University of Jordan
Faculty of Engineering and Technology
Computer Engineering Department

Student Name: Yousef Yaseen
Student No.: 8090586

Report Subject:
Multi-core Processing – Advantages and Challenges

Multi-core Processing - Advantages and Challenges

Yousef Yaseen

December 2, 2009

Abstract

Today, multicore processors dominate server, desktop and notebook computer's market. It is clear that multicore processors will be with us for the foreseeable future; there seems to be no alternative way to provide substantial increases of microprocessor performance in the coming years. While processors with a few cores are common today, the number of cores is projected to grow as we enter the era of many-core computing. This paper will review this trend towards multicore processing, advantages and challenges that accompany it.

1. Introduction

In April 1965, Gordon Moore wrote an article for Electronics magazine titled "Cramming more components onto integrated circuits" [1]. He predicted that the number of transistors on a chip would double every 12 months into the near future (he later refined this, in 1975, to every two years). This exponential trend remains the driving force behind the integrated circuits industry. The constant decrease in feature size leads to an increase on transistor count on chip area which enables processor architects to improve performance by designing more complex processors. This amount of transistors on the chip area increases power consumption and produces more heat. These two factors, power consumption and thermal management, are the most important design limitation factors that chip manufacturers have struggled to cap today [2,3]. Even performance-enhancing approaches like running multiple instructions per thread have bottomed out. These reasons began to reveal some limitations in using predominately frequency as a way of improving performance and processor performance increases have begun slowing [5].

Chip manufacturers have opted to use the extra transistors to build multiple processor cores on the same die, they are building chips with multiple cooler-running, more energy-efficient processing cores instead of one increasingly powerful core. Multicore processing is the best way to meet performance demands within a constrained power envelope. This led to the IBM POWER4 the industry's first multicore design in 2001. The multicore chips don't necessarily run as fast as the highest performing single-core models, but they improve overall performance by handling more work in parallel. As an example, Ramanthan [4] noted that cranking up the frequency on a single-core processor by a factor of 1.13X can increase power consumption by 1.73X. However, with a modest decrease in clock speed, it's possible to cut the power in half. This makes it possible to add a second core and gain 1.7X the performance without increasing the original power consumption.

Multicore processors have several other benefits as well they improve an operating system's ability to multitask applications, and another benefit comes from individual applications optimized for multicore processors. These applications, when properly programmed, can split a task into multiple smaller tasks and run them in separate threads [4,6].

Having multiple cores on a single chip gives rise to some problems and challenges. Power and temperature management are two concerns that can increase exponentially with the addition of multiple cores [9]. Cache coherence is another challenge, since most designs have distributed L1 and in some cases L2 caches which must be coordinated. And finally, using a multicore processor to its full potential is another issue. If programmers don't write applications that take advantage of multiple cores there is no gain, and in some cases there is a loss of performance. Application need to be written so that different parts can be run concurrently.

For the future, there are a number of open issues: Interconnection networks, Homogeneous vs. Heterogeneous, Parallel programming and Software licensing.

2. Multi-core Processors Advantages

Multicore processors take advantages of the relationship between power and frequency. By connecting multiple cores, each core is able to run at lower frequency, dividing among them the power given to a single core. The result is a big performance increase over a single core processor. The following illustration based on Intel's lab experiments illustrates this key advantage.

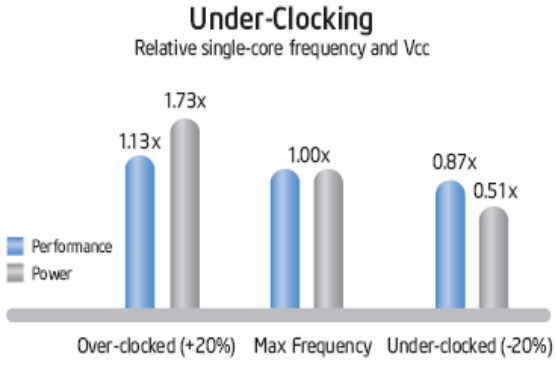

Figure 1. [4] Single Core Performance

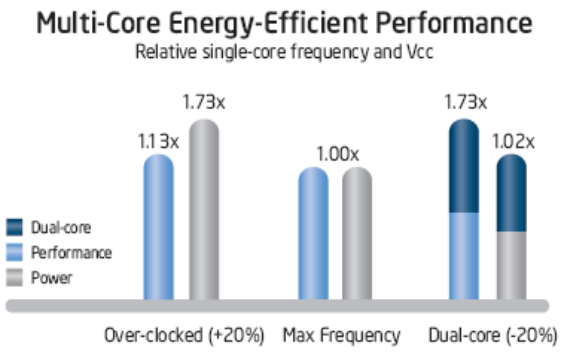

Figure 2. [4] Multi-core Performance

As seen in Figure 1, increasing clock frequency by 20 percent to a single core delivers a 13 percent performance gain, but requires 73 percent greater power. Conversely, decreasing clock frequency by 20 percent reduces power usage by 49 percent, but results in just a 13 percent performance loss. While in Figure 2 when a second core is added on the underclocked example in Figure 1, this results in a dual-core processor that at 20

percent reduced clock frequency effectively delivers 73 percent more performance while using approximately the same power as a single-core processor at maximum frequency.

This relationship between power and frequency can be effectively used to multiply the number of cores from two to four, and then eight and more, to deliver continuous increases in performance without increasing power usage.

Multicore processors have several other benefits as well they improve an operating system's ability to multitask applications. For instance, say you have a virus scan running in the background while you're working on your word-processing application. This often degrades responsiveness so much that when you strike a key, there can be a delay before the letter actually appears on the screen. On multicore processors, the operating system can schedule the tasks in different cores so that each task runs at full performance. And another benefit comes from individual applications optimized for multicore processors. These applications, when properly programmed, can split a task into multiple smaller tasks and run them in separate threads. For instance, a word processor can have "find and replace" run as a separate thread so doing a "find and replace" on a big document doesn't have to keep you from continuing to write or edit. [4,6].

3. Multi-core Challenges

Having multiple cores on a single chip gives rise to some problems and challenges. Power and temperature management are two concerns that can increase exponentially with the addition of multiple cores. Cache coherence is another challenge, since most designs have distributed L1 and in some cases L2 caches which must be coordinated. And finally, using a multicore processor to its full potential is another issue. If programmers don't write applications that take advantage of multiple cores there is no gain, and in some cases there is a loss of performance. Application need to be written so that different parts can be run concurrently (without any ties to another part of the application that is being run simultaneously).

3.1 Power and Temperature

If two cores were placed on a single chip without any modification, the chip would, in theory, consume twice as much power and generate a large amount of heat. In the extreme case, if a processor overheats your computer may even combust. To account for this most designs run the multiple cores at a lower frequency to reduce power consumption [6].

To combat unnecessary power consumption many designs also incorporate a power control unit that has the authority to shut down unused cores or limit the amount of power. By powering off unused cores and using clock gating the amount of leakage in the chip is reduced.

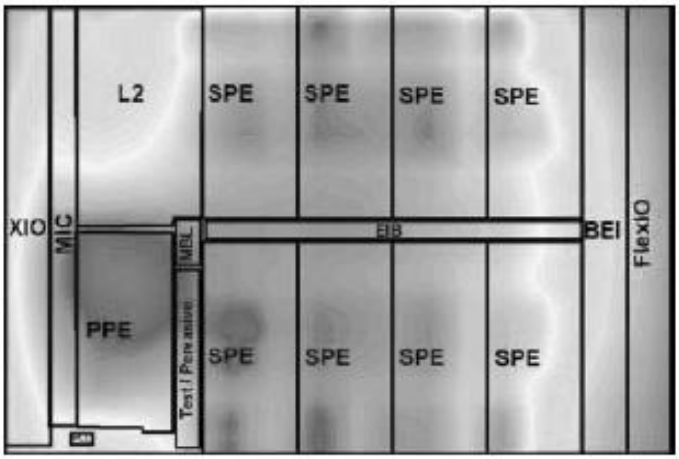

Figure 3. [6] CELL Thermal

To lessen the heat generated by multiple cores on a single chip, the chip is architected so that the number of hot spots doesn't grow too large and the heat is spread out across the chip. As seen in Figure 3, the majority of the heat in the CELL processor is dissipated in the Power Processing Element and the rest is spread across the Synergistic Processing Elements. The CELL processor follows a common trend to build temperature monitoring into the system, with its one linear sensor and ten internal digital sensors [12].

3.2 Cache Coherence

Cache coherence is a concern in a multicore environment because of distributed L1 and L2 cache. Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date version. For example, imagine a dual-core processor where each core brought a block of memory into its private cache. One core writes a value to a specific location; when the second core attempts to read that value from its cache it won't have the updated copy unless its cache entry is invalidated and a cache miss occurs. This cache miss forces the second core's cache entry to be updated. If this coherence policy wasn't in place garbage data would be read and invalid results would be produced, possibly crashing the program or the entire computer. In general there are two schemes for cache coherence, a snooping protocol and a directory-based protocol. The snooping protocol only works with a bus-based system, and uses a number of states to determine whether or not it needs to update cache entries and if it has control over writing to the block. The directory-based protocol can be used on an arbitrary network and is, therefore, scalable to many processors or cores, in contrast to snooping which isn't scalable. In this scheme a directory is used that holds information about which memory locations are being shared in multiple caches and which are used exclusively by one core's cache. The directory knows when a block needs to be updated or invalidated [16].

Directory based protocols are alternatives to snoopy based protocols which achieve low latencies and high bandwidth because of broadcasting and this protocol is implemented in present day technologies like in Core2Duo processors.

3.3 Multithreading

The most important, issue is using multithreading or other parallel processing techniques to get the most performance out of the multicore processor. "With the possible exception of Java, there are no widely used commercial development languages with [multithreaded] extensions." [14] Rebuilding applications to be multithreaded means a complete rework by programmers in most cases. Programmers have to write applications with subroutines able to be run in different cores, meaning that data dependencies will have to be resolved or accounted for (e.g. latency in communication or using a shared cache). Applications should be balanced. If one core is being used much more than another, the programmer is not taking full advantage of the multicore system. Some companies have heard the call and designed new products with multicore capabilities; Microsoft and Apple's newest operating systems can run on up to 4 cores, for example. [13, 14]

4. Open Issues

4.1 Interconnection Networks

The cores on a die must be connected to each other, and there are several possibilities, Classical buses, Rings, Crossbars, Switched networks, and Hierarchical interconnects.

It is quite clear that manycore processors will have neither buses, rings nor crossbars. For buses, long lines give high power consumption and low speed. Crossbars scale as the square of the number of ports and thus become untenable. Rings scale in terms of area and power. This leaves switched networks and hierarchical interconnects as the main competitors for the future. Note also that if cache coherency is to be supported, the coherency mechanism interacts heavily with the interconnect structure. A mesh network, for instance, fits naturally with a directory based coherency mechanism, whereas a hierarchical system could have snooping in the leaves using rings or buses and use directories between the groups [8].

State of the art Crossbars are often used in designs with few processors, but rings and meshes are becoming more common.

Current challenges Rings and buses fit well with snooping cache coherence protocols, but for meshes directory based protocols are needed, and they have some scaling issues. Here hierarchical organizations might help.

4.2 Homogeneous vs. Heterogeneous Cores

Cores in a multicore environment could be homogeneous or heterogeneous. Homogenous cores are all exactly the same: equivalent frequencies, cache sizes, functions. However, each core in a heterogeneous system may have a different function, frequency, memory model and heterogeneous cores may have the same instruction set or not.

Homogeneous cores are easier to produce since the same instruction set is used across all cores and each core contains the same hardware. Each core in a heterogeneous environment could have a specific function and run its own specialized instruction set. Building on the CELL model, a heterogeneous model could have a large centralized core built for generic processing and running an OS, a core for graphics, a communications core, an enhanced mathematics core, an audio core, a cryptographic core, and the list goes on. This model is more complex, but may have efficiency, power, and thermal benefits that outweigh its complexity [8].

State of the art Most designs targeting desktops, laptops and servers are homogeneous, but in the embedded sphere, heterogeneity is more common, evidenced by for instance the Cell processor and the typical architecture of mobile phones.

Current challenges For heterogeneous systems, programming tools remain a challenge as compared to on a homogeneous system.

4.3 Parallel Programming

In May 2007, Intel fellow Shekhar Borkar stated that "The software has to also start following Moore's Law, software has to double the amount of parallelism that it can support every two years." [15] Since the number of cores in a processor is set to double every 24 months, it only makes sense that the software running on these cores takes this into account. Ultimately, programmers need to learn how to write parallel programs that can be split up and run concurrently on multiple cores instead of trying to exploit single-core hardware to increase parallelism of sequential programs. Once programmers have a basic grasp on how to multithread and program in parallel, instead of sequentially, ramping up to follow Moore's law will be easier.

As seen in Figure 4, Amdahl's Law is frequently used in parallel programming to predict the theoretical maximum speedup using multiple processors. "If 50% of your program is serial and the other half can be parallelized, the biggest speedup you're going to see is a factor of two," says Microsoft's Larus [10].

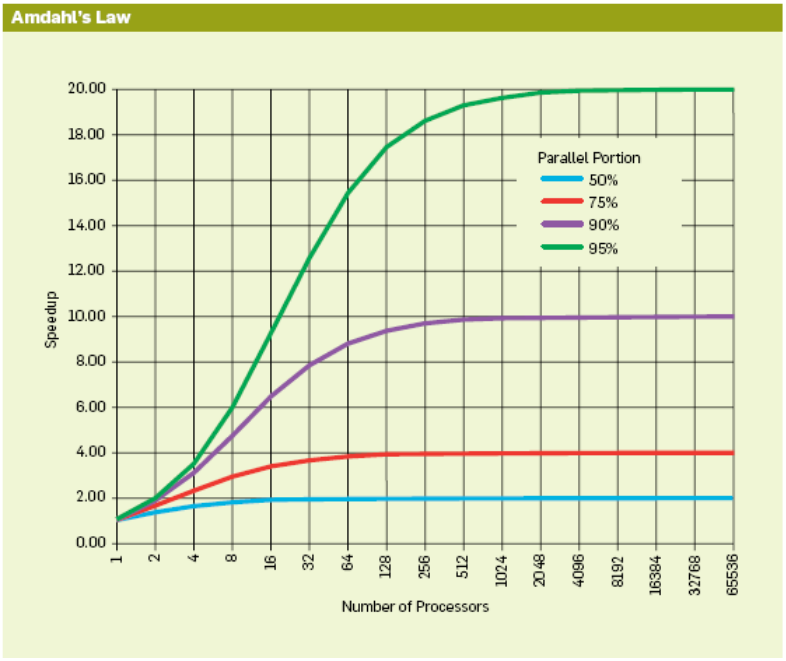

Figure 4. [10]

State of the art Today, most multicore programming is done using either thread (pthreads, Windows threads or Java threads), OpenMP or the Intel TBB.

Current challenges New programming languages generally take quite long to be widely adopted, and when it happens, it is often because of a change in the computing environment. Thus Java adoption was driven by the arrival of the web. Multicore is an even more disruptive technology change, which could drive the adoption of new languages. However, the multicore problem has a large legacy aspect, which was less true of the advent of the web, which might push development in the direction of conservative extensions of existing languages.

4.4 Software Licensing

Software vendors charge customers in various ways for using their products. Intel defines a processor as a unit that plugs into a single socket on the motherboard, regardless of whether it has one or more cores, and advocates that software vendors charge accordingly, explained Jeff Austin, the company's desktop product manager. Microsoft agree and don't charge extra for using their software on multicore processors. BEA Systems and Oracle, on the other hand, charge more to use their software on multicore

chips for per-processor licensing. “Customers get added performance benefit by running our software on a chip with two cores, so we charge a fraction of the single CPU price for additional cores,” said Bill Roth [5], the company’s vice president of product marketing. Multicore-chip makers are concerned that this type of policy will hurt their products’ sales.

As seen in Figure 5, Oracle Assigns “Processor Factors” to classes of CPUs.

Oracle Processor Licensing:	Cores	Processor Factor	CPUs for SW (other than SE and SE One programs) Licenses
UltraSPARC T1	8	0.25	2
AMD / Intel	4	0.50	2
All other Multi-core Chips (IBM Pseries, SM, USIV, etc.)	2	0.75	2
Single Core Servers	1	1.00	1

Figure 5. [11] Oracle Licensing

Current challenges

- Multicore price calculations are perceived to be complicated.
- Approach taken by Oracle requires hardware benchmarking be established and maintained.
- Benchmarks (as they influence pricing) likely to be challenged by customers and therefore should be independent and verifiable.
- If a customer has not yet purchased the hardware, software costs may vary depending on the hardware purchased

5. Conclusion

Before multicore processors the performance increase from generation to generation was easy to see, an increase in frequency. This model broke when the high frequencies caused processors to run at speeds that caused increased power consumption and heat dissipation at detrimental levels. Adding multiple cores within a processor gave the solution of running at lower frequencies, but added interesting new problems.

Multicore processors are architected to adhere to reasonable power consumption, heat dissipation, and cache coherence protocols. However, many issues remain unsolved. In order to use a multicore processor at full capacity the applications run on the system must be multithreaded. There are relatively few applications (and more importantly few programmers with the know-how) written with any level of parallelism. The interconnection networks also need improvement.

With so many different designs it is nearly impossible to set any standard for cache coherence, interconnections. The greatest difficulty remains in teaching parallel programming techniques (since most programmers are so versed in sequential programming) and in redesigning current applications to run optimally on a multicore system. Multicore processors are an important innovation in the microprocessor timeline. With skilled programmers capable of writing parallelized applications multicore efficiency could be increased dramatically. In years to come we will see much in the way of improvements to these systems. These improvements will provide faster programs and a better computing experience.

References

[1] Gordon E. Moore: Cramming More Components onto Integrated Circuits. Electronics, April 19, 1965.

[2] Brooks, D., Martonosi, M.: Dynamic Thermal Management for High-Performance Microprocessors, In: Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, January 2001.

[3] Naveh, A., Rotem, E., Mendelson, A., Gochman, S.: Power and Thermal Management in the Intel® Core Duo Processor, Intel Technology Journal, (2006), 10(2).

[4] R.M. Ramanathan: Intel® Multicore Processors - Making the Move to Quad-Core and Beyond, 2007.

[5] Geer, D.: Chip makers turn to multicore processors, 2005.

[6] Shekhar Borkar: Thousand Core Chips - A Technology Perspective, 2007.

[7] R. Merritt, "CPU Designers Debate Multi-core Future", EETimes Online, February 2008, http://www.eetimes.com/showArticle.jhtml?articleID=206105179.

[8] Fax´en, K., Bengtsson, C., Brorsson, M., Grahn, H.: Multicore Computing - the State of the Art, December 3, 2008.

[9] Agarwal, A., Levy, M.: The KILL Rule for Multicore, At 44th DAC, June 2007.

[10] Goth, G.: Entering A parallel Universe ,communications of the acm, (2009), 53(9).

[11] Williams, E.: Software Licensing Metrics - The Challenge in a Multicore Environment, SoftSummit, 2007.

[12] H. P. Hofstee. Power Efficient Processor Architecture and The Cell Processor. HPCA, 00:258–262, 2005.

[13] D. Geer, "For Programmers, Multicore Chips Mean Multiple Challenges", Computer, September 2007.

[14] M. Creeger, "Multicore CPUs for the Masses", QUEUE, September 2005.

[15] T. Holwerda, "Intel: Software Needs to Heed Moore‟s Law", http://www.osnews.com/story/17983/Intel-Software-Needs-to-Heed-Moores-Law/

[16] Jeffery A. "Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures". Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures SPAA, 2007.