especially to the L1-Cache and L2-Cache blocks. The parameters (latency, energy, leakage power) belonging to the five memories systems must be entered in the parameters of the caches, in this way we can give the characteristics of the different types of memory to simulate. The most important parameter is the latency, for this purpose we have developed two solutions:

1. Adding buffers to the L1-cache (Data and Instruction) blocks that allow you to enter the latency (read and write) in seconds (Figure 6-a)
2. Configuring the L1-Cache blocks (Data and Instruction) that allow to manage the latency parameters in cycles (read and write) independently (Figure 6-b)

Both solutions were designed, however, we took option number 2 for subsequent analysis and results.

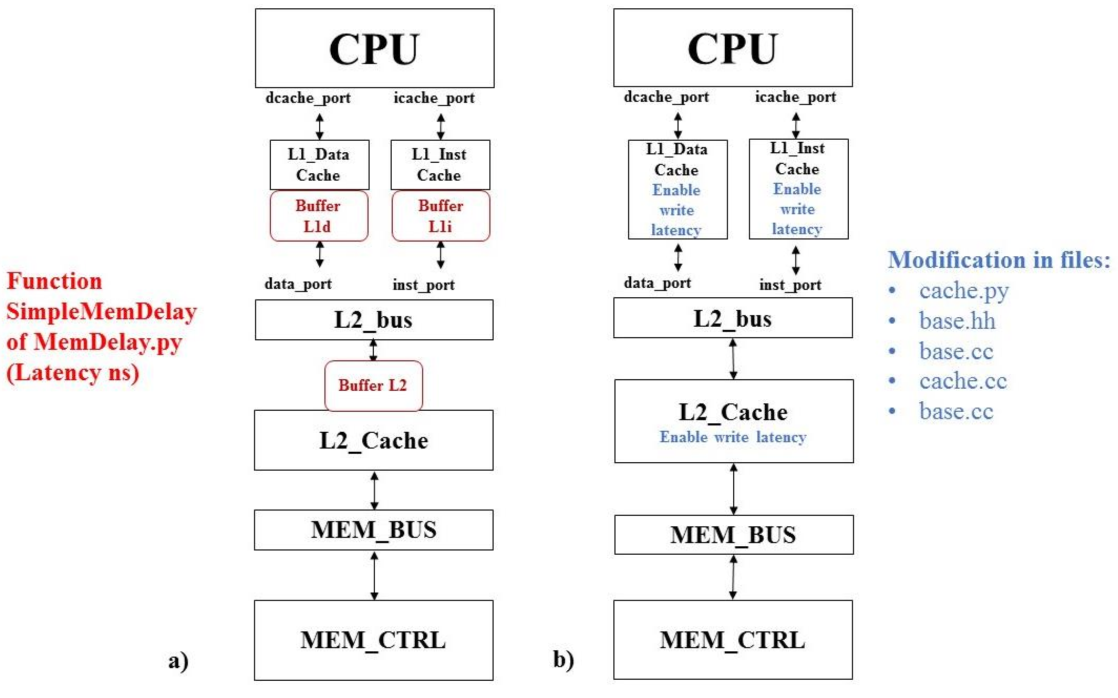


Figure 6: a) Framework single-core system with blocks of buffers for Latency (ns) b) Framework single-core system configuration for latency cycles read and write

Next, we detail the design process of the single-core system framework configuration for read and write latency cycles. Gem5 is a development and simulation environment whose interface is Linux, which employs the use of predefined libraries (configuration of the Caches, script files of the system framework, input parameters) that keep all the information related to the blocks that constitute our framework. It is Open source and uses C language for the base configuration and for development uses Python. The figure 7 shows the general interaction that these files handle.

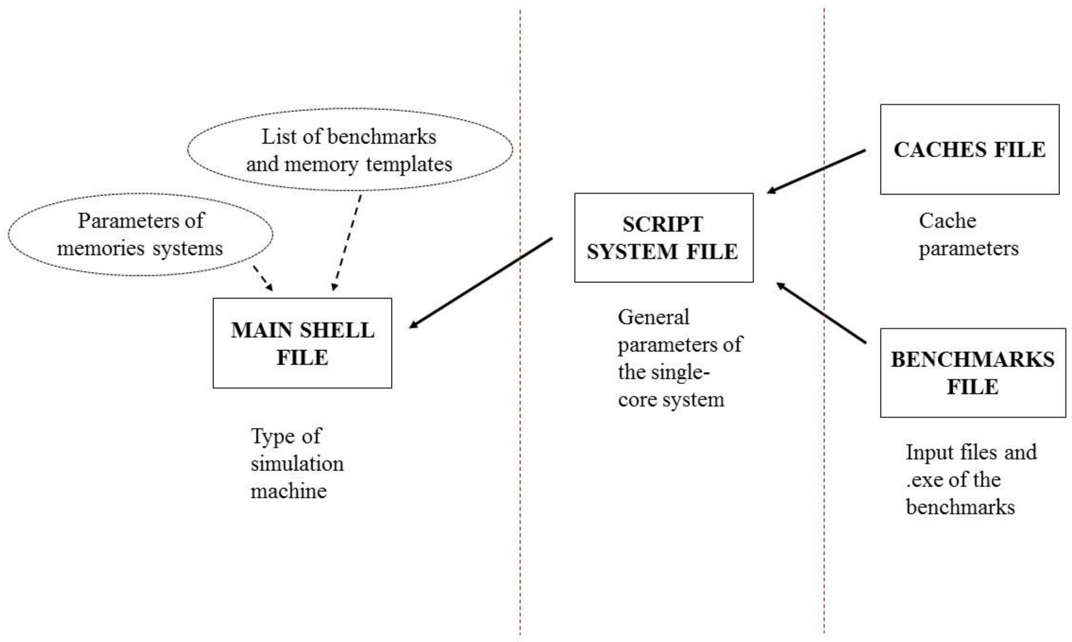

List of benchmarks and memory templates
Parameters of memories systems
MAIN SHELL FILE
Type of simulation machine
SCRIPT SYSTEM FILE
General parameters of the single-core system
CACHES FILE
Cache parameters
BENCHMARKS FILE
Input files and .exe of the benchmarks

Figure 7: Interaction between python files and shell of the single-core system framework

As shown in the figure 7, the framework handles 4 main files which we detail below:

Main Shell File. A Shell file contains the necessary instructions for the execution of the system that must be entered in the interface of Gem5 (Ubuntu). These instructions detail the directories of the files to be executed, the type of machine or cpu, the parameters that characterize each one of the memory systems that we want to simulate among other things. For that reason, executing all these instructions manually is a long and repetitive process, so we make shell files that contain all these instructions in an orderly and easy way to execute with just call the file in the Gem5 interface.

The main shell in turn uses more shells files that contain the aforementioned information in an orderly and easy-to-configure way (memory systems, list of benchmarks) which we will detail below:

List of benchmarks and memory templates. This Shell file only contains the names of the benchmarks to simulate (Table 11) next to the name of the memory system the size of the L2-Cache and the technological node to simulate (Table 12 shows a group of templates), this file orders by means of arrangements for the list of benchmarks for each of the templates of the memory system that we wish to simulate being this the main way to facilitate the simulations, in addition to granting the name of the folder to store the output files at the end of each simulation.

Table 11: List of benchmarks files

Benchmark bench_list
sha
patricia
FFT
susan_c
dijkstra
blowfish
basicmath_large
qsort_large
bitcount
stringsearch
CRC32

Table 12: List of memory system templates for 1MB at 28 nm

System_memory system_list
SRAM_SRAM_1MB_28nm
SRAM_SB_1MB_28nm
SRAM_DB_1MB_28nm
SB_SB_1MB_28nm
DB_DB_1MB_28nm

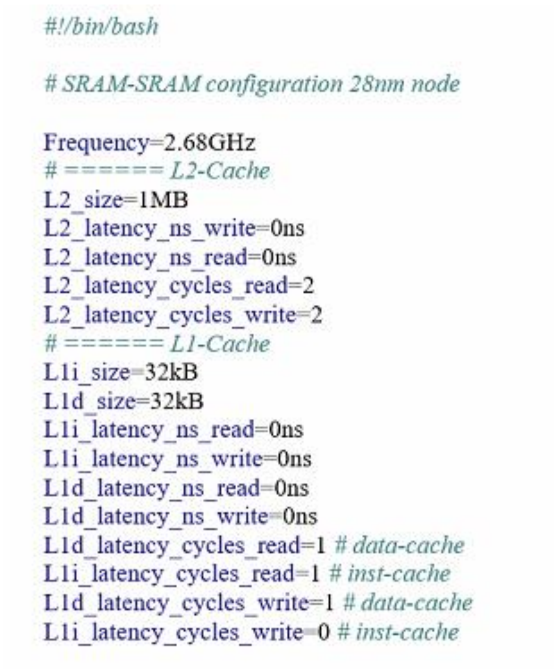
In the Annexes 3 we can visualize the complete Shell file.

Parameters of memory systems. These shells files are the templates of the memory systems, a group of these were shown in the table 12. In total there are 75 templates (5 memory systems, 5 sizes of L2-Cache, 3 technological nodes) each of these templates contains the main features of the memory system (Table 13)

Table 13: Parameters of memory system templates

Memory system parameters
Frequency
L2_size
L2_latency_ns_write
L2_latency_ns_read
L2_latency_cycle_write
L2_latency_cycle_read
L1i_size
L1d_size
L1i_latency_ns_read
L1i_latency_ns_write
L1d_latency_ns_read
L1d_latency_ns_write
L1d_latency_cycles_read
L1i_latency_cycles_read
L1d_latency_cycles_write
L1i_latency_cycles_write

They mainly handle the parameters of frequency, size, latency of writing and latency of reading for both levels of caches, they show the two types of latency so they can be used in both frameworks for single-core system, all these parameters are declared in each one of the 75 templates (figure 8), its configuration values can be found in the part of Annexes 2.



```
#!/bin/bash

# SRAM-SRAM configuration 28nm node

Frequency=2.68GHz
# ====== L2-Cache
L2_size=1MB
L2_latency_ns_write=0ns
L2_latency_ns_read=0ns
L2_latency_cycles_read=2
L2_latency_cycles_write=2
# ====== L1-Cache
L1i_size=32kB
L1d_size=32kB
L1i_latency_ns_read=0ns
L1i_latency_ns_write=0ns
L1d_latency_ns_read=0ns
L1d_latency_ns_write=0ns
L1d_latency_cycles_read=1 # data-cache
L1i_latency_cycles_read=1 # inst-cache
L1d_latency_cycles_write=1 # data-cache
L1i_latency_cycles_write=0 # inst-cache
```

Figure 8: Python code for SRAM-SRAM 1MB 28nm system template

The remaining 74 templates follow the same programming logic, the main shell file calls the respective template when starting the simulation, this data together with the name of the benchmark, the system script file, make up the complete instruction that the Gem5 platform needs to start with the simulation. The main shell file can be found in the annexes 4.

System script file. A script file is nothing more than a configuration file formatted in Python. The script makes use of 2 external files (cache.py file and MiBench_benchmarks.py file) that have the configuration of the caches and the configuration of the benchmarks files.
The script file mainly handles the declaration of the simulation blocks that the framework is composed (figure 6b), in addition to handling the parameters declared in the previous shell file, which configure each of the simulation blocks and the system in general (nominal voltage, frequency, latency), in the tables 14 and 15 we can see the parameters inside the script file.

Table 14: System script file parameters

System script file parameters
system_clock = Frequency
cpu_voltage = Vdd
cpu_clock = Frequency
system_memory_ranges
system_cache_line_size   # Cache line size in bytes
L1-Cache:
l1d_read_latency = options.L1d_latency_ns_read   #read latency
l1d_write_latency = options.L1d_latency_ns_write   #write latency
l1i_read_latency = options.L1i_latency_ns_read   #read latency
l1i_write_latency = options.L1i_latency_ns_write   #write latency
L2-Cache:
STT_L2_write_latency = options.L2_latency_ns_write #write latency
STT_L2_read_latency = options.L2_latency_ns_read #read latency
Memory:
main_memory = DDR3_1600_8x8()

Table 15: System script file parameters values

Overall System		
Parameter	**Units**	**Value**
System Voltage	V	0,8
CPU Voltage	V	0,8
Main Memory Size	MB	8192
Cache Line Size	-	64
ISA	-	X86

In the section of annexes 5 we have the complete script file.

File Caches. This file contains the main configuration of the Cache blocks of our system (parameters, connection functions), it is a function declared to objects, and the main parameters that the caches.py file handles are shown in the table 16 and 17

Table 16: Caches file parameters

Caches file parameters	
assoc	#Associativity
tag_latency	#Tag lookup latency
data_latency	#Data access latency
write_latency	#Write access latency
response_latency	#Latency for the return path on a miss
mshrs	#Number of MSHRs (max outstanding requests)
tgts_per_mshr	#Max number of accesses per MSHR
size	#Capacity

Table 17: Caches file parameters values

Cache Levels				
Parameter	**Units**	**L1-Inst**	**L1-Data**	**L2**
Associativity	-	2		8
tag_latency	Cycles	2		20
data_latency	Cycles	0	0	0
response_latency	Cycles	2		20
mshrs	-	4		20
tag_per_mshr	Cycles	20		12
Size	-	32 kB	32 kB	1 MB
writeback_clean	-	-	True	-

One of the main problems of the original cache.py file in Gem5 is that they do not have the write_latency parameter because the original system handles the same read and write value, this particularity makes it impossible to use the latency_cycle of the memory systems. Therefore, it was necessary to make a modification in the source files of Gem5, in the part of Annexes 6 we show step by step all this procedure.

Once our cache block has been adapted to the parameters, we need to rebuild the Gem5 simulation machine. After this, in the part of annexes 7, we show the file caches.py.

Benchmarks file. This Python file has the name of the executable files of each benchmark along with the input files and configuration parameters, so that these execution processes are activated when the name of the benchmark is configured by the main shell file. All executables and input files must be in the same folder within the Gem5 directory, the code corresponding to the MiBench_benchmarks.py file can be found in annexes 8.

Multi-core system framework

As an additional part of this work we configure one of the systems models that Gem5 has, se.py (system_call - emulation) and fs.py (full-system), in this investigation we chose the se.py model which is a system that avoids the need to configure operating systems or input and output devices for its operation, for this part of the simulation we have found the need to use other types of benchmarks (Table 18) designed for this type of operations, taking into account the weight of the benchmarks another reason why we select the SE simulation model is because of the speed it provides, especially when the configuration of the cpu is of the Timing type (cpu model recommended for real-time operations)

Table 18: Benchmarks of the Rodinia group for multi-core simulations

Benchmark bench_list
heartwall
lavaMD
nw
particlefilter
streamcluster

Next, we show the general framework that manages the multicore system (Figure 9)

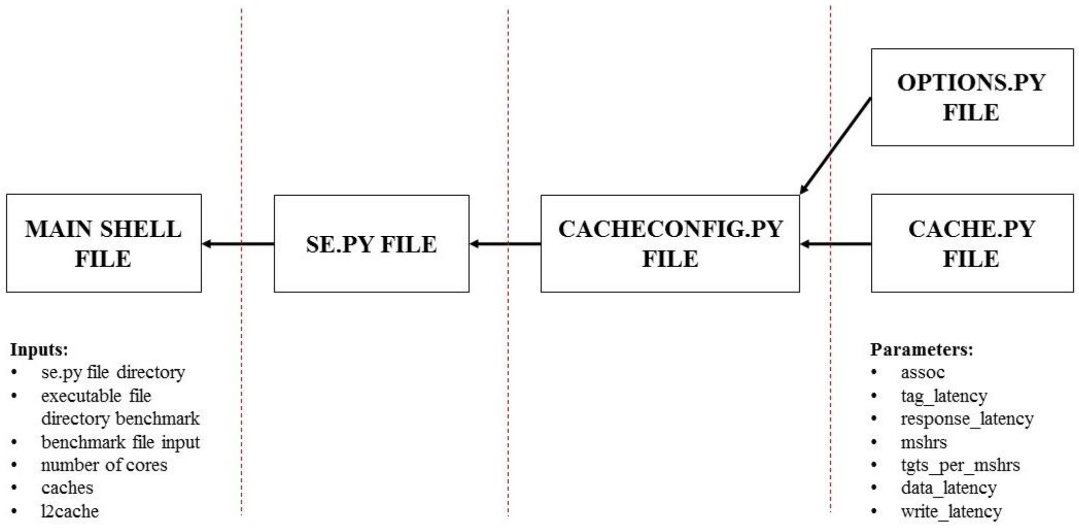
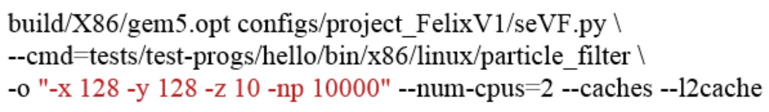


Figure 9: Interaction between python and shell files of the multi-core system framework

Each of these files has the same functions indicated in the single-core simulation framework.

Main Shell file. This file saves the syntax of the instructions that must be entered in the Gem5 interface. Before starting the simulation, it is necessary to have the directory of the se.py file and the benchmarks files. In the figure 10 we show an example of simulation, the instruction has the following order: Dir_se.py --cmd = Dir_benchmark -o "input_benchmark" --num-cpus = cores_number --caches --l2cache



```
build/X86/gem5.opt configs/project_FelixV1/seVF.py \
--cmd=tests/test-progs/hello/bin/x86/linux/particle_filter \
-o "-x 128 -y 128 -z 10 -np 10000" --num-cpus=2 --caches --l2cache
```

Figure 10: Instruction for simulation benchmark particle_filter with 2 cores

The --caches --l2cache instructions enable the cache simulation blocks respectively. Python se.py and cacheconfig.py files are files of the Gem5 platform, so their modification was made at specific points, in annexes 9 and 10 we can see the complete code of these files.

Cache.py file. This file has the main functions of the system caches, as well as its main configuration parameters. This work shows the simulation of a SRAM-SRAM and SRAM-DB type template at 28 nm, in annexes 13 we have the code of the Cache.py file that has the latency values of 28nm technology.

Options.py file. The options.py file has the multi-core system parameters for each of the different simulation modules (SE and FS), in this file you must modify the voltage, memory size, frequency (Tables 15 and 17) parameters of our system to simulate.

Stats files. As a result, at the end of the simulation Gem5 provides us with information such as simulated instructions, simulation time, written data, read data, through a text file called stats, the stats file contains all this information from each of the simulation blocks that make up the framework, for both simulation frameworks different directories were handled where stats files are stored, for the single-core system the output directory is declared in the code of the main shell file, while in the multi-core The stats are stored in the default output directory.

CHAPTER 3: ANALYSIS AND RESULTS.

Single-core system.

Once the methodology with which the simulation framework was developed and has been established, the simulation each of the 11 benchmarks was carried out by the 75 templates of the memory systems. At the end of each stats files, only the pertinent parameters were extracted, which we will pass to process them and finally analyze the behavior of performance and energy of each system with respect to the SRAM-SRAM reference system. The parameters extracted from each simulation are detailed below in the following table.

Table 19: Output parameters of stats files

Framework blocks	Name of variables in stats file	Comment
System	host_seconds sim_ticks	Real time elapsed on the host Number of ticks simulated
Reads$_{L2\_Cache}$	system.l2cache.ReadExReq_hits::total system.l2cache.ReadExReq_misses::total system.l2cache.ReadExReq_accesses::total	number of ReadExReq hits number of ReadExReq misses number of ReadExReq accesses (hits+misses)
Writes$_{L2\_Cache}$	system.l2cache.writebacks::total	number of writebacks
L1Cache+L2Cache	sim_insts	Number of instructions simulated
Reads$_{L1\_Cache-Instructions}$	system.cpu.icache.ReadReq_hits::total system.cpu.icache.ReadReq_misses::total system.cpu.icache.ReadReq_accesses::total	number of ReadReq hits number of ReadExReq misses number of ReadReq accesses (hits+misses)
Reads$_{L1\_Cache-Data}$	system.cpu.dcache.ReadReq_hits::total system.cpu.dcache.ReadReq_misses::total system.cpu.dcache.ReadReq_accesses::total	number of ReadReq hits number of ReadExReq misses number of ReadReq accesses (hits+misses)
Writes$_{L1Cache-Data}$	system.cpu.dcache.WriteReq_accesses::total	# number of WriteReq accesses (hits+misses)

The data shown in the table 19 were located in 5 templates referring to the sizes of L2-Cache that we want to simulate (128kB, 256kB, 512kB, 1MB, 2MB), the table shows a fragment of how these templates are composed.

Table 20: Fragment of calculation template for single-core simulations (FFT benchmarks and susan_c)

Tag. Used	**Memory System**			**Benchmark**
	L1 cache		**L2 cache**	
	inst. cache	**data cache**		
1	SRAM	SRAM	SRAM	FFT
2	SRAM	SRAM	STT-SB	FFT
3	SRAM	SRAM	STT-DB	FFT
4	STT-SB	STT-SB	STT-SB	FFT
5	STT-DB	STT-DB	STT-DB	FFT
1	SRAM	SRAM	SRAM	susan_c
2	SRAM	SRAM	STT-SB	susan_c
3	SRAM	SRAM	STT-DB	susan_c
4	STT-SB	STT-SB	STT-SB	susan_c
5	STT-DB	STT-DB	STT-DB	susan_c

The fragment shown in the table 20 must be completed by each of the 11 benchmarks of the table. This template will be a reference for one technological node (28nm) so then it will be repeated for the remaining the technological nodes (24nm, 20nm). All these sets of templates will form a general template referring to a single size of L2-Cache. Finally, this whole process is repeated for the other sizes of L2-Cache (128kB, 256kB, 512kB, 1MB, 2MB).

For each L2-Cache template, the data from the stats files were recorded together with the energy data and leakage power data obtained in the architecture simulation. All these data sets were processed as follows:

$$Sim\_time(s) = sim\_ticks * 10^{-12} \quad [3.1]$$
$$Sim\_time(ms) = Sim\_time(s) * 1000 \quad [3.2]$$
$$Sim.Time\ +\ write\ estim\ (s) = Sim_{time(s)} + Writes_{L2_{Cache}} * Latency_{write_{L2_{Cache}}} * 10^{-9} \quad [3.3]$$
$$Static\_Energy_{L2Cache}(mJ) = Sim\_time(s) * Leakage_{L2Cache}(mW) \quad [3.4]$$
$$Energy\_Reading_{L2Cache}(pJ) = ReadExReq\_accesses_{L2Cache} * Energy\_Read_{L2Cache}(pJ) \quad [3.5]$$
$$Energy\_Writing_{L2Cache}(pJ) = Writes_{L2Cache} * Energy\_Write_{L2Cache}(pJ) \quad [3.6]$$
$$Energy\_Reading_{L2Cache}(uJ) = Energy\_Reading_{L2Cache}(pJ) * 10^{-6} \quad [3.7]$$
$$Energy\_Writing_{L2Cache}(uJ) = Energy\_Writing_{L2Cache}(pJ) * 10^{-6} \quad [3.8]$$
$$Energy\_Total_{L2Cache}(uJ) = Energy\_Reading_{L2Cache}(uJ) + Energy\_Writing_{L2Cache}(uJ) \quad [3.9]$$
$$Static\_Energy_{L1Cache-Inst.}(mJ) = Sim\_time(s) * Leakage_{L1Cache}(mW) \quad [3.10]$$
$$Energy\_Reading_{L1Cache-Inst}(pJ) = ReadExReq\_accesses_{L1Cache-Inst} * Energy\_Read_{L1Cache}(pJ) \quad [3.11]$$
$$Energy\_Reading_{L1Cache-Inst}(uJ) = Energy\_Reading_{L1Cache-Inst}(pJ) * 10^{-6} \quad [3.12]$$
$$Energy\_Total_{L1Cache-Inst}(uJ) = Energy\_Reading_{L1Cache-Inst}(uJ) \quad [3.13]$$
$$Static\_Energy_{L1Cache-Data.}(mJ) = Sim\_time(s) * Leakage_{L1Cache}(mW) \quad [3.14]$$
$$Energy\_Reading_{L1Cache-Data}(pJ) = ReadExReq\_accesses_{L1Cache-Data} * Energy\_Read_{L1Cache}(pJ) \quad [3.15]$$
$$Energy\_Writing_{L1Cache-Data}(pJ) = Write_{L1Cache-Data} * Energy\_Write_{L1Cache}(pJ) \quad [3.16]$$

$$Energy\_Reading_{L1Cache-Data}(uJ) = Energy\_Reading_{L1Cache-Data}(pJ) * 10^{-6} \quad [3.17]$$
$$Energy\_Writing_{L1Cache-Data}(uJ) = Energy\_Writng_{L1Cache-Data}(pJ) * 10^{-6} \quad [3.18]$$
$$Energy\_Total_{L1Cache-Data}(uJ) = Energy\_Reading_{L1Cache-Data}(uJ) + Energy\_Writing_{L1Cache-Data}(uJ) \quad [3.19]$$

The above formulas were used for each of the data obtained. In this way the 5 templates of processed data were constructed to obtain the values of simulation time, read energy, and write energy and static energy for each cache block and for each performed simulation.

Next, we will determine the total energy of all Caches and the instructions per second of each simulation:

$$Energy\_Total\_Read_{AllCaches}(uJ) = Energy\_Reading_{L2Cache}(uJ) + Energy\_Reading_{L1Cache-Inst}(uJ) + Energy\_Reading_{L1Cache-Data}(uJ) \quad [3.20]$$

$$Energy\_Total\_Write_{AllCaches}(uJ) = Energy\_Writing_{L2Cache}(uJ) + Energy\_Writing_{L1Cache-Data}(uJ) \quad [3.21]$$

$$Leakage\_Energy\_Total_{AllCaches}(mJ) = Static\_Energy_{L2Cache}(mJ) + Static\_Energy_{L1Cache-Inst.}(mJ) + Static\_Energy_{L1Cache-Data.}(mJ) \quad [3.22]$$

$$Leakage\_Energy\_Total_{AllCaches}(uJ) = Leakage\_Energy\_Total_{AllCaches}(mJ) * 10^{-3} \quad [3.23]$$

$$Energy\_Total\_Abs_{AllCaches}(uJ) = Energy\_Total\_Read_{AllCaches}(uJ) + Energy\_Total\_Write_{AllCaches}(uJ) + Leakage\_Energy\_Total_{AllCaches}(uJ) \quad [3.24]$$

$$Ints_{sec}\ (\#/s) = \frac{sim\_inst}{Sim\_time(s)} \quad [3.25]$$

$$Mega - Inst\_per\_\mathrm{sec}(MIPS) = Ints_{sec}\ (\#/s) * 10^{-6} \quad [3.26]$$

Calculated the values of Energy and instructions for each of the simulations, we determine the arithmetic mean in relation to the systems of simulated memories. That is to say to add all the values of energy and instructions concerning the simulations (for example, SRAM-SRAM) of the 11 benchmarks and obtain their arithmetic mean. In the following we show the formulas referring to a memory system (SRAM-SRAM), it being understood that this whole process was performed for each of the other memory systems.

$$Energy\_Total\_mean_{SRAM-SRAM} \frac{\sum_{i=1}^{n} Energy\_Total\_Abs_{AllCaches\_SRAM-SRAM}(uJ)_n}{n} \quad [3.27]$$

$$Ints_{sec}\ (\#/s)\_mean_{SRAM-SRAM} \frac{\sum_{i=1}^{n} Ints_{sec}\ (\#/s)_{SRAM-SRAM_n}}{n} \quad [3.28]$$

$$Sim\_inst\_mean_{SRAM-SRAM} \frac{\sum_{i=1}^{n} Sim\_inst_{SRAM-SRAM_n}}{n} \qquad [3.29]$$

Furthermore, the calculations to determine the energy per instruction (EPI) and the instructions per second (IPS) are shown below, the formulas are shown only for the SRAM-SRAM system:

$$EPI\_abs_{SRAM-SRAM} = \frac{Energy\_Total\_mean_{SRAM-SRAM}}{Sim\_inst\_mean_{SRAM-SRAM}} \qquad [3.30]$$

$$IPS\_abs_{SRAM-SRAM} = Ints_{sec}\ (\#/s)\_mean_{SRAM-SRAM} \qquad [3.31]$$

Finally, we normalize the EPI and IPS values with reference to the SRAM-SRAM system. That is, the 5 values of each memory system will be divided for the SRAM-SRAM system values. The EPI and IPS values are shown in the next tables.

Table 21: EPI for L2-Cache 128kB

EPI L2_Cache 128kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,7379	0,7288	0,8545	0,5973
24	1	0,8803	0,8760	1,1200	0,7844
28	1	0,8979	0,8950	1,2061	0,8183

Table 22: IPS for L2-Cache 128kB

IPS L2_Cache 128kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,9989	0,9996	0,5166	0,3506
24	1	0,9988	0,9997	0,6802	0,4176
28	1	0,9988	0,9875	0,9187	0,5144

Table 23: EPI for L2-Cache 256kB

EPI L2_Cache 256kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,5965	0,5825	0,7129	0,5208
24	1	0,7895	0,7821	1,0076	0,7170
28	1	0,8171	0,8104	1,0945	0,7518

Table 24: IPS for L2-Cache 256kB

IPS L2_Cache 256kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,9989	0,9996	0,5157	0,3495
24	1	0,9989	0,9997	0,6792	0,4167
28	1	0,9989	0,9996	0,9186	0,5165

Table 25: EPI for L2-Cache 512kB

EPI L2_Cache 512kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,4541	0,4279	0,5629	0,4396
24	1	0,6603	0,6485	0,8474	0,6209
28	1	0,6973	0,6864	0,9288	0,6532

Table 26: IPS for L2-Cache 512kB

IPS L2_Cache 512kB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,9778	0,9996	0,5153	0,3492
24	1	0,9990	0,9995	0,6790	0,4164
28	1	0,9989	0,9996	0,9186	0,5162

Table 27: EPI for L2-Cache 1MB

EPI L2_Cache 1MB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,3211	0,2977	0,4366	0,3713
24	1	0,5100	0,4931	0,6610	0,5090
28	1	0,5501	0,5340	0,7252	0,5319

Table 28: IPS for L2-Cache 1MB

IPS L2_Cache 1MB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,9990	0,9997	0,5151	0,3490
24	1	0,9991	0,9996	0,6786	0,4162
28	1	0,9990	0,9995	0,9184	0,5160

Table 29: EPI for L2-Cache 2MB

EPI L2_Cache 2MB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,2340	0,2076	0,3494	0,3243
24	1	0,3708	0,3491	0,4883	0,4047
28	1	0,4056	0,3845	0,5253	0,4814

Table 30: IPS for L2-Cache 2MB

IPS L2_Cache 2MB					
Nodes	**SRAM-SRAM**	**SRAM-SB**	**SRAM-DB**	**SB-SB**	**DB-DB**
20	1	0,9990	0,9997	0,5148	0,3488
24	1	0,9991	0,9996	0,6784	0,4183
28	1	0,9991	0,9995	0,9185	0,4654

Taking into account the standardized EPI and IPS values shown in the previous tables, in the following we show the graphics of the energy and performance for each memory system. The value of 1 represents our SRAM-SRAM reference system as it was explained previously.

Energy per instruction (EPI)

This analysis refers to the energy consumption or low power that the 5 systems showed for the different technological nodes and the different memory sizes for L2-Cache

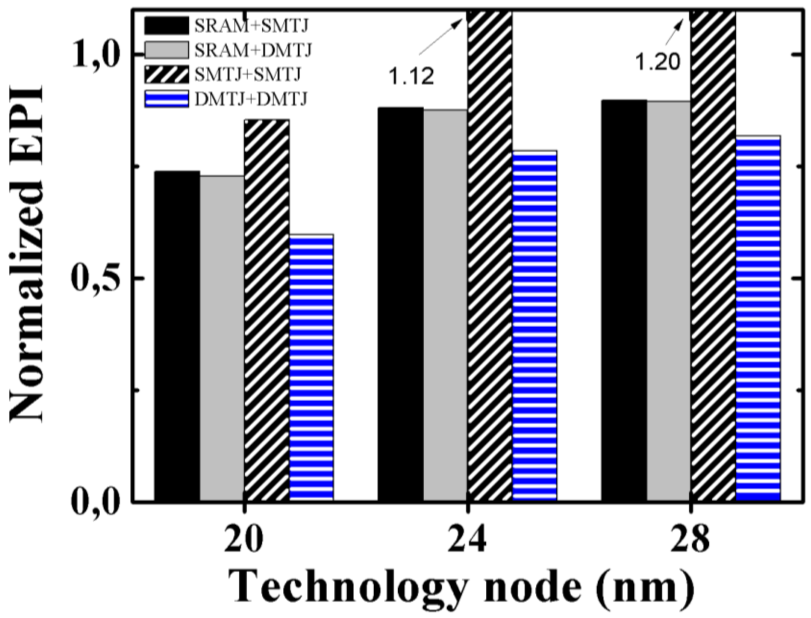


Figure 11: Energy per Instruction L2-Cache 128kB

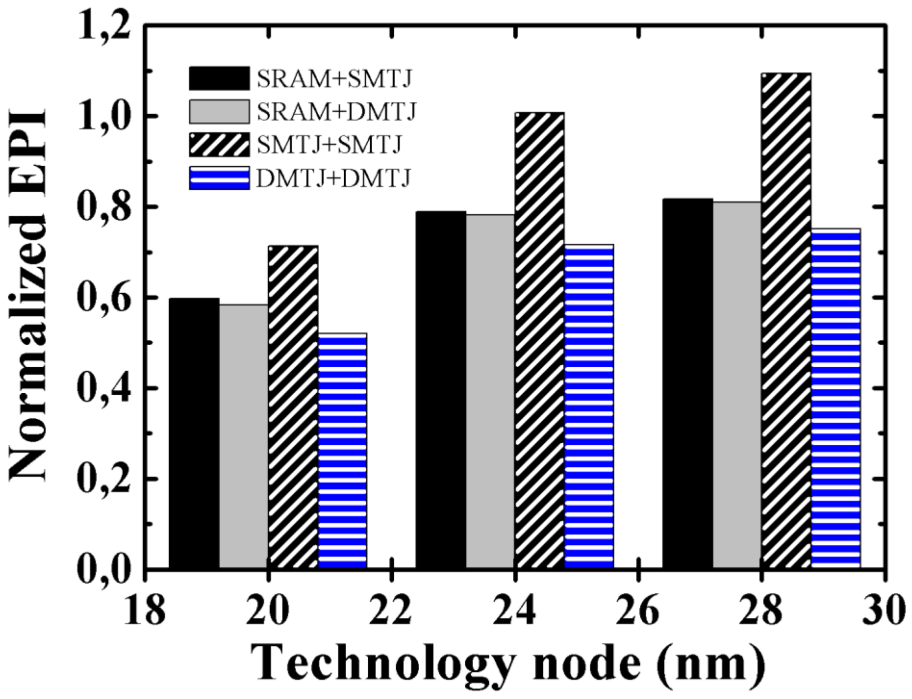


Figure 12: Energy per Instruction L2-Cache 256kB

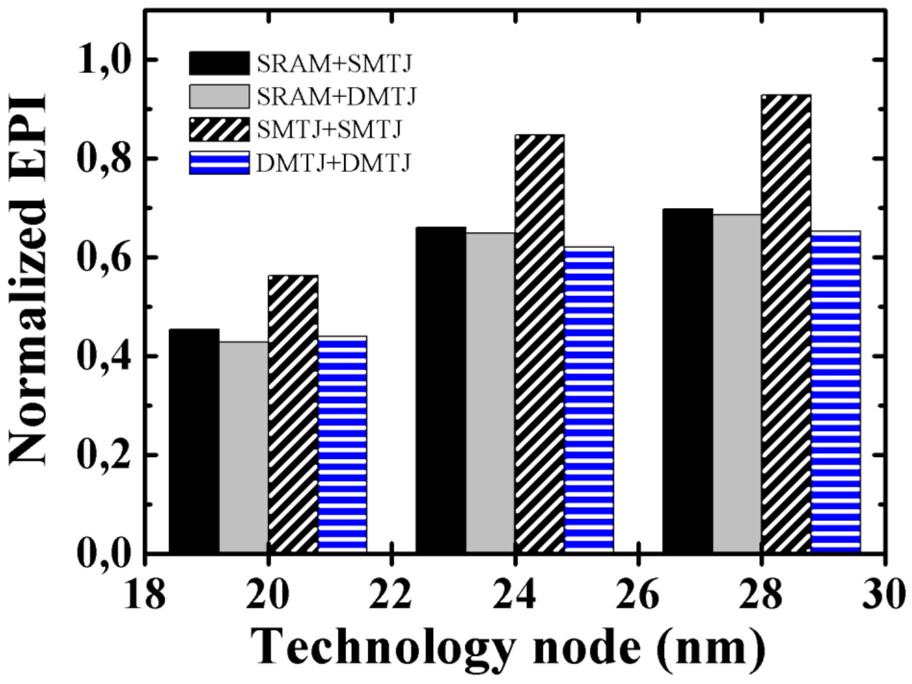


Figure 13: Energy per Instruction L2-Cache 512kB

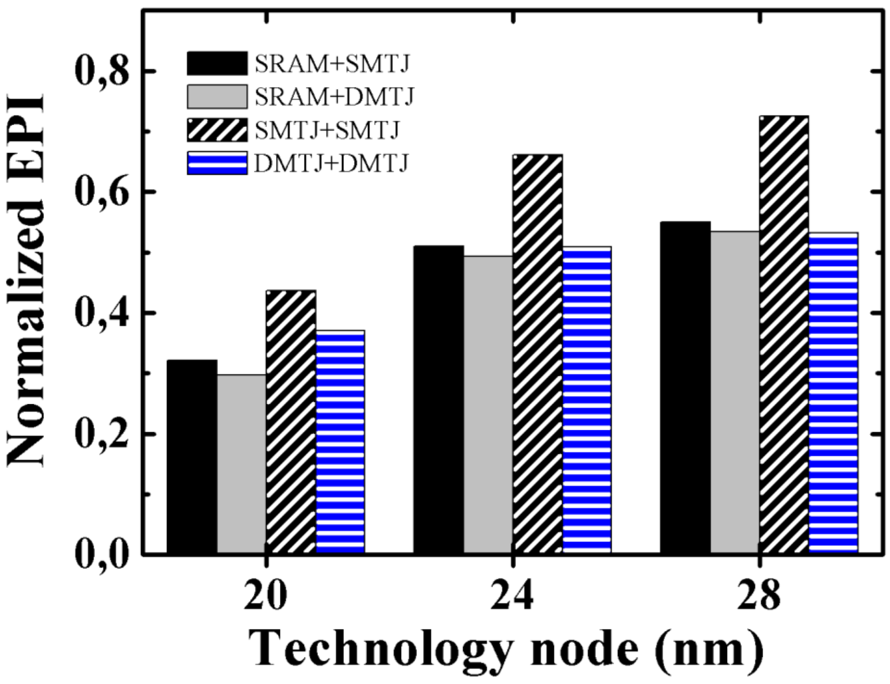


Figure 14: Energy per Instruction L2-Cache 1MB

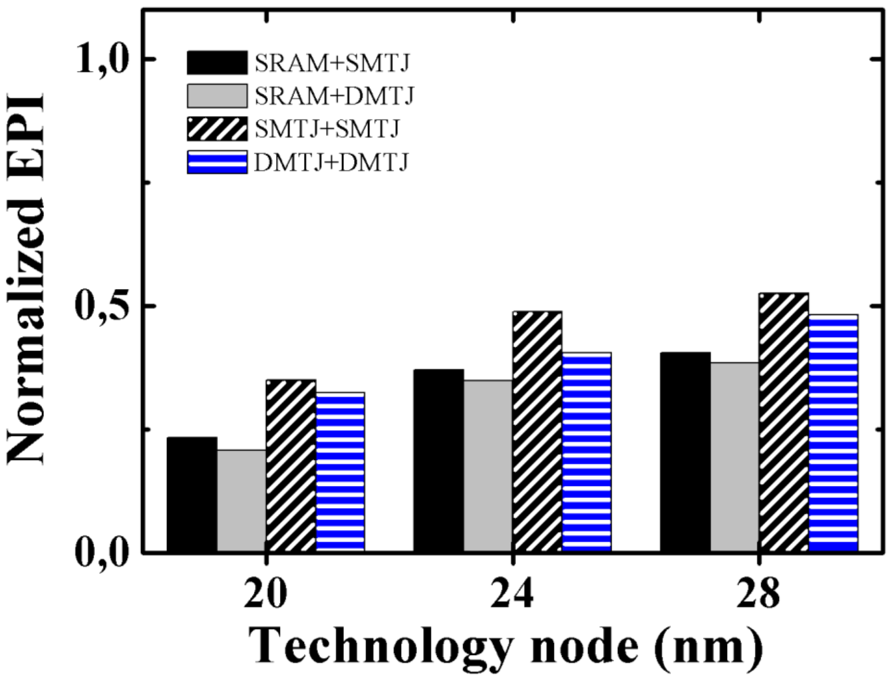


Figure 15: Energy per Instruction L2-Cache 2MB

The previous figures indicate the behavior of the memory systems at two levels with respect to the energy per instruction (EPI) and the technological nodes, in the five figures we can see higher EPI values in technological nodes of 28nm and it is reduced as the node scales. Observing the energy values in Annexes 1, the energy values are always higher for a 28nm node, so it is an expected behavior in the 5 cases. However, the most interesting point is to analyze the behaviors of systems 3 (SRAM-DMTJ) and systems 5 (DMTJ-DMTJ) remembering that the read energy values for type memories MTJ double barrier are the lowest in this group of memories (Annexes 1).

For sizes smaller of L2-Cache (128kB and 256kB) we observe a lower consumption of EPI for system 5 (Figures 11 and 12) because the number of accesses for the L2-Cache is smaller due to the memory size is lower. For this reason, the contribution of energy for the L2-Cache is lower compared to other cases that is why the system 5 is the best option in an energy-efficient point of view. On the other hand, we observe that system 4 (SMTJ-SMTJ) has the worst energy performance in relation to energy consumption (Figure 11 and 12). It is an expected behavior if we take into account that MTJ memories with a single barrier have a higher energy consumption compared to SRAM and DMTJ memories. Moreover, with respect to systems 2 and 3, their behavior is a little lower than system 1 (reference), due to the presence of accesses in the L2-Cache, so these systems are acceptable as future replacement options for SRAM memories under these circumstances.

For the other sizes (512kB, 1MB and 2MB) we see a change in the behavior of system 5, this begins to have a similar behavior to systems 2 and 3. However for larger sizes of L2-Cache such as 1MB and 2MB, this leaves to be the best option compared to systems 2 and 3. This behavior is justifiable since these sizes of Cache have a greater number of accesses at level 2, therefore the number of writes increases causing greater energy consumption by DMTJ memories that have a higher consumption of writing energy. For this reason, for larger Cache sizes, system 3 has greater benefits at low power, keeping the cache with the same number of accesses (L1-Cache) with SRAM type memories, but changing to DMTJ type memories for L2-Cache. In the 5 cases we can see that system 4 is the worst of all.

Instruction per second (IPS)

This analysis refers to the system performance below. We show the graphs that indicate the behavior of the IPS for the technology nodes and the L2-Cache memory size.

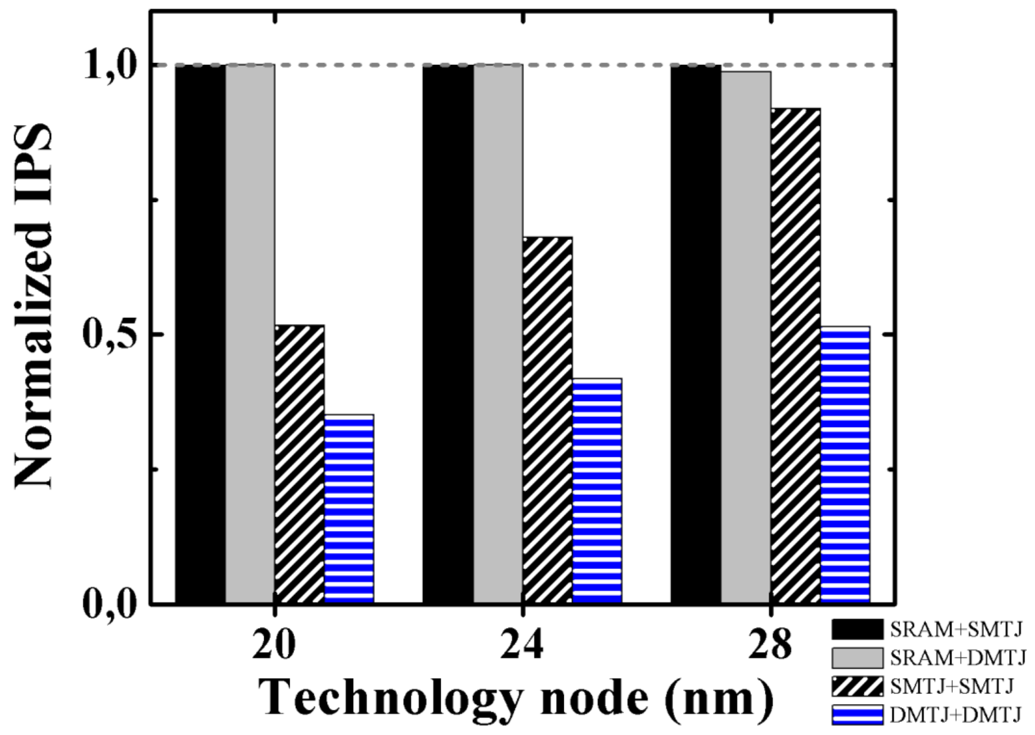


Figure 16: Instruction per second L2-Cache 128kB

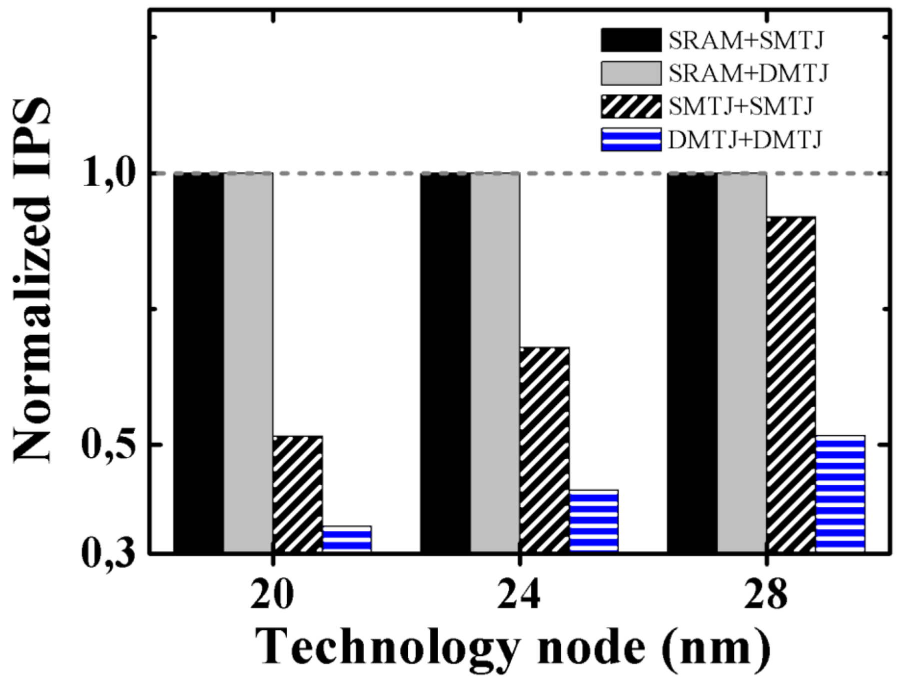


Figure 17: Instruction per second L2-Cache 256kB

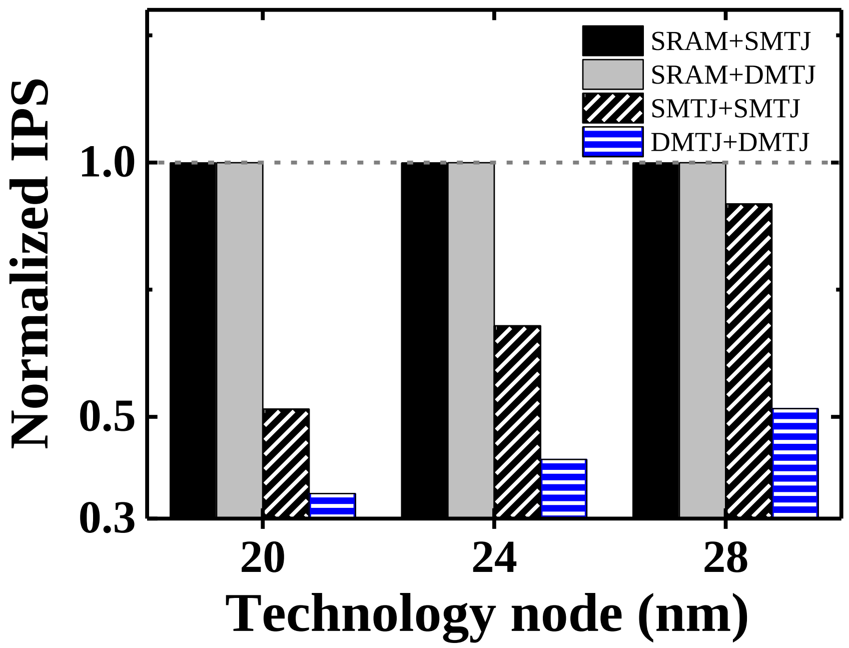


Figure 18: Instruction per second L2-Cache 512kB

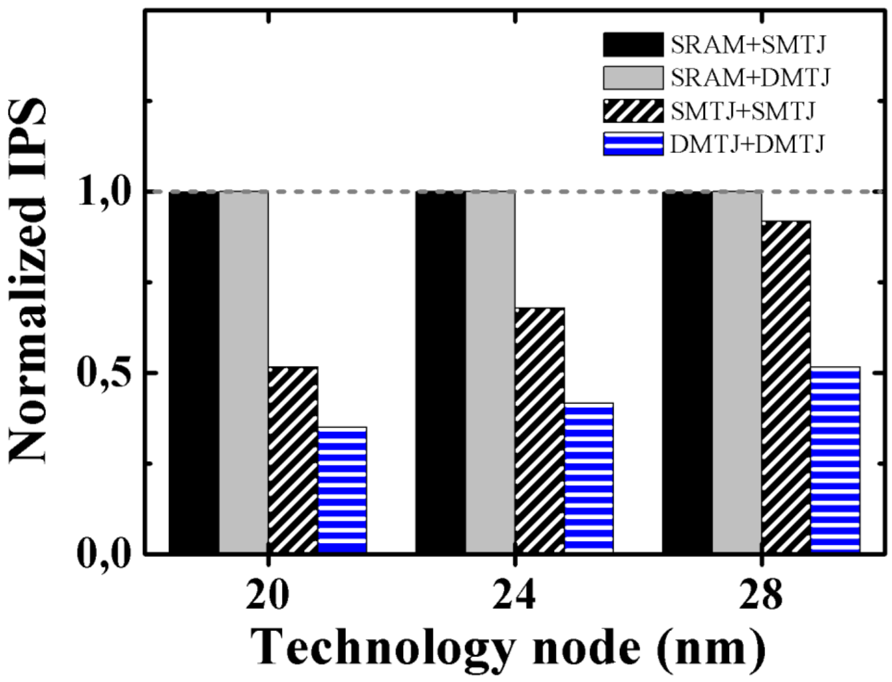


Figure 19: Instruction per second L2-Cache 1MB

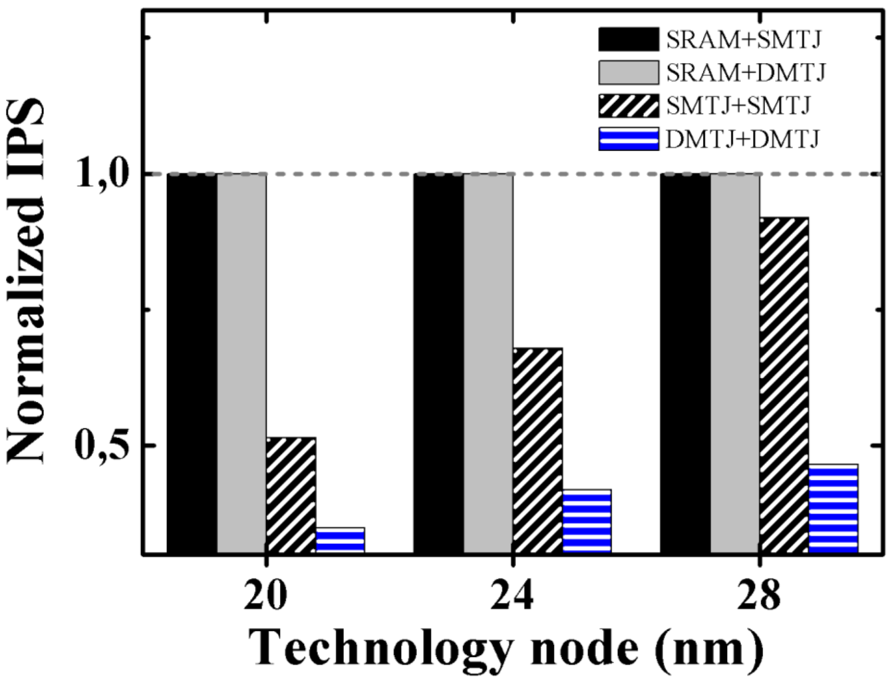


Figure 20: Instruction per second L2-Cache 2MB

Figures 16, 17, 18, 19 and 20 show us the performance of the 5 memory systems. In the systems 2 and 3 we see a greater number of instructions per second for the technological node at 28nm. Moreover, we can mention, in general for all the systems, as the reference the memory capacity of L2 -Cache taking into account that the lower the memory capacity, the lower the latency for the reading and writing accesses (See annexes 1). However, the interesting thing about the analysis can be observed in systems 2 and 3 respectively, taking as an emphasis that the latency values favor more to SRAM type memories. We can see the same behavior for these systems in all the technological nodes and in all the memory capacities of the L2-Cache. In this point SRAM-DMTJ hybrid systems is a good option, especially for the higher L2-Cache sizes where the number of accesses at this level of Cache is high and this does not have a major impact on the final performance of the system because the L1-Cache maintains the same topology of our reference system using SRAM memories.

Based on the results obtained from the point of view of the power, using a system 5 (DMTJ-DMTJ) for low memory capacities, considerable energy savings are expected, however, based on performance this type of system affects the performance in (50% - 70%) as shown in figures

16 and 17. On the other hand, using a system 3 (SRAM-DMTJ) for low L2-Cache capacities, they have a considerable saving with respect to the reference system. Although this does not compare to the expected savings with a system 5 but the system 3 has the advantage of maintaining the same performance with respect to the reference system.

Multi-core system

Finally, this chapter analyzes the simulation referring to the multi-core simulation system, for the analysis of this system most of the above-mentioned processes (Single-core) are not considered, for this analysis we observe the behavior of the IPS and the energy in the cache levels in the system, due to the number of cores analyzing the instructions or accesses in each cache level is a complex process because the number of cores increases the number of L1-caches for this reason we concentrate on the total energy that each cache manages based on the number of total accesses in this level.

The IPS is analyzed for system 1 (SRAM-SRAM) based on the results shown in single core systems. The behavior that interests us is system 3, taking into account that the performance behavior is similar for both cases and that the simulated benchmark (Particle_filter) has a considerably high number of instructions, and we consider that the variation in performance between these systems is negligible. Next, the figure 21 shows the performance behavior based on the number of cores.

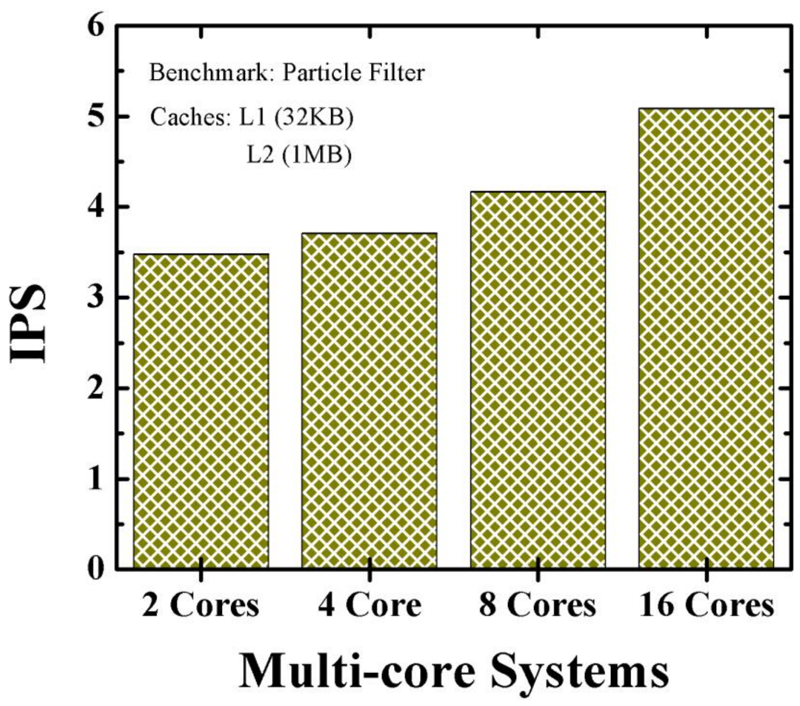


Figure 21: Instruction per second for Multi-core System SRAM-SRAM 28nm L2-Cache 1MB

As shown in the figure, the performance increases as the number of cores increases, a normal behavior which validates the operation of our simulation framework. The values referring to the figure can be seen in the following table:

Table 31: Instruction per second for multi-core system SRAM-SRAM 28nm L2-Cache 1MB

Cores	IPS
2	3,48085
4	3,70915
8	4,16672
16	5,09029

The reader may be wondering why the number of IPS has such small values. The reason is because in the environment simulated (“System emulator”), the simulation reaches to a saturation making an equal number of instructions for all the cases simulated (2- to 16-cores). Thus, the only parameter that changes is the simulation time. On the other hand, the energy consumed by each cache based on the number of cores is shown in the figure 22 which simultaneously shows the behavior of both caches.

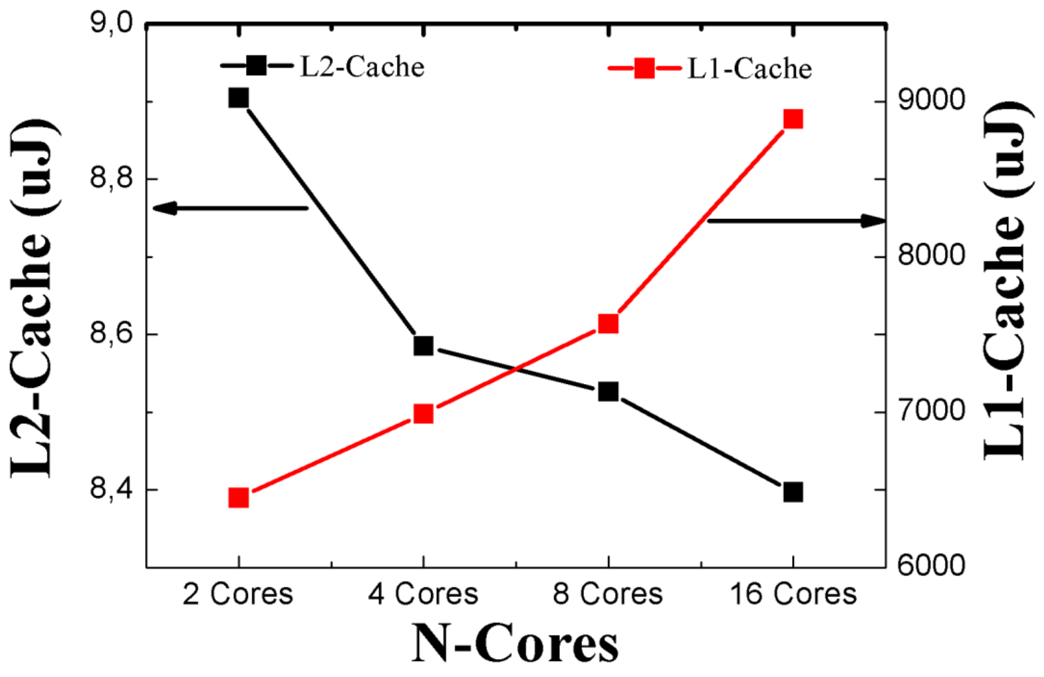


Figure 22: L1-Cache Energy vs L2-Cache Energy vs number of cores

The most interesting thing about this figure is the simultaneous behavior that the cache levels show as the number of cores increases. We can see that at a smaller number of cores the number of instructions that access the L1-Cache reaches a point that becomes saturated, so the vast majority of these are in the L2-Cache causing a higher energy consumption at this level. As the number of cores increases, the access capacity increases at the L1-Cache level, which causes the number of accesses to decrease in the L2-Cache, at the same time making the energy contribution by L2-Cache decrease to as the energy consumption in the L1-Cache increases. Next, we show the table of energy consumed in the levels of Caches:

Table 32: Energy per cache level for SRAM-SRAM 28nm L2-Cache multicore system 1MB

Cores	**SRAM+SRAM**			
	E_L2tot (uJ)	**E_L1_Itot (uJ)**	**E_L1_Dtot (uJ)**	**E_L1tot (uJ)**
2	8,9049	5482,1632	965,8025	6447,9657
4	8,5850	5825,2877	1163,7264	6989,0140
8	8,5262	6333,1142	1236,1885	7569,3027
16	8,3967	7505,0143	1382,5786	8887,5929

CHAPTER 4: CONCLUSIONS AND RECOMMENDATIONS

Conclusions

- Among all the systems simulated, the DMTJ-based (Double-barrier MTJ) system configuration presents optimal results in terms of energy, (Figures 11 - 15). For systems with lower capacities of L2-Cache and with only DMTJ-based caches (i.e. L1 and L2 with DMTJ structures) provides the best energy configuration (see table 23), while for a larger L2-Cache capacity the hybrid system (SRAM-DMTJ) exhibits the best energy case (see table 29) owing to the L2-Cache access activities.
- The hybrid SRAM-MTJ configurations obtained a similar SRAM-SRAM performance. This points out an attractive configuration, which allows us to work from the point of view of power reduction without affecting the actual state-of-the-art system performance. However, this can be done only by small computational applications in which the L2-cache is not exploited.
- Most of the instructions are processed in the L1-Cache, so the performance between an SRAM-SRAM system and a hybrid SRAM-MTJ system does not present significant changes. This allows to take the advantage of MTJ memories that despite having a lower performance at SRAMs, the consumption of write energy and leakage current is much lower than SRAMs. Thus, allowing an energy-efficient design without diminishing performance. Again, the latter advantage was proved in this work by considering low computational applications.
- Using a simulation model System-Call Emulation (SE) allows us to perform a simple simulation in real time, without the need for complex configurations in the operating system. However, the accuracy is compromised in the multi-core simulation framework,

reflected in the number of sim_ticks (ticks is the unit of measure for the Software, see table 19) that is the same in each case.

- The energy consumption between the L1-Cache and L2-Cache levels within the multi-core simulation maintains a relationship based on the number of cores. Lower number of cores means a higher energy consumption of the L2-Cache level while the L1 level - Cache contributes to the total energy consumption with a minimum percentage (saturation of the instructions in L1-Cache), by increasing the number of cores these relationships act in a contrary way, L1-Cache increases its energy consumption and L2-cache decreases its energy consumption. A similar total energy value is observed in different cores due to saturation in the simulated benchmark instructions. Thus, the final system energy and performance cannot be seen in a multi-core configuration when using the System-Call Emulation option.

Recommendations

- For simulations of type System-Call Emulation (SE) and CPU Timing_Simple we focus on the use of latency in cycles, taking into account the characteristics of the simulation model (Speed, Real Time) so we use a latency configuration in cycles using latency Lower as base cycle allows to maintain a relationship of latencies more easily. If we want to use latency in seconds (nano-seconds) it is recommended to configure the simulation framework for Full-System (FS) mode.
- An interesting point for future work with MTJ memories would be to analyze the performance behavior and energy behavior for cache memories with a greater storage capacity (greater than 2MB) as well as systems with a hierarchy greater than 2 levels, taking as a basis The results of this research can be certain that future work will better consolidate MTJ memories as strong candidates to replace SRAM technology.
- For multi-core type simulations, we recommend changing the simulation model, working on the Full System (FS) type simulation script, a script specialized in simulation accuracy.
- Using benchmarks with a smaller number of instructions than the one used in this work (particle_filter) will help to have a better understanding of the system's performance, in our simulations the system reached a saturation point due to the high density of benchmark instructions

Bibliography

Bishnoi, R., Ebrahimi, M., Oboril, F., & Tahoori, M. (s.f.). *Architectural Aspects in Design and Analysis of SOT-based Memories.* Baden-Wuerttemberg: National Research Center of the Helmholtz Association.

Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., . . . Wood, D. A. (2/05/2011). The gem5 Simulator. *http://gem5.org*.

Dong, X., Xu, C., Jouppi, N., & Xie, Y. (2012). NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Counc. Electron. Des. Autom.*(994–1007), 31.

Jason. (s.f.). *Learning gem5*. (github) Recuperado el 29 de 08 de 2019, de http://learning.gem5.org/book/part1/cache_config.html

Mittal, S., Wang, R., & Vetter, J. (11/09/2017). DESTINY: A Comprehensive Tool with 3D and. *Journal of Low Power Electronics and Applications MDPI*.

Syu, S., Shao, Y.-H., & Lin, I.-C. (2–3 May 2013.). High-endurance hybrid cache design in CMP architecture with cache partitioning and access-aware policy. *In Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI*.

Turmero, P. (s.f.). *Jerarquía de las memorias*. (Monografias.com) Recuperado el 29 de 08 de 2019, de https://www.monografias.com/trabajos104/jerarquia-memorias/jerarquia-memorias.shtml

Zhang, Y., Zhang, Z., Jiang, N., Zheng, Z., Wang, Y., Zeng, L., . . . Zhao, W. (2017). Compact Modeling of High Spin Transfer Torque Efficiency Double-Barrier Magnetic Tunnel Junction. *IEEE*.

G. Prenat et al., “Ultra-Fast and High-Reliability SOT-MRAM: From Cache Replacement to Normally-Off Computing,” IEEE Trans. Multi-Scale Comp. Syst., vol. 2, no. 1, pp. 49–60, Jan.-Mar. 2016.

M. Poremba et al., “DESTINY: A Tool for Modeling Emerging 3D NVM and eDRAM caches,” Proc. of 2015 DATE Conf. & Exh, Mar. 2015, pp. 1543–1546

E. Garzón et al., "Exploiting Double-Barrier MTJs for Energy-Efficient Nanoscaled STT-MRAMs," 2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Lausanne, Switzerland, 2019, pp. 85-88.

R. De Rose et al., “A Compact Model with Spin-Polarization Asymmetry for Nanoscaled Perpendicular MTJs,” IEEE Trans. Elec. Dev., vol. 64, no. 10, pp. 4346–4353, Oct. 2017

R. De Rose et al., “A Variation-Aware Timing Modeling Approach for Write Operation in Hybrid CMOS/STT-MTJ Circuits,” IEEE Trans. Circ. Syst. I: Reg. Pap., vol. 65, no. 3, pp. 1086–1095, Mar. 2018

K. C. Chun et al., "A Scaling Roadmap and Performance Evaluation of In-Plane and Perpendicular MTJ Based STT-MRAMs for High-Density Cache Memory," IEEE Journal of Solid-State Circuits, vol. 48, no. 2, pp. 598-610, Feb. 2013

G. Hu et al., “STT-MRAM with double magnetic tunnel junctions,” Proc. of 2015 IEDM, Dec. 2015, pp. 26.3.1–26.3.4

G. Wang et al., “Compact modeling of high spin transfer torque efficiency double barrier Magnetic tunnel junction,” Proc. of 2017 IEEE/ACM NANOARCH, Oct. 2017, pp. 49–54

R. De Rose et al., “Variability-Aware Analysis of Hybrid MTJ/CMOS Circuits by a Micromagnetic-Based Simulation Framework,” IEEE Trans. on Nanotech. vol. 16, no. 2, pp. 160–168, Mar. 2017

E. Garzón, et. al. Assessment of STT-MRAM Performance at Nanoscaled Technology Nodes Using a Device-To-Memory Simulation Framework, Journal of Microelectronics Engineering, Vol. 215, 2019

ANNEXES

Annexes 1) Templates for memories systems

Table: DESTINY simulation, 5 memory systems, 3 technology nodes, Latency (Read and Write), Energy (Read and Write), Leakage power for L1-Cache (32kB) and L2-Cache (128kB)

				DESTINY DATA									
				Latency				**Energy**				**Leakage**	
				32KB		128kB		32KB		128kB		32KB	128kB
Nodes	**System**			**L1-Cache**		**L2-Cache**		**L1-Cache**		**L2-Cache**		**L1-Cache**	**L2-Cache**
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**-**	**-**
nm	-	GHz	GHz	ns	ns	ns	ns	(pJ)	(pJ)	(pJ)	(pJ)	(mW)	(mW)
28	Sys_A	2,68	2,68	0,373	0,349	0,381	0,353	60,081	40,298	75,823	52,815	2,655	10,552
	Sys_B	2,68	2,68	0,373	0,349	0,776	3,798	60,081	40,298	93,552	192,662	2,655	1,308
	Sys_C	2,68	2,68	0,373	0,349	1,355	1,476	60,081	40,298	66,354	102,224	2,655	0,981
	Sys_D	2,68	2,68	0,767	3,790	0,776	3,798	77,517	179,280	93,552	192,662	0,329	1,308
	Sys_E	2,68	2,68	1,352	1,470	1,355	1,476	53,531	91,625	66,354	102,224	0,247	0,981
24	Sys_A	3,18	3,18	0,314	0,296	0,320	0,299	56,100	37,500	65,526	45,998	3,559	14,148
	Sys_B	3,18	3,18	0,314	0,296	0,803	3,394	56,100	37,500	78,267	153,181	3,559	1,708
	Sys_C	3,18	3,18	0,314	0,296	1,436	1,267	56,100	37,500	56,397	84,370	3,559	1,281
	Sys_D	3,18	3,18	0,796	3,392	0,803	3,394	69,800	144,640	78,267	153,181	0,430	1,708
	Sys_E	3,18	3,18	1,432	1,266	1,436	1,267	49,242	76,254	56,397	84,370	0,322	1,281
20	Sys_A	3,79	3,79	0,264	0,247	0,269	0,254	47,325	32,833	54,388	38,275	11,282	44,839
	Sys_B	3,79	3,79	0,264	0,247	0,830	3,062	47,325	32,833	65,675	119,595	11,282	5,415
	Sys_C	3,79	3,79	0,264	0,247	1,598	1,089	47,325	32,833	46,583	65,900	11,282	4,062
	Sys_D	3,79	3,79	0,822	3,043	0,830	3,062	59,278	114,597	65,675	119,595	1,363	5,415
	Sys_E	3,79	3,79	1,587	1,080	1,598	1,089	38,795	61,864	46,583	65,900	1,022	4,062

Table: DESTINY simulation, 5 memory systems, 3 technology nodes, Latency (Read and Write), Energy (Read and Write), Leakage power for L1-Cache (32kB) and L2-Cache (256kB)

				DESTINY DATA									
				Latency				**Energy**				**Leakage**	
				32KB		256kB		32KB		256kB		32KB	256kB
Nodes	**System**			**L1-Cache**		**L2-Cache**		**L1-Cache**		**L2-Cache**		**L1-Cache**	**L2-Cache**
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**-**	**-**
nm	-	GHz	GHz	ns	ns	ns	ns	(pJ)	(pJ)	(pJ)	(pJ)	(mW)	(mW)
28	Sys_A	2,68	2,68	0,373	0,349	0,387	0,356	60,081	40,298	87,892	64,884	2,655	21,105
	Sys_B	2,68	2,68	0,373	0,349	0,782	3,801	60,081	40,298	105,618	204,727	2,655	2,616
	Sys_C	2,68	2,68	0,373	0,349	1,359	1,478	60,081	40,298	75,542	111,411	2,655	1,962
	Sys_D	2,68	2,68	0,767	3,790	0,782	3,801	77,517	179,280	105,618	204,727	0,329	2,616
	Sys_E	2,68	2,68	1,352	1,470	1,359	1,478	53,531	91,625	75,542	111,411	0,247	1,962
24	Sys_A	3,18	3,18	0,314	0,296	0,324	0,301	56,100	37,500	75,873	56,345	3,559	28,296
	Sys_B	3,18	3,18	0,314	0,296	0,807	3,397	56,100	37,500	88,597	163,511	3,559	3,415
	Sys_C	3,18	3,18	0,314	0,296	1,438	1,268	56,100	37,500	64,262	92,236	3,559	2,562
	Sys_D	3,18	3,18	0,796	3,392	0,807	3,397	69,800	144,640	88,597	163,511	0,430	3,415
	Sys_E	3,18	3,18	1,432	1,266	1,438	1,268	49,242	76,254	64,262	92,236	0,322	2,562
20	Sys_A	3,79	3,79	0,264	0,247	0,272	0,256	47,325	32,833	63,010	46,897	11,282	89,679
	Sys_B	3,79	3,79	0,264	0,247	0,833	3,063	47,325	32,833	74,278	128,198	11,282	10,831
	Sys_C	3,79	3,79	0,264	0,247	1,600	1,090	47,325	32,833	53,136	72,452	11,282	8,123
	Sys_D	3,79	3,79	0,822	3,043	0,833	3,063	59,278	114,597	74,278	128,198	1,363	10,831
	Sys_E	3,79	3,79	1,587	1,080	1,600	1,090	38,795	61,864	53,136	72,452	1,022	8,123

Table: DESTINY simulation, 5 memory systems, 3 technology nodes, Latency (Read and Write), Energy (Read and Write), Leakage power for L1-Cache (32kB) and L2-Cache (512kB)

				DESTINY DATA									
				Latency				**Energy**				**Leakage**	
				32KB		512kB		32KB		512kB		32KB	512kB
Nodes	**System**			**L1-Cache**		**L2-Cache**		**L1-Cache**		**L2-Cache**		**L1-Cache**	**L2-Cache**
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**-**	**-**
nm	-	GHz	GHz	ns	ns	ns	ns	(pJ)	(pJ)	(pJ)	(pJ)	(mW)	(mW)
28	Sys_A	2,68	2,68	0,373	0,349	0,411	0,368	60,081	40,298	111,400	88,392	2,655	42,209
	Sys_B	2,68	2,68	0,373	0,349	0,806	3,813	60,081	40,298	129,019	228,129	2,655	5,232
	Sys_C	2,68	2,68	0,373	0,349	1,373	1,485	60,081	40,298	93,352	129,221	2,655	3,924
	Sys_D	2,68	2,68	0,767	3,790	0,806	3,813	77,517	179,280	129,019	228,129	0,329	5,232
	Sys_E	2,68	2,68	1,352	1,470	1,373	1,485	53,531	91,625	93,352	129,221	0,247	3,924
24	Sys_A	3,18	3,18	0,314	0,296	0,342	0,310	56,100	37,500	96,024	76,496	3,559	56,593
	Sys_B	3,18	3,18	0,314	0,296	0,825	3,405	56,100	37,500	108,632	183,546	3,559	6,831
	Sys_C	3,18	3,18	0,314	0,296	1,448	1,273	56,100	37,500	79,510	107,483	3,559	5,123
	Sys_D	3,18	3,18	0,796	3,392	0,825	3,405	69,800	144,640	108,632	183,546	0,430	6,831
	Sys_E	3,18	3,18	1,432	1,266	1,448	1,273	49,242	76,254	79,510	107,483	0,322	5,123
20	Sys_A	3,79	3,79	0,264	0,247	0,285	0,262	47,325	32,833	79,802	63,690	11,282	179,358
	Sys_B	3,79	3,79	0,264	0,247	0,845	3,069	47,325	32,833	90,963	144,883	11,282	21,662
	Sys_C	3,79	3,79	0,264	0,247	1,607	1,093	47,325	32,833	65,838	85,155	11,282	16,246
	Sys_D	3,79	3,79	0,822	3,043	0,845	3,069	59,278	114,597	90,963	144,883	1,363	21,662
	Sys_E	3,79	3,79	1,587	1,080	1,607	1,093	38,795	61,864	65,838	85,155	1,022	16,246

Table: DESTINY simulation, 5 memory systems, 3 technology nodes, Latency (Read and Write), Energy (Read and Write), Leakage power for L1-Cache (32kB) and L2-Cache (1MB)

				DESTINY DATA									
				Latency				Energy				Leakage	
				32KB		1MB		32KB		1MB		32KB	1MB
Nodes	System			L1-Cache		L2-Cache		L1-Cache		L2-Cache		L1-Cache	L2-Cache
	Config	CPU_clk	System_clk	Read	Write	Read	Write	Read	Write	Read	Write	-	-
nm	-	GHz	GHz	ns	ns	ns	ns	(pJ)	(pJ)	(pJ)	(pJ)	(mW)	(mW)
28	Sys_A	2,68	2,68	0,373	0,349	0,508	0,417	60,081	40,298	158,445	135,437	2,655	84,419
	Sys_B	2,68	2,68	0,373	0,349	0,902	3,861	60,081	40,298	175,815	274,925	2,655	10,464
	Sys_C	2,68	2,68	0,373	0,349	1,428	1,512	60,081	40,298	128,964	164,834	2,655	7,848
	Sys_D	2,68	2,68	0,767	3,790	0,902	3,861	77,517	179,280	175,815	274,925	0,329	10,464
	Sys_E	2,68	2,68	1,352	1,470	1,428	1,512	53,531	91,625	128,964	164,834	0,247	7,848
24	Sys_A	3,18	3,18	0,314	0,296	0,413	0,345	56,100	37,500	136,349	116,821	3,559	113,185
	Sys_B	3,18	3,18	0,314	0,296	0,897	3,440	56,100	37,500	148,695	223,609	3,559	13,662
	Sys_C	3,18	3,18	0,314	0,296	1,489	1,293	56,100	37,500	109,999	137,972	3,559	10,246
	Sys_D	3,18	3,18	0,796	3,392	0,897	3,440	69,800	144,640	148,695	223,609	0,430	13,662
	Sys_E	3,18	3,18	1,432	1,266	1,489	1,293	49,242	76,254	109,999	137,972	0,322	10,246
20	Sys_A	3,79	3,79	0,264	0,247	0,334	0,287	47,325	32,833	113,406	97,294	11,282	358,715
	Sys_B	3,79	3,79	0,264	0,247	0,893	3,094	47,325	32,833	124,328	178,248	11,282	43,324
	Sys_C	3,79	3,79	0,264	0,247	1,635	1,107	47,325	32,833	91,237	110,554	11,282	32,493
	Sys_D	3,79	3,79	0,822	3,043	0,893	3,094	59,278	114,597	124,328	178,248	1,363	43,324
	Sys_E	3,79	3,79	1,587	1,080	1,635	1,107	38,795	61,864	91,237	110,554	1,022	32,493

Table: DESTINY simulation, 5 memory systems, 3 technology nodes, Latency (Read and Write), Energy (Read and Write), Leakage power for L1-Cache (32kB) and L2-Cache (2MB)

				DESTINY DATA									
				Latency				**Energy**				**Leakage**	
				32KB		2MB		32KB		2MB		32KB	2MB
Nodes	**System**			**L1-Cache**		**L2-Cache**		**L1-Cache**		**L2-Cache**		**L1-Cache**	**L2-Cache**
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**-**	**-**
nm	-	GHz	GHz	ns	ns	ns	ns	(pJ)	(pJ)	(pJ)	(pJ)	(mW)	(mW)
28	Sys_A	2,68	2,68	0,373	0,349	0,895	0,610	60,081	40,298	252,652	229,644	2,655	168,838
	Sys_B	2,68	2,68	0,373	0,349	1,284	4,052	60,081	40,298	269,490	368,600	2,655	20,928
	Sys_C	2,68	2,68	0,373	0,349	1,649	1,623	60,081	40,298	200,249	236,119	2,655	15,696
	Sys_D	2,68	2,68	0,767	3,790	1,284	4,052	77,517	179,280	269,490	368,600	0,329	20,928
	Sys_E	2,68	2,68	1,352	1,470	1,649	1,623	53,531	91,625	200,249	236,119	0,247	15,696
24	Sys_A	3,18	3,18	0,314	0,296	0,698	0,487	56,100	37,500	217,099	197,571	3,559	226,370
	Sys_B	3,18	3,18	0,314	0,296	1,186	3,581	56,100	37,500	228,894	303,808	3,559	27,324
	Sys_C	3,18	3,18	0,314	0,296	1,651	1,375	56,100	37,500	171,027	199,000	3,559	20,493
	Sys_D	3,18	3,18	0,796	3,392	1,186	3,581	69,800	144,640	228,894	303,808	0,430	27,324
	Sys_E	3,18	3,18	1,432	1,266	1,651	1,375	49,242	76,254	171,027	199,000	0,322	20,493
20	Sys_A	3,79	3,79	0,264	0,247	0,531	0,386	47,325	32,833	180,698	164,586	11,282	717,431
	Sys_B	3,79	3,79	0,264	0,247	1,088	3,191	47,325	32,833	191,117	245,037	11,282	86,647
	Sys_C	3,79	3,79	0,264	0,247	1,748	1,164	47,325	32,833	142,078	161,394	11,282	64,985
	Sys_D	3,79	3,79	0,822	3,043	1,088	3,191	59,278	114,597	191,117	245,037	1,363	86,647
	Sys_E	3,79	3,79	1,587	1,080	1,748	1,164	38,795	61,864	142,078	161,394	1,022	64,985

Annexes 2) Latency cycles for simulation in Gem5

Table: Latency cycles for L2-Cache template (128kB)

				GEM5 Simulated Data							
				32KB				128kB			
Nodes	System			L1-Cache-Exact		L1-Cache-Aprox		L2-Cache-Exact		L2-Cache-Aprox	
	Config	CPU_clk	System_clk	Read	Write	Read	Write	Read	Write	Read	Write
nm	-	GHz	GHz	cycles	cycles	cycles	cycles	cycles	cycles	cycles	cycles
28	Sys_A	2,68	2,68	1	0,934	1	1	1,020	0,946	1	1
	Sys_B	2,68	2,68	1	0,934	1	1	2,079	10,174	3	11
	Sys_C	2,68	2,68	1	0,934	1	1	3,630	3,954	4	4
	Sys_D	2,68	2,68	2,056	10,152	3	10	2,079	10,174	3	11
	Sys_E	2,68	2,68	3,622	3,938	4	4	3,630	3,954	4	4
24	Sys_A	3,18	3,18	1	0,943	1	1	1,017	0,950	1	1
	Sys_B	3,18	3,18	1	0,943	1	1	2,555	10,802	3	11
	Sys_C	3,18	3,18	1	0,943	1	1	4,569	4,031	5	4
	Sys_D	3,18	3,18	2,533	10,795	3	11	2,555	10,802	3	11
	Sys_E	3,18	3,18	4,556	4,028	5	5	4,569	4,031	5	4
20	Sys_A	3,79	3,79	1	0,937	1	1	1,019	0,963	1	1
	Sys_B	3,79	3,79	1	0,937	1	1	3,141	11,593	4	12
	Sys_C	3,79	3,79	1	0,937	1	1	6,051	4,122	7	5
	Sys_D	3,79	3,79	3,112	11,520	4	12	3,141	11,593	4	12
	Sys_E	3,79	3,79	6,011	4,088	6	5	6,051	4,122	7	5

Table: Latency cycles for L2-Cache template (256kB)

				GEM5 Simulated Data							
				32KB				**256kB**			
Nodes	**System**			**L1-Cache-Exact**		**L1-Cache-Aprox**		**L2-Cache-Exact**		**L2-Cache-Aprox**	
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
nm	**-**	**GHz**	**GHz**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**
28	Sys_A	2,68	2,68	1	0,934	1	1	1,037	0,954	1	1
	Sys_B	2,68	2,68	1	0,934	1	1	2,095	10,182	2	11
	Sys_C	2,68	2,68	1	0,934	1	1	3,640	3,958	4	4
	Sys_D	2,68	2,68	2,056	10,152	3	10	2,095	10,182	2	11
	Sys_E	2,68	2,68	3,622	3,938	4	4	3,640	3,958	4	4
24	Sys_A	3,18	3,18	1	0,943	1	1	1,032	0,957	1	1
	Sys_B	3,18	3,18	1	0,943	1	1	2,569	10,809	3	11
	Sys_C	3,18	3,18	1	0,943	1	1	4,577	4,035	5	4
	Sys_D	3,18	3,18	2,533	10,795	3	11	2,569	10,809	3	11
	Sys_E	3,18	3,18	4,556	4,028	5	5	4,577	4,035	5	4
20	Sys_A	3,79	3,79	1	0,937	1	1	1,031	0,969	1	1
	Sys_B	3,79	3,79	1	0,937	1	1	3,153	11,599	4	12
	Sys_C	3,79	3,79	1	0,937	1	1	6,058	4,126	7	5
	Sys_D	3,79	3,79	3,112	11,520	4	12	3,153	11,599	4	12
	Sys_E	3,79	3,79	6,011	4,088	6	5	6,058	4,126	7	5

Table: Latency cycles for L2-Cache template (512kB)

				GEM5 Simulated Data							
				32KB				512kB			
Nodes	**System**			**L1-Cache-Exact**		**L1-Cache-Aprox**		**L2-Cache-Exact**		**L2-Cache-Aprox**	
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
nm	**-**	**GHz**	**GHz**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**
28	Sys_A	2,68	2,68	1	0,934	1	1	1,101	0,986	2	1
	Sys_B	2,68	2,68	1	0,934	1	1	2,159	10,214	3	11
	Sys_C	2,68	2,68	1	0,934	1	1	3,677	3,977	4	4
	Sys_D	2,68	2,68	2,056	10,152	3	10	2,159	10,214	3	11
	Sys_E	2,68	2,68	3,622	3,938	4	4	3,677	3,977	4	4
24	Sys_A	3,18	3,18	1	0,943	1	1	1,088	0,985	2	1
	Sys_B	3,18	3,18	1	0,943	1	1	2,626	10,837	3	11
	Sys_C	3,18	3,18	1	0,943	1	1	4,609	4,051	5	5
	Sys_D	3,18	3,18	2,533	10,795	3	11	2,626	10,837	3	11
	Sys_E	3,18	3,18	4,556	4,028	5	5	4,609	4,051	5	5
20	Sys_A	3,79	3,79	1	0,937	1	1	1,077	0,993	2	1
	Sys_B	3,79	3,79	1	0,937	1	1	3,199	11,622	4	12
	Sys_C	3,79	3,79	1	0,937	1	1	6,085	4,139	7	5
	Sys_D	3,79	3,79	3,112	11,520	4	12	3,199	11,622	4	12
	Sys_E	3,79	3,79	6,011	4,088	6	5	6,085	4,139	7	5

Table: Latency cycles for L2-Cache template (1MB)

				GEM5 Simulated Data							
				32KB				1MB			
Nodes	**System**			**L1-Cache-Exact**		**L1-Cache-Aprox**		**L2-Cache-Exact**		**L2-Cache-Aprox**	
	Config	**CPU_clk**	**System_clk**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
nm	**-**	**GHz**	**GHz**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**	**cycles**
28	Sys_A	2,68	2,68	1	0,934	1	1	1,361	1,116	2	2
	Sys_B	2,68	2,68	1	0,934	1	1	2,415	10,342	3	11
	Sys_C	2,68	2,68	1	0,934	1	1	3,825	4,051	4	5
	Sys_D	2,68	2,68	2,056	10,152	3	10	2,415	10,342	3	11
	Sys_E	2,68	2,68	3,622	3,938	4	4	3,825	4,051	4	5
24	Sys_A	3,18	3,18	1	0,943	1	1	1,314	1,099	2	2
	Sys_B	3,18	3,18	1	0,943	1	1	2,856	10,948	3	11
	Sys_C	3,18	3,18	1	0,943	1	1	4,739	4,116	5	5
	Sys_D	3,18	3,18	2,533	10,795	3	11	2,856	10,948	3	11
	Sys_E	3,18	3,18	4,556	4,028	5	5	4,739	4,116	5	5
20	Sys_A	3,79	3,79	1	0,937	1	1	1,264	1,086	2	2
	Sys_B	3,79	3,79	1	0,937	1	1	3,383	11,714	4	12
	Sys_C	3,79	3,79	1	0,937	1	1	6,191	4,192	7	5
	Sys_D	3,79	3,79	3,112	11,520	4	12	3,383	11,714	4	12
	Sys_E	3,79	3,79	6,011	4,088	6	5	6,191	4,192	7	5

Table: Latency cycles for L2-Cache template (2MB)

				GEM5 Simulated Data							
				32KB				2MB			
Nodes	System			L1-Cache-Exact		L1-Cache-Aprox		L2-Cache-Exact		L2-Cache-Aprox	
	Config	CPU_clk	System_clk	Read	Write	Read	Write	Read	Write	Read	Write
nm	-	GHz	GHz	cycles	cycles	cycles	cycles	cycles	cycles	cycles	cycles
28	Sys_A	2,68	2,68	1	0,934	1	1	2,398	1,635	3	2
	Sys_B	2,68	2,68	1	0,934	1	1	3,440	10,855	4	11
	Sys_C	2,68	2,68	1	0,934	1	1	4,419	4,348	5	5
	Sys_D	2,68	2,68	2,056	10,152	3	10	3,440	10,855	4	11
	Sys_E	2,68	2,68	3,622	3,938	4	4	4,419	4,348	5	5
24	Sys_A	3,18	3,18	1	0,943	1	1	2,220	1,551	3	2
	Sys_B	3,18	3,18	1	0,943	1	1	3,774	11,395	4	12
	Sys_C	3,18	3,18	1	0,943	1	1	5,255	4,374	6	5
	Sys_D	3,18	3,18	2,533	10,795	3	11	3,774	11,395	4	12
	Sys_E	3,18	3,18	4,556	4,028	5	5	5,255	4,374	6	5
20	Sys_A	3,79	3,79	1	0,937	1	1	2,012	1,460	2	2
	Sys_B	3,79	3,79	1	0,937	1	1	4,119	12,082	5	13
	Sys_C	3,79	3,79	1	0,937	1	1	6,618	4,406	7	5
	Sys_D	3,79	3,79	3,112	11,520	4	12	4,119	12,082	5	13
	Sys_E	3,79	3,79	6,011	4,088	6	5	6,618	4,406	7	5

Annexes 3) List of benchmarks and memory templates

```
# ========================= Creating list of benchmarks ========================= #
declare -a benches
bench_list=(
sha
patricia
FFT
susan_c
dijkstra
blowfish
basicmath_large
qsort_large
bitcount
stringsearch
CRC32
)
# ============================================================ #

# ============= Creating system configurations ========================= #
declare -a system
system_list=(

SRAM_SRAM_1MB_28nm
SRAM_SB_1MB_28nm
SRAM_DB_1MB_28nm
SB_SB_1MB_28nm
DB_DB_1MB_28nm

#SRAM_SRAM_128kB_28nm
#SRAM_SB_128kB_28nm
#SRAM_DB_128kB_28nm
#SB_SB_128kB_28nm
#DB_DB_128kB_28nm

#SRAM_SRAM_256kB_28nm
#SRAM_SB_256kB_28nm
#SRAM_DB_256kB_28nm
#SB_SB_256kB_28nm
#DB_DB_256kB_28nm

#SRAM_SRAM_512kB_28nm
#SRAM_SB_512kB_28nm
#SRAM_DB_512kB_28nm
#SB_SB_512kB_28nm
#DB_DB_512kB_28nm

#SRAM_SRAM_2MB_28nm
#SRAM_SB_2MB_28nm
#SRAM_DB_2MB_28nm
#SB_SB_2MB_28nm
#DB_DB_2MB_28nm

#SRAM_SRAM_1MB_24nm
#SRAM_SB_1MB_24nm
#SRAM_DB_1MB_24nm
#SB_SB_1MB_24nm
#DB_DB_1MB_24nm

#SRAM_SRAM_128kB_24nm
#SRAM_SB_128kB_24nm
#SRAM_DB_128kB_24nm
#SB_SB_128kB_24nm
#DB_DB_128kB_24nm

#SRAM_SRAM_256kB_24nm
#SRAM_SB_256kB_24nm
#SRAM_DB_256kB_24nm
#SB_SB_256kB_24nm
#DB_DB_256kB_24nm

#SRAM_SRAM_512kB_24nm
#SRAM_SB_512kB_24nm
#SRAM_DB_512kB_24nm
#SB_SB_512kB_24nm
#DB_DB_512kB_24nm

#SRAM_SRAM_2MB_24nm
#SRAM_SB_2MB_24nm
#SRAM_DB_2MB_24nm
#SB_SB_2MB_24nm
#DB_DB_2MB_24nm

#SRAM_SRAM_1MB_20nm
#SRAM_SB_1MB_20nm
#SRAM_DB_1MB_20nm
#SB_SB_1MB_20nm
#DB_DB_1MB_20nm

#SRAM_SRAM_128kB_20nm
#SRAM_SB_128kB_20nm
#SRAM_DB_128kB_20nm
#SB_SB_128kB_20nm
#DB_DB_128kB_20nm

#SRAM_SRAM_256kB_20nm
#SRAM_SB_256kB_20nm
#SRAM_DB_256kB_20nm
#SB_SB_256kB_20nm
#DB_DB_256kB_20nm

#SRAM_SRAM_512kB_20nm
#SRAM_SB_512kB_20nm
#SRAM_DB_512kB_20nm
#SB_SB_512kB_20nm
#DB_DB_512kB_20nm

#SRAM_SRAM_2MB_20nm
#SRAM_SB_2MB_20nm
#SRAM_DB_2MB_20nm
#SB_SB_2MB_20nm
#DB_DB_2MB_20nm
)

# ============================================================#
```

Annexes 4) Main shell file

```
#!/bin/bash
# This is a shell file

source DeclaringBenchesAndSystems.sh

for bench in ${bench_list[*]} #======================================== begin main
(first) loop
do

for system_confing in ${system_list[*]}  # begin second loop
do

Src_Dir_Exe=/mnt/d/Documents/gem5-master_1-4-2019_VF1/configs/project_FelixV1/Shell_files

source $Src_Dir_Exe/$system_confing.sh


start=$(date +%s) # time in seconds
#=================¡=== Stetting the paths and their targets
============================#
InputFileName=Two_Cache_System
Input_setup=$InputFileName.py # Its the input python file containing all the configuration for our system
Target=BenchmarkOut_$InputFileName # Its the name of the output folder where the benchmark simulation
results will be placed
ISA=X86 # Istructions set architecture. NOTE! This should match with the compiled architecture of the object
file (binary)

Output_Dir=/mnt/d/Documents/gem5-master_1-4-2019_VF1/m5out/"$system_confing"_"$bench" # Its the
output directory for the benchmark simulations
Output_log_file=$Output_Dir/runscript.log # Log file to debug the inputs
Gem5_Dir=/mnt/d/Documents/gem5-master_1-4-2019_VF1  # Its the gem5 directory instalation folder
ScriptSim_Dir=/mnt/d/Documents/gem5-master_1-4-2019_VF1/configs/project_FelixV1 # Its path to the python
script where we will run the simulation
SPEC2006_Dir=/mnt/d/Documents/gem5-master_1-4-2019_VF1/GEM5_BENCHMARKS/GEM5_SPEC2006 #
Spec2006 binaries directory
MiBench_Dir=/mnt/d/Documents/gem5-master_1-4-2019_VF1/GEM5_BENCHMARKS/GEM5_MiBench#
MiBench binaries directory
WL1_bin_Dir=$SPEC2006_Dir/bin/"$ISA" # Workload (number 1) binaries to run
WL2_bin_Dir=$MiBench_Dir/bin/"$ISA"   # Workload (number 2) binaries to run
#===============================================================
===#

#==============¡¡=== Setting input & output environment
==============================#
source $Src_Dir_Exe/SettingOutEnv_CheckErrors.sh # Making the necessary directories and checking errors
source $Src_Dir_Exe/Benchmark_Options.sh #Benchmark options
source $Src_Dir_Exe/StartSim_message.sh #Starting simulation message
#===============================================================
===#

#======================= Running Gem5 simulation
===============================#
cd $Run_Bin
        $Gem5_Dir/build/$ISA/gem5.opt --outdir=$Output_Dir $ScriptSim_Dir/$Input_setup \
                            --benchmark=$Benchmark --benchmark_stdout=$Output_Dir/$Benchmark.out \
--Frequency=$Frequency --Voltage=0.8V --L2_latency_ns_write=$L2_latency_ns_write --
L2_latency_ns_read=$L2_latency_ns_read \
--L2_latency_cycles_read=$L2_latency_cycles_read --L2_latency_cycles_write=$L2_latency_cycles_write \
--L2_size=$L2_size --L1i_size=$L1i_size --L1d_size=$L1d_size \
--L1i_latency_ns_read=$L1i_latency_ns_read --L1i_latency_ns_write=$L1i_latency_ns_write \
--L1d_latency_ns_read=$L1d_latency_ns_read --L1d_latency_ns_write=$L1d_latency_ns_write \
--L1d_latency_cycles_read=$L1d_latency_cycles_read --L1i_latency_cycles_read=$L1i_latency_cycles_read \
--L1d_latency_cycles_write=$L1d_latency_cycles_write --
L1i_latency_cycles_write=$L1i_latency_cycles_write \
 --benchmark_stderr=$Output_Dir/$Benchmark.err  | tee -a $Output_log_file

echo "================= Simulation finished ===================" | tee -a $Output_log_file

end=$(date +%s)
runtime=$(echo "$end - $start" | bc)
echo "It takes $runtime seconds to complete the simulation with the benchmark $Benchmark" | tee -a
$Output_log_file
echo ""
echo ""
echo ""
#=================================================================
===#

done; # ============================================= end second loop

done; # ============================================= end main (first) loop
```

Annexes 5) System script file (Two_Cache_System)

```
from __future__ import print_function      # Bringing the print function from Python 3 into Python 2.6+ (For possible compatibility problems)
from __future__ import absolute_import
import MiBench_benchmarks              # Importing MiBench benchmarks
import optparse     # Populate it with options, and parse the command line. It can generates usage and help messages.
import sys                     # Making the script possible to interact with the host system (e.g. the exit of the script)
import os                     # Allowing Python to interface with the operating system (Windows, Mac, Linux)
import m5                      # Import the m5 (gem5) library created when gem5 is built
from m5.defines import buildEnv           # Useful depending on the system. ??????
from m5.objects import *             # Importing all the SimObjects
from m5.util import addToPath, fatal, warn  # Importing useful functionalities

addToPath('../')

from common import SimpleOpts
from caches import *                                  # import the caches which we made
from optparse import OptionParser


#=============== Calling, adding and setting options ==============================#
   # Adding personal options:
SimpleOpts.add_option("--Frequency", type="string", default="", help="The system frequency.")
SimpleOpts.add_option("--Voltage", type="string", default="", help="The system voltage.")
SimpleOpts.add_option("--L2_latency_ns_read", type="string", default="", help="L2-Cache Read latency in ns (Buffer).")
SimpleOpts.add_option("--L2_latency_ns_write", type="string", default="", help="L2-Cache Write latency in ns (Buffer).")
SimpleOpts.add_option("--L1i_latency_ns_read", type="string", default="0ns", help="L1i-Cache Read latency in ns (Buffer).")
SimpleOpts.add_option("--L1i_latency_ns_write", type="string", default="0ns", help="L1i-Cache Write latency in ns (Buffer).")
SimpleOpts.add_option("--L1d_latency_ns_read", type="string", default="0ns", help="L1d-Cache Read latency in ns (Buffer).")
SimpleOpts.add_option("--L1d_latency_ns_write", type="string", default="0ns", help="L1d-Cache Write latency in ns (Buffer).")
SimpleOpts.add_option("-b", "--benchmark", type="string", default="", help="The benchmark to be loaded.")
SimpleOpts.add_option("--benchmark_stdout", type="string", default="", help="Absolute path for stdout redirection for the benchmark.")
SimpleOpts.add_option("--benchmark_stderr", type="string", default="", help="Absolute path for stderr redirection for the benchmark.")

(options, args) = SimpleOpts.parse_args()   # Finalize the arguments and grab the options so we can pass it on to our objects
if args:
   print("Error: script doesn't take any positional arguments")
   sys.exit(1)
#==================================================================#

#================== Adding Benchmark options ===============================#
benchmarks = MiBench_benchmarks
if options.benchmark:
   if options.benchmark == 'basicmath_large':
      print ("Selected MiBench benchmark:")
      print ("--> basicmath_large")
      process = benchmarks.basicmath_large
   elif options.benchmark == 'basicmath_small':
      print ("Selected MiBench benchmark:")
      print ("--> basicmath_small")
```

```
    process = benchmarks.basicmath_small
elif options.benchmark == 'bitcount':
    print ("Selected MiBench benchmark:")
    print ("--> bitcount")
    process = benchmarks.bitcount
elif options.benchmark == 'qsort_large':
    print ("Selected MiBench benchmark:")
    print ("--> qsort_large")
    process = benchmarks.qsort_large
elif options.benchmark == 'qsort_small':
    print ("Selected MiBench benchmark:")
    print ("--> qsort_small")
    process = benchmarks.qsort_small
elif options.benchmark == 'susan_s':
    print ("Selected MiBench benchmark:")
    print ("--> susan_s")
    process = benchmarks.susan_s
elif options.benchmark == 'susan_e':
    print ("Selected MiBench benchmark:")
    print ("--> susan_e")
    process = benchmarks.susan_e
elif options.benchmark == 'susan_c':
    print ("Selected MiBench benchmark:")
    print ("--> susan_c")
    process = benchmarks.susan_c
elif options.benchmark == 'cjpeg':
    print ("Selected MiBench benchmark:")
    print ("--> cjpeg")
    process = benchmarks.cjpeg
elif options.benchmark == 'djpeg':
    print ("Selected MiBench benchmark:")
    print ("--> djpeg")
    process = benchmarks.djpeg
elif options.benchmark == 'lame':          # Buffer overflow problems
    print ("Selected MiBench benchmark:")
    print ("--> lame")
    process = benchmarks.lame
elif options.benchmark == 'mad':          #error
    print ("Selected MiBench benchmark:")
    print ("--> mad")
    process = benchmarks.madplay
elif options.benchmark == 'FFT':
    print ("Selected MiBench benchmark:")
    print ("--> FFT")
    process = benchmarks.FFT
elif options.benchmark == 'CRC32':
    print ("Selected MiBench benchmark:")
    print ("--> CRC32")
    process = benchmarks.crc
elif options.benchmark == 'sha':
    print ("Selected MiBench benchmark:")
    print ("--> sha")
    process = benchmarks.sha
elif options.benchmark == 'blowfish':
    print ("Selected MiBench benchmark:")
    print ("--> blowfish")
    process = benchmarks.blowfish
elif options.benchmark == 'rijndael':
    print ("Selected MiBench benchmark:")
    print ("--> rijndael")
    process = benchmarks.rijndael
elif options.benchmark == 'stringsearch':
    print ("Selected MiBench benchmark:")
```

```
        print ("--> stringsearch")
        process = benchmarks.stringsearch
    elif options.benchmark == 'dijkstra':
        print ("Selected MiBench benchmark:")
        print ("--> dijkstra")
        process = benchmarks.dijkstra
    elif options.benchmark == 'patricia':
        print ("Selected MiBench benchmark:")
        print ("--> patricia")
        process = benchmarks.patricia
    else:
        print ("Benchmark selected is not recognized! Exiting.")
        sys.exit(1)
else:
    print >> sys.stderr, "Need --benchmark switch to specify workload. Exiting!\n"
    sys.exit(1)

    # Setting the stdout and stderr files for each benchmark
if options.benchmark_stdout:
    process.output = options.benchmark_stdout
    print ("  The output of the executed benchmark is:")
    print ("  Process stdout file: " + process.output)
if options.benchmark_stderr:
    process.errout = options.benchmark_stderr
    print ("  Process stderr file: " + process.errout)
    print ("  ================================")

#==================================================================#

#==========================System Features===================================
freq = options.Frequency
Vdd = options.Voltage
system_voltage = Vdd        # Top-level voltage for blocks running at system power supply
system_clock = freq        # Top-level clock for blocks running at system speed
cpu_voltage = Vdd           # CPU voltage
cpu_clock = freq           # Clock for blocks running at CPU speed
system_memory_ranges = '8192MB'  # System memory ranges fo the MAIN MEMORY. Memory size
system_cache_line_size = '64'   # Cache line size in bytes (On x86 cache lines are 64 bytes)
    # L1-Cache:
l1d_read_latency = options.L1d_latency_ns_read          #read latency
l1d_write_latency = options.L1d_latency_ns_write       #write latency
l1i_read_latency = options.L1i_latency_ns_read          #read latency
l1i_write_latency = options.L1i_latency_ns_write       #write latency
    # L2-Cache:
STT_L2_write_latency = options.L2_latency_ns_write          #write latency
STT_L2_read_latency = options.L2_latency_ns_read            #read latency
    # Memory
main_memory = DDR3_1600_8x8()
#==================================================================


#========== Definition of the System and its features ================================#
system = System() # Defining the system

    # System voltage and clock (the Top level values)
system.voltage_domain = VoltageDomain(voltage = system_voltage) # Create a top-level voltage domain
system.clk_domain = SrcClockDomain(clock =  system_clock,
                                  voltage_domain = system.voltage_domain) # Create a source clock for the system and set
the clock period

    # Creating CPU and its type
system.cpu = TimingSimpleCPU() # Establishing the CPU type, CPU to run with
system.mem_mode = 'timing'     # Use timing accesses in order to match the CPU type (timingSimpleCPU must
match with mem_mode=timing)

   # CPU clock and voltage
system.cpu_voltage_domain = VoltageDomain(voltage = cpu_voltage)# Create a CPU voltage domain
system.cpu_clk_domain = SrcClockDomain(clock = cpu_clock,
                                       voltage_domain = system.cpu_voltage_domain) # Create a separate clock domain for
the CPUs

   # Addiontal must-have system specifications
system.mem_ranges = AddrRange(system_memory_ranges) # Create an address range/Ranges that constitute
main memory, these can be passed
                                                    # from the I/O subsystem through an I/O bridge or cache
system.cache_line_size = system_cache_line_size # Cache line size in bytes
#==================================================================#

#====================== Defining Memories ================================#
   # Defining main memory controller
system.mem_ctrl = main_memory        # Definition of the main memory
system.mem_ctrl.range = system.mem_ranges[0]

   # Level 1 cache memories definition (instruction and data cache)
system.cpu.icache = L1ICache(options) # Instruction cache
system.cpu.dcache = L1DCache(options) # Data cache
system.l1d_Latency_buffer = SimpleMemDelay(read_resp = l1d_read_latency, write_resp = l1d_write_latency)
system.l1i_Latency_buffer = SimpleMemDelay(read_resp = l1i_read_latency, write_resp = l1i_write_latency)
   # Level 2 cache memory definition (instruction and data cache)
system.l2cache = L2Cache(options)
system.l2_Latency_buffer = SimpleMemDelay(read_resp = STT_L2_read_latency, write_resp =
STT_L2_write_latency)

   # Busses creation
system.l2bus = L2XBar()            # L2 Bus
system.membus = SystemXBar()       # Memory bus
#==================================================================

#================ Making the interconnections ¡¡================================#
   # Connections of L1-I and L1-D to CPU
system.cpu.icache.connectCPU(system.cpu)    # Connect the instruction cache to the CPU
system.cpu.dcache.connectCPU(system.cpu)    # Connect the data cache to the CPU

   # Connection of L1-I and L1-D to Buffer
system.cpu.icache.connectBottomPort(system.l1i_Latency_buffer.slave)  # Connect the instruction cache to the
buffer
system.cpu.dcache.connectBottomPort(system.l1d_Latency_buffer.slave)  # Connect the data cache to the buffer

   # Connection of L1-I and L1-D to L2-Bus
system.l1d_Latency_buffer.master = system.l2bus.slave
system.l1i_Latency_buffer.master = system.l2bus.slave

   # Connection of L2 cache
system.l2_Latency_buffer.slave = system.l2bus.master
system.l2cache.connectTopPort(system.l2_Latency_buffer.master)
system.l2cache.connectMemSideBus(system.membus) # Connect L2 cache to L2 Memory Bus (Bottom terminal)

   # Connection of Memory bus to the Main Memory
system.cpu.createInterruptController()         # For the case of a X86 architecture
if m5.defines.buildEnv['TARGET_ISA'] == "x86":
   system.cpu.interrupts[0].pio = system.membus.master
   system.cpu.interrupts[0].int_master = system.membus.slave
   system.cpu.interrupts[0].int_slave = system.membus.master

system.system_port = system.membus.slave
system.mem_ctrl.port = system.membus.master
#==================================================================#


#=========== Calling the desired process and creating the necessary threads ================#
system.cpu[0].workload = process # Set the cpu to use the process as its workload
print ("")
print ("  The executed process is:",process.cmd)
print ("")
system.cpu[0].createThreads() # Creating threads to start the process
#==================================================================#

#================= Starting the Simulation ==================================#
root = Root(full_system = False, system = system) # set up the root SimObject and start the simulation
m5.instantiate() # instantiate all of the objects we've setted above
print("************ Starting Simulation ************")
exit_event = m5.simulate()
print('Exiting @ tick %i because %s' % (m5.curTick(), exit_event.getCause()))
#==================================================================#
```

Annexes 6) Modification code to add the write_latency parameter

(1°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\cache.py

```
class BaseCache(MemObject):
    type = 'BaseCache'
    abstract = True
    cxx_header = "mem/cache/base.hh"

    size = Param.MemorySize("Capacity")
    assoc = Param.Unsigned("Associativity")

    tag_latency = Param.Cycles("Tag lookup latency")
    write_latency=Param.Cycles("The cache write latency ")      → ADD  THIS OPTION
    data_latency = Param.Cycles("Data access latency")  [corresponds to read latency according to the author script]
    response_latency = Param.Cycles("Latency for the return path on a miss")
```

(2°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\base.hh

Search for "const Cycles dataLatency" in sublime text, you should see something like this:

```
    /**
     * The latency of data access of a cache. It occurs when there is
     * an access to the cache.
     */
    const Cycles dataLatency;
```

Now add the following lines next to the last line (const Cycles dataLatency;**)**

```
    /**
     * The latency of a write in a memory.
     */
    const Cycles writeLatency;
```

(3°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\base.cc

You will see the following code:

```
BaseCache::BaseCache(const BaseCacheParams *p, unsigned blk_size)
    : MemObject(p),
      cpuSidePort (p->name + ".cpu_side", this, "CpuSidePort"),
      memSidePort(p->name + ".mem_side", this, "MemSidePort"),
      mshrQueue("MSHRs", p->mshrs, 0, p->demand_mshr_reserve), // see below
      writeBuffer("write buffer", p->write_buffers, p->mshrs), // see below
      tags(p->tags),
      prefetcher(p->prefetcher),
      writeAllocator(p->write_allocator),
      writebackClean(p->writeback_clean),
      tempBlockWriteback(nullptr),
      writebackTempBlockAtomicEvent([this]{ writebackTempBlockAtomic(); },
                                    name(), false,
                                    EventBase::Delayed_Writeback_Pri),
      blkSize(blk_size),
      lookupLatency(p->tag_latency),
      dataLatency(p->data_latency),
      writeLatency(p->write_latency),      → ADD THIS LINE
      forwardLatency(p->tag_latency),
      fillLatency(p->data_latency),
      responseLatency(p->response_latency),
      sequentialAccess(p->sequential_access),
```

(4°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\cache.cc

Look for this part of the code

```
                // How many bytes past the first request is this one
                int transfer_offset =
                    tgt_pkt->getOffset(blkSize) - initial_offset;
                if (transfer_offset < 0) {
                    transfer_offset += blkSize;
```

```
            }

            // If not critical word (offset) return payloadDelay.
            // responseLatency is the latency of the return path
            // from lower level caches/memory to an upper level cache or
            // the core.
            completion_time += clockEdge(responseLatency) +
               ~~(transfer_offset ? pkt->payloadDelay : 0);~~ → cancel this and put:
            writeLatency * clockPeriod() + (transfer_offset ? pkt->-payloadDelay : 0);

            assert(!tgt_pkt->req->isUncacheable());
```

(5°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\cache.cc

Now we have to declare this <u>writeLatency</u> inside this code. Look for the function:

```
Cache::access(PacketPtr pkt, CacheBlk *&blk, Cycles &lat,
              PacketList &writebacks)
{

    if (pkt->req->isUncacheable()) {
        assert(pkt->isRequest());

        chatty_assert(!(isReadOnly && pkt->isWrite()),
                      "Should never see a write in a read-only cache %s\n",
                      name());

        DPRINTF(Cache, "%s for %s\n", __func__, pkt->print());

        // flush and invalidate any existing block
        CacheBlk *old_blk(tags->findBlock(pkt->getAddr(), pkt->isSecure()));
        if (old_blk && old_blk->isValid()) {
            BaseCache::evictBlock(old_blk, writebacks);
        }

        blk = nullptr;
        // lookupLatency is the latency in case the request is uncacheable.
        lat = lookupLatency;

      // Update latency decided by if it is read or write    → ADD THE FOLLOWING BLOCK
        if(pkt->isWrite()) {
            lat = writeLatency;
        }
        return false;
    }
    // Update latency decided by if it is read or write
    if(pkt->isWrite()) {
        lat = writeLatency;
    }
    }

    return BaseCache::access(pkt, blk, lat, writebacks);  → here lat is receiving the pertinent latency
}
```

(6°) C:\Users\PC\Desktop\gem5\gem5\src\mem\cache\base.cc

Go to the following part:

```
   DPRINTF(Cache, "Block addr %#llx (%s) moving from state %x to %s\n",
           addr, is_secure ? "s" : "ns", old_state, blk->print());

    // if we got new data, copy it in (checking for a read response
    // and a response that has data is the same in the end)
    if (pkt->isRead()) {
        // sanity checks
        assert(pkt->hasData());
        assert(pkt->getSize() == blkSize);

        pkt->writeDataToBlock(blk->data, blkSize);
    }
    // The block will be ready when the payload arrives and the fill is done
    blk->setWhenReady(clockEdge(fillLatency) + pkt->headerDelay + writeLatency * clockPeriod() +
                    pkt->payloadDelay);

    return blk;
}
```

Annexes 7) Cache file code

```
from __future__ import print_function
from __future__ import absolute_import
import m5
from m5.objects import Cache              #Instruccion para importar los SimObjects al archivo

# Add the common scripts to our path
from m5.util import addToPath, fatal, warn  # Importing useful functionalities

addToPath('../')
from common import SimpleOpts
# Some specific options for caches
# For all options see src/mem/cache/Cache.py
class L1Cache(Cache):
    """Simple L1 Cache with default values"""
    assoc = 2
    tag_latency = 2
    data_latency = 0 # 2 cycle. This is the reading latency
    response_latency = 2 # 2 cycles
    mshrs = 4
    tgts_per_mshr = 20

    def __init__(self, options=None):
        super(L1Cache, self).__init__()
        pass

    def connectoBus(self, bus):
        """Connect this cache to a memory-side bus"""
        self.mem_side = bus

    def connectBus(self, bus):
        """Connect this cache to a memory-side bus"""
        self.mem_side = bus.slave

    def connectCPU(self, cpu):
        """Connect this cache's port to a CPU-side port
          This must be defined in a subclass"""
        raise NotImplementedError

    def connectBottomPort(self, bus): # def connectCPUSideBus(self, bus): For latency l1d and l1i, using
mem_side
        self.mem_side = bus  # self.cpu_side = bus.master

class L1ICache(L1Cache):
    """Simple L1 instruction cache with default values"""
    # Set the default size
    size = '32kB' #16kB default
    write_latency = 0
    data_latency = 2 # 2 cycles
    SimpleOpts.add_option('--L1i_size',
                    help="L1 instruction cache size. Default: %s" % size)
    SimpleOpts.add_option('--L1i_latency_cycles_read',
                    help="L1 instruction cache read latency. Default: %s" % data_latency)
    SimpleOpts.add_option('--L1i_latency_cycles_write',
                    help="L1 instruction cache write latency. Default: %s" % write_latency)
    def __init__(self, opts=None):
        super(L1ICache, self).__init__(opts)
        if not opts or not opts.L1i_size or not opts.L1i_latency_cycles_read or not opts.L1i_latency_cycles_write:
            return
        self.size = opts.L1i_size
        self.data_latency = opts.L1i_latency_cycles_read
        self.write_latency = opts.L1i_latency_cycles_write

    def connectCPU(self, cpu):
        """Connect this cache's port to a CPU icache port"""
        self.cpu_side = cpu.icache_port

class L1DCache(L1Cache):
    """Simple L1 data cache with default values"""
    # Set the default size
    size = '32kB' #64kB default
    data_latency = 3 # 2 cycles
    write_latency = 0
    writeback_clean = True
    SimpleOpts.add_option('--L1d_size',
                          help="L1 data cache size. Default: %s" % size)
    SimpleOpts.add_option('--L1d_latency_cycles_read',
                          help="L1 instruction cache read latency. Default: %s" % data_latency)
    SimpleOpts.add_option('--L1d_latency_cycles_write',
                          help="L1 instruction cache write latency. Default: %s" % write_latency)
    def __init__(self, opts=None):
        super(L1DCache, self).__init__(opts)
        if not opts or not opts.L1d_size or not opts.L1d_latency_cycles_read or not
opts.L1d_latency_cycles_write:
            return
        self.size = opts.L1d_size
        self.data_latency = opts.L1d_latency_cycles_read
        self.write_latency = opts.L1d_latency_cycles_write

    def connectCPU(self, cpu):
        """Connect this cache's port to a CPU dcache port"""
        self.cpu_side = cpu.dcache_port

class L2Cache(Cache):
    # Default parameters
    size = '1MB'
    assoc = 8
    tag_latency = 20
    data_latency = 0
    write_latency = 0
    response_latency = 20
    mshrs = 20
    tgts_per_mshr = 12
    SimpleOpts.add_option('--L2_size', help="L2 cache size. Default: %s" % size)
    SimpleOpts.add_option('--L2_latency_cycles_read', help="L2 cache read latency. Default: %s" %
data_latency)
    SimpleOpts.add_option('--L2_latency_cycles_write', help="L2 cache read latency. Default: %s" %
write_latency)
    def __init__(self, opts=None):
        super(L2Cache, self).__init__()
        if not opts or not opts.L2_size or not opts.L2_latency_cycles_read or not opts.L2_latency_cycles_write:
            return
        self.size = opts.L2_size
        self.data_latency = opts.L2_latency_cycles_read
        self.write_latency = opts.L2_latency_cycles_write

    def connectTopPort(self, bus): # def connectCPUSideBus(self, bus):
        self.cpu_side = bus  # self.cpu_side = bus.master

    def connectCPUSideBus(self, bus):
        self.cpu_side = bus.master

    def connectMemSideBus(self, bus):
        self.mem_side = bus.slave
```

Annexes 8) Benchmarks file code

```
import m5
from m5.objects import *
m5.util.addToPath('../common')

binary_dir = '/home/gyz/code/architecture/mibench/'
data_dir = '/home/gyz/code/architecture/mibench/data/'

#--------------------------- basicmath_large-------------------------------------------#
basicmath_large = Process()
basicmath_large.executable = 'basicmath_large' # Testing
data = ''
basicmath_large.cmd = [basicmath_large.executable] + [data]
#----------------------------------------------------------------------------------#

#-------------------------- basicmath_small------------------------------------------#
basicmath_small = Process()
basicmath_small.executable = 'basicmath_small' # Testing
data = ''
basicmath_small.cmd = [basicmath_small.executable] + [data]
#----------------------------------------------------------------------------------#

#---------------------------------- bitcount -------------------------------------------#
bitcount = Process()
bitcount.executable = 'bitcnts' # bitcount parameter is fixed in the code
data = '1125000' # For testing large simulation
#data = '75000'   # For testing small simulation
bitcount.cmd = [bitcount.executable] + [data]
#-----------------------------------------------------------------------------------#

#---------------------------------- qsort_large ---------------------------------------#
qsort_large = Process()
qsort_large.executable = 'qsort_large'
data = 'input_large.dat'
qsort_large.cmd = [qsort_large.executable] + [data]
#-----------------------------------------------------------------------------------#

#---------------------------------- qsort_small ---------------------------------------#
qsort_small = Process()
qsort_small.executable = 'qsort_small'
data = 'input_small.dat'
qsort_small.cmd = [qsort_small.executable] + [data]
#-----------------------------------------------------------------------------------#

#------------------------------------ susan -------------------------------------------#
#susan_s
susan_s = Process()
susan_s.executable = 'susan'
#data = 'input_large.pgm'
data = 'input_small.pgm'
susan_s.cmd = [susan_s.executable] + [data] + ['-s']

#susan_e
susan_e = Process()
susan_e.executable = 'susan'
#data = 'input_large.pgm'
data = 'input_small.pgm'
susan_e.cmd = [susan_s.executable] + [data] + ['-e']

#susan_c
```

```
susan_c = Process()
susan_c.executable = 'susan'
data = 'input_large.pgm'
#data = 'input_small.pgm'
susan_c.cmd = [susan_c.executable] + [data] + ['-c']
#------------------------------------------------------------------------------------------#

#------------------------------------- FFT -------------------------------------------------#
FFT = Process()
FFT.executable = 'fft'
outputfile = 'output.txt'
#data = ['4'] + ['4096']
data = ['8'] + ['32768']
#data_large = ['32'] + ['131072']
#data = ['2'] + ['2048']
FFT.cmd = [FFT.executable] + data
#---------------------------------------------------------------------------------------------#

#------------------------------------- CRC -------------------------------------------------#
crc = Process()
crc.executable = 'crc'
data = 'large_crc.pcm' # example data large
#data = 'small_crc.pcm' # example data small
crc.cmd = [crc.executable] + [data]
#---------------------------------------------------------------------------------------------#

#------------------------------------- sha -------------------------------------------------#
sha  = Process()
sha.executable =  'sha'
data = 'input_large.asc'
#data = 'input_small.asc'
sha.cmd = [sha.executable] + [data]
#---------------------------------------------------------------------------------------------#

#------------------------------------- blowfish ---------------------------------------------#
blowfish = Process()
blowfish.executable =  'bf'
data = 'input_large_blowfish.asc'
#data = 'input_small_blowfish.asc'
blowfish.cmd = [blowfish.executable] + ['e'] + [data] + ['output_large.enc'] +
['1234567890abcdeffedcba0987654321'] #change output to large or small according the input
#---------------------------------------------------------------------------------------------#

#------------------------------------- stringsearch ----------------------------------------#
stringsearch = Process()
stringsearch.executable =  'search_large' # large data
#stringsearch.executable =  'search_small' # Small data
stringsearch.cmd = [stringsearch.executable]
#---------------------------------------------------------------------------------------------#

#------------------------------------- dijkstra --------------------------------------------#
dijkstra = Process()
dijkstra.executable =  'dijkstra_large' # dijkstra_large is also available
data = 'input_dijkstra.dat'
dijkstra.cmd = [dijkstra.executable] + [data]
#---------------------------------------------------------------------------------------------#

#------------------------------------- patricia --------------------------------------------#
patricia  = Process()
patricia.executable =  'patricia'
data = 'large.udp' # large data
#data = 'small.udp' # small data
patricia.cmd = [patricia.executable] + [data]
```

```
#---------------------------------------------------------------------------------------#

#typeset
typeset = Process()
typeset.executable = binary_dir + 'consumer/typeset/lout-3.24/lout'
typeset_param = '-I lout-3.24/include -D lout-3.24/data -F lout-3.24/font -C lout-3.24/maps -H lout-3.24/hyph
large.out'
typeset_base = '/home/gyz/code/architecture/mibench/consumer/typeset/'
typeset_include_dir = typeset_base + 'lout-3.24/include'
typeset_data_dir = typeset_base + 'lout-3.24/data'
typeset_font_dir = typeset_base + 'lout-3.24/font'
typeset_map_dir = typeset_base + 'lout-3.24/map'
typeset_hyph_dir = typeset_base + 'lout-3.24/hyph'
typeset_largelout=typeset_base + 'large.lout'

typeset.cmd = [typeset.executable]  + ['-I'] + [typeset_include_dir] + ['-D'] + [typeset_data_dir] + ['-F'] +
[typeset_font_dir] + ['-C'] + [typeset_map_dir] + ['-H'] + [typeset_hyph_dir] + [typeset_largelout]

#FFT_i
FFT_i = Process()
FFT_i.executable = binary_dir + 'telecomm/FFT/fft'
FFT_i.cmd = [FFT_i.executable] + ['8'] + ['32768'] + ['-i']
#FFT_i.output  = output_dir+ 'FFT_i.out'
#====================
```

Annexes 9) File code seV1.py

```
from __future__ import print_function
from __future__ import absolute_import

import optparse
import sys
import os

import m5
from m5.defines import buildEnv
from m5.objects import *
from m5.util import addToPath, fatal, warn

addToPath('../')

from common import OptionsV1
#from common import SimpleOpts
from common import Simulation

from common import CacheConfigV1
from common import CpuConfig
from common import BPConfig
from common import MemConfig
from common.Caches import *
from common.cpu2000 import *
from caches import *                    # import the caches which we made
from optparse import OptionParser

def get_processes(options):
    """Interprets provided options and returns a list of processes"""

    multiprocesses = []
    inputs = []
    outputs = []
    errouts = []
    pargs = []

    #cmd benchmark and benchmark path
    workloads = options.cmd.split(';')
    if options.input != "":
        inputs = options.input.split(';')
    if options.output != "":
        outputs = options.output.split(';')
    if options.errout != "":
        errouts = options.errout.split(';')
    if options.options != "":
        pargs = options.options.split(';')

    idx = 0
    for wrkld in workloads:

        process = Process(pid = 100 + idx)
        process.executable = wrkld
        process.cwd = os.getcwd()

        if options.env:
            with open(options.env, 'r') as f:
                process.env = [line.rstrip() for line in f]

        if len(pargs) > idx:
            process.cmd = [wrkld] + pargs[idx].split()
        else:
            process.cmd = [wrkld]

        if len(inputs) > idx:
            process.input = inputs[idx]
        if len(outputs) > idx:
            process.output = outputs[idx]
        if len(errouts) > idx:
            process.errout = errouts[idx]

        multiprocesses.append(process)
        idx += 1

    if options.smt:
        assert(options.cpu_type == "DerivO3CPU")
        return multiprocesses, idx
    else:
        return multiprocesses, 1

parser = optparse.OptionParser()
OptionsV1.addCommonOptions(parser)
OptionsV1.addSEOptions(parser)
(options, args) = parser.parse_args()

if args:
    print("Error: script doesn't take any positional arguments")
    sys.exit(1)

multiprocesses = []
numThreads = 1

if options.bench:
    apps = options.bench.split("-")
    if len(apps) != options.num_cpus:
        print("number of benchmarks not equal to set num_cpus!")
        sys.exit(1)

    for app in apps:
        try:
            if buildEnv['TARGET_ISA'] == 'alpha':
                exec("workload = %s('alpha', 'tru64', '%s')" % (
                        app, options.spec_input))
            elif buildEnv['TARGET_ISA'] == 'arm':
                exec("workload = %s('arm_%s', 'linux', '%s')" % (
                        app, options.arm_iset, options.spec_input))
            else:
                exec("workload = %s(buildEnv['TARGET_ISA', 'linux', '%s')" % (
                        app, options.spec_input))
            multiprocesses.append(workload.makeProcess())
        except:
            print("Unable to find workload for %s: %s" %
                  (buildEnv['TARGET_ISA'], app),
                  file=sys.stderr)
            sys.exit(1)
elif options.cmd:
    multiprocesses, numThreads = get_processes(options)

else:
    print("No workload specified. Exiting!\n", file=sys.stderr)
    sys.exit(1)

(CPUClass, test_mem_mode, FutureClass) = Simulation.setCPUClass(options)
CPUClass.numThreads = numThreads
```

```
# Check -- do not allow SMT with multiple CPUs
if options.smt and options.num_cpus > 1:
   fatal("You cannot use SMT with multiple CPUs!")

np = options.num_cpus
system = System(cpu = [CPUClass(cpu_id=i) for i in range(np)],
                mem_mode = test_mem_mode,
                mem_ranges = [AddrRange(options.mem_size)],
                cache_line_size = options.cacheline_size)

if numThreads > 1:
   system.multi_thread = True

# Create a top-level voltage domain
system.voltage_domain = VoltageDomain(voltage = options.sys_voltage)

# Create a source clock for the system and set the clock period
system.clk_domain = SrcClockDomain(clock =  options.sys_clock,
                                   voltage_domain = system.voltage_domain)

# Create a CPU voltage domain
system.cpu_voltage_domain = VoltageDomain()

# Create a separate clock domain for the CPUs
system.cpu_clk_domain = SrcClockDomain(clock = options.cpu_clock,
                                       voltage_domain =
                                       system.cpu_voltage_domain)

# If elastic tracing is enabled, then configure the cpu and attach the elastic
# trace probe
if options.elastic_trace_en:
   CpuConfig.config_etrace(CPUClass, system.cpu, options)

# All cpus belong to a common cpu_clk_domain, therefore running at a common
# frequency.
for cpu in system.cpu:
   cpu.clk_domain = system.cpu_clk_domain

# Sanity check
if options.simpoint_profile:
   if not CpuConfig.is_noncaching_cpu(CPUClass):
      fatal("SimPoint/BPProbe should be done with an atomic cpu")
   if np > 1:
      fatal("SimPoint generation not supported with more than one CPUs")

for i in range(np):
   if options.smt:
      system.cpu[i].workload = multiprocesses
   elif len(multiprocesses) == 1:
      system.cpu[i].workload = multiprocesses[0]
   else:
      system.cpu[i].workload = multiprocesses[i]

   if options.simpoint_profile:
      system.cpu[i].addSimPointProbe(options.simpoint_interval)

   if options.checker:
      system.cpu[i].addCheckerCpu()

   if options.bp_type:
      bpClass = BPConfig.get(options.bp_type)
```

```
        system.cpu[i].branchPred = bpClass()

    system.cpu[i].createThreads()

MemClass = Simulation.setMemClass(options)
system.membus = SystemXBar()
system.system_port = system.membus.slave
CacheConfigV1.config_cache(options, system)
MemConfig.config_mem(options, system)

root = Root(full_system = False, system = system)
Simulation.run(options, root, system, FutureClass)
```

Annexes 10) File codes CacheConfigV1.py

```
from __future__ import print_function
from __future__ import absolute_import
import m5
from m5.objects import *
from .Caches import *

# Add the common scripts to our path

from m5.util import addToPath, fatal, warn  # Importing useful functionalities

addToPath('../')
from common import SimpleOpts

def config_cache(options, system):

    if options.external_memory_system and (options.caches or options.l2cache):
        print("External caches and internal caches are exclusive options.\n")
        sys.exit(1)

    if options.external_memory_system:
        ExternalCache = ExternalCacheFactory(options.external_memory_system)

    if options.cpu_type == "O3_ARM_v7a_3":
        try:
            import cores.arm.O3_ARM_v7a as core
        except:
            print("O3_ARM_v7a_3 is unavailable. Did you compile the O3 model?")
            sys.exit(1)

        dcache_class, icache_class, l2_cache_class, walk_cache_class = \
            core.O3_ARM_v7a_DCache, core.O3_ARM_v7a_ICache, \
            core.O3_ARM_v7aL2, \
            core.O3_ARM_v7aWalkCache
    elif options.cpu_type == "HPI":
        try:
            import cores.arm.HPI as core
        except:
            print("HPI is unavailable.")
            sys.exit(1)

        dcache_class, icache_class, l2_cache_class, walk_cache_class = \
            core.HPI_DCache, core.HPI_ICache, core.HPI_L2, core.HPI_WalkCache
    else:

        dcache_class, icache_class, l2_cache_class, walk_cache_class = \
            L1_DCache, L1_ICache, L2Cache, None

        if buildEnv['TARGET_ISA'] == 'x86':
            walk_cache_class = PageTableWalkerCache

    # Set the cache line size of the system
    system.cache_line_size = options.cacheline_size

    # If elastic trace generation is enabled, make sure the memory system is
    # minimal so that compute delays do not include memory access latencies.
    # Configure the compulsory L1 caches for the O3CPU, do not configure
    # any more caches.
    if options.l2cache and options.elastic_trace_en:
        fatal("When elastic trace is enabled, do not configure L2 caches.")
```

```
if options.l2cache:
    # Provide a clock for the L2 and the L1-to-L2 bus here as they
    # are not connected using addTwoLevelCacheHierarchy. Use the
    # same clock as the CPUs.
    system.l2 = l2_cache_class(clk_domain=system.cpu_clk_domain,
                               size=options.l2_size,
                               assoc=options.l2_assoc
                               #latency_cycles_read=options.l2_latency_cycles_read,
                               #latency_cycles_write=options.l2_latency_cycles_write)
                               )

    system.tol2bus = L2XBar(clk_domain = system.cpu_clk_domain)

    system.l2.cpu_side = system.tol2bus.master
    system.l2.mem_side = system.membus.slave

if options.memchecker:
    system.memchecker = MemChecker()

for i in range(options.num_cpus):
    if options.caches:

        icache = icache_class(size=options.l1i_size,
                              assoc=options.l1i_assoc
                              #latency_cycles_read=options.l1i_latency_cycles_read,
                              #latency_cycles_write=options.l1i_latency_cycles_write)
                              )

        dcache = dcache_class(size=options.l1d_size,
                              assoc=options.l1d_assoc
                              #latency_cycles_read=options.l1d_latency_cycles_read,
                              #latency_cycles_write=options.l1d_latency_cycles_write)
                              )

        # If we have a walker cache specified, instantiate two
        # instances here
        if walk_cache_class:
            iwalkcache = walk_cache_class()
            dwalkcache = walk_cache_class()
        else:
            iwalkcache = None
            dwalkcache = None

        if options.memchecker:
            dcache_mon = MemCheckerMonitor(warn_only=True)
            dcache_real = dcache

            # Do not pass the memchecker into the constructor of
            # MemCheckerMonitor, as it would create a copy; we require
            # exactly one MemChecker instance.
            dcache_mon.memchecker = system.memchecker

            # Connect monitor
            dcache_mon.mem_side = dcache.cpu_side

            # Let CPU connect to monitors
            dcache = dcache_mon

        # When connecting the caches, the clock is also inherited
        # from the CPU in question
        system.cpu[i].addPrivateSplitL1Caches(icache, dcache,
                                              iwalkcache, dwalkcache)
            if options.memchecker:
                # The mem_side ports of the caches haven't been connected yet.
                # Make sure connectAllPorts connects the right objects.
                system.cpu[i].dcache = dcache_real
                system.cpu[i].dcache_mon = dcache_mon

        elif options.external_memory_system:

            # These port names are presented to whatever 'external' system
            # gem5 is connecting to.  Its configuration will likely depend
            # on these names.  For simplicity, we would advise configuring
            # it to use this naming scheme; if this isn't possible, change
            # the names below.
            if buildEnv['TARGET_ISA'] in ['x86', 'arm']:
                system.cpu[i].addPrivateSplitL1Caches(
                        ExternalCache("cpu%d.icache" % i),
                        ExternalCache("cpu%d.dcache" % i),
                        ExternalCache("cpu%d.itb_walker_cache" % i),
                        ExternalCache("cpu%d.dtb_walker_cache" % i))
            else:
                system.cpu[i].addPrivateSplitL1Caches(
                        ExternalCache("cpu%d.icache" % i),
                        ExternalCache("cpu%d.dcache" % i))

        system.cpu[i].createInterruptController()
        if options.l2cache:
            system.cpu[i].connectAllPorts(system.tol2bus, system.membus)
        elif options.external_memory_system:
            system.cpu[i].connectUncachedPorts(system.membus)
        else:
            system.cpu[i].connectAllPorts(system.membus)

    return system

# ExternalSlave provides a "port", but when that port connects to a cache,
# the connecting CPU SimObject wants to refer to its "cpu_side".
# The 'ExternalCache' class provides this adaptation by rewriting the name,
# eliminating distracting changes elsewhere in the config code.
class ExternalCache(ExternalSlave):
    def __getattr__(cls, attr):
        if (attr == "cpu_side"):
            attr = "port"
        return super(ExternalSlave, cls).__getattr__(attr)

    def __setattr__(cls, attr, value):
        if (attr == "cpu_side"):
            attr = "port"
        return super(ExternalSlave, cls).__setattr__(attr, value)

def ExternalCacheFactory(port_type):
    def make(name):
        return ExternalCache(port_data=name, port_type=port_type,
                             addr_ranges=[AllMemory])
    return make
```