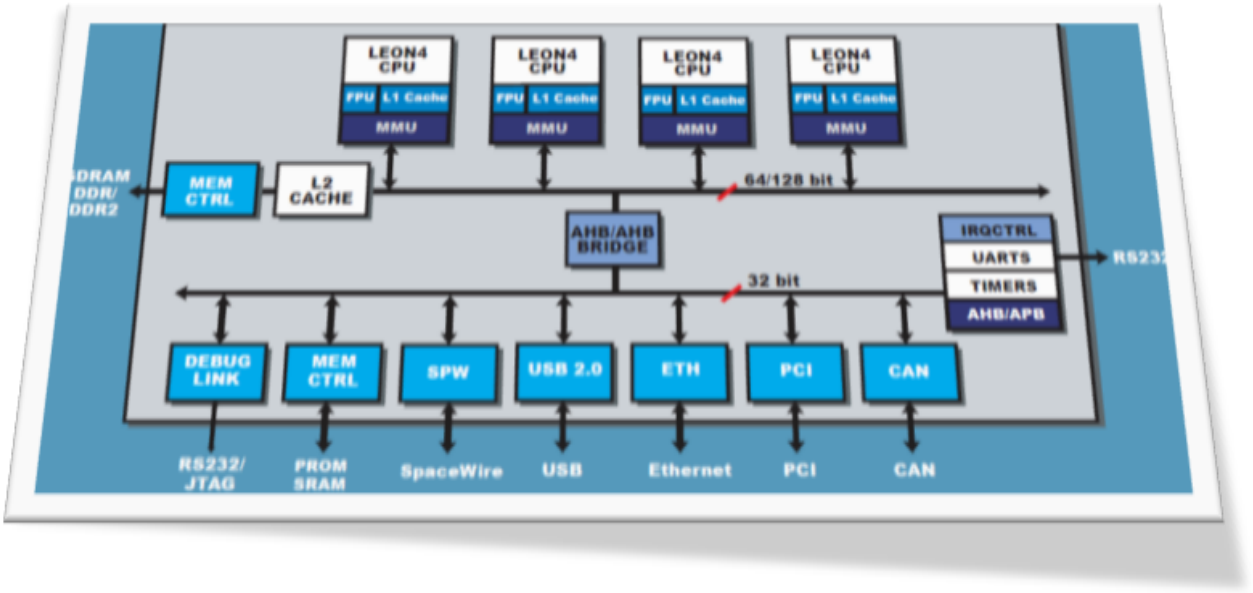
CHALMERS



A Method for Estimation of Safe and Tight WCET in Multicore Memory Hierarchies

Master of Science Thesis in Embedded Electronics System Design

Feysal Hashi

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, August 2013

A Method for Estimation of Safe and Tight WCET in Multicore Memory Hierarchies

FEYSAL HASHI

Examiner: PER STENSTRÖM

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Quad-core LEON4, a fault-tolerant SPARC V8 architecture based 32-bit processor. See Section 3.1.1 for more details about LEON4.

Department of Computer Science and Engineering
Göteborg, Sweden, August 2013

Abstract

Multicore microprocessors have become the new way to improve the processor performance. Moreover, the multicore processor systems are currently the dominating computer resource in many products like mobile appliances, automotive and space-borne applications. However, the last category's challenge is that tasks can have various degrees of criticality concerning the response times. Therefore, the aim of this Master's Thesis is to develop a method to derive a safe (the task execution time can never be longer) and a tight (worst-case execution time is off the real execution time by typically a small factor) worst-case execution time estimates in multicore system with memory hierarchies. The produced method consists of an analytical model and gem5, which is a state-of-the-art computer architecture simulator. This Thesis has surprisingly discovered that the simulated multicore system's throughput increases by close to linearly with the number of cores. This suggests that, contrary to common belief, it is possible to guarantee a safe worst-case execution time and still enjoy the increase in throughput offered by multicore systems. These findings are therefore encouraging for further investigations. Although, the outcome of this Thesis mainly focuses on LEON4FT, which is a SPARC V8 based System-on-a-Chip, it is also applicable to other multicore systems. Furthermore, the Thesis report introduces the previous studies' arguments about the transition from uniprocessor to the multicore processors, memory hierarchies as well as worst-case execution time.

Keywords: Multicore microprocessors, multilevel memory hierarchies, worst-case execution time, gem5, throughput, System-on-a-Chip, parallel execution, serial execution, cache, memory wall, power wall, SPARC V8.

Acknowledgements

I would like to express my gratitude to Professor Per Stenstrom for making this thesis possible in many ways, particularly the lectures, encouragements, insightful comments and suggestions. Without his guidance and persistent help, this thesis would not have been possible. I would also like to show my greatest appreciation to Madhavan Manivannan for helping with the simulation tool and always been there to support me. Special thanks also to Angelos Arelakis for introducing me to the simulation tool. Last but not least I would like to thank my family and all the people who helped me to complete this thesis.

Table of Contents

List of Figures

List of Tables

1. Introduction

Multicore microprocessors have in a few years' time become the mainstream computer resource in a spectrum of products such as mobile appliances i.e. tablets and smart phones as well as in automotive and space-borne applications. In the latter category, a special challenge is that tasks can have various degrees of criticality concerning the response times. Some computational tasks will not jeopardize the functionality if computational deadlines are not met (non-critical tasks) whereas for safety-critical tasks (or critical tasks), a missed deadline can lead to catastrophic consequences. In general, a catastrophic consequence may arise when a control function is not executed on time such as a too late effect on the brake actuator.

The goal of this project is to develop a method to derive a safe (the task execution time can never be longer) and a tight (worst-case execution time (WCET) is off the real execution time by typically a small factor) WCET estimates in multicore system with memory hierarchies. What is needed is to set up methods to analyse a task before its execution to establish the WCET.

WCET must make sufficiently conservative assumptions to be safe and tight. To illustrate the challenges involved, imagine that we want to establish WCET for a task that does a memory access. A memory-access either hits or misses at any level in the memory hierarchy. A safe estimate would be to assume it misses at all levels. Such a WCET estimate would not be tight; it would be a large factor longer than the actual execution time. Good news is that prior art has established methods to estimate safe and acceptably tight bounds on memory access time [1] in *single-processor* systems. In multicore systems, the new issue is that several processors may concurrently access some of the levels in the memory hierarchy creating a new source of pessimism that make multicores unattractive in mixed criticality applications.

Individual processors on a multicore platform logically share the same memory system. The memory system is typically implemented using a traditional memory hierarchy in which the first cache level is physically private to each processor whereas the next level is physically shared. The goal of this thesis is to define a method by which we can make safe and tight WCET estimates on a multicore platform taking into account that some of the levels of the memory hierarchy are shared.

The method chosen consists of three components, namely: a simulation tool instead of a costly hardware option, an application that is representative for the application domain and test case that would verify the method before a full scale simulation is undertaken. The choice of the method would be based on the above project scope, time span and cost. Most importantly, the method should be able to derive a safe and a tight WCET estimates in multicore system with memory hierarchies as LEON4FT (LEON4), which is the target System-on-a-Chip (SoC) for space-born applications.

The Computer Science and Engineering (CSE) department at Chalmers University of technology that facilitated this project has internationally recognised professors in the field of the computer architecture as well as Ph.D. students who are working with the latest researches in the genre of this master's thesis. Therefore, it was obvious to start the search for the most appropriate method at the CSE department by presenting and discussing above criteria.

After consultation with the CSE department experts and studying other researchers' work [2] [3] in the field, it has been identified that gem5, which is a state-of-the-art computer architecture simulator widely used by the computer architecture research community, would be the best tool to simulate a complete single/multicore system with multilevel caches and memory. One of the outstanding strengths that gem5 has over other similar simulators such as SimpleScalar is that it supports many different ISAs, CPU types, cache levels, memory and other components. Moreover, all the gem5 components are highly configurable i.e. the CPU speed, the cache size and associativity etc. More details about the gem5 simulator and scripts created to automatize the simulation and the data collection are presented in the system assumption chapter.

A matrix multiplication (MM) application that multiplies two matrices and saves the result in a third matrix has been chosen to run on the simulation tool for measuring the WCET. MM was found to be flexible as test case since it is easy to increase the size of the matrix and/or the system capabilities while measuring the WCET changes due to the increase of the application workload. MM was also found to utilise the cache/memory locality hence is a good way to test the benefits of a multilevel cache systems and its influence on the WCET. Furthermore, it was found that MM is a common method used by many other researchers in the area [4] [5] [6].

Before porting the MM application onto the gem5 simulator it was important to verify that the tool is working as it was intended. Therefore, simple applications that make a controlled number of memory reads and writes were created to run on to the simulation tool. After the simulation tool parameters were configured correctly and many tests were carried out, the output statistics from the tool has been compared against the expected number of cache/memory access to verify that the simulator is working as expected. Finally, the software was ready to run a full scale simulation and to output the data needed to calculate the WCET.

Prior to the simulation result, the presumption has been that by doing a pessimistic but safe estimation of the execution time, each task would execute so slowly that it does not pay off to run tasks in parallel. On the contrary, and quite surprisingly, it was found that the derived WCET estimate for parallel tasks has shown that it pays off to run tasks in parallel. The main result of this thesis is based on simulations carried out for 18 applications that carry out matrix multiplications, whereas the first application's matrix size is 32x32 and the 18th is 304x304. The cache miss-rates, the numbers of instructions and other parameters/variables from the simulation of these applications have been used to calculate the execution time for 4 tasks running serially and 4 running in parallel. The 4 serial tasks' WCET are then divided by the 4 parallel tasks' WCET. This revealed that the parallel tasks' throughput is very close to 4, which is the optimal upper limit for 4 tasks. This result confirms that by running 4 tasks in parallel in a multicore system, one achieves safer and tighter WCET estimate than running them in serial.

This thesis's project is limited to 30 weeks. Therefore, the experiments are carried out using gem5, which is a state-of-the-art computer architecture simulator widely by the computer architecture research community [2]. Nevertheless, a continuation of the project, as a doctoral project, will consider the architectural support to bring down the software overheads further to open up for safe and tighter estimates.

The multicore system being considered in the project is LEON4FT core, a SPARC V8 based System-on-a-Chip. This system is used by RUAG, a company in Gothenburg, for space-borne applications (e.g., satellites) and this project is done in collaboration with RUAG.

The remaining chapters of the report are structured as follows. Chapter 2 gives the theoretical background of this master's thesis. Chapter 3 describes the simulated system, the application used to estimate the WCET, the simulation system and the equations used to measure the execution time. Chapter 4 presents the results whereas it is analysed and discussed in Chapter 5. Finally, the conclusion of this master's thesis is presented in Chapter 6.

2. Theory

In this chapter, a brief history of the transformation of single-core systems to multicore systems will be presented. Then, memory hierarchies in general are discussed and multicore caches in particular will be introduced. Finally, the WCET, which is a central topic of this thesis, will be considered.

2.1 Multicore Systems

2.1.1 From Uniprocessors to Multicore processor

Uniprocessors a.k.a. single-core processors have been the dominating technology of the desktop/general purpose computers, embedded systems and other specialised processors for decades until the beginning of the 2000s. In 1965 Gordon Moore predicted that transistor count would double every 18 month, this prophecy was later called Moore's Law [7]. Figure 1 illustrates the Moore's prediction. The doubling transistor count was mainly apparent from 1986 to 2003 [8]. During these days, higher chip speed and lower production cost has been achieved by shrinking the transistor size (also called transistor scale down) to integrate as many transistors as possible in the same die while increasing the clock frequency [7]. This approach that is to increase the uniprocessors speed to gain higher performance for the existing applications has been quite straightforward [9].

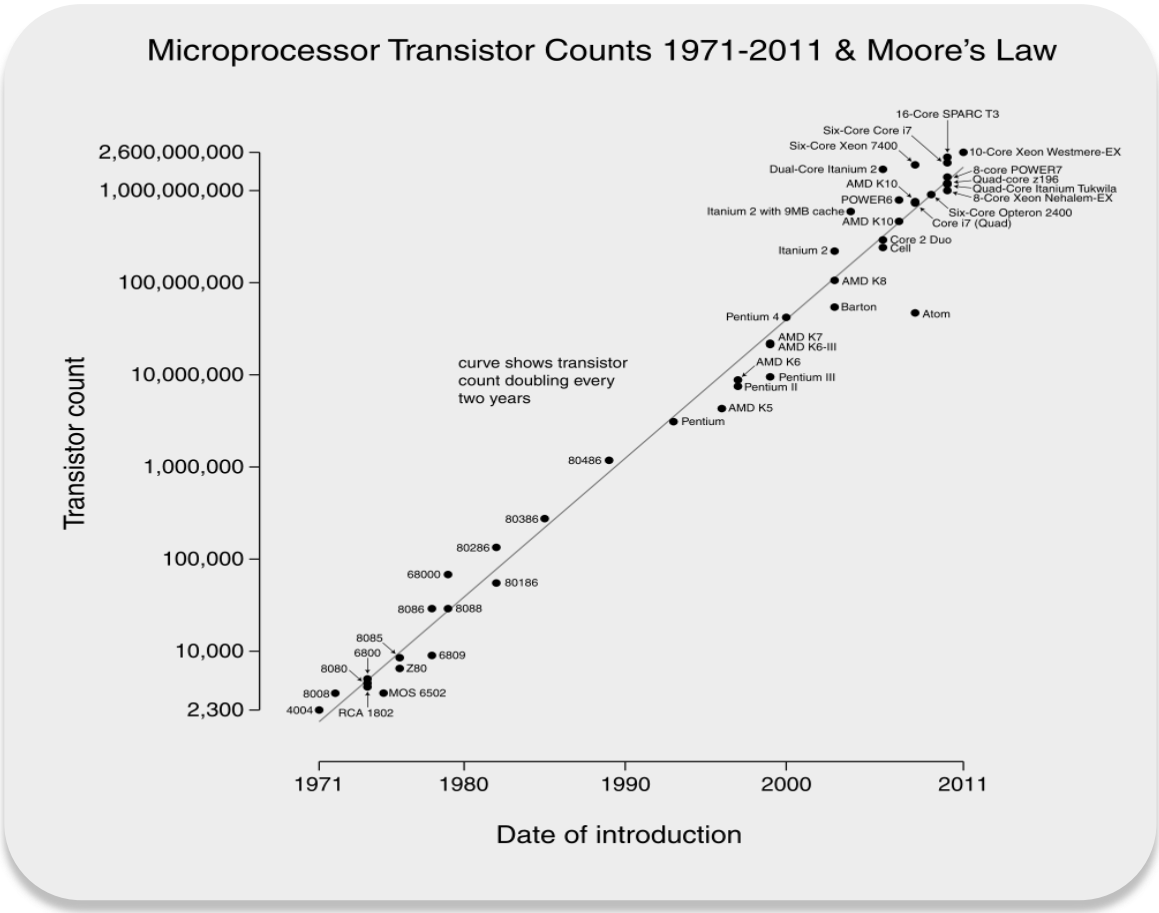


Figure 1: Moore's prediction [10].

Nevertheless, this trend of uniprocessor speed growth through transistor size scale down and clock frequency increase, as mentioned above, ended in the early 2000s. According to [11], [12] and others, the major cause that this trend could not be continued was the intense temperature rise due to the ever escalating power dissipation (see Equation 1, the power consumption). This is presumably a bigger drawback for the embedded systems since they usually have limited power source. One can say that the performance advancements of the single-core processors came to the point that the physical constants such as the speed of light, the size of electrons and the silicon operation temperature become the limit [13]. This is called the power wall [14].

Equation 1: Power = CV^2f, (where C is capacitance, f is frequency and V is voltage)

Another, challenge has been the memory wall phenomena, which is the widening speed gap been the uniprocessors and the memory [15]. Figure 2 is showing this phenomena starting from 1980 to 2010. Furthermore, deeper pipelines, instruction-level parallelism, and speculative execution no longer significantly improve uniprocessor performance [11] and it therefore became inevitable to take another path. It finally became inevitable for the semiconductor manufacturers to push forward the multi-core approach [14].

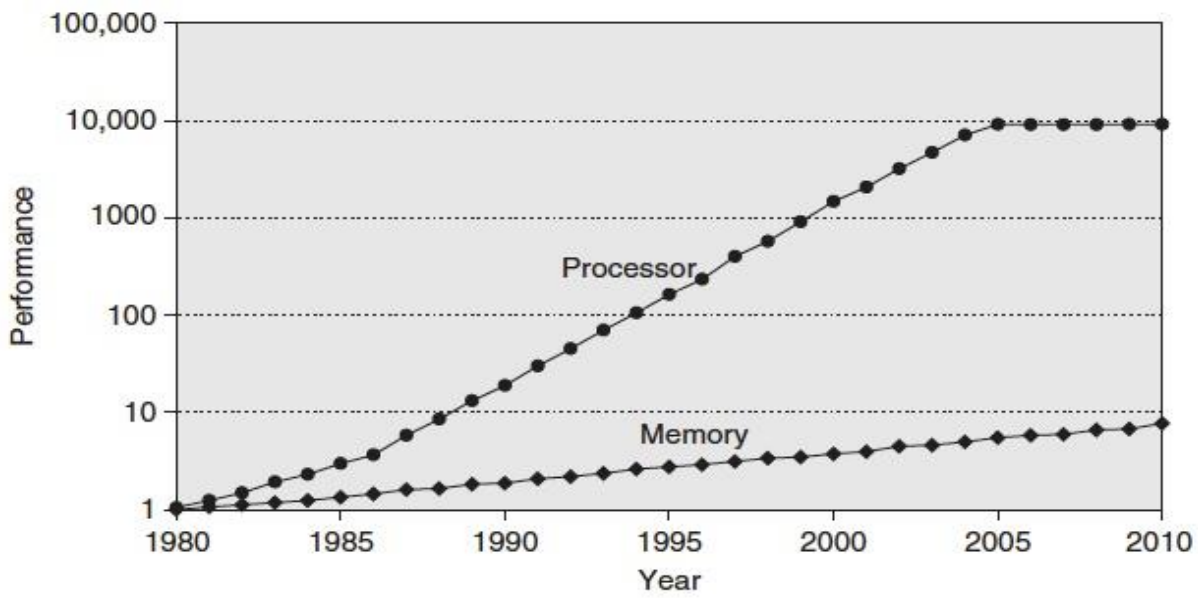


Figure 2: Processor and memory gap [8].

In general, the aim of multicore computing has been to achieve higher system performance through computational parallelism. In fact, Wilkinson described the multicore processors as multiple processors integrated into a single die to gain higher overall performance [14]. The multicore processors made it possible to overcome the power dissipation problem by for instance doubling the cores in a single chip to execute more instructions per cycle at a lower clock frequency [16]. This new model has also reduced the memory versus the CPU speed constraints.

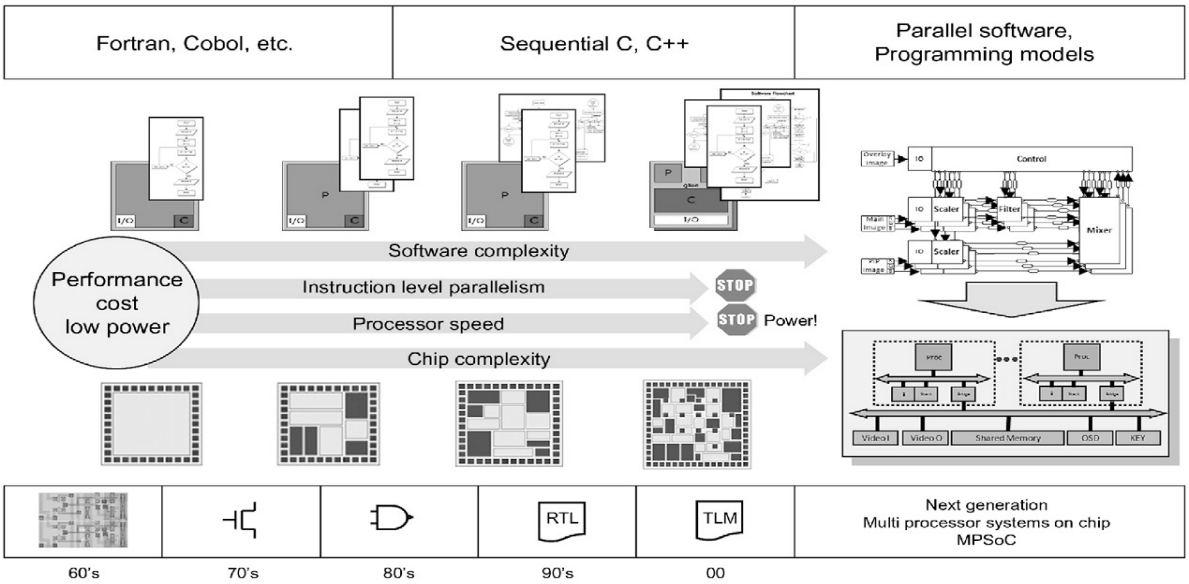


Figure 3: processor evolution: from layout to multicore [12].

The performance improvement is no longer gained through single-core speed increase but by adding multiple cores while keeping the clock frequency relatively constant. The image in Figure 3 summarises the computer/embedded systems' both software and hardware's evolution starting from the layout oriented to the current multicore oriented approach [12].

2.1.2 Memory Hierarchies

As described above the speed gap between the processor and the main memory known as memory wall has been one of the obstacles to continue increasing the single-core processor speed. To reduce this gap, multiple-levels of caches have been inserted between the CPU and the main memory [17] (see Figure 4).

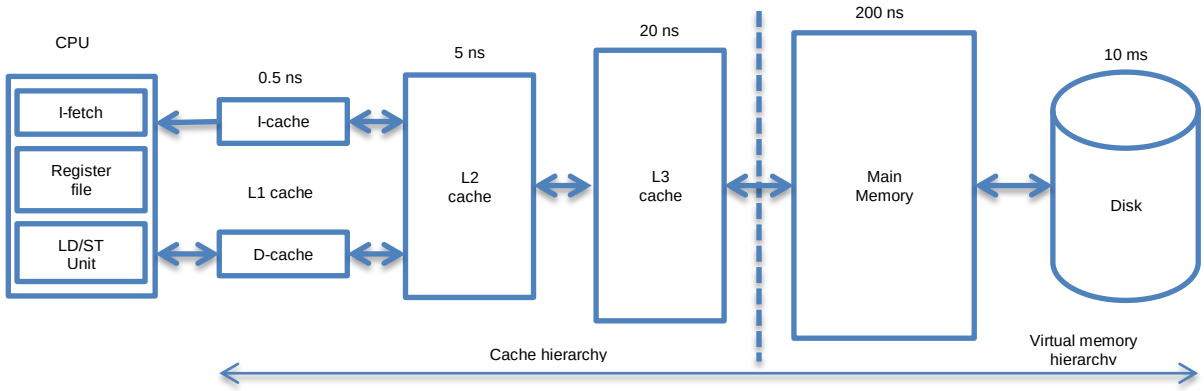


Figure 4: memory hierarchy [17].

Figure 4 illustrates a typical multilevel memory hierarchy that is implemented in generic processors as well as embedded systems. The figure also shows that the levels closer to the CPU have shorter access time while the size of the memory gets smaller. Figure 5 shows the comparison between the speed, cost per bit and the size of the memory levels.

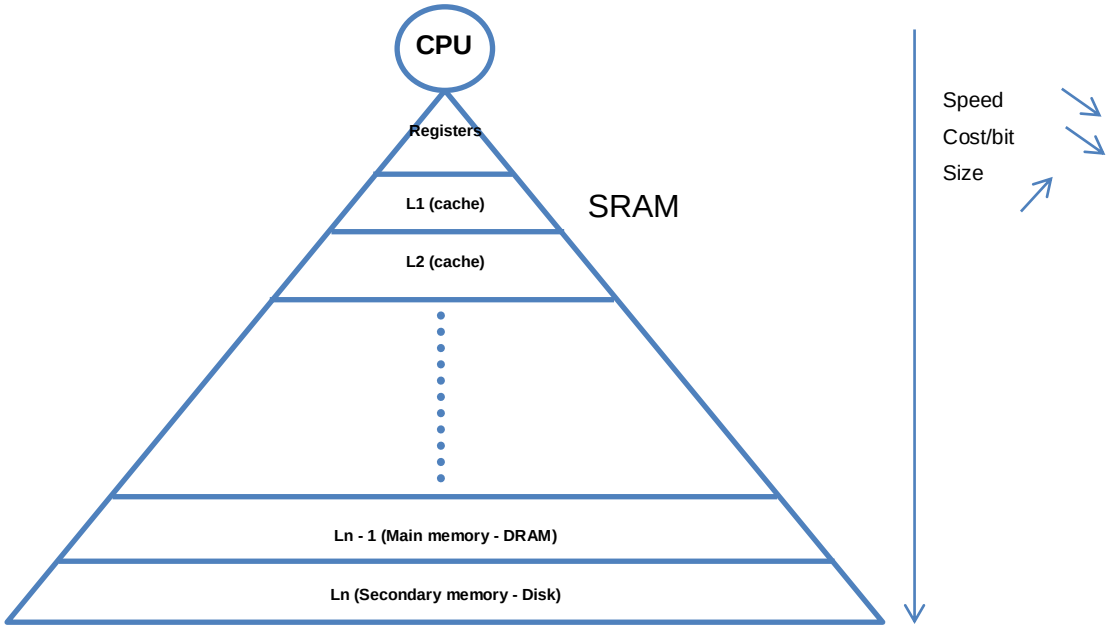


Figure 5: memory hierarchy's speed vs. size and cost/bit [17].

As the above figure shows L1 and L2 caches are built from fast SRAM cells that are more expensive than the DRAM cells that are used to build the main memory and sometimes the L3, L4 and etc. cache levels [17].

Let's also look into common multicore processors' memory architectures:

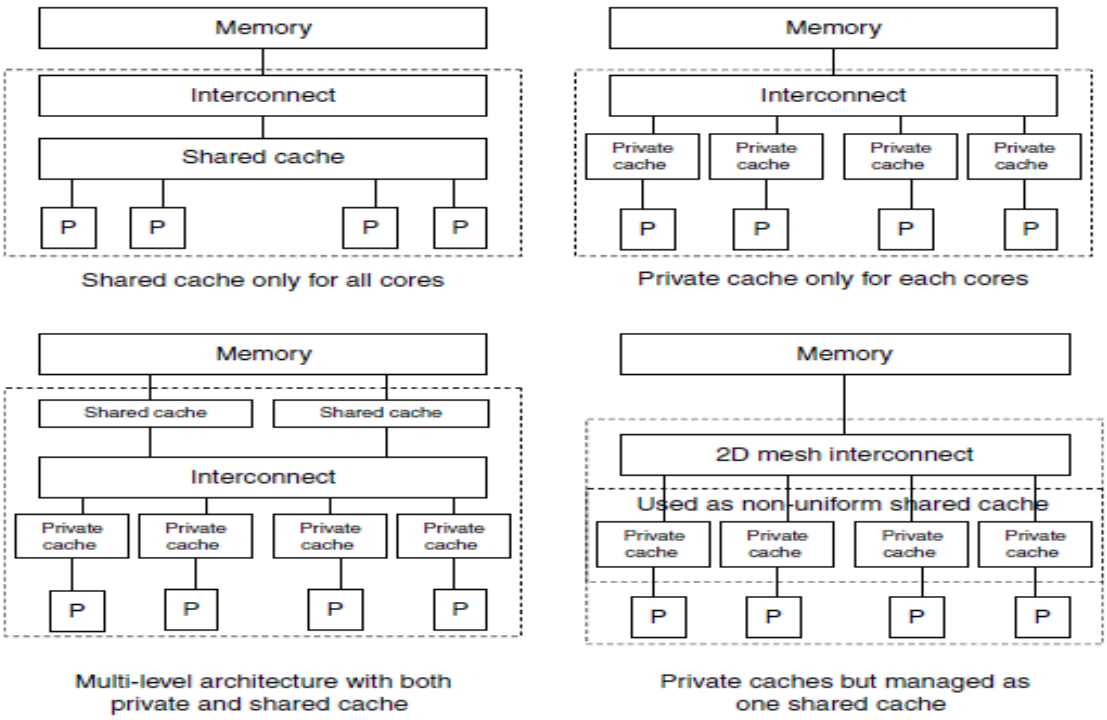


Figure 6: multicore memory architecture examples [18].

Since, multicore systems usually have cores that are running in parallel the memory bandwidth and memory access latency are critical. Consequently, the cache structure, shared or private for each core, needs to be chosen with extra care [19]. [Jain] also emphasises in [19] the importance of the choice of coherence protocol that matches the cache architecture characteristics. Coherence mechanisms are commonly divided into directory-based and snooping protocols.

2.1.3 The Locality Principle and Cache Parameters

The desktop or general-purpose computer systems are designed to run different types of applications that are not predefined. Therefore their memory systems are also designed to be as generic as possible [19]. On the other hand, the embedded systems' memory, particularly the cache architecture, can be customised since they run predefined applications that their memory access patterns can be established in advance [19]. Kumar et al. has also stated in [20] that designing a memory is a critical part of the SoC since the memory organisation has direct impact on the system parameters such as the area, power and performance. These constraints are even more limited in embedded systems than i.e. desktop. Therefore, choosing an optimal cache architecture for a given application is a significant step to boosting the overall system performance.

Let us not forget that cache memory was mainly invented to mask the speed gap between the processor and the main memory by exploiting the instruction/data spatial and temporal locality [20]. The temporal locality (a.k.a. locality of reference) assumes that the latest accessed memory address is soon to be accessed. The spatial locality on the other hand assumes that the addresses close to the latest accessed memory location are likely to be accessed next. Below is a list of some cache parameters a designer can choose to optimise an embedded system for a specific application (more details about these parameters can be found in most textbooks on computer architecture).

Cache Design Parameters [21], [17] and [8]:

1. **Cache mapping policies**:
   - **Direct Mapping**: a given memory block always appears on the same cache line. It is fast but the mapping restriction increases conflict misses.
   - **Fully associative Mapping**: any given memory block can be mapped to any cache line. It is a flexible technique that reduces conflict misses but it is expensive and slower since a parallel search has to be made on all cache lines.
   - **Set-Associative Mapping**: a memory block is mapped to a fixed cache set, but this block can be located in any line of the set. This technique is a trade-off between the fast direct-mapping and the flexible full associative.
2. **Write Policy:**
   - **Write through:** all writes/stores are simultaneously done on the cache and the lower memory levels. This may lead to bottlenecks since the communication between the cache and the memory is frequent.
   - **Write back:** writes/stores are done to the cache but when a cache block is replaced it is written back to the main memory. This may cause inconsistency between the cache and memory contents.
3. **Replacement function:**
   - **Least Recently Used (LRU):** records when each block is used and replaces the longest time ago used block on miss. It is in general costly to implement.
   - **Pseudo-LRU:** Approximates LRU by lower cost approximations.
   - **First-In-First-Out (FIFO):** Replacing the oldest block on a miss.
   - **Least Frequently Used (LFU):** replaces the least used block on miss.
   - **Random:** Replacing the blocks randomly.

4. **Size:**
    - **Cache size** - Large cache size means higher hit rate. On the other hand, larger size caches are more expensive and slightly slower than smaller caches due to the larger number of control gates.
5. **Number of Caches:**
    - **Multilevel Caches:** Hierarchies of cache from L1 to Ln
    - **Unified versus Split Caches:** Unified cache is instructions and data shared L1 cache. Split cache is 2 L1 caches where one is for instructions and the other for data.
6. **Memory Block (Line) Sizes:**
    - Large memory block size increase hit ratio. However, this effect is diminishes when certain point is reached since more cache data has to be replaced.

Metrics such as miss-rate (MR), hit-rate (HR) and misses per instruction (MPI) are used to measure the performance of a certain cache configuration. Where the miss-rate is the number of misses per total cache access and the hit-rate is the number of cache hits per total accesses. However, two metrics that gives better overall cache performance measurement are the average memory access time (AMAT) and the impact of a miss on the CPI [17].

2.2 WCET

According to Lundqvist in [22] "*the worst-case execution time (WCET) of a program that runs uninterrupted on a processor is defined as being the maximum possible execution time considering all combinations of input data and all possible execution histories of the system before the program is executed*".

Then again, why would one want to know the WCET? The most obvious answer to this question would be that if a task executes longer than expected it would miss its deadline. The consequence of a task missing its deadline can be quite serious [23] for example a task that responds few seconds later to a car-brake request.

Safety-critical embedded systems usually run tasks that have hard real-time structures [24]. It is therefore crucial to generate a safe and tight estimate of the WCET of the application [25] to provide the basis for the schedulability analysis of its real-time tasks [23].

The problem with finding the WCET is that it is not always easy to obtain upper bounds of execution times for a program [Wilhelm et al.]. Nevertheless, as stated by Wilhelm; et al. in [26] real-time systems only applies a restricted programming methods that guarantee program termination, prohibits recursion and loop iteration counts unless they are explicitly bounded.

There are two different classes of the WCET estimation methods namely Static methods and Measurement-based methods [26].

1. Static methods do not run the application code on hardware or on a simulator but analyse the task code's possible control flow paths with possibly some annotations. In addition, to obtain upper bounds control flow is combined with some abstract hardware architecture model.

2. Measurement-based methods run a task or part of it on a certain hardware or simulator for a combination of inputs. The maximum and minimum observed execution is then driven from the measurement.

Mälardalen University in Sweden has during 2006, 2008 and 2011 hosted an event called WCET Tool Challenge. The aim of this event has been to study, compare and discuss the properties of the different WCET tools and methods. An important outcome of this event is the list of the participating tools that is updated each time the event is held [27]. This is the biggest collection of the WCET tools that could be found. Details of these tools, which consist of commercial and research tool, can be been in Table 1.

Table 1: WCET Tools [27].

	Tool	Source	Contact	Processors	Analysis method	Analysis type
1	aiT	AbsInt	Simon Wegener	simple (ARM7, ST10), complex (MPC55xx)	static	flow analysis, WCET analysis
2	Astrée	AbsInt	Simon Wegener	source-level tool (C)	static	flow analysis
3	Bound-T	Tidorum	Niklas Holsti	simple (ARM7)	static	control-flow analysis, WCET analysis
4	FORTAS	TU Vienna	Sven Bünte	TriCore 1796	measurement-based	WCET estimation
5	METAMOC	Aalborg Univ.	Mads Chr. Olesen	simple (ARM7)	static	WCET analysis
6	oRange+OTAWA	IRIT	Christine Rochange	simple (ARM7)	static	control-flow analysis, WCET analysis
7	TimeWeaver	AbsInt	Simon Wegener	with NEXUS-like tracing facilities	measurement based	WCET estimation
8	TuBound	TU Vienna	Jakob Zwirchmayr	simple (Infineon C167)	static	WCET analysis
9	WCA	TU Vienna, TU Denmark	Martin Schoeberl	JOP (Java processor)	static	simple loop bound and receiver analysis (DFA), WCET analysis
10	SWEET	WCET group, Mälardalen University, Sweden	Jan Gustafsson	source-level tool (C)	static	flow analysis

3. System Assumptions

The simulated processor (LEON4), the simulated tool (gem5), the simulation model overview, and the applications ported onto the simulation tool to estimate the WCET are described in this chapter.

3.1 LEON4

3.1.1 Background

Aeroflex Gaisler AB, a Swedish company that develops and supports embedded system solutions, has been chosen by European Space Agency (ESA) to develop the Next Generation Microprocessor (NGMP) [28]. Aeroflex Gaisler has therefore since completing the LEON3 core processor in 2006 started to develop LEON4FT (LEON4), a fault-tolerant quad-core SoC. Although the current prototype implementation of the LEON4 is based on FPGA that has a system frequency of 150MHz [29] [30], its final product is expected to run at 1.5GHz using 32nm ASIC technology [30] (see Table 2).

Table 2: Proposed LEON4 technology implementations [31].

Technology	MHz	Area
32 ns ASIC	1500 MHz	30 kgates
45 ns ASIC	1200 MHz	30 kgates
65 ns ASIC	800 MHz	30 kgates
130 ns ASIC	400 MHz	30 kgates
180 ns ASIC	250 MHz	30 kgates
Stratix3 FPGA	150 MHz	4000 LUT
Vertex5 FPGA	125 MHz	4000 LUT

Although, Aeroflex Gaisler developed LEON4 in cooperation with ESA, its licence is also available to other customers. RUAG Space AB, a Sweden based company that is specialised in on-board satellite equipment is one of the companies that is currently using this SoC.

RUAG Space AB (RUAG) is considering utilising the 4 LEON4 cores concurrently. However, an important requirement is to establish safe and tight WCET estimate to avoid unexpected consequences. Hence, RUAG has together with the Computer Science and Engineering (CSE) department at Chalmers University of technology initiated this master thesis project. The goal of this project has been to estimate the WCET of a multicore processor where the target system is the quad-core LEON4, although the method is generally applicable to any multicore system with a multi-level cache memory hierarchy.

3.1.1 The Quad-core LEON4 Architecture

The LEON4 core is a 32-bit processor based on SPARC V8 architecture. It is a highly configurable VHDL model that is particularly suitable for SoC designs on FPGA and ASIC.

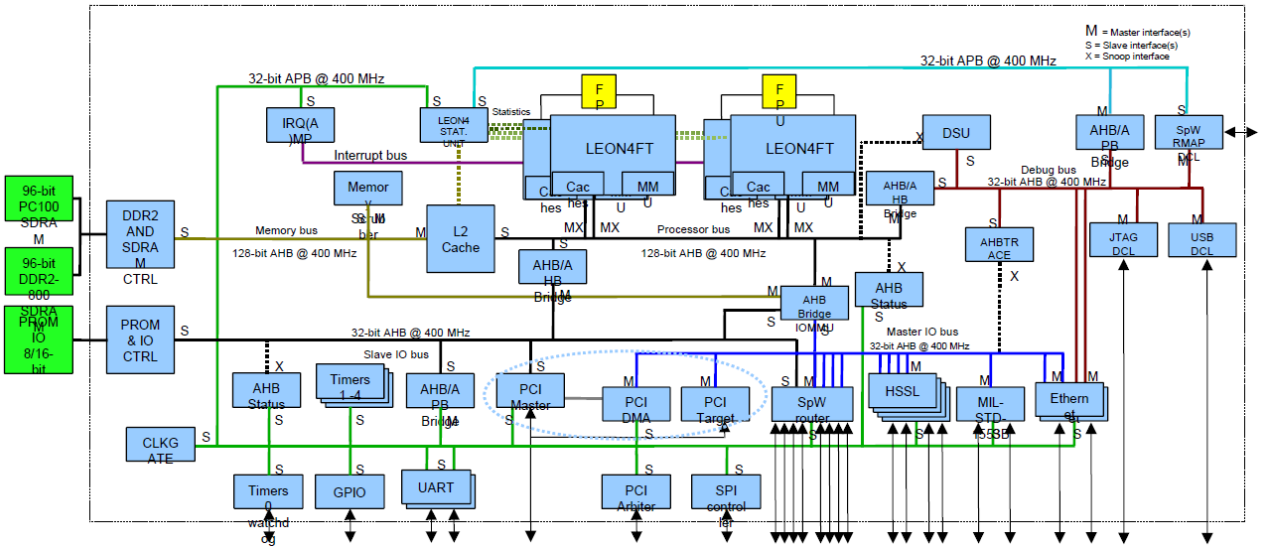


Figure 7: Quad-core LEON4FT Architecture overview.

As can be seen in Figure 7, each pair of the quad-cores shares one FPU unit. In addition, each core has a private split L1 cache, where the L2 cache is shared by all 4 cores. The architecture overview is also showing a 128-bit AHB bus connecting the memory levels to the cores as well as between the cores. The peripherals on the other hand are connected to 32-bit AHB bus. Figure 8 on the other hand is also displaying more details about the LEON4 core architecture.

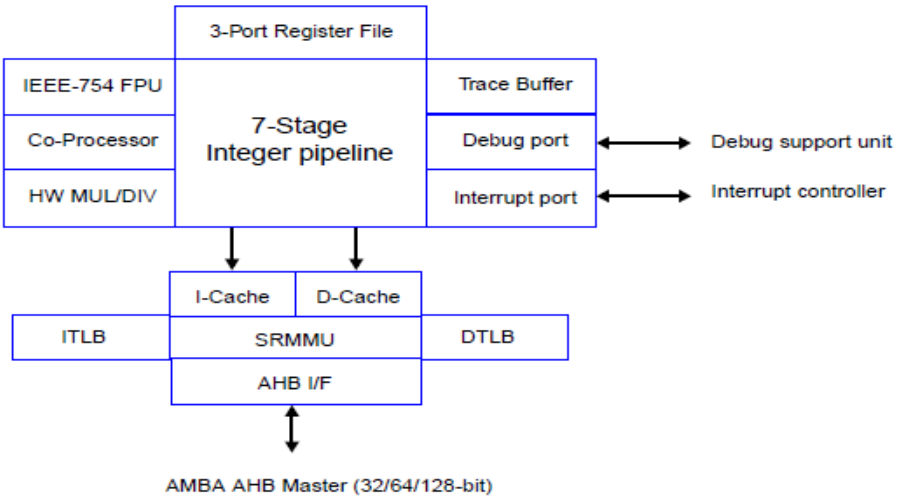


Figure 8: LEON4 core block diagrams.

The quad-core leon4 details that are useful for the simulation tool are the system clock frequency, and the cache configuration, levels and sizes. These configuration parameters are collected from the latest LEON4 implementation on FPGA [30] (see Section 3.2).

3.2 Simulation Model

gem5, which is a state-of-the-art modular platform for computer system architecture research [32], is this project's simulation tool. The choice of this simulator was based on the project criteria and the CSE department's subject experts' advice. Furthermore, it is widely used by the computer architecture research community around the world. The main requirements of the simulator were:

- It can be configured to a desired architecture (as LEON4 in this case)
- It supports multilevel memory hierarchies
- It can run C/C++ based applications
- It can isolate a region of interest in the application's code to extract this region's simulation result separately

Good news is that the selected simulator, gem5, has more than above described requirements. Its system-level architecture as well as processor microarchitecture can be configured to support many different systems. It currently supports Alpha, ARM, SPARC, MIPS, POWER and x86 ISA [33]. It even has the option to configure multicores. Nevertheless, the last option was not applied, since its complexity would require extra time and resources that perhaps would not be proportionate to the project duration. Therefore, an analytical model (see Table 4) is added to the simulation model to get the assumed multicore system result.

This analytical model, offered by professor Per Stenstrom at the CSE department, calculates the WCET estimate and guarantees safe and tight WCET estimates. To guarantee a safe estimate, the model uses a worst-case assumption for estimating the cache miss penalty, i.e., the time it takes to service a cache miss. This worst-case assumption is that when a cache miss is serviced, there will be P-1 cores that have cache misses pending and these will be serviced first. As a result, assuming that the cache miss penalty is MP with no pending requests from other cores, it will take P x MP to service the cache miss. To get a tight WCET estimate, we ideally assume that the compiler can statically analyse whether a cache request will hit or miss the cache. This will provide an upper-bound on how well static analysis could be used to derive a tight estimate for the number of cache hits.

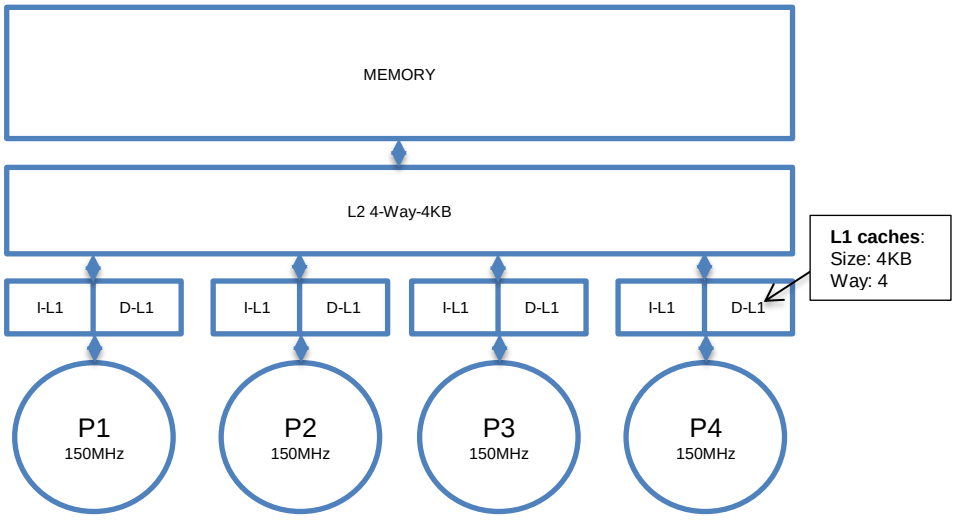


Figure 9: Quad-core LEON4 simulation setup.

Figure 9 shows the simulated quad-core LEON4 and its memory levels. Also, Table 3 displays the simulators key settings such as the clock frequency and the cache size. The simulation stats such as cache miss-penalties (MP) as well as constants like the CPU cycle time (Tc) and the number of cores (P) are also listed in Table 3. The equations listed in Table 4, use the aforementioned values to calculate the WCET estimate of the quad-cores when they execute the 4 tasks serially and 4 in parallel. Moreover, the parallel tasks' execution throughput (K), relative to the serial execution throughput, is calculated. It is assumed that, when 4 cores are running 4 tasks in parallel the closer K gets to 4 the safer and tighter the WCET estimate gets.

Table 3: Simulation parameters and Cache latencies scaled for future technology nodes.

Parameter	Value (FPGA*)	Value (ASIC**)	Value (ASIC**)	Value (ASIC**)	Unit
CPU_Freq =	150	300	600	1200	MHz
CPU_Cycles =	6,67	3,33	1,67	0,83	ns
CPU_Type =	timing	timing	timing	timing	-
L1_Size =	4	8	16	32	kB
L1_Assoc =	4	4	4	4	-
L2_size =	256	256	256	256	kB
L2_assoc =	4	4	4	4	-
P =	4, 8, 16	4, 8, 16	4, 8, 16	4, 8, 16	CPUs
L1_MP	40	20	10	5	ns
L2_MP	200	100	50	25	ns
** Current Impementation ** assumed future technology node upgrade*					

Table 4: calculating the AMAT, the Tproc, the Texe and the throughput.

AMAT = MP-L1*L1-Miss-rate + MP-L2*L2-Miss-rate	Average memory access time
Tproc=Instructions*Tc	The total time the CPU is busy to decode instr. per task
Texe-1 = P(Tproc+AMAT)	The WCET for executing 1 task at a time
Texe-4 = Tproc+P*AMAT	The WCET for executing 4 tasks in parallel
K = Texe-1 / Texe-4	4 parallel tasks execution Throughput

In order to speed up the simulation a sort of automation mechanism was needed. Therefore two bash scripts were created. These two scripts were designed to initiate the compiler and the simulator commander, simplify the simulation data collection and restructuring it into a desired format. Figure 10 illustrates the complete simulation model that includes the two scripts, the simulation tool and the analytical equations.

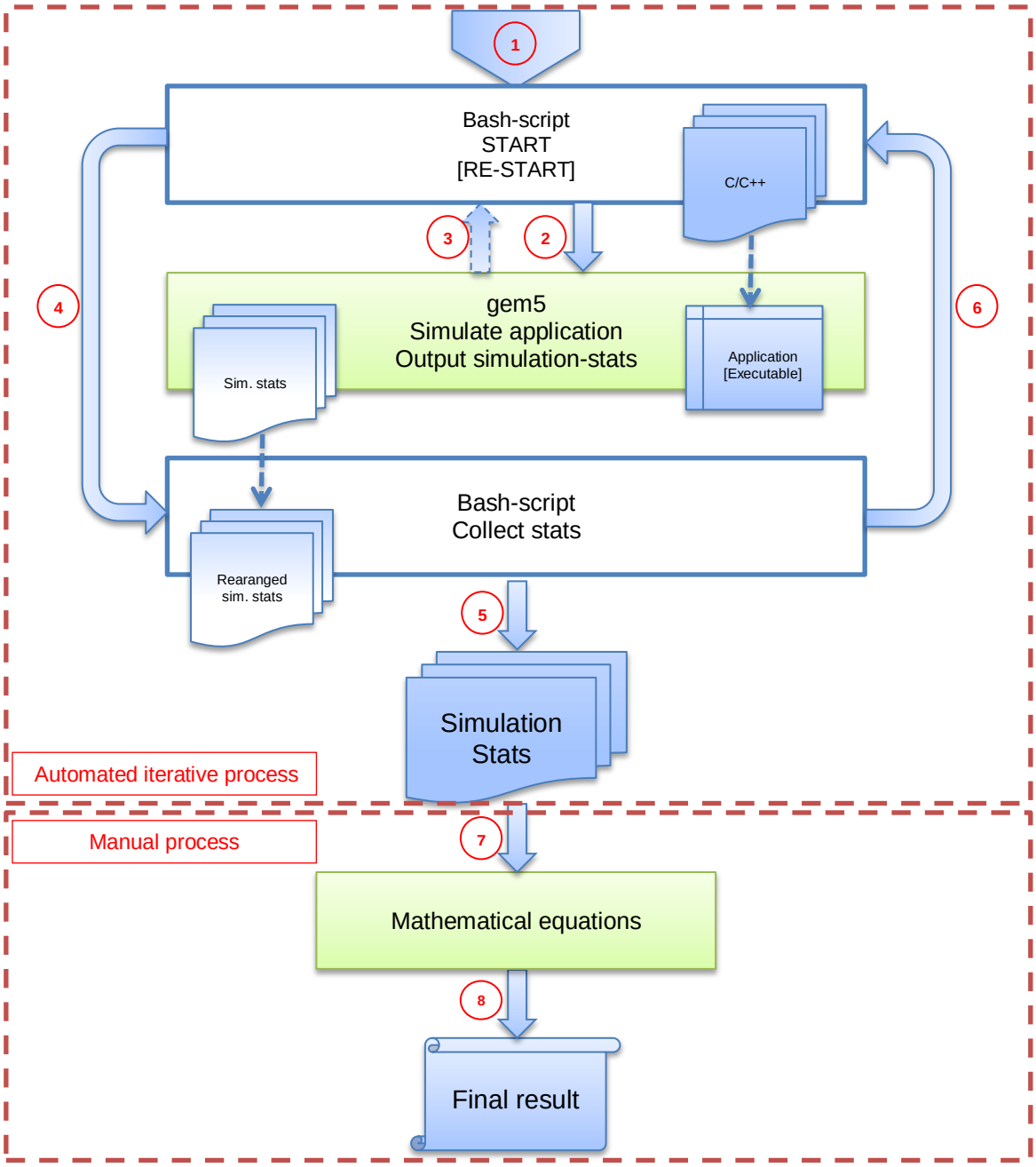


Figure 10: the simulation model overview.

The diagram according to Figure 10 shows an overview of the simulation model and numbered labels highlight the order of the main simulation steps. One can also see that the simulation steps are split by two dotted boxes. This is to show that the simulation is divided into two phases. The first phase is an automated process starting from step 1 to step 5 then through step 6 feeds back to step 1. This iteration is carried out a number of times depending on the number of required simulation samples per application. During these iterations either the application or the simulation tool parameters are incremented. For example if an application is to be simulated with different cache sizes then the simulation tool's cache size parameters are incremented while the application's input parameter(s) is/are kept unchanged. This project's application workload is a matrix multiplication. Therefore, by increasing the matrices' sizes, the application workload increases. Below is a summary of the simulation steps.

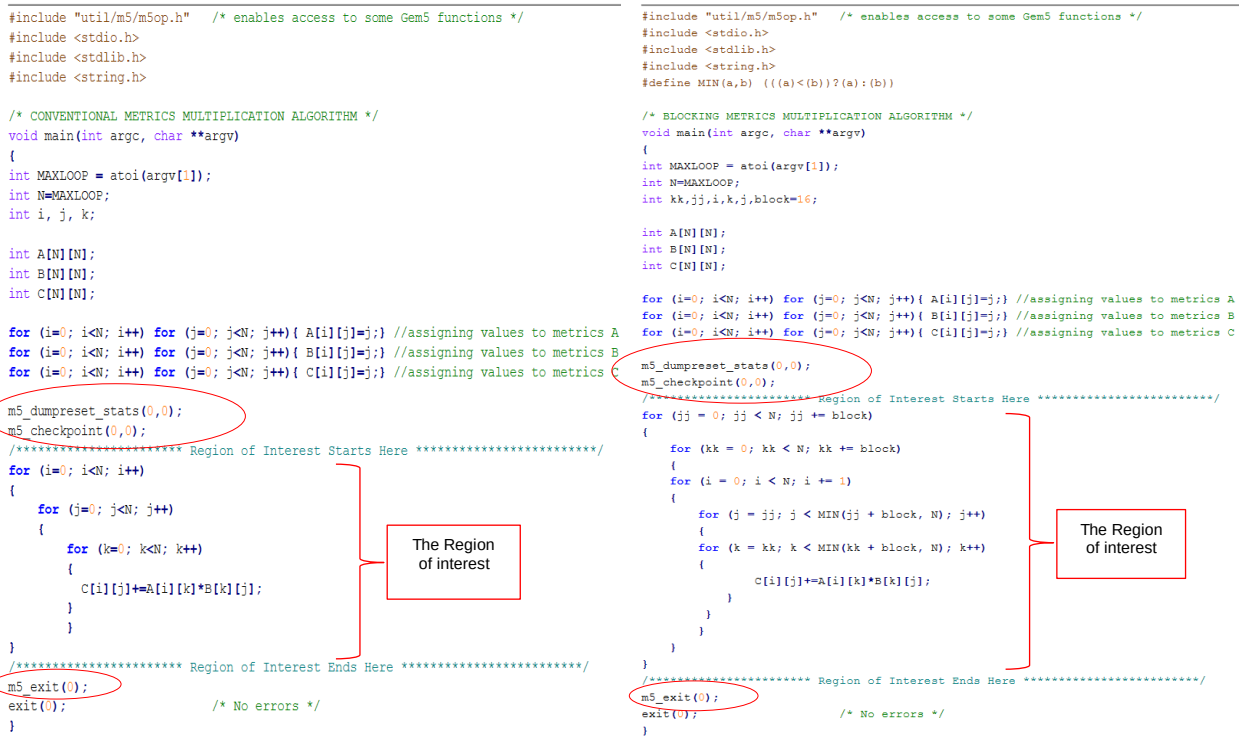
The simulation steps:

1. The first bash script of the simulation model is initiated through the terminal command by entering its name and the C file of the application be simulated [./runsim matrix]
   1.1. The start script calls then a predefined makefile that compiles the C code using GCC with -2 optimisation option
   1.2. Once the executable file is created the script calls then gem5 by using terminal command format. All the necessary settings, parameters and variables such as CPU type and speed, cache levels, cache size etc. and of course the name and the path of the application under simulation are passed through this commands string.
2. gem5 simulation
   2.1. gem5 applies the new settings and starts the simulation by running the ported application
   2.2. Once the simulation is finished, gem5 saves the simulation stats in text file
3. The start script gets reactivated again.
4. The first script calls then the second script
5. The second script
   5.1. It starts to search for the latest saved gem5 simulation stats to extract certain results.
   5.2. It then rearranges the extracted data into Microsoft Excel and Matlab friendly format and saves it into a new text file.
6. This step loops back to the first script which increments some simulation tool or application parameters as was described above. Steps from 1 to 6 are repeated number of times, depending on the required number of simulation samples.
7. The WCET and the throughput calculation
   7.1. Once, the automated steps from 1 to 6 are finished, the rearranged simulation stats file is then opened with Excel.
   7.2. Then the equations shown in Table 4 are applied to calculate the WCET and the parallel tasks' throughput.
   7.3. The final result is then compiled into lists and diagram that will be presented in the result chapter as well as in the discussion chapter.

3.3 Applications

Finding a simulation tool that simplifies the WCET estimate was the first step of the execution phase of the project. Then creating an application that would run on the simulation software became the next challenge. Essentially, an application that utilises all system resources, particularly the multicore and the memory hierarchies, was the criterion. It should simply be a data and instruction intensive application. Moreover, since gem5, the simulation software, can only execute applications written in C/C++, C was chosen as the programming language.

After considering above requirements and number of previous research that was discussed in the theory chapter, two applications carrying out matrix multiplication has been created. One that is doing conventional matrix multiplication and another one that is doing blocking (a.k.a. tiling) matrix multiplication, which is a cache optimisation algorithm [4]. Figure 11(a) shows the conventional matrix multiplication C code while Figure 11(b) shows the blocking algorithm code.



(a) (b)

Figure 11: a) Conventional and b) blocking metrics multiplication C code.

The middle parts of the above codes are labelled as the region of interest (ROI) to indicate that these lines are where the interested workload is carried out. By inserting the gem5 functions *m5_dumpreset_stats*, *m5_checkpoint* and *m5_exet* before and after the ROI, gem5 returns a simulation result for only this region.

Another, important fact about the matrix multiplication applications is that they both have one input parameter. This input parameter which is passed into the application at the start of simulation dictates the size of the matrix for example if it is equal to 32 then the A, B and C matrices sizes becomes 32x32. As was stated before this is used to increase or decrease the simulation workload and to observe how it influenced the WCET estimate.

More details about the gem5 commands, bash scripts and the application can be read in the appendices.

4. Experimental Results

This chapter outlines the simulation and the calculation results of this Master's Thesis. The two top categories of the result are the two matrix multiplication algorithms, namely the conventional and the blocking algorithm. Equally important is to mention that the subcategories of the result are system clock frequency and the number of processors.

Furthermore, the data is presented in the same order as the simulation model (see Figure 10). This means that the simulation result collected from the automated simulation will be presented first. Then the calculated result from the manual stage of the simulation will be presented.

As was stated in Section 3, a single core execution was carried out in the first stage of the simulation. Then this stage's outcome is used to calculate a multicore system WCET estimate by using the equations presented in Table 4. The multicore WCET estimate is calculated for both serial and parallel execution of the applications.

4.1 Simulation Results

Again, according to Section 3, the conventional matrix multiplication application is first simulated then the blocking algorithm application. These two applications are repeated for four different system clock frequencies: 150MHz, 300MHz, 600MHz and 1200MHz. In addition, the caches are reconfigured for each clock frequency to match the system for an anticipated technology scaling. Each time the simulation is carried out for certain algorithm using a particular system clock frequency (for example: blocking algorithm running at 150MHz), 18 result samples are collected. Each sample represents simulation result for a certain algorithm, clock frequency and matrix size.

Table 5 shows the conventional and the blocking matrix multiplication algorithms' simulation result. The columns of the table, which are the matrix sizes (NxN_Matrix), the number of application instructions (N_Instr), L1 miss-rate (L1_MR) and the L1 miss-rate (L2_MR), are the parameters necessary for the WCET calculation. Moreover, the results in Table 5 are based on a system Clock=150MHz, L1_Size=4KB, L1_Assoc=4, L2_size=256kB and L2_assoc=4.

Table 5: Conventional and blocking matrix multiplication result for a system freq. 150MHz.

Conventional Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR
32x32	303303	0,114759	0,024884
48x48	1014055	0,075524	0,025391
64x64	2392455	0,50856	0,00284
80x80	4659687	0,537189	0,002156
96x96	8036935	0,517409	0,001869
112x112	12745383	0,535634	0,00155
128x128	19006215	0,504027	0,001442
144x144	27040615	0,534674	0,00121
160x160	37069767	0,51709	0,001127
176x176	49314855	0,534058	0,000992
192x192	63997063	0,508409	0,000956
208x208	81337575	0,53363	0,000841
224x224	101557575	0,516951	0,001007
240x240	124878247	0,533315	0,002047
256x256	151520775	0,501922	0,025146
272x272	181706343	0,533074	0,035306
288x288	215656135	0,516972	0,060681
304x304	253591335	0,532883	0,058856

Blocking Algorithm Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR
32x32	354104	0,008273	0,32879
48x48	1195019	0,007811	0,231551
64x64	2832558	0,148204	0,009145
80x80	5532257	0,007576	0,143044
96x96	9559652	0,007605	0,118726
112x112	15180279	0,007475	0,10352
128x128	22659674	0,495896	0,001365
144x144	32263373	0,007419	0,081366
160x160	44256912	0,007464	0,074505
176x176	58905827	0,007383	0,080645
192x192	76475654	0,171363	0,002979
208x208	97231929	0,007358	0,070642
224x224	121440188	0,007405	0,102349
240x240	149365967	0,007339	0,170298
256x256	181274802	0,494452	0,004791
272x272	217432229	0,007439	0,257628
288x288	258103784	0,007376	0,273407
304x304	303555003	0,007314	0,274105

Table 6 displays the conventional and blocking algorithm simulation result for a system frequency at 300MHz. In addition, these results are based on cache configurations were L1_Size=8KB, L1_Assoc=4, L2_size=256kB and L2_assoc=4.

Table 6: Conventional and blocking matrix multiplication result for a system freq. 300MHz.

Conventional Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR
32x32	303303	0,002929	0,974747
48x48	1014055	0,021949	0,087369
64x64	2392455	0,50841	0,002841
80x80	4659687	0,044128	0,02625
96x96	8036935	0,516922	0,001871
112x112	12745383	0,121472	0,006833
128x128	19006215	0,503974	0,001442
144x144	27040615	0,386968	0,001672
160x160	37069767	0,516795	0,001127
176x176	49314855	0,534058	0,000992
192x192	63997063	0,508341	0,000956
208x208	81337575	0,53363	0,000841
224x224	101557575	0,51674	0,001008
240x240	124878247	0,533315	0,002047
256x256	151520775	0,501922	0,025153
272x272	181706343	0,533074	0,035306
288x288	215656135	0,516709	0,060711
304x304	253591335	0,532883	0,058856

Blocking Algorithm Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR
32x32	354104	0,005863	0,463942
48x48	1195019	0,007811	0,231551
64x64	2832558	0,007707	0,175852
80x80	5532257	0,007576	0,143044
96x96	9559652	0,007517	0,120111
112x112	15180279	0,007475	0,10352
128x128	22659674	0,134349	0,00504
144x144	32263373	0,007419	0,081366
160x160	44256912	0,007399	0,075163
176x176	58905827	0,007383	0,081437
192x192	76475654	0,007391	0,068583
208x208	97231929	0,007358	0,070642
224x224	121440188	0,007348	0,103078
240x240	149365967	0,007339	0,170298
256x256	181274802	0,493437	0,004801
272x272	217432229	0,007325	0,261619
288x288	258103784	0,007319	0,275507
304x304	303555003	0,007314	0,274105

Table 7 displays the conventional and blocking algorithm simulation result for a system frequency at 600MHz. Also, cache configurations of these results are L1_Size=16KB, L1_Assoc=4, L2_size=256kB and L2_assoc=4.

Table 7: Conventional and blocking matrix multiplication result for a system freq. 600MHz.

Conventional Matrix Multiplication				Blocking Algorithm Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR	NxN_Matrix	N_Instr	L1_MR	L2_MR
32x32	303303	0,002855	1	32x32	354104	0,00272	1
48x48	1014055	0,001922	0,997696	48x48	1195019	0,003362	0,537888
64x64	2392455	0,054315	0,026589	64x64	2832558	0,006069	0,223287
80x80	4659687	0,03179	0,036438	80x80	5532257	0,007257	0,149322
96x96	8036935	0,04938	0,019584	96x96	9559652	0,007517	0,120111
112x112	12745383	0,031605	0,026261	112x112	15180279	0,007448	0,103899
128x128	19006215	0,503936	0,001443	128x128	22659674	0,007454	0,090828
144x144	27040615	0,031513	0,020527	144x144	32263373	0,007419	0,081366
160x160	37069767	0,516135	0,001129	160x160	44256912	0,007399	0,075163
176x176	49314855	0,031601	0,016766	176x176	58905827	0,007383	0,081437
192x192	63997063	0,508219	0,000956	192x192	76475654	0,007369	0,068769
208x208	81337575	0,071637	0,006263	208x208	97231929	0,007358	0,070642
224x224	101557575	0,516264	0,001008	224x224	121440188	0,007348	0,104057
240x240	124878247	0,15126	0,007569	240x240	149365967	0,007339	0,170443
256x256	151520775	0,501909	0,025127	256x256	181274802	0,128265	0,018434
272x272	181706343	0,296399	0,063492	272x272	217432229	0,007325	0,261955
288x288	215656135	0,516337	0,060755	288x288	258103784	0,007319	0,275507
304x304	253591335	0,465541	0,067369	304x304	303555003	0,007314	0,274105

Table 8 is displays the conventional and blocking algorithm simulation result for system frequency at 1200MHz. Cache configurations used are L1_Size=32KB, L1_Assoc=4, L2_size=256kB and L2_assoc=4.

Table 8: Conventional and blocking matrix multiplication result for a system freq. 1200MHz.

Conventional Matrix Multiplication				Blocking Algorithm Matrix Multiplication			
NxN_Matrix	N_Instr	L1_MR	L2_MR	NxN_Matrix	N_Instr	L1_MR	L2_MR
32	303303	0,00284	1	32	354104	0,00272	1
48	1014055	0,001913	1	48	1195019	0,001808	1
64	2392455	0,001451	0,993532	64	2832558	0,002881	0,470336
80	4659687	0,001746	0,662617	80	5532257	0,002651	0,408781
96	8036935	0,019548	0,049441	96	9559652	0,002868	0,314822
112	12745383	0,031605	0,02625	112	15180279	0,003355	0,230596
128	19006215	0,503847	0,001442	128	22659674	0,007443	0,090958
144	27040615	0,031513	0,020522	144	32263373	0,005682	0,106233
160	37069767	0,035904	0,016221	160	44256912	0,006647	0,083663
176	49314855	0,031458	0,01684	176	58905827	0,007208	0,084252
192	63997063	0,507949	0,000956	192	76475654	0,007369	0,068769
208	81337575	0,031422	0,014277	208	97231929	0,007358	0,070642
224	101557575	0,073961	0,006993	224	121440188	0,007348	0,10465
240	124878247	0,031396	0,029172	240	149365967	0,007339	0,171231
256	151520775	0,501899	0,02504	256	181274802	0,007334	0,268935
272	181706343	0,031377	0,592052	272	217432229	0,007325	0,262334
288	215656135	0,337485	0,092958	288	258103784	0,007319	0,275507
304	253591335	0,031404	0,999047	304	303555003	0,007314	0,274105

4.2 Calculated Results

After the simulation result (above) has been acquired, the parameters presented in Table 3 and the equations presented in Table 4 have been applied to calculate the final result. The average memory access time (AMAT) and the processor's instruction execution time (Tproc) are first calculated, and then these values together with the number of tasks that are equivalent to the number of processor cores are used to calculate the WCET estimate for the serial and the parallel executions (Texe_1 and Texe_N respectively, where N is the number of tasks/Cores) of the tasks.

Due to the large quantity of the calculated data, only calculated tables for the system clock frequency at 150MHz are presented in this section. The remaining tables are added into the appendices. Instead, charts illustrating the calculated throughput (K) for a number of multicore arrangements (P=4, P=8 and P=16) are presented in this section, since the throughput shows if the WCET estimate is safe and tight.

Table 9 presents the calculated result for 4 cores. Texe_1 is the calculated WCET estimate for 4 serially executed tasks. Texe_4 on the other hand is the WCET estimate for 4 tasks executed in parallel. Finally, K, which is the throughput of the 4 the tasks running in parallel, is also presented in the table (below).

Table 9: Calculated WCET and Throughput for a system freq. 150MHz and P=4.

Conventional/Blocking Matrix Multiplication										
NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32x32	9,57E-09	6,61E-08	0,0020220	0,0023607	0,0080881	0,0094430	0,0020221	0,0023610	3,9999432	3,9996641
48x48	8,10E-09	4,66E-08	0,0067604	0,0079668	0,0270415	0,0318674	0,0067604	0,0079670	3,9999856	3,9999298
64x64	2,09E-08	7,76E-09	0,0159497	0,0188837	0,0637989	0,0755349	0,0159498	0,0188838	3,9999843	3,9999951
80x80	2,19E-08	2,89E-08	0,0310646	0,0368817	0,1242584	0,1475270	0,0310647	0,0368818	3,9999915	3,9999906
96x96	2,11E-08	2,40E-08	0,0535796	0,0637310	0,2143184	0,2549242	0,0535797	0,0637311	3,9999953	3,9999955
112x112	2,17E-08	2,10E-08	0,0849692	0,1012019	0,3398770	0,4048075	0,0849693	0,1012019	3,9999969	3,9999975
128x128	2,04E-08	2,01E-08	0,1267081	0,1510645	0,5068325	0,6042581	0,1267082	0,1510646	3,9999981	3,9999984
144x144	2,16E-08	1,66E-08	0,1802708	0,2150892	0,7210832	0,8603567	0,1802709	0,2150892	3,9999986	3,9999991
160x160	2,09E-08	1,52E-08	0,2471318	0,2950461	0,9885272	1,1801844	0,2471319	0,2950461	3,9999990	3,9999994
176x176	2,16E-08	1,64E-08	0,3287657	0,3927055	1,3150629	1,5708221	0,3287658	0,3927056	3,9999992	3,9999995
192x192	2,05E-08	7,45E-09	0,4266471	0,5098377	1,7065884	2,0393508	0,4266472	0,5098377	3,9999994	3,9999998
208x208	2,15E-08	1,44E-08	0,5422505	0,6482129	2,1690021	2,5928515	0,5422506	0,6482129	3,9999995	3,9999997
224x224	2,09E-08	2,08E-08	0,6770505	0,8096013	2,7082021	3,2384051	0,6770506	0,8096013	3,9999996	3,9999997
240x240	2,17E-08	3,44E-08	0,8325216	0,9957731	3,3300867	3,9830926	0,8325217	0,9957733	3,9999997	3,9999996
256x256	2,51E-08	2,07E-08	1,0101385	1,2084987	4,0405541	4,8339948	1,0101386	1,2084988	3,9999997	3,9999998
272x272	2,84E-08	5,18E-08	1,2113756	1,4495482	4,8455026	5,7981930	1,2113757	1,4495484	3,9999997	3,9999996
288x288	3,28E-08	5,50E-08	1,4377076	1,7206919	5,7508304	6,8827678	1,4377077	1,7206921	3,9999997	3,9999996
304x304	3,31E-08	5,51E-08	1,6906089	2,0237000	6,7624357	8,0948003	1,6906090	2,0237002	3,9999998	3,9999997

Similar to Table 9, Tables 10 and 11 also show the calculated results for a system frequency of 150MHz, however both the processor cores and the executed tasks are increased to 8 and 16.

Overall, it is surprising to see that the throughput increases by close to linearly with the number of cores. This suggests that, contrary to common belief, it is possible to guarantee a safe WCET and still enjoy the increase in throughput offered by multicore systems. Of course, we have assumed that the compiler is capable of fully analysing the code statically. As Lundqvist points out [22], one has to conservatively assume a cache miss if the compiler cannot be certain of the outcome of a cache access. In any case, these results are promising and warrant further investigations.

Table 10: Calculated WCET and Throughput for a system freq. 150MHz and P=8

Conventional/Blocking Matrix Multiplication										
NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_8_Conv	Texe_8_Block	K_Conv	K_Block
32x32	9,56716E-09	6,60889E-08	0,0020220	0,0023607	0,0161762	0,0188861	0,0020221	0,0023612	7,9997350	7,9984326
48x48	8,09916E-09	4,66226E-08	0,0067604	0,0079668	0,0540830	0,0637347	0,0067604	0,0079672	7,9999329	7,9996723
64x64	2,09104E-08	7,75716E-09	0,0159497	0,0188837	0,1275978	0,1510698	0,0159499	0,0188838	7,9999266	7,9999770
80x80	2,19188E-08	2,89118E-08	0,0310646	0,0368817	0,2485168	0,2950539	0,0310648	0,0368819	7,9999605	7,9999561
96x96	2,10702E-08	2,40494E-08	0,0535796	0,0637310	0,4286367	0,5098483	0,0535797	0,0637312	7,9999780	7,9999789
112x112	2,17354E-08	2,1003E-08	0,0849692	0,1012019	0,6797539	0,8096150	0,0849694	0,1012020	7,9999857	7,9999884
128x128	2,04495E-08	2,01088E-08	0,1267081	0,1510645	1,0136650	1,2085161	0,1267083	0,1510647	7,9999910	7,9999925
144x144	2,1629E-08	1,657E-08	0,1802708	0,2150892	1,4421663	1,7207134	0,1802709	0,2150893	7,9999933	7,9999957
160x160	2,0909E-08	1,51996E-08	0,2471318	0,2950461	1,9770544	2,3603688	0,2471319	0,2950462	7,9999953	7,9999971
176x176	2,15607E-08	1,64243E-08	0,3287657	0,3927055	2,6301258	3,1416442	0,3287659	0,3927056	7,9999963	7,9999977
192x192	2,05276E-08	7,45032E-09	0,4266471	0,5098377	3,4131769	4,0787016	0,4266473	0,5098378	7,9999973	7,9999992
208x208	2,15134E-08	1,44227E-08	0,5422505	0,6482129	4,3380042	5,1857030	0,5422507	0,6482130	7,9999978	7,9999988
224x224	2,08794E-08	2,0766E-08	0,6770505	0,8096013	5,4164042	6,4768102	0,6770507	0,8096014	7,9999983	7,9999986
240x240	2,1742E-08	3,43532E-08	0,8325216	0,9957731	6,6601733	7,9661852	0,8325218	0,9957734	7,9999985	7,9999981
256x256	2,51061E-08	2,07363E-08	1,0101385	1,2084987	8,0811082	9,6679896	1,0101387	1,2084988	7,9999986	7,9999990
272x272	2,83842E-08	5,18232E-08	1,2113756	1,4495482	9,6910052	11,5963860	1,2113758	1,4495486	7,9999987	7,9999980
288x288	3,28151E-08	5,49764E-08	1,4377076	1,7206919	11,5016608	13,7655356	1,4377078	1,7206923	7,9999987	7,9999982
304x304	3,30865E-08	5,51136E-08	1,6906089	2,0237000	13,5248715	16,1896006	1,6906092	2,0237005	7,9999989	7,9999985

Table 11: Calculated WCET and Throughput for a system freq. 150MHz and P=16

Conventional/Blocking Matrix Multiplication										
NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_16_Conv	Texe_16_Block	K_Conv	K_Block
32x32	9,57E-09	6,61E-08	0,0020220	0,0023607	0,0323525	0,0377722	0,0020222	0,0023618	15,9988645	15,9932841
48x48	8,10E-09	4,66E-08	0,0067604	0,0079668	0,1081660	0,1274694	0,0067605	0,0079675	15,9997125	15,9985956
64x64	2,09E-08	7,76E-09	0,0159497	0,0188837	0,2551955	0,3021396	0,0159500	0,0188838	15,9996854	15,9999014
80x80	2,19E-08	2,89E-08	0,0310646	0,0368817	0,4970336	0,5901079	0,0310649	0,0368822	15,9998307	15,9998119
96x96	2,11E-08	2,40E-08	0,0535796	0,0637310	0,8572734	1,0196966	0,0535799	0,0637314	15,9999056	15,9999094
112x112	2,17E-08	2,10E-08	0,0849692	0,1012019	1,3595079	1,6192301	0,0849696	0,1012022	15,9999386	15,9999502
128x128	2,04E-08	2,01E-08	0,1267081	0,1510645	2,0273299	2,4170322	0,1267084	0,1510648	15,9999613	15,9999681
144x144	2,16E-08	1,66E-08	0,1802708	0,2150892	2,8843326	3,4414267	0,1802711	0,2150894	15,9999712	15,9999815
160x160	2,09E-08	1,52E-08	0,2471318	0,2950461	3,9541088	4,7207375	0,2471321	0,2950463	15,9999797	15,9999876
176x176	2,16E-08	1,64E-08	0,3287657	0,3927055	5,2602515	6,2832885	0,3287660	0,3927058	15,9999843	15,9999900
192x192	2,05E-08	7,45E-09	0,4266471	0,5098377	6,8263537	8,1574032	0,4266474	0,5098378	15,9999885	15,9999965
208x208	2,15E-08	1,44E-08	0,5422505	0,6482129	8,6760083	10,3714060	0,5422508	0,6482131	15,9999905	15,9999947
224x224	2,09E-08	2,08E-08	0,6770505	0,8096013	10,8328083	12,9536204	0,6770508	0,8096016	15,9999926	15,9999938
240x240	2,17E-08	3,44E-08	0,8325216	0,9957731	13,3203467	15,9323704	0,8325220	0,9957737	15,9999937	15,9999917
256x256	2,51E-08	2,07E-08	1,0101385	1,2084987	16,1622164	19,3359792	1,0101389	1,2084990	15,9999940	15,9999959
272x272	2,84E-08	5,18E-08	1,2113756	1,4495482	19,3820104	23,1927719	1,2113761	1,4495490	15,9999944	15,9999914
288x288	3,28E-08	5,50E-08	1,4377076	1,7206919	23,0033216	27,5310712	1,4377081	1,7206928	15,9999945	15,9999923
304x304	3,31E-08	5,51E-08	1,6906089	2,0237000	27,0497429	32,3792012	1,6906094	2,0237009	15,9999953	15,9999935

The remainder of this section presents charts comparing the calculated throughput. The varied parameters when calculating the throughput are among others the number of cores, the system frequency, the caches, algorithms etc. with respect to increased workload (increased matrix size).

Figure 12, illustrates the calculated throughput for 4 cores running 4 tasks in parallel. The figure also shows 8 curves of throughput divided into 4 pairs (one conventional and one blocking algorithm), where each pair has the same system clock frequency. Additionally, the series of the clock frequencies used to calculate the 4 pairs are 150MHz, 300MHz, 600MHz and 1200MHz.

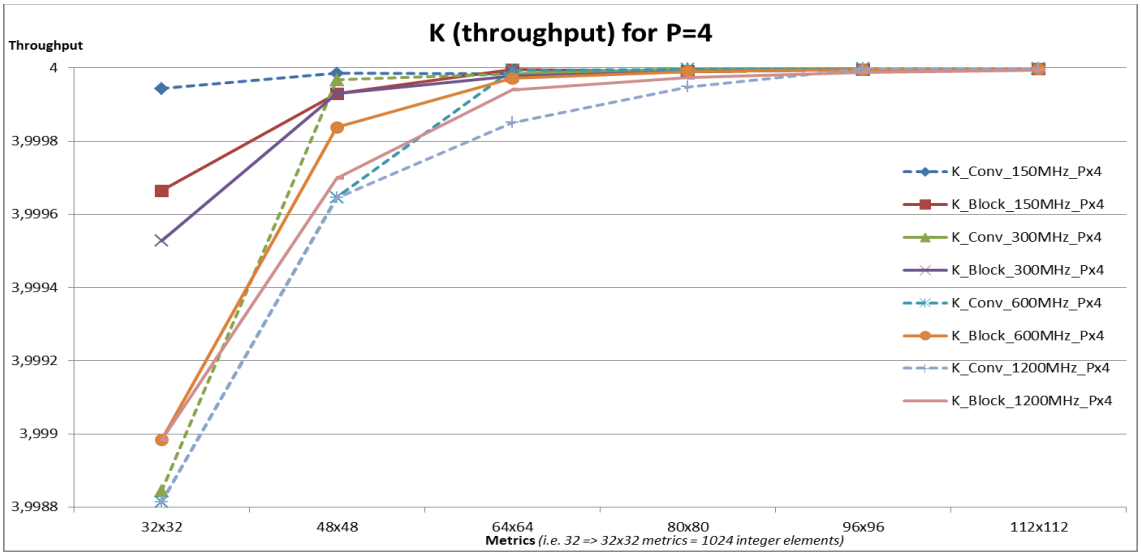


Figure 12: P=4, calculated throughputs for the simulated system clock frequencies.

Figures 13 and 14 are also showing the calculated throughput for 8 cores running 8 tasks in parallel and 16 cores running 16 tasks in parallel respectively.

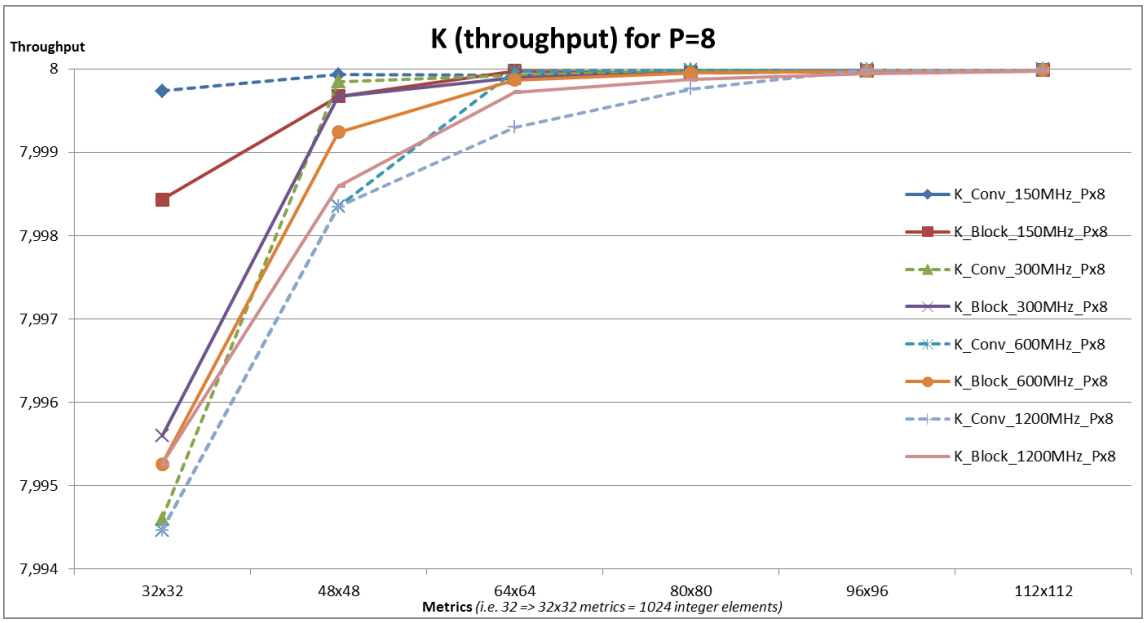


Figure 13: P=8, calculated throughputs for the simulated system clock frequencies.

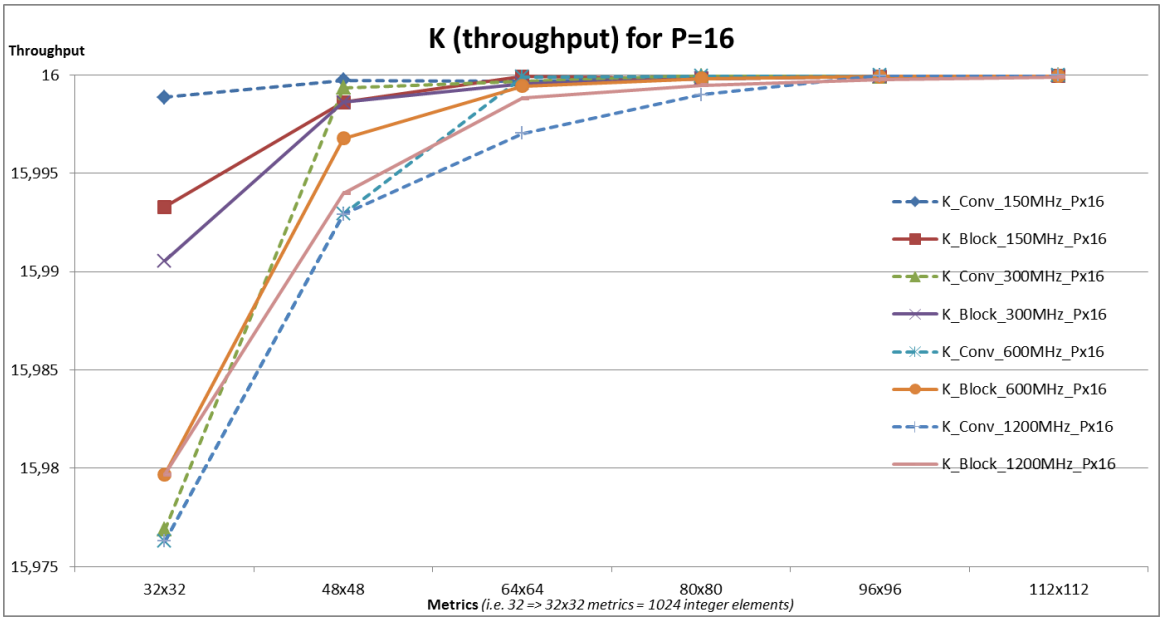


Figure 14: P=16, calculated throughputs for the simulated system clock frequencies.

5. Discussion

The findings of this Master's Thesis are presented in the result chapter. These results are acquired by using the method described in the system assumption chapter, which mainly consists of a simulation tool and an analytical model and supporting scripts. The goal has been to develop a method that makes it possible to estimate a safe and tight WCET in multicore system like LEON4FT.

Two matrix multiplication applications are used to evaluate the method. Prior to choosing this approach it is found that matrix multiplication is used in previous studies to evaluating cache efficiency and/or WCET. Moreover, it is found that matrix multiplication is a data intensive application that suites assessing multi-level memory systems as well as multicore performance. Matrix multiplication is a particularly good evaluation benchmark since by simply increasing or decreasing the matrix size, the application workload becomes higher or lower.

The data collected from the simulation tool that is used to calculate the multicore system's throughput, a metrics used to evaluate the safeness and tightness of an estimated WCET, is presented in Tables 5 through 8. These 4 tables show the results for the 4 different system clock frequencies and the 2 matrix multiplication algorithms. The measured parameters (the number of instruction, the L1 miss-rate and the L2 miss-rate) listed in these tables are used to calculate the average memory access-time, the worst case execution time and the throughput. These values were presented in Tables from 9 to 11. These tables display the result calculated for the system clock frequency at 150MHz, the algorithms and the number of processors are however changed. Since, these values are not directly illustrative and do not show whether the WCET estimated is safe or tight the result for the system clock frequencies are added into the appendices. Nevertheless, charts illustrating the throughput of the multicore running in parallel are added into the last part of the result chapter.

Figures from 12 to 14 show the throughput of the tasks executed in parallel. As can be seen in these figures the throughput is lowest at the lowest simulated matrix size (32x32) for all the different number of multicores, algorithms and clock frequencies. Furthermore, the throughput improves as the matrix size gets bigger. This can be interpreted as follows: as the parallel tasks' workload increases, the system's throughput improves in reference to the serially executed tasks.

Nevertheless, the aforementioned figures expose that there is relative throughput gap between the different system clock frequencies. The highest throughput for all the simulated matrix sizes is reached when the system is running conventional matrix multiplication at 150MHz (the lowest simulated system clock frequency). This is to be expected as the bottleneck is then in instruction processing. Conversely, the lowest throughput for all the simulated matrix sizes is reached when the system is running conventional matrix multiplication at 1200MHz (the highest simulated system clock frequency). Again, as was discussed in Section 2 the higher the CPU's clock frequency gets the bigger the speed gap between the CPU and memory gets a.k.a. memory wall. The two simulated algorithms in relation to the different matrix sizes seem to have different effect on the throughput. For example, at the lower matrix size, the conventional matrix multiplication applications have the lowest throughput for all system clocks except 300MHz. Yet again, this supports that the lower system frequencies' bottleneck is the instruction processing. Looking into the simulation result tables presented in Section 4.1 and in the appendices one can see that the blocking algorithm has higher instructions than the conventional algorithm.

An overall observation made is that the blocking algorithm has a bit higher throughput for most of the smaller matrices compared to the conventional algorithm. However, this slight improvement of the throughput with the blocking algorithm was not as significant as was anticipated. Previous studies, discussed in Section 2, present that the blocking algorithm would be more optimal than the conventional in terms of the cache utilisation etc. On the other hand, this Master Thesis did not seek to optimise the blocking algorithm's parameters such as the so called the blocking factor. The blocking factor, which is the size of the block of data the algorithm can copy from a memory to a cache or vice-versa in each access, can be calculated based on the cache size to get an optimal blocking algorithm.

Finally, it was found that the throughput of the simulated matrix sizes from 112x112 to 304x304 is almost optimal for all the multicore arrangements (P=4, P=8 and P=16) regardless the range of system speed and algorithm.

6. Conclusion & Recommendations

6.1 Conclusion

The aim of this Master's Thesis has been to find a method to estimate a safe and tight WCET in multicore system with multilevel memory hierarchies. The first step taken to find a solution was to study the prior work in the field to lay a foundation for the project. It has been found that prior research has established methods to estimate safe and acceptably tight bounds on memory access time in single-processor systems. The multicore systems, the embedded multicore systems in particularly, on the other hand do not have well established WCET estimation methods. The multicores have in a few years become the new performance improvement approach, since single-core solutions are no longer able to deliver a performance boost. However, the software designers are still searching better ways to fully utilise the multicore performance.

LEON4 is an example of the new multicore SoC systems that the designers are currently exploring to fully utilise its multicores concurrently. Actually, RUAG, which is the company that together with Chalmers initiated this Master's Thesis project, is currently using the LEON4 in serial execution mode. This thesis project is therefore part of on-going efforts that are striving to explore new ways to improve the multicore systems throughput, particularly by developing a method to estimate LEON4's WCET estimate.

The simulation results were presented in the result chapter, and then discussed in the discussion chapter. These results reveal that the simulated system's throughput is almost close to the maximum expected throughput when the tasks are executed in parallel. Additionally, the calculated WCET for the parallel tasks (Texe_N) is significantly lower than the serial tasks (Texe_1). Therefore, the conclusion drawn is that these findings indicate that by running 4 tasks on the LEON4's 4 cores, the WCET is not only safe and tight but also the system's throughput is better off than when running the tasks serially.

6.2 Recommendations

Although, this Master's Thesis project's findings are encouraging there are some work left. Below are the future recommendations for possible continuation of the project. These recommendations can be followed sequentially or any desired point can be picked and carried out.

Recommendations point:

1. Carry out further testing on the this project's method by using Mibench (the automotive category) benchmark
2. Use gem5's SPARC architecture, which is the architecture that LEON4 is based, if the simulation tool is made stable to support this option.
3. Use gem5 in multicore configuration
4. Implement LEON4 on FPGA and carry out WCET estimation on hardware

References

[1] T. Lundqvist and P. Stenstrom, “A method to improve the estimated worst-case performance of data caching,” in *In RTCSA ’99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, 1999.

[2] N. Binkert, B. Beckmann and Et Al., “The gem5 Simulator,” *ACM SIGARCH Computer Architecture News,* vol. 39, no. 2, pp. 1-7, May 2011.

[3] T. Sherwood and J. Yi, “Guest Editors' Introduction: Computer Architecture Simulation and Modeling,” *IEEE Computer Society,* vol. 26, no. 4, pp. 5-7, July/August 2006.

[4] M. Lam, E. Rothberg and M. Wolf, “The cache performance and optimizations of blocked algorithms,” in *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, New York, 1991.

[5] C. Ferdinad and R. Wilhelm, “On Predicting Data Cache Behavior for Real-Time Systems,” in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, London, 1998.

[6] A. Demirhan, “On Increasing the Effective Blocking Factor of a Matrix for a Given Cache Organization,” Monterey, CA, 1992.

[7] D. Harris and N. Weste , Integrated Circuit Design, Fourth ed., Boston: Pearson Education, Inc., 2011.

[8] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Fourth ed., Amsterdam: Elsevier, 2012.

[9] A. Chandrakasan, Integrated Circuits and Systems, S. Keckler, K. Olukotun and H. Hofstee, Eds., New York: Springer Science+Business Media, LLC, 2009.

[10] “Wikipedia.org,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Moore%27s_law. [Accessed 20 July 2013].

[11] V. Pankratius, A.-R. Adl-Tabatabai and W. Tichy, “Introduction,” in *Development, Fundamentals of Multicore Software*, T. Walter, Ed., Boca Raton, CRC Press, Inc., 2011, pp. 1-5.

[12] F. Schirrmeister, “Multicore Architectures,” in *Real World Multicore Embedded Systems*, B. Moyer, Ed., Amsterdam, Elsevier Inc., 2013, pp. 33-73.

[13] D. Fittes, “Using Multicore Processors in Embedded Systems,” Coventry, 2009.

[14] B. Wilkinson, “Fundamentals of Multicore Hardware and Parallel Programming,” in *Fundamentals of Multicore Software Development*, W. Tichy, Ed., Boca Raton, CRC Press, LLC, 2011, pp. 9-29.

[15] S. McKee and W. A. Wulf, “Hitting the Memory Wall: Implications of the Obvious,” *Computer Architecture News,* vol. 23, no. 1, pp. 20-24, March 1995.

[16] M. Rani and A. Asaduzzaman, “Power Aware Design of Second Level Cache for Multicore Embedded Systems,” in *Proceedings of the IEEE SoutheastCon*, Concord, 2010.

[17] M. Dubois, M. Annavaran and P. Stenstrom, Parallel Computer Organization and Design, Cambridge: Cambridge University Press, 2012.

[18] A. Vajda, “Multi-core and Many-core Processor Architectures,” in *Programming Many-Core Chips*, New York, Springer, 2011, pp. 9-43.

[19] G. Jain, “Memory Models for Embedded Multicore Architecture,” in *Real World Multicore Embedded Systems*, B. Moyer, Ed., Amsterdam, Elsevier Inc., 2013, pp. 75-116.

[20] T. R. Kumar, C. Ravikumar and R. Govindarajan, “Memory Architecture Exploration Framework for Cache Based Embedded SoC,” in *Proceedings of the 21st International Conference on VLSI Design*, Hyderabad, 2008.

[21] “Computer Science & Information Technology,” July 2013. [Online]. Available: http://csitnotes.blogspot.se/2011/12/cache-design-parameters.html.

[22] T. Lundqvist, “A WCET Analysis Method for Pipelined Microprocessors with Cache Memories,” Gothenburg, 2002.

[23] J. Yan and W. Zhang, “WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches,” in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, St. Louis, 2008.

[24] R. Heckmann and C. Ferdinand, “Worst-Case Execution Time Prediction by Static Program Analysis,” in *Proceesings of 18th International Parallel and Distributed Processing Symposium*, Santa Fe, 2004.

[25] L. Kong, “A Worst-Case Execution Time Analysis Approach Based on Independent Paths for ARM Programs,” *Wuhan University journal of natural sciences,* vol. 17, no. 5, pp. 391-399, 2012.

[26] Wilhelm and et al., “The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools,” *ACM Transactions on Embedded Computing Systems,* vol. 7, no. 3, pp. 1-53, April 2008.

[27] July 2013. [Online]. Available: http://www.mrtc.mdh.se/projects/WCC/.

[28] “Microelectronics,” European Space Agency, 31 05 2013. [Online]. Available: http://microelectronics.esa.int/ngmp/ngmp.htm. [Accessed 04 08 2013].

[29] “Microelectronics,” 31 05 2013. [Online]. Available: http://microelectronics.esa.int/ngmp/LEON4-NGMP-DRAFT-2-1.pdf. [Accessed 05 08 2013].

[30] “Microelectronics,” 31 05 2013. [Online]. Available: http://microelectronics.esa.int/ngmp/LEON4-N2X-DS-1-8.pdf. [Accessed 05 08 2013].

[31] ”Gaisler,” 2013. [Online]. Available: http://www.gaisler.com/doc/LEON4_32-bit_processor_core.pdf. [Använd 05 08 2013].

[32] “The gem5 Simulator System,” 18 02 2013. [Online]. Available: http://www.m5sim.org/Main_Page. [Accessed 06 08 2013].

[33] “The gem5 Simulator System,” 18 02 2013. [Online]. Available: http://www.m5sim.org/Architecture_Support. [Accessed 08 08 2013].

[34] D. Hardy, T. Piquet and I. Puaut, “Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches,” in *Proceedings of the 30th Real-Time Systems Symposium*, Washington D.C., 2009.

[35] B. Lesage, D. Hardy and I. Puauat, “Shared Data Cache Conflicts Reduction for WCET Computation in Multicore Architectures,” in *18th International Conference on Real-Time and Network Systems*, 2010.

[36] A. Saulsbury, F. Pong and A. Nowatzyk, “Missing the Memory Wall: The Case for Processor/Memory Integration,” in *Proceedings of the 23rd annual international symposium on Computer architecture*, New York, 1996.

[37] V. Suhendra and T. Mitra, “Exploring locking & partitioning for predictable shared caches on multi-cores,” in *In DAC ’08: Proceedings of the 45th annual Design Automation Conference*, New York, 2008.

Appendix A: Calculated Result for a System Frequency at 300MHz

Calculated WCET and Throughput for a system freq. 300MHz and P=4.

Conventional/Blocking Metrics Multiplication										
NxN_Metrics	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	9,75333E-08	4,65115E-08	0,00101101	0,001180347	0,00404443	0,004721573	0,0010114	0,001180533	3,998842793	3,999527216
48	9,17588E-09	2,33113E-08	0,003380183	0,003983397	0,01352077	0,01593368	0,00338022	0,00398349	3,999967425	3,999929776
64	1,04523E-08	1,77393E-08	0,00797485	0,00944186	0,031899442	0,037767511	0,007974892	0,009441931	3,999984272	3,999977455
80	3,50756E-09	1,44559E-08	0,01553229	0,018440857	0,062129174	0,073763484	0,015532304	0,018440914	3,99999729	3,999990593
96	1,05255E-08	1,21614E-08	0,026789783	0,031865507	0,107159175	0,127462075	0,026789825	0,031865555	3,999995285	3,99999542
112	3,11274E-09	1,05015E-08	0,04248461	0,05060093	0,169938452	0,202403762	0,042484622	0,050600972	3,999999121	3,99999751
128	1,02237E-08	3,19098E-09	0,06335405	0,075532247	0,253416241	0,302128999	0,063354091	0,075532259	3,999998064	3,999999493
144	7,90656E-09	8,28498E-09	0,090135383	0,107544577	0,360541565	0,43017834	0,090135415	0,10754461	3,999998947	3,999999076
160	1,04486E-08	7,66428E-09	0,12356589	0,14752304	0,494263602	0,590092191	0,123565932	0,147523071	3,999998985	3,999999377
176	1,07804E-08	8,29136E-09	0,16438285	0,196352757	0,657531443	0,78541106	0,164382893	0,19635279	3,999999213	3,999999493
192	1,02624E-08	7,00612E-09	0,213323543	0,254918847	0,853294214	1,019675415	0,213323584	0,254918875	3,999999423	3,99999967
208	1,07567E-08	7,21136E-09	0,27112525	0,32410643	1,084501043	1,296425749	0,271125293	0,324106459	3,999999524	3,999999733
224	1,04356E-08	1,04548E-08	0,33852525	0,404800627	1,354101042	1,619202548	0,338525292	0,404800668	3,99999963	3,99999969
240	1,0871E-08	1,71766E-08	0,416260823	0,497886557	1,665043337	1,991546295	0,416260867	0,497886625	3,999999687	3,999999586
256	1,25537E-08	1,03488E-08	0,50506925	0,60424934	2,02027705	2,416997401	0,5050693	0,604249381	3,999999702	3,999999794
272	1,41921E-08	2,63084E-08	0,60568781	0,724774097	2,422751297	2,899096492	0,605687867	0,724774202	3,999999719	3,999999564
288	1,64053E-08	2,76971E-08	0,718853783	0,860345947	2,875415199	3,441383897	0,718853849	0,860346057	3,999999726	3,999999614
304	1,65433E-08	2,75568E-08	0,84530445	1,01185001	3,381217866	4,04740015	0,845304516	1,01185012	3,999999765	3,999999673

Calculated WCET and Throughput for a system freq. 300MHz and P=8.

Conventional/Blocking Metrics Multiplication										
NxN_Metrics	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	9,75333E-08	9,29057E-08	0,00101101	0,001180347	0,00808886	0,009443517	0,00101179	0,00118109	7,994601783	7,995594987
48	9,17588E-09	2,33113E-08	0,003380183	0,003983397	0,02704154	0,03186736	0,003380257	0,003983583	7,999847985	7,999672297
64	1,04523E-08	1,77393E-08	0,00797485	0,00944186	0,063798884	0,075535022	0,007974934	0,009442002	7,999926604	7,999894789
80	3,50756E-09	1,44559E-08	0,01553229	0,018440857	0,124258348	0,147526969	0,015532318	0,018440972	7,999987354	7,999956101
96	1,05255E-08	1,21614E-08	0,026789783	0,031865507	0,214318351	0,254924151	0,026789868	0,031865604	7,999977998	7,999978628
112	3,11274E-09	1,05015E-08	0,04248461	0,05060093	0,339876905	0,404807524	0,042484635	0,050601014	7,999995897	7,999988378
128	1,02237E-08	3,19098E-09	0,06335405	0,075532247	0,506832482	0,604257999	0,063354132	0,075532272	7,999990963	7,999997634
144	7,90656E-09	8,28498E-09	0,090135383	0,107544577	0,72108313	0,86035668	0,090135447	0,107544643	7,999995088	7,999995686
160	1,04486E-08	7,66428E-09	0,12356589	0,14752304	0,988527204	1,180184381	0,123565974	0,147523101	7,999995265	7,999997091
176	1,07804E-08	8,29136E-09	0,16438285	0,196352757	1,315062886	1,57082212	0,164382936	0,196352823	7,999996327	7,999997635
192	1,02624E-08	7,00612E-09	0,213323543	0,254918847	1,706588429	2,039350829	0,213323625	0,254918903	7,999997306	7,999998461
208	1,07567E-08	7,21136E-09	0,27112525	0,32410643	2,169002086	2,592851498	0,271125336	0,324106488	7,999997778	7,999998754
224	1,04356E-08	1,04548E-08	0,33852525	0,404800627	2,708202083	3,238405097	0,338525333	0,40480071	7,999998274	7,999998554
240	1,0871E-08	1,71766E-08	0,416260823	0,497886557	3,330086674	3,983092591	0,41626091	0,497886694	7,999998538	7,999998068
256	1,25537E-08	1,03488E-08	0,50506925	0,60424934	4,0405541	4,833994803	0,50506935	0,604249423	7,999998608	7,999999041
272	1,41921E-08	2,63084E-08	0,60568781	0,724774097	4,845502594	5,798192984	0,605687924	0,724774307	7,999998688	7,999997967
288	1,64053E-08	2,76971E-08	0,718853783	0,860345947	5,750830398	6,882767795	0,718853915	0,860346168	7,999998722	7,999998197
304	1,65433E-08	2,75568E-08	0,84530445	1,01185001	6,762435732	8,0948003	0,845304582	1,01185023	7,999998904	7,999998475

Calculated WCET and Throughput for a system freq. 300MHz and P=16.

Conventional/Blocking Metrics Multiplication										
NxN_Metrics	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	9,75333E-08	4,65115E-08	0,00101101	0,001180347	0,016177721	0,018886291	0,001012571	0,001181091	15,97688261	15,99054878
48	9,17588E-09	2,33113E-08	0,003380183	0,003983397	0,05408308	0,06373472	0,00338033	0,00398377	15,99934852	15,99859562
64	1,04523E-08	1,77393E-08	0,00797485	0,00944186	0,127597767	0,151070044	0,007975017	0,009442144	15,99968545	15,9995491
80	3,50756E-09	1,44559E-08	0,01553229	0,018440857	0,248516696	0,295053938	0,015532346	0,018441088	15,9999458	15,99981186
96	1,05255E-08	1,21614E-08	0,026789783	0,031865507	0,428636702	0,509848301	0,026789952	0,031865701	15,99990571	15,9999084
112	3,11274E-09	1,05015E-08	0,04248461	0,05060093	0,67975381	0,809615048	0,04248466	0,050601098	15,99998242	15,99995019
128	1,02237E-08	3,19098E-09	0,06335405	0,075532247	1,013664964	1,208515998	0,063354214	0,075532298	15,99996127	15,99998986
144	7,90656E-09	8,28498E-09	0,090135383	0,107544577	1,44216626	1,720713359	0,09013551	0,107544709	15,99997895	15,99998151
160	1,04486E-08	7,66428E-09	0,12356589	0,14752304	1,977054407	2,360368763	0,123566057	0,147523163	15,99997971	15,99998753
176	1,07804E-08	8,29136E-09	0,16438285	0,196352757	2,630125772	3,141644239	0,164383022	0,196352889	15,99998426	15,99998987
192	1,02624E-08	7,00612E-09	0,213323543	0,254918847	3,413176858	4,078701659	0,213323708	0,254918959	15,99998845	15,9999934
208	1,07567E-08	7,21136E-09	0,27112525	0,32410643	4,338004172	5,185702995	0,271125422	0,324106545	15,99999048	15,99999466
224	1,04356E-08	1,04548E-08	0,33852525	0,404800627	5,416404167	6,476810194	0,338525417	0,404800794	15,9999926	15,9999938
240	1,0871E-08	1,71766E-08	0,416260823	0,497886557	6,660173347	7,966185181	0,416260997	0,497886831	15,99999373	15,99999172
256	1,25537E-08	1,03488E-08	0,50506925	0,60424934	8,081108201	9,667989606	0,505069451	0,604249506	15,99999403	15,99999589
272	1,41921E-08	2,63084E-08	0,60568781	0,724774097	9,691005187	11,59638597	0,605688037	0,724774518	15,99999438	15,99999129
288	1,64053E-08	2,76971E-08	0,718853783	0,860345947	11,5016608	13,76553559	0,718854046	0,86034639	15,99999452	15,99999227
304	1,65433E-08	2,75568E-08	0,84530445	1,01185001	13,52487146	16,1896006	0,845304715	1,011850451	15,9999953	15,99999346

Appendix B: Calculated Result for a System Frequency at 600MHz

Calculated WCET and Throughput for a system freq. 600MHz and P=4.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	5,00286E-08	5,00272E-08	0,000505505	0,000590173	0,00202222	0,002360893	0,000505705	0,000590373	3,99881286	3,998983141
48	4,9904E-08	2,6928E-08	0,001690092	0,001991698	0,006760566	0,007966901	0,001690291	0,001991806	3,999645713	3,999837767
64	1,8726E-09	1,1225E-08	0,003987425	0,00472093	0,015949707	0,018883765	0,003987432	0,004720975	3,999994364	3,999971468
80	2,1398E-09	7,53867E-09	0,007766145	0,009220428	0,031064589	0,036881743	0,007766154	0,009220458	3,999996694	3,999990189
96	1,473E-09	6,08072E-09	0,013394892	0,015932753	0,053579573	0,063731038	0,013394898	0,015932778	3,99999868	3,99999542
112	1,6291E-09	5,26943E-09	0,021242305	0,025300465	0,084969227	0,101201881	0,021242312	0,025300486	3,99999908	3,999997501
128	5,11151E-09	4,61594E-09	0,031677025	0,037766123	0,12670812	0,151064512	0,031677045	0,037766142	3,999998064	3,999998533
144	1,34148E-09	4,14249E-09	0,045067692	0,053772288	0,180270772	0,21508917	0,045067697	0,053772305	3,999999643	3,999999076
160	5,2178E-09	3,83214E-09	0,061782945	0,07376152	0,247131801	0,295046095	0,061782966	0,073761535	3,999998987	3,999999377
176	1,15431E-09	4,14568E-09	0,082191425	0,098176378	0,328765705	0,39270553	0,08219143	0,098176395	3,999999831	3,999999493
192	5,12999E-09	3,51214E-09	0,106661772	0,127459423	0,426647107	0,509837707	0,106661792	0,127459437	3,999999423	3,999999669
208	1,02952E-09	3,60568E-09	0,135562625	0,162053215	0,542250504	0,648212874	0,135562629	0,162053229	3,999999909	3,999999733
224	5,21304E-09	5,27633E-09	0,169262625	0,202400313	0,677050521	0,809601274	0,169262646	0,202400334	3,99999963	3,999999687
240	1,89105E-09	8,59554E-09	0,208130412	0,248943278	0,832521654	0,995773148	0,208130419	0,248943313	3,999999891	3,999999586
256	6,27544E-09	2,20435E-09	0,252534625	0,30212467	1,010138525	1,208498689	0,25253465	0,302124679	3,999999702	3,999999912
272	6,13859E-09	1,3171E-08	0,302843905	0,362387048	1,211375645	1,449548246	0,30284393	0,362387101	3,999999757	3,999999564
288	8,20112E-09	1,38485E-08	0,359426892	0,430172973	1,437707599	1,720691949	0,359426924	0,430173029	3,999999726	3,999999614
304	8,02386E-09	1,37784E-08	0,422652225	0,505925005	1,690608932	2,023700075	0,422652257	0,50592506	3,999999772	3,999999673

Calculated WCET and Throughput for a system freq. 600MHz and P=8.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	5,00286E-08	5,00272E-08	0,000505505	0,000590173	0,00404444	0,004721787	0,000505905	0,000590574	7,994462206	7,995256267
48	4,9904E-08	2,6928E-08	0,001690092	0,001991698	0,013521133	0,015933802	0,001690491	0,001991914	7,998346856	7,999242955
64	1,8726E-09	1,1225E-08	0,003987425	0,00472093	0,031899415	0,03776753	0,00398744	0,00472102	7,999973701	7,99986685
80	2,1398E-09	7,53867E-09	0,007766145	0,009220428	0,062129177	0,073763487	0,007766162	0,009220489	7,99998457	7,999954214
96	1,473E-09	6,08072E-09	0,013394892	0,015932753	0,107159145	0,127462075	0,013394903	0,015932802	7,999993842	7,999978628
112	1,6291E-09	5,26943E-09	0,021242305	0,025300465	0,169938453	0,202403762	0,021242318	0,025300507	7,999995705	7,999988337
128	5,11151E-09	4,61594E-09	0,031677025	0,037766123	0,253416241	0,302129024	0,031677066	0,03776616	7,999990964	7,999993155
144	1,34148E-09	4,14249E-09	0,045067692	0,053772288	0,360541544	0,43017834	0,045067702	0,053772321	7,999998333	7,999995686
160	5,2178E-09	3,83214E-09	0,061782945	0,07376152	0,494263602	0,590092191	0,061782987	0,073761551	7,999995271	7,999997091
176	1,15431E-09	4,14568E-09	0,082191425	0,098176378	0,657531409	0,78541106	0,082191434	0,098176411	7,999999214	7,999997635
192	5,12999E-09	3,51214E-09	0,106661772	0,127459423	0,853294214	1,019675415	0,106661813	0,127459451	7,999997307	7,999998457
208	1,02952E-09	3,60568E-09	0,135562625	0,162053215	1,084501008	1,296425749	0,135562633	0,162053244	7,999999575	7,999998754
224	5,21304E-09	5,27633E-09	0,169262625	0,202400313	1,354101042	1,619202549	0,169262667	0,202400356	7,999998275	7,99999854
240	1,89105E-09	8,59554E-09	0,208130412	0,248943278	1,665043308	1,991546295	0,208130427	0,248943347	7,999999491	7,999998066
256	6,27544E-09	2,20435E-09	0,252534625	0,30212467	2,02027705	2,416997378	0,252534675	0,302124688	7,999998608	7,999999591
272	6,13859E-09	1,3171E-08	0,302843905	0,362387048	2,422751289	2,899096492	0,302843954	0,362387154	7,999998865	7,999997965
288	8,20112E-09	1,38485E-08	0,359426892	0,430172973	2,875415199	3,441383897	0,359426957	0,430173084	7,999998722	7,999998197
304	8,02386E-09	1,37784E-08	0,422652225	0,505925005	3,381217864	4,04740015	0,422652289	0,505925115	7,999998937	7,999998475

Calculated WCET and Throughput for a system freq. 600MHz and P=16.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	5,00286E-08	5,00272E-08	0,000505505	0,000590173	0,00808888	0,009443574	0,000506305	0,000590974	15,97628536	15,97968348
48	4,9904E-08	2,6928E-08	0,001690092	0,001991698	0,027042265	0,031867604	0,00169089	0,001992129	15,99291677	15,99675587
64	1,8726E-09	1,1225E-08	0,003987425	0,00472093	0,06379883	0,07553506	0,003987455	0,00472111	15,99988729	15,99942937
80	2,1398E-09	7,53867E-09	0,007766145	0,009220428	0,124258354	0,147526974	0,007766179	0,009220549	15,99993387	15,99980378
96	1,473E-09	6,08072E-09	0,013394892	0,015932753	0,21431829	0,254924151	0,013394915	0,015932851	15,99997361	15,9999084
112	1,6291E-09	5,26943E-09	0,021242305	0,025300465	0,339876906	0,404807524	0,021242331	0,025300549	15,99998159	15,99995001
128	5,11151E-09	4,61594E-09	0,031677025	0,037766123	0,506832482	0,604258047	0,031677107	0,037766197	15,99996127	15,99997067
144	1,34148E-09	4,14249E-09	0,045067692	0,053772288	0,721083088	0,86035668	0,045067713	0,053772355	15,99999286	15,99998151
160	5,2178E-09	3,83214E-09	0,061782945	0,07376152	0,988527203	1,180184381	0,061783028	0,073761581	15,99997973	15,99998753
176	1,15431E-09	4,14568E-09	0,082191425	0,098176378	1,315062818	1,57082212	0,082191443	0,098176445	15,99999663	15,99998987
192	5,12999E-09	3,51214E-09	0,106661772	0,127459423	1,706588429	2,03935083	0,106661854	0,12745948	15,99998846	15,99999339
208	1,02952E-09	3,60568E-09	0,135562625	0,162053215	2,169002016	2,592851498	0,135562641	0,162053273	15,99999818	15,99999466
224	5,21304E-09	5,27633E-09	0,169262625	0,202400313	2,708202083	3,238405098	0,169262708	0,202400398	15,99999261	15,99999374
240	1,89105E-09	8,59554E-09	0,208130412	0,248943278	3,330086617	3,983092591	0,208130442	0,248943416	15,99999782	15,99999171
256	6,27544E-09	2,20435E-09	0,252534625	0,30212467	4,0405541	4,833994755	0,252534725	0,302124705	15,99999404	15,99999825
272	6,13859E-09	1,3171E-08	0,302843905	0,362387048	4,845502578	5,798192984	0,302844003	0,362387259	15,99999514	15,99999128
288	8,20112E-09	1,38485E-08	0,359426892	0,430172973	5,750830398	6,882767795	0,359427023	0,430173195	15,99999452	15,99999227
304	8,02386E-09	1,37784E-08	0,422652225	0,505925005	6,762435728	8,0948003	0,422652353	0,505925225	15,99999544	15,99999346

Appendix C: Calculated Result for a System Frequency at 1200MHz

Calculated WCET and Throughput for a system freq. 1200MHz and P=4.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	2,50142E-08	2,50136E-08	0,000252753	0,000295087	0,00101111	0,001180447	0,000252853	0,000295187	3,998812864	3,998983141
48	2,50096E-08	2,5009E-08	0,000845046	0,000995849	0,003380283	0,003983497	0,000845146	0,000995949	3,999644896	3,999698671
64	2,48456E-08	1,17728E-08	0,001993713	0,002360465	0,007974949	0,009441907	0,001993812	0,002360512	3,999850464	3,999940151
80	1,65742E-08	1,02328E-08	0,003883073	0,004610214	0,015532356	0,018440898	0,003883139	0,004610255	3,999948781	3,999973365
96	1,33377E-09	7,88489E-09	0,006697446	0,007966377	0,026789789	0,031865538	0,006697451	0,007966408	3,99999761	3,999988123
112	8,14275E-10	5,78168E-09	0,010621153	0,012650233	0,042484613	0,050600953	0,010621156	0,012650256	3,99999908	3,999994516
128	2,55529E-09	2,31117E-09	0,015838513	0,018883062	0,06335406	0,075532256	0,015838523	0,018883071	3,999998064	3,999998531
144	6,70615E-10	2,68424E-09	0,022533846	0,026886144	0,090135386	0,107544587	0,022533849	0,026886155	3,999999643	3,999998802
160	5,85045E-10	2,12481E-09	0,030891473	0,03688076	0,123565892	0,147523048	0,030891475	0,036880768	3,999999773	3,999999309
176	5,7829E-10	2,14234E-09	0,041095713	0,049088189	0,164382852	0,196352765	0,041095715	0,049088198	3,999999831	3,999999476
192	2,56365E-09	1,75607E-09	0,053330886	0,063729712	0,213323554	0,254918854	0,053330896	0,063729719	3,999999423	3,999999669
208	5,14035E-10	1,80284E-09	0,067781313	0,081026608	0,271125252	0,324106437	0,067781315	0,081026615	3,999999909	3,999999733
224	5,4463E-10	2,65299E-09	0,084631313	0,101200157	0,338525252	0,404800637	0,084631315	0,101200167	3,999999923	3,999999685
240	8,8628E-10	4,31747E-09	0,104065206	0,124471639	0,416260827	0,497886574	0,104065209	0,124471656	3,999999898	3,999999584
256	3,1355E-09	6,76005E-09	0,126267313	0,151062335	0,505069263	0,604249367	0,126267325	0,151062362	3,999999702	3,999999463
272	1,49582E-08	6,59498E-09	0,151421953	0,181193524	0,60568787	0,724774123	0,151422012	0,181193551	3,999998815	3,999999563
288	4,01138E-09	6,92427E-09	0,179713446	0,215086487	0,718853799	0,860345974	0,179713462	0,215086514	3,999999732	3,999999614
304	2,51332E-08	6,8892E-09	0,211326113	0,252962503	0,845304551	1,011850038	0,211326213	0,25296253	3,999998573	3,999999673

Calculated WCET and Throughput for a system freq. 1200MHz and P=8.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	2,50142E-08	2,50136E-08	0,000252753	0,000295087	0,00202222	0,002360893	0,000252953	0,000295287	7,994462223	7,995256267
48	2,50096E-08	2,5009E-08	0,000845046	0,000995849	0,006760567	0,007966993	0,000845246	0,000996049	7,998343044	7,998593939
64	2,48456E-08	1,17728E-08	0,001993713	0,002360465	0,015949899	0,018883814	0,001993911	0,002360559	7,9993022	7,999720711
80	1,65742E-08	1,02328E-08	0,003883073	0,004610214	0,031064713	0,036881795	0,003883205	0,004610296	7,999760983	7,999875705
96	1,33377E-09	7,88489E-09	0,006697446	0,007966377	0,053579577	0,063731076	0,006697457	0,00796644	7,999988848	7,999944573
112	8,14275E-10	5,78168E-09	0,010621153	0,012650233	0,084969227	0,101201906	0,010621159	0,012650279	7,999995707	7,999974406
128	2,55529E-09	2,31117E-09	0,015838513	0,018883062	0,12670812	0,151064512	0,015838533	0,01888308	7,999990965	7,999993146
144	6,70615E-10	2,68424E-09	0,022533846	0,026886144	0,180270772	0,215089175	0,022533851	0,026886166	7,999998333	7,999994409
160	5,85045E-10	2,12481E-09	0,030891473	0,03688076	0,247131785	0,295046097	0,030891477	0,036880777	7,999998939	7,999996774
176	5,7829E-10	2,14234E-09	0,041095713	0,049088189	0,328765705	0,39270553	0,041095717	0,049088206	7,999999212	7,999997556
192	2,56365E-09	1,75607E-09	0,053330886	0,063729712	0,426647107	0,509837707	0,053330906	0,063729726	7,999997308	7,999998457
208	5,14035E-10	1,80284E-09	0,067781313	0,081026608	0,542250504	0,648212874	0,067781317	0,081026622	7,999999575	7,999998754
224	5,4463E-10	2,65299E-09	0,084631313	0,101200157	0,677050504	0,809601275	0,084631317	0,101200178	7,99999964	7,999998532
240	8,8628E-10	4,31747E-09	0,104065206	0,124471639	0,832521654	0,995773148	0,104065213	0,124471674	7,999999523	7,999998058
256	3,1355E-09	6,76005E-09	0,126267313	0,151062335	1,010138525	1,208498734	0,126267338	0,151062389	7,999998609	7,999997494
272	1,49582E-08	6,59498E-09	0,151421953	0,181193524	1,21137574	1,449548246	0,151422072	0,181193577	7,999994468	7,999997962
288	4,01138E-09	6,92427E-09	0,179713446	0,215086487	1,437707599	1,720691949	0,179713478	0,215086542	7,99999875	7,999998197
304	2,51332E-08	6,8892E-09	0,211326113	0,252962503	1,690609101	2,023700075	0,211326314	0,252962558	7,99999334	7,999998475

Calculated WCET and Throughput for a system freq. 1200MHz and P=16.

Conventional/Blocking Matrix Multiplication

NxN_Matrix	AMAT_Conv	AMAT_Block	Tproc_Conv	Tproc_Block	Texe_1_Conv	Texe_1_Block	Texe_4_Conv	Texe_4_Block	K_Conv	K_Block
32	2,50142E-08	2,50136E-08	0,000252753	0,000295087	0,00404444	0,004721787	0,000253153	0,000295487	15,97628543	15,97968348
48	2,50096E-08	2,5009E-08	0,000845046	0,000995849	0,013521133	0,015933987	0,000845446	0,000996249	15,99290044	15,99397523
64	2,48456E-08	1,17728E-08	0,001993713	0,002360465	0,031899798	0,037767628	0,00199411	0,002360653	15,99700973	15,9988031
80	1,65742E-08	1,02328E-08	0,003883073	0,004610214	0,062129425	0,07376359	0,003883338	0,004610378	15,99897568	15,99946732
96	1,33377E-09	7,88489E-09	0,006697446	0,007966377	0,107159155	0,127462153	0,006697467	0,007966503	15,99995221	15,99976246
112	8,14275E-10	5,78168E-09	0,010621153	0,012650233	0,169938453	0,202403813	0,010621166	0,012650325	15,9999816	15,99989031
128	2,55529E-09	2,31117E-09	0,015838513	0,018883062	0,253416241	0,302129024	0,015838553	0,018883099	15,99996128	15,99997063
144	6,70615E-10	2,68424E-09	0,022533846	0,026886144	0,360541544	0,43017835	0,022533857	0,026886187	15,99999286	15,99997604
160	5,85045E-10	2,12481E-09	0,030891473	0,03688076	0,494263569	0,590092194	0,030891482	0,036880794	15,99999545	15,99998617
176	5,7829E-10	2,14234E-09	0,041095713	0,049088189	0,657531409	0,785411061	0,041095722	0,049088223	15,99999662	15,99998953
192	2,56365E-09	1,75607E-09	0,053330886	0,063729712	0,853294214	1,019675415	0,053330927	0,06372974	15,99998846	15,99999339
208	5,14035E-10	1,80284E-09	0,067781313	0,081026608	1,084501008	1,296425749	0,067781321	0,081026636	15,99999818	15,99999466
224	5,4463E-10	2,65299E-09	0,084631313	0,101200157	1,354101009	1,619202549	0,084631321	0,101200199	15,99999846	15,99999371
240	8,8628E-10	4,31747E-09	0,104065206	0,124471639	1,665043308	1,991546296	0,10406522	0,124471708	15,99999796	15,99999168
256	3,1355E-09	6,76005E-09	0,126267313	0,151062335	2,02027705	2,416997468	0,126267363	0,151062443	15,99999404	15,99998926
272	1,49582E-08	6,59498E-09	0,151421953	0,181193524	2,422751479	2,899096492	0,151422192	0,18119363	15,99997629	15,99999126
288	4,01138E-09	6,92427E-09	0,179713446	0,215086487	2,875415198	3,441383897	0,17971351	0,215086597	15,99999464	15,99999227
304	2,51332E-08	6,8892E-09	0,211326113	0,252962503	3,381218202	4,04740015	0,211326515	0,252962613	15,99997146	15,99999346

Appendix D: Conventional Matrix Multiplication

CONVENTIONAL MATRIX MULTIPLICATION

```
/*     mconv.c     */
#include "util/m5/m5op.h"   /* enables access to some Gem5 functions */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* CONVENTIONAL METRICS MULTIPLICATION ALGORITHM */
void main(int argc, char **argv)
{
int MAXLOOP = atoi(argv[1]);
int N=MAXLOOP;
int i, j, k;

int A[N][N];
int B[N][N];
int C[N][N];

for (i=0; i<N; i++) for (j=0; j<N; j++){ A[i][j]=j;} //assigning values
to metrics A
for (i=0; i<N; i++) for (j=0; j<N; j++){ B[i][j]=j;} //assigning values
to metrics B
for (i=0; i<N; i++) for (j=0; j<N; j++){ C[i][j]=j;} //assigning values
to metrics C

m5_dumpreset_stats(0,0);
m5_checkpoint(0,0);
/****************** Region of Interest Starts Here ********************/
for (i=0; i<N; i++)
{
      for (j=0; j<N; j++)
      {
            for (k=0; k<N; k++)
            {
              C[i][j]+=A[i][k]*B[k][j];
            }
        }
}
/******************* Region of Interest Ends Here ********************/
m5_exit(0);
exit(0);                          /* No errors */
}
```

Appendix E: Blocking Algorithm Matrix Multiplication

```
#include "util/m5/m5op.h"   /* enables access to some Gem5 functions */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MIN(a,b) (((a)<(b))?(a):(b))

/* BLOCKING METRICS MULTIPLICATION ALGORITHM */
void main(int argc, char **argv)
{
int MAXLOOP = atoi(argv[1]);
int N=MAXLOOP;
int kk,jj,i,k,j,block=16;

int A[N][N];
int B[N][N];
int C[N][N];

for (i=0; i<N; i++) for (j=0; j<N; j++){ A[i][j]=j;} //assigning values
to metrics A
for (i=0; i<N; i++) for (j=0; j<N; j++){ B[i][j]=j;} //assigning values
to metrics B
for (i=0; i<N; i++) for (j=0; j<N; j++){ C[i][j]=j;} //assigning values
to metrics C

m5_dumpreset_stats(0,0);
m5_checkpoint(0,0);
/************************    Region    of    Interest    Starts    Here
*************************/
for (jj = 0; jj < N; jj += block)
{
    for (kk = 0; kk < N; kk += block)
    {
      for (i = 0; i < N; i += 1)
      {
          for (j = jj; j < MIN(jj + block, N); j++)
          {
            for (k = kk; k < MIN(kk + block, N); k++)
            {
                C[i][j]+=A[i][k]*B[k][j];
            }
           }
      }
    }
}
/************************    Region    of    Interest    Ends    Here
*************************/
m5_exit(0);
exit(0);                          /* No errors */
}
```

Appendix F: BASH SCRIPT 1 - runsim.sh

```
#!/bin/bash

SHOME="/home/feysal/gem5"
clear
#runsim.sh
################### (1)BASH SCRIPT SIM CONFIG ##############

INC_START=32  #Start matrix size 32x32
INC_STOP=305  #Last matrix size 304x304
INC_STEPS=16  #18 matrix size intervals: 32x32, 48x48 ... 304x304

SIM_FILE_NAME=$2  #reads  the  file  through  parameter  2:  example:
"mconv.c" #"mblock.c"

#########################################################################

COUNT=$INC_START
Nfiles=$INC_STOP
#

      rm multipalgorithm.c       #Temp  file  to  copy  the  code  to  for
compiling and then simulating
      cp SIMFILES/$SIM_FILE_NAME multipalgorithm.c  #enter the file to
be compiled and then simulated
      sleep 1

      #n=1
      simfile="output"

      make clean

      echo Starting compilation ...
      make EXEC=$simfile      #call makefile to compile the code


#########################################################
#     Running a number of simulation iterations     #
#########################################################
while [ $COUNT -lt $Nfiles ]; do

      echo Starting simulation [$COUNT] ...
      sleep 1

      rm -rf m5out/trial/cpt*
      rm m5out/trial/config*
      sleep 1
      ./build/X86/gem5.opt --outdir=m5out/trial configs/example/se.py -n
1 -o $COUNT -c ./output --caches --l2cache --l1i_size=4kB --l1i_assoc=4
--l1d_size=4kB  --l1d_assoc=4  --l2_size=256kB  --l2_assoc=4  --cpu-type
timing --clock=150MHz
      sleep 1
      ./build/X86/gem5.opt --outdir=m5out/trial configs/example/se.py -n
1 -o $COUNT -c ./output --caches --l2cache --l1i_size=4kB --l1i_assoc=4
--l1d_size=4kB  --l1d_assoc=4  --l2_size=256kB  --l2_assoc=4  --cpu-type
timing --clock=150MHz --checkpoint-dir=m5out/trial -r 1
```

```
      echo simulation ended successfully

      #Calling the second bash script: script name + iterations number+
start matrix size+C code file name
      source collectSats.sh $COUNT $INC_START $SIM_FILE_NAME

      let COUNT=COUNT+$INC_STEPS     #Count the number of application
iterations

      sleep 1
done



##--help -> lists the options available
##--debug-flag=list of flags that need to be traced
##--debug-flags=WorkItems
```

Appendix G: BASH SCRIPT 2 - collectStats.sh

```
#!/bin/bash
SHOME="/home/feysal/gem5"
fstatsSummary="${SHOME}/stats_summary.txt"


COUNT=$1                #Reading parameter from script 1
INC_START=$2            # --//--
SIM_FILE_NAME=$3 # --//--

# Function: searching and extracting digits from the stats file text #
function findDigits()
{
      #----------  findig digits in the stats file
      stringTmp=$(grep "$1" m5out/trial/stats.txt)
      stringTmp=${stringTmp//"l2"/"lt"}   #Changed l2 into lt to again
digit search function return the "2" after l in l2 cache string
      charCount="${stringTmp//[^#]}"
      if [ ${#charCount} -gt '1' ]; then
            string=${stringTmp:`expr index "$stringTmp" '#'`}
      else
            string=$stringTmp
      fi
      pattern1='([[:digit:]].[[:digit:]]+)'
      [[ $string =~ $pattern1  ]]
      tmpvar1=${BASH_REMATCH[1]}
      pattern2='([[:digit:]]+)'
      [[ $string =~ $pattern2  ]]
      tmpvar2=${BASH_REMATCH[1]}

      if [ -n "$tmpvar1" ]; then
            digits=$tmpvar1
        elif [ -n "$tmpvar2" ]; then

            digits=$tmpvar2
        else
            digits=0
      fi
}
#########################################################################


#%%%%%%%%%%%%%%%% Create empty stats_summary.txt file if it does not
exist %%%%%%%%%%%%%%%%%%%
echo "..............................."
echo "Saving statistics summary in stats_summary.txt "
sleep 1
fstatssearch=$(find -name $fstatsSummary)
if [ $fstatssearch and "./stats_summary.txt" ]   #save stats if file
found else create file then save stats
then
echo "Done ......"
else
touch $fstatsSummary   # create file if not found"
echo "Done ......"
fi
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
#%%%%%%%%  Passing search string to findDigits function %%%%%%%%
findDigits "sim_insts"
n_insts=$digits

findDigits "l2.overall_hits::switch_cpus.data"
l2hits=$digits
findDigits "l2.overall_misses::switch_cpus.data"
l2misses=$digits
findDigits "l2.overall_accesses::switch_cpus.data"
l2accesses=$digits
findDigits "l2.overall_miss_rate::switch_cpus.data"
l2miss_rate=$digits

findDigits "dcache.overall_hits::switch_cpus.data"
dhits=$digits
findDigits "dcache.overall_misses::switch_cpus.data"
dmisses=$digits
findDigits "dcache.overall_accesses::switch_cpus.data"
daccesses=$digits
findDigits "dcache.overall_miss_rate::switch_cpus.data"
dmiss_rate=$digits
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

#%%%%%%%%   Saving   the   collected/recalculated   data   into   the
stats_summary.txt file %%%%%%%%%%

if [ $daccesses -gt '0' ]; then
  dmiss_rate=$(bc <<< "scale = 6; $dmisses/$daccesses") #BC is used to
divide 2 variables and return the result with floating-point
else
  let dmiss_rate=0
fi


if [ $COUNT -eq $INC_START ]; then      #  Prints  the  headers  and  the
first line
  echo -e "\n$SIM_FILE_NAME ..." >> $fstatsSummary
  echo  -e  "N-Metrics\tL1-Hits\tL1-Misses\tL1-Accesses\tL1-Miss-rate\tN-
Instractions\tL2-Hits\tL2-Misses\tL2-Accesses\tL2-Miss-rate"          >>
$fstatsSummary
  echo                                                                -e
"$COUNT\t\t$dhits\t$dmisses\t\t$daccesses\t\t$dmiss_rate\t\t$n_insts\t\
t$l2hits\t$l2misses\t\t$l2accesses\t\t$l2miss_rate">>$fstatsSummary
else                          # Prints the remaining lines (if more than
1 line is to be printed)
  echo                                                                -e
"$COUNT\t\t$dhits\t$dmisses\t\t$daccesses\t\t$dmiss_rate\t\t$n_insts\t\
t$l2hits\t$l2misses\t\t$l2accesses\t\t$l2miss_rate">>$fstatsSummary
fi
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

echo "Stats summary saved in gem5/stats_summary.txt...."
sleep 1
```

Appendix H: MAKEFILE

```
#####################################################################


CC=gcc
#CC=gcc
CFLAGS= -D_M5 -O2 -g
#CFLAGS= -O2 -g
#LD_FLAGS= -lpthread
LD_FLAGS=--static
OUTPUT= output
OBJS= multipalgorithm.o
M5_OBJ=util/m5/m5op_x86.S
all: $(OUTPUT)

$(OUTPUT):$(OBJS)
	$(CC) -o $(OUTPUT) $(OBJS) $(M5_OBJ) $(LD_FLAGS)
	#$(CC) -o $(OUTPUT) $(OBJS) $(M5_OBJ) $(LD_FLAGS)

.c.o:
	$(CC) -c $(CFLAGS) $*.c

clean:
	rm *.o output


#####################################################################
```

Appendix I: gem5 – Simulation configuration file

config.ini

```
[root]
type=Root
children=system
full_system=false
time_sync_enable=false
time_sync_period=100000000000
time_sync_spin_threshold=100000000

[system]
type=System
children=cpu l2 membus physmem tol2bus
boot_osflags=a
clock=1000
init_param=0
kernel=
load_addr_mask=1099511627775
mem_mode=timing
mem_ranges=
memories=system.physmem
num_work_ids=16
readfile=
symbolfile=
work_begin_ckpt_count=0
work_begin_cpu_id_exit=-1
work_begin_exit_count=0
work_cpus_ckpt_count=0
work_end_ckpt_count=0
work_end_exit_count=0
work_item_id=-1
system_port=system.membus.slave[0]

[system.cpu]
type=TimingSimpleCPU
children=dcache dtb dtb_walker_cache icache interrupts isa itb
itb_walker_cache tracer workload
branchPred=Null
checker=Null
clock=6667
cpu_id=0
do_checkpoint_insts=true
do_quiesce=true
do_statistics_insts=true
dtb=system.cpu.dtb
function_trace=false
function_trace_start=0
interrupts=system.cpu.interrupts
isa=system.cpu.isa
itb=system.cpu.itb
max_insts_all_threads=0
max_insts_any_thread=0
max_loads_all_threads=0
max_loads_any_thread=0
numThreads=1
```

```
profile=0
progress_interval=0
switched_out=false
system=system
tracer=system.cpu.tracer
workload=system.cpu.workload
dcache_port=system.cpu.dcache.cpu_side
icache_port=system.cpu.icache.cpu_side

[system.cpu.dcache]
type=BaseCache
addr_ranges=0:18446744073709551615
assoc=4
block_size=64
clock=6667
forward_snoops=true
hit_latency=2
is_top_level=true
max_miss_count=0
mshrs=1
prefetch_on_access=false
prefetcher=Null
response_latency=2
size=4096
system=system
tgts_per_mshr=1
two_queue=false
write_buffers=8
cpu_side=system.cpu.dcache_port
mem_side=system.tol2bus.slave[1]

[system.cpu.dtb]
type=X86TLB
children=walker
size=64
walker=system.cpu.dtb.walker

[system.cpu.dtb.walker]
type=X86PagetableWalker
clock=6667
system=system
port=system.cpu.dtb_walker_cache.cpu_side

[system.cpu.dtb_walker_cache]
type=BaseCache
addr_ranges=0:18446744073709551615
assoc=2
block_size=64
clock=6667
forward_snoops=true
hit_latency=2
is_top_level=true
max_miss_count=0
mshrs=10
prefetch_on_access=false
prefetcher=Null
response_latency=2
size=1024
```

```
system=system
tgts_per_mshr=12
two_queue=false
write_buffers=8
cpu_side=system.cpu.dtb.walker.port
mem_side=system.tol2bus.slave[3]

[system.cpu.icache]
type=BaseCache
addr_ranges=0:18446744073709551615
assoc=4
block_size=64
clock=6667
forward_snoops=true
hit_latency=2
is_top_level=true
max_miss_count=0
mshrs=1
prefetch_on_access=false
prefetcher=Null
response_latency=2
size=4096
system=system
tgts_per_mshr=1
two_queue=false
write_buffers=8
cpu_side=system.cpu.icache_port
mem_side=system.tol2bus.slave[0]

[system.cpu.interrupts]
type=X86LocalApic
clock=106667
int_latency=1000
pio_addr=2305843009213693952
pio_latency=100000
system=system
int_master=system.membus.slave[2]
int_slave=system.membus.master[2]
pio=system.membus.master[1]

[system.cpu.isa]
type=X86ISA

[system.cpu.itb]
type=X86TLB
children=walker
size=64
walker=system.cpu.itb.walker

[system.cpu.itb.walker]
type=X86PagetableWalker
clock=6667
system=system
port=system.cpu.itb_walker_cache.cpu_side

[system.cpu.itb_walker_cache]
type=BaseCache
addr_ranges=0:18446744073709551615
```

```
assoc=2
block_size=64
clock=6667
forward_snoops=true
hit_latency=2
is_top_level=true
max_miss_count=0
mshrs=10
prefetch_on_access=false
prefetcher=Null
response_latency=2
size=1024
system=system
tgts_per_mshr=12
two_queue=false
write_buffers=8
cpu_side=system.cpu.itb.walker.port
mem_side=system.tol2bus.slave[2]

[system.cpu.tracer]
type=ExeTracer

[system.cpu.workload]
type=LiveProcess
cmd=./output 2
cwd=
egid=100
env=
errout=cerr
euid=100
executable=./output
gid=100
input=cin
max_stack_size=67108864
output=cout
pid=100
ppid=99
simpoint=0
system=system
uid=100

[system.l2]
type=BaseCache
addr_ranges=0:18446744073709551615
assoc=4
block_size=64
clock=6667
forward_snoops=true
hit_latency=20
is_top_level=false
max_miss_count=0
mshrs=1
prefetch_on_access=false
prefetcher=Null
response_latency=20
size=262144
system=system
tgts_per_mshr=1
```

```
two_queue=false
write_buffers=8
cpu_side=system.tol2bus.master[0]
mem_side=system.membus.slave[1]

[system.membus]
type=CoherentBus
block_size=64
clock=1000
header_cycles=1
system=system
use_default_range=false
width=8
master=system.physmem.port                          system.cpu.interrupts.pio
system.cpu.interrupts.int_slave
slave=system.system_port                                    system.l2.mem_side
system.cpu.interrupts.int_master

[system.physmem]
type=SimpleMemory
bandwidth=73.000000
clock=1000
conf_table_reported=false
in_addr_map=true
latency=30000
latency_var=0
null=false
range=0:536870911
zero=false
port=system.membus.master[0]

[system.tol2bus]
type=CoherentBus
block_size=64
clock=6667
header_cycles=1
system=system
use_default_range=false
width=32
master=system.l2.cpu_side
slave=system.cpu.icache.mem_side                  system.cpu.dcache.mem_side
system.cpu.itb_walker_cache.mem_side
system.cpu.dtb_walker_cache.mem_side
```

Appendix H: gem5 – Simulation stats file example

Stats.txt

```
---------- Begin Simulation Statistics ----------
sim_seconds                                                 0.000191
# Number of seconds simulated
sim_ticks                                                 190896211
# Number of ticks simulated
final_tick                                                190896211
# Number of ticks from beginning of simulation (restored from
checkpoints and never reset)
sim_freq                                             1000000000000
# Frequency of simulated ticks
host_inst_rate                                               66084
# Simulator instruction rate (inst/s)
host_op_rate                                                130339
# Simulator op (including micro ops) rate (op/s)
host_tick_rate                                           3573265852
# Simulator tick rate (ticks/s)
host_mem_usage                                              637244
# Number of bytes of host memory used
host_seconds                                                  0.05
# Real time elapsed on the host
sim_insts                                                     3524
# Number of instructions simulated
sim_ops                                                       6957
# Number of ops (including micro ops) simulated
system.physmem.bytes_read::cpu.inst                          12032
# Number of bytes read from this memory
system.physmem.bytes_read::cpu.data                           8512
# Number of bytes read from this memory
system.physmem.bytes_read::total                             20544
# Number of bytes read from this memory
system.physmem.bytes_inst_read::cpu.inst                     12032
# Number of instructions bytes read from this memory
system.physmem.bytes_inst_read::total                        12032
# Number of instructions bytes read from this memory
system.physmem.num_reads::cpu.inst                             188
# Number of read requests responded to by this memory
system.physmem.num_reads::cpu.data                             133
# Number of read requests responded to by this memory
system.physmem.num_reads::total                                321
# Number of read requests responded to by this memory
system.physmem.bw_read::cpu.inst                          63029014
# Total read bandwidth from this memory (bytes/s)
system.physmem.bw_read::cpu.data                          44589675
# Total read bandwidth from this memory (bytes/s)
system.physmem.bw_read::total                            107618689
# Total read bandwidth from this memory (bytes/s)
system.physmem.bw_inst_read::cpu.inst                     63029014
# Instruction read bandwidth from this memory (bytes/s)
system.physmem.bw_inst_read::total                        63029014
# Instruction read bandwidth from this memory (bytes/s)
system.physmem.bw_total::cpu.inst                         63029014
# Total bandwidth to/from this memory (bytes/s)
```

```
system.physmem.bw_total::cpu.data                            44589675
# Total bandwidth to/from this memory (bytes/s)
system.physmem.bw_total::total                              107618689
# Total bandwidth to/from this memory (bytes/s)
system.l2.replacements                                              0
# number of replacements
system.l2.tagsinuse                                        143.252491
# Cycle average of tags in use
system.l2.total_refs                                                0
# Total number of references to valid blocks.
system.l2.sampled_refs                                              0
# Sample count of references to valid blocks.
system.l2.avg_refs                                                nan
# Average number of references to valid blocks.
system.l2.warmup_cycle                                              0
# Cycle when the warmup percentage was hit.
system.l2.occ_blocks::writebacks                           14.403625
# Average occupied blocks per requestor
system.l2.occ_blocks::cpu.inst                             93.592710
# Average occupied blocks per requestor
system.l2.occ_blocks::cpu.data                             35.256156
# Average occupied blocks per requestor
system.l2.occ_percent::writebacks                           0.003517
# Average percentage of cache occupancy
system.l2.occ_percent::cpu.inst                             0.022850
# Average percentage of cache occupancy
system.l2.occ_percent::cpu.data                             0.008607
# Average percentage of cache occupancy
system.l2.occ_percent::total                                0.034974
# Average percentage of cache occupancy
system.l2.ReadReq_hits::cpu.inst                                   19
# number of ReadReq hits
system.l2.ReadReq_hits::cpu.data                                   24
# number of ReadReq hits
system.l2.ReadReq_hits::total                                      43
# number of ReadReq hits
system.l2.Writeback_hits::writebacks                               72
# number of Writeback hits
system.l2.Writeback_hits::total                                    72
# number of Writeback hits
system.l2.ReadExReq_hits::cpu.data                                  8
# number of ReadExReq hits
system.l2.ReadExReq_hits::total                                     8
# number of ReadExReq hits
system.l2.demand_hits::cpu.inst                                    19
# number of demand (read+write) hits
system.l2.demand_hits::cpu.data                                    32
# number of demand (read+write) hits
system.l2.demand_hits::total                                       51
# number of demand (read+write) hits
system.l2.overall_hits::cpu.inst                                   19
# number of overall hits
system.l2.overall_hits::cpu.data                                   32
# number of overall hits
system.l2.overall_hits::total                                      51
# number of overall hits
system.l2.ReadReq_misses::cpu.inst                                188
# number of ReadReq misses
```

```
system.l2.ReadReq_misses::cpu.data                 59
# number of ReadReq misses
system.l2.ReadReq_misses::total                   247
# number of ReadReq misses
system.l2.ReadExReq_misses::cpu.data               74
# number of ReadExReq misses
system.l2.ReadExReq_misses::total                  74
# number of ReadExReq misses
system.l2.demand_misses::cpu.inst                 188
# number of demand (read+write) misses
system.l2.demand_misses::cpu.data                 133
# number of demand (read+write) misses
system.l2.demand_misses::total                    321
# number of demand (read+write) misses
system.l2.overall_misses::cpu.inst                188
# number of overall misses
system.l2.overall_misses::cpu.data                133
# number of overall misses
system.l2.overall_misses::total                   321
# number of overall misses
system.l2.ReadReq_miss_latency::cpu.inst        56243734
# number of ReadReq miss cycles
system.l2.ReadReq_miss_latency::cpu.data        17647611
# number of ReadReq miss cycles
system.l2.ReadReq_miss_latency::total           73891345
# number of ReadReq miss cycles
system.l2.ReadExReq_miss_latency::cpu.data      22138329
# number of ReadExReq miss cycles
system.l2.ReadExReq_miss_latency::total         22138329
# number of ReadExReq miss cycles
system.l2.demand_miss_latency::cpu.inst         56243734
# number of demand (read+write) miss cycles
system.l2.demand_miss_latency::cpu.data         39785940
# number of demand (read+write) miss cycles
system.l2.demand_miss_latency::total            96029674
# number of demand (read+write) miss cycles
system.l2.overall_miss_latency::cpu.inst        56243734
# number of overall miss cycles
system.l2.overall_miss_latency::cpu.data        39785940
# number of overall miss cycles
system.l2.overall_miss_latency::total           96029674
# number of overall miss cycles
system.l2.ReadReq_accesses::cpu.inst              207
# number of ReadReq accesses(hits+misses)
system.l2.ReadReq_accesses::cpu.data               83
# number of ReadReq accesses(hits+misses)
system.l2.ReadReq_accesses::total                 290
# number of ReadReq accesses(hits+misses)
system.l2.Writeback_accesses::writebacks           72
# number of Writeback accesses(hits+misses)
system.l2.Writeback_accesses::total                72
# number of Writeback accesses(hits+misses)
system.l2.ReadExReq_accesses::cpu.data             82
# number of ReadExReq accesses(hits+misses)
system.l2.ReadExReq_accesses::total                82
# number of ReadExReq accesses(hits+misses)
system.l2.demand_accesses::cpu.inst               207
# number of demand (read+write) accesses
```

```
system.l2.demand_accesses::cpu.data                               165
# number of demand (read+write) accesses
system.l2.demand_accesses::total                                  372
# number of demand (read+write) accesses
system.l2.overall_accesses::cpu.inst                              207
# number of overall (read+write) accesses
system.l2.overall_accesses::cpu.data                              165
# number of overall (read+write) accesses
system.l2.overall_accesses::total                                 372
# number of overall (read+write) accesses
system.l2.ReadReq_miss_rate::cpu.inst                        0.908213
# miss rate for ReadReq accesses
system.l2.ReadReq_miss_rate::cpu.data                        0.710843
# miss rate for ReadReq accesses
system.l2.ReadReq_miss_rate::total                           0.851724
# miss rate for ReadReq accesses
system.l2.ReadExReq_miss_rate::cpu.data                      0.902439
# miss rate for ReadExReq accesses
system.l2.ReadExReq_miss_rate::total                         0.902439
# miss rate for ReadExReq accesses
system.l2.demand_miss_rate::cpu.inst                         0.908213
# miss rate for demand accesses
system.l2.demand_miss_rate::cpu.data                         0.806061
# miss rate for demand accesses
system.l2.demand_miss_rate::total                            0.862903
# miss rate for demand accesses
system.l2.overall_miss_rate::cpu.inst                        0.908213
# miss rate for overall accesses
system.l2.overall_miss_rate::cpu.data                        0.806061
# miss rate for overall accesses
system.l2.overall_miss_rate::total                           0.862903
# miss rate for overall accesses
system.l2.ReadReq_avg_miss_latency::cpu.inst            299168.797872
# average ReadReq miss latency
system.l2.ReadReq_avg_miss_latency::cpu.data            299112.050847
# average ReadReq miss latency
system.l2.ReadReq_avg_miss_latency::total               299155.242915
# average ReadReq miss latency
system.l2.ReadExReq_avg_miss_latency::cpu.data          299166.608108
# average ReadExReq miss latency
system.l2.ReadExReq_avg_miss_latency::total             299166.608108
# average ReadExReq miss latency
system.l2.demand_avg_miss_latency::cpu.inst             299168.797872
# average overall miss latency
system.l2.demand_avg_miss_latency::cpu.data             299142.406015
# average overall miss latency
system.l2.demand_avg_miss_latency::total                299157.862928
# average overall miss latency
system.l2.overall_avg_miss_latency::cpu.inst            299168.797872
# average overall miss latency
system.l2.overall_avg_miss_latency::cpu.data            299142.406015
# average overall miss latency
system.l2.overall_avg_miss_latency::total               299157.862928
# average overall miss latency
system.l2.blocked_cycles::no_mshrs                             8025
# number of cycles access was blocked
system.l2.blocked_cycles::no_targets                              0
# number of cycles access was blocked
```

```
system.l2.blocked::no_mshrs                                   321
# number of cycles access was blocked
system.l2.blocked::no_targets                                   0
# number of cycles access was blocked
system.l2.avg_blocked_cycles::no_mshrs                         25
# average number of cycles each access was blocked
system.l2.avg_blocked_cycles::no_targets                      nan
# average number of cycles each access was blocked
system.l2.fast_writes                                           0
# number of fast writes performed
system.l2.cache_copies                                          0
# number of cache copies performed
system.l2.ReadReq_mshr_misses::cpu.inst                       188
# number of ReadReq MSHR misses
system.l2.ReadReq_mshr_misses::cpu.data                        59
# number of ReadReq MSHR misses
system.l2.ReadReq_mshr_misses::total                          247
# number of ReadReq MSHR misses
system.l2.ReadExReq_mshr_misses::cpu.data                      74
# number of ReadExReq MSHR misses
system.l2.ReadExReq_mshr_misses::total                         74
# number of ReadExReq MSHR misses
system.l2.demand_mshr_misses::cpu.inst                        188
# number of demand (read+write) MSHR misses
system.l2.demand_mshr_misses::cpu.data                        133
# number of demand (read+write) MSHR misses
system.l2.demand_mshr_misses::total                           321
# number of demand (read+write) MSHR misses
system.l2.overall_mshr_misses::cpu.inst                       188
# number of overall MSHR misses
system.l2.overall_mshr_misses::cpu.data                       133
# number of overall MSHR misses
system.l2.overall_mshr_misses::total                          321
# number of overall MSHR misses
system.l2.ReadReq_mshr_miss_latency::cpu.inst             30707920
# number of ReadReq MSHR miss cycles
system.l2.ReadReq_mshr_miss_latency::cpu.data              9637060
# number of ReadReq MSHR miss cycles
system.l2.ReadReq_mshr_miss_latency::total                40344980
# number of ReadReq MSHR miss cycles
system.l2.ReadExReq_mshr_miss_latency::cpu.data           12087160
# number of ReadExReq MSHR miss cycles
system.l2.ReadExReq_mshr_miss_latency::total              12087160
# number of ReadExReq MSHR miss cycles
system.l2.demand_mshr_miss_latency::cpu.inst              30707920
# number of demand (read+write) MSHR miss cycles
system.l2.demand_mshr_miss_latency::cpu.data              21724220
# number of demand (read+write) MSHR miss cycles
system.l2.demand_mshr_miss_latency::total                 52432140
# number of demand (read+write) MSHR miss cycles
system.l2.overall_mshr_miss_latency::cpu.inst             30707920
# number of overall MSHR miss cycles
system.l2.overall_mshr_miss_latency::cpu.data             21724220
# number of overall MSHR miss cycles
system.l2.overall_mshr_miss_latency::total                52432140
# number of overall MSHR miss cycles
system.l2.ReadReq_mshr_miss_rate::cpu.inst                 0.908213
# mshr miss rate for ReadReq accesses
```

```
system.l2.ReadReq_mshr_miss_rate::cpu.data                  0.710843
# mshr miss rate for ReadReq accesses
system.l2.ReadReq_mshr_miss_rate::total                     0.851724
# mshr miss rate for ReadReq accesses
system.l2.ReadExReq_mshr_miss_rate::cpu.data                0.902439
# mshr miss rate for ReadExReq accesses
system.l2.ReadExReq_mshr_miss_rate::total                   0.902439
# mshr miss rate for ReadExReq accesses
system.l2.demand_mshr_miss_rate::cpu.inst                   0.908213
# mshr miss rate for demand accesses
system.l2.demand_mshr_miss_rate::cpu.data                   0.806061
# mshr miss rate for demand accesses
system.l2.demand_mshr_miss_rate::total                      0.862903
# mshr miss rate for demand accesses
system.l2.overall_mshr_miss_rate::cpu.inst                  0.908213
# mshr miss rate for overall accesses
system.l2.overall_mshr_miss_rate::cpu.data                  0.806061
# mshr miss rate for overall accesses
system.l2.overall_mshr_miss_rate::total                     0.862903
# mshr miss rate for overall accesses
system.l2.ReadReq_avg_mshr_miss_latency::cpu.inst             163340
# average ReadReq mshr miss latency
system.l2.ReadReq_avg_mshr_miss_latency::cpu.data             163340
# average ReadReq mshr miss latency
system.l2.ReadReq_avg_mshr_miss_latency::total                163340
# average ReadReq mshr miss latency
system.l2.ReadExReq_avg_mshr_miss_latency::cpu.data           163340
# average ReadExReq mshr miss latency
system.l2.ReadExReq_avg_mshr_miss_latency::total              163340
# average ReadExReq mshr miss latency
system.l2.demand_avg_mshr_miss_latency::cpu.inst              163340
# average overall mshr miss latency
system.l2.demand_avg_mshr_miss_latency::cpu.data              163340
# average overall mshr miss latency
system.l2.demand_avg_mshr_miss_latency::total                 163340
# average overall mshr miss latency
system.l2.overall_avg_mshr_miss_latency::cpu.inst             163340
# average overall mshr miss latency
system.l2.overall_avg_mshr_miss_latency::cpu.data             163340
# average overall mshr miss latency
system.l2.overall_avg_mshr_miss_latency::total                163340
# average overall mshr miss latency
system.l2.no_allocate_misses                                       0
# Number of misses that were no-allocate
system.cpu.workload.num_syscalls                                   6
# Number of system calls
system.cpu.numCycles                                           28633
# number of cpu cycles simulated
system.cpu.numWorkItemsStarted                                     0
# number of work items this cpu started
system.cpu.numWorkItemsCompleted                                   0
# number of work items this cpu completed
system.cpu.committedInsts                                       3524
# Number of instructions committed
system.cpu.committedOps                                         6957
# Number of ops (including micro ops) committed
system.cpu.num_int_alu_accesses                                 6850
# Number of integer alu accesses
```

```
system.cpu.num_fp_alu_accesses                                 123
# Number of float alu accesses
system.cpu.num_func_calls                                        0
# number of times a function call or return occured
system.cpu.num_conditional_control_insts                       545
# number of instructions that are conditional controls
system.cpu.num_int_insts                                      6850
# number of integer instructions
system.cpu.num_fp_insts                                        123
# number of float instructions
system.cpu.num_int_register_reads                            16884
# number of times the integer registers were read
system.cpu.num_int_register_writes                            7756
# number of times the integer registers were written
system.cpu.num_fp_register_reads                               199
# number of times the floating registers were read
system.cpu.num_fp_register_writes                               99
# number of times the floating registers were written
system.cpu.num_mem_refs                                       1285
# number of memory refs
system.cpu.num_load_insts                                      585
# Number of load instructions
system.cpu.num_store_insts                                     700
# Number of store instructions
system.cpu.num_idle_cycles                                       0
# Number of idle cycles
system.cpu.num_busy_cycles                                   28633
# Number of busy cycles
system.cpu.not_idle_fraction                                     1
# Percentage of non-idle cycles
system.cpu.idle_fraction                                         0
# Percentage of idle cycles
system.cpu.icache.replacements                                 143
# number of replacements
system.cpu.icache.tagsinuse                              51.798109
# Cycle average of tags in use
system.cpu.icache.total_refs                                  3277
# Total number of references to valid blocks.
system.cpu.icache.sampled_refs                                 143
# Sample count of references to valid blocks.
system.cpu.icache.avg_refs                               22.916084
# Average number of references to valid blocks.
system.cpu.icache.warmup_cycle                           125958340
# Cycle when the warmup percentage was hit.
system.cpu.icache.occ_blocks::cpu.inst                   51.798109
# Average occupied blocks per requestor
system.cpu.icache.occ_percent::cpu.inst                   0.809345
# Average percentage of cache occupancy
system.cpu.icache.occ_percent::total                      0.809345
# Average percentage of cache occupancy
system.cpu.icache.ReadReq_hits::cpu.inst                       4349
# number of ReadReq hits
system.cpu.icache.ReadReq_hits::total                          4349
# number of ReadReq hits
system.cpu.icache.demand_hits::cpu.inst                        4349
# number of demand (read+write) hits
system.cpu.icache.demand_hits::total                           4349
# number of demand (read+write) hits
```

```
system.cpu.icache.overall_hits::cpu.inst                         4349
# number of overall hits
system.cpu.icache.overall_hits::total                            4349
# number of overall hits
system.cpu.icache.ReadReq_misses::cpu.inst                        207
# number of ReadReq misses
system.cpu.icache.ReadReq_misses::total                           207
# number of ReadReq misses
system.cpu.icache.demand_misses::cpu.inst                         207
# number of demand (read+write) misses
system.cpu.icache.demand_misses::total                            207
# number of demand (read+write) misses
system.cpu.icache.overall_misses::cpu.inst                        207
# number of overall misses
system.cpu.icache.overall_misses::total                           207
# number of overall misses
system.cpu.icache.ReadReq_miss_latency::cpu.inst              67216694
# number of ReadReq miss cycles
system.cpu.icache.ReadReq_miss_latency::total                 67216694
# number of ReadReq miss cycles
system.cpu.icache.demand_miss_latency::cpu.inst               67216694
# number of demand (read+write) miss cycles
system.cpu.icache.demand_miss_latency::total                  67216694
# number of demand (read+write) miss cycles
system.cpu.icache.overall_miss_latency::cpu.inst              67216694
# number of overall miss cycles
system.cpu.icache.overall_miss_latency::total                 67216694
# number of overall miss cycles
system.cpu.icache.ReadReq_accesses::cpu.inst                      4556
# number of ReadReq accesses(hits+misses)
system.cpu.icache.ReadReq_accesses::total                         4556
# number of ReadReq accesses(hits+misses)
system.cpu.icache.demand_accesses::cpu.inst                       4556
# number of demand (read+write) accesses
system.cpu.icache.demand_accesses::total                          4556
# number of demand (read+write) accesses
system.cpu.icache.overall_accesses::cpu.inst                      4556
# number of overall (read+write) accesses
system.cpu.icache.overall_accesses::total                         4556
# number of overall (read+write) accesses
system.cpu.icache.ReadReq_miss_rate::cpu.inst                 0.045435
# miss rate for ReadReq accesses
system.cpu.icache.ReadReq_miss_rate::total                    0.045435
# miss rate for ReadReq accesses
system.cpu.icache.demand_miss_rate::cpu.inst                  0.045435
# miss rate for demand accesses
system.cpu.icache.demand_miss_rate::total                     0.045435
# miss rate for demand accesses
system.cpu.icache.overall_miss_rate::cpu.inst                 0.045435
# miss rate for overall accesses
system.cpu.icache.overall_miss_rate::total                    0.045435
# miss rate for overall accesses
system.cpu.icache.ReadReq_avg_miss_latency::cpu.inst      324718.328502
# average ReadReq miss latency
system.cpu.icache.ReadReq_avg_miss_latency::total         324718.328502
# average ReadReq miss latency
system.cpu.icache.demand_avg_miss_latency::cpu.inst       324718.328502
# average overall miss latency
```

```
system.cpu.icache.demand_avg_miss_latency::total          324718.328502
# average overall miss latency
system.cpu.icache.overall_avg_miss_latency::cpu.inst      324718.328502
# average overall miss latency
system.cpu.icache.overall_avg_miss_latency::total         324718.328502
# average overall miss latency
system.cpu.icache.blocked_cycles::no_mshrs                         9254
# number of cycles access was blocked
system.cpu.icache.blocked_cycles::no_targets                          0
# number of cycles access was blocked
system.cpu.icache.blocked::no_mshrs                                 207
# number of cycles access was blocked
system.cpu.icache.blocked::no_targets                                 0
# number of cycles access was blocked
system.cpu.icache.avg_blocked_cycles::no_mshrs                44.705314
# average number of cycles each access was blocked
system.cpu.icache.avg_blocked_cycles::no_targets                    nan
# average number of cycles each access was blocked
system.cpu.icache.fast_writes                                         0
# number of fast writes performed
system.cpu.icache.cache_copies                                        0
# number of cache copies performed
system.cpu.icache.ReadReq_mshr_misses::cpu.inst                     207
# number of ReadReq MSHR misses
system.cpu.icache.ReadReq_mshr_misses::total                        207
# number of ReadReq MSHR misses
system.cpu.icache.demand_mshr_misses::cpu.inst                      207
# number of demand (read+write) MSHR misses
system.cpu.icache.demand_mshr_misses::total                         207
# number of demand (read+write) MSHR misses
system.cpu.icache.overall_mshr_misses::cpu.inst                     207
# number of overall MSHR misses
system.cpu.icache.overall_mshr_misses::total                        207
# number of overall MSHR misses
system.cpu.icache.ReadReq_mshr_miss_latency::cpu.inst         61537332
# number of ReadReq MSHR miss cycles
system.cpu.icache.ReadReq_mshr_miss_latency::total            61537332
# number of ReadReq MSHR miss cycles
system.cpu.icache.demand_mshr_miss_latency::cpu.inst          61537332
# number of demand (read+write) MSHR miss cycles
system.cpu.icache.demand_mshr_miss_latency::total             61537332
# number of demand (read+write) MSHR miss cycles
system.cpu.icache.overall_mshr_miss_latency::cpu.inst         61537332
# number of overall MSHR miss cycles
system.cpu.icache.overall_mshr_miss_latency::total            61537332
# number of overall MSHR miss cycles
system.cpu.icache.ReadReq_mshr_miss_rate::cpu.inst            0.045435
# mshr miss rate for ReadReq accesses
system.cpu.icache.ReadReq_mshr_miss_rate::total               0.045435
# mshr miss rate for ReadReq accesses
system.cpu.icache.demand_mshr_miss_rate::cpu.inst             0.045435
# mshr miss rate for demand accesses
system.cpu.icache.demand_mshr_miss_rate::total                0.045435
# mshr miss rate for demand accesses
system.cpu.icache.overall_mshr_miss_rate::cpu.inst            0.045435
# mshr miss rate for overall accesses
system.cpu.icache.overall_mshr_miss_rate::total               0.045435
# mshr miss rate for overall accesses
```

```
system.cpu.icache.ReadReq_avg_mshr_miss_latency::cpu.inst  297281.797101
# average ReadReq mshr miss latency
system.cpu.icache.ReadReq_avg_mshr_miss_latency::total     297281.797101
# average ReadReq mshr miss latency
system.cpu.icache.demand_avg_mshr_miss_latency::cpu.inst   297281.797101
# average overall mshr miss latency
system.cpu.icache.demand_avg_mshr_miss_latency::total      297281.797101
# average overall mshr miss latency
system.cpu.icache.overall_avg_mshr_miss_latency::cpu.inst  297281.797101
# average overall mshr miss latency
system.cpu.icache.overall_avg_mshr_miss_latency::total     297281.797101
# average overall mshr miss latency
system.cpu.icache.no_allocate_misses                                0
# Number of misses that were no-allocate
system.cpu.itb_walker_cache.replacements                            0
# number of replacements
system.cpu.itb_walker_cache.tagsinuse                               0
# Cycle average of tags in use
system.cpu.itb_walker_cache.total_refs                              0
# Total number of references to valid blocks.
system.cpu.itb_walker_cache.sampled_refs                            0
# Sample count of references to valid blocks.
system.cpu.itb_walker_cache.avg_refs                              nan
# Average number of references to valid blocks.
system.cpu.itb_walker_cache.warmup_cycle                            0
# Cycle when the warmup percentage was hit.
system.cpu.itb_walker_cache.blocked_cycles::no_mshrs                0
# number of cycles access was blocked
system.cpu.itb_walker_cache.blocked_cycles::no_targets              0
# number of cycles access was blocked
system.cpu.itb_walker_cache.blocked::no_mshrs                       0
# number of cycles access was blocked
system.cpu.itb_walker_cache.blocked::no_targets                     0
# number of cycles access was blocked
system.cpu.itb_walker_cache.avg_blocked_cycles::no_mshrs           nan
# average number of cycles each access was blocked
system.cpu.itb_walker_cache.avg_blocked_cycles::no_targets         nan
# average number of cycles each access was blocked
system.cpu.itb_walker_cache.fast_writes                             0
# number of fast writes performed
system.cpu.itb_walker_cache.cache_copies                            0
# number of cache copies performed
system.cpu.itb_walker_cache.no_allocate_misses                      0
# Number of misses that were no-allocate
system.cpu.dtb_walker_cache.replacements                            0
# number of replacements
system.cpu.dtb_walker_cache.tagsinuse                               0
# Cycle average of tags in use
system.cpu.dtb_walker_cache.total_refs                              0
# Total number of references to valid blocks.
system.cpu.dtb_walker_cache.sampled_refs                            0
# Sample count of references to valid blocks.
system.cpu.dtb_walker_cache.avg_refs                              nan
# Average number of references to valid blocks.
system.cpu.dtb_walker_cache.warmup_cycle                            0
# Cycle when the warmup percentage was hit.
system.cpu.dtb_walker_cache.blocked_cycles::no_mshrs                0
# number of cycles access was blocked
```

```
system.cpu.dtb_walker_cache.blocked_cycles::no_targets                  0
# number of cycles access was blocked
system.cpu.dtb_walker_cache.blocked::no_mshrs                           0
# number of cycles access was blocked
system.cpu.dtb_walker_cache.blocked::no_targets                         0
# number of cycles access was blocked
system.cpu.dtb_walker_cache.avg_blocked_cycles::no_mshrs               nan
# average number of cycles each access was blocked
system.cpu.dtb_walker_cache.avg_blocked_cycles::no_targets             nan
# average number of cycles each access was blocked
system.cpu.dtb_walker_cache.fast_writes                                 0
# number of fast writes performed
system.cpu.dtb_walker_cache.cache_copies                                0
# number of cache copies performed
system.cpu.dtb_walker_cache.no_allocate_misses                          0
# Number of misses that were no-allocate
system.cpu.dcache.replacements                                        101
# number of replacements
system.cpu.dcache.tagsinuse                                     50.901255
# Cycle average of tags in use
system.cpu.dcache.total_refs                                          573
# Total number of references to valid blocks.
system.cpu.dcache.sampled_refs                                        101
# Sample count of references to valid blocks.
system.cpu.dcache.avg_refs                                       5.673267
# Average number of references to valid blocks.
system.cpu.dcache.warmup_cycle                                  96330340
# Cycle when the warmup percentage was hit.
system.cpu.dcache.occ_blocks::cpu.data                          50.901255
# Average occupied blocks per requestor
system.cpu.dcache.occ_percent::cpu.data                          0.795332
# Average percentage of cache occupancy
system.cpu.dcache.occ_percent::total                             0.795332
# Average percentage of cache occupancy
system.cpu.dcache.ReadReq_hits::cpu.data                              502
# number of ReadReq hits
system.cpu.dcache.ReadReq_hits::total                                 502
# number of ReadReq hits
system.cpu.dcache.WriteReq_hits::cpu.data                             618
# number of WriteReq hits
system.cpu.dcache.WriteReq_hits::total                                618
# number of WriteReq hits
system.cpu.dcache.demand_hits::cpu.data                              1120
# number of demand (read+write) hits
system.cpu.dcache.demand_hits::total                                 1120
# number of demand (read+write) hits
system.cpu.dcache.overall_hits::cpu.data                             1120
# number of overall hits
system.cpu.dcache.overall_hits::total                                1120
# number of overall hits
system.cpu.dcache.ReadReq_misses::cpu.data                             83
# number of ReadReq misses
system.cpu.dcache.ReadReq_misses::total                                83
# number of ReadReq misses
system.cpu.dcache.WriteReq_misses::cpu.data                            82
# number of WriteReq misses
system.cpu.dcache.WriteReq_misses::total                               82
# number of WriteReq misses
```

```
system.cpu.dcache.demand_misses::cpu.data                       165
# number of demand (read+write) misses
system.cpu.dcache.demand_misses::total                          165
# number of demand (read+write) misses
system.cpu.dcache.overall_misses::cpu.data                      165
# number of overall misses
system.cpu.dcache.overall_misses::total                         165
# number of overall misses
system.cpu.dcache.ReadReq_miss_latency::cpu.data              24221211
# number of ReadReq miss cycles
system.cpu.dcache.ReadReq_miss_latency::total                 24221211
# number of ReadReq miss cycles
system.cpu.dcache.WriteReq_miss_latency::cpu.data             26547994
# number of WriteReq miss cycles
system.cpu.dcache.WriteReq_miss_latency::total                26547994
# number of WriteReq miss cycles
system.cpu.dcache.demand_miss_latency::cpu.data               50769205
# number of demand (read+write) miss cycles
system.cpu.dcache.demand_miss_latency::total                  50769205
# number of demand (read+write) miss cycles
system.cpu.dcache.overall_miss_latency::cpu.data              50769205
# number of overall miss cycles
system.cpu.dcache.overall_miss_latency::total                 50769205
# number of overall miss cycles
system.cpu.dcache.ReadReq_accesses::cpu.data                      585
# number of ReadReq accesses(hits+misses)
system.cpu.dcache.ReadReq_accesses::total                         585
# number of ReadReq accesses(hits+misses)
system.cpu.dcache.WriteReq_accesses::cpu.data                     700
# number of WriteReq accesses(hits+misses)
system.cpu.dcache.WriteReq_accesses::total                        700
# number of WriteReq accesses(hits+misses)
system.cpu.dcache.demand_accesses::cpu.data                      1285
# number of demand (read+write) accesses
system.cpu.dcache.demand_accesses::total                         1285
# number of demand (read+write) accesses
system.cpu.dcache.overall_accesses::cpu.data                     1285
# number of overall (read+write) accesses
system.cpu.dcache.overall_accesses::total                        1285
# number of overall (read+write) accesses
system.cpu.dcache.ReadReq_miss_rate::cpu.data                0.141880
# miss rate for ReadReq accesses
system.cpu.dcache.ReadReq_miss_rate::total                   0.141880
# miss rate for ReadReq accesses
system.cpu.dcache.WriteReq_miss_rate::cpu.data               0.117143
# miss rate for WriteReq accesses
system.cpu.dcache.WriteReq_miss_rate::total                  0.117143
# miss rate for WriteReq accesses
system.cpu.dcache.demand_miss_rate::cpu.data                 0.128405
# miss rate for demand accesses
system.cpu.dcache.demand_miss_rate::total                    0.128405
# miss rate for demand accesses
system.cpu.dcache.overall_miss_rate::cpu.data                0.128405
# miss rate for overall accesses
system.cpu.dcache.overall_miss_rate::total                   0.128405
# miss rate for overall accesses
system.cpu.dcache.ReadReq_avg_miss_latency::cpu.data      291821.819277
# average ReadReq miss latency
```

```
system.cpu.dcache.ReadReq_avg_miss_latency::total          291821.819277
# average ReadReq miss latency
system.cpu.dcache.WriteReq_avg_miss_latency::cpu.data      323756.024390
# average WriteReq miss latency
system.cpu.dcache.WriteReq_avg_miss_latency::total         323756.024390
# average WriteReq miss latency
system.cpu.dcache.demand_avg_miss_latency::cpu.data        307692.151515
# average overall miss latency
system.cpu.dcache.demand_avg_miss_latency::total           307692.151515
# average overall miss latency
system.cpu.dcache.overall_avg_miss_latency::cpu.data       307692.151515
# average overall miss latency
system.cpu.dcache.overall_avg_miss_latency::total          307692.151515
# average overall miss latency
system.cpu.dcache.blocked_cycles::no_mshrs                        6955
# number of cycles access was blocked
system.cpu.dcache.blocked_cycles::no_targets                         0
# number of cycles access was blocked
system.cpu.dcache.blocked::no_mshrs                                165
# number of cycles access was blocked
system.cpu.dcache.blocked::no_targets                                0
# number of cycles access was blocked
system.cpu.dcache.avg_blocked_cycles::no_mshrs                42.151515
# average number of cycles each access was blocked
system.cpu.dcache.avg_blocked_cycles::no_targets                   nan
# average number of cycles each access was blocked
system.cpu.dcache.fast_writes                                        0
# number of fast writes performed
system.cpu.dcache.cache_copies                                       0
# number of cache copies performed
system.cpu.dcache.writebacks::writebacks                            72
# number of writebacks
system.cpu.dcache.writebacks::total                                 72
# number of writebacks
system.cpu.dcache.ReadReq_mshr_misses::cpu.data                     83
# number of ReadReq MSHR misses
system.cpu.dcache.ReadReq_mshr_misses::total                        83
# number of ReadReq MSHR misses
system.cpu.dcache.WriteReq_mshr_misses::cpu.data                    82
# number of WriteReq MSHR misses
system.cpu.dcache.WriteReq_mshr_misses::total                       82
# number of WriteReq MSHR misses
system.cpu.dcache.demand_mshr_misses::cpu.data                     165
# number of demand (read+write) MSHR misses
system.cpu.dcache.demand_mshr_misses::total                        165
# number of demand (read+write) MSHR misses
system.cpu.dcache.overall_mshr_misses::cpu.data                    165
# number of overall MSHR misses
system.cpu.dcache.overall_mshr_misses::total                       165
# number of overall MSHR misses
system.cpu.dcache.ReadReq_mshr_miss_latency::cpu.data          21954493
# number of ReadReq MSHR miss cycles
system.cpu.dcache.ReadReq_mshr_miss_latency::total             21954493
# number of ReadReq MSHR miss cycles
system.cpu.dcache.WriteReq_mshr_miss_latency::cpu.data         24298437
# number of WriteReq MSHR miss cycles
system.cpu.dcache.WriteReq_mshr_miss_latency::total            24298437
# number of WriteReq MSHR miss cycles
```

```
system.cpu.dcache.demand_mshr_miss_latency::cpu.data            46252930
# number of demand (read+write) MSHR miss cycles
system.cpu.dcache.demand_mshr_miss_latency::total               46252930
# number of demand (read+write) MSHR miss cycles
system.cpu.dcache.overall_mshr_miss_latency::cpu.data           46252930
# number of overall MSHR miss cycles
system.cpu.dcache.overall_mshr_miss_latency::total              46252930
# number of overall MSHR miss cycles
system.cpu.dcache.ReadReq_mshr_miss_rate::cpu.data              0.141880
# mshr miss rate for ReadReq accesses
system.cpu.dcache.ReadReq_mshr_miss_rate::total                 0.141880
# mshr miss rate for ReadReq accesses
system.cpu.dcache.WriteReq_mshr_miss_rate::cpu.data             0.117143
# mshr miss rate for WriteReq accesses
system.cpu.dcache.WriteReq_mshr_miss_rate::total                0.117143
# mshr miss rate for WriteReq accesses
system.cpu.dcache.demand_mshr_miss_rate::cpu.data               0.128405
# mshr miss rate for demand accesses
system.cpu.dcache.demand_mshr_miss_rate::total                  0.128405
# mshr miss rate for demand accesses
system.cpu.dcache.overall_mshr_miss_rate::cpu.data              0.128405
# mshr miss rate for overall accesses
system.cpu.dcache.overall_mshr_miss_rate::total                 0.128405
# mshr miss rate for overall accesses
system.cpu.dcache.ReadReq_avg_mshr_miss_latency::cpu.data  264511.963855
# average ReadReq mshr miss latency
system.cpu.dcache.ReadReq_avg_mshr_miss_latency::total     264511.963855
# average ReadReq mshr miss latency
system.cpu.dcache.WriteReq_avg_mshr_miss_latency::cpu.data
296322.402439                               # average WriteReq mshr miss
latency
system.cpu.dcache.WriteReq_avg_mshr_miss_latency::total    296322.402439
# average WriteReq mshr miss latency
system.cpu.dcache.demand_avg_mshr_miss_latency::cpu.data   280320.787879
# average overall mshr miss latency
system.cpu.dcache.demand_avg_mshr_miss_latency::total      280320.787879
# average overall mshr miss latency
system.cpu.dcache.overall_avg_mshr_miss_latency::cpu.data  280320.787879
# average overall mshr miss latency
system.cpu.dcache.overall_avg_mshr_miss_latency::total     280320.787879
# average overall mshr miss latency
system.cpu.dcache.no_allocate_misses                                  0
# Number of misses that were no-allocate

---------- End Simulation Statistics   ----------
```