Design or Architecture of Multicore Multi-threading Processors

Himani Patel – EECS Department - SUID: 495936092 – htpatel@syr.edu

Samarth Shah – EECS Department – SUID: 748609888 – sshah31@syr.edu

1. Title and Abstract:

The new technologies in the current era, which is known as the Information Age, requires lot of power and efficient processor to handle such processes with heavy computations. For this reason, it is necessary to build processors that gives better performance and consumes less time and energy. Current microprocessors support complex design and they are built by integrating distinct components like interconnection networks, processing units, threads and memory units on a single chip. The overall processing performance is improved using multicore processors. Multi-core processor is the processing system which consists of more than one independent core working on a single chip. It supports processing of multiple instructions at the same time on separate cores which increases the overall speed of a processor. Multithreading in multi-core processors boosts up the overall performance of a processor. Multithreading is a technique in which multiple threads are provided to a processing unit for execution. It executes two or more parts of a program simultaneously to utilize the CPU time efficiently. In this paper, we focus on demonstrating and evaluating the architecture or design of such multicore multi-threading processors to understand its functionality and determine the suitable architecture or design which can give the best performance-to-cost ratio. We observed that the need of such in-demand processor is fulfilled by a multicore multi-threading processor.

2. Introduction and Background:

The establishment of Internet has caused a major outbreak in the field of networking which led to the urgent need of large number of server application which are built with multiple threads that serve multiple clients. Also, in the business world of electronics, there are millions of customers who are served with large enterprises that run on applications having contrasting memory and processing characteristics and architecture than that of the desktop applications. Also, due to the number of customers they serve, they require processors that can fulfill such huge amount of data processing and multitasking. Certainly, to handle and assist such large number of clients and customers, these server applications are in need of higher main memory bandwidth and proportionately less efficient behavior of cache.

In today's world, usage of multicore and multithreading systems is ubiquitous. It is now also used widely in mobile and high-end routers across the globe. Implementing

multicore/multithread technology increases the performance significantly.

The advancement in the technology, which enabled the evolution of microprocessors has guided the designs to a complex level. In these designs, multiple processing units are combined on a chip to give the view of having multiple processors to the Operating System which will allow to schedule different software processes simultaneously.

The major components of this processor model include: Cache Hierarchy, Interconnection Network and Microprocessor Core. In any way, if we improve the design of this components, it will lead to a gain in performance over the entire system. Therefore, the trending architecture of the processor brings ample amount of opportunities for the researchers to explore in depth in this direction of new microarchitectural proposals. Beneath, we will discuss about the issues that could be faced during the design of these components.

In this paper, we have studied and analyzed many research works that are referenced below in the References section. Hereby, we present the background and analysis of some systems that can help us in overcoming the issues that we have taken into consideration and are faced by the conventional processors.

As we studied in [1] and [2], the traditional processors were built with the reference of focusing on instruction level parallelism in order to improve the system performance and that is the reason why the size of the processors was immense. The complexity of pipeline structure was also high because of the processor size. Because of the complex pipeline structure, in the processors like INTEL, the die consists of only one core. In the correspondence to this, a long latency event like a branch miss prediction or a cache miss for the main memory causes a stall which makes the pipeline inactive for a noticeable amount of time. Consequently, if the server has to serve applications that have less efficient locality of cache, inaccurate branch predictability and heavy memory footprints, it probably leans towards having small amount of instruction level of parallelism for each thread. The quality of hardware utilization decreases as a result of implementing such characteristics of the traditional processors running applications that have workloads from server. Also, due to the implementation of instruction level parallelism centric conventional processors, lot of power is dissipated which is not necessary and inefficient for a system.

Concisely, using such processors that focus on instruction level parallelism, the overall performance in terms of power, die size and complexity is very much affected and the value is worsening. Below is a graph which states the relationship between the size or power and the performance of processors with respect to instruction level parallelism.

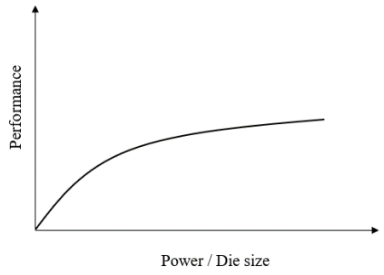


Figure 1. Performance - Power/Die Size Graph

As shown in the above figure, at some point, the performance does not increase linearly with respect to the increasing rate of die area and power of the traditional processors and this is because of the limitations imposed by the architecture of instruction level parallelism which is incorporated by the processors. The processor families like POWER, ITANIUM, PENTIUM and ULTRASPARC are some of the conventional processors that centers instruction level parallelism.

All the problems mentioned above are the reasons and motivation for developing a processor with an appropriate architecture that can utilize multithreading characteristics on a multicore processor to solve the maximum problems for server applications with heavy workloads.

The key constituent to hide the request latency that is off-chip is to establish on-chip last level cache (typically L2, L3 or L4 caches) and it is usually shared by multicores in a CMP i.e. Chip-Multiprocessors architecture. Hence, the performance of simultaneously executing applications on different multiple cores is significantly affected by the last level cache sharing.

The study in [3] intends to exam such adaptivity of the multicore/multiprocessor. Various designs and their effect on the program execution were studied in this paper. Focus was put on configuration parameters and cache structures of the system. The first study of [3] examines if a particular cache architecture is proportional to better performance than the other. The second study in [3] focuses on cache configuration and examining how performance varies with the cache size, associativity and block size. This study, as a whole, also intends on giving researchers in the area considering the quantitative information regarding potential benefits of radical design of this kind.

The complexity of the cache architecture is directly proportional to the number of processors. As the number of processors gets higher, the cache architecture becomes complex. The single shared cache architectures like L2, L3, and L4 bus, ring-shaped architecture and hierarchical bus are profoundly known and they are examined independently.

According to our study in [4], we present a simulator named Multi2Sim, which incorporates the memory hierarchies, networks that are interconnected and processor cores. The present overview of processor simulator, the structure of Multi2Sim and discussion of Multithreading and Multicore Simulation is described in this paper.

3. Related Work:

Environments have been developed to assess the computer architecture proposals. In recent years, Simple-Scalar [7] is the most widely used simulator. Simple-Scalar are modeled from the out-of-order Super-Scalar processors. To design it with more accuracy, many extensions are applied to the Simple-Scalar processors. Like the leakage energy consumption is measured by the HotLeak-Age Simulator.[8]

To demonstrate new parallel microarchitectures without changing the

structure significantly is quite a challenging task in the Simple-Scalar processor. Even after knowing this fact, implementation of various Simple-Scalar extensions has taken place to support multithreading. For example, M-Sim [10], SSMT [9] or SMTSim [11]. But the drawbacks of this extension are that, they have a resource sharing policy which is shared by all and they only execute a bulk of workloads that are sequential.

Extensions for multithread and multicore are also implemented to the Turandot Simulator [12] [13]. The Turandot Simulator is used with the aims of power measurement (PowerTimer [14]) and a PowerPC architecture is also modeled by this simulator. These extensions are not publicly available, and they are mostly cited to parallel microarchitectures.

Contrary to the application-only simulator, full-system simulators are also available. Full-system simulators are capable of booting an unmodified operating system and simultaneously we can also run the application. Full-System Simulators are sometimes involved in unnecessary simulation accuracy and are also involved in huge computational load, but they also provide high simulation power.

One of the examples for Full-System Simulator is SIMICS [15]. SIMICS is generally used for multiprocessor systems simulation but sadly it is not available freely. Multiple tools derived from the SIMICS has been used as an apparatus for the research works in this field. One such tool is GEMS [16], GEMS is used to model a cache coherence that allows memory hierarchy and a complete processor pipeline by introducing the time simulation module to this model. The drawbacks of GEMS are, it does not integrate interconnection network model and does not cooperate in modelling of multithreaded designs.

Timing-First approach is a significant characteristic that should be incorporated in a processor simulator. This approach was provided by GEMS and will be used in Multi2Sim. In the timing-first approach, the state of the processor pipeline is traced by the timing module while they are traversing. Now, to actually execute the instructions, function module is called. By this method, the correct execution path is always guaranteed because of the robust simulator which was developed previously. Robustness and efficiency are the two aspects which confers the timing-first approach. With a functional support from the timing-support simulation, unlike GEMS, our proposal will not stimulate the whole operating system, but still it will be able to execute work-loads parallelly and creating threads dynamically.

The drawbacks for this simulator are not very different than the previous simulators, these simulators also have low flexibility for multithreaded pipeline designs.

4. Analysis:

Our study from [1]and [2] leads to the design or architecture of such multicore multi-threading processors that can efficiently deal with server applications serving multiple clients and customers simultaneously. Here in this study, we provide some representations of the desired architecture of multicore multi-threading processors.

In the first representation, there is a processor consisting of minimum two cores. These cores are built with first level cache memory individually. All of the cores are multi-threading cores. In another representation, all cores consist of precisely four threads with a crossbar. The crossbar is used as a communication medium between the cores and the second level cache memories. These second level cache bank memories communicate with the interface of the main memory. This structure also includes a core with buffer switch that communicates with each second level cache bank memory.

There is another representation which provides a server that consists of a processor chip for applications. This processor chip for applications comprises of several cores having multithreaded central processing units. A first level cache memory is incorporated in each of the cores with multi-threading central processing units. In addition, a crossbar with several cache memories is provided in the processor chip for application which uses the crossbar for carrying out the communication process between the cores and the bank of cache memories. The main memory interface receives communication from each of the level two cache bank memories. These bank of cache memories are in communication with the application processor chip using a core with buffer switch included in the representation.

The last representation consists of a utilization optimizing method for a processor core with multi-threading. Initialization of the method is performed by an operation method in which the processor core is accessed via an operation of the first thread. As a result of completion of accessing the processor core by using the operation of first thread, the first thread carries out a latency operation which is long. After this process ends, the first thread is exempted by the system. Then, a second thread is searched and selected which is ready for performing the operation of accessing the processor core through second thread operation method. At the same time in the background, when the first thread was carrying out the execution of long latency operation, the second selected thread goes under the processing held by the processor core.

These are some representations which can be taken into consideration for building the architecture or design of multi-threaded multicore processors in order to reduce the workload of the systems performing strenuous tasks. We will discuss the representations in more detail with some related diagrams in the following detailed description of the proposed system.

The motive of designing a multicore multi-threading processor is to enhance the processor's overall efficiency and throughput because of the overloading tasks and strain related to the commercial client serving applications. It can be done by proposing some representations and methods mentioned above. The proposed design for the architecture of a processor with multi-threaded multicores can be implemented using some of the details specified here, or also it can be practiced by using some or none of the details. In the detailed description, some of the representations are not discussed as they might create confusion and doubts regarding the proposal unnecessarily. Thus, here we discuss one of the representations that is most relevant, better

performing and optimized solution for the problems faced by the server applications in electronic commerce world. The terms and values used here are in the range of -10 to +10 of the corresponding respective obtained values.

The representations that we discussed earlier have a design that depicts a chip on which there are multiple cores, and these cores have their own separate individual first level caches. It also comprises of a crossbar through which they communicate with the second level cache memory. This second level cache memory is shared by all of the cores. As they are multi-threaded cores, each core contains their own minimum of two threads. The purpose of using multi-threading is that it completely conceals the long latency situations like branch miss prediction, cache misses, memory strains and many more. As we discussed in one of the representations, instructions of the long latency events release the current thread till the outcome of that instruction is known by the system. After the suspension of the current thread, a thread is identified and selected from the system's remaining collection of threads that are already prepared and ready to run. On the next clock, this selected thread is executed into the pipeline. All this is done in the absence of context switching overhead. The other representation states that in each core, the thread from ready to run threads is selected by a scheduling algorithm implemented by the system. This process is advantageous as it helps in increasing the throughput of the architecture and it leads to an optimized use of central processing unit (CPU) which is done by those multiple threads. The increased throughput is noticeably high, and this is attained because the long latency events are carried out in the background.

Now, we dive into the discussion of our selected representation on the basis of relevance and performance. A precise and brief idea can be conveyed by the following symbolic diagram of our proposed design which contains a processor chip having multiple thread containing multiple cores. Each of the cores have their own individual first level caches and they share the second level cache memory among all other cores through a crossbar.

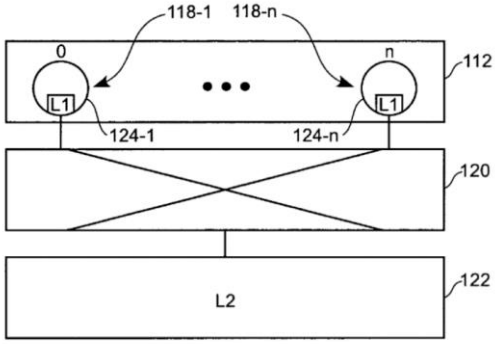


Figure 2. Architectural Design for Multicore Multi-threading Processors

In this figure 2, '112' is the processor chip for applications, the processor cores are numbered from '0' to 'n' which means there are 'n' number of processor cores and they are named as '118-1' and so on till '118-n'. The first level cache is represented by 'L1' which is included in each core separately named as '124-1' and so on till '124-n' respectively for all the 'n' number of first level caches. '120' is for representing the crossbar of the system and '122' is representing the second level cache memory 'L2'.

The processor chip named '112' consists of 'n' number of processor cores starting from

0 to ‘n’ i.e. from processor core ‘118-1’ to processor core ‘118-n’. The processor cores contain separate first level cache memory ‘L1’ from ’124-1’ to ‘124-n’, respectively. It means the processor core ‘118-1’ has ‘124-1’ first level cache memory, ‘118-2’ processor core includes ‘124-2’ named first level cache, and so on till ‘118-n’ processor core which contains ‘124-n’ corresponding first level cache memory. The ‘L2’ i.e. second level cache memory named as ‘120’ is shared by all of these processor cores using the ‘122’ crossbar which acts as a medium of communication between them. The crossbar ‘122’ carries out individual communication between ‘118-1’ to ‘118-n’ processor cores and ‘122’ second level cache memory ‘L2’. One of the representation states that the ‘120’ crossbar is capable of handling huge amount of independent accesses whose processing is performed on each clock cycle.

To know the whole process of optimization of utilizing the multithreading multicore processor in depth, we have the following flow chart diagram.

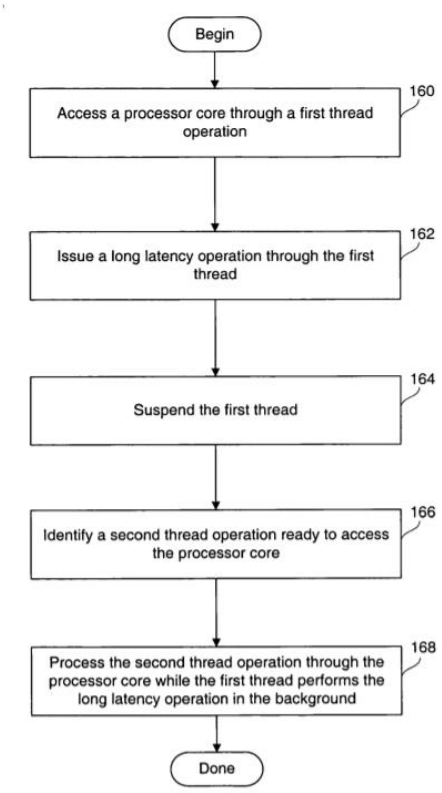


Figure 3. Flowchart Diagram of the operations for optimization of multithreading multicore processor utilization

As we begin with the method, first the operation ‘160’ is initiated where the first thread operation accesses the processor core. Then, after accessing the processor core, it moves to the ‘162’ operation where a long latency event is handled by the first thread operation. As we know, after the long latency operation, the first thread is released which is done at operation ‘164’. Also, under the long latency situation, branch, a cache miss and a floating-point operation is carried out. Further, the process reaches to the operation ‘166’ where it selects the second thread operation from the ready to run threads which is performed at the pipeline’s switch stage using a scheduling algorithm and then it goes under processing by the processor

core at the operation '168'. At this operation, while the second thread operation is getting processed, in the background the long latency operation is carried out on the first thread operation. During this operation, the second thread operation executes several instructions until the long latency operation of the first thread operation ends. This way, the pipeline consumption is maximized as multiple threads overlap at the time of execution.

In another study, we have used a simulation framework to analyze the design and architecture of multi-core multi-threading processors. The designed simulation framework incorporates a multicore simulation tool and a subdivision of standard programs to examine the amplification of multicore architecture which is caused due to the heavy trading of multiple threads in the multicore by implementing one of the standard programs proposed in [4]. Further, we discuss and give the description of the Multicore Simulation and Benchmark Suite of our simulation.

4.1 Multicore Simulation Description:

Benchmark that were put into practice to analyze the outcome of these processors are outdated because of their emphasizing accessibility of new architectures and designs in relation to cache, I/O bandwidth, memory and clock cycles according to [5]. The development and use of better and new standards should be practiced. The research in [6] has analyzed and compared two systems on chip processors that are SPLASH-2 and PARSEC. Accordingly, SPLASH-2 is the standard framework that centered workloads on the basis of real-time applications and is the progeny of the SPLASH benchmark. CPUSPEC2006, and Mini Bench are other benchmarks of this kind. SPLASH-2 is an aggregate of eleven benchmarks that are parallel in nature and the configuration files are provided in order to retrieve the data size of the input. Some testing of the Multi2Sim simulator were carried out on 32-bit systems built on LINUX operating system and the systems that use Dual Core processors of Intel. As a result, the Barnes program of the standard aggregate was used in this study for performance examination.

In other study, integration of characteristics of some of the most popular simulators resulted in the development of Multi2Sim. The characteristics involved in this are SMT and Multiprocessor support, separate functional and timing simulation and cache coherence. The aim of the Multi2Sim tool is to stimulate final MIPS32 [18] executable files and is an application-only tool. One can compile his owe program sources and would be able to test them by using the MIPS32 Cross-Compiler.

4.2 Benchmark Suite Description:

Multi2Sim simulator was used for the experiments, which was developed involving the important and noticeable characteristics of other standard simulators, like SMT (Simultaneous Multithreading), different timing and functional simulation, cache coherence and multiprocessor support. The only application capable of executing x86 binary executable files is Multi2Sim. As mentioned earlier, the simulation

was run by numerous multicore architectures. The simulation in each group is evaluated by altering the organization of number of threads, cache or cores.

4.3 Proposed Methodology for Simulation:

The simulation is tested on many thread groups like groups having 1 thread, 2 threads and 4 threads. We change the configurations like decreasing the number of threads and increasing the number of cores in every group, in order to test the simulation. For instance, in the group of 4 threads, first we test simulation with 1 thread and 32 cores, then a simulation with 2 threads and 16 cores and then at last we test the simulation with 4 threads and cores. Different configurations and architectures are used to test these cases.

Below are the tables that depict different configurations and architectures of cache memory and the machine on which the testing is performed for the cases mentioned above.

Item	Structure
Architecture 1 (A1)	Shared-L2 cache memory
Architecture 2 (A2)	Shared-L2-H cache memory
Architecture 3 (A3)	Shared-L3 cache memory
Architecture 4 (A4)	Shared-L3-H cache memory
Architecture 5 (A4)	Shared-L4 cache memory
Architecture 6 (A6)	Shared-L4-H cache memory

Table 1. Cache Memory Architectures

Item	Configuration
Configuration 1 (C1)	L1(32KB+32KB),L2 2MB,8-Way,64B
Configuration2 (C2)	L1(64KB+64KB),L2 2MB,8-Way,64B
Configuration3 (C3)	L1(32KB+32KB),L2 4MB,8-Way,64B
Configuration4 (C4)	L1(32KB+32KB),L2 4MB,16-Way,64B
Configuration5 (C5)	L1(64KB+64KB),L2 4MB,8-Way,64B
Configuration6 (C6)	L1(64KB+64KB),L2 4MB,16-Way,64B
Cache memory L3 : 8MB	
Cache memory L4 : 16 MB	

Table 2. Cache Memory Configurations

Item	Specification
Cores	Intel 2.13GHz(Dual core)
L1 cache	128 KB
L2 cache	512 KB
L3 cache	3.0 MB

Table 3. Information of Machine Configuration

Table 1. depicts the cache memory architectures, Table 2. depicts the cache memory configurations and Table 3. depicts the machine configuration that is used as the base for performing the simulation tests.

4.4 Program Loading:

The mapping of an executable file into a different virtual memory is known as Program Loading. To start execution, its register file and stack are initialized. The operating system oversees these actions in a real machine but during initialization, the Program Loading should be handled by 'application-only' tool.

Executable File Loading. ELF (Executable and Linkable Format) specification is followed by the executable files and outputs are filed by gcc. A header with a set of sections makes an ELF file. Some of the Linux versions include the library libbfd, this

library helps in providing the list of types and functions of an ELF file and helps in tracking their main attributes. The contents are copied into the memory when it is indicated that the flags of an ELF section are loadable.

Program Stack. Initialization of Process stack is the next step of program loading further ahead. To store the parameters and the function local variables is the aim of the program stack. While the execution is in process, the program code itself manages the stack pointer ($sp register). Program Stack expects some environment variables and program arguments to be in it while the program starts its execution.

Register File. Initialization of the register file is the last step. This includes the progressively updated $sp register and the Program Counter and Non-Program Counter registers. In MIPS32 architecture, NPC register is not defined explicitly, the simulator uses the NPC register internally which handles the branch delay slot.

4.5 Simulation Model:

A Functional Simulation Engine, An Event-Driven Module and A Detailed Simulator; these three are the three different simulation models of Multi2Sim. Event-Driven Module and Detailed Simulator oversee timing simulation. Further ahead, we will use the context to describe a software entity. This software entity would be defined by the status of a logical register file and virtual memory image. On the other hand, the processor hardware entity will be referred by the term thread, this hardware entity will be comprised of some entries in the pipeline queues, some physical memory pages, a physical register file etc. Further we will discuss the main techniques of simulation.

Functional Simulation. Functional Simulation is also known as simulator kernel. Interface to the rest of the simulator is provided by the Simulator Kernel. The engine owns different functions to create/destroy software contexts, enumerate existing contexts, execute machine instruction, consult their status, handle speculative execution and perform program loading as this engine is unaware of hardware threads. MIPS32 specification is followed by the supported machine instructions. Because of fixed instruction size and formats the above mention choice was encouraged, later which resulted in simple instruction decoding.

The checkpointing capability which is a significant characteristic of a simulation kernel was derived from the Simple-Scalar [9]. The register file and the memory status, both are saved by using this feature when a wrong execution path starts.

4.6 Multithreaded and Multicore Architectures Support:

This portion of the paper discusses about the feature of basic simulator which provides an environment that can handle multi-threading multicore processor simulation in an efficient way. The features can be categorized into two divisions where the first could affect the functioning of simulation

engine. (2) One which are involved in the detailed simulation of the module.

Functional Simulation: To support the parallel workloads, the functional machine has been extended. From this perspective, tasks which can create child processes dynamically at runtime can be seen as parallel workloads who carry out the operations of synchronization and communication. POSIX thread library (pthread) shared memory model [19] is one of the most used and it's a model which supports the parallel programming.

Using sets of sequential workloads is suggested in many studies for the multithreaded environment. The main purpose behind this is that the evaluation of the processor throughput could be done more accurately, and the hardware threads can share multiple resources. Contrasting this, the pipelines of multicore processor are fully duplicated and here the important point of contention is the Interconnection Network.

In the form of L2-l1 cache transfers, only certain interconnect activity is executed from multiple sequential workloads, but as they have disjoint memory maps coherence actions cannot take place between them. So, it makes more sense if the parallel workloads are executed using a memory location that is shared to evaluate multicore processors.

Special hardware support is required for actual parallel workloads, also the software support that is considered to be of low level would enable the threads-spawning, termination and synchronization. A brief discussion of thread management in POSIX and these issues is carried out below.

Instruction Set Support. When the concurrent thread execution is supported by the processor hardware, the existence of the critical sections of the architecture are affected directly by the parallel programming and in which the execution of more than one thread cannot be done simultaneously. It means the multi-threaded processors or the CMPs should create a stall when a hardware thread makes a way in the critical section.

Operating System Support. On completion of tracking the implementation of a parallel workload, system call creates the support for the pthread from the operating system. The formed system calls are: to wait for child threads; to spawn/destroy a thread; to communicate and synchronize with system pipes; to retain the released threads with the system signals.

POSIX Threads Parallelism Management. Without making any change, using Multi2Sim, application which are programmed with pthread can be simulated. In this library, the code by which the parallelism is handled by the subset of system calls and machine instructions. Yet the idea of combining the thread management code and the application code should not be ignored, as this could affect the final results.

Detailed Simulation: Multithreading Support

The organization of the stages in a multithreaded design can be specified by a set of parameters which is supported by the Multi2Sim. The stages can be shared among private per thread or the threads, but the execute stage couldn't be shared. An algorithm is needed to be set for every cycle on the stage and this should be scheduled by the algorithm. The model of the pipeline can be seen in the figure below. It is divided into 5 stages:

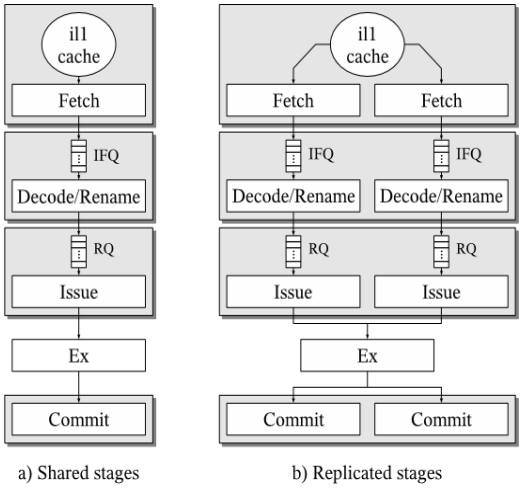


Figure 4. Pipeline Organization

Fetch Stage:

- Fetches instruction from the L1 cache and hand over them to IFQ.
- IFQ takes instructions from decode/rename stage, decodes them, registers are renamed and recruits a memory slot in the ROB and IQ.
- The IQ returns the instructions and then they are sent to the respective functional units.
- The results are written back into the register file during the execution stage.
- In the last stage i.e., commit, the ROB releases the instructions in the order of program.

However, instead of using the RUU (Register Update Unit) this model uses an IQ Register, a ROB and a Physical Register.

Figure 4. The Figure shows two examples of Pipeline Structures that are possible. In the figure a) The threads share the stages among each other and in (b) Except the execute stage, all the stages are replicated according to the hardware support number.

A multithread processor can be classified depending upon the thread selection and the stage sharing policies. Fine-Grain (FGMT), Coarse-Grain (CGMT) or Simultaneous Multithreaded (SMT); these are the classifications of a multithread processor.

Typically, on every processor cycle, switching of thread of FGMT processor follows a quick schedule on the other hand in CGMT processor, thread switch is persuaded by a thread quantum expiration or the long latency operation. Finally, SMT processor has the most aggressive instruction issue policy. It improves the previous ones and issues the instructions are issued from different threads in every single cycle.

	FGMT	CGMT	SMT
fetch_kind fetch_priority	timeslice -	switchonevent -	timeslice/multiple equal/icount
decode_kind	shared/ timeslice / duplicated	shared/ timeslice	shared/ timeslice/ duplicated
issue_kind	timeslice	shared/ timeslice	duplicated
retire_kind	timeslice	timeslice	timeslice/ duplicated

Table 4. Combination of Parameters

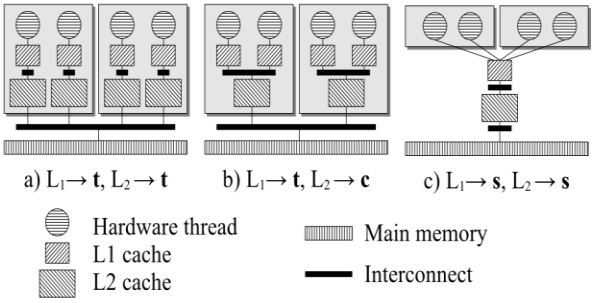


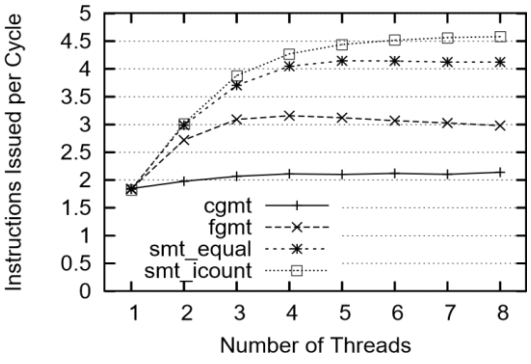
Figure 5. Evaluated Cache Distribution Design

Detailed Simulation: Multicore Support: By replicating the data structures which are used to represent the single core processor, a multicore simulation environment is basically achieved. In a multicore processor, the zone for the shared resources starts with hierarchy in the memory. When the caches are accessed simultaneously, some dispute might exist as the caches are shared among the cores. Contrary to this, at the time when they are private per core, to guarantee memory consistency a coherence protocol is implemented. A split-transaction bus is implemented in the Multi2Sim's current version as interconnection network.

Depending on the sharing strategy of instruction caches and data, the location and number of interconnects varies. From Figure 5 we can visualize 3 arrangements of sharing the L1 and L2 caches *(In the figure – t = private per thread; c = private per core; and s = shared)*

Results:

In this section, using Multi2Sim, we will see some simulation experiments. On one part we will illustrate the simulator application and on the other part we will check its correctness. The experiments which will take place will be (a) testing configuration of different multithread pipeline, (b) bus of different widths will be explored and (c) while executing parallel workload we will trace the net simulated traffic. In all the cases mentioned, the machine will include an independent L1 cache and data caches of size 64 KB, 1MB sized L2 cache that is shared, a physical register with 128 possible entries that is not shared and fetch, decode, issue and commit stages have capacity of 8 instructions per cycle.



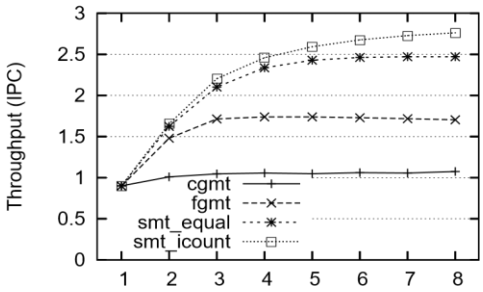


Figure 6. IPC and Issue Rate of Different Multithreaded Designs

Multithreaded Pipeline Organizations. The results of all the four different multithreaded implementations can be seen in figure 6: Multithreading's with equal thread priorities are FGMT, SMT and CGMT and SMT with ICOUNT (Threads with less instructions in pipeline is given priority here). Average number of instructions are shown in figure 6a while in 6b the global IPC is represented (i.e., it represents the sum of IPC's which are achieved by various threads).

Bus Width Evaluation. The impacts on the processor performance because of bus width is experimented over here, which resulted in having different number of contention cycles during the data transfers. For this experiment, Let's assume that the MOESI requests of 8 bytes and the cache blocks of 64 bytes. Hence the network message can have either 8 bytes or 72 bytes.

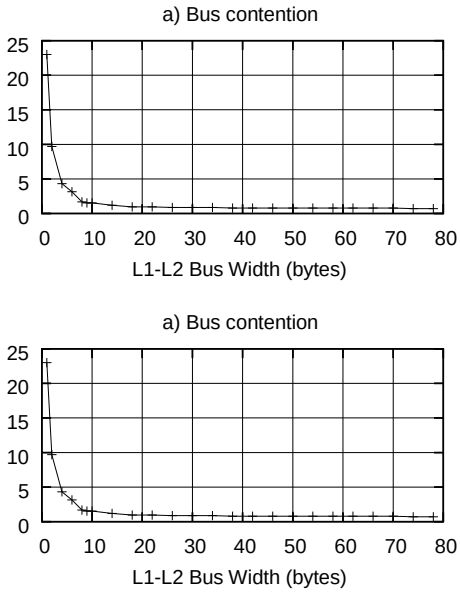


Figure 7. Different Bus-widths simulating fft for Performance

In Figure 7, the average of the contention cycle per transfer is represented. As no message which is larger than 72 bytes could be transferred, we will at least need a bus which is whose width is 72-Byte to send certain message in a single bus cycle which will lead in minimizing the contention. Yet some way, the conclusions specifies that a bus whose width is more than 3 times smaller, will provide you almost the same benefits.

Interconnect Traffic Evaluation. The activity of the interconnection network at the time of execution of the fft benchmark which has the same processor configuration as given above, is experimented here for a 16-Byte bus width.

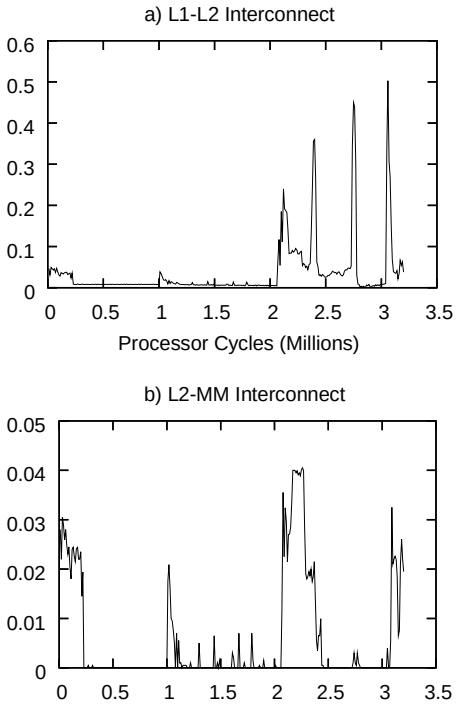


Figure 8. Distribution of traffic in L1-L2 and L2-MM Interconnects

Figure 8a: The fraction of used bandwidth to interconnect between L1 cache and L2 cache is represented, it takes intervals of 10^4 cycles.

Figure 8b: The fraction of used bandwidth to interconnect between the main memory (MM) and L2 cache is represented.

As per the figure we can say that the traffic distribution is quite irregular.

5. Future Work and Conclusion:

Succinctly, we have presented some proposed designs in which we have a processor chip consisting of multiple cores with at least two threads. The architecture of these multithreaded processor cores is designed in such a way that it achieves high throughput and also works effectively for the server applications that have heavy workload of large number of customers and clients. Simplified structure of pipeline which is known as single issue pipeline and thread level parallelism leads to the simplification of the architecture of the multithreaded processor cores. This architecture utilizes the server's multiple threads to the fullest.

We can apply this proposed design on other configurations of computer system like consumer related electronics that are programmable, applications based on microprocessor, mainframe computers, microprocessor devices, minicomputers, hand-held devices and many more. We can also execute this architecture on distributed systems that have computing environment, where the processing devices are remote and connected through a network to perform various tasks.

Our designed architecture requires usage of many computer-oriented operations that might involve using the data or information stored in the system. Physical manipulations are required in such operations to deal with the physical quantities. After manipulating these quantities, we can perform operations on the like data transfer, data storing, comparison, combination and other manipulations as the quantities are converted to magnetic or electric signals.

The manipulations listed above can be referred to as identifying, producing, comparing or determining.

We can say that these manipulations are functional machine related operations if they are participating in the designing of the architecture of our proposed system. It can also be useful in implementing a simulation machine or apparatus which incorporates these operations. The machines can be built with a special purpose or a general purpose depending on the requirements using which the apparatus is built. The general-purpose machines execute the computer programs that are coded by keeping the system architecture in mind and the special purpose machines perform the specialized requirements.

The designed architecture for a multi-threading multicore processor is open for modifications within the scope and it is not restrictive. The model and representations are not constrained within the description given in this paper and can be enhanced within the scope. The discussion and detailed description of the architecture is for understanding purpose only.

The simulation we used in of our study evaluates the performance of multi-threading in multicore processors for different architectures of cache using a simulator with keeping configuration parameters and L2, L3, and L4 architectures under focus. It is observed that to improve the overall performance i.e. Instruction Per Cycle number and Hit Ratio of a multicore processor is done by increasing the number of processor nodes, which is the result of the multiplication of the number of cores with the number of threads. It was also observed that as the number of nodes is increased, the performance also increases with the number of nodes till 32 nodes. Starting 64 nodes and higher, the performance will be decreased. The evaluation shows that the best performance is achieved by 16 cores and 2 threads for the multicore with 32 number of processor nodes. The study also helped in achieving the best architecture and configuration of cache for a mentioned number of threads and cores in order to achieve good performance.

According to the last analysis we performed, the technique to improve the existing simulators and how to extend them which will provide us with additional functionality is presented in this paper using the Multi2Sim framework. Apart from the features which can be adopted from the tools we can also refer to the timing first simulation (Simics-GEMS), basic pipeline architecture or the support to cache coherence protocols.

In the extensions for Multi2Sim we found sharing strategies for the simulation of pipeline stages, multicore multithread combinations, memory hierarchy configurations and also a interface which is integrated on the on chip interconnection network. For the state-of-the-art processors who cover the hot topics in the field of computer architecture, such features are necessary, and this makes Multi2Sim suitable for that kind of evaluation. Similar simulator characteristics are guided using examples in this paper.

6. References:

[1] Leslie D. Kohn, Kunle A. Olukotun, Michael K. Wong. Multi-core Multi-thread Processor. *United States Patent Documents*, 24 April 2007.

[2] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. *Proceedings of the 27th International Symposium on Computer Architecture*, 14 June 2000.

[3] Wael Mohamed Abd El-Samad, Maher Mansour Abd El-Aziz. Evaluation of Multithreading in Multicore Processors. *Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 8(2): pages 58-63, Scholarlink Research Institute Journals*, 2017. Retrieved from *jeteas.scholarlinkresearch.com*.

[4] Rafael Ubal, Julio Sahuquillo, Salvador Petit, Pedro Lopez. Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. *19th International Symposium on Computer Architecture and High Performance Computing, IEEE*, 24-27 October 2007. *SBAC-PAD*, 2007.

[5] S. Akhter, J. Roberts. Multi-core Programming. *Intel Press, Vol. 33*, 2006.

[6] C. Bienia, Sanjeev Kumar, K. Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of two multithreaded benchmark suites on Chip-multiprocessors. *2008 IEEE International Symposium on Workload Characterization*, 14-16 September 2008. *IISWC* 2008.

[7] D.C. Burger and T.M Austin. *The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342*,1997.

[8] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. *Univ. of Virginia Dept. of Computer Science Technical Report CS-2003-05,* 2003.

[9] D. Madon, E Sanchez, and S. Monnier. A Study if Simultaneous Multithreaded Processor Implementation. In *European Conference of Parallel Processing*, pages 716-726, 1999.

[10] J. Sharkey, M-Sim: A Flexible, Multithreaded Architectural Simulation Environment. *Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton*, 2005.

[11] D. M. Tullsen. Simulation and Modeling of a Simultaneous Multithreading Processor, *22nd Annual Computer Measurement Group Conference,* December 1996.

[12] M. Moudgill, P. Bose, and J. Moreno. Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration. *IEEE International Performance, Computing, and Communications Conference,* pages 451-457, 1999.

[13] M. Moudgill, J.Wellman, and J. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro,* pages 15-25, 1999.

[14] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, and M. Rosenfield P. Emma. Microarchitecture-Level-Performance Analysis: The PowerTimer Approach. *IBM J. Research and Development,* 47(5/6), 2003.

[15] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2), 2002.

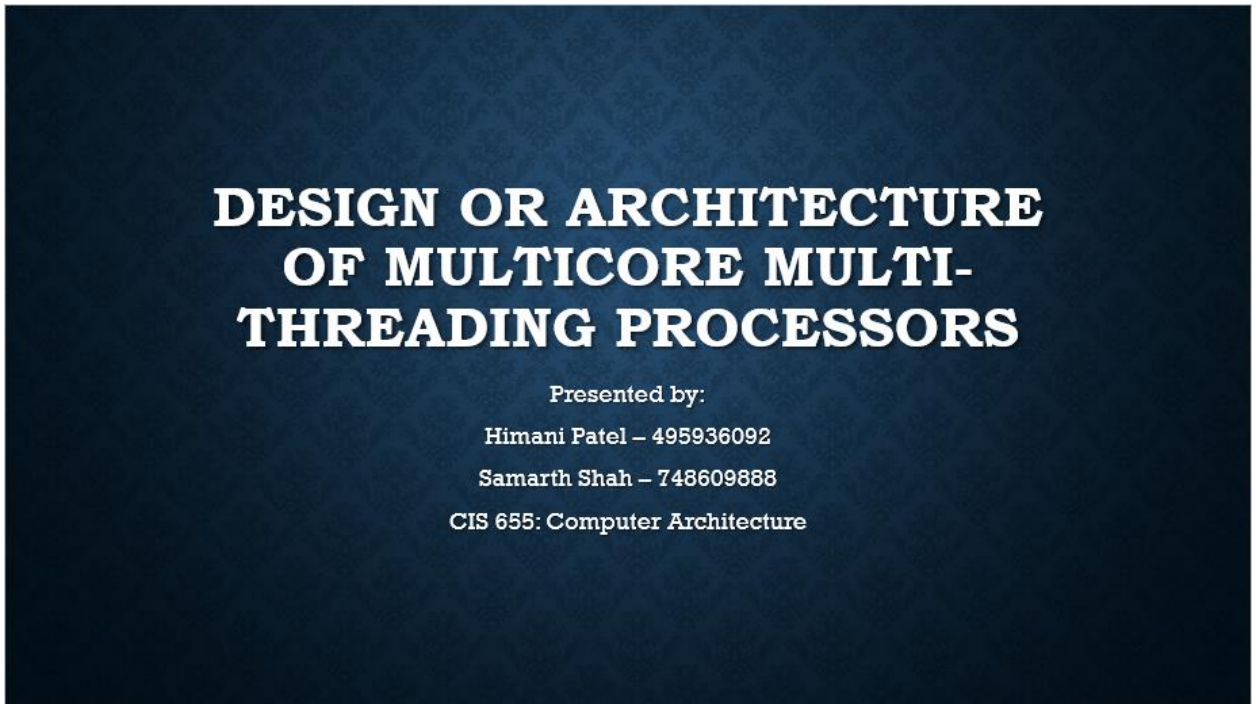
[16] M. R. Marty, B. Beckmann, L. yen, A.R. Alameldeen, M. Xu and K. Moore. GEMS: Multifacet's general Execution-driven Multiprocessor Simulator. *International Symposium on Computer Architecture,* 2006.

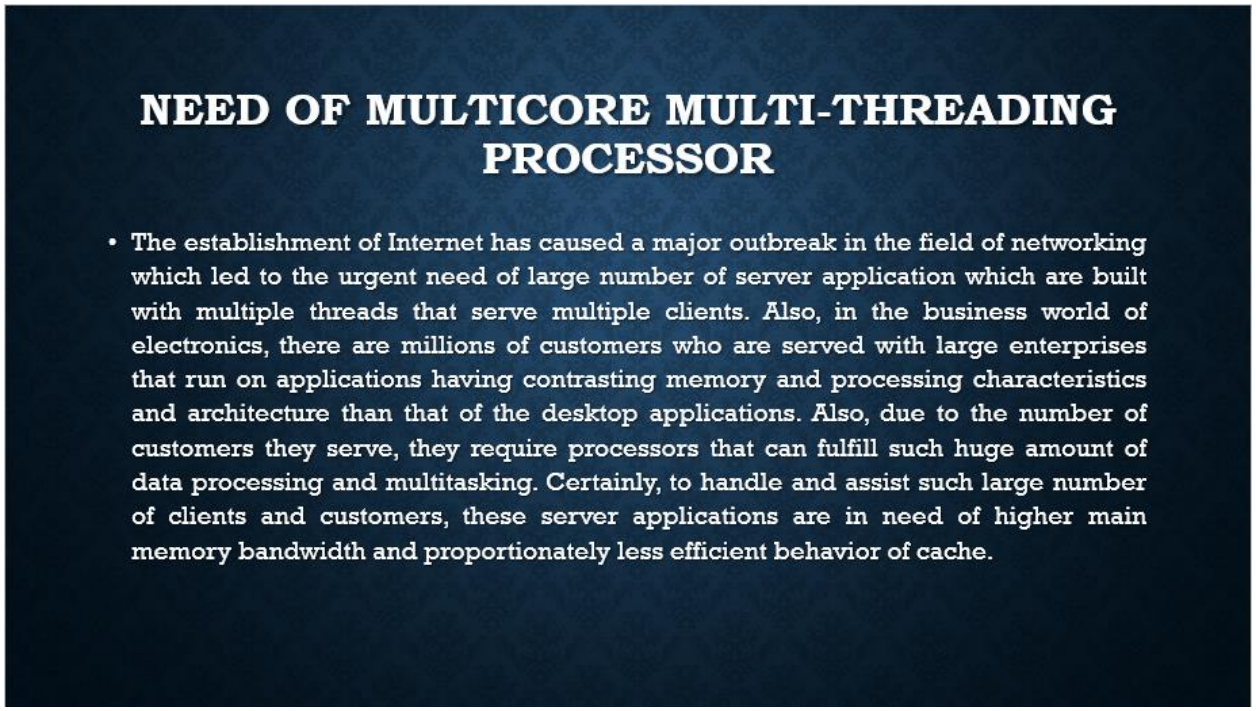
[17] N. L. Binkert, E. G. hallnor, and S. K> Reinhardt. Network-Oriented Full-System Simulation Using M5. *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW),* pages 36-43, Feb. 2003.

[18] MIPS Technologies, Inc. *MIPS™ Architecture For Programmers, volume I*: Introduction to the MIPS32™ Architecture. 2001.

[19] D. R. Butenhof. *Programming with POSIX® Threads.* Addition Wesley Professional, 1997.

7. Appendix:


DESIGN OR ARCHITECTURE OF MULTICORE MULTI-THREADING PROCESSORS
Presented by:
Himani Patel – 495936092
Samarth Shah – 748609888
CIS 655: Computer Architecture


NEED OF MULTICORE MULTI-THREADING PROCESSOR
• The establishment of Internet has caused a major outbreak in the field of networking which led to the urgent need of large number of server application which are built with multiple threads that serve multiple clients. Also, in the business world of electronics, there are millions of customers who are served with large enterprises that run on applications having contrasting memory and processing characteristics and architecture than that of the desktop applications. Also, due to the number of customers they serve, they require processors that can fulfill such huge amount of data processing and multitasking. Certainly, to handle and assist such large number of clients and customers, these server applications are in need of higher main memory bandwidth and proportionately less efficient behavior of cache.

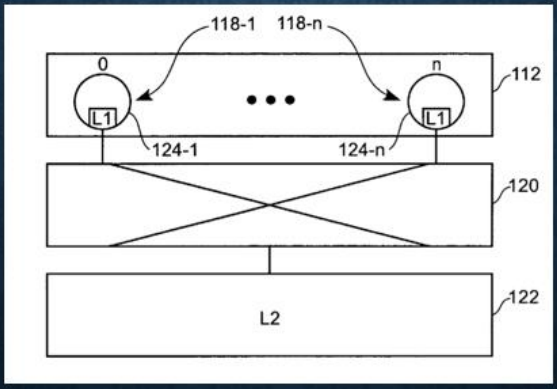
GIST OF OUR PROPOSED SYSTEM

- In this paper, we focus on demonstrating and evaluating the architecture or design of a multicore multi-threading processors in which multiple threads carry out processing of multiple instructions at the same time on separate cores which increases the overall speed of a processor to understand its functionality and determine the suitable architecture or design which can give the best performance-to-cost ratio. We observed that the need of such in-demand processor is fulfilled by a multicore multi-threading processor.

ANALYSIS

- Our embodiment provides a server that consists of a processor chip for applications. This processor chip for applications comprises of several cores having multithreaded central processing units. A first level cache memory is incorporated in each of the cores with multi-threading central processing units. In addition, a crossbar with several cache memories is provided in the processor chip for application which uses the crossbar for carrying out the communication process between the cores and the bank of cache memories. The main memory interface receives communication from each of the level two cache bank memories. These bank of cache memories are in communication with the application processor chip using a core with buffer switch included in the representation.

DESIGN OF THE ARCHITECTURE



BRIEF DESCRIPTION AND WORKING

- The processor chip named '112' consists of 'n' number of processor cores starting from 0 to 'n' i.e. from processor core '118-1' to processor core '118-n'. The processor cores contain separate first level cache memory 'L1' from '124-1' to '124-n', respectively. It means the processor core '118-1' has '124-1' first level cache memory, '118-2' processor core includes '124-2' named first level cache, and so on till '118-n' processor core which contains '124-n' corresponding first level cache memory. The 'L2' i.e. second level cache memory named as '120' is shared by all of these processor cores using the '122' crossbar which acts as a medium of communication between them. The crossbar '122' carries out individual communication between '118-1' to '118-n' processor cores and '122' second level cache memory 'L2'. One of the representation states that the '120' crossbar is capable of handling huge amount of independent accesses whose processing is performed on each clock cycle.

EVALUATIONS PERFORMED ON SIMULATION

- Bus Width Evaluation:

The impacts on the processor performance because of bus width is experimented over here, which resulted in having different number of contention cycles during the data transfers.

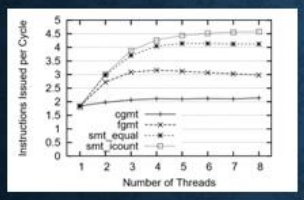
- Interconnect Traffic Evaluation:

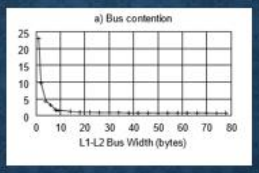
The activity of the interconnection network at the time of execution of the 'fft' benchmark which has the same processor configuration as that of the simulation is experimented here for a 16-Byte bus width.

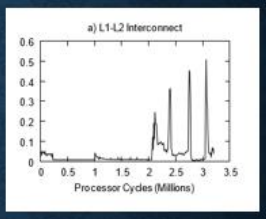
- Different Multi-threaded Implementations:

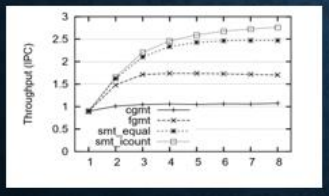
The experiments are performed on the simulation by changing the number of threads in the cores.

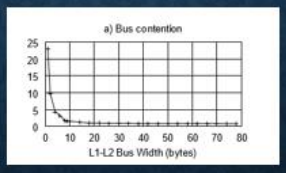
RESULTS OF TESTS ON SIMULATION

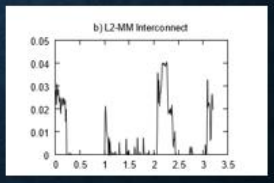












CONCLUSION

- In summary, we have presented some proposed designs in which we have a processor chip consisting of multiple cores with at least two threads. The architecture of these multithreaded processor cores is designed in such a way that it achieves high throughput and also works effectively for the server applications that have heavy workload of large number of customers and clients. Simplified structure of pipeline which is known as single issue pipeline and thread level parallelism leads to the simplification of the architecture of the multithreaded processor cores. This architecture utilizes the server's multiple threads to the fullest.