ARCHITECTURE AND IMPLEMENTATION ISSUES OF MULTI-CORE PROCESSORS AND CACHING – A SURVEY

Bhaskar Das[1], Ashim Kumar Mahato[2], Ajoy Kumar Khan[3]

[1]*Department of Information Technology, Assam University, Silchar*
[2]*Department of Information Technology, Assam University, Silchar*
[3]*Department of Information Technology, Assam University, Silchar*

Abstract

As the performance gap between processors and main memory continues to widen, increasingly aggressive implementations of cache memories are needed to bridge the gap. This paper includes what brought about the change from single processor architecture to having multiple processors on a single die and some of the hurdles involved, and the technologies behind it. Having each processor on a single die allows much greater communication speeds between the processors. For multi-threading and multitasking, security and virtualization and physical restraints such as heat generation and die size, we need multi-core processor. Processor cache is the performance bottleneck in most current architectures. Next, we consider some of the issues involved in the implementation of highly optimized cache memories and survey the techniques that can be used to help achieve the increasingly stringent design targets and constraints of multi-processors.

Keywords: *Cache, Multi-core, Multi-tasking, Multi-Threading, virtualization.*

----------***----------

1. INTRODUCTION

Traditional processor architectures have included the transistor count into the hundreds of millions. This transistor, nano-scale electronic switch, can switch between 1 and 0 states billions of times in a second. So, power is very much needed. One way to counteract the power consumed is to reduce the size of the transistor. However, the transistor can only shrink so much before the functionality of the electronic switch breaks down and allows current to pass improperly [1]. These power consumptions lead to heat production, another side-effect of high transistor counts. These issues point toward a shift in architectures: greater parallelism.

Multiple applications run on a single core processor, so the operating system acts as scheduler-switching contexts between the applications. This can require a complete dump of all processor registers and possibly the cache(s), which is costly in terms of completion time. For example, if there are two processors working in parallel then no need to switch contexts between the two applications in a running computer. The main advantage from multiple cores, the programmer must divide the application into simultaneous threads or be done by the operating system for multitasking. A thread is a lightweight sub-program that shares the same memory space as other threads under the same program process. This notion of multi-threading is challenging relatively new and isn't yet taught to be as fundamental as, say, data structures.

Cache memory is small faster memory which is used to store data temporarily portion of main memory that data are frequently used. The main concepts of using cache memory to improve processor performance have been easy to understood and readable. Today, caches have become an important part of every processor. Our main aim is to reduce the performance gap between processor and main memory that is why we insert a cache memory between processor and main memory.

The ability of caches to bridge the performance gap is depends on two primary factors - the time needed to retrieve data from the cache and the fraction of memory references that can be satisfied by the cache. These two factors are commonly referred to as access (hit) time and hit ratio respectively [16], [17]. The access time is most important for first level caches because a longer access time means a slower processor clock rate, more pipeline stages. In order to minimize access time, cache access should be triggered as soon as the address of the memory reference is available. The hit ratio is also critical, both because misses impose delays, and because off-chip bandwidth, especially when there is a shared bus, is a very limited resource.

Next, in section 2 we will discuss basics of multi-core architecture and cache technologies. In section 3

We will show how we can critically analyze the performance of multi-core processor. Next, we will tell about the architecture of multi-core processors. In section 5 we state about some implementation issues of modern caches. Section 6 says how we use the multi-core processor.

2. BASICS

2.1 Computer Architectures

Past architectures have multiple physically separate processors. Those architectures become gradually backdated to the multiple on-chip processors due to mainly wire delay and caching techniques. Wire delay is the time it takes for data to traverse the physical wires. This can have a drastic effect on frequencies. There is also the added problem of limited intra-processor communication pins for multiple separate processors - a problem not seen in multi-core processors.

2.2 Cache

Cache memory is small faster memory which is used to store data temporarily portion of main memory that data are frequently used. The main concepts of using cache memory to improve processor performance have been easy to understood and readable. Today, caches have become an important part of every processor. Our main aim is to reduce the performance gap between processor and main memory that is why we insert a cache memory between processor and main memory.

Data cache was designed using two key concepts in mind –
1. Temporal locality [3] –i) when elements are require again in the near future.
ii) Arrange the code such way that element in cache is reused often.
2. Special locality [3] – i) when other element in the block will be needed soon.
ii) Cache line is fetched together.
iii) In the same cache line works on consecutive data elements.

When the processor find data item in the cache if it is available in the cache then it is called a cache hit and if the data item is not available in the cache then it is called cache miss. If we want to reduce the miss rate then we should concentrate on both the latency and bandwidth of the memory.

The detail of cache operation leads to different cache design choices: Cache mapping techniques, Cache replacement policy and Cache write techniques

Cache mapping techniques [5]: Three basic cache mapping techniques –
In case of direct mapping each block in main memory has particularly one and only one location in cache it can be copied to. See figure 1 for an example. This technique is less costly as no searching is required. However, if thrashing occurs, when one cache block is continually swapped between two or more memory blocks, the overhead becomes an issue.

Fully-associative mapping each memory blocks can be stored anywhere in the cache. In this technique, the entire cache must be searched for each memory access. This requires more hardware and is thus very costly.

A combination of the two mapping concept, direct and fully-associative, forms the most common mapping technique: set-associative. Here, the cache block is divided into separate sets. Every set is made up of two or more blocks. A two block set-associative mapping is referred to as 2-way, because the data retrieved from main memory can be put in two different locations, instead of just one. This technique is flexible and limits the amount of thrashing that could occur.

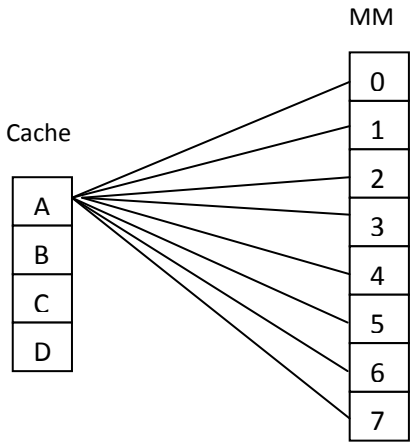

Fig.1. Direct Main Memory to Cache mapping

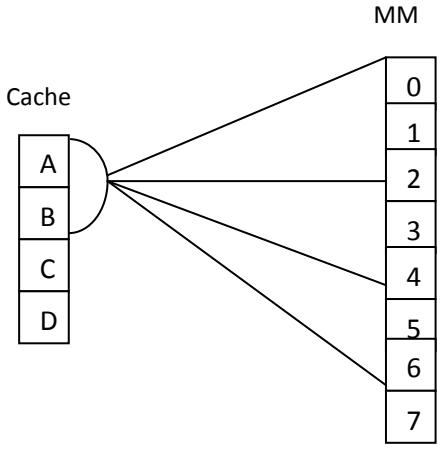

Fig.2. Associative Main Memory to Cache mapping

Note:
i) Direct Mapping ≡ 1way set associative.
ii) Fully Associative Mapping ≡ n way set associative.

Block Identification:
-Block address is used to identify the contents of a cache block uniquely.
- Block size are changes according to different mapping technique.

Consider fig 4.

Block Address		Block offset
Tag	Index	

Fig.4. Addressing Cache Block

Offset-Within block least significant bits of an address index words.

Block address divided into two parts one is the Tag another is the Index.
-Tag is used to identify each cache block uniquely.
-Index is used to particular slot set (set associative) or slot (direct).

In case fully associative mapping technique index size-

$$\text{Index} \quad 2 = \frac{\text{Cache Size}}{\text{Block Size} \times \text{Set Associativity}}$$

Writing to cache from the CPU presents another opportunity for optimization. There are two simple write policies: write-through and write-back [4][5]. During a typical write, the CPU stores its computed data to a location in cache, which is stored back into main memory. Write-through stores the data into the cache and into the main memory at the same time. Write-back stores the data in the cache, and only writes to main memory when evicted.

1. ANALYSIS PROCEDURE

We must need mathematical equations to verify the performance. This is very important in cache design.
Miss rates are a common metric of cache implementations where miss rate is the ratio of misses to memory accesses. This analysis is calculated by involving the times associated with miss penalties and hit times. From [4], the average memory access time (AMAT) in seconds or clock cycles can be found by the following equation.

AMAT = Hit time + (Miss rate * Miss penalty)

Where hit time is the time it takes to get a memory location and miss penalty is the time involved when the requested memory is not found in the cache. Miss penalties take larger time than hit. Via emulation and simulation software is the most common way to test configurations before a complete physical implementation. In [2], hardware prototyping and testing is analyzed using a Xilinx Virtex-II Pro FPGA. Using an FPGA as a test bed gives great reconfigurability. As stated in [6], random program generators and simulation methods are used to test the basic structures when combined. Upon implementation, benchmarking software, such as SPEC CPU2006 [7], is used to test the many aspects of a processor.

4. ARCHITECTURE

In 2006 Intel and AMD started multi-core processor to the user and server markets. The AMD Athlon 64 FX dual-core processor has two L1 caches, data and instruction, and one L2 cache, unified, for each core [8] (see Figure 5). Intel uses a shared L2 cache in what is referred to as the Advanced Smart Cache" [8] (see Figure 6).This implementation technique we used to reducing the cache misses and increasing the performance.

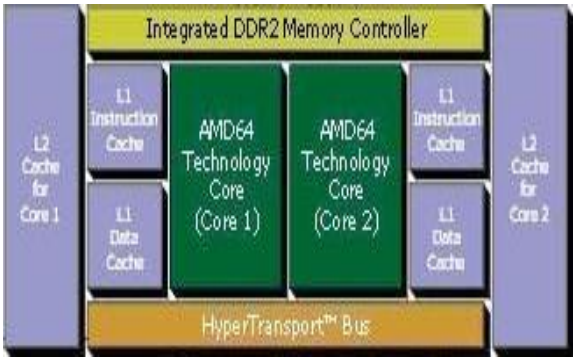

Fig.5: AMD Athlon 64 FX Architecture.

Another concept [9] proposes is non-uniform cache architecture to share cache between cores dynamically. When one core uses cache space unnecessarily and intrudes on another core's space then cache pollution occurs in the architecture address. The proposal is done with a quad-core processor and three levels of cache. The third level, L3, is partly shared and partly private. Each core is allotted a certain amount of space in L3 to be private and cannot be intruded upon.

In [10], the idea of specializing is the cores for virtual machines. They proposed two main designs: a single virtual machine core shared by all other general-purpose and specialized cores (for system virtualization); or each general purpose core can have a virtual-machine-specific core (for process virtualization).

System security and dependability is addressed in [11] with an integrated framework for dependable and revivable architectures", or INDRA. INDRA uses a core set at a higher privilege that is protected from remote attacks, a resurrection, and monitors the execution of the other cores, the resurrectees.

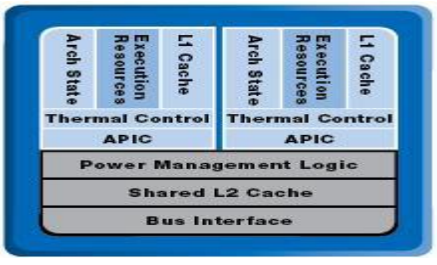

Fig.6: Intel Core Duo Architecture.

5. IMPLEMENTATION ISSUES

5.1 Addressing Constraints

In order to minimize effective memory access time, the access should be triggered as soon as the effective address of the memory reference becomes available. In most computers there must be a delay in translation. This delay cannot be removed completely.

5.1.1 Physical Address Cache

The caches are organized as 2-dimensional arrays and are accessed in a two phase cycle. In the first phase, a cache set is selected by using a portion of the address known as the index bits. In the second phase, the remaining part of the address is used to make a further selection from within this cache set to yield either a cache miss determination or the requested data. There are various techniques that exploit this two phase access cycle to enable a physically addressed cache to be accessed without requiring an extra address translation cycle. However, there is a practical limit to this approach because increasing the set-Associativity provides only a diminishing return on cache hit ratio but adds hard-ware complexity and adversely impacts the access time [12]. A technique that can be used to increase the number of address bits available before address translation is to restrict the virtual to physical page mapping so that the low-order bits of the physical and virtual page numbers are identical [14]. Another way to make more address bits available before address translation is to predict the additional address bits. An example of a good predictor is the content of the base register that is used to compute the effective address [13].

5.1.2 Virtual Address Cache

Instead of using bits from the virtual address as a predictor for the physical address, a different approach is to use the virtual address to directly access the cache [15], [16]. This avoids the delay for translation. In addition, all the addresses must be tagged with an address space identifier or else the cache must be purged on every task switch [16]. The most serious drawback of the virtual address cache is that multiple virtual addresses may be mapped to the same physical address, i.e. synonyms may occur [16]. The usual approach to handling synonyms is to prevent them from being present in the cache at the same time. In general, a reverse translation buffer (RTB) is needed in order for this approach to be feasible. One way to reduce the complexity in handling synonyms is to make sure that the index bits used to select the cache set are the same for both the physical and virtual addresses.

5.2 Access Time and Miss Ratio Targets

The performance of a cache is determined both by the fraction of memory requests it can satisfy (hit/miss ratio) and the speed at which it can satisfy them (access time). There have been numerous studies on cache hit/miss ratios with respect to the cache and line sizes, and the set Associativity [16], [17].

5.2.1 Decoupled Caches

The data array access and line selection are carried out independently of the tag array access and comparison so as to circumvent the delay imbalance between the paths through the tag and data arrays. This is trivially true in the direct-mapped case because in such a cache, there is only one cache line in each cache set. However, it tends to have an inferior hit ratio due to conflict misses.

5.2.2 Multiple-Access Caches

A direct-mapped cache is accessed sequentially more than once in order to achieve the access time of a direct-mapped cache for the fast access and the hit ratio of a set-associative cache as a whole. In [20], a simple rehashing function based on flipping the highest-order index bit is used. Upon a hit to a secondary location, the lines located in the primary and secondary locations are swapped.

5.2.3 Multi-level Caches

A small and fast upstream cache is used for the fast access while one or larger and slower downstream caches are used to capture the fast- access misses with minimal penalties. The organization and performance of multi-level caches have been studied extensively [18].

5.3 Area and Bandwidth Constraints

In order to bridge the growing performance gap between processor and memory, more and more silicon area is being dedicated to the on-chip caches. For example, the Intel Pentium Pro consists of a pair of 8KB on-die instruction and data L1 caches and an on-module 512KB L2 cache. Together these caches occupy 65% of the total die area and account for 88% of the total number of transistors. There are several approaches to increasing cache bandwidth. A straightforward way is to have separate instruction and data caches so that the instruction and data references can be handled simultaneously. The trace cache [19] alleviates this problem by storing the logically contiguous instructions in a physically contiguous block in a separate cache.

6. USES OF MULTI-CORE PROCESSOR

6.1 Servers

Servers have very good use of multi-core processor. A server can potentially have many simultaneous connections to many users. To accept these connections, the server will either spawn a new process or fork off a new thread. It makes the main process to wait for a connection. The operating system can then allocate these workloads across the available cores.

6.2 Consumers

The consumer market has adopted these new processors, banking on the multi-tasking parallelism granted by the multiple cores. These applications reap direct benefit from a multi-core

architecture by either multi-threaded programs or via scheduling by the operating system. Multi-core processors are not limited to traditional computers. Two such examples are the Cell processor and NVIDIA Tesla GPU.

6.3 Virtualization

The idea of virtualization tracks back to the days of mainframes. Now the costs are far lower. However, one thing remains to be true: under-utilization. A system administrator can configure the computer to virtualises" its devices, or operating system, to allow one or more simultaneous virtual machine(s) to use the computer as if each virtual machine (VM) was its own computer.

7. CONCLUSIONS

Multi-core processors are already expanding their niche and are finding many new and creative uses. Due to physical limitations and increased multitasking requirements, the multi-core architecture is expected to become the standard over the single-core predecessors. Further caching schemes, both specialized and general, will continue to be honed, narrowing the performance gap between the processor and main memory. Again During the past decade the performance of processors has improved by almost 60% each year. Current trends in the industry suggest that in the future, it may become economically feasible to integrate a processor on the same die as the DRAM. Such integration has the potential to reduce system cost and improve both DRAM latency and available bandwidth. For general purpose computing, cache memories will continue to play a crucial role in bridging the processor-DRAM performance gap.

REFERENCES

[1] D. Geer, “Industry Trends: Chip Makers Turn to Multi-core Processors," Computer.org, IEEE, pp. 11-13, May 2005.

[2] C. R. Clark, R. Nathuji, H. S. Lee, “Using an FPGA as a Prototyping Platform for Multi-core Processor Applications", Georgia Institute of Technology, Atlanta, GA.

[3] V. P. Heuring and H. F. Jordan, “Computer Systems Design and Architecture”, Prentice Hall, 2nd Edition, 2003.

[4] J. L. Hennessy, D. A. Patterson, “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann Publishers, 4th Edition, 2007.

[5] L. Null, J. Lobur, “Computer Organization and Architecture”, Jones and Bartlett Publishers, 2003.

[6] D. Lewin, D. Lorenz, S. Ur, “A Methodology for Processor Implementation Verification", Technion, Haifa, Israel.

[7] J. L. Henning, SPEC CPU Subcommittee, “SPEC CPU2006 Benchmark Descriptions", Standard Performance Evaluation Corporation, 2006.

[8] Jeremy W. Langston and Xubin He, “Multi-core Processors and Caching - A Survey”, Tennessee Technological University, 2007.

[9] H. Dybdahl, P. Stenstrom, “An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors", HiPEAC Network of Excellence.

[10] D. Upton, K. Hazelwood, \Heterogeneous Chip Multiprocessor Design for Virtual Machines", University of Virginia.

[11] W. Shi, H. S. Lee, L. Falk, M. Ghosh, “An Integrated Framework for Dependable and Revivable Architectures Using Multicore Processors", Georgia Institute of Technology, Atlanta, GA, 2006.

[12] M. Hill, A. Smith, “Evaluating Associativity in CPU Caches," IEEE Trans. Computers, Vol. 22(12), Dec. 1989.

[13] K. Hua, et al., “Early Resolution of Address Translation in Cache Design," Int'l Conf. Comp. Designs, pp. 408-412, Oct. 1990.

[14] K. Inoue, H. Nonogaki, T. Urakawa, K. Shimizu, “Plural virtual address space processing system," US Patent No. 4145738, March 20, 1979.

[15] F. Reiley, J. Richcreek, “Parallel Addressing of A Storage Hierarchy in A Data Processing System Using Virtual Address," US Patent No. 3693165, Sep. 19, 1972.

[16] A. Smith, “Cache Memories," Computing Surveys, Vol. 14(4), Sep.1982, pp. 473-530.

[17] A Smith, “Cache Memory Design: An Evolving Art," IEEE Spectrum, Dec. 1987, pp. 40-44.

[18] F. Sparacio, “Data Processing System with Second Level Cache," IBM Tech. Disc., 21(6), Nov. 1978, pp. 2468-2469..

[19] E. Rotenberg, S. Bennett, J. Smith, “Trace Cache: A Low-Latency Approach to High-Bandwidth Instruction Fetching," MICRO'29, Dec. 1996, pp. 24-34.

[20] A. Agarwal, J. Hennessy, M. Horowitz, “Cache Performance of Operating Systems and Multiprogramming," ACM Trans. Computer Systems, Vol. 6(4), Nov. 1988, pp. 393-431.