Deanship of Graduate Studies

Al-Quds University

A Non-blocking Interconnection Network-Shared Cache Organization for Multi-core Processors

Allam Rebhi Mohammad Abumwais

M.Sc Thesis

Jerusalem-Palestine

1434-2013

A Non-blocking Interconnection Network-Shared Cache Organization for Multi-core Processors

Prepared By:

Allam Rebhi Mohammad AbuMwais

B.Sc.: Computer Engineering, 2013, Palestine

Alquds University, Palestine

Supervisor: Dr. Abdualkareem Ayyad

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Electronic and Computer Engineering/Department of Electronic and Computer Engineering/ Faculty of Engineering/ Graduate Studies

Jerusalem-Palestine

1434-2013

Al-Quds University

Deanship of Graduate Studies

Master of Electronics and Computer Engineering

Thesis Approval

A Non-blocking Interconnection Network-Shared Cache Organization for Multi-core Processors

Prepared By: Allam Rebhi Mohammad AbuMwais

Registration No: 21011963

Supervisor: Dr. Abdulkareem Ayyad

Master thesis submitted and accepted, Date: 00/00/2013

The name and signatures of examining committee members are as follows:

1- Head of committee	Dr. Abdulkareem Ayyad	Signature: …….
2- Internal Examiner	Dr.	Signature: …….
3- External Examiner	Dr.	Signature: …….

Jerusalem-Palestine

1434-2013

Declaration:

I certify that this thesis submitted for the degree of Master, is the result of my own research, except where otherwise acknowledged, and that this study (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed: ……

Allam Rebhi Mohammad Abu Mwais

Date: 28/9/2013

Acknowledgments

I thank God for everything.

I would like to express my deepest appreciation to my family especially my parents, for the endless care they have provided me, advices, support and patience.

Special thanks goes out to my supervisor, Dr. Abdulkareem Ayyad, whose give me more knowledge and skills in the computer architecture field. Also, his guidance helped me throughout this research and the writing of this thesis.

My deep gratitude goes to all my friends, especially at Al-Quds University.

I would like to thank all those who helped me during the period of my study and give me motivation and encouragement.

Abstract

In modern on-chip multi-core processors and multiprocessor systems, the communication between the processor cores and the shared memory modules of the system, (here in after, the terms ‘shared memory’ and ‘shared cache’ can be used interchangeably), suffers from a bottleneck problem. In the best interconnection network, the crossbar switch, when two or more cores make a request to access the same shared cache module, only one request will be accepted and the other requests have to be honoured in a sequence decided by the arbiter of that module. This increases the latency of the shared variable access. This considerably slows the performance of the cores in executing the program threads and hence, the whole execution process.

In our proposed model for multi-core processor architecture, we have redesigned the shared cache modules and the interconnection network organization. This resulted in a Multi-port Content Addressable Memory (MPCAM) and the bottleneck has been totally eliminated. All the cores of the system can write to and read from this memory simultaneously. The shared variable communication process through this memory, by itself guarantees the snooping cache coherence process automatically. It worth noting that the cache coherence process increases the communication overhead in the current systems. In this organization, there is no queuing, no arbitration, and hence no additional latency. A latency of less than or equal five nanoseconds per shared variable access has been achieved. The simulation results of the MPCAM as part of a multi-core architecture have shown high bandwidth, and negligible cache miss ratio, negligible cache coherence and synchronization overhead as compared to the eight core AMD architecture. At the end of this work, the authors have proposed a linear scalable scheme which expands the system to include large number of cores with

linear growing cost and minimum fixed latency of 1.5t, where t is the MPCAM access time.

Keywords: Multi-core, Multi-threading, Micro-threading, Content Addressable Memory (CAM), Cache Coherence, shared Variables, Interconnection Network, Bottleneck, Contention, Arbitration.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In shared memory multiprocessor systems, there are three crucial issues to be considered; The interconnection networks and the communication bottleneck problem, the scheduling policy which needs load balancing and true dependency, and the cache coherence problem[Joh07][Chi06][jess96]. All these issues apply equally to the on chip shared memory multi-core processors. Multi-core processors are multiple pipelined processors on a chip. Symmetric shared memory multi-core processors exchange information via shared variables. Shared variables reside in shared memory modules and are accessible by all cores. Accessing the shared memory without delay necessities the existence of efficient interconnection network. Any bottleneck in the network causes delay and hence, degrades the performance of the multiprocessor/multi-core system. Provided that dependence rules are observed in the scheduling policy, the processor needs to read/write the right version of the data. This needs synchronization among the processors through hardware and software cache coherence protocols. Synchronization and cache coherence operations add burden on the communication in the interconnection network. The ideal multiprocessor case occur

when the software perfectly observes the dependence rules and load balancing, and when there is no communication overhead due to the bottleneck in the network and cache coherence operations.

In this thesis we present multi-core architecture to solve the communication among cores and between shared memories, this architecture totally eliminates the bottleneck on the interconnection networks and need for arbitration. It also eliminates the need for cache coherence operations. We can say that, if the program is carefully partitioned and scheduled, it nearly can be executed in time equals to 1/n of the execution time on a single processor, i.e., the speed up of the system is ~n, where n is the number of processors (cores).

1.2 Problem Specification and Justification

Scheduling policy is other problems were migrated from discrete multiprocessor to multi-core processor. Scheduling policy is concerned with dividing program among the processors to be executed in shortest possible time. Scheduling problem will be discussed in next chapter. Scheduling involves partitioning the program into chunks (nodes, grains) and scheduling these chunks to the processors without violating the dependence rules. The nodes have local and shared variable. The local variable is accessible by its node only, whereas the shared variable is accessible by all nodes which need them. Access by number of node must be synchronized so that the node reads or writes the correct version of data. Hardware synchronization and software cache coherence protocols are needed.

All multi-core architecture has distributed cache Level, level1 (L1), level2 (L2) and in some cases level3 (L3) caches which must be coordinated. Multi-core architecture uses

shared caches among the cores. This led us to the last and main problem which is the communication among the cores through accessing shared variables.

In the next chapter we will discussed these two problems; the communication via the interconnection network and the scheduling including the cache coherence operations. Then the proposed architecture will be justified in details. A primary justification is the elimination of these problems and their overhead.

1.3 Thesis Contribution

In modern on-chip multi-core processor systems [Intel12, AMD07, Joh07, Tor10], the communication between the processor cores and the shared cache modules of the system suffers from a bottleneck problem that negatively affects the performance of system. So we proposed a multi-pipelined processor on a chip to solve these problems. In this model we have designed a new interconnection networks based of small cache organization modules that are embedded at the cross points of the crossbar network. It provides a non blocking communication among the cores of the system and the shared memory module including snooping cache coherence at the same time. This organization, i.e., the network plus the dual port CAM (DPCAM) modules, has formed a multiport content addressable memory (MPCAM).

1.4 Thesis Organization

The remaining chapter in this thesis are organized as follows. In the chapter 2, we will discuss the modern multi-core system architecture and analyse its main problems; the contention in the interconnection network, the scheduling policy and the cache coherence. In the chapter 3, we will present the above design, its mathematical model and the results

of its performance as standalone component. In chapter 4, we will display the simulation results within the multi-core architecture and compare it with AMD multi-core architecture. In chapter 5, conclusion and future work will be drawn.

Chapter 2

Background and Related Work

2.1 Current Approach

2.1.1 Multiprocessors Systems

The main idea for using the Multiple Processors (MP) is to increase the performance of system. It presents a great hope for massive parallel processing. The multiprocessor computer system is the use of two or more central processing units (CPUs) within a single computer system.

According to Flynn classification of computers, multiprocessor systems are classified as Multiple Instruction Stream Multiple Data Stream computers (MIMD) [Barry96]. In this class of Computers, a number of processors can execute multiple streams of program instructions which operate on a number of different data streams simultaneously.

Multiprocessor systems are divided into two major classes; tightly coupled and loosely coupled [Joh07]. Loosely coupled multiprocessors are run on multicomputer system and use message passing techniques over the computer networks to communicate among the processes running on different computers. Tightly coupled multiprocessors [M.Ab12] is widely used in multiprocessor computer architecture. Tightly coupled mostly execute single program on a number of processors that share a space of memory of the same computer. Communication among the processes on different processors takes place via

shared variable in the shared memory. These variables are accessible by all the processors of the system.

Multiprocessor power can, theoretically, be expanded to any scale we like. The processing power of a multiprocessor System can be increased; simply by adding more processing elements (PEs) to the system. However, expanding a multiprocessor system is not as straightforward as it looks. There are many issues usually have to be resolved. In this thesis we have proposed design and simulate new shared cache modules and interconnection network organization to improve the communication between PEs (cores) and to eliminate the bottleneck. The network plus the memory modules result in a new component called Multi-Port Content Addressable Memory MPCAM. Used as a second level shared cache in the multi-core processor system.

2.1.2 Multi-core System

With the tremendous advent of semiconductor technology in the last two decades, Integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip is called system on a chip (SoC). As a result, it became possible to implement a number of pipelined processors (cores) with their caches and interconnection network on one chip named multi-core processor. All pros and cons of multiprocessors have moved with them from on-board to on-chip. They have the same architectural issues problems. The two extra problems is the space allowed on the chip because as Moore's Law said (the doubling of transistors on chip every 18 months) [Moo74] and the power temperature consumption of multiple core on a single chip, which limits the number of implemented processors. These extra problems tended to be solved in recent years.

2.2 Alternate Approach

2.2.1 Multi-core Processor Architecture

As we explained on a chip multiprocessors (OCM), known as multi-core processor. This involves a number of pipelined processor implemented on a single chip. This can be considered as tightly-coupled multiprocessor on a chip. Mainframe systems with multiple processors are often tightly-coupled [M.Ab12].

Multi-core systems architecture is divided into two architecture classes; Symmetric Shared Memory (SSM) systems and Distributed Shared Memory (DSM) systems. These two classes will be discussed later.

In OCM systems, three major issues are of prime concern; they have the same architectural problems as discrete multiprocessor system, namely, the effective scheduling policy and the effective communication among the processor elements PEs of the system. A third issue is the cache coherence among the processors of the system.

2.2.1.1 The SSM Multi-core Processor Architecture

In this type of architecture the cores and the shared memory modules are placed on different sides of the interconnection network. Each core includes a pipelined processor and one or two levels of local cache. Figure (2.1-a) depicts a block diagram of a typical SSM system, and figure (2.1-b) depicts a processing element of the system.

Going through this architecture, we can note the following things:

1- Any processor can access any shared memory module through the supposed interconnection network.

2- Even with the best network (crossbar switch) there is still a bottleneck when more than one processor tries to access the same module at the same time.

3- The cache coherence policy puts a severe overhead and bottleneck on the network. Only one processor can broadcast the latest version of the data at a time. All these issues will increase the latency [Joh07].

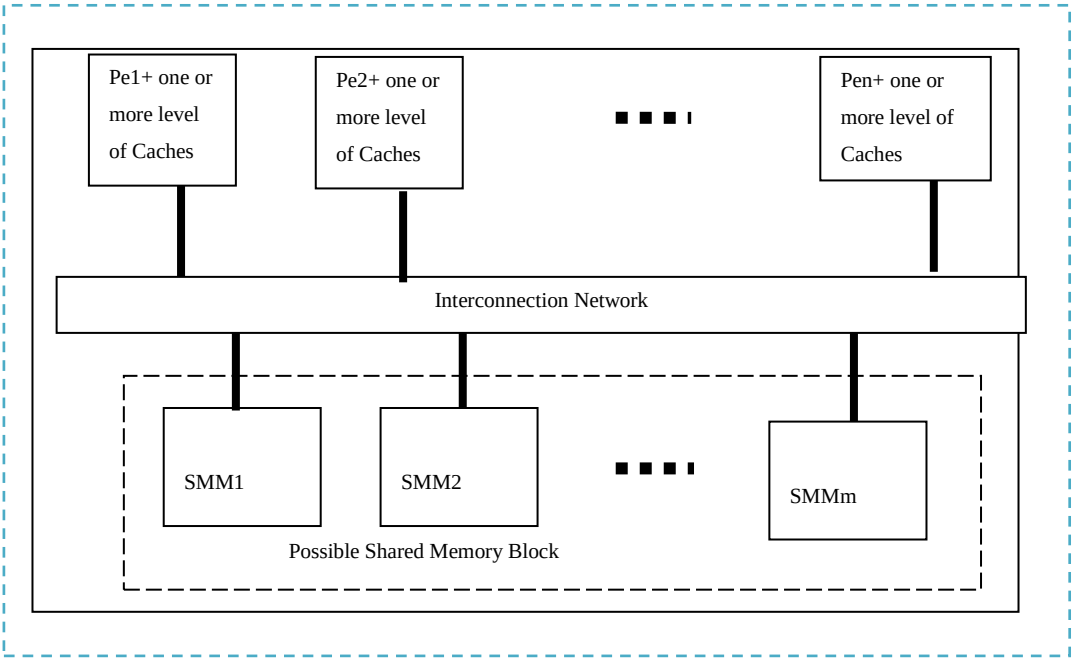

Figure 2.1(a): A Block Diagram of SMM System [Joh07].

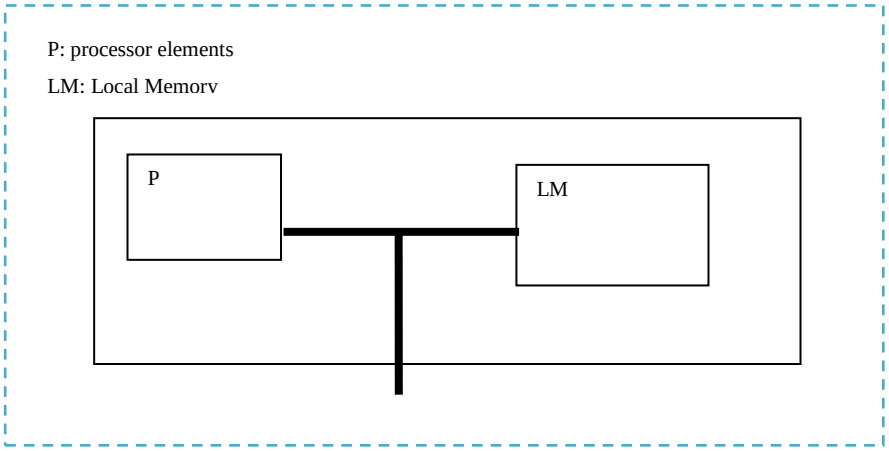

Figure 2.1(b): Processing Elements (PEs) [Barry96].

2.2.1.2 DSM Multi-core Processor Architecture

In this architecture, each core has its own part of shared cache in addition to its local caches connected to the core local bus as shown in figure 2.2. A survey of these systems is shown in [Joh07] [Pro96]. All cores are connected to the interconnection network. The computational task must communicate with one or more remote processors. In this architecture, we can note the following:

1- Each processor can access any shared memory which belongs to other processor via the interconnection network.

2- This architecture also has the same communication bottleneck (even for best interconnection network; the crossbar switch).

3- Also the cache coherence presents a heavy overhead on the network.

New DSM architectures are organized called a hybrid Distributed Shared Memory. The main goal of hybrid DSM organization is to support fast physical memory accesses for private data [Xia11].

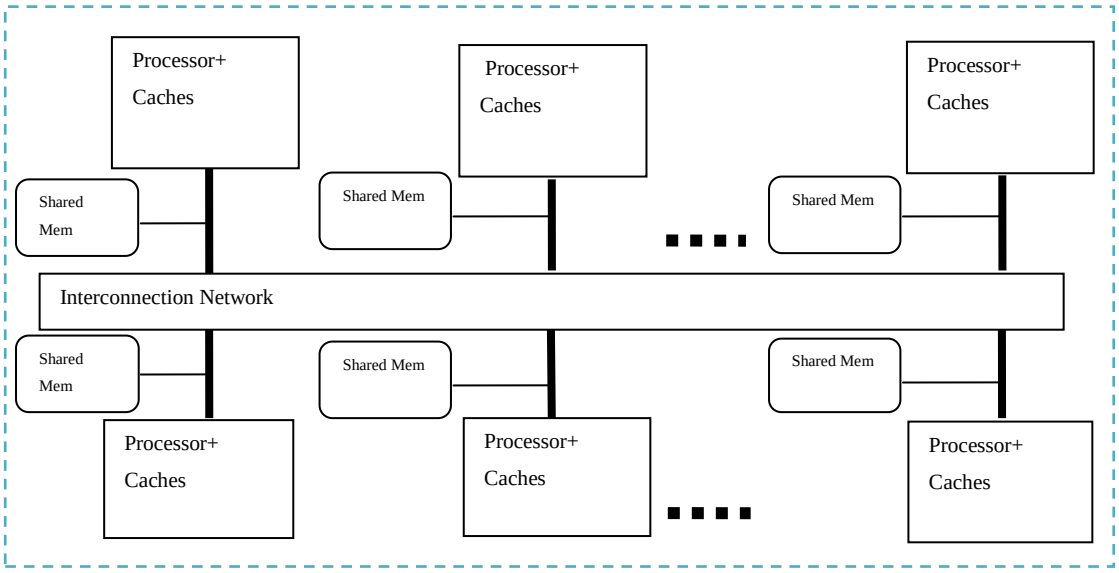

Figure 2.2: A block Diagram of DSM System [Joh07].

2.2.2 Multithreading and Micro-threading

In Von Neumann model architecture, the instruction is executed when its turn comes on the program counter, whereas in dataflow model, the instruction becomes ready to execute when its operands are available. If dataflow is applied on the instruction level, massive parallelism is expected. Applying dataflow model on the instructional level is referred to as fine grain dataflow, and produces high degree of parallelism [D.E99]. In early eighties, several dataflow multiprocessor systems were built in USA and Europe. However, the results represented disappointment to the builders of these machines. This is because applying parallelism on instructional level necessitates exchanging large number of variables among the system PEs through the interconnection network. This leads to a network congestion, which means a large delay in communicating the variables, hence a large delay in executing the supposed ready-to-execute instructions [Barry96] [Fira01]. The result is a whole degradation of parallelism of the system. Large grain dataflow represented a solution for the congestion problem. In this model, the level of parallelism is increased to the grain level where the grain is part of the program of a function, a process or, a task level. The grain includes a number of instructions that are executed sequentially depending on an internal program counter to the grain. A number of independent grains of the program can be executed in parallel. The grain is sent to the queue when its arguments are available. The grains exchange the shared arguments through the interconnection network. The production of arguments is less frequent than the production of operands in the instruction level parallelism. This represents less pressure and less congestion on the network. The optimisation of the grain size is an issue by itself. One needs to choose the size which suits the machine on which he runs the program. Here, we need to introduce multithreading concept before we continue to the micro-threading.

Multithreading was introduced on a single pipelined processor in order to avoid out-of-order and interrupt problems in the pipelined processor. A number of instructions from different programs (threads) equals to the number of the stages of the pipeline are interleaved in the pipeline so that there is no dependency among the instructions [jess01]. This is simply because the instructions belong to different programs (threads). (See figure 2.3).

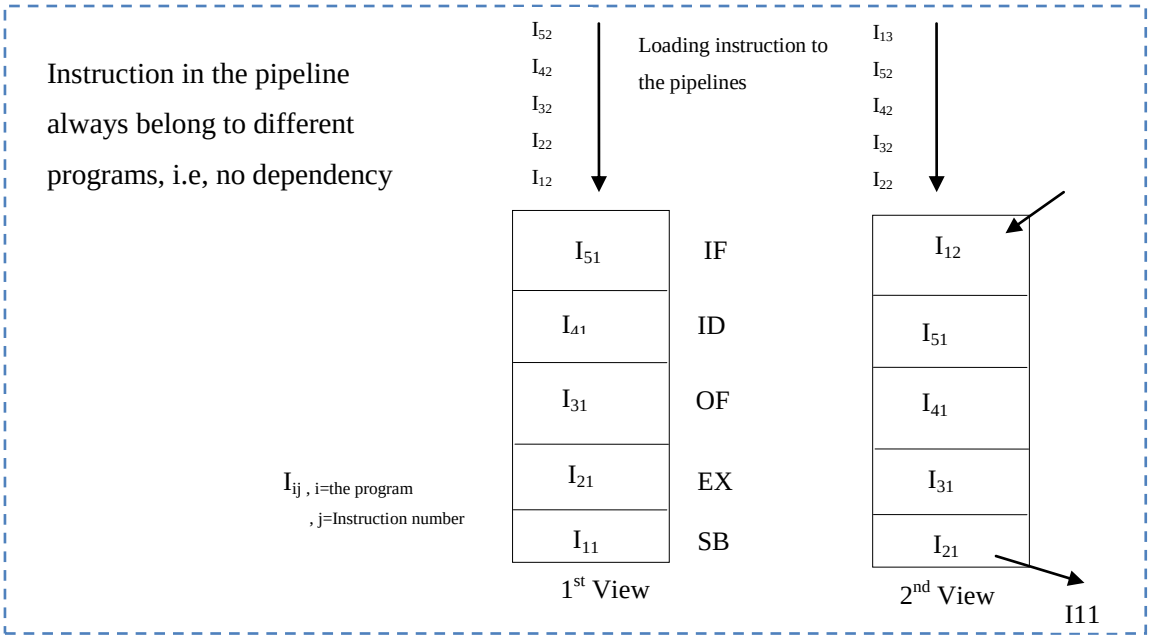

Figure 2.3: Multithreading Pipelining

The multithreading concept was extended so that threads become independent nodes (grains) of the same program. Again the size of the grain has to be optimal for better parallelism. Through his research, Jesshope and his team found, through statistical analysis, that if the program is divided into grains, each includes a number of instructions nearly equal to the number of the pipeline stages, a maximum parallelism will be obtained. Nodes (grains) of this size were given the name micro-threads [jess01, luo02].

Multithreading use a different set of solutions by utilizing coarse-grained parallelism. A multi threaded processor is able to concurrently execute instructions of different threads of control with a single pipeline [Theo02]. The need for multithreading mechanism in

multiprocessor system, especially in multi-pipelined cores is obvious. In order to use multiple cores simultaneously, multithreading mechanism are required.

Jesshope et. al. [jess96] demonstrated in his work that dynamic scheduling of micro-threads on several pipelined processors would result in massive parallelism, reducing the effect of out-of-order instructions in the pipeline.

2.2.3 Valgrind Simulator Tool

We will use simulator called Valgrind which is Open Source / Free Software under GNU/Linux. Valgrind is a runtime instrumentation framework; Valgrind catches all the memory accesses and gives the possibility to analyses these memory accesses. Other tool called Callgrind based to Valgrind which it performs detailed simulation of the L1, L2 and Last Level (LL) caches. Based to Valgrind developed by Josef Weidendorfer [valgrind, Fra08].

Josef Weidendorfer tool of valgrind which is developed at Department of Informatics at the Technical University Munich. This tool is able to handle problems with the cache synchronization, concerning an AMD multi-core system. This tool must handle all the problems occurring on a multi-core system. This tool implements the L1, L2 and LL of caches as the AMD multi-core architecture, crossbar switch interconnection networks where used. The main work in our thesis is to compare the performance of our architecture multi-core system that uses the embedded DPCAM in shared memory with AMD architecture, so we must modify the Valgrind to be simulated with our architecture.

Micro-benchmarking programs were designed to measure the performance of a multi-core system. This will be used to compare the results between AMD and our design. OpenMP is a good micro-benchmarking program to get some concurrent software [Ope08]. This type

of micro-benchmarks program is used to measure the overheads of synchronization, loop scheduling and it has a best feature to be compared in multi-core system.

Several parameters will be tested in this simulator.

1. The number of instruction, the program requires running.
2. The number of cache hits (hits ratio) in both L1 and L2.
3. The number of cache misses (miss ratio) in both L1 and L2.
4. The number of invalidation access the shared variable through the networks.
5. The execution time delay for the micro-benchmarking program.

2.3 Major Issues in Multi-core System

2.3.1 The Scheduling Policy for Multiprocessor

Regarding to the scheduling policy in multiprocessor systems can be generally defined as how we can execute a set of processes on a set of processors. These criteria called scheduling policy that aimed to minimize the expected runtime (execution time) the program and other parameters as minimizing the cost and the communication delay [Ste02, Rama77].

The Scheduling policy includes two activities; first, partitioning the program into nodes (grains) arranged in a dependence graph, where the node size varies from one instruction to a process or a task [Barry96], and second assigning the nodes to the processing elements of the system. The portioning policy decides the level of parallelism, where as the assignment policy decides whether the scheduling policy is static (deterministic) or dynamic [jess96].

In static scheduling, the nodes are assigned to the processing elements (PEs) (the pipelined cores) at the compile time [Cha10]. The set of nodes assigned to a processing element (without violating the dependency rules) forms a program stream. The program stream is

stored in the local memory of the processing element. PEs executes their streams in parallel. Also, dependency rules across the streams must be observed. Several Static and Dynamic Scheduling in multiprocessor systems were built in the last years, since 1980's this issue has been over killed by research [Fang90, C.D88]. Easily one can find a scheduling policy that suits his architecture. Therefore, this is outside the subject of our research.

2.3.2 Communication among the Cores

Regarding to the communication issue, it implies that the interconnection network must provide effective communication among the processors and between the processors and the shared memory in both SSM and DSM in figure 2.1and figure 2.2. This issue represents the core of this thesis.

Several topologies of multiprocessor interconnection networks were designed, implemented, and tested. Common shared bus, multiple bus, crossbar switch, multistage interconnection networks (Omega, Butterfly…)**,** and tree and mesh topology are examples [Imr07]. All of them suffer from a bottleneck when more than one processor accesses the shared bus or when they request the same memory module. These different interconnection networks will be discussed in later sections.

Also, two communication strategies among the PEs (the cores) of the system were identified; communication through shared variables, and message passing communication [Barry96]. The shared variable communication needs synchronization in order to make sure that the PEs are accessing the variable in the right time and the right order to satisfy best and correct performance. This issue causes the cache coherence problem which needs the synchronization of the variables among caches in each core.

2.3.3 Cache Coherence Issue

As we explain in the communication issue the bottleneck happen when more than one core need to access the same module at the same time. The cache coherence is the second problem which adds burden to the interconnection networks.

Cache coherence is a concern in all multi-core architecture because in all shared memory multi-core system, each core has its own distributed L1, L2 and L3 caches [Intel12] [Bar08]. Since each core has its own caches, the copy of the data in that cache may not always be the most up-to-date version. It is possible to have many copies, one copy may store in the main memory another copies in each cache memory. Therefore when one copy in any location is changed all other copy must be changed also, Figure 2.4 illustrates the cache coherence problem and shows how two different processors in dual core system can have two different values for the same location. This difficulty is generally referred to as the cache coherence problem. Without any additional precautions we can see different versions of the same variable in each core [Joh07].

Time	event	Cache content for core 1	Cache content for core 2	Memory content for Location X
0				1
1	Core 1 read X	1		1
2	Core2 read X	1	1	1
3	Core 1 store 0 to X	0	1	0

Figure 2.4: The Cache Coherence Problem in Dual Core System [Joh07]

2.3.3.1 Cache Coherence Protocols

A cache coherency protocol is a protocol that preserves the consistency between all caches in shared memory system [Arch86]. In general in multi-core architecture there are two protocols for cache coherence, a snooping protocol and a directory-based protocol.

The snooping protocol always works with a buses based system, and use number of different states to decide if the value must be updated or not. The simple way is to broadcast the update value to all caches.

The directory-based protocol can be used on any networks; it is scalable to many cores, in contrast to snooping protocol which is not scalable. In this protocol a directory is used that holds information about which memory locations are being shared in multiple caches. Also, it knows when it needs to be invalidated. As in snooping protocol, directory-based protocol use number of states to decide if the value is valid or invalid. Most modern architecture use Modified, Exclusive, shared and Invalid (MESI) states Modified Data means the cache data can be read and written locally without accessing the bus system. Exclusive as modified means that data are in one cache only, but it has not modified at all, and is exactly the same as in main memory. Shared means that the data are held in more than one cache and this cache data are up to date. Invalid means that the cache data defined as invalid if the cache line is empty, or data in it are invalid [Joh07]. In the last years different protocol have been proposed with different number of states as (MSI) and (ESI) [AMD07].

In all interconnection networks the cache coherence protocol puts a severe overhead and bottleneck on the network. Only one processor can broadcast the latest version of the data at a time. Also adding more cores in multi-core system has more impact on the amount of time necessary to validate the protocol [M.Zah10]. So, it poses a scalability problem.

In this work, we present a multi-core system architecture in which the communication medium solves and nearly eliminates the problems of bottleneck, the latency, the cache coherence and the scalability of the system. In the following sections, the traditional networks will be introduced in order to pave the way for clearing the features of the proposed architecture in chapter 3.

2.4 Interconnection Networks in Multi-core Architecture

In this section we will discuss the common Networks on a Chip (NoC) used in multi-core system, the discussion will include the networks topologies with some of its problems and the performance of these networks.

2.4.1 The Common Bus Networks

The common bus is the first and simplest interconnection topology. It is a shared path to which all units of the system are connected figure 2.5.

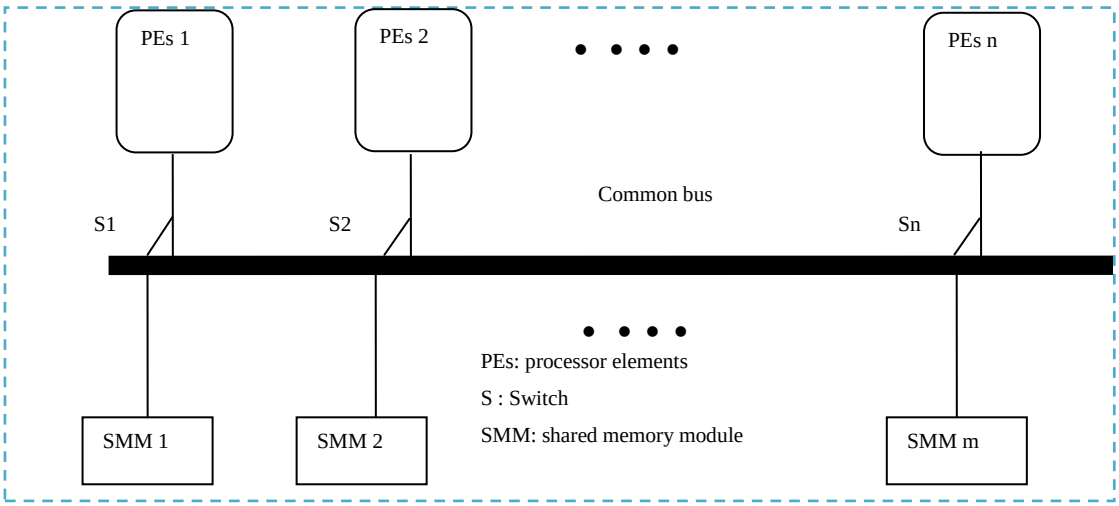

Figure 2.5: Common Bus Interconnection Networks [Edw86].

The common bus is the least complex and easiest to configure. The common bus represents a bottleneck by itself. It can serve only one communication request at a time, so to prevent

contention on the bus an arbitrator is used and any core need to use the networks must first request the arbitrator to determine the bus status [Edw86].

2.4.2 The Crossbar Switch Networks

Crossbar switch networks are obtained by increasing the number of buses for every core and memory module in horizontal and vertical links, such that buses can be accessed at cross points as Figure 2.6.

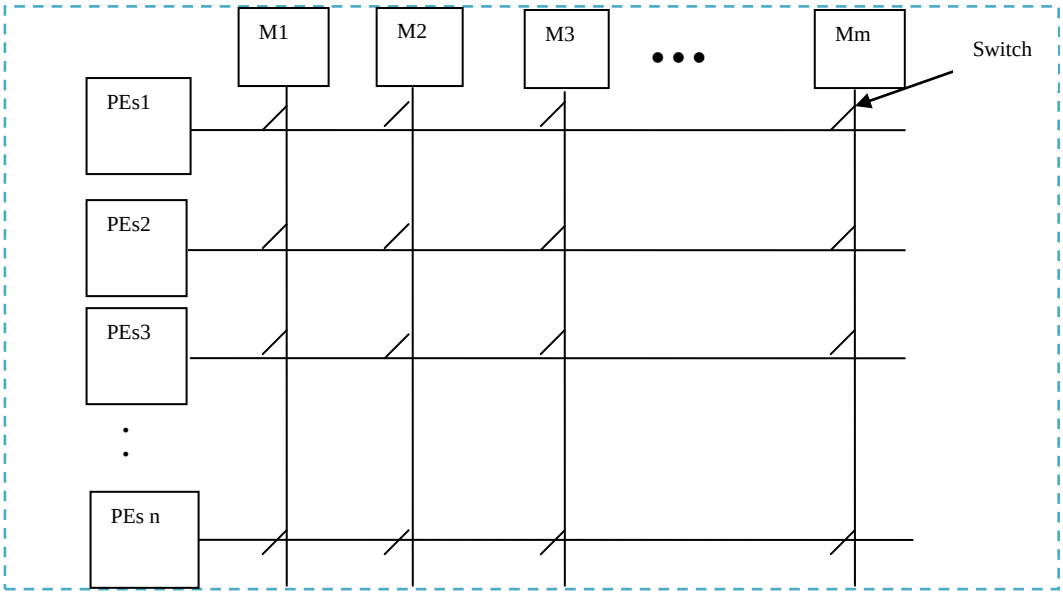

Figure 2.6: Crossbar Switch Interconnection Networks [Imr07].

The number of vertical and horizontal buses equal the number of PEs and memory module. The crossbar switch is the most effective interconnection network. It has complete connectivity with memory module. However, the conflict appears if more than one source request is competing for the same memory module. This case is called memory interference. Only one request will be served [Imr07]. To solve this conflict, every bus leading to a memory module must be provided with an arbitrator see Figure 2.7. But the arbitrator adds to the implementation of crossbar more complexity and cost when we increase the number of processors.

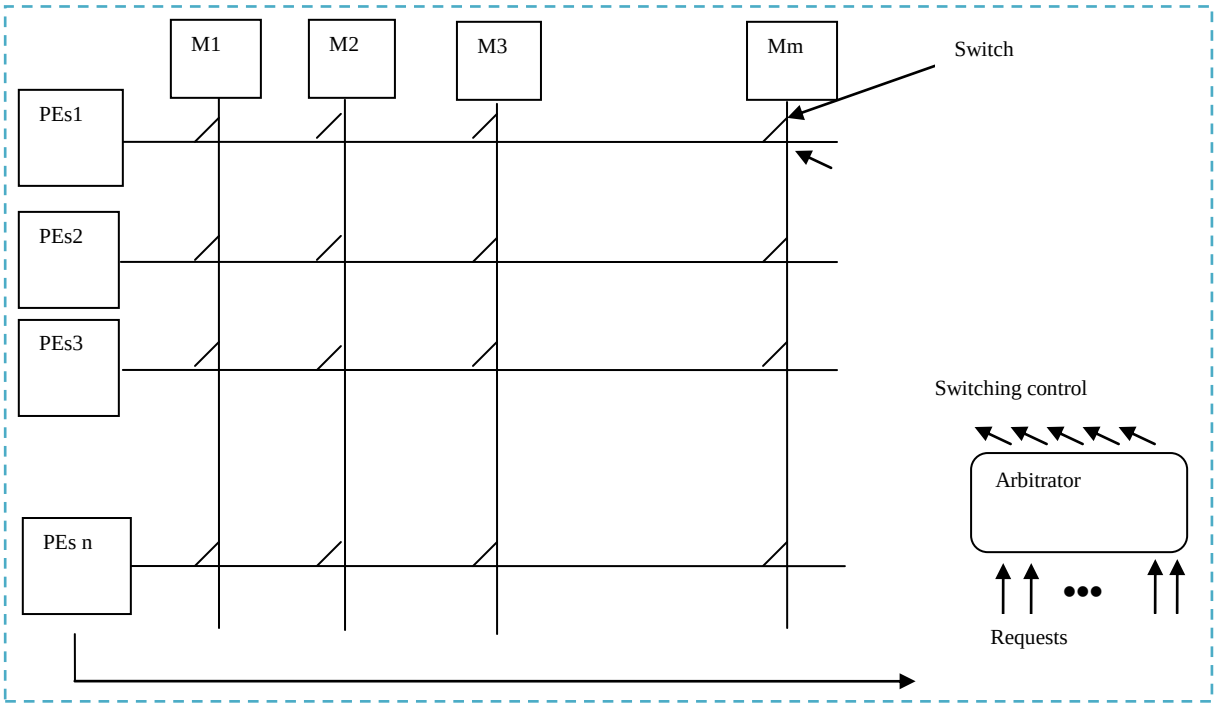

Figure 2.7: Crossbar Switch with Arbitrator Interconnection Networks [Imr07].

Crossbar networks can be improved by including small buffers at sources and destinations this called buffered crossbar. In last year many research were published and suggested in this topic, Critical internal Buffer First (CBF), input queued (IQ) and internally buffered crossbar (IBC) switches [Lot05].

More recent papers focused on the development of the arbitrator in different networks. Some paper suggested a new design by adding an asynchronous First IN First Out (FIFO) after each processor requests to hide the delay of waiting time during request. But it still requires a complex arbitrator [Moor02, Rig02]. Other mechanism create intelligent arbitrator that provides priority for processor to make request, it is shown that this arbitrator can be extended easily to support large numbers of processors [Has06].

To conclude we can say that the crossbar switch is the most effective interconnection network [Lan82] and the most modern multi-core system adopts this interconnection networks, but it still suffer from bottleneck. In addition to the problem of the cost and

complexity of crossbar grows rapidly as a function of square of inputs number, where if N is the number of processors then the number of switches equalN^2.

2.4.3 The Multiple Bus Network

Crossbar network is meant to include same number of cores and memory modules. This to meet the demand of the processor at a rate of a request per processor in each cycle. But after each request the processors do an internal operation after each request so the probability to make new request will drop from 1 to 0.5 [Ayy93]. Studies have suggested that connecting N processor to M memory using number of shared buses B equal to half the number of processors, pluses one, this design called multiple bus networks as we explain in Figure 2.8.

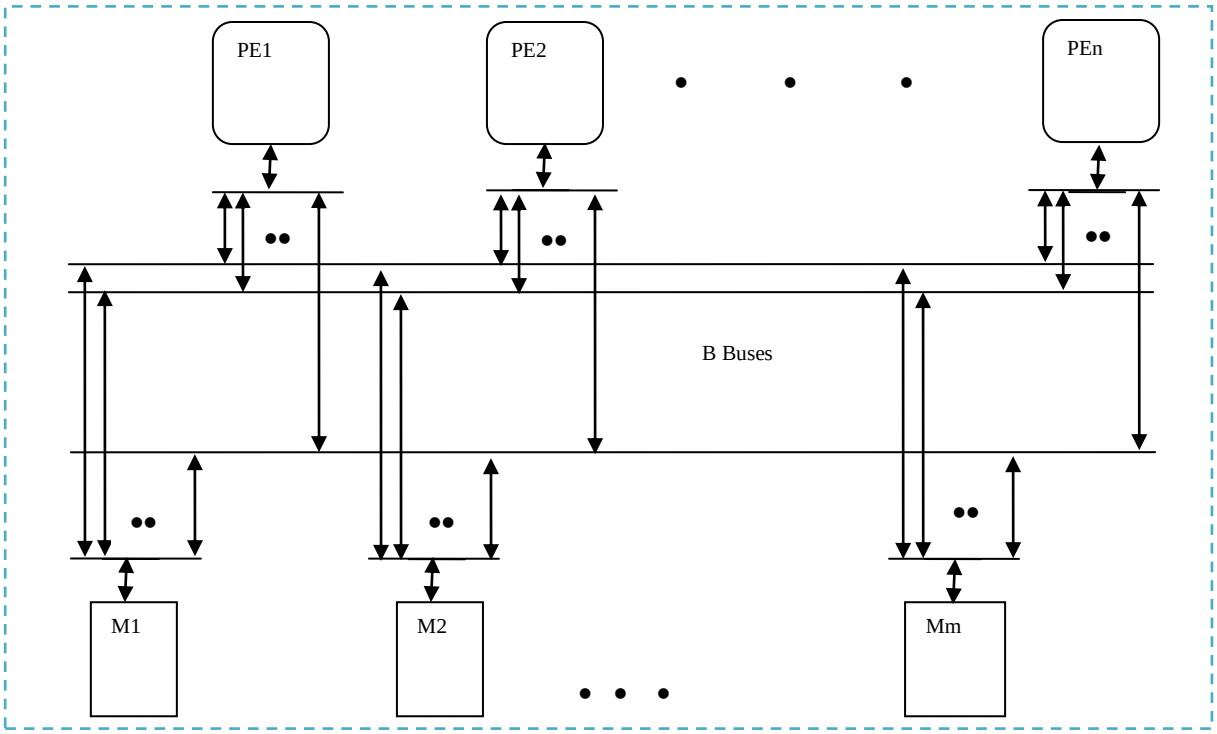

Figure 2.8: Multiple Bus Interconnection Networks [Ayy93].

The Multiple bus can serve only a number of requests less than or equal to the number of buses. If two or more requests are going to the same shared memory module, only one of them can be served [Mud84].

2.4.4 Multistage Interconnection Networks

Multistage Interconnection Networks (MINs) are mostly used in parallel multiprocessors systems to connect processors to processors and/or to memory modules [Imr07]. Crossbar switch provides a full connection between source and destination. But, as we mentioned the cost and complexity of crossbar grows rapidly as a function of square of inputs number (in a full crossbar the number of inputs, N, equal the number of outputs). If the 2 x 2 switching elements are used in building a crossbar system then the number of switching elements needed is $(N/2)^2$.

In MINs we can use different size of switching elements 2 x 2, 4 x 2 or any size, the most commonly used size of the switching elements is the 2 x 2 switching elements. Hwang and Briggs [Hwa93] have shown how to provide complete interconnection of one set of N devices to another set of N devices using multistage network of $\log_2 N$ stages with N/2 switching elements of size 2 x 2 in each stage.

There exists various types of MINs where proposed and implemented as Delta networks, Omega networks, Butterfly networks, Self Routing and shuffle-exchange MINs [Barry96][Chi06, Bhu89, Var89]. The difference between each of these networks is the topology of interconnection links between the crossbar stages. In all MINs if more than one request need the same intermediate link or the same destination shared module, bottleneck will be happened then only one of them will be served as we shown in Figure 2.9.

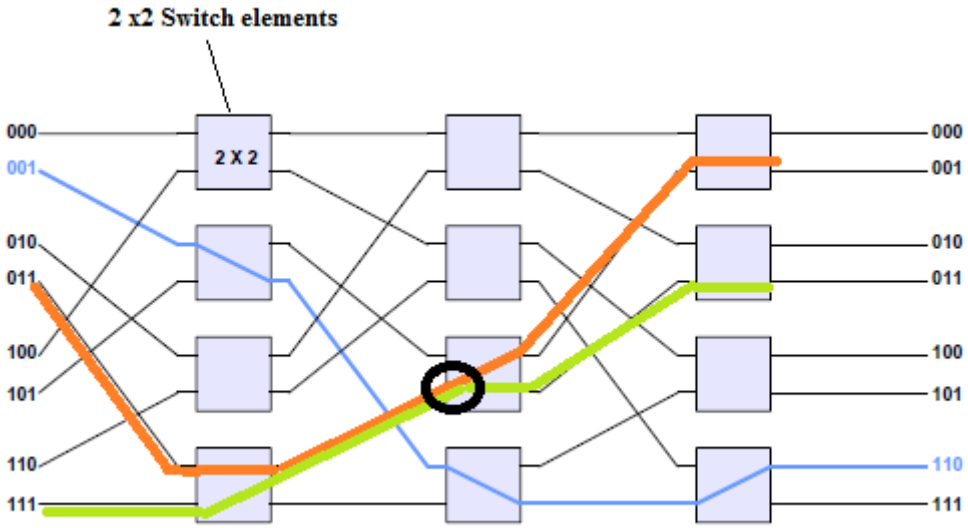

Figure 2.9: Multistage Interconnections (MIN) with 2 x2 Switch Elements [Imr07]

2.5 Network on Chip on Modern Multi-core System

The most modern multi-core system use different Network on Chip (NoC) to communicate between processors and shared memories. Sun T1 is a multi-core multiprocessor introduced by Sun in 2005 as a server processor; each core has private Level 1 caches, each core access shared level 2 caches via a crossbar switch networks as shown in Figure 2.10 [Joh07]. IN AMD multi-core system use System Request Interface & Crossbar Switch to organize communications among the cores [Bar08]. Intel multi-core especially in Intel i7 add the L3 smart and shared cache and increase its size about 16M to reduce the traffic among the cores. But it still needs buses interface. Intel developed their Quick Path Interconnect bus (QPI), which is a 20 bit wide bus running about 4.8 and 6.4 GHz ,as shown in figure 2.11[Neh11]. All of this topology still suffers of bottleneck problems, and the variable synchronization between caches.

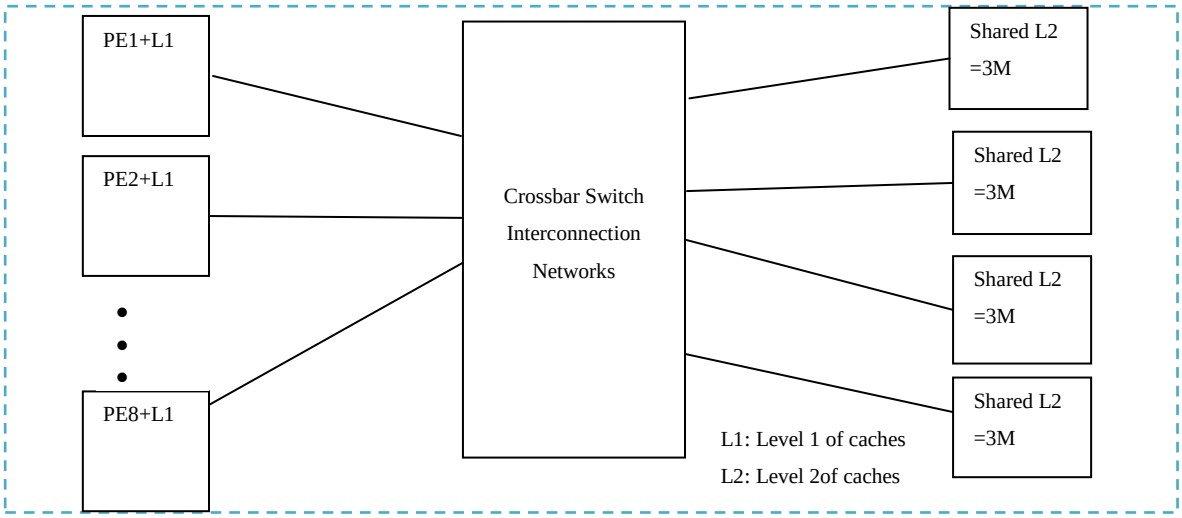

Figure 2.10: The T1 Sun Multi-core Architecture [Joh07].

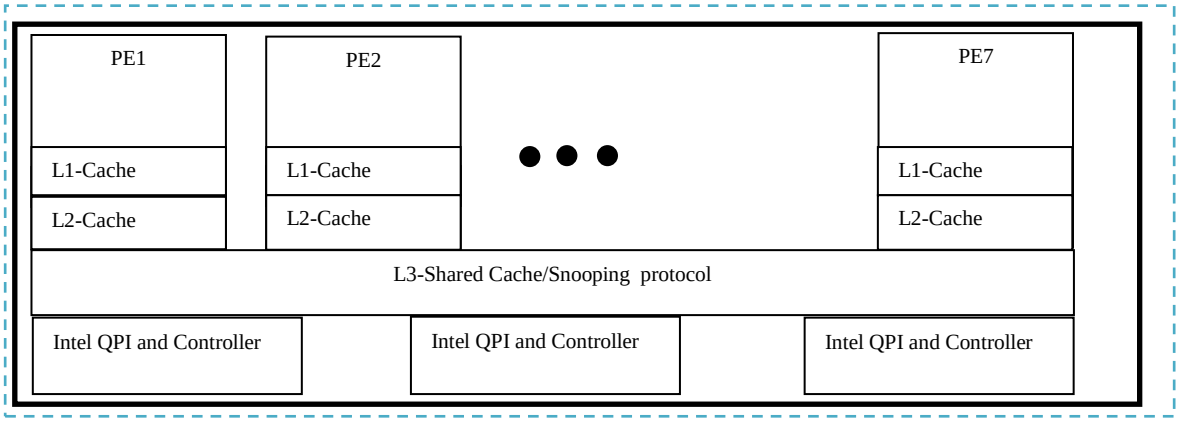

Figure 2.11: Intel I7 Multi-core Architecture [Neh11].

2.6 Types of Cache Memory

There are three types of cache organization; the direct mapping, the set associative and the fully associative cache.

2.6.1 Fully Associative

In this type of cache both memory address and data are store in the cache. The memory is of the Content Addressable Memory (CAM) type. Memory address is simultaneously

compared with all addresses store in caches using the internal comparator in each location. Figure 2.12 illustrates the architecture of fully associative cache memory.

The main advantage of fully associative cache is its high performance compared to its size. The main disadvantage of the CAM cache is the complexity in design and the cost. Also, the fully associative cache needs an algorithm to select where to store information in it. Different algorithms were used to solve this problem, as Random Selection, First In First Out (FIFO) and Least Recently Used algorithm (LRU) [Sha96] [Bary96]. The Content-addressable memory (CAM) is a special type of fully associative cache.

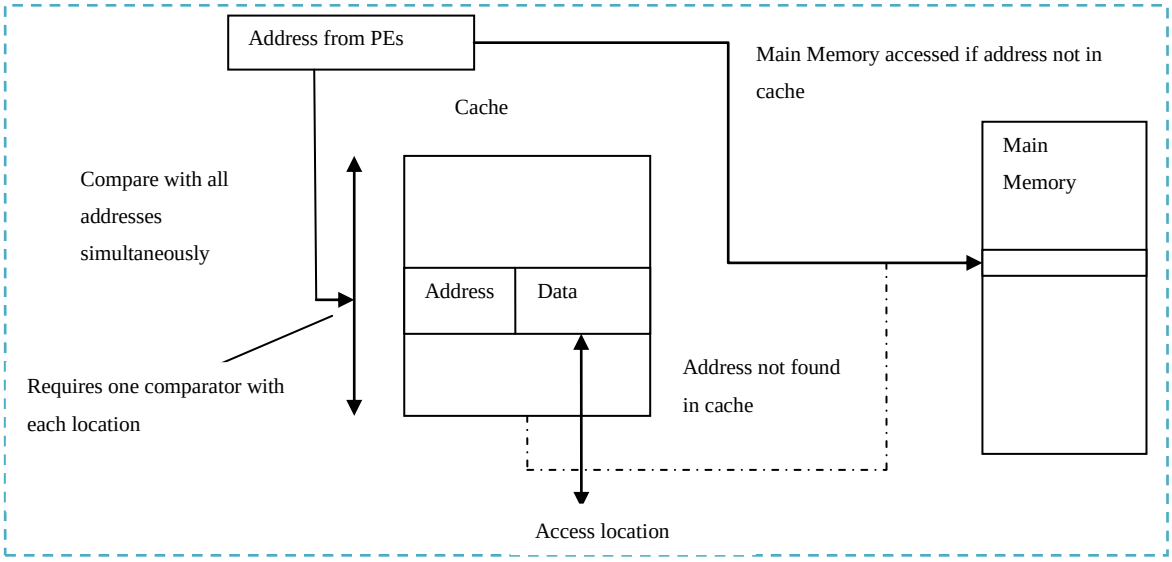

Figure 2.12: Fully Associative Cache Memory [Barry96]

2.6.2 Direct Mapped

In this case, the memory is divided into blocks called sets. So in one cache line we can hold one entry of a set of the main memory. The main advantages of this type are very easy and cheap to implement, but the performance will be dropped if accesses are made to different locations with the same index.

2.6.3 Set-Associative Mapping

This is a combination between a fully associative cache and a direct mapped cache. Cache divided into sets of lines. One address of a set of the main memory can be stored in n possible cache. It is not as complex as fully associative cache, but offers an improvement of performance because more than one address can be stored. Figure 2.13 represent set associative cache memory.

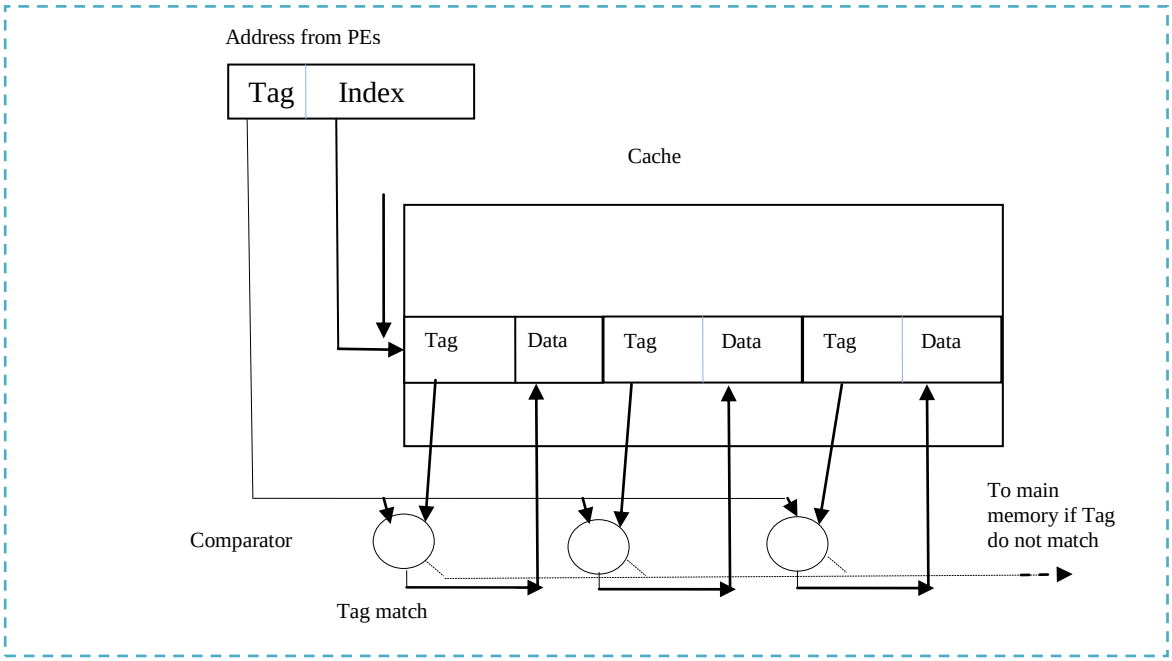

Figure 2.13: Set Associative Cache Memory [Barry96]

2.7 Cache Events Classification

When a processor accesses the data, the data may be cached or not. These two events are called cache-hit and cache-miss. A cache-hit and cache-miss may occur on any level of caches. Both cache hit and miss are an important event when looking to cache coherency. In this section we will focus on how miss and hit cache exactly works?

2.7.1 Cache-Hit and Cache-Miss

Usually, the processor presents its request for data to both cache and the main memory. If the data found in the cache, the processor cache it within its cycle time and the main memory is informed. This is called a cache hit case. If the data not found in the cache, the processor has to wait for the slow response of the main memory. Upon receiving the data, a copy of it is kept in the cache for future accesses. This is called a cache miss case.

Cache-Miss can be classified into four types [Rob08] [Joh07].

1- Cold Miss: the processor requests the data for the first time and data is not stored in the cache system.
2- Capacity Miss: because the cache size is limited, some data was stored in the cache, but it has been replaced with new data.
3- Conflict Miss: we can't store some data in a set associative cache because the number of blocks is limited. Thus a miss occurs, but this type can be avoided in fully associative cache.
4- Coherency Miss: this kind of misses happened when processor writes some data. The cache line in other cache has to be invalidated so when other processor accesses these data a coherency miss occurs (these data called shared variable). This type of miss is the most important miss associated with multi-core systems, more techniques proposed to reduce this type of miss[Hyu09][Jae04][Mil12][Che10]. So in our multi-core architecture design simulator Coherency Miss is the biggest concern to reduce miss ratio as we will explain in chapter 4.

2.8 Embedded Memory Unit Crossbars Interconnection Networks

In 1995, Ayyad and Radolf designed, implemented and successfully tested an embedded crossbar buffer using 4 Kbyte (2KbytesX2) FIFO memories [Ayy96]. The design was implemented on board to connect four of the 32 Intel 486 motherboards of The Makbilan Multiprocessor at Hebrew University. Makbilan was a DSM system. At that time the FPGA technology was not advanced enough to accommodate this architecture. Message passing and packets were used to send, multicast, or broadcast messages from source/s to destination/s simultaneously.16-bit data buses were used. Packetizing, de-packetizing, and storing the received data in the right local destination memory location/s represented a considerable communication overhead.

New designs of Dual Port content addressable memory (DPCAM) was suggested; a port for reading and another for writing. Under the supervision of Abu-Mwais, two students (S. Surkhy and A. shawar) managed to implements the idea using Veriloge language. Later, the author modified this design, then successfully implemented and simulated it. In the next chapter, the architecture of this memory will be presented.

After that we took this as a basis for multiport CAM (MPCAM). CAM is not feasible to be implemented as a standalone memory (we need very large number of pins). However it proved too useful as a part of multi-core architecture if implemented as shared memory. Also, it proved useful for fast cache coherence. So we decided to design and simulate it as a part of multi-core architecture as we will show in chapter 4.

Chapter 3

The Design of New Crossbar Embedded DPCAM and the Simulation Results

3.1 Cache Memory in Multi-core System

There has been always a speed gap between processor and memory [Car02, Pat97]. A great deal of the researchers' effort went to narrow this gap. Cache memory is extremely fast memory that is built into processor to decrease the speed gap between the processor and the main memory. In fact, cache memory responds to the processor request in the real time, and the processor does not need to insert wait states. The processor uses cache to store instructions and data that are repeatedly needed to run programs, when processor access data, it normally fetches it from main memory, this access need long time as compared with the cache. If data already stored in the cache, the processor does not need to wait. The cache responds within the processor cycle. This improves the overall system performance. There are different types of cache as we will discuss later in this chapter.

The most modern multi-core systems use more than on level of cache [Intel12][Neh11]. As figure 3.1 illustrates, the AMD Barcelona multicore system [AMD07], Level 1 (L1) of the cache is the fastest one and is directly next to the PEs (the cores) with 64KB per core. L1-Cache in AMD multi-core architecture exists in two versions. One for data, and the other for instructions. They are Called L1D for data and L1I for instructions. The second cache is Cache level 2 (L2-cache) with 512 KB per core followed by the third level of cache (L3). Cache L3 represents the shared memory with total size about 3MB. This multilevel of caches required to improve the communications between cores in multi-core system, thus synchronization between all these caches is required.

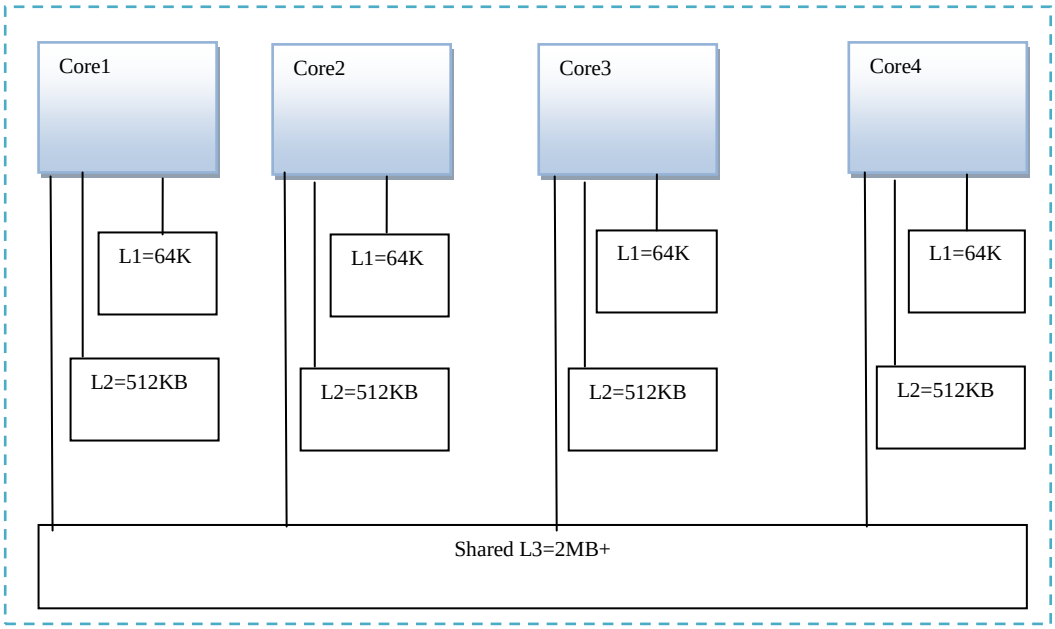

Figure 3.1: AMD Multi-core Cache Level [AMD07]

3.2 The Design of Dual-Port CAM and Multi-Port CAM

The widespread application of multi-core system makes cache memory highly accessible by the cores of the system. The multi-port cache memory can provide the needed

accessibility to multiple cores. In this section, we propose a Dual Port CAM (DPCAM) to be used as an integrated part of a new crossbar interconnection network organization. This organization results in what we call Multi-Port CAM. This MPCAM guarantees that all cores of the system can access data for read and write operations simultaneously. Queuing, contention and the need for arbitration are totally eliminated. Also, as large number of data versions can be accommodated in this MPCAM and accessed by presenting their unique tags, the cache coherence problem is totally eliminated because each core broadcast the last version of shared variable to other cores, so the variables always update.

3.2.1 Single Port CAM

Content Addressable Memory CAM is a memory which its locations are accessed through comparing Tags rather than providing their addresses. In 1980s, the designers of dataflow machines badly needed these devices. However, the available semiconductor technology did not allow implementing large CAMs at that time. Small CAMs (up to 64 memory location CAMs) were available). So, designers resorted to employ RAMs as lookup tables in order to emulate the CAMs. With the advent in semiconductor technology, large CAMs are available now (an order of 100 Kbytes CAMs). Also, universal dual port cache memories have been designed [Kos06, Ara12, Hassa05].

In CAM, each stored data is associated with a unique tag. When we like to retrieve the data, we apply its tag with a read signal to all locations simultaneously. The applied tag is compared with all stored tags simultaneously. If any of the stored tags matches the applied tag, the equality signal of the location comparator enables the output of the location, and the data is placed on the data bus in order to be read by the processor. Figure 3.2 depicts a single port CAM.

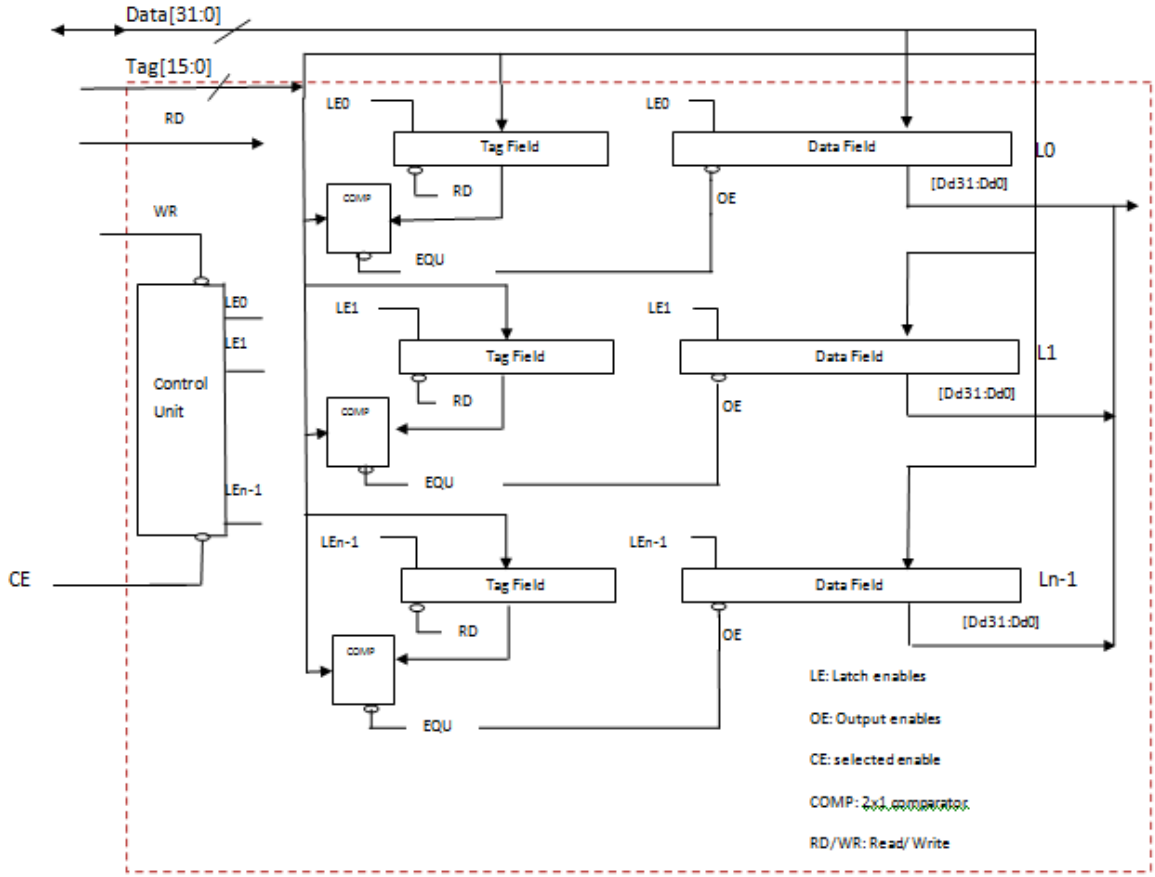

Figure 3.2: Single Port CAM Design

3.2.2 Dual Port CAM

In our crossbar embedded CAM interconnection network among multi-core the need arises for implementing a Dual Port CAM (DPCAM) cache memory modules at the cross-points of the crossbar; to overcome the problems in memory as addressing and unused spaces. The shared data will be residing in these modules and simultaneously accessed by all cores.

These DPCAMs have two ports; one for writing (broadcasting), and the other for reading. They allow simultaneous access operations from the two ports as far as they are not accessing the same memory location. In our organization we only need one port to write, and the other to read, including writing and reading to the same location. This is because in our crossbar embedded cache organization, simultaneous writing and reading on the same

location is likely to happen even with very small probability. The fact that we are designing this DPCAM to meet the needs of our cache organization has affected our design decisions. This included the data width, the tag size, the storage type, and the control circuit of the cache module.

With reference to figure 3.3, in this design, each memory line is composed of two fields; the data field and the tag field. The length of each field depends on the architecture in which the CAM is used. The data field can be 32 or 64 bits in order to go with 32 or 64 bit processors. The tag field can be varied to suit the number of shared data versions, e.g., 32 bit tag can accommodate up to 4 Giga versions of data.

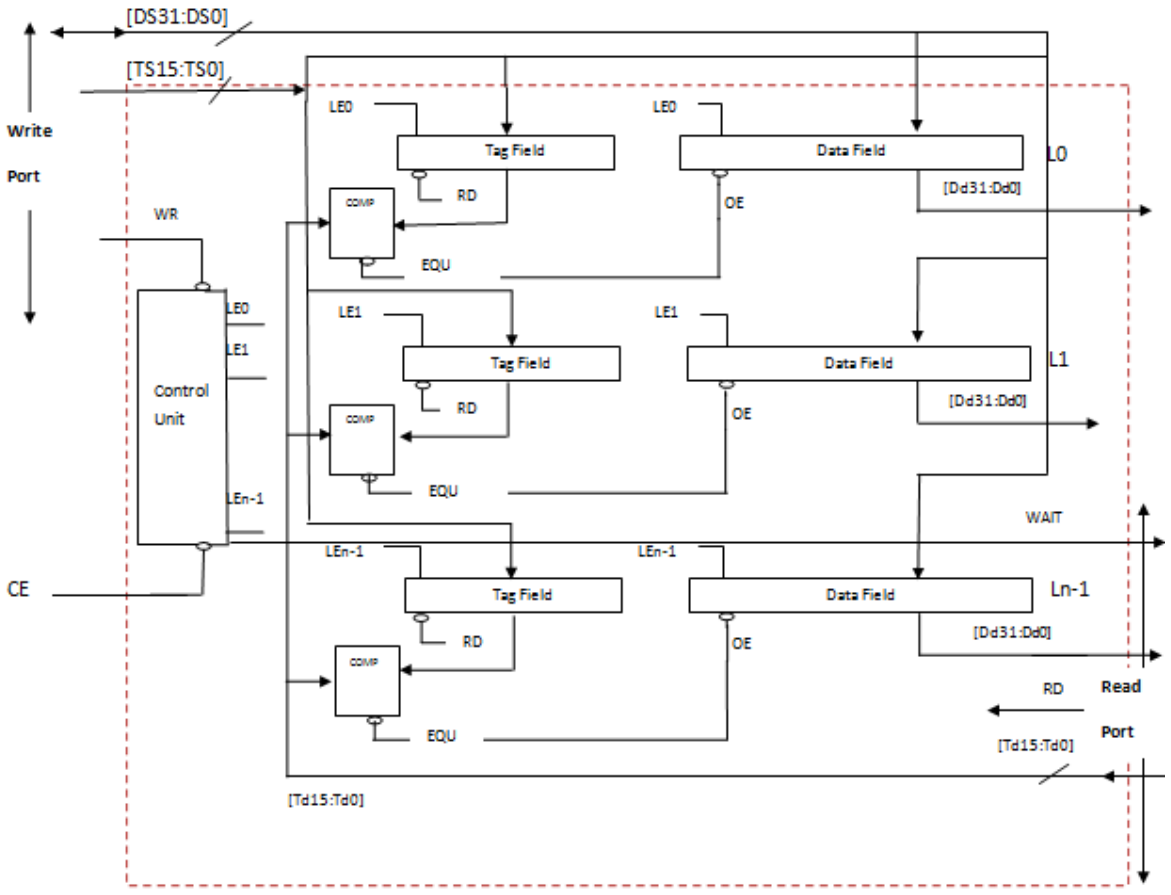

Figure 3.3: The Dual Port CAM Design

The writing process is controlled by the control circuit and the WR signal. The control circuit of the DPCAM includes a pointer to produce an active high Latch Enable (LE)

signal for each memory line on a rotating basis. When the system is reset, this pointer points to the LE0 first memory location, so that the first writing operation will be performed on line 0 of the memory. After writing to the current location, the pointer points to the next location, and so on until Ln-1. In the case of simultaneous write and read process, the circuit gives the priority for writing and gives the reading processor a WAIT signal. The WAIT signal can be obtained by ORing the active low RD and WR signal coming from the writing and the reading processors.

The Store Back (SB) of writing core provides the data [Ds31-Ds0], the tag [Ts15-Ts0] (the tag can be the address plus the version number), and the active low WR signal. With the rising edge of WR signal (the end of WR), the control circuit moves the LE to LE1 in preparation for the next writing which will be to line 1. This process can be repeated until LEn-1 is reached, after which it moves back to LE0 where it starts the overwriting process over the old data and tags. The memory lines are made of latches which means that the writing process is level triggered.

The READ process occurs when the Operand Fetch stage (OF) of the reading core applies the destination tag [Td15-Td0], and an active low read (RD) signal to all tag fields simultaneously. The RD signal outputs the stored source tags to the comparator of each memory line in order to be compared with the applied tag simultaneously. If a match occurs, the equality signal (EQU) of the comparator is used as an output enable (OE) signal which outputs the stored data from the data field to the destination data bus where it can be read by the OF unit of the reading core.

3.2.3 Multi Port CAM

In the proposed architecture in figure 3.4 we have redesigned the shared cache to become a two dimensional array of DPCAM elements. The number of rows and columns of the array are equal and equals the number of the pipelined cores. An extra row is added for the primary shared data which will be loaded by the global memory management unit. This design eliminates the need for arbitration because there is no contention. Also, this cache eliminates the need for global register.

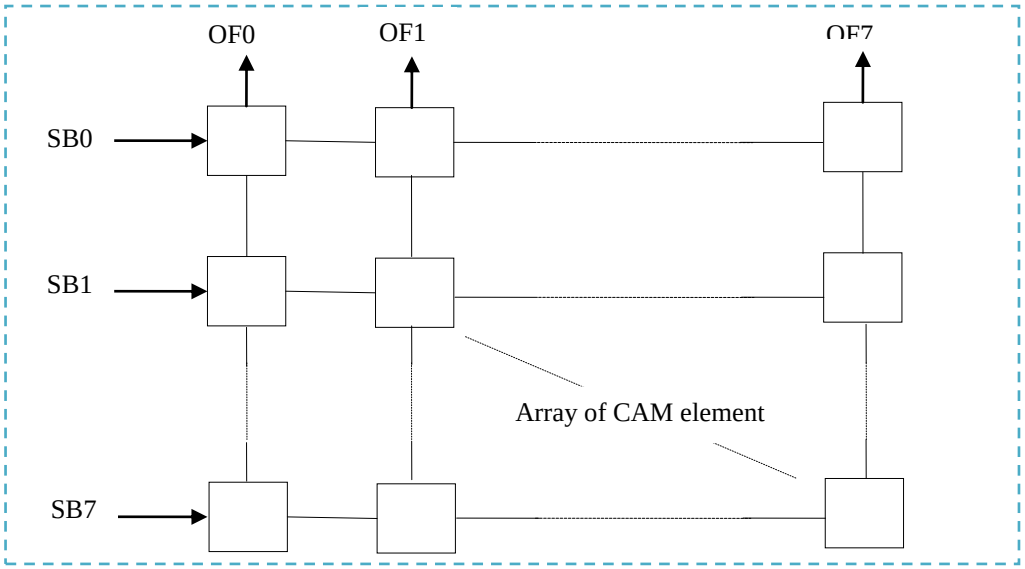

Figure 3.4: The Multi-port Content Addressable Memory

3.3 The Crossbar Embedded DPCAM Architecture (The MPCAM)

In chapter two we discussed the problems in different type of interconnection networks in DSM and SSM multiprocessor. All problems come from the fact that, in spite of moving the architecture into the chip, the designers are still stuck with the past usage of the off-shelf components and using in the same way they were used on the printed circuit board (PCB). There is no reason to suggest that the cache must be one piece and no busses can be implemented within the cache architecture. In Fact Crossbar like buses can be embedded

among a matrix of small caches. This is the same as saying “Small cache modules can be embedded at the cross points of a crossbar architecture”. So the DPCAM which we designed and simulated can be implemented at the cross points of the crossbar.

In our design in figure 3.4, a SSM Multi-core architecture is assumed, where the shared cache modules are distributed on the cross points of the crossbar. The Store Back (SB) of the pipelined core is connected to the horizontal buses, and the Operand Fetch (OF) units of these cores are connected to the vertical buses of the network. The shared cache modules are specially designed (DPCAM) modules. While loading a program stream, the memory management unit, which loads the instructions and the data to cache L1, loads the primary shared data to the last row of DPCAMs in the MPCAM. This organization is considered cache L2 in our proposed architecture.

3.3.1 The Claims for Embedded DPCAM Networks

With the above MPCAM organization we can claim the following:

1. SBs of all cores can broadcast data, each to the DPCAM modules in its row simultaneously. There is no queuing and no arbitration. Note that this is not the case if the modules were placed on the edge of the crossbar as in the usual way, where the broadcast forces queuing with severe latency.
2. OFs of all cores can search for the data, each in the CAM modules of its column simultaneously. Also, there is no queuing and hence, no arbitration is needed.
3. In the writing (broadcasting) process, the DPCAM module is designed to place the data in the first empty (or least recently written to) location. So, there is no unused memory location.

4. The data in the DPCAM location won't be overwritten before n clock cycles elapses, where n is the number of memory lines in the module. The compiler and the scheduler must guarantee the usage of data within the allowed time. Also, if the core needs the read version of data for beyond the allowed time, it can keep it in its local cache.
5. As an alternative to point 4, we can implement two DPCAM modules at each cross point; one for short reaching and the other (less frequently used) for far reaching communication. The cores will be less frequently writing to the far reaching parts so that longer time elapsed before they need to overwrite a datum.

3.3.2 The Mathematical Model of Crossbar Embedded DPCAM Network

In this section we will compare the performance of our network design (Crossbar Embedded DPCAM) with normal crossbar switch networks. The bandwidth (BW) is the most important parameter in any multicore system networks (number of requests served by the network). The mathematical models of these networks are derived from probabilities theories. The derivation of the model is based on the model assumptions. For the traditional crossbar and the MPCAM performance models, the following assumptions are made:

1- The number of processor cores **= (n).**
2- The probability that a core is making a request to the crossbar network during a cycle is **(r).**
3- In the crossbar switch architecture, **n** processor cores are connected to the row buses of the crossbar. The cross point switch connects the row bus to the required

column bus to which the requested memory module is connected. Not more than a single core can access the destination module at a time.

4- In Crossbar embedded cache (MPCAM), n (SB) units of n cores are connected to n row buses, while n (OF) units of the same n cores are connected to the column buses of the crossbar.

5- So, 2n units are expected to present requests to the MPCAM, **and 2nr requests** are expected during the cycle.

6- As claimed before, all SBs can broadcast and all OFs can read simultaneously, unless two are addressing the same memory location, which has nearly zero probability.

7- The **bandwidth** is the number of requests served by the network **(BW).**

8- The bandwidth of the crossbar switch is given by equation 3.1 [K.W99, Barry96].

$$BW1 = m - m\left(1 - \frac{r}{m}\right)^{n} \quad \text{.....3.1}$$

Where m is a memory module and n is the processor elements in ideal cases n= m.

9- The bandwidth of crossbar embedded DPCAM is given by equation 3.2.

$$BW2 = 2(nr) \quad \text{.....3.2}$$

i.e., provided that the dependency rules are satisfied, all requests will be served.

3.3.3 Comparing Results Between Crossbar Embedded DPCAM and Normal Crossbar Switch.

Using the mathematical model in section 3.3.2 we can draw the bandwidth function with different values of r and n, and then we will compare the results from the crossbar embedded DPCAM and the normal crossbar switch. Figure 3.5 to 3.7 show the bandwidth

versus number of cores (n), results shown that bandwidth function increases when the number of core increases linearly.

In figure 3.5 we notice that when r=0.8, in normal crossbar switch the bandwidth always less than number of cores. But in crossbar embedded DPCAM network can serve more requests. In figure 3.5, when (n=64), the bandwidth in crossbar switch about 35 whereas in crossbar embedded DPCAM the bandwidth about 102, when (n=8) the bandwidth in crossbar switch about 5, in crossbar embedded DPCAM the bandwidth about 13. So the crossbar embedded DPCAM can serve more request than normal crossbar.

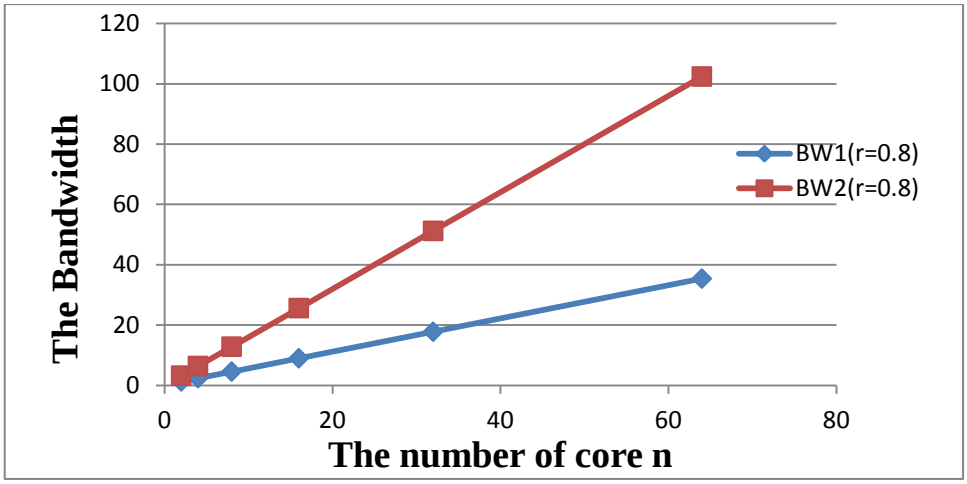

Figure 3.5: Bandwidth Function with(r=0.8)

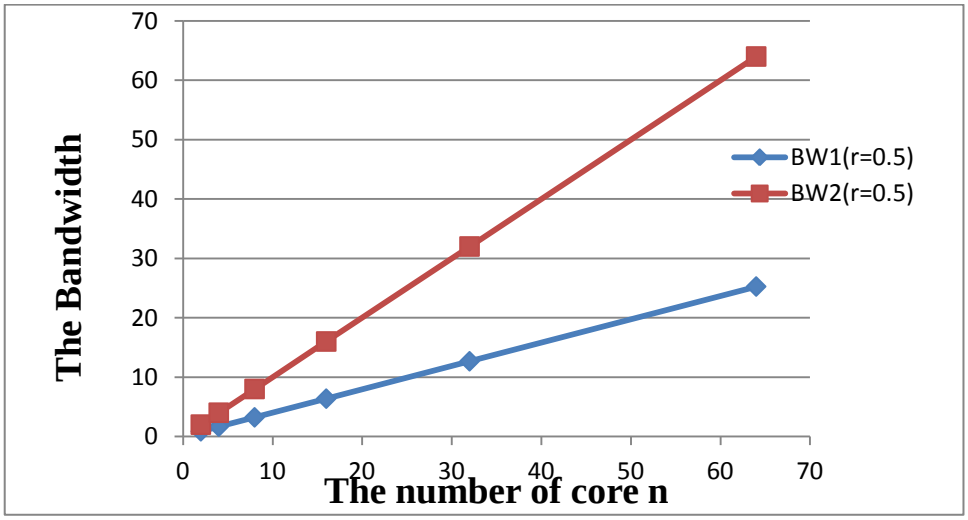

Figure 3.6: Bandwidth Function with(r=0.5)

Figure 3.6 displays the bandwidth function when the r=0.5. This means that the probability to make request from any core is down to half, this cause the reduction of the bandwidth when compared to request rate 0.8. The Crossbar embedded DPCAM still better than normal crossbar switch because there is no competing between cores across the network. When (n=64) the bandwidth in crossbar switch about 25, in crossbar embedded DPCAM the bandwidth about 64. This is due to the fact that in the case of crossbar embedded DPCAM the number of requests is double those of the traditional crossbar. Moreover, the crossbar embedded DPCAM is not blocking, i.e., all the requests are accepted, whereas some are rejected in the case of the traditional crossbar.

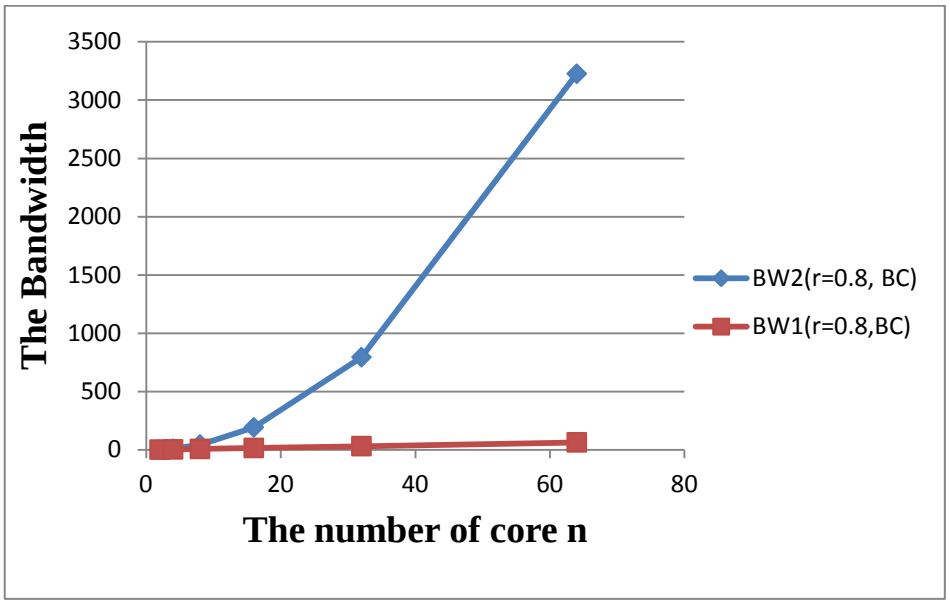

Figure 3.7: Bandwidth Function at Broadcast Situation

Figure 3.7 Depicts the broadcast situation, in normal crossbar switch only one processor can broadcast the shared variable at a time so the maximum bandwidth equals $(r(n-1))$, while in crossbar embedded DPCAM there is no competition among the cores of the system. So all cores can broadcast their variables and the maximum bandwidth equal $(r(n)(n-1))$.

In conclusion we can say that the crossbar embedded DPCAM network has better bandwidth than normal crossbar switch. This is because there is no competition between cores in both write and read operation. But in the traditional crossbar switch interconnection networks only read or write (one core) can be served at a time, whereas in our shared cache two cores, one for read and another for write can be served simultaneously.

3.4 Simulation Results of Crossbar Embedded DPCAM Circuit

Using Verilog Hardware Description Language (VHDL) in Quartus II software from ALTERA, DPCAM and the crossbar embedded DPCAM network have been designed and implemented. In our work we use a Stratix III FPGA which has good specifications for this type of design as a target. It provides high-performance, lowest possible power consumption and high-integration capabilities. The main advantage of a Stratix III FPGA actually increase with design size because it has large number of I/O pins and gates compared to other available chips family [Alt08, Alt13]. More of other characteristics can be found in ALTERA-site [Altera]. The simulation results show that latency of less than or equal four nanoseconds per shared variable access in multi-core system has been achieved where the latency to access the shared memory is, approximately, 65 nanoseconds in modern multi-core system [Neh11], and about 40 nanoseconds in Intel multi-core system that use QPI. Figures 3.8 to 3.11 explain the simulation results in both functional and timing simulations.

Figure 3.8 shows the functional simulation of crossbar Embedded DPCAM , as we explained in section 3.2.2 for write process core add Data and tags(Tag source) with active low WR signal and in case of read core generate tagd (Tag destination) with active low RD

signal. In functional simulation the outputs appear on the pins without considering the delay of the gates in design. In this figure we have executed the network in different situations. In the first situation all cores write their shared variable simultaneously. In the second situation all cores read the same variable produced by the second core simultaneously. In the third situation both the first and second core write shared variable also the third core read variable produced by first core. Finally both first and second core read simultaneously and the third core writes variable. In all situations we made sure that the embedded DPCAM network operates in multi-core system with full non-blocking and without bottleneck also no arbitration is needed.

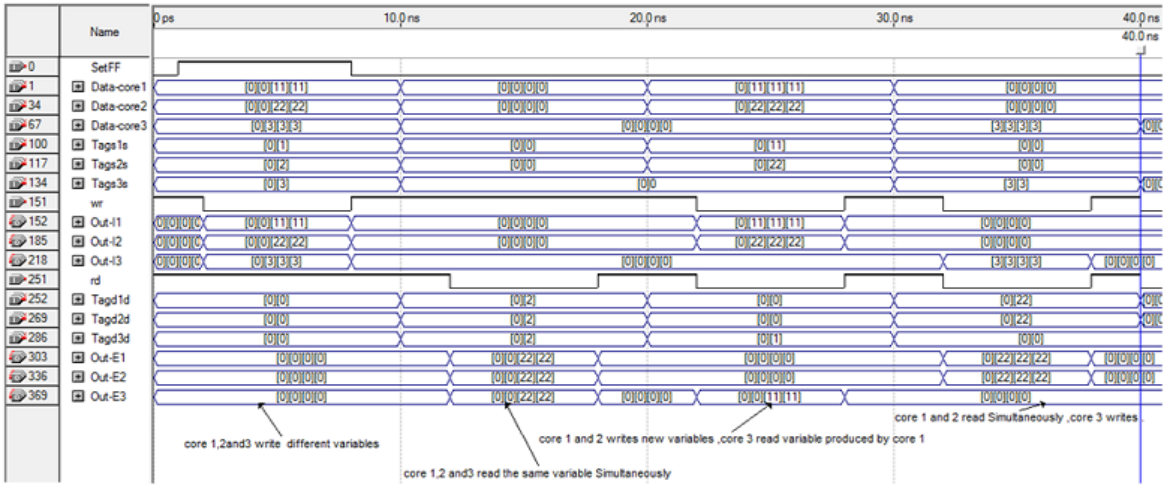

Figure 3.8: Crossbar Embedded DPCAM Functional Simulation

Figures 3.9 to 3.11 show the Timing simulation of crossbar Embedded DPCAM. In timing simulation we can measure the delay of accessing a shared variable in nanosecond. Figure 3.9 shows that when the first core broadcasts variable, the delay is about three nanosecond and when all cores read shared variable simultaneously four and half nanosecond this due to the delay of comparator units. Figure 3.10 shows that when all cores write different variable simultaneously, the delay is about three nanosecond and when all cores read shared variable simultaneously four and half nanosecond. Figure 3.11 displays that when

the first core write shared variable and the third core wants to read the same variable the delay is about three nanosecond for first core and about nine nanosecond for third core. This because when the writing operation has just completed the reading operation starts to compare the tags.

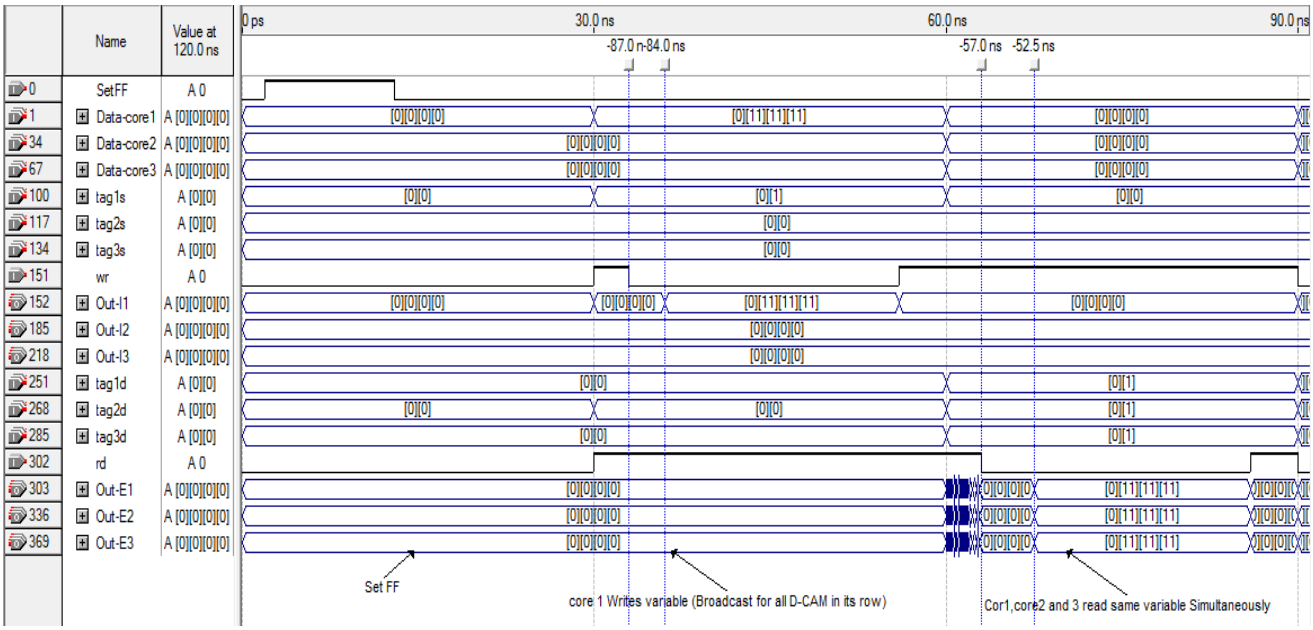

Figure 3.9: Crossbar Embedded DPCAM Timing 1 Simulation

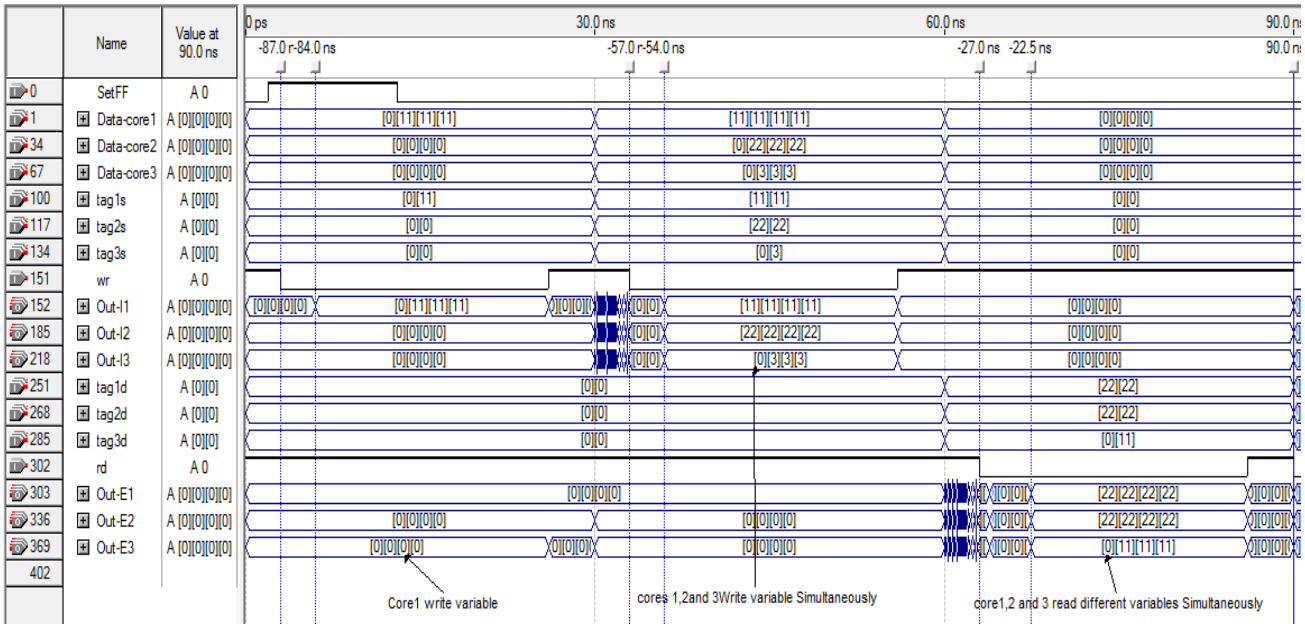

Figure 3.10: Crossbar Embedded DPCAM Timing 2 Simulation

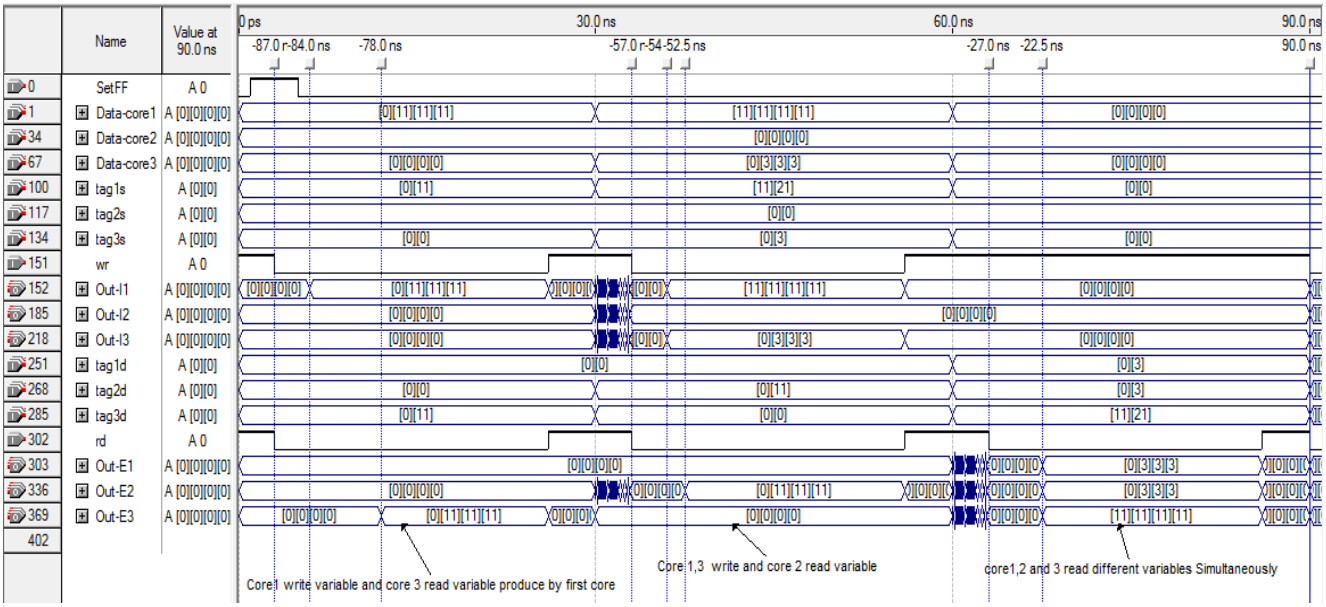

Figure 3.11: Crossbar Embedded DPCAM Timing 3 Simulation

So, this simulation process proved that all cores can access the shared variables in the shared CAM simultaneously without the problems of contention, queuing, and arbitration.

3.5 Area Estimations and Model Complexity

Because the technology continue to shrink and the size of chip continue to be larger. So estimating the area of our proposed architecture would help use in judging this design. In this section we have adopted on published papers, especially for cache estimated area to decide and compare the die area of our model with AMD model. Figure 3.12 displays the total area for AMD multi-core architecture this architecture use two level of caches, L1 as private cache and L2 as shared cache. Figure 3.13 displays the total area for our multi-core architecture this architecture uses two level of caches also, L1 as private cache and L2 (embedded DPCAM) as shared cache.

Components	Size of cache	λ^2 Area	Area mm^2	num	Total area mm^2	Ref#
L1 Cache	64 KB (32I+32D)	8.06e+8	12.593	8	100.75	[Kie98][Lin99]
L2 Cache	512 KB shard	7.25e+9	113.281	8	906.24	[Bat98][Gua09]
Total Area (mm^2)					1006.9	

Figure 3.12: AMD Model Area Estimation

Components	Size	λ^2 Area	Area mm^2	num	Total area mm^2	Ref#
L1 Cache	64 KB (32I+32D)	8.06e+8	12.593	8	100.75	[Kie98][Lin99]
Embedded DPCAM	16 KB shard	5.10e+8	7.987	64	511.16	[Sha02]
Total Area model (mm^2)					611.91	

Figure 3.13: Embedded DPCAM Model Area Estimation

As stated earlier, the design of the multi-port CAM is meant to allow simultaneous access of a number of cores to the memory. This includes simultaneous writing and reading of the shared variables to and from the memory. As a crossbar topology is used in this design, the cost and complexity is expected to grow as a square function of the number of ports. Note that the number of input ports is equal to the number of output ports. The growth of complexity is limited by two factors; the available silicone area on the chip and the number of pins.

The design of the multiport CAM as a standalone chip is really limited by the number of pins rather than the silicone area. It needs a number of pins equal to 2nw, where n is the number of port and w is the number of pins per port. For example, designing a four 8-byte ports CAM needs 2X4X64= 512 pins. This is we could not design and simulate more than 3-port standalone CAM as targeted to Altera StratixIII FPGA chip which has less than 512 pins[Alt08, Alt13]. However, if the design is integrated as part of a multi-core processor, it is fairly feasible to design eight or sixteen port CAM of 2Mbytes as a shared cache of the processor. This is less than or equal to the shared cache used in Intel and AMD multi-core processors so far.

Regarding the scalability of the system as we will explain in chapter 5, it has been shown in this thesis that if n-core processor with n-port CAM shared cache is feasible, then as far as the available silicone allows, the system can be linearly expanded to n blocks where each block is an n-core processor. This means a system of n square cores. Its cost will be nXc, where c is the cost of the n-core processor. The expected latency is fixed at an average of 1.5t, where t is latency of accessing the local cache or the shred cache (the same) in an n-core processor.

We conclude that the design presented and simulated in this thesis represents a quite simple, efficient, fast, and powerful system.

Chapter 4

Simulation Results within Multi-Core System

4.1 AMD Vs. Our Model Architecture

In the AMD multi-core simulator it is necessary to understand the main parts are required for the simulator. The simulator is based on the crossbar switch interconnection network, ESI (Exclusion-Shared-Invalid) cache coherence protocol, and accessing through cache and main memory. In the memory access works as first check whether there is a L1 cache hit, then if there is a L2 cache hit. If no cache hits, a miss is returned (in this design there is no L3 of caches) Figure 4.1 display the AMD multi-core architecture. But as we know in case of the multi-core simulator, shared caches must be checked whether the variables have been changed by another core or not. Further actions depend on the fact whether the requested data are already cached or not. This will be done by (ESI) cache coherence protocol.

In our multi-core architecture the embedded DPCAM behaves as an internal register file. At the same time it is considered as shared L2 of caches.

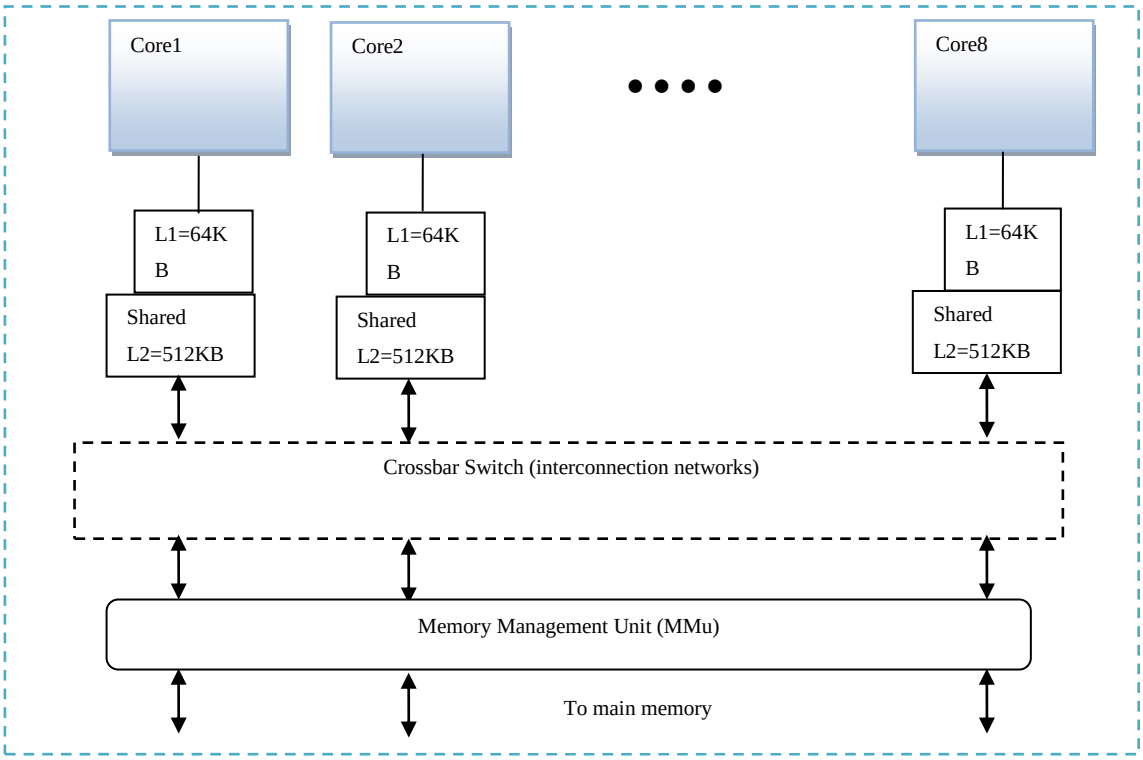

Figure 4.1: AMD Multi-core Architecture [AMD07]

The ESI cache coherence protocol can be abandoned in our design because any core modifies variable can broadcast it to all DPCAM caches in its row. So, any shared variable is up to date for all caches. So, the state is either valid or invalid. In other word all the valid variables in caches is a shared at all time. Figure 4.2 display the Embedded DPCAM multi-core architecture.

AMD Vallgrind simulator has two caches. One first level of cache and one second level shared cache for each core; each of them has instruction and data. So there is various numbers of caches dependent on the number of cores. This simulator implements an algorithm to generate these caches by using a one dimensional array of caches for each cache level. The maximum number of caches is assigned by the parameter multi-core in simulator, in this simulator there is a maximum eight cores.

In Embedded DPCAM simulator there are also two levels of caches. The first level of cache for each core and the shared second level of cache had been put at crossbars. in second level of cache the number of caches equal the square of number of cores n^2 where n is the number of cores, this can be done by using a two dimensional array of MPCAM (which is explained in chapter 3), the size of array is assigned by the parameter multi-core number in simulator, then any OF for all cores want to search data, read the column of MPCAM simultaneously and any SB of all cores want to write data broadcast data to row of MPCAM simultaneously. The interconnection networks between cores represents the main differences between AMD and our model, where in AMD used crossbar switch interconnection networks while in our model we used crossbar embedded DPCAM as new design of interconnection networks.

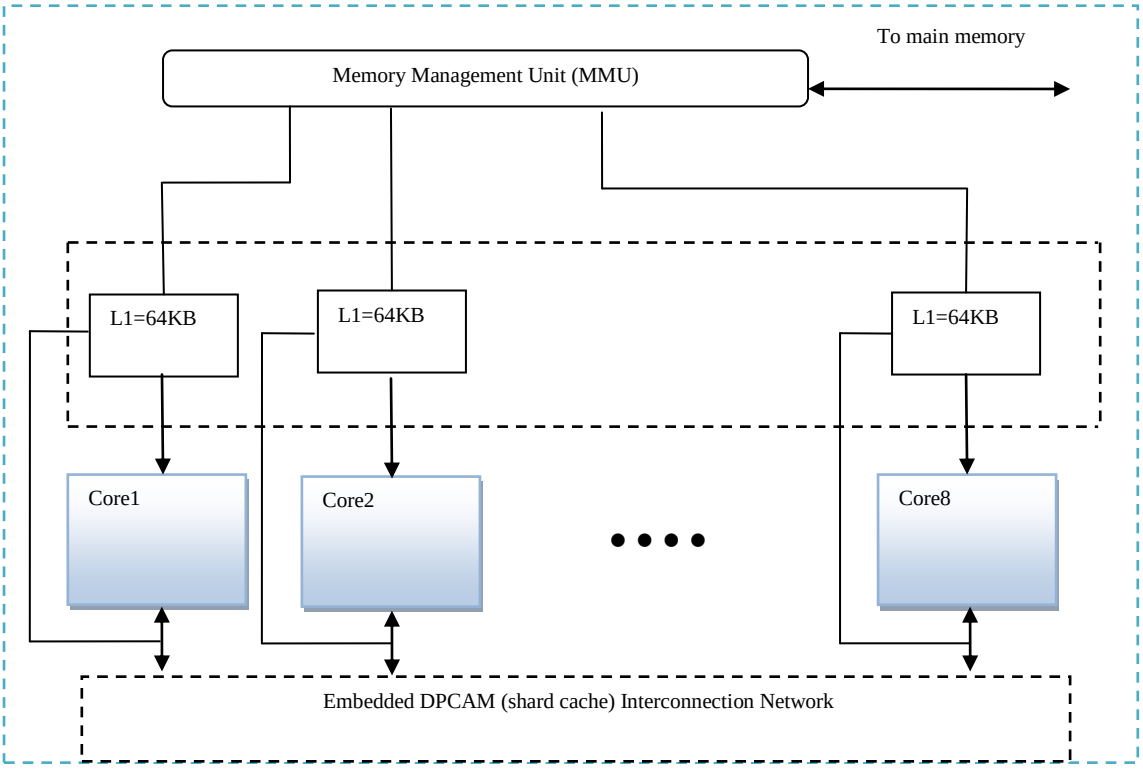

Figure 4.2: Crossbar Embedded DPCAM Multi-core Architecture

4.2 Results

In this section we will handle the results that have been obtained by the valgrind simulator. This results help use to compare between the AMD multi-core and our proposed multi-core system. Many benchmarking program used to test multi-core system, by using this program we can study the behaviour of our proposed system with AMD multi-core. This includes studying the five parameters listed in section 2.2.3.

4.2.1 Benchmarking program

Three types of benchmarking program were used. The first type is used in order to be sure if the multi-core cache simulator works without any problems, this program contain with single threaded so the results in both AMD and our proposed system will be similar. We use two program for this type (date) and (df). (date) that returns the current date, as well as time and time zone. We tested this program in both design. Another program is (df) that returns the amount of space used on all mounted volumes. The second type of program is a multithreaded program used in order to check the performance of multi-core system this program called performance program (PP) based on OpenMP. The (pp) program shown in figure 4.3. When we look to source code, line 9 contains the following command: omp_set_num_threads (n); to control the number of threads that shall be simulated this parameter can be changed. In line 11 to creates threads in parallel and runs the for loop variable as shared variable. Line 16 identifies a synchronization point for parallel threads. The third type of program is a multithreaded dependency program used to compare the execution time between our model and AMD model.

```
1 #include <omp. h>
2#include < s t d i o . h>
3#include < s t d l i b . h>
4  int main ( void )
5{
6    int  *shared ;
7   int abc=0;
8   shared =&abc ;
9  omp_set_num_threads ( n ) ;
10  int numof_proc=8888000/n;
11 #pragma omp parallel for shared ( abc )
12 for ( int i =0; i < numof_proc; i ++)
13 {
14   abc++;
15   abc--;
 16#pragma omp barrier
 17}
18return EXIT_SUCCESS;
19}
```

Figure 4.3: (pp) Benchmarking Program

4.2.2 Single thread testing (date) and (df) program

To check if the proposed simulator works without problem using (date) and (df) program we run these program in both simulators as the following:

#valgrind --tool=callgrind --multicore=n date , n change from 1 to 8 for AMD.

The results of the AMD multi-core simulator can be seen in Appendix A, where numerous tests were run:

#valgrind --tool=callgrind --newmulticore =n date , n change from 1 to 8 for proposed multi-core.

The results of the proposed multi-core simulator can be seen in Appendix A, where numerous tests were run:

In Appendix A. We can see that results in both AMD and proposed multi-core system are exactly same. The number of L1 misses and L2 misses is the same.

The second program is df. We run this program in both simulators as the following:

#valgrind --tool=callgrind --multicore=n df , n change from 1 to 8 for AMD

The results of the AMD multi-core simulator can be seen in Appendix A Where numerous tests were run.

#valgrind --tool=callgrind – newmulticore =n df , n change from 1 to 8 for proposed multi-core

The results of the AMD multi-core simulator can be seen in Appendix A where numerous tests were run. When comparing these results we show that in both multi-core simulator results are the same.

However these results in (date) and (df) programs confirm that proposed multi-core system works without problems.

4.2.3 Multithreading Testing Performance Program (pp)

This section describes and analysis the results by the AMD and proposed multi core simulator using (PP) benchmarking program.

We run this program in both simulators as the following:

gcc -fopenmp ./pp.c -o pp.o ,this command to compile the (pp) program and save it pp.o .

valgrind --tool=callgrind --multicore=n pp.o , n change 1 to 8 this for AMD

valgrind --tool=callgrind --newmulticore=n pp.o , n change 1 to 8 this for proposed design .

The results of both AMD and proposed multi-core simulator at n=2 and n=8 can be seen in Appendix B. All these results will be explained later.

4.2.4 Execution Time Using (PP)

If you want to test the performance of two multi-core architecture systems and compare the results, the execution time is the most important parameter, so we use the same multithreaded program (pp) in order to get comparable results. The time command runs the specified program command with the given arguments. When command finishes, time

writes a message giving timing statistics about this program. One of this statistics is CPU real time between invocation and termination the program which is the execution time.

This program runs in both simulators as the following:

time valgrind --tool=callgrind --multicore=n ./pp.o ,n chang 1:8 AMD

time valgrind --tool=callgrind --newmulticore=n ./pp.o ,n chang 1:8 proposed multi-core

When looking at all the results as we explain in Appendix B, results can be summarized as in figure 4.4. It is possible to say, that the new design multi-core system works well compared to AMD multi-core. When looking at the results concerning the execution time obviously it needs less time to complete the same program.

Number of core	**AMD multi-core time in second**	**New proposed multi-core time in second**
one	16.2	16.2
two	10.1	9.8
three	8.43	6.8
four	7.16	6.1
eight	4.98	3.13

Figure 4.4: Execution Time in Second (pp) Program.

4.3 Result Analysis

4.3.1 Number of Instructions per Core

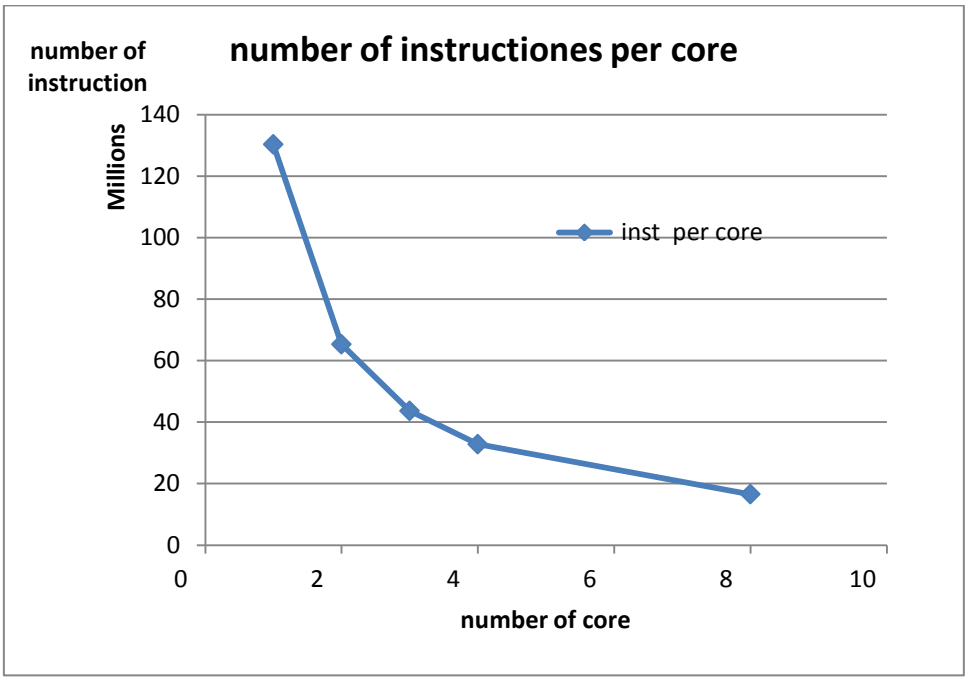

Figure 4.5: Number of Instruction per Core

Figure 4.5 displays the number of instructions per cores. When using more cores, this means that more instructions are required in order to do synchronization between all cores. As we show in figure 4.5 the number of instructions in one core is about 129.2 million of instructions (MI), after we use two cores the number of instructions equal about 6.529 MI per core, or about 130.39 MI for both cores and so on, when we increase the number of cores, the instructions that required for program to run is also increased. In our design it is expected to decrease this synchronization and the instruction required to execute the program become what about 64.6 MI are scheduled for each core. So there is no need for synchronization between cores.

4.3.2 Cache L2 (Shared Variable) Misses

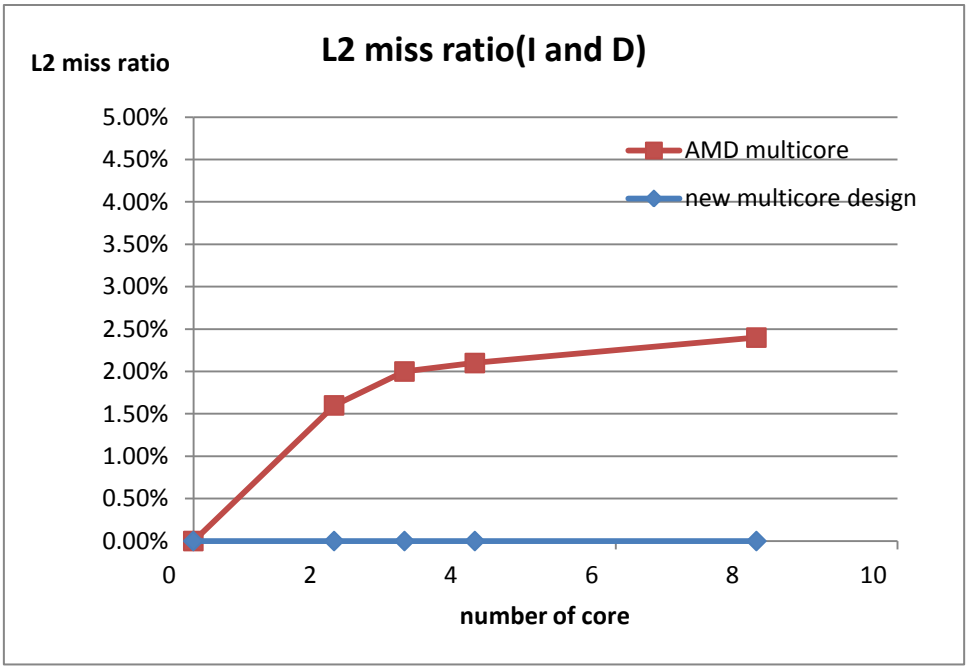

Figure 4.6: Shard L2 Miss Ratio

Figure 4.6 presents the L2 miss ratio on both AMD and new proposed multi-core simulator (Appendix B display the count number of this miss). In dual cores when we looking at the L2 miss, about one million misses occur when reading and writing variable which equal 1.6% miss rate. While in our proposed design the number of misses about 1.8 thousand which is near 0.00288% miss rate. In eight cores the number of misses in AMD simulator is about 1.8 million which equal 2.4% miss rate. But in our proposed design the number of misses about 3.1 thousand which is near 0.00413% miss rate. This result because in AMD L2 of caches when any core need to access shared variable it must access this variable through interconnection networks where the bottlenecks and cache coherency miss appear, While in our proposed the crossbar embedded DPCAM there is no bottlenecks and no Coherency Miss in shared variable caches so the miss rate in L2 of caches will drop because most variable where cached in embedded DPCAM. In our design the cache miss

occurs only if the data is not produced yet (scheduling problem). If produced all versions will be available and the operand Fetch unit can read the specified version. So, if the scheduling observes the dependency correctly, there will be no cache misses or cache invalidation problems.

4.3.3 Number of Invalidation Access the Shared Variable

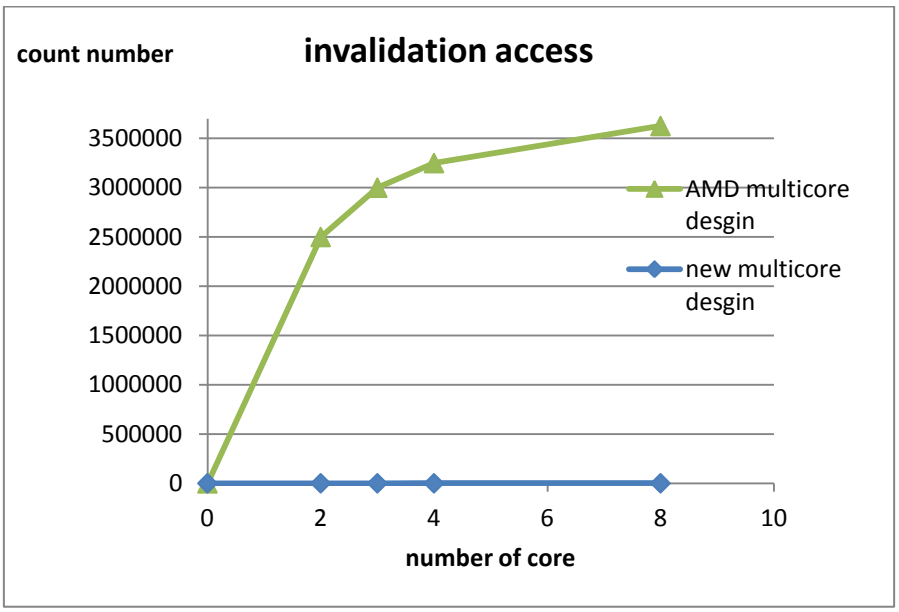

Figure 4.7: Invalidation Number per Core

Figure 4.7 depicts the number of invalidation when any core wants to access the shared variable through interconnection networks. In AMD multi-core when looking at the dual-core simulation, it can be seen, that about one million invalidations occur, in three cores about 3 million invalidation, in four cores about 3.2 millions invalidation and in eight cores about 3-6 million invalidation. While in our proposed multi-core system dual-core simulation, it can be seen, that about 44 invalidations occur, in three core about 80 invalidation, in four cores about 105 invalidation and in eight cores about 216 million invalidation, these invalidations invalidate when any core needs to access variable produced from other cores.

We can analyze this result as the bandwidth of the interconnection networks in crossbar switch which is used in AMD that can't serve all requests when more than one core want to broadcast shared variable to all caches then bottlenecks happens. Also the cache coherence consistency produces additional problem so more variables become invalid. But in crossbar embedded DPCAM there is no bottlenecks and no need for cache coherence protocol, so the crossbar embedded DPCAM can serve all requests, the only invalid variable happens when there is no core produced this variable.

4.3.4 Execution Time

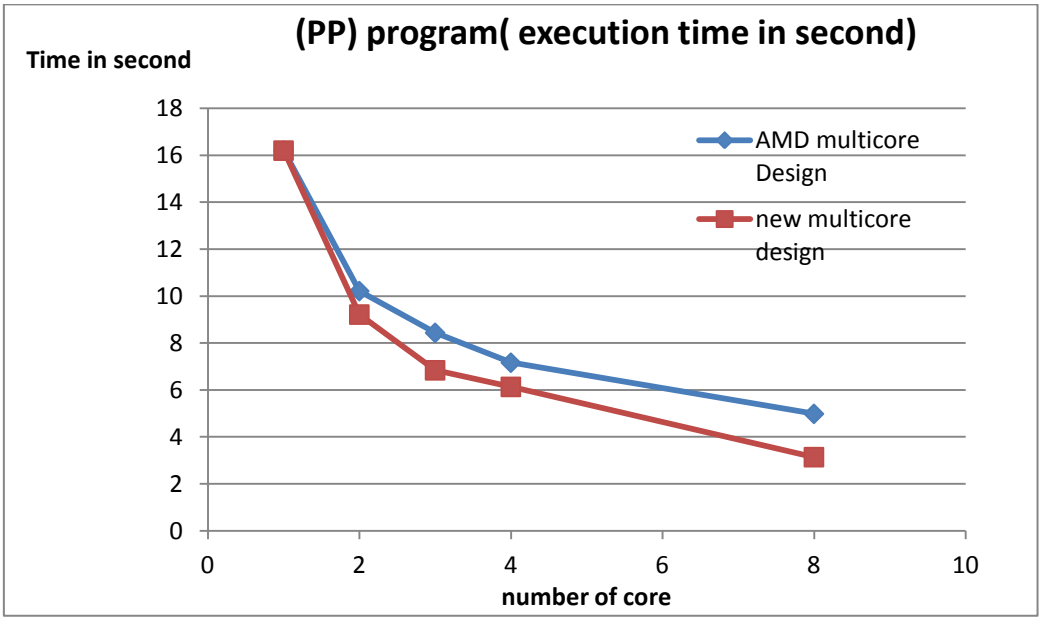

Figure 4.8: (pp) Execution Time in Multi-core

As we know the main goal of multiprocessor system is to increase the processing power processor so logically when we increase the number of processors from one to two, the execution time of any program is reduced to half. But the major problems which appear in multiprocessor system, we focused on the most important communication, synchronization between PEs. These problems prevent the execution time from decreasing regularly. Figure 4.8 shows the execution time in second versus number of cores. When we compare

between execution time in AMD and new design multi-core system as summarized in figure 4.4, we conclude that the new embedded DPCAM network and its improvement in communication between PES and synchronization between shared variable in caches reflects positively on the speed of execution time.

We used other dependency multithreaded benchmark program to compare the execution time between our model and AMD model when the threads of cores dependent each other, so we used benchmark program that multiply two matrices and calculate the determinant for the result matrix. Figure 4.9 displays the results of execution time when we run this program in two simulators.

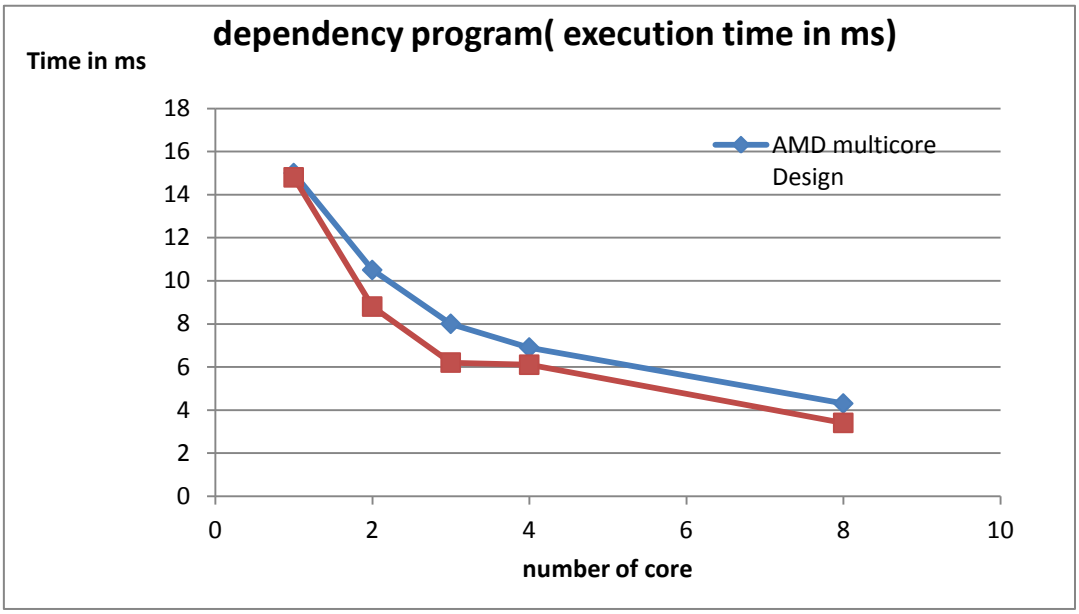

Figure 4.9: Dependency Program Execution Time in Multi-core

In conclusion we can say that also in dependency programs the new embedded DPCAM architecture improve the speed of execution time.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Expanding a multiprocessor in computer system is not as straightforward as it appears. There are many issues usually have to be resolved; the communication among the PEs of the system and its overhead, the scheduling policy and the Synchronization of cache coherence of the system processors. These three issues were thoroughly investigated in the last years. This thesis proposed a new architecture to solve these problems. This thesis consists of two parts: First, redesigned new shared cache and using it in designing interconnection networks for multi-core systems. The interconnection networks which connect between cores in multi-core systems are still suffering from a bottleneck problem. The new interconnection networks have been called crossbar embedded DPCAM, the bottleneck has been totally eliminated. So all the cores of the system can write to and read from the shard memory simultaneously. We have presented the simulation results of the crossbar embedded DPCAM using VHDL. Results have shown that there are no bottlenecks, no arbitration, and hence no additional latency, A latency is about five nanoseconds per shared variable access.

In the second part we simulated the crossbar embedded DPCAM network as a part of the multi-core architecture system, then we compared the performance of our design with

AMD multi-core system using different type of micro benchmarking program. The results of this architecture have shown less invalidation, miss ratio when access shared cache memory, and better execution time achieved, than in AMD multi-core system.

5.2 Future Work

The scalability of multiprocessor (multi-core) systems is a very important issue. In fact multiprocessor systems represent a great hope for computing if we can build a multiprocessor with large number of processors without paying heavy penalty of queuing and delay. The system we proposed in this thesis is a non blocking one. It is contention free, with no queuing, no arbitration and no delay. The big question is: can we expand this system further and keep its feature at the same time?

For scaling this system up with minimum penalty we propose the following work for future. In our proposed architecture, Assume that a block of n core processor on a chip is technologically feasible, and its cost is C. Assume also that the latency in accessing a shared variable is t. then the system can be expanded to include n blocks (n^2cores) at a cost of nC and latency 1.5 t in accessing a shared variable.

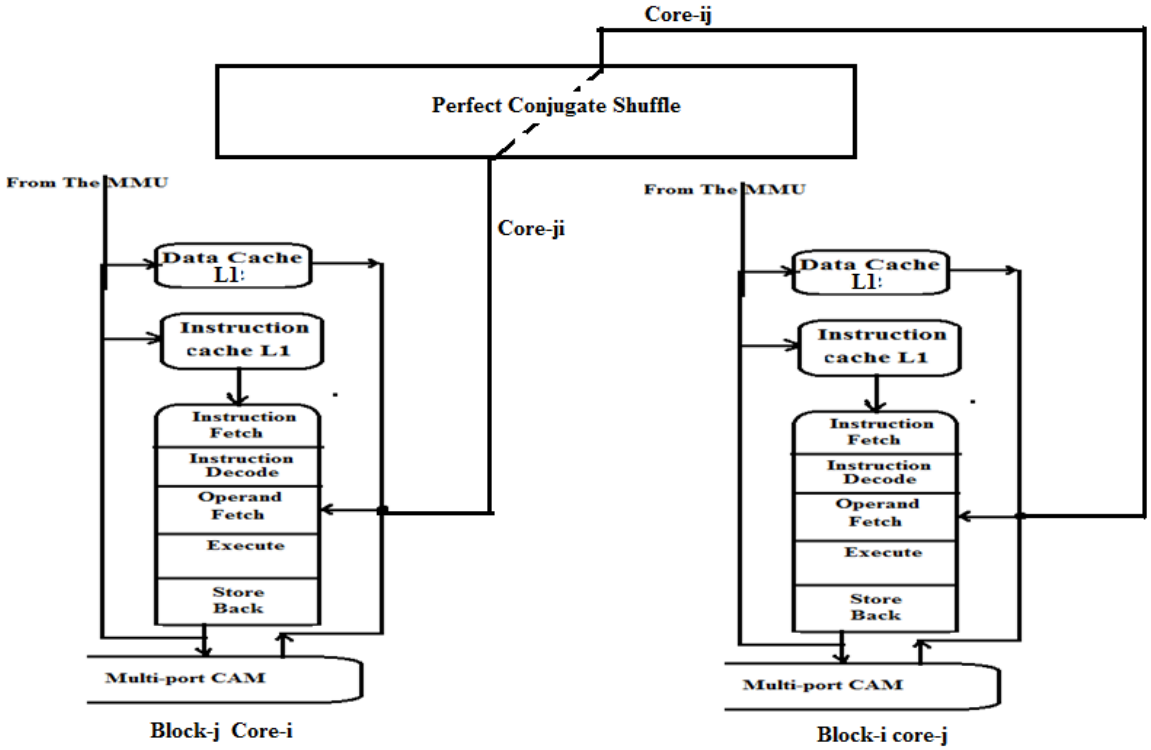

Figure 5.1: Connecting to Cores in Two Different Blocks via the Perfect Conjugate Shuffle

In this system, in the block j, the ith core is a client of block i, and in block i the jth core is a client of block j. obviously any core in the block is capable of accessing any shared variable. So, if a shared variable is produced in block j and needed by block i, then the operand fetch unit of core-ij of block i can take over the operand fetch unit of core-ji and fetches the shared variable, uses it and store (using the store back unit) it in the multi-port CAM i as shown in figure 5.1. Example, if 32 core block is feasible at cost 100$ and latency 6 ns, then the system can be expanded to 32 blocks (1024 cores) at cost 3200$ and 9 ns latency in shared variable access. Intuitively, only two cores will be competing for the shared variable and this in average gives 1.5t latency. Also, no arbitration or synchronization is needed. In future work the simulation of this system with large number of cores paves the way for producing many core processor systems.

Glossary

MP: Multiple Processors

CPUs: Central Processing Units

MIMD: Multiple Instruction Stream Multiple Data Stream

PEs: Processing Elements

IC: Integrated Circuit

SoC: System on a Chip

DTM: Dynamic Thermal managements

L1: Level one of caches

L1D: Level one for Data cache

L1I: Level one for Instruction cache

LL: Last level of cache

LRU: Least Recently Used algorithm

CAM: Content Addressable Memory

DPCAM: Dual Port CAM

MPCAM: Multi Port CAM

OCM: On Chip Multiprocessor

SSM: Symmetric Shared Memory

DSM: Distributed Shared Memory

MESI: Modified, Exclusive, Shared , Invalid. Cache coherence protocol

MSI: Modified, Shared, Invalid. Cache coherence protocol

ESI: Exclusive, Shared, Invalid. Cache coherence protocol

NoC: Network on a Chip

SMM: Shared Memory Module

CBF: Critical Internal Buffer first

IQ: Input Queued

IBC: Internally Buffered Crossbar

MINs: Multistage Interconnection Networks

FPGA: Field Programmable Gate Array

RAM: Random Access memory

PCB: Printed Circuit Board

SB: Store Back. Pipeline stages

OF: Operand Fetch. Pipeline stages

VHDL: Verilog Hardware Description Language

QPI: Quick path interconnection

Bibliography

[A.Cri05] A. Cristal, O.J. Santana, F. Cazorla, M. Galluzzi, T. Ramirez, M. Pericas, and M. Valero, "Kilo-instruction Processors: Overcoming the Memory Wall." IEEE, Vol. 25, No. 3, pp. 48-57, May-June 2005 .

[Alt08] Altera ," Stratix III Development Kit", Document Version: 1.1, San Jose, CA 95134,Agest 2008 (www.altera.com)

[Alt13] Altera ,"Stratix III 3SL150 Development Board ", Refernce Manual , San Jose, CA 95134,May 2013

[Altera] The homepage of Altera. http:// http://www.altera.com/

[AMD07] AMD. Family 10h AMD Phenom^{TM} Processor Product Data Sheet. Advanced Micro Devices, revision 3^{d} edition, November 2007.

[Ara12] Arash Azizi Mazreah"Low-leakage soft error tolerant dualport SRAM cells for cache memory applications"Microelectronics Journal,Vol 43,pp.766-792,November 2012

[Arch86] J. Archibald and J.L. Baer, "Cache Coherence protocols: Evaluation Using a Multiprocessor Simulation Model", ACM transaction on Computer System, Vol 4, No.4, PP 273-298, 1986.

[Ayy93] A. Ayyad, "The Design of a Macro-Dataflow Computer System", PhD thesis, Brighton University, UK, 1993.

[Ayy96] A. Ayyad, I. Exman, M. Land, L. Rudolph, "An Experimental Cross-Bar Switch For Support Of Collective Communications In Parallel Processing",

Proceedings of the Nineteenth Convention of IEEE in Israel, November 5-6, 1996.

[Bar08] Baryan Schauer, " Multicore processor –A Necessity ", September 2008 .

[Barry96] Barry Wilkinson, "Computer Architecture; Design and Performance", 2nd edition, Prentice Hall Europe, 1996.

[Bat98] B. Baterman, C. Freeman nad E. Resse, "A 450MHz 512KB second-Level cache with a 3,9GB/s data bandwidth ", 1998.

[Bhu89] L. Bhuyan and D. Agrawal, "Design and performance of Generalised shuffle networks", IEEE Transaction on Computer, Vol. C-32, No.12, Dec, 1989.

[C.D88] C.D Polychronopoulos and D. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers", IEEE transaction on Computer, Vol 36, No.12, 1988, PP 1425-1439.

[Car02] Carlos Carvalho, "The Gap between Processor and Memory Speeds", University of do Minho, 2002.

[Cha10] Chao Xu and Ying Ding, "An Improved Static-Priority Scheduling Algorithm for Multi-Processor Real-Time Systems", master thesis, University of Gothenburg, UK, 2010.

[Che10] Cheng-Yang Fu, Meng-Huan Wu, and Ren-Song Tsay," A Shared-Variable-Based Synchronization Approach to Efficient Cache Coherence Simulation for Multi-Core Systems", 2011.

[Chi06] Chiara Francalanci and Paolo Giacomazzi,"High-performance self-routing algorithm for multiprocessor systems with shuffle interconnection networks",IEEE Trans.on parallel and Destributed system,VOL.17,NO. 1, january 2006 .

[D.E99] D.E. Culler and J. P. Singh, "Parallel Computer Architecture; A Hardware/ Software Approach", Morgan Kaufmann Publishers, Inc., San Francisco, California, USA, 1999.

[Edw86] R. Edwards, "Futurebus- The Independent Standard for 32 bit Systems", Microprocessors and Microsystems, Vol 10, No, 2 March, 1986.

[Fang90] Z.Fang. Tang, P-C. Yew, and C.Zho, "dynamic processor Self-Scheduling for General Parallel Nested Loops", IEEE transaction on Computer, Vol 39, No.7, 1990, PP 410-420.

[Fira01] Firas Al-Ali and Chris Jesshope, "Survey of High-Latency Tolerance in Future Microprocessor Architecture", Proceedings of the 4th New Zealand Computer Science Research Students' Conference (NZCSRSC2001), pages 86-97, Christchurch, New Zealand, University of Canterbury, April 19th & 20th, 2001, TR-COSC 02/01.

[Fra08] Robert Franz and Josef Weidendorfer, "Development of a multicore cache simulator for performance analysis", bachelor thesis, Technical University Munich , july 2008.

[Gua09] Guangyu Sun, Xiangyu Dong and Yiran Chen, "A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs", 2009.

[Has06] Hasasneh N M, Bell, I and Jesshope C. R., "Scalable and Partitionable Asynchronous Arbiter for Micro-threaded Chip Multiprocessors", (Frankfurt/Main), 2006.

[Hassa05] Hassan Bajwa and Xinghao Chen,"Area-Efficient Dual-Port Memory Architecture for Multi-Core Processors", City University of New York, New York, NY, USA,2005

[Hwa93] Kai Hwang, “Advanced Computer Architecture; Parallelism, Scalability and Programmability”, McGraw-Hill Inc., USA, 1993.

[Hyu09] Hyunhee Kim, Sungjune Youn, Jihong Kim: “Reusability-aware cache memory sharing for chip multiprocessors with private L2 caches”,Vol 55,pp. 446-456”, Octobet-December 2009 .

[Imr07] Imran-Rafiq Quadri , Pierre Boulet and Jean-Luc Dekeyser, “Modeling of Topologies of Interconnection Networks ”, INRIA Report , May 2007 .

[Intel12] Intel® 64 and IA-32, Architectures Optimization Reference Manual, Order Number: 248966-026, April 2012.

[Jae04] Jaehyuk Huh , Jichuan Changy, Doug Burger and Gurindar S. Sohi ,” Coherence Decoupling: Making Use of Incoherence”, Appears in the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, Massachusetts, USA.,2004

[jess01] Jesshope, C. R., “Implementing an efficient vector instruction set in a chip multiprocessor using micro-threaded pipelines”, Proc. ACSAC 2001, Australia Computer Science Communications, Vol 23, No 4., pp80-88, IEEE Computer Society (Los Alimitos, CA, USA), ISBN 0-7695-0954-1

[jess03] Jesshope C. R.,”Multithreaded microprocessors evolution or revolution” (Keynote presentation), *Proc. ACSAC 2003:* Advances in Computer Systems Architecture, *Omondo and Sedukhin* (Eds.), pp 21-45, Springer, LNCS 2823 (Berlin, Germany), ISSN0302-9743, Aizu, Japan, 22-26 Sept 2003.

[jess96] A Bolychevsky, C R Jesshope and V B Muchnick, “Dynamic scheduling in RISC architectures”, IEE Trans. E, Computers and Digital Techniques, 143, 1996, pp309-317.

[Joh 07] John L.Hennessy and David A.Patterson,"Computer Architecture A Quantitative Approach", Fourth Edition Standford University, 2007.

[K.W99] K.Wang, "Design and implementation of fault-tolerant and costeffective crossbar switches for multiprocessor systems", IEE Proc.-Comput. Digit. Tech., Vol. 146, No. I , January I999

[Kei98] Keith Diefendorff, "K7 challenges Intel. New AMD processor could beat Intel Katamai", Microprocessor report, 1998.

[Kos06] Kostas Pagiamtzis,"Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey", IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 3, MARCH 2006

[Lan82] T.Lang, M. Valero, and I. Alegre, "Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors", IEEE Transactions on Computer, Vol C-31, No. 12, Dec 1982, Pages 1227-1234.

[Lin99] Linley Gwennap, "G4 First PowerPC with AltiVec. Due Mid-1999, Motorola Next Chip Aims at Macintosh, Networking", Microprocessor report, 1999.

[Lot05] Lotfi Mhamdi , Mounir Hamdi , Christopher Kachris , Stephan Wong and Stamatis Vassiliadis ," High-performance switching based on buffered crossbar fabrics" , Delft University , April 2005;

[Luo02] Luo B and Jesshope. C, "Performance of a Micro-threaded Pipeline", Proc. ACSAC 2002 *Australia Computer Science Communications*, Vol 24,

[M.Ab12] M. Abragam Siyon Sing and K. Vidya, "Reconfigurable Embedded Multiprocessor Architecture with ARISE interface using FPGA ", International Journal of Scientific & Engineering Research, VOL.3,NO. 1, january 2012.

[M.Zah10] M. Zhang, A. R. Lebeck, and D. J. Sorin," Fractal Coherence: Scalably Verifiable Cache Coherence ". IEEE/ACM International Symposium on Microarchitecture, Dec. 2010.

[Mad84] T.N. Mudge, J.P Hayes, "Analysis of multiple bus interconnection networks" International conference on parallel processing1984, pages 228-232.

[Mil12] Milo M. K. Martin , Mark D. Hill and Daniel J. Sorin," Why On-Chip Cache Coherence is Here to Stay",2012 .

[Moo74] G. E. Moore,"Cramming more components onto integrated circuits", Electronics, April 1974.

[Moor02] Moore, S., Taylor, G., Mullins, R. and Robinson, "Point to Point GALS Interconnect. Proc. of the Eighth International Symposium on Asynchronous Circuits and Systems" , 2002

[Neh11] Michael E. Thomadakis, Ph.D. "The Architecture of the Nehalem Processor And Nehalem-EP SMP Platforms" , A Research Report ,Texas A&M University , March, 2011

[**Ope08**] The homepage of openmp. http://openmp.org, 2008

[Pat97] Patterson D., Anderson T., et al., "A Case for Intelligent RAM: IRAM" IEEE Micro, 1997.

[Pro96] J. Protic, et.al," Distributed shared Memory; Concept and Systems", IEEE parallel and distributed technology, 1996.

[Rig02] Rigaud, J-B., Quartana, J., Fesquet, L. and Renaudin, "High-Level Modeling and Design of Asynchronous Arbiters for On-Chip Communication Systems", IEEE, 2002

[Rob08] Robert Franz ," Development of a multicore cache simulator for Performance analysis", master thesis, Technische University, 2008.

[Sha02] Shashank Gupta, Stephen and Doug Burger, "Technology Independent Area and Delay Estimates for Microprocessor Building Blocks", 2000.

[Sha96] Shaul Dar ,Michael J. Franklin Bjorn.T. J6nssont and Divesh Srivastava," Semantic Data Caching and Replacement", Proceedings of the 22nd VLDB Conference , India, 1996

[Ste02] Steve J.Chapin and Jon B. Weissman,, "Destributeed and Multiprocesor Sceduling ",University of Minnesota and Syracuse University, 2002.

[Theo02] Theo Ungerer , Borut Robic and Jurij Silc, " Multithreaded processors ", the computer journal , Inc., University of Augsburg , VOL.45,NO. 3, 2002.

[Tor10] Gabriel Torres, "Inside the AMD Bulldozer Architecture", hardwaresecrets.com, retrieved December 16, 2010.

[valgrind] The homepage of Valgrind. http://www.valgrind.org

[Var89] A. Varma and C. Raghavendra, "Fault-Tolerant Routing in MIN", IEEE Transaction on Computer, Vol. 38, No.3, March, 1989.

[Xia11] Xiaowen Chen , Shuming Chen and Zhonghai Lu "Hybrid Distributed Shared Memory Space in Multi-core Processors" , Journal , VOL.6,NO. 12, DECEMBER 2011.

الموضوع: هيكلية غير صادة تتضمن شبكة وذاكرة سريعة للمعالجات متعددة النوى

إعداد:علام ربحي محمد ابوموىس

إشراف: د.عبدالكريم عياد

الملخص

تعاني انظمة الحاسبات متعددة المعالجات المبنية على لوحات مطبوعة أو المبنية على شريحة سيلكون متكاملة وتدعى (متعددة النوى)(Multi-Core) من مشكلة عنق الزجاجة في الاتصال بين المعالجات (المنوى) والذاكرة المشتركة. ففي افضل انواع الشبكات (مفاتيح التقاطع)(Crossbar Switch) يتم قبول طلب خدمة واحدة فقط اذا كانت الطلبات متجهة لنفس وحدة الذاكرة (Memory Module) ويجري رفض الباقي حيث يقوم جهاز التحكيم بجدولة خدماتها في دورات لاحقة.هذا يؤدي الى تعويق عمل المعالجات بانتظار خدمة طلباتها من الذاكرة المشتركة وبالتالي يطيل مدة معالجة النظام للبرنامج. ويضيف الى هذا بشكل حاد في عملية تنسيق وتحديث المتغيرات المشتركة في الذاكرة المشتركة حيث تضاف كطلبات اضافية الى الشبكة.

وكجزء من مشروع هذه الرسالة المتضمن تصميم وتشبيه معالج متعدد النوى تم اعادة تصميم وحدات الذاكرة السريعة من النوع الذي يعنون فية المحتوى لجزء منه, بحيث يكون لها بناء من ميناءين واحد للكتابة واخر للقراءة وتم زراعتها على تقاطعات شبكة مفاتيح التقاطع بدل المفاتيح فكانت النتيجة ذاكرة معنونة بالمحتوى متعدد الموانئ. هذه الذاكرة ازالت عنق الزجاجة من النظام ولم يعد هناك تنافس ولا تحكيم ولا تعويق لعمل المعالجات (النوى) حيث تستطيع جميع المعالجات الوصول الى المعلومة بالتوازي في نفس الوقت. كما ان طبيعة الكتابة الى هذه الذاكرة تضمن وجود جميع النسخ من المعلومة وبالتالي لا حاجة لعملية تنسيق وتحديث المتغيرات وهذا ازال عبئاً معوقاً اضافياً.

واخيراً تم اقتراح نظام لتوسيع هذا النظام بحيث يحتوي عدد كبير من النوى وبكلفة تنمو خطياً مع عدد المنوى وبوقت وصول ثابت وصغير جداً لا يتجاوز مرة ونصف زمن الوصول الى المتغير في هذه الذاكرة متعددة الموانئ.

Appendices

Appendix A

Appendix A-1: date program benchmarking

A-1: 1.1 AMD simulator eight cores

```
==2583== Number of simulated cores: 8
==2583== I   refs:      305,033
==2583== I1  misses:      1,361
==2583== L2i misses:      1,308
==2583== I1  miss rate:    0.44%
==2583== L2i miss rate:    0.42%
==2583==
==2583== D   refs:      153,947  (108,429 rd + 45,518 wr)
==2583== D1  misses:      2,557  (  2,130 rd +    427 wr)
==2583== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2583== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
==2583== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2583==
==2583== L2 refs:         3,918  (  3,491 rd +    427 wr)
==2583== L2 misses:       3,142  (  2,779 rd +    363 wr)
==2583== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2583==
==2583== Multicore Cache Simulator
==2583== Thread Stats: Instructions
==2583== Thread 0:305033
==2583== Thread 1:0
==2583== Thread 2:0
==2583== Thread 3:0
==2583== Thread 4:0
==2583== Thread 5:0
==2583== Thread 6:0
==2583== Thread 7:0
```

A-1: 1.2 new proposed simulator eight cores

```
==2601== Number of simulated cores: 8
==2601== I   refs:      304,917
==2601== I1  misses:      1,359
==2601== L2i misses:      1,307
==2601== I1  miss rate:    0.44%
==2601== L2i miss rate:    0.42%
==2601==
==2601== D   refs:      153,870  (108,382 rd + 45,488 wr)
==2601== D1  misses:      2,557  (  2,130 rd +    427 wr)
```

```
==2601== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2601== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
==2601== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2601==
==2601== L2 refs:         3,916  (  3,489 rd +    427 wr)
==2601== L2 misses:       3,141  (  2,778 rd +    363 wr)
==2601== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2601==
==2601== New Multicore Simulator
==2601== Thread Stats: Instructions
==2601== Thread 0:304917
==2601== Thread 1:0
==2601== Thread 2:0
==2601== Thread 3:0
==2601== Thread 4:0
==2601== Thread 5:0
==2601== Thread 6:0
==2601== Thread 7:0
```

A-1: 2.1 AMD simulator four cores

```
==2586== Number of simulated cores: 4
==2586== I   refs:      304,917
==2586== I1  misses:      1,359
==2586== L2i misses:      1,307
==2586== I1  miss rate:    0.44%
==2586== L2i miss rate:    0.42%
==2586==
==2586== D   refs:      153,870  (108,382 rd + 45,488 wr)
==2586== D1  misses:      2,557  (  2,130 rd +    427 wr)
==2586== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2586== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
==2586== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2586==
==2586== L2 refs:         3,916  (  3,489 rd +    427 wr)
==2586== L2 misses:       3,141  (  2,778 rd +    363 wr)
==2586== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2586==
==2586== Multicore Cache Simulator
==2586== Thread Stats: Instructions
==2586== Thread 0:304917
==2586== Thread 1:0
==2586== Thread 2:0
==2586== Thread 3:0
```

A-1: 2.2 new proposed simulator four cores

```
==2610== Number of simulated cores: 4
==2610== I   refs:      304,917
==2610== I1  misses:      1,359
==2610== L2i misses:      1,307
==2610== I1  miss rate:    0.44%
==2610== L2i miss rate:    0.42%
==2610==
```

```
==2610== D   refs:      153,870  (108,382 rd + 45,488 wr)
==2610== D1  misses:      2,557  (  2,130 rd +    427 wr)
==2610== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2610== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
==2610== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2610==
==2610== L2 refs:         3,916  (  3,489 rd +    427 wr)
==2610== L2 misses:       3,141  (  2,778 rd +    363 wr)
==2610== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2610==
==2610== New Multicore Simulator
==2610== Thread Stats: Instructions
==2610== Thread 0:304917
==2610== Thread 1:0
==2610== Thread 2:0
==2610== Thread 3:0
```

A-1: 3.1 AMD simulator two cores

```
==2589== Number of simulated cores: 2
==2589== I   refs:      304,917
==2589== I1  misses:      1,359
==2589== L2i misses:      1,307
==2589== I1  miss rate:    0.44%
==2589== L2i miss rate:    0.42%
==2589==
==2589== D   refs:      153,870  (108,382 rd + 45,488 wr)
==2589== D1  misses:      2,557  (  2,130 rd +    427 wr)
==2589== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2589== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
==2589== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2589==
==2589== L2 refs:         3,916  (  3,489 rd +    427 wr)
==2589== L2 misses:       3,141  (  2,778 rd +    363 wr)
==2589== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2589==
==2589== Multicore Cache Simulator
==2589== Thread Stats: Instructions
==2589== Thread 0:304917
==2589== Thread 1:0
```

A-1: 3.2 new proposed simulator two cores

```
==2615== Number of simulated cores: 2
==2615== I   refs:      304,917
==2615== I1  misses:      1,359
==2615== L2i misses:      1,307
==2615== I1  miss rate:    0.44%
==2615== L2i miss rate:    0.42%
==2615==
==2615== D   refs:      153,870  (108,382 rd + 45,488 wr)
==2615== D1  misses:      2,557  (  2,130 rd +    427 wr)
==2615== L2d misses:      1,834  (  1,471 rd +    363 wr)
==2615== D1  miss rate:     1.6% (    1.9%   +    0.9%  )
```

```
==2615== L2d miss rate:     1.1% (    1.3%   +    0.7%  )
==2615==
==2615== L2 refs:         3,916  (  3,489 rd +    427 wr)
==2615== L2 misses:       3,141  (  2,778 rd +    363 wr)
==2615== L2 miss rate:      0.6% (    0.6%   +    0.7%  )
==2615==
==2615== New Multicore Simulator
==2615== Thread Stats: Instructions
==2615== Thread 0:304917
==2615== Thread 1:0
```

Appendix A-2: df program benchmarking

A-2: 1.1 AMD simulator eight cores

```
==2594== Number of simulated cores: 8
==2594== I   refs:      398,496
==2594== I1  misses:      1,630
==2594== L2i misses:      1,478
==2594== I1  miss rate:    0.40%
==2594== L2i miss rate:    0.37%
==2594==
==2594== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2594== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2594== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2594== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2594== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2594==
==2594== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2594== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2594== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2594==
==2594== Multicore Cache Simulator
==2594== Thread Stats: Instructions
==2594== Thread 0:398496
==2594== Thread 1:0
==2594== Thread 2:0
==2594== Thread 3:0
==2594== Thread 4:0
==2594== Thread 5:0
==2594== Thread 6:0
==2594== Thread 7:0
```

A-2: 1.2 new proposed simulator two cores

```
==2622== Number of simulated cores: 8
==2622== I   refs:      398,496
==2622== I1  misses:      1,630
==2622== L2i misses:      1,478
==2622== I1  miss rate:    0.40%
==2622== L2i miss rate:    0.37%
==2622==
==2622== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2622== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2622== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2622== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2622== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2622==
==2622== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2622== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2622== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2622==
==2622== New Multicore Simulator
==2622== Thread Stats: Instructions
==2622== Thread 0:398496
==2622== Thread 1:0
==2622== Thread 2:0
==2622== Thread 3:0
==2622== Thread 4:0
==2622== Thread 5:0
==2622== Thread 6:0
==2622== Thread 7:0
```

A-2: 2.1 AMD simulator four cores

```
==2596== Number of simulated cores: 4
==2596== I   refs:      398,496
==2596== I1  misses:      1,630
==2596== L2i misses:      1,478
==2596== I1  miss rate:    0.40%
==2596== L2i miss rate:    0.37%
==2596==
==2596== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2596== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2596== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2596== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2596== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2596==
==2596== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2596== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2596== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2596==
==2596== Multicore Cache Simulator
==2596== Thread Stats: Instructions
==2596== Thread 0:398496
==2596== Thread 1:0
==2596== Thread 2:0
```

```
==2596== Thread 3:0
```

A-2: 2.2 new proposed simulator four cores

```
==2627== Number of simulated cores: 4
==2627== I   refs:      398,496
==2627== I1  misses:      1,630
==2627== L2i misses:      1,478
==2627== I1  miss rate:    0.40%
==2627== L2i miss rate:    0.37%
==2627==
==2627== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2627== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2627== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2627== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2627== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2627==
==2627== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2627== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2627== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2627==
==2627== New Multicore Simulator
==2627== Thread Stats: Instructions
==2627== Thread 0:398496
==2627== Thread 1:0
==2627== Thread 2:0
==2627== Thread 3:0
```

A-2: 3.1 AMD simulator two cores

```
==2598== Number of simulated cores: 2
==2598== I   refs:      398,496
==2598== I1  misses:      1,630
==2598== L2i misses:      1,478
==2598== I1  miss rate:    0.40%
==2598== L2i miss rate:    0.37%
==2598==
==2598== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2598== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2598== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2598== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2598== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2598==
==2598== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2598== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2598== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2598==
==2598== Multicore Cache Simulator
==2598== Thread Stats: Instructions
==2598== Thread 0:398496
==2598== Thread 1:0
```

A-2: 3.2 new proposed simulator two cores

```
==2627== Number of simulated cores: 4
==2627== I   refs:      398,496
==2627== I1  misses:      1,630
==2627== L2i misses:      1,478
==2627== I1  miss rate:    0.40%
==2627== L2i miss rate:    0.37%
==2627==
==2627== D   refs:      193,321  (133,782 rd + 59,539 wr)
==2627== D1  misses:      2,213  (  1,837 rd +    376 wr)
==2627== L2d misses:      1,691  (  1,370 rd +    321 wr)
==2627== D1  miss rate:     1.1% (    1.3%   +    0.6%  )
==2627== L2d miss rate:     0.8% (    1.0%   +    0.5%  )
==2627==
==2627== L2 refs:         3,843  (  3,467 rd +    376 wr)
==2627== L2 misses:       3,169  (  2,848 rd +    321 wr)
==2627== L2 miss rate:      0.5% (    0.5%   +    0.5%  )
==2627==
==2627== New Multicore Simulator
==2627== Thread Stats: Instructions
==2627== Thread 0:398496
==2627== Thread 1:0
```

Appendix B

Appendix B: (pp) program benchmarking

B: 1.1 AMD simulator two cores

```
==2664== Number of simulated cores: 2
==2664== I   refs:      130,293,891
==2664== I1  misses:          1,126
==2664== L2i misses:          1,106
==2664== I1  miss rate:         0.0%
==2664== L2i miss rate:         0.0%
==2664==
==2664== D   refs:       60,696,770  (70,103,936 rd + 20,044,724 wr)
==2664== D1  misses:      2,508,981  (1,007,305 rd +   1,501,676 wr)
==2664== L2d misses:      1,004,927  (1,004,802 rd +         842 wr)
==2664== D1  miss rate:         4.1% (       3.0%   +        5.5%  )
==2664== L2d miss rate:         1.6% (       2.9%   +        0.0%  )
==2664==
==2664== L2  refs:      2,510,156     ( 1,008,480 rd +  1,501,676 wr)
==2664== L2 misses:     1,006,044     ( 1,005,202 rd +        842 wr)
==2664== L2 miss rate:          0.5% (        0.6%   +        0.0% )
==2664== Multicore Cache Simulator
==2664== Thread Stats: Instructions
==2664== Thread 0:65290662
==2664== Thread 1:65003229
==2664== Bus Transactions:
==2664== Thread 0: Reads:146185; Writes: 10044084; Shared Reads:
859012; Invalidation: 1141244
==2664== Thread 1: Reads: 859031; Writes: 10000642; Shared Reads:
141013; Invalidation: 1359245
==2664== Summary:  Reads: 1005216; Writes: 20044726;Shared
Reads:1000025; Invalidation: 2500489
==2664==
==2664== Number of invalidations per write access
==2664== 1;2;<5;>4
==2664== Thread 0: 1359245,0,0,0
==2664== Thread 1: 1141244,0,0,0
==2664== Thread 2: 0,0,0,0
==2664== Thread 3: 0,0,0,0
==2664== Thread 4: 0,0,0,0
==2664== Thread 5: 0,0,0,0
==2664== Thread 6: 0,0,0,0
```

B: 1.2 new proposed simulator two cores

```
==2654== Number of simulated cores: 2
==2654== I   refs:      130,293,891
==2654== I1  misses:          1,126
==2654== L2i misses:          1,106
==2654== I1  miss rate:         0.0%
==2654== L2i miss rate:         0.0%
==2654==
==2654== D   refs:       60,696,770  (70,103,936 rd + 20,044,724 wr)
```

```
==2654== D1  misses:           2,608 (     2,080 rd +        528 wr)
==2654== L2d misses:           1,821 (     1,244 rd +        577 wr)
==2654== D1  miss rate:          0.0% (       0.0%   +        0.0%  )
==2654== L2d miss rate:          0.0% (       0.0%   +        0.0%  )
==2654==
==2654== L2  refs:        2,656      (     2,108 rd +        548 wr)
==2654== L2 misses:       2,244      (     1,898 rd +        346 wr)
==2654== L2 miss rate:          0.0% (        0.0%   +         0.0% )
==2654== New Multicore Simulator
==2654== Thread Stats: Instructions
==2654== Thread 0:65290662
==2654== Thread 1:65003229
==2654== Bus Transactions:
==2654== Thread 0: Reads:146185; Writes: 10044084; Shared Reads: 00;
Invalidation: 22
==2654== Thread 1: Reads: 859031; Writes: 10000642; Shared Reads: 00;
Invalidation: 21
==2654== Summary:  Reads: 1005216; Writes: 20044726; Shared Reads:
00; Invalidation: 43
```

B: 2.1 AMD simulator four cores

```
==2684== Number of simulated cores: 4
==2684==
==2684== I   refs:      130,297,832
==2684== I1  misses:          1,273
==2684== L2i misses:          1,253
==2684== I1  miss rate:         0.0%
==2684== L2i miss rate:         0.0%
==2684==
==2684== D  refs:       68,696,915 (37,987,746 rd + 30,709,169 wr)
==2684== D1 misses:      3,259,033 ( 1,507,358 rd +  1,751,675 wr)
==2684== L2d misses:     1,505,020 ( 1,504,130 rd +        890 wr)
==2684== D1 miss rate:         4.7% (       3.9%   +        5.7%  )
==2684== L2d miss rate:        2.1% (       3.9%   +        0.0%  )
==2684==
==2684== L2 refs:        3,260,274 ( 1,508,599 rd +  1,751,675 wr)
==2684== L2 misses:      1,506,206 ( 1,505,316 rd +        890 wr)
==2684== L2 miss rate:         0.7% (      0.8%    +        0.0%  )
==2684==
==2684== Multicore Cache Simulator
==2684== Thread Stats: Instructions
==2684== Thread 0:32791387
==2684== Thread 1:32501060
==2684== Thread 2:32503187
==2684== Thread 3:32502198
==2684== Bus Transactions:
==2684== Thread 0: Reads:196996; Writes: 7764630; Shared Reads:
597517;
Invalidation: 807424
==2684== Thread 1: Reads:435502; Writes: 7650724; Shared Reads:
554033;
Invalidation: 813903
==2684== Thread 2: Reads:436057; Writes: 7651259; Shared Reads:
551164;
```

```
Invalidation: 814836
==2684== Thread 3: Reads:436775; Writes: 7642569; Shared Reads:
547525;
Invalidation: 814297
==2684== Summary: Reads: 1505330; Writes: 30709182; Shared Reads:
2250239;
Invalidation: 3250460
```

B: 2.2 new proposed simulator four cores

```
==3507== Number of simulated cores: 4
==3507==
==3507== I   refs:      130,297,832
==3507== I1  misses:          1,273
==3507== L2i misses:          1,253
==3507== I1  miss rate:         0.0%
==3507== L2i miss rate:         0.0%
==3507==
==3507== D   refs:       68,696,915  (37,987,764 rd + 30,709,169 wr)
==3507== D1  misses:          3,052  (     2,467 rd +        585 wr)
==3507== L2d misses:          2,298  (     1,785 rd +        513 wr)
==3507== D1  miss rate:         0.0% (       0.0%   +        0.0%  )
==3507== L2d miss rate:         0.0% (       0.0%   +        0.0%  )
==3507==
==3507== L2 refs:             4,093  (     3,613 rd +        480 wr)
==3507== L2 misses:           3,343  (     2,924 rd +        419 wr)
==3507== L2 miss rate:          0.0% (       0.0%   +        0.0%  )
==3507==
==3507== New Multicore Simulator
==3507== Thread Stats: Instructions
==3507== Thread 0:32791387
==3507== Thread 1:32501060
==3507== Thread 2:32503187
==3507== Thread 3:32502198
==3507== Bus Transactions:
==3507== Thread 0: Reads:196996; Writes: 7764630; Shared Reads:00;
Invalidation: 38
==3507== Thread 1: Reads:435502; Writes: 7650724; Shared Reads:00;
Invalidation: 18
==3507== Thread 2: Reads:436057; Writes: 7651259; Shared Reads: 00;
Invalidation: 25
==3507== Thread 3: Reads:436775; Writes: 7642569; Shared Reads: 00;
Invalidation: 24
==3507== Summary:  Reads: 1505330; Writes: 30709182; Shared Reads:
00; Invalidation: 105
```

B: 3.1 AMD simulator four cores

```
==2696== Number of simulated cores: 8
==2696== I   refs:      130,305,664
==2696== I1  misses:          1,492
==2696== L2i misses:          1,472
==2696== I1  miss rate:         0.0%
==2696== L2i miss rate:         0.0%
```

```
==2696==
==2696== D   refs:       72,703,459  (40,241,090 rd  + 32,462,369 wr)
==2696== D1  misses:      3,634,995  ( 1,757,390 rd  +  1,877,605 wr)
==2696== L2d misses:      1,755,134  ( 1,754,148 rd  +        986 wr)
==2696== D1  miss rate:         4.9% (       4.3%    +         57%  )
==2696== L2d miss rate:         2.4% (       4.3%    +         0.0% )
==2696==
==2696== L2 refs:         3,636,368  ( 1,758,763 rd  +  1,877,605 wr)
==2696== L2 misses:       1,756,452  ( 1,755,466 rd  +        986 wr)
==2696== L2 miss rate:          0.8% (       0.9%    +        0.0%  )
==2696==
==2696== Multicore Cache Simulator
==2696== Thread Stats: Instructions
==2696== Thread 0:16544757
==2696== Thread 1:16251060
==2696== Thread 2:16252374
==2696== Thread 3:16251083
==2696== Thread 4:16251060
==2696== Thread 5:16253187
==2696== Thread 6:16251083
==2696== Thread 7:16251060
==2696== Bus Transactions:
==2696== Thread 0: Reads:113223; Writes: 4208223; Shared Reads:
551199;
Invalidation: 448046
==2696== Thread 1: Reads:234511; Writes: 4035640; Shared Reads:
543774;
Invalidation: 453595
==2696== Thread 2: Reads:234674; Writes: 4037809; Shared Reads:
548067;
Invalidation: 454361
==2696== Thread 3: Reads:234758; Writes: 4038131; Shared Reads:
548596;
Invalidation: 454495
==2696== Thread 4: Reads:234279; Writes: 4026468; Shared Reads:
540909;
Invalidation: 452352
==2696== Thread 5: Reads:234670; Writes: 4038398; Shared Reads:
549506;
Invalidation: 454428
==2696== Thread 6: Reads:235007; Writes: 4043821; Shared Reads:
549973;
Invalidation: 455544
==2696== Thread 7: Reads:234358; Writes: 4033884; Shared Reads:
544045;
Invalidation: 453488
==2696== Summary: Reads: 1755480; Writes: 32462374; Shared
Reads:4376069;
Invalidation: 3626309
```

B: 3.2 new proposed simulator four cores

```
==2706== Number of simulated cores: 8
==2706== I   refs:      130,305,664
==2706== I1  misses:          1,492
```

```
==2706== L2i misses:          1,472
==2706== I1  miss rate:         0.0%
==2706== L2i miss rate:         0.0%
==2706==
==2706== D   refs:       72,703,459  (40,241,090 rd  + 32,462,369 wr)
==2706== D1  misses:          4,432  (     3,269 rd  +      1,136 wr)
==2706== L2d misses:          3,142  (     2,616 rd  +        526 wr)
==2706== D1  miss rate:         0.0% (       0.0%    +        0.0%  )
==2706== L2d miss rate:         0.0% (       0.0%    +        0.0%  )
==2706==
==2706== L2 refs:             6,298  (     5,413 rd +         885 wr)
==2706== L2 misses:           4,465  (     3,424 rd +       1,041 wr)
==2706== L2 miss rate:          0.0% (       0.0%   +         0.0%  )
==2706==
==2706== New Multicore Simulator
==2706== Thread Stats: Instructions
==2706== Thread 0:16544757
==2706== Thread 1:16251060
==2706== Thread 2:16252374
==2706== Thread 3:16251083
==2706== Thread 4:16251060
==2706== Thread 5:16253187
==2706== Thread 6:16251083
==2706== Thread 7:16251060
==2706== Bus Transactions:
==2706== Thread 0: Reads:113223; Writes: 4208223; Shared Reads: 00;
Invalidation: 63
==2706== Thread 1: Reads:234511; Writes: 4035640; Shared Reads: 00;
Invalidation: 19
==2706== Thread 2: Reads:234674; Writes: 4037809; Shared Reads: 00;
Invalidation: 32
==2706== Thread 3: Reads:234758; Writes: 4038131; Shared Reads: 00;
Invalidation: 20
==2706== Thread 4: Reads:234279; Writes: 4026468; Shared Reads: 00;
Invalidation: 19
==2706== Thread 5: Reads:234670; Writes: 4038398; Shared Reads: 00;
Invalidation: 25
==2706== Thread 6: Reads:235007; Writes: 4043821; Shared Reads: 00;
Invalidation: 20
==2706== Thread 7: Reads:234358; Writes: 4033884; Shared Reads: 00;
Invalidation: 18
==2706== Summary:  Reads: 1755480; Writes: 32462374; Shared Reads:
00; Invalidation: 216
```