Development of a Simultaneously Threaded Multi-Core Processor

Eng. Soha S. Zaghloul [1], Dr. Muhamed Mudawar [2], Dr. M.G.Darwish [3]

[1] Coordinator for Cisco Programs, American University in Cairo

[2] Computer Engineering Department, King Fahd University of Petroleum & Minerals

[3] Professor & Vice Dean, Faculty of Computers & Information, Cairo University

Abstract

Simultaneous Multithreading (SMT) is becoming one of the major trends in the design of future generations of microarchitectures. Its key strength comes from its ability to exploit both thread-level and instruction-level parallelism; it uses hardware resources efficiently. Nevertheless, SMT has its limitations: contention between threads may cause conflicts; lack of scalability, additional pipeline stages, and inefficient handling of long latency operations. Alternatively, Chip Multiprocessors (CMP) are highly scalable and easy to program. On the other hand, they are expensive and suffer from cache coherence and memory consistency problems.

This paper proposes a microarchitecture that exploits parallelism at instruction, thread, and processor levels. It merges both concepts of SMT and CMP. Like CMP, multiple cores are used on a single chip. Hardware resources are replicated in each core except for the secondary-level cache which is shared amongst all cores. The processor applies the SMT technique within each core to make full use of available hardware resources. Moreover, the communication overhead is reduced due to the inter-independence between cores.

Results show that the proposed microarchitecture outperforms both SMT and CMP. In addition, resources are more evenly distributed amongst running threads.

Keywords: *Parallel Architecture, Simultaneous Multithreading, Chip Multiprocessors, Chip Multithreading*

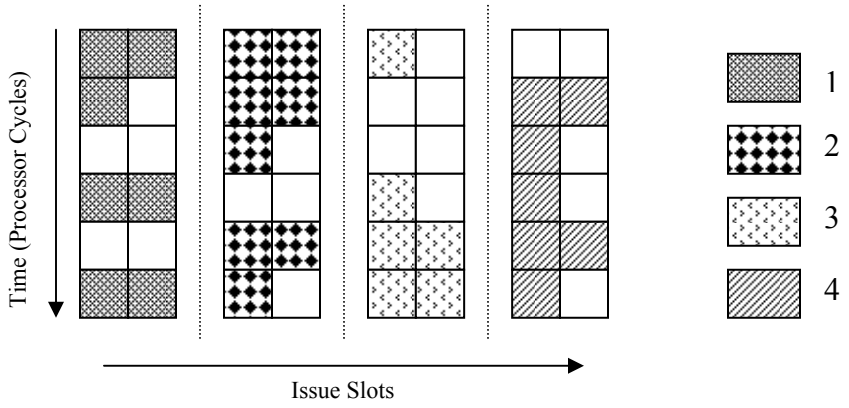
1. INTRODUCTION

In the last few years, two approaches evolved and proved success in the market: namely, *simultaneous multithreading (SMT)* and *Chip Multi-Processing (CMP)*. SMT was first introduced by [TUL'95]. The motivation of SMT is that the available number of functional units in modern superscalar processors is not fully utilized [HEN'03]. They are based on a superscalar architecture to which several physical execution contexts are added. Therefore, such processors can issue and execute a few instructions simultaneously. Unlike conventional multithreaded architectures [AGA'90] [ALV'90] [LAU'94] [SMI'81], which depend on fast context switching to share processor execution resources, all hardware contexts in an SMT processor are active simultaneously, competing each cycle for all available resources [TUL'96].

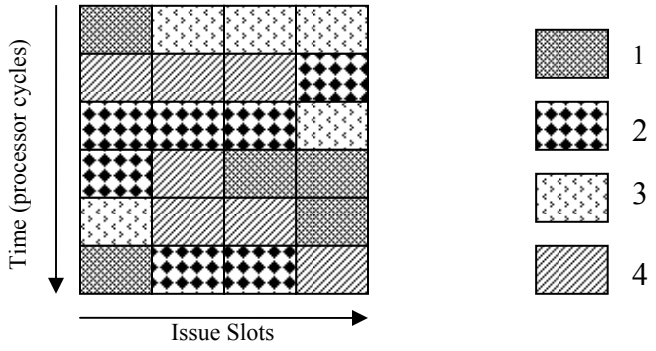
The diversity of hardware resources to be used by the issued instructions is the key of efficacy of SMT. SMT processors are considered cost-effective since resources are shared by all threads [KRI'98]. They are utilized more efficiently, which yields more instructions per cycle (IPC) and consequently better throughput [HEN'03]. In addition, hardware resources, such as the functional units, reservation stations, etc… are more efficiently used [REI'00]. Moreover, SMT can better tolerate pipeline and memory latencies, coping with deeper pipelines, branch mispredictions, and longer cache miss penalties [MUD'04]. On the other hand, SMT has its limitations: the high competition of threads on the resources may result in a conflict on the resources [TUL'01]. SMT is difficult to scale [BUR'02]. Large number of threads definitely needs more registers which entails either longer latency or increased number of stages in the pipeline. The latter solution is not suitable since it implies more complexity in recovering branch mispredictions and pipeline forwarding logic [RED'03]. A thread that regularly sweeps through the level-one data cache will evict data needed by the other thread [TUL'01].

Alternatively, CMP are highly scalable and easy to program [MAN'04]. On the other hand, they have some limitations: they are expensive due to the duplication of resources, the competition of threads on the cache negatively impacts the processor throughput and cache fairness [KIM'04]. CMP share cache at either primary or secondary level; in addition, private caches may be provided to each processor individually. This architecture initiates cache coherence and memory consistency problems.

Figure 1 compares the issue slot partitioning in the two previously mentioned architectures. Figure 1a demonstrates a CMP with four two-issue CPUs on a single chip. Therefore, each CPU can issue up to two instructions per cycle. In other words, each CPU has the same opportunity cost as in a two-issue superscalar model. Although the CMP is not able to hide latencies by issuing instructions of other threads in the same CPU, but the demonstrated CPU processor reaches a higher utilization than that of an equivalent eight-issue superscalar processor because of the smaller horizontal losses in the former. On the other hand, the SMT processor model in figure 1b exploits instruction-level parallelism (ILP) by selecting instructions from any thread (four in this case) that it can potentially issue. Instructions are issued whenever its corresponding operands are ready and the needed functional unit is available for use. If one thread has high instruction-level parallelism, it may fill all horizontal slots depending on the issue strategy of the SMT processor. If multiple threads each has a low ILP, instructions of several threads can be issued and executed simultaneously [ŠILC99].



(a) A four 2-issue CPUs CMP model



(b) Issue Slot Partitioning of 4 threads in an SMT Processor

Figure 1. Issue Slot Partitioning in (a) CMP and (b) SMT

The rest of the paper is organized in the following way: Section 2 discusses the proposed microarchitecture. Section 3 reveals the pipelines of the three architectures in concern. Section 4 exposes the results of the new microarchitecture. The conclusion is presented in Section 5. Finally, we give some ideas for the enhancement of this work in the future in section 6.

2. PROPOSED MICROARCHITECTURE

Like CMPs, the proposed microarchitecture consists of multiple cores. Within each core, one or more threads are running simultaneously with the concept of SMT. Figure 2 completes the comparison performed in figure 1 by demonstrating the issue slot partitioning of the proposed microarchitecture. The figure illustrates a 2-core processor using the concept of Chip Multi-Threading (CMT). Within each core, the instructions are issued in the same way as figure 1 (b).

Therefore, core-level parallelism is added to the instruction-level and thread-level parallelisms that characterize the SMT processors.

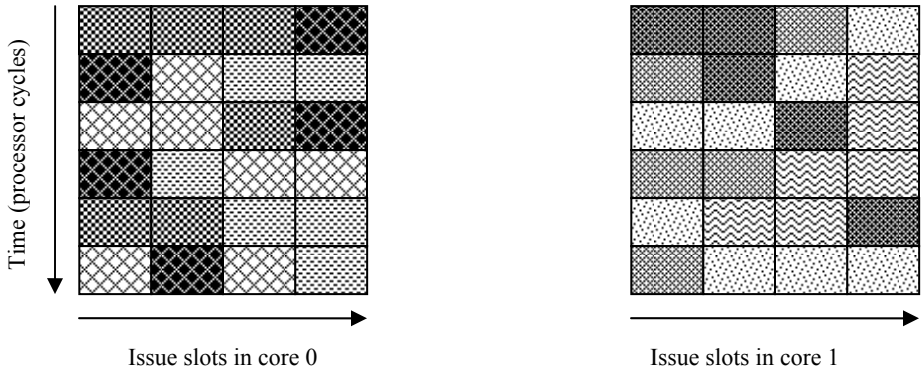


Figure 2. Issue Slot Partitioning in Chip Multi-Threading (CMT)

Unlike CMPs, the cores of the proposed processor are independent of each other. This eliminates the communication overhead and avoids the problems of cache coherence and memory consistency that CMPs are subject to. In addition, this design divides the register file avoiding the deep pipeline that SMT suffers from. The memory hierarchy consists of separate primary instruction and data caches local to each core. In addition, an on-chip secondary level cache is shared between all cores.

Threads are assigned to cores in a round-robin fashion. Therefore, one core may contain more than one thread, but one thread runs within a single core. Threads running within the same core may originate either from individual applications. Each core is provided with private resources that are not accessible by threads running in other cores.

3. PIPELINES

In this section, we compare between the pipelines of the superscalar, SMT and CMT processors. Figures 4 - 6 illustrate such comparison. The pipeline of the superscalar processor is identical to that of a single core in a CMP. As shown in figure 4, the pipeline consists of seven stages. Therefore, it takes seven cycles for an instruction to complete execution as soon as it is fetched from the fetch queue. After being fetched, the instruction is decoded and register renaming takes place in the next stage. The instruction is queued until the operands are ready and the corresponding functional unit is made available. As soon as these two conditions are satisfied, the instruction is issued and executed. The instruction is then committed if it is correctly predicted and it is not included in the path of another predicted instruction. As shown in the figure, we have six cycles penalty for each mispredicted instruction.

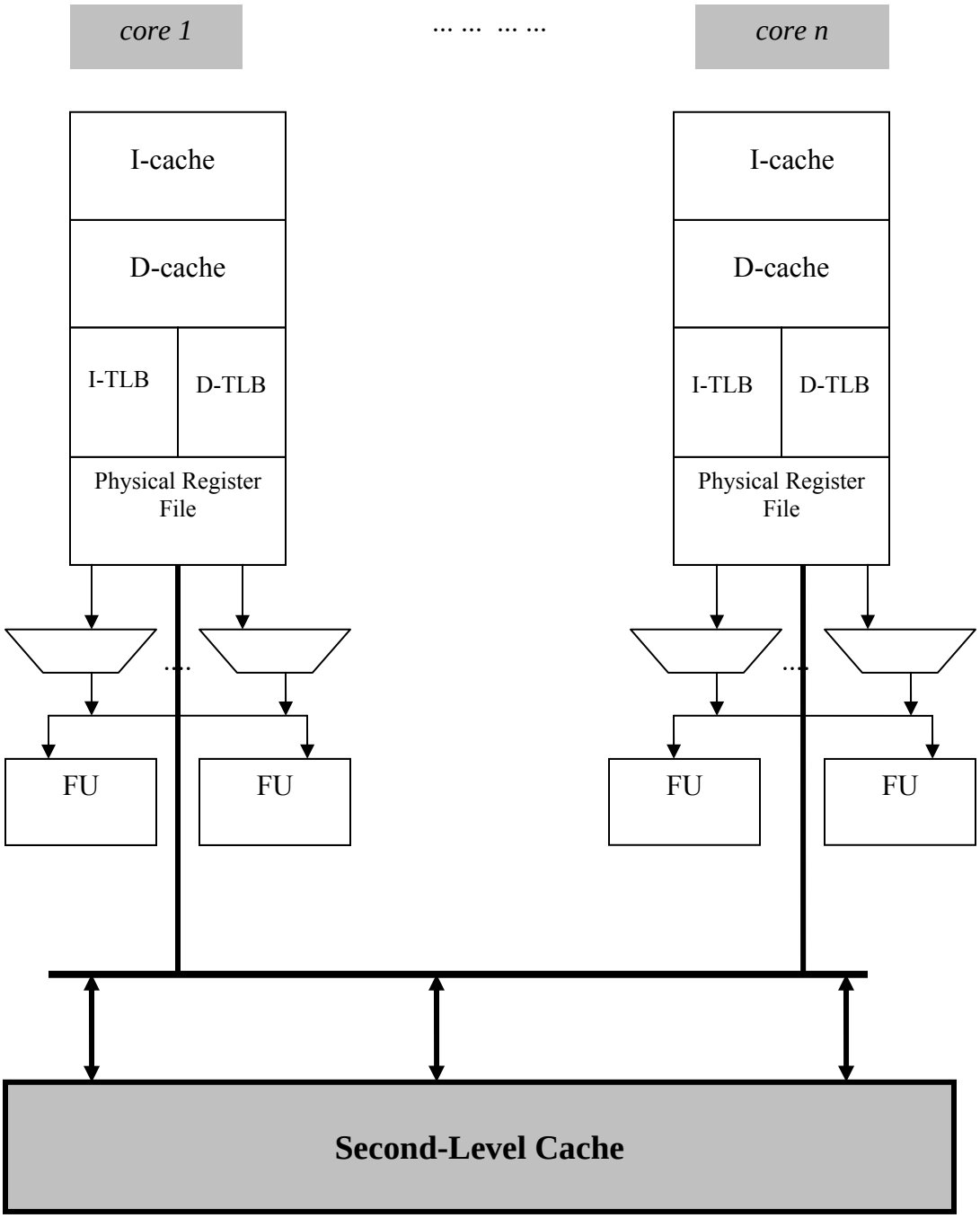


Figure 3. General View of the Proposed Microarchitecture

These resources include instruction and data caches, instruction and data TLBs, physical register file, reservation stations, and functional units. The functional units include an integer ALU, floating-point ALU, a multiplier and a load/store unit. All cores share a multi-ported secondary level cache. Figure 3 illustrates the above description.

The number of small cores should be a power of two up to a maximum of 8. Since all cores compete on the secondary cache, a higher number may negatively affect the results. In multithreading environment, competition between threads should be restricted in a certain way. Otherwise, the individual thread performance will degrade. It has been shown that the optimum number of threads in a SMT processor shouldn't be more than four. Therefore, the number of threads per core in the proposed processor is limited to four. Within each core, there is a single fetching unit, a number of program counters, next program counters, and predicted program counters equal to the number of running threads within the core.

In each core, instructions are fetched from a single thread following the *ICOUNT* algorithm which proved to give better performance according to [TUL'96]. In this technique, priority is given to the thread within the core that has the least instructions in the pipeline. Once an instruction is fetched, it is decoded and kept in the reservation station waiting to be issued. An instruction is issued when all its operands are ready and the relevant functional unit is available. The fetch queue is designed to have a width of sixteen instructions. The fetch rate is four: this means that at most four instructions may be fetched from the instruction cache in a single cycle. After being issued, instructions are then executed. Finally, they are committed and results are written back in the register file.

The previously described scheme has the advantage that it exploits all levels of parallelism: namely, instruction-level, thread-level and processor-level parallelism. Hardware resources are more efficiently used. Consequently, the processor is cost effective. On the other hand, the distribution of hardware resources amongst the processor elements and their inter-independence decreases the conflict that may occur between different threads on the hardware resources. Therefore, it is expected to have higher performance than SMT processors that suffers from such limitation. The inter-independence of cores and the fact that each of them has its own hardware resources makes it highly scalable and easy-to-program.

On the other hand, the proposed microarchitecture may not give satisfactory results in case threads are not evenly distributed among cores. In other words, two heavy threads may share the same processor element, whereas a lighter thread occupies a core all alone.

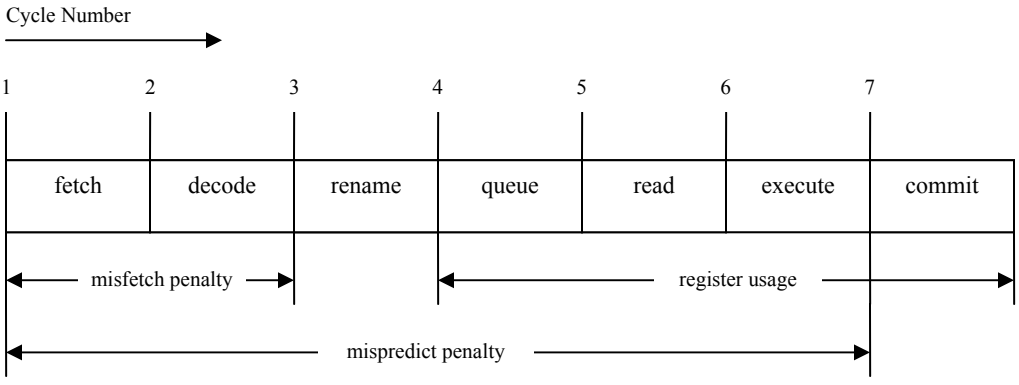


Figure 4. Superscalar Pipeline

Figure 5 demonstrates the SMT pipeline. Since the register file is large in SMT processors, an additional stage is assigned to the "read" phase. Also, a new phase is introduced into the pipeline which is the "write" stage. This makes the pipeline two stages deeper with the aim of giving the processor enough time to access the target register. However, hardware resources are utilized more efficiently than in the case of a CMP. The deep pipeline in SMT processors has a crucial drawback especially in misprediction cases: the penalty is augmented by two cycles. In addition, the pipeline forward logic becomes more complicated.

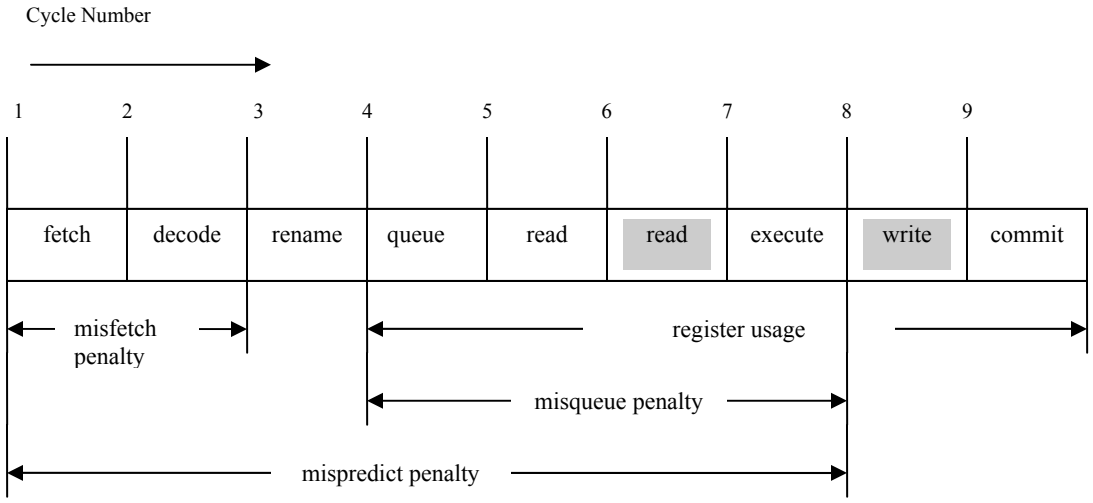


Figure 5. SMT Pipeline

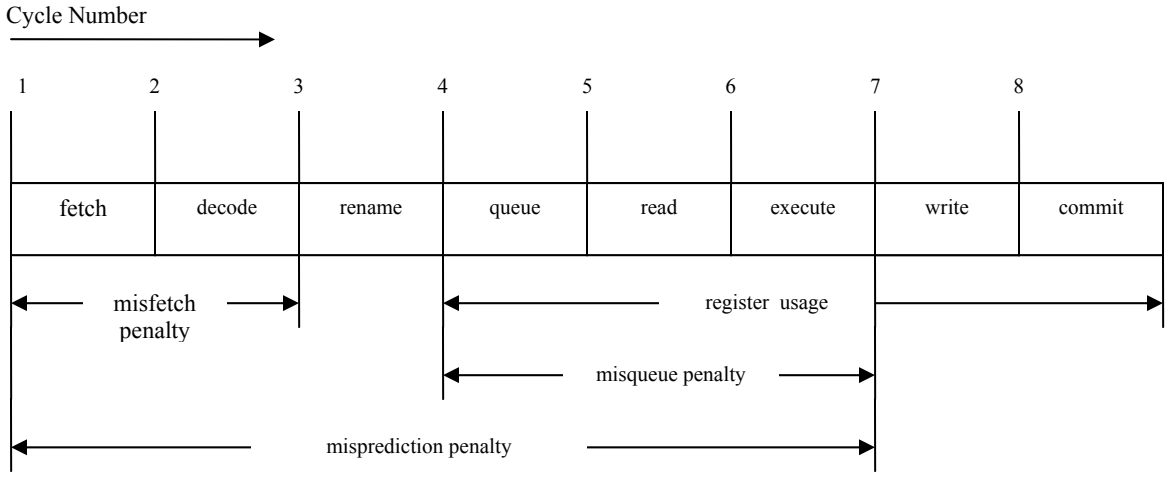


Figure 6. CMT Pipeline

On the other hand, the pipeline of the proposed microarchitecture is one stage less than that of the SMT processor. This is because the large register file of the SMT processor is divided amongst the cores. Therefore, the register usage, the misqueue penalty and the misprediction penalty are reduced by one cycle each as compared to the SMT pipeline. This boosts the processor performance while making use of the advantages of the SMT in fully utilizing the processor hardware resources.

4. RESULTS

The ss_smt simulator was used to evaluate the performance of the proposed microarchitecture. This simulator was developed as part of SEMPRE Project [GON'00]. It is written in C, runs under Linux, and was originally built on SimpleScalar [SS'04]. Major modifications were made to the ss_smt in order to simulate the proposed microarchitecture.

In the modified ss_smt, a *slot* designs a thread. A *module* or a *core* designates the replicated structure in the multi-core processor. A *bank* is a part of the memory address: the interleaving concept is applied. In other words, the memory system is divided into memory modules, each consisting of a number of banks. The advantages of this design can be summarized in higher bandwidth and to allow tolerance in case of the failure of a memory module.

Four benchmarks of SPEC2000 were used in the experiments. These are *mcf, gzip, vortex* and *ammp*. The first three benchmarks belong to SPECint; whereas the last one belongs to SPECfp. *mcf* is used to calculate the combinatorial optimization; *gzip* is used for file compression; *vortex* is an object-oriented database and *ammp* is used in computational chemistry. All benchmarks are coded in C language.

The main objectives of the following experiments may be classified into three categories. First, it is required to find out the architecture that gives the highest performance. Performance is measured in instructions per cycle (IPC). The second main objective is to specify the architecture that gives the least miss rates for different caches. Finally, the third objective is to find out the most efficient memory hierarchy. All experiments are undertaken with 1,600,000 instructions. The number of banks is equal to 4 whereas the block size is set to 64. Associativity is kept constant to 4. The *least replacement used* policy is applied for all experiments. Since the system is provided with a single port, then only one load/store instruction is allowed to be served per cycle. The cache size is calculated as the product of all the following parameters: number of sets, number of banks, block size and associativity.

Two extreme cases are to be highlighted in our experiments. If the number of modules is equal to one and we have multiple threads, then all threads are assigned to the single available module. They run with the concept of SMT. On the other hand, if the number of threads is equal to the number of modules, then a single thread runs in each module, and we have a case of CMP. Otherwise, we have a case of *chip multithreading.*

In the first experiment, the number of modules is varied, and we obtained the results shown in figure 7. Figure 7 shows the values of IPC for individual threads as well as the total IPC. The experiment is repeated three times for different number of modules (1, 2 and 4). This corresponds to SMT, CMT, and CMP.

The figure reveals an important fact: the highest performance corresponds to a number of modules equal two (CMT). The least performance takes place with the CMP architecture. This is explained by the fact that the processor may remain idle for a few cycles in case of issuing a long latency operation. No other instruction in the same thread may take place because of the interdependency between instructions. At the same time, there is no other thread that can make use of the processor. With the same number of modules running within a single core, the results reveal a slightly better performance. Although the processor is kept busy for a longer time than the CMP, but the competition of the threads on the available hardware degrades the value of the

IPC. For example, the fetch queue becomes full and no other instruction of any thread is fetched. In case if there is an instruction waiting for an operand to be issued; it might wait for a longer

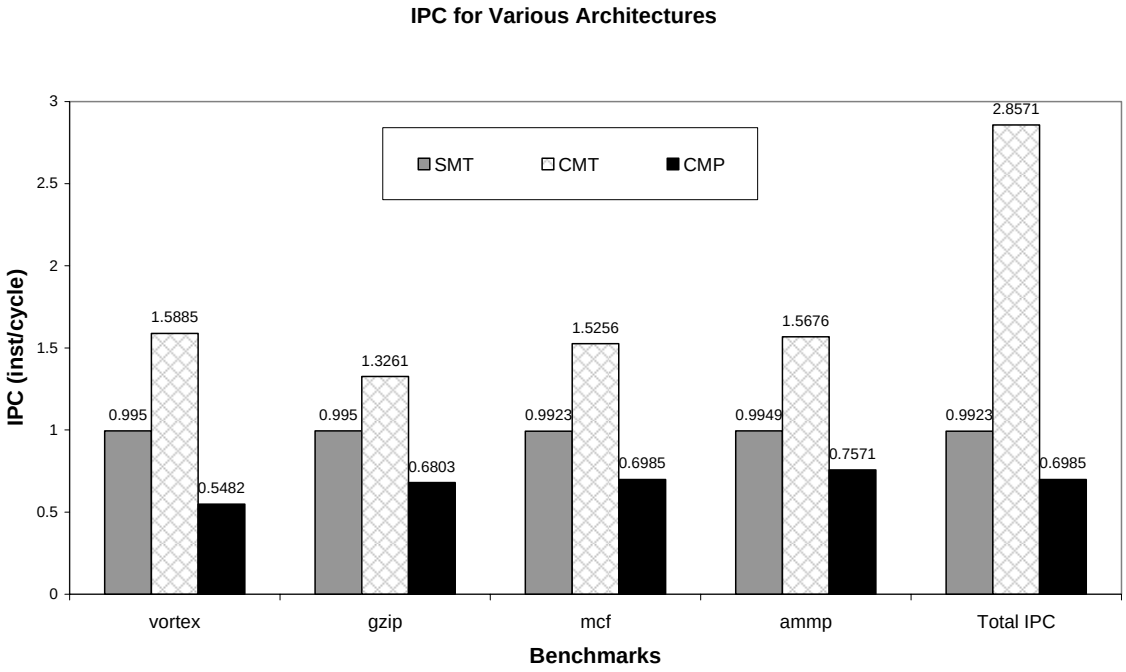


Figure 7. IPC for various architectures

time. Therefore, the relatively low performance is due to the bottleneck in the fetch queue rather than keeping the processor idle. On the other hand, running two threads per module (CMT) gives the best results. This is because the processor is properly used, and the competition on the available resources is relatively moderate. The total IPC takes the values of approximately 0.7 for CMP, 0.99 for SMT, and 2.86 for CMT.

In the next three experiments, we test the distribution of hardware resources exemplified in the first-level instruction cache, first-level data cache, and the second-level unified shared cache.

Figure 8 shows the values of the miss ratios in the first level instruction cache. The following configuration was given: separate instruction and data caches in the first level, and a unified shared second level cache.

The figure shows that the average miss ratio for the CMP architecture is almost zero. However, it is at its maximum (0.09) with the SMT architecture. The CMT gives a mean value (0.05). This proves the point of efficient hardware distribution. In case of CMP, the hardware is abundant. This raises the processor cost. On the other hand, there is a high competition on the cache in case of SMT: every thread evacuates the data in the cache that may be needed by another thread.

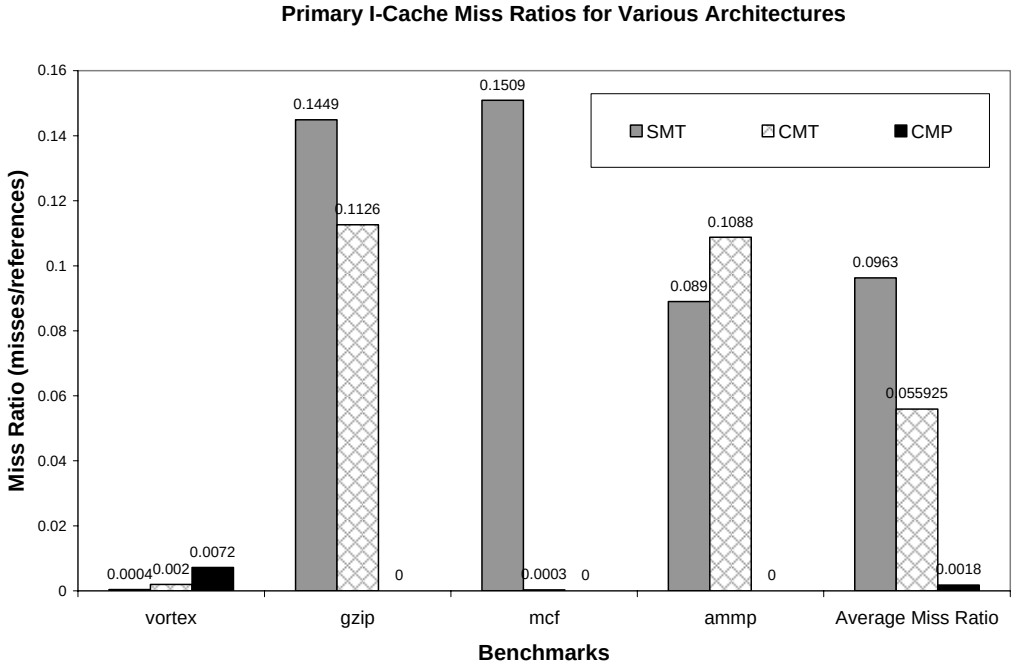


Figure 8. Primary Level Instruction Cache Miss Ratio for Various Architectures

Figure 9 illustrates the primary level data cache miss ratios for various architectures. Figure 10 shows that of the secondary level shared cache.

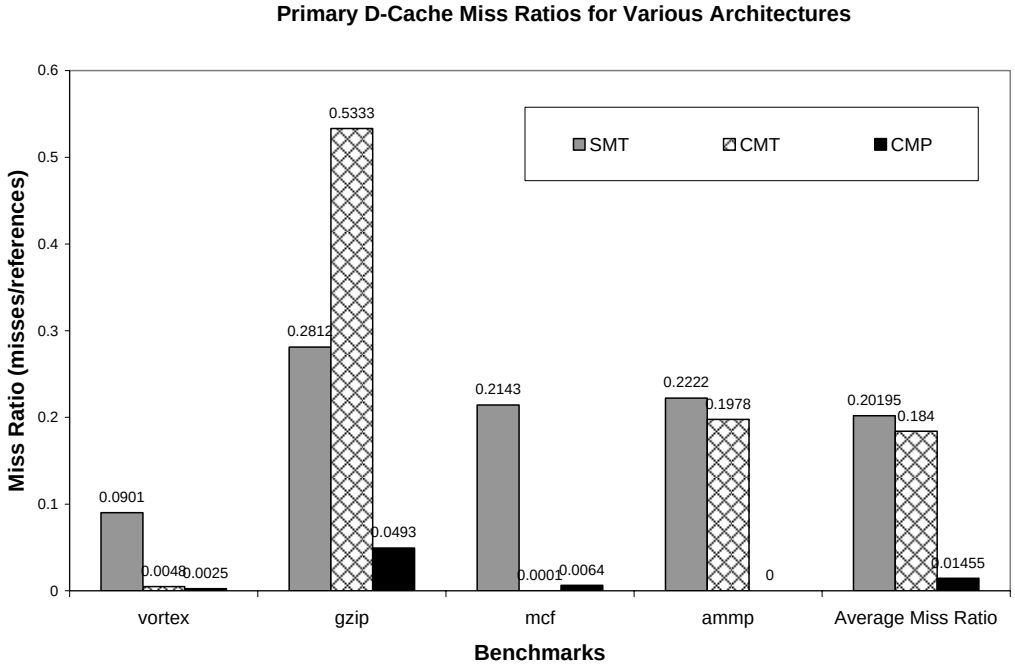


Figure 9. Primary Data cache Miss Ratios for Various Architectures

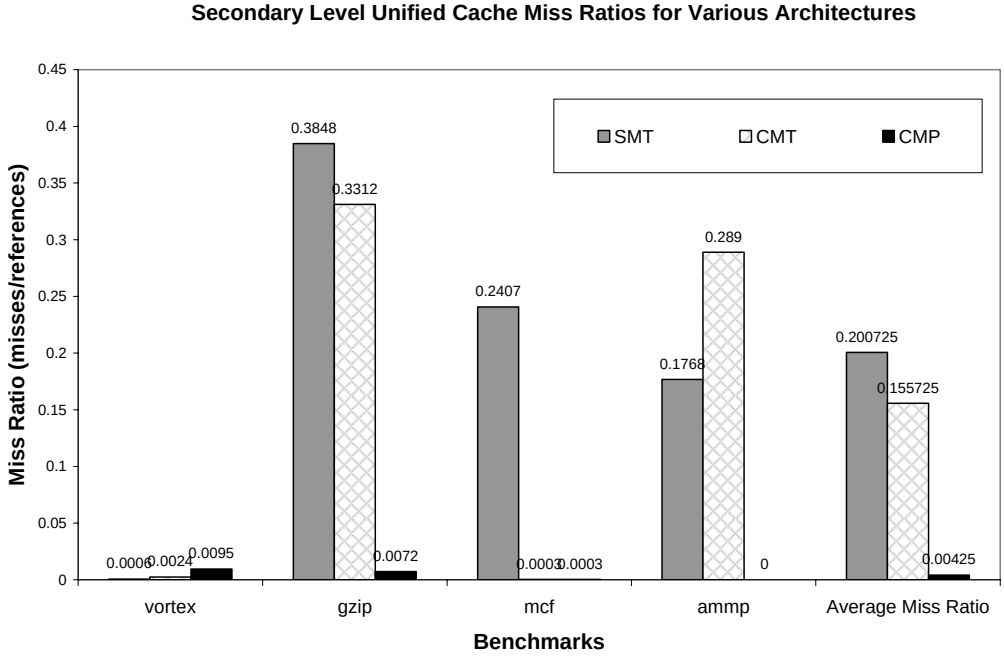


Figure 10. Secondary level unified cache miss ratios for various architectures

Referring to figure 9, it is found that the least average miss ratio is achieved with the CMP architecture (0.01); followed by the CMT architecture (0.18); then the SMT (0.2) is found to have the maximum average cache miss ratio for the primary level data cache.

Similarly, figure 10 shows that the unified secondary level data cache average miss rate is at its minimum in case of CMP architecture (approximately zero) due to the abundant use of hardware resources. The maximum value is obtained in case of the SMT architecture (0.2). The mean value is achieved with the CMT architecture (0.15). This proves that the most proper use of hardware resources is achieved with the CMT architecture.

Figure 11 shows the impact of the variation of the memory hierarchy on the processor performance. Four memory hierarchies were tested. These are:

a. Separate primary instruction and data local caches, with a shared secondary level unified cache.
b. Separate primary instruction and data local caches with no secondary level cache defined.
c. Separate primary instruction and data local caches, with two shared separate secondary level instruction and data caches.
d. No primary level caches are defined; and separate secondary level instruction and data caches are used.

The graph shows that the total IPC is almost the same for the first three configurations (3.12). However, it drops when no primary caches are defined (2.53). This proves the importance of having local primary level caches for each core. However, it does not make a big difference between using separate and data caches, providing a unified secondary level cache, or even suppressing the second-level cache completely.

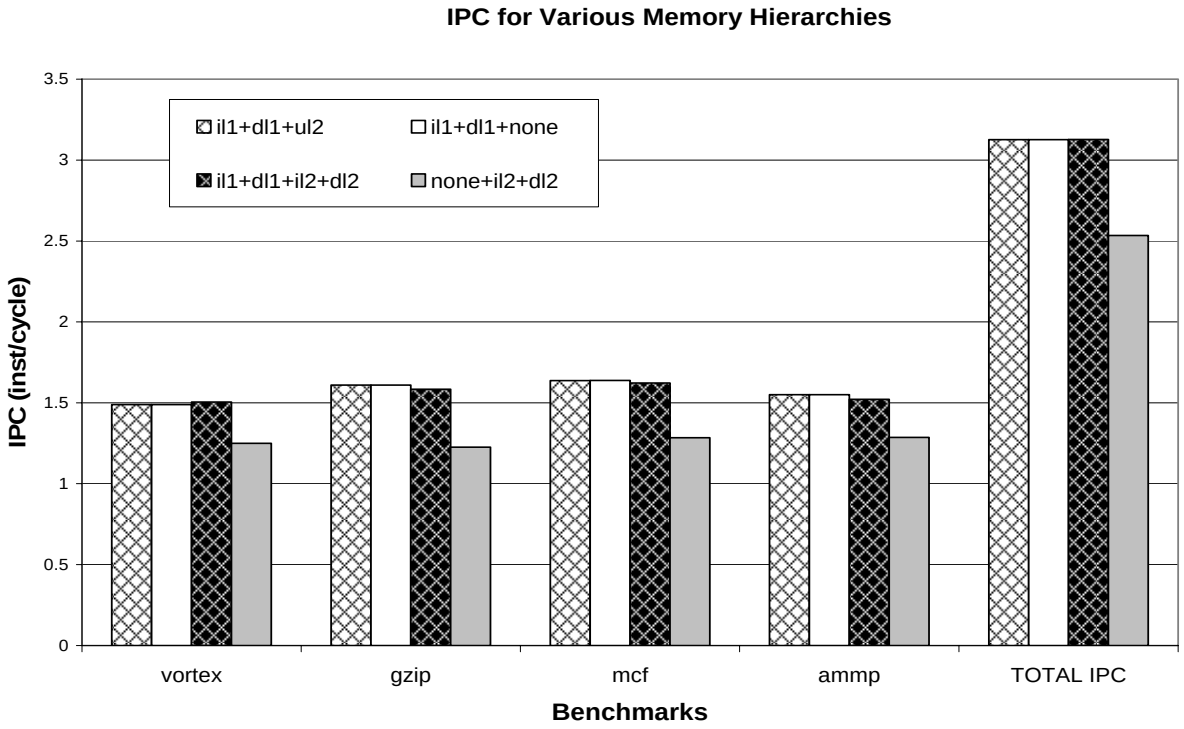


Figure 11. IPC for various memory hierarchies

5. CONCLUSION

The experiments discussed in this paper prove that the CMT architecture outperforms both the SMT and CMP architectures. However, the least miss rates are obtained with the CMP architectures since they use abundant hardware resources. However, CMP is the most expensive architecture since resources are not used properly. The CMT avoids the major problems encountered in both SMT and CMT. Unlike SMT, it is easy to scale; it avoids the high competition of threads on the available resources; it avoids using a huge number of registers since they are distributed on multiple cores. On the other hand, the CMT avoids the problem of cache coherence and memory consistency that the CMP suffers from. The communication overhead between cores is eliminated since they are totally independent. Moreover, the use of local caches in the primary level for each core has a positive impact on the processor performance.

6. FUTURE TRENDS

Although this work highlights the efficiency of the CMT architecture using independent threads, but the implementation of dependent threads that share data might also be investigated. Better results might be expected if threads were assigned on the cores differently. Other cache replacement policies may also be explored to know its impact on the results. It is also expected to have better results if the system was designed with multiple cache ports. Different fetching policies within each core may be also explored in the future.

7. REFERENCES

[AGA'90] Anant Agarwal, B.H.Lim, D. Kranz, and J. Kubiatowicz. "*APRIL: A Processor Architecture for Multiprocessing*". In the Proceedings of the 17th ISCA, May 1990, pp. 104 -114.

[ALV'90] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. *"The Tera Computer System*". In the Proceedings of the International Conference on Supercomputing, June 1990, pp. 1 – 6.

[BUR'02] J.Burns and J.-L.Gaudiot. "*SMT Layout Overhead and Scalability*". IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 2, February 2002, pp. 142 – 155.

[GON'00] R.Gonçalves, E.Ayguadé, M.Valero, and P.Navaux. "*A Simulator for SMT Architectures: Evaluating Instruction Cache Topologies*". SBAC-PAD 2000. Symposium on Computer Architecture and High Performance Computing, October 2000.

[HEN'03] John L. Hennessy and David A. Patterson. "*Computer Architecture: A Quantitative Approach*". 3rd edition, Morgan Kaufmann, 2003.

[KIM'04] S.Kim, D.Chandra and Y.Solihin. "*Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture*". In the Proceedings of the IEEE 13th International Conference on Parallel Architecture and Compilation Technique (PACT'04), September 2004.

[KRI'98] V.Krishnan and J.Torrellas. "*A Clustered Approach to Multithreaded Processors*". International Parallel Processing Symposium, March 1998.

[LAU'94] J.Laudon, A. Gupta, and M. Horowitz. "*Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations*". In the Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, October 1994, pp. 308 – 318.

[MAD'99] Dominik Madoń, Eduardo Sánchez and Stefan Monnier. "*A Study of a Simultaneous Multithreaded Processor Implementation*". In the Proceedings of the 5th International Euro-Par Conference on Parallel Processing, August 1999, pp. 716 – 726.

[MAN'04] N.Manjikian, H.Jin, J.Reed, and N.Cordeiro. "*Architecture and Implementation of Chip Multiprocessors: Custom Logic Components and Software for Rapid Prototyping*". In the Proceedings of he 33rd International Conference on Parallel Processing (ICPP'04), August 2004, pp. 483 – 492.

[MUD'04] M.F.Mudawar. "*Scalable Cache Memory Design for Large-Scale SMT Architectures*". In the Proceedings of the 3rd Workshop on Memory Performance Issues held in conjunction with the 31st IEEE/ACM ISCA, June 2004.

[RED'03] J.Redstone, S.Eggers and H.Levy. "*Mini-Threads: Increasing TLP on Small-Scale SMT Processors*". In the Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA-9 03), February 2003, p. 19.

[REI'00] S.K.Reinhardt and S. Mukherjee. "*Transient Fault Detection via Simultaneous Multithreading*". ACM SIGARCH Computer Architecture News, Vol. 28, No. 2, May 2000, pp. 25 – 36.

[Šil'99] Jurij Šilc, Borut Robič and Theo Ungerer. "Processor Architecture: From Dataflow to Superscalar and Beyond". Springer-Verlag, Berlin, 1999.

[SMI'81] B.J.Smith. "*Architecture and Applications of the HEP Multiprocessor Computer System*". In the Proceedings of the 4th Symposium on Real Time Signal Processing IV, 1981, pp. 241 – 248.

[SS'04] http://www.simplescalar.com

[TUL'01] D.Tullsen and J.Brown. "*Handling Long-Latency Loads in a Simultaneous Multithreading Processor*". In the Proceedings of the 34th Annual International Symposium on Microarchitecture MICRO'01, December 2001, pp. 318 – 327.

[TUL'95] Dean M. Tullsen, Susan J. Eggers, Henry M. Levy. "*Simultaneous Multithreading: Maximizing On-Chip Parallelism*". In the Proceedings of the 22nd ISCA, June 1995, pp. 392 – 403.

[TUL'96] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo and Rebecca L. Stamm. "*Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*". In the Proceedings of the 23rd ISCA, June 1996, pp. 191 – 202.