Research and Implement of Multi-core Processor

Huida Tao

George Washington University

huidatao@gwu.edu

Abstract

Because of the rapid increase of the hardware performance, the use of the single-core processor, even it becomes very complex, has reached the limit, which include the instruction level parallelism and the power limitation, the demand of a multi-core processor becomes more and more intense. By 2000s, the Intel and AMD developed the first generation of the multi-core processor, which highly improved the capacity of the computer, and also made changes the way of software development. This paper will demonstrate the reason of the present of the multi-core processor. The structure of the multi-core processor, and the improvement it brings to the computer industry is analyzed. Also, the pro and con of the multi-core processor will be introduced in this paper. At the end of the essay, the latest techniques of the multi-core processor from Intel and AMD will be presented.

1. Introduction

A multi-core processor technique is a technique that can compose two or more independent processing unit into a single computing component. The multi-core processing unit, as we seen, has the same functions with the single-core processor, which can read and execute program instructions. The different is, not only can multi-core processor handling CPU instructions such as add, branch, move data, but also it can run those instructions at the same time, thanks to the multi-core. By enabling the computer to do the parallel computing, it can increase the overall speed of the program.

1.1 Why Multi-core processor?

The reason that the multi-core processor is developed is complex. The main reason is that the use of the even more complex single core processor has reached its limit, due to the hardware performance issues, in power limitation and limits in instruction level parallelism. Besides this, there still several other motivations of the raise of the multi-core processor.

Commercial incentives

In old days, computer engineers can improve the performance of the CPU by shrinking the area of the integrated circuit, which also can drove down the cost. The clock rate of the CPU increased by order of magnitude. But as the rate of the clock rate improvement slows down, increasing the use of parallel computing performance became the hottest issue of all the companies. Using multiple cores on the same CPU chip can lead to better sales. This kind of competition pushes the multi-core processor forward.

Software incentives

With the present of the multi-core processor, many software companies started to make their software adaptable to the multi-core architecture. By doing this, not only can the software run in these new generation computer, but also can increase the performance of the software itself, which can attract more users. The hardware companies push the software companies to make contribution on multi-core architecture, vice versa. This relationship make the multi-core architecture so popular right now.

1.2 Development of the Multi-core processor

The original computer processors are single core processors, with the development of the manufacturing techniques, more and more powerful single core processors show up. But with the physical limits of the semiconductor-based microelectronics, a bottleneck comes up. The present of the multi-core processor breaks this bottleneck, improving the performance to a new level. In the mid-1980s, the Rockwell International introduced versions of the 6502 with two 6502 cores on one chip as the R65C00, R65C21, and R65C29, sharing the chip's pins on alternate clock phases [1, 2]. Nowadays, the bellwethers of multi-core processor industries are Intel and AMR. The multi-core processors from these two companies will be introduced later in this paper.

1.3 Advantages and disadvantages

There is more and more software coding in thread level, the software can use the multiple cores to handle parallel thread, which can highly improve the speed of the software. Software like Office, IE are all using thread level coding, which can make the most use of the multiple core. Even for those single thread programs, they can also take advantages of the multiple cores because the operating system can support parallel execution, which means he will send instructions from different programs to different cores to increase the speed of the programs. Besides this, multi-core chips can also save a lot of energy. The energy consuming maybe is not a big problem for PC but it means a lot in mobile device area since they only got limited battery. Using multi-core can make every core more energy-efficient, allows mobile devices more powerful without being a battery killer.

Multi-core processor is not perfect, it also generate lots of challenges to both operating system and application software. To make the most use of the multiple cores, the software should adapt multiple threads, which is a really big challenge for programmers. The operating system is also need to be update. In this paper, there will be example about the software company adapting multi-core processor.

1.4 Paper Overview

In the section 2, the hardware issue of the multi-core processor is discussed; the section 3 follows the introduction of the architecture of the multi-core processor. The section 4 focuses on the performance issue of the multi-core processor, what we should consider about when we want to improve the performance of the multi-core processor and also some techniques and mechanism that we could implement to accomplish this. In the last chapter, the two most famous multi-core processors example, the Intel Core i7 and AMR11 MPCore will be introduced and give us a better view about the multi-core processor.

2. Hardware Performance Issue

The performance of the microprocessor has increased for over decade, this is due partly to the refinement of the architecture of the CPU, and partly to the increase of the clock frequency.

2.1 Increase in parallelism

There are many changes happened on the architecture of the processor, include:

- Pipeline

  Instructions are executed through a pipeline of stages. It means when one instruction is executing on one stage of the pipeline, another instruction is executing on a different stage of pipeline, which can increase the parallel level.

- Superscalar

  Construct multiple pipelines by copy the resource of the execution, instructions can execute in different pipeline simultaneously if there is no confliction.

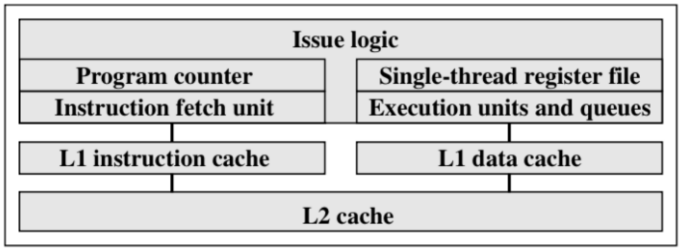

Figure 1. Superscalar

- Simultaneous Multithreading

  Replicating the register banks to let the multiple thread share the resource of the pipeline.

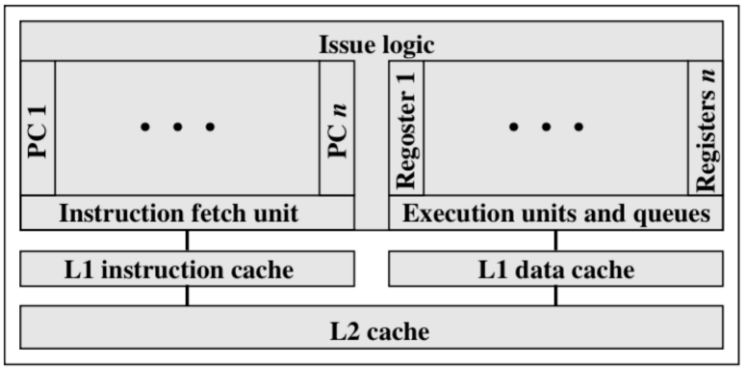

Figure 2. SMT

2.3 Trend in nowadays

With the pipeline, superscalar and SMT, we do can increase the processor's parallelism level, and many people trying so hard to improve the performance of the processor by using five or even more pipelines and many different ways. But there still a bottleneck of these techniques. Using pipeline as an example, applying more stage on pipeline can increase the level of the parallelism, but it can also bring us more problems. More pipelines require more logic, more interconnections and more control signals. Also if you want to improve the superscalar, you need to add more pipelines, which can also bring the performance of the processor down. From the figure 3 we can notice that the performance of the processor reached its limit after 2000.

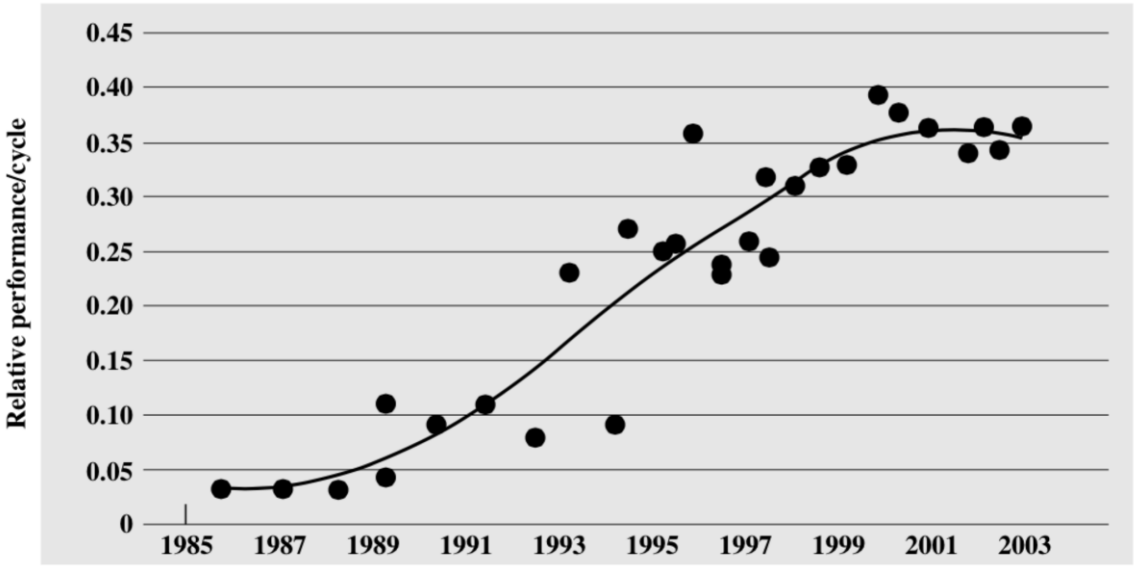

Figure 3

The general trend in processor development has moved from those techniques to multiple cores on one chip, which is the multi-core processor. In addition, they also have simultaneous multithreading, memory-on-chip and so on which can further increase the performance of the processor. In figure 4 shows the chip organization of the multi-core architecture, the detail of the Multi-core Organization will be introduced in the next chapter.

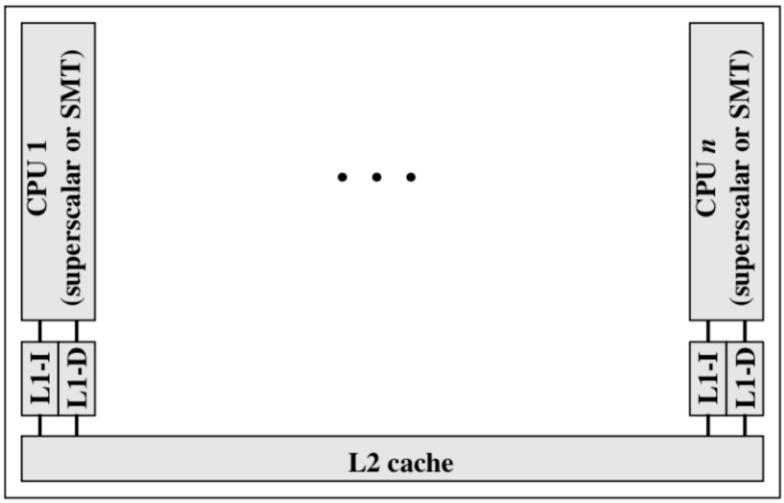

Figure 4. Multi-core Chip Organization

3. Multi-core Architecture

3.1 Multi-core Organization Alternatives

There are three main variables in the Multi-core architecture:

- The number of the processors on the chip
- The number of levels of the cache
- The number of the cache that is shared

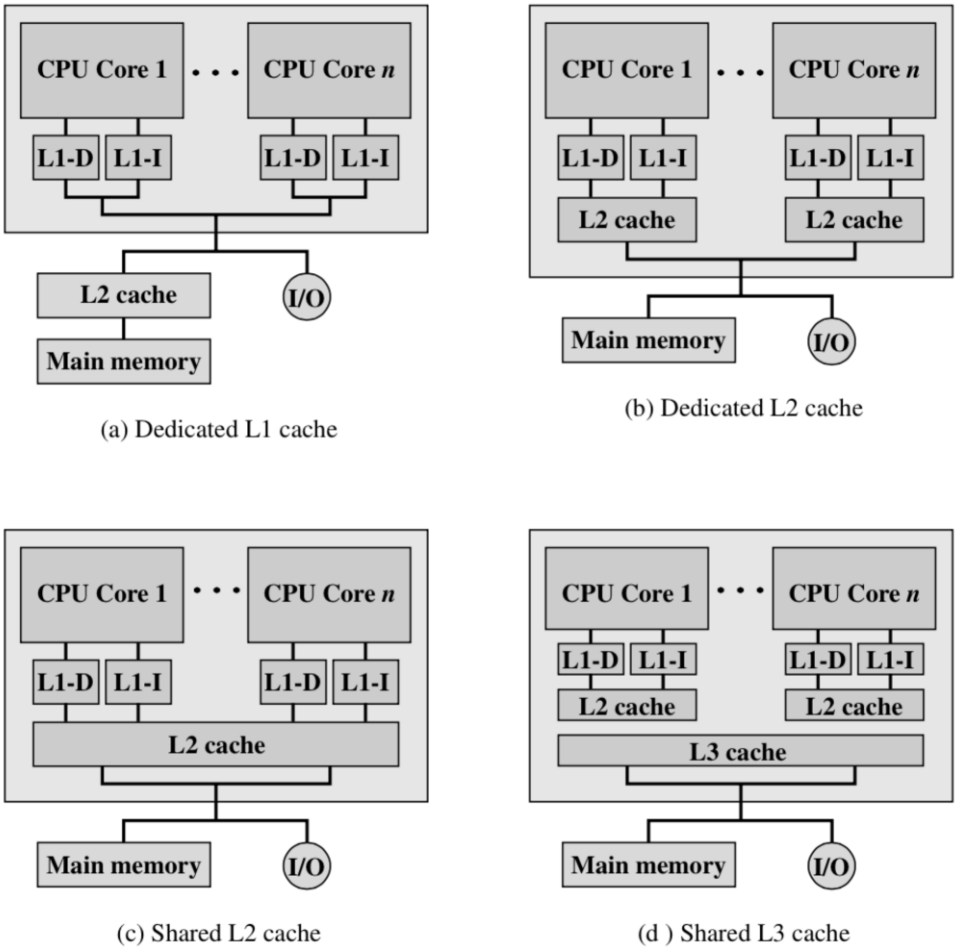

Figure-5. Multi-core Organization Alternatives

There are four general organization of the multi-core system showing in figure-5. The figure-5a mostly shows in the multi-core computer chips in old days, but this organization still been used in some embedded systems. In this organization, L1 is the only cache on the chip, and every core will has its own L1 cache. Each L1 cache was divided to the instruction cache and data cache.

The figure-5b also doesn't have any sharing cache on the chip. But differ from the first one, in this organization, the L2 cache is also on the chip.

The organization shows in figure-5c has a sharing L2 cache, the Intel Core Duo is using this kind of organization.

At last, in figure-5d, we can see the L3 shows up shared by all the cores. Still each core will have their L1 cache and L2 cache. The well-known Intel Core i7 is using this organization.

3.2 The L2 cache

As we can see, the update of the processor is accompanied with the increasing of the levels of cache. With the speed of the processor highly increased, the speed of the processor read from the main memory was challenged because the increasing step of the main memory read/write speed is much slower than the processor's executing speed. Also, the high-speed memory is too expensive to use a lot, so the Intel comes up an idea, which is to combine the expensive high-speed memory and cheap low-speed memory, using a few high-speed memory and a lot of low-speed memory. That's the reason the cache was used. When they found the L1 cache is not enough, they put one more cache into the processor, which is the L2 cache. L2 cache is one of the most important components of the processor. To have a better knowledge about the structure of the L2 cache, we'll take OpenSPARC T1 processor as an example.

Using a shared L2 Cache can bring a lot of benefits. You don't need to duplicate the data that on the shared cache level, which save time and space. Also we can allocate the shared cache to the different chips dynamically, algorithm may apply. Shared L2 Cache can confine the cache coherency problem within L1 level, which can approve the total performance. And with the L2 shared Cache, the interprocessor communication becomes

very easy, we can use the shared cache to accomplish it.

The potential advantage of the dedicated L2 Cache is that every chip can access their private L2 cache rapidly, with the L3 shared cache, the dedicated L2 cache seems better than a big shared L2 cache.

3.3 The L2 cache Structure

The OpenSPARC T1 processor L2 cache is 3 Mbytes in size and is composed of four symmetrical banks that are interleaved on a 64-byte boundary. Each bank operates independently of each other. The banks are 12-way set associative and 768 Kbytes in size. The block (line) size is 64 bytes, and each L2 cache bank has 1024 sets. The L2 cache uses a directory-based cache coherence scheme. The L2 cache is responsible for maintaining the on- chip coherency across all L1 caches on the chip by keeping a copy of all L1 tags in a directory structure. Since the OpenSPARC T1 processor implements system on a chip, with single memory interface and no L3 cache, there is no off-chip coherency requirement for the OpenSPARC T1 L2-cache other than it needs to be coherent with the main memory [3].

The Figure. 6 shows the structure of the L2 cache.

According to functions, each bank can be divided into three modules:

- *sctag* is the control and tag module, which contains L2 tag and VUAD arrays, directory, arbitration, input queue(IQ), output queue(OQ), and snoop input queue (SNPIQ);
- *scbuf* is the buffer module, which contains miss buffer(MB),write back buffer(WB),fill buffer(FB),and remote DMA buffer(Rdma);

- *scdata* is the data storage module, which contains L2- data arrays[4].

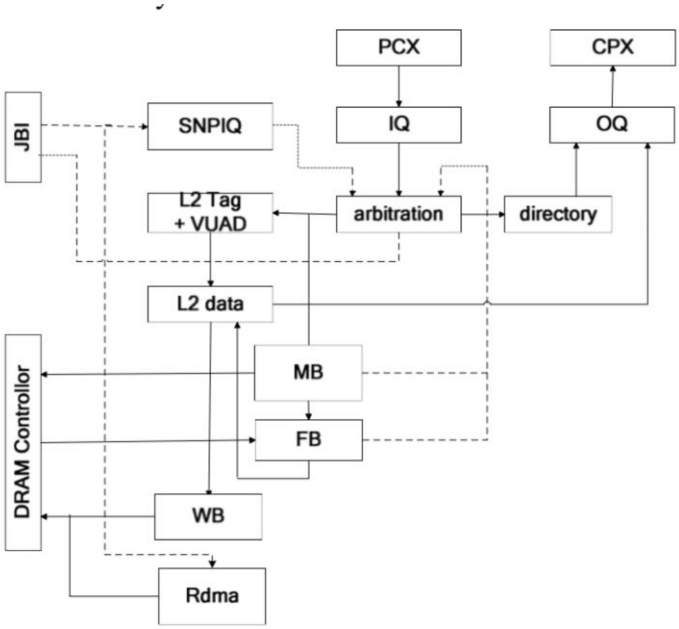

Figure 6. The structure of L2 cache [3]

The PCX will send request to the arbitration via IQ, the arbitration will monitors the instruction sent from SNPIQ, WB and FB. Arbitration will decides which request will be sent to the L2 Tag and MB according to the result. Then the L2 Tag and VUAD will determine the legitimacy of the address and pass the request to L2-data [4].

4. Performance Issue

The pace of the improvement of the processor's performance is extremely fast. But for a variety of reason, there is a limit for the large uniprocessor. The multi-core processor, for

now, is the only way to build high performance processors. A lot of the problems for the uniprocessor such as the limit of the parallelism, power issues, the difficulty of the design and debug for the large-scale uniprocessor, can be easily solved by the multi-core architecture. But even we found the multi-core processor is the best architecture for now, it not enough. It has more potential to be discovered.

There are two radically different kinds of workloads that the multi-core processor used: the throughput-sensitive application, which has high-level parallel, and latency-sensitive application, which has less parallel. This chapter will briefly introduce both of these two workloads.

4.1 Throughput issue

Decades ago, the throughput of a single system is, in today's perspective, really small, the processor can easily handle all the throughput. But with the rise of the Internet, whether a server can deal with the huge amount of independent request arriving rapidly becomes a critical issue for the developer. Since individual network requests are typically completely independent tasks, whether those requests are for web pages, database access, or file service, they are typically spread across many separate computers built using high-performance conventional microprocessors, a technique that has been used at places like Google for years, in order to match the overall computation throughput to the input request rate [5]. The multi-core processor can also play an important part in this condition. By replace the servers with multi-core processor, the system can be physically smaller and use less power than using two separate uniprocessor because the hard drive, memory and power supply can be shared by those cores.

The first CMPs targeted toward the server market implement two or more conventional

superscalar processors together on a single die [6]. When using the multi-core processor, it can fit in the space that can only contain one single processor. It's obvious that multi-core architecture can reduce the total volume of the whole system. All cores in that chip can share the same connection to the system, so it also can guarantee that power can be saved.

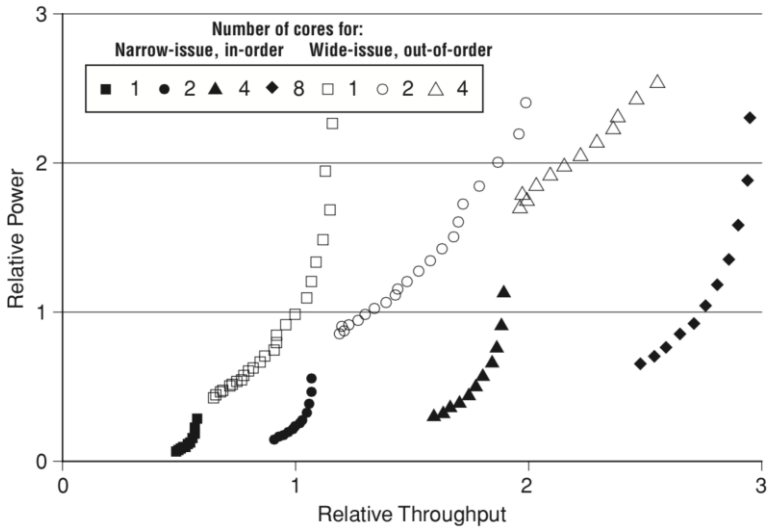

Figure 7. Comparison of power usage by equivalent narrow-issue, in-order and wide-issue, out- of-order processors on throughput-oriented software, from [7].

4.2 Latency Issue

For many applications, the high throughput is very important, but there are still huge amounts of application, which take the latency issue as the most important criteria. Take the desktop processor as an example, the user always except their desktop has a high response speed of their commands.

By using multi-core processor, you can speed up those applications, which you need them response as quickly as possible, but it requires the programmer break the long latency thread into several small thread that can processed by the processors in parallel because the secret of the speed improvement of the multi-core processor is parallelism. You have to make sure that the threads of the application are running in the separate processor.

4.2.1 "Helper" Thread

The simplest way to apply parallelism within the multi-core processor to improve the performance of the single thread is to use "helper" threads. Those "helper" threads can work for the main thread and improve the overall processing speed. By using the "helper" threads, you can start a long-latency operation early or make preparation for the operation by computing the key value of the operation first. Two problems that have been addressed using this technique are making hard-to-predict branch predictions early [8] and prefetching irregular data into on-chip caches [9-11]. In most cases, the hardware branch prediction and the prefetch units are very well, but for other complex and irregular codes, it better to start the prediction and prefetch before the main thread actually need them. This section will mainly talk about the prefetch using helper threads.

To accomplish the prefetching step, the helper threads will run using the copy of the main thread, which reduced all the unnecessary codes that are not related to their main task, then the helper threads will prefetch all the important data into the cache, which can be shared by other threads. Those data that we prefetched will have the same address as the one in the main thread in case the unnecessary data coming into the cache. Doing this can

save us a lot amount of time because the processor do not have to wait for a miss to main memory (processor sometimes waits for hundreds of cycles).

But there are still some problems about the “helper” threads. The first is that only a certain amount of the single-thread programs can be sped up with this method. Most of the integer application will have a modest footprint, which can help the application eliminate most of the cache miss. And for those large footprint such database, it’s easy for them to work parallel into some true threads instead of the helper threads. Besides this, there are also many applications can be parallelized by the hardware mechanisms or prefetch instructions right in the main thread. The second problem is that between the helper threads and the main thread they need a tight synchronization to keep the helper threads in a proper distance ahead of the main thread. If this distance is too far, the prefetched data may be replaced by subsequent prefetches before the main thread use it; if the distance is not far enough, the prefetching process may not be finished when the main thread needs those data. So in conclusion, the amount of speedup that can be achieved by the helper thread is modest.

4.2.2 Thread-level speculation

When we want to improve the performance of a single-thread application by using parallel thread within a multi-core processor, we should parallelize the program so each core can work independently. This is more difficult than “helper” thread because you need to split the single thread and also make sure they will produce the right result. The technique we used here is thread-level speculation (TLS).

TLS is what we can use to parallelize a single-thread program across the cores of a multi-core processor. To make sure that the result of the program is correct after we break it

into several threads, the hardware must track all the interthread dependencies. When hazards happened, such as a thread try to read data too early, we must re-execute it with the right data. This process can be extremely complex because you need to find all the dependences in the program. Speculation allows parallelization of the program into thread without knowing the where the dependency will occur. All the threads can keep going until a true dependency is detected.

To support speculation, one needs special coherency hardware to monitor data shared by the threads. This hardware must fulfill five basic requirements, illustrated in Fig. 8. The figure shows some typical data access patterns in two threads, i and i + 1. Figure 8(a) shows how data flows through these accesses when the threads are run sequentially on a normal uniprocessor. Figures 8(b)–8(e) show how the hardware must handle key situations that occur when running threads in parallel [12,13].

Besides this, the system that supports speculation should also have the mechanism to sequencing threads across multiple cores at runtime. This always accomplished by both the software and hardware.

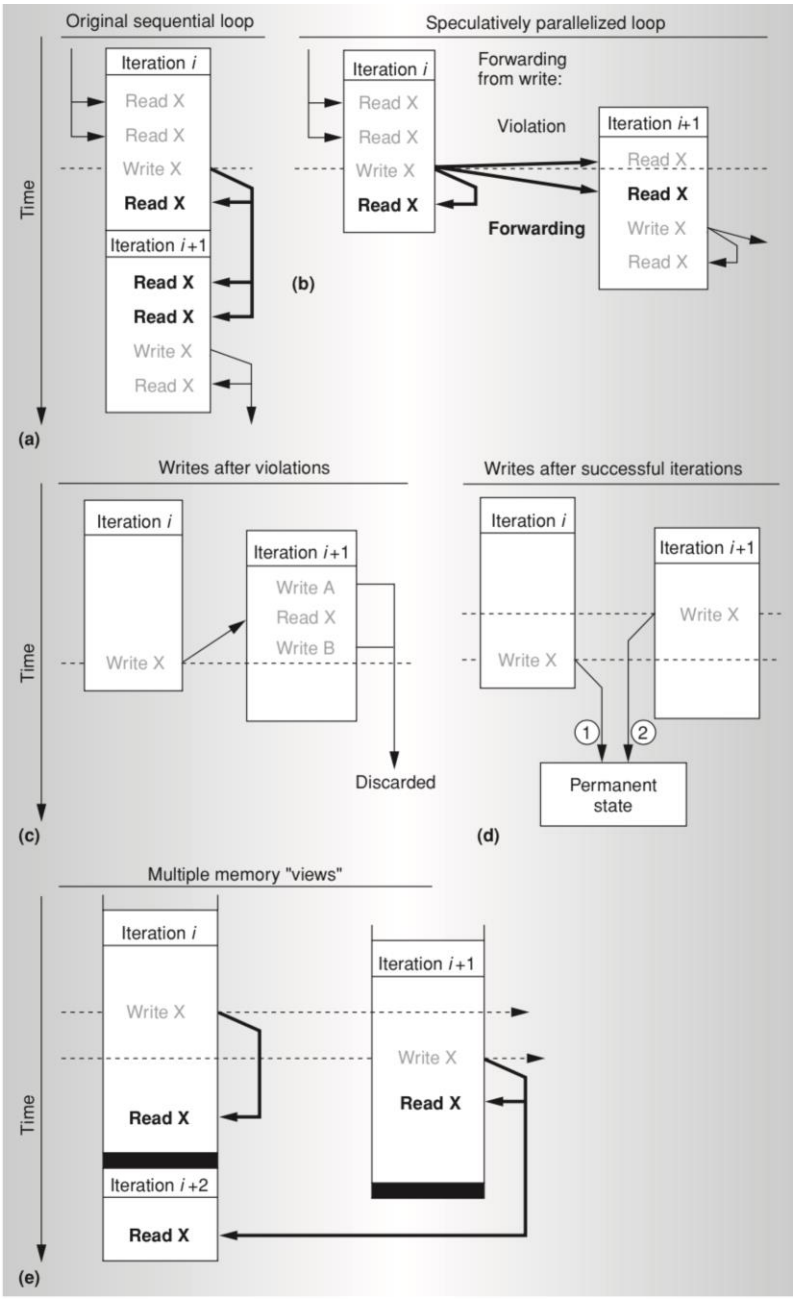

Figure 8. Five basic requirements for special coherency hardware: a sequential program that can be broken into two threads (a); forwarding and violations caused by intersections of reads and writes (b); speculative memory state eliminated following violations (c); reordering of writes into thread commit order (d); and memory renaming among threads (e)[12].

5. Intel and AMR Multi-core Architecture

5.1 Intel x86 Multi-core Architecture

This section will mainly introduce Intel Core i7.

Intel Core i7 was produced in 2007, it implement four x86 SMT processor, each processor has a L2 cache and a shared L3 cache (Fig 5(d)).

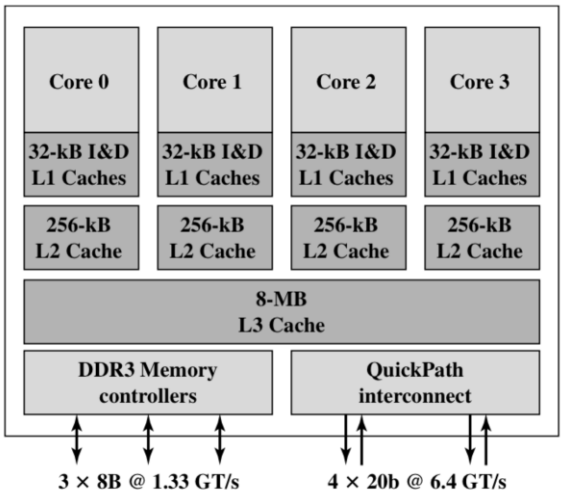

Figure 9. Intel Core i7 Block Diagram

The organization of the Intel Core i7 is shown in Fig 9, every core has its own L2 Cache, and all the cores share the 8-MB L3 Cache. To make the performance of the processor better, Intel uses prefetching, in which the hardware will check the pattern of the memory access, try to fill the cache with the data that will be needed soon. Table 1 shows the cache access latency, by using dedicated L2 Cache and providing high speed L3 Cache, the Intel Core i7 improved the performance on the L2 Cache.

Table 1. Cache Latency

CPU	Clock Frequency	L1 Cache	L2 Cache	L3 Cache
Core 2 Quad	2.66 GHz	3 cycles	15 cycles	—
Core i7	2.66 GHz	4 cycles	11 cycles	39 cycles

The Intel Core i7 support two kinds of external communication method with other chips. The DDR3 Memory Controller let the controller of the DDR main memory on the chip. The interface including 3 8 bytes wide channel, so the total wide of the bus is 192 bits, for an aggregation data rate up to 32GB/s. With the memory controller on the chip, the Front Side Bus was eliminated [13].

The QuickPath Interconnect (QPI) is a cache-coherent, point-to-point link based electrical interconnect specification for Intel processors and chipsets. It en- ables high-speed communications among connected processor chips. The QPI link operates at 6.4 GT/s (transfers per second). At 16 bits per transfer, that adds up to 12.8 GB/s, and since QPI links involve dedicated bidirectional pairs, the total band- width is 25.6 GB/s [13].

5.2 ARM11 MPCore

AMR11 MPCore is a multi-core processor within AMR11 processor family. AMR11 MPCore can implement up to four processors in a chip, every processor have their own L1 instruction and data cache. The Figure 10 shows the diagram of the AMR11 MPCore.

The main elements of the AMR11 MPCore is following:

- Distributed interrupt controller (DIC): Process the interrupt, including interrupt detection and interrupt prioritization. The DIC distributes interrupt to each processor.

- CPU interface: handle interrupt acknowledgement and interrupt masking.
- Timer: every CPU has its own timer, which can produce interrupt.

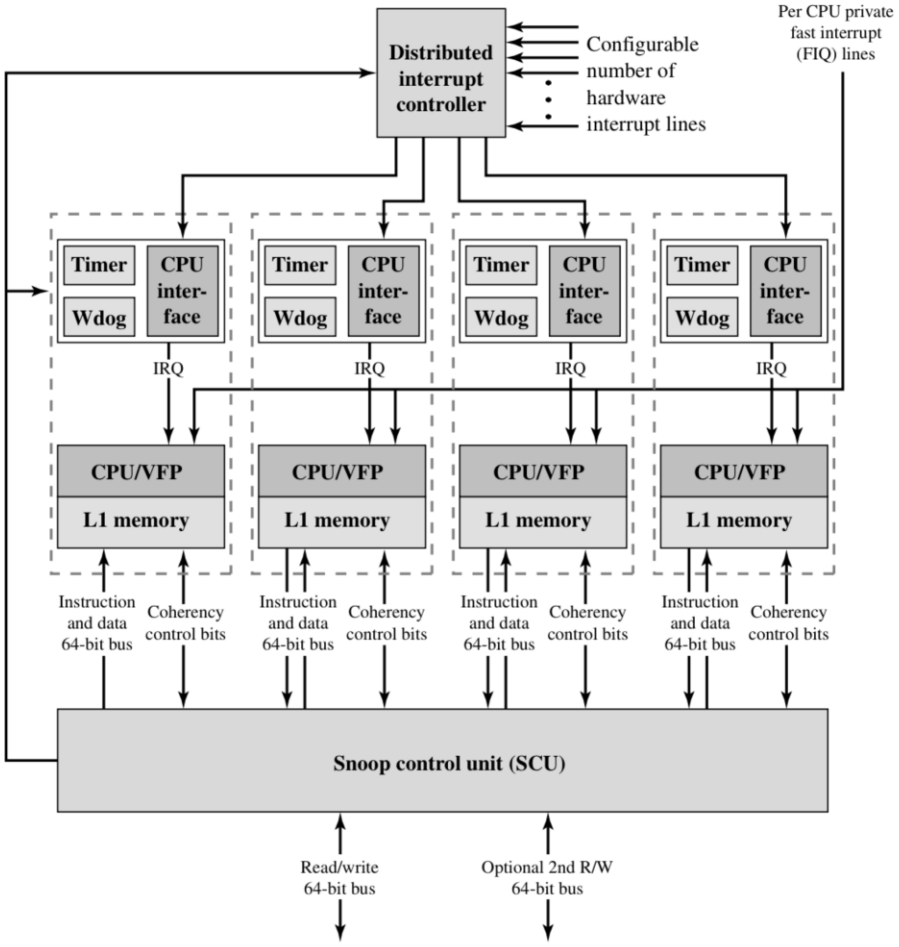

Figure 11. AMR11 MPCore Diagram [13]

- Watchdog: make alarm when software error happens.
- CPU: an ARM11 processor.
- VFP: a processor that can implement floating-point operations into hardware.
- L1 Cache: the instruction and data cache that every processor has.
- Snoop Control Unit: maintaining the coherency of the L1 data cache.

From the diagram we can see that the DIC handles the interrupt in the ARM11 MPCore. It is independent to the CPU, and the CPU can access the DIC via SCU by a private interface.

5.2.1 Interrupt Handling

The distributed Interrupt Controller (DIC) can provide a lot of functions to handle interrupt. It can mask the interrupt; prioritization the interrupt; tacking interrupt status and allocate the interrupt to the MP11 CPU. The DIC is an independent unit set aside the CPU, which make the interrupt independent to the CPU. It can use three different ways to send the interrupt the processor:

- Send the interrupt to the specific processor
- Send the interrupt to all the processor
- Send the interrupt to a defined set of processor

From the point of view of software running on a particular CPU, the OS can generate an interrupt to all but self, to self, or to specific other CPUs. For communication between threads running on different CPUs, the interrupt mechanism is typically combined with shared memory for message passing. Thus, when a thread is interrupted by an interprocessor communication interrupt, it reads from the appropriate block of shared memory to retrieve a message from the thread that triggered the interrupt. A total of 16 interrupt IDs per CPU are available for interprocessor communication [13].

5.2.2 Cache Coherency

The SCU is designed to solve the problems related to access to shared data and the limitation introduced by coherence traffic.

The SCU provide three types of optimization to improve the performance of MESI state migration:

Direct Data intervention (DDI) can duplicate removed data from one CPU's L1 Cache to other CPU's L1 Cache without accessing the outer memory. This can decrease the numbers of times L1 Cache read from L2 Cache.

Duplicated tag RAMs are duplicated L1 tag RAM using by SCU, checking the availability of the data before sending coherence instruction to the CPU. Those instructions were only be sent to CPU that need update the cache coherency and this will save a lot of power.

The migratory lines allow dirty data migrate from one CPU to another directory, without write to L2 Cache or read from memory.

6. Discussion

Multi-core processor is one of the most important technologies during the development of the modern microprocessor. A lot of researches focus on exploring the technique to over come the issues, both hardware and software issues, to improve the performance of the processor. This paper began with brief introduction of the multi-core processor, what are the advantages of it; then explain the hardware issues about the multi-core processor, the problems we should concerned about when implementing the multi-core. Following is the research of the multi-core architecture. It focuses on the chip organizations, comparing different kinds of organizations, and also explained the function of the L2 Cache and why it can improve the overall performance of the processor. The performance issues are also introduced. Both the throughput and latency issue are mention and fully discussed. And

at last, two examples of the modern multi-core processors are provided to give us a better view of the multi-core technology.

Nowadays, the world cannot only satisfy by the dual-, tri-, quad-, hex-, oct-core chips. The general trend has moved from them to ones with tens or even thousands of cores. Also, there are multiple of different technologies and mechanisms are added and work with the multi-core processor such as simultaneous multithreading, memory-on-chip, and special-purpose "heterogeneous" core promise further performance.

The performance issue is not the only driven of the development, the general trend also focus on improving energy-efficiency (performance-per-watt). Advanced fine-grain or ultra fine-grain power management and dynamic voltage and frequency scaling are developed to achieve this target.

From all-mentioned-above, the single core processor has reached its limit and the multi-core processor is the general trend. Even it has some disadvantages, but still it's one of the most important methods to further improving the performance of the processor.

References

[1]. Rockwell R65C00/21 Dual CMOS Microcomputer and R65C29 Dual CMOS Microprocessor". Rockwell International. October 1984.

[2]. "Rockwell 1985 Data Book". Rockwell International Semiconductor Products Division. January 1985.

[3]. Sun Microsystems, Int. OpenSPARC T1 Microarchitecture Specification [R], 2006.8.

[4]. Wang Lei, Fan Xiao-ya. Study on L2 Cache of Multi-core Processor and Optimization for Embedded[R], Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference, 2011. 9.

[5]. L. Barroso, J. Dean, and U. Hoezle, "Web search for a planet: the architecture of the Google cluster," IEEE Micro., Vol. 23, No. 2, pp. 22–28, Mar.–Apr. 2003.

[6] S. Kapil, "UltraSPARC Gemini: dual CPU processor," in *Hot Chips 15*, Stanford, CA, Aug. 2003.

[7] T. Agerwala and S. Chatterjee, "Computer architecture: challenges and opportunities for the next decade," *IEEE Micro*, Vol. 25, No. 3, pp. 58–69, May/June 2005.

[8] R. S. Chappell, F. Tseng, A. Yoaz, and Y. N. Patt, "Difficult-path branch prediction using subordinate microthreads," in *Proc. 29th Annual Int. Symp. Computer Architecture*, Anchorage, AK, June 2002, pp. 307–317.

[9] D. Kim, S. S. Liao, P. H. Wang, J. del Cuvillo, X. Tian, X. Zou, H. Wang, D.

Yeung, M. Girkar, and J. P. Shen, “Physical experimentation with prefetching helper threads on Intel’s hyper-threaded processors,” in *Proc. Int. Symp. Code Generation and Optimization (CGO 2004),* Palo Alto, CA, Mar. 2004, pp. 27–38.

[10] Y. Song, S. Kalogeropulos, and P. Tirumalai, “Design and implementation of a compiler framework for helper threading on multi-core processors,” in *Proc. 14th Int. Conf. on Parallel Architectures and Compilation Techniques (PACT-2005),* St. Louis, MO, Sept. 2005, pp. 99–109.

[11] C.-K. Luk, “Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processors,” in *Proc. 28th Annual Int. Symp. Computer Architecture (ISCA-28)*, Go¨teborg, Sweden, June 2001, pp. 40–51.

[12] Olukotun K, Hammond L, Laudon J. Chip multiprocessor architecture: techniques to improve throughput and latency[J]. Synthesis Lectures on Computer Architecture, 2007, 2(1): 1-145.

[13] William S. Computer Organization and Architecture: Design for Performance, Eight Edition[M]. 2009. 4.