

A Review of Architectures - Intel Single Core, Intel Dual Core and AMD Dual Core Processors and the Benefits

A.I. Fasiku¹, J. B. Olawale², O. T. Jinadu³

¹Department of Computer Science,
The Federal University of Technology, Akure, Ondo State, Nigeria.

²Department of Computer Engineering,
Rufus Giwa Polytechnic, Owo, Ondo State, Nigeria.

³Department of Computer Science,
Rufus Giwa Polytechnic, Owo, Ondo State, Nigeria

ABSTRACT

Computer architectures are approaching physical and technological barriers which make increasing the speed of a single core exceedingly difficult and economically infeasible. As a result, hardware architects have begun to design microprocessors with multiple processing cores that operate independently and share the same address space. However, increasing the number of cores on a single chip, generate a lot of challenges that arises with memory and cache coherence as well as communication between the cores. AMD and Intel have found solution to this problem by redesigning the processor architecture differently by putting multiple CPU cores on a chip. This paper describes the architecture of single core and Multi core processor technology

Keywords: *Computer Architectures, Single and Multicore Processors*

1. INTRODUCTION

In 1945, mathematician John von Neumann, with the aid of J. Presper Eckert and John Mauchly, wrote a memo proposing the creation of an Electronic Discrete Variable Automatic Computer, more famously known as the EDVAC. In that paper, von Neumann suggested the stored-program model of computing. In the von Neumann architecture, the program is a sequence of instructions stored sequentially in the computer's memory. The program's instructions are executed one after the other in a linear, single-threaded fashion.

As time went on, advancements in mainframe technology expanded upon the ideas presented by von Neumann. The 1960s saw the advent of time-sharing operating systems. Run on large mainframe computers, these operating systems first introduced the concept of concurrent program execution. Multiple users could access a single mainframe computer simultaneously and submit jobs for processing. The operating system handled the details of allocating Central Processing Units (CPU) time for each individual program. At this time, concurrency existed at the process level, and the job of task switching was left to the systems programmer [1].

The last 30 years have seen the computer industry driven primarily by faster and faster uniprocessors; those days have come to a close. Emerging in their place are microprocessors containing multiple processor cores that are expected to exploit parallelism. Modern Computer Systems are made up of multicore processor e.g Dual Core CPUs. Dual Core is an architecture that refers to a Central Processing Unit with two complete execution cores in a single processor. The two cores, their caches and cache controllers are all built together in a single IC. A Dual Core processor executes two threads by running each thread on a different core. Thus the Dual Core processors improve multithreaded throughput, and delivering the advantages of parallel computing to properly thread mainstream applications [3].

A dual core processor is an integrated circuit with two central processing units. This allows an operating system to multitask, or perform different functions simultaneously, at the hardware level as opposed to the software level. This uses less energy and generates less heat than multitasking on one or even two single-core processors of similar performance. Additionally, central processing units-intensive operations will not overload dual core processors as they can be assigned one core in the background and the other core can handle foreground operations. In order for a computer to utilize both

processor cores, the operating system must support thread-level parallelism (TLP), that is, the ability to send new instructions to the central processing units before receiving the results of previous instructions. Most modern operating systems, such as Linux and Microsoft windows, support TLP. While dual core processors are essentially two processors on a single chip, the advantages of having both processors in such close proximity go far beyond the obvious space savings [8].

II. THREAD LEVEL PARALLELISM

A thread is a flow of control through a program with a single execution point. It is a sequence or stream of executable code within a process that is scheduled for execution by the operating system on a processor or core. All processes have a primary thread; the primary thread is a process's flow of control or thread of execution. A process with multiple threads has as many flows of controls as there are threads. Each thread executes independently and concurrently with its own sequence of instructions. A process with multiple threads is called multithreaded.

Threads execute independent concurrent tasks of a program. Threads can be used to simplify the program structure of an application with inherent concurrency in the same way that functions and procedures make an application's structure simpler by encapsulating functionality. Threads can encapsulate concurrent functionality and use minimal resources shared in the address space of a single process as compared to an application, which uses multiple processes. This contributes to an overall simpler program structure being seen by the operating system. Threads can improve the throughput and performance of the application if used correctly, by utilizing multicore processors concurrently. Each thread is assigned a subtask for which it is responsible, and the thread independently manages the execution of the subtask. Each thread can be assigned a priority reflecting the importance of the subtask it is executing [6].

A. Threads Model Implementation

There are three implementation models for threads:

a) User or application level thread is a process or thread executing instructions in the program or linked library. They are not making any calls to the operating system kernel. They are not visible to the operating system and cannot be scheduled to a processor core. Each thread does not have its own thread context. So, as far as simultaneous execution

of threads, there is only one thread per process that will be running at any given time and only a single processor core allocated to that process. There may be thousands or tens of thousands user - level threads for a single process, but they have no impact on the system resources.

- b) Kernel level thread is the process or thread making system calls such as accessing resources or throwing exceptions. Kernel - level threads reside in kernel space and are kernel objects. With kernel threads, each user thread is mapped to or bound to a kernel thread. The user thread is bound to that kernel thread for the life of the user thread. Once the user thread terminates, both threads leave the system. The operating system scheduler manages, schedules, and dispatches these threads. The runtime library requests a kernel level thread for each of the user level thread. The operating system's memory management and scheduling subsystem must be considered for very large numbers of user - level threads and creates a context for each thread and each of the threads from a process can be assigned to a processor core as the resources become available.
- c) Hybrid threads are the combination of user and kernel level threads. A hybrid thread implementation is a cross between user and kernel threads and allows both the library and the operating system to manage the threads. User threads are managed by the runtime library scheduler, and the kernel threads are managed by the operating system scheduler. With this implementation, a process has its own pool of kernel threads. The user threads that are runnable are dispatched by the runtime library and are marked as available threads ready for execution. The operating system selects a user thread and maps it to one of the available kernel threads in the pool. More than one user thread may be assigned to the same kernel thread.

One of the big differences between these implementations is the mode in which they exist in and the ability of the threads to be assigned to a processor. User and kernel - level threads also become important when determining a thread's scheduling model and contention scope. Contention scope determines which threads a given thread contends with for processor usage, and it also becomes very important in relation to the operating system 's memory management for large numbers of threads.

III. ARCHITECTURE OF SINGLE CORE PROCESSOR

Pentium 4 processor brand, refers to Intel's line of single-core desktop and laptop Central Processing Units (CPUs)

introduced on November 20, 2000 at a speed rate of 1.5GHz and shipped through August 8, 2008. They are the 7th-generation of the X86 micro architecture, called Netburst, this is the company's first new design since the introduction of P6 micro architecture of Pentium pro CPUs in 1995.

The Pentium 4 processor is designed to deliver performance across applications where end users can truly appreciate and experience better performance. For example, it allows a much better user experience in areas such as Internet, audio and streaming video, image processing, video content creation, speech recognition, 3D applications and games, multi-media and multi-tasking user environments. The Pentium 4 processor enables real-time MPEG2 video encoding and near real-time MPEG4 encoding, allowing efficient video editing and video conferencing. It delivers world-class performance on 3D applications and games, such as Quake 3, enabling a new level of realism and visual quality to 3D applications [8].

The Pentium 4 processor has 42 million transistors implemented on Intel's 0.18u CMOS process; with six levels of aluminium interconnect. It has a die size of 217mm² and it consumes 55watts of power at 1.5GHz. It has 3.2GB/second system bus that helps to provide the high data bandwidths needed to supply data to today's and tomorrow's demanding applications. It adds 144 new, 128-bit Single Instruction Multiple Data (SIMD) instructions called SSE2 (Streaming SIMD Extension 2) that improves performance of multi-media, content creation, scientific, and engineering applications.

A. The Netburst Microarchitecture

Pentium 4 is a processor with a deep pipeline supporting multiple issues with speculation. The Pentium 4 uses an aggressive out-of-order speculative microarchitecture, called Netburst, that is deeply pipelined with the goal of achieving high instruction throughput by combining multiple issue and high clock rates, Unlike the microarchitecture used in the Pentium III [7].

This processor requires balancing and tuning of many microarchitectural features that compete for processor die cost and for design and validation efforts. Fig. 1 shows the basic Intel NetBurst microarchitecture of the Pentium 4 processor. There are four main sections in this architectural diagram of Pentium 4:

- a) The in-order front end
- b) The out-of-order execution engine
- c) The integer and floating-point execution units
- d) The memory subsystem microarchitecture of Pentium 4 [4].

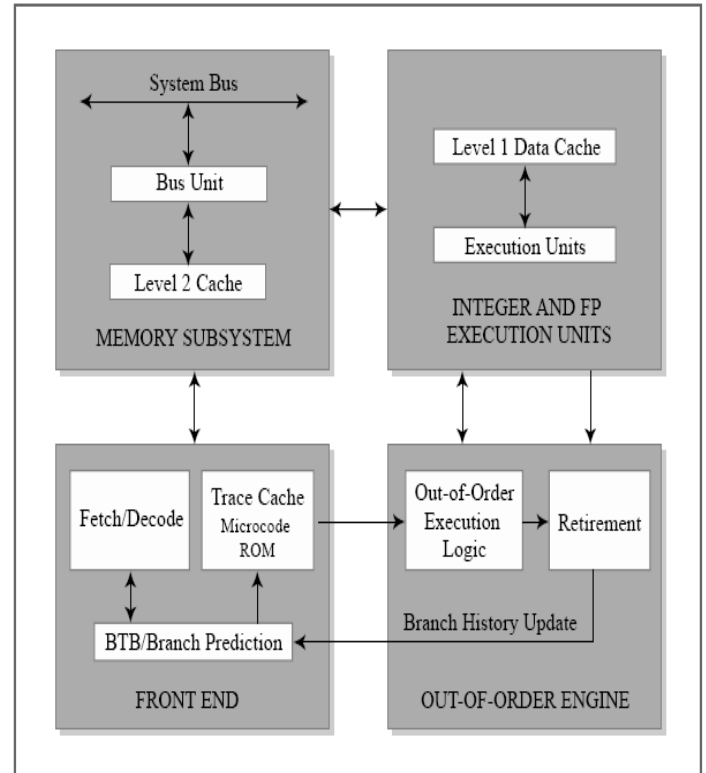


Fig. 1: Basic block diagram of Intel NetBurst

- 1) **In-Order Front End:** The in-order front end is part of the machine that fetches the instructions to be executed next in the program and prepares them to be used later in the machine pipeline. Its job is to supply a high-bandwidth stream of decoded instructions to the out-of-order execution core, which will do the actual completion of the instructions. The front end has highly accurate branch prediction logic that uses the past history of program execution to speculate where the program is going to execute next. The predicted instruction address, from this front-end branch prediction logic, is used to fetch instruction bytes from the Level 2 (L2) cache. These IA-32 instruction bytes are then decoded into basic operations called micro-operations (micro-operations) that the execution core is able to execute [5].

The NetBurst microarchitecture has an advanced Level 1 (L1) instruction cache called the Execution Trace Cache. Unlike conventional instruction caches, the Trace Cache sits between the instructions decode logic and the execution core as shown in Fig. 1. In this location the Trace Cache is able to store the already decoded IA-32 instructions or micro-operations (uops). Storing already decoded instructions removes the IA-32 decoding from the

main execution loop. Typically the instructions are decoded once and placed in the Trace Cache and then used repeatedly from there like a normal instruction cache on previous machines. The IA-32 instruction decoder is used only when the machine misses the Trace Cache and needs to go to the L2 cache to get and decode new IA-32 instruction bytes.

The front-end decoder translates each IA-32 instruction to a series of micro-operations (uops), which are similar to typical RISC instructions. The micro-operations are then executed by a dynamically scheduled speculative pipeline. A trace cache is a type of instruction cache that holds sequences of instructions to be executed including nonadjacent instructions separated by branches; a trace cache tries to exploit the temporal sequencing of instruction execution rather than the spatial locality exploited in a normal cache.

- 2) **Out-of-Order Execution Logic:** The out-of-order execution engine is where the instructions are prepared for execution. The out-of-order execution logic has several buffers that it uses to smooth and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. Instructions are aggressively reordered to allow them to execute as quickly as their input operands are ready. This out-of-order execution allows instructions in the program following delayed instructions to proceed around them as long as they do not depend on those delayed instructions. Out-of-order execution allows the execution resources such as the ALUs and the cache to be kept as busy as possible executing independent instructions that are ready to execute.

The retirement logic is what reorders the instructions, executed in an out-of-order manner, back to the original program order. This retirement logic receives the completion status of the executed instructions from the execution units and processes the results so that the proper architectural state is committed (or retired) according to the program order. The Pentium 4 processor can retire up to three micro-operations per clock cycle. This retirement logic ensures that exceptions occur only if the operation causing the exception is the oldest, non-retired operation in the machine. This logic also reports branch history information to the branch predictors at the front end of the machine so they can train with the latest known-good branch-history information [5].

- 3) **Integer and Floating-Point Execution Units:** The execution units are where the instructions are actually executed. This section includes the register files that store the integer and floating-point data operand values that the instructions need to execute. The

execution units include several types of integer and floating-point execution units that compute the results and also the L1 data cache that is used for most load and store operations.

- 4) **Memory Subsystem:** The memory subsystem showed in Fig. 1, includes the L2 cache and the system bus. The L2 cache stores both instructions and data that cannot fit in the Execution Trace Cache and the L1 data cache. The external system bus is connected to the backside of the second-level cache and is used to access main memory when the L2 cache has a cache miss, and to access the system I/O resources [4].

IV. OVERVIEW OF MULTICORE PROCESSOR

Multi-core processor is the most recent evolution in computing technology. A multi-core processor is composed of two or more independent cores. It can be describe as an integrated circuit which has two or more individual processors called cores. Manufacturers typically integrate the cores into a single integrated circuit die known as a Chip Multiprocessors (CMPs). A dual-core processor contains two cores (Such as AMD Phenom II X2, AMD Turion II P520 Dual-Core, Intel Pentium Dual-Core and Intel Core Duo), a quad-core processor contains four cores (Such as the AMD Phenom II X4 and Intel 2010 core line, which includes 3 levels of quad core processors), and a hexa-core processor contains six cores (Such as the AMD Phenom II X6 or Intel Core i7 Extreme Edition 980X). A multi-core processor implement multiprocessing in a single physical package. Designers may couple cores in a multi-core device together tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared memory inter-core communication methods.

In today's digital world, the demands for complex 3D simulations, streaming media files, added levels of security, more sophisticated user interfaces, larger databases, and more on-line users are beginning to exceed single-core processor capabilities. Multi-core processors enable true multitasking. On single-core systems, multitasking can max out CPU utilization, resulting in decreased performance as operations have to wait to be processed. On multi-core systems, since each core has its own cache, the operating system has sufficient resources to handle most compute intensive tasks in parallel [8].

The most recent advances in microprocessor design for desktop computers involve putting multiple processors on a single computer chip. These multicore designs are completely replacing the traditional single core designs that have been the foundation of desktop computers. IBM,

Sun, Intel, and AMD have all changed their chip pipelines from single core processor production to multicore processor production. This has prompted computer vendors such as Dell, HP, and Apple to change their focus to selling desktop computers with multicores.

Multicore Programming showed that approaches to designing and implementing application software that will take advantage of the multicore processors are radically different from techniques used in single core development. The focus of software design and development will have to change from sequential programming techniques to parallel and multithreaded programming techniques. The standard developer's workstation and the entry-level server are now multiprocessors capable of hardware-level multithreading, multiprocessing and parallel processing. Although sequential programming and single core application development have a place and will remain with us, the ideas of multicore application design and development are now in the mainstream [6].

A. Multicore Architectures

Chip Multiprocessors (CMPs) come in multiple flavors: two processors (dual core), four processors (quad core), and eight processors (octa - core) configurations. Some configurations are multithreaded; some are not. There are several variations in how cache and memory are approached in the new Chip Multiprocessors. The approaches of processor - to - processor communication vary among different implementations. The Chip Multiprocessors implementations from the major chip manufacturers on which each processor handle the Input/Output (I/O) bus and the Front Side Bus (FSB) are different.

Again, most of these differences are not visible when looking strictly at the logical view of an application that is being designed to take advantage of a multicore architecture. Fig. 2 illustrates three common architectural configurations that support multiprocessing [6].

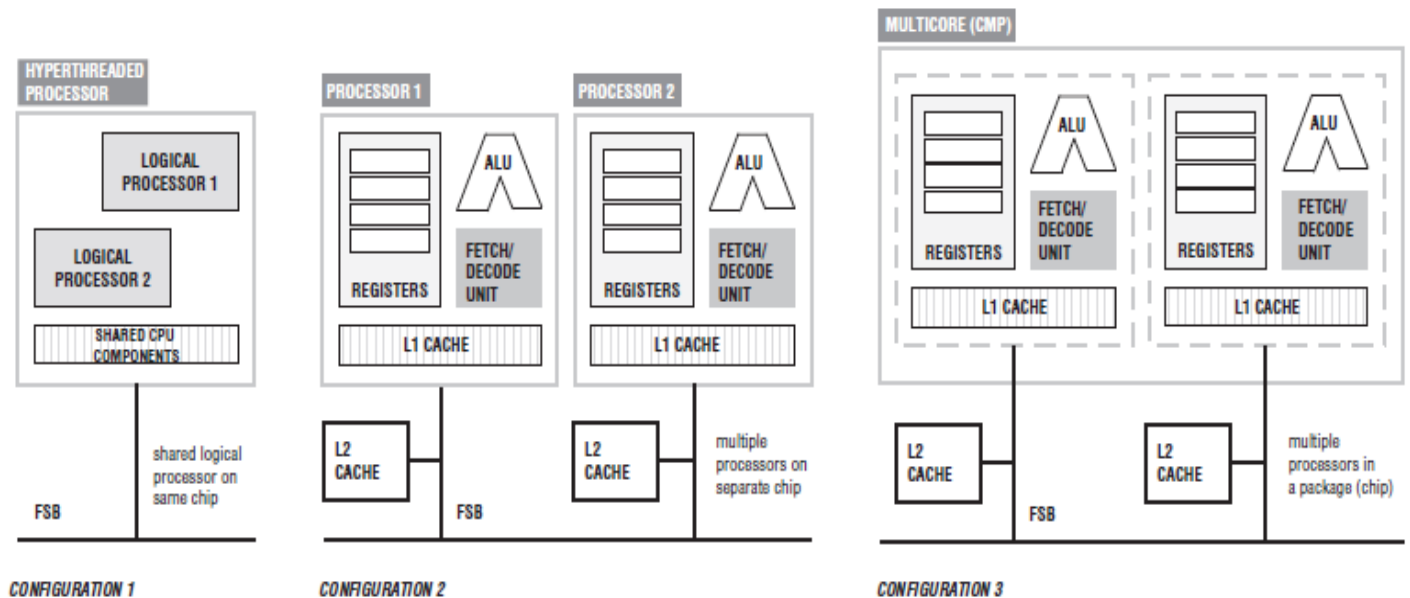


Fig. 2: Architectural Configuration of Multiprocessing

- ❖ Configuration 1 in Fig. 2 uses hyperthreading. Like Chip Multiprocessors, an hyperthreaded processor allows two or more threads to execute on a single chip. However, in a hyperthreaded package the multiple processors are logical instead of physical. There is some duplication of hardware but not enough to qualify a separate physical processor. So hyperthreading allows the processor to present itself to the operating system as

complete multiple processors when in fact there is a single processor running multiple threads.

- ❖ Configuration 2 in Fig. 2 is the classic multiprocessor. In configuration 2, each processor is on a separate chip with its own hardware.
- ❖ Configuration 3 represents the current trend in multiprocessors. It provides complete processors on a single chip.

What important to remember is that each configuration presents itself to the developer as a set of two or more logical processors capable of executing multiple tasks concurrently. The challenge for system programmers, kernel programmers, and application developers is to know when and how to take advantage of it.

V. INTEL CORE 2 DUO PROCESSOR ARCHITECTURE

Intel's Core 2 Duo is one of Intel series of multicore processors. Multicore processors are enhanced with hyperthreading, giving each core two logical processors. The first of Intel's multicore processors was Intel®

Pentium® processor Extreme Edition, in April of 2005. It had dual cores and supported hyperthreading, giving the system eight logical cores. The Core Duo multicore processor was introduced in 2006 and offered not only multiple cores but also multiple cores with lower power consumption and it has no hyperthreading but supports 64 bit architecture. Fig. 4 shows a block diagram of Intel Core 2 Duo. The Core 2 Duo processor has two 64 - bit cores and two 64KB level 1 caches, one for each core. Level 2 cache is shared between cores. Level 2 cache can be up to 4MB. Either core can utilize up to 100 percent of the available L2 cache. This means that when the other core is underutilized and is, therefore, not requiring much L2 cache, the more active core can increase its usage of L2.

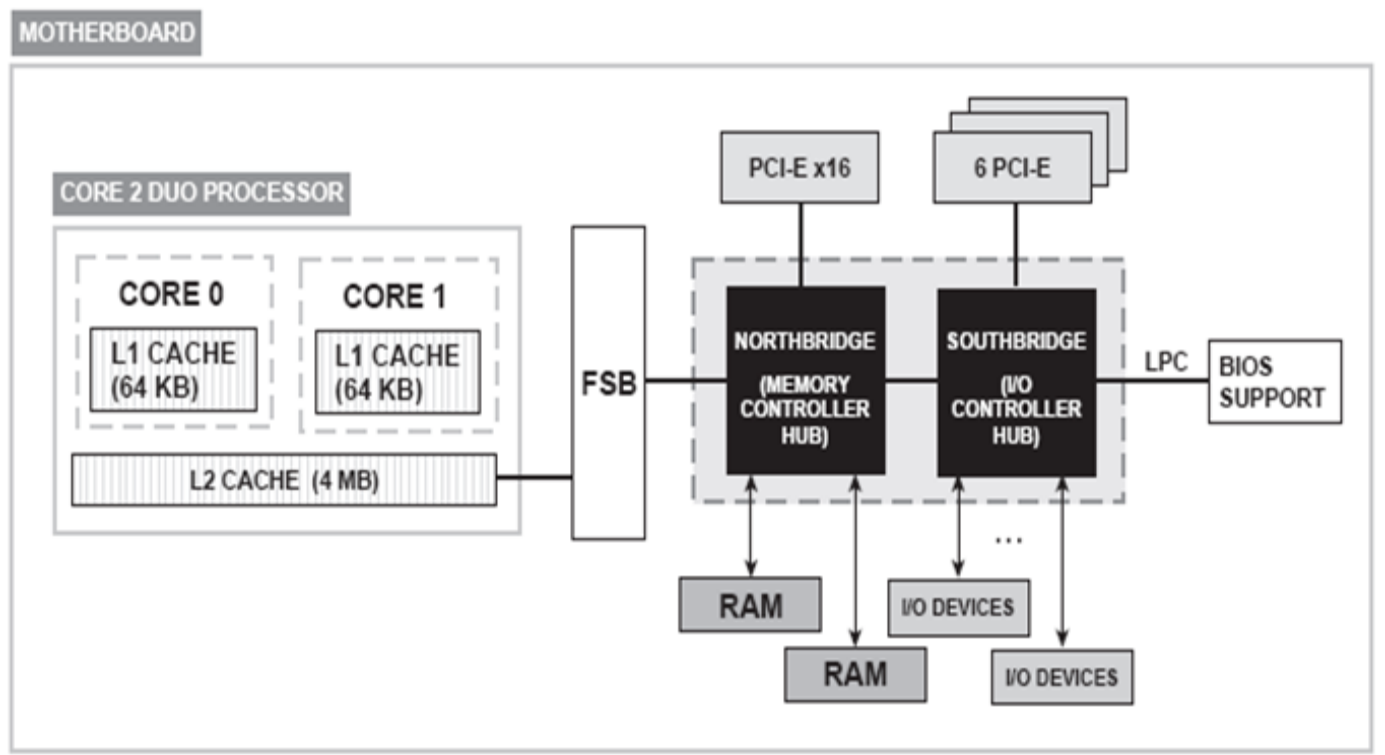


Fig. 3: Block diagram of Intel Core 2 Duo Processor Architecture

1) Northbridge and Southbridge:

Besides the Central Processing Units (CPUs), the next most important component of the motherboard is the chipset. The chipset, shown in Fig. 3, is a group of integrated circuits designed to work together that connects the CPUs to the rest of the components on the motherboard. It is an integrated part of the motherboard and, therefore, cannot be removed or upgraded. It is manufactured to work with a specific class or series of

CPUs in order to optimize its performance and the performance of the system in general. The chipset moves data back and forth from CPU to the various components of the motherboard, including memory, graphics card, and Input/Output (I/O) devices, as showed in Fig. 4. All communication to the CPU is routed through the chipset. The chipset comprises of two chips: Northbridge and Southbridge. These names were adopted based on the locations of the chips on the motherboard and the purposes they serve. The Northbridge is located in the

northern region, north of the components on the motherboard, and the Southbridge is located in the southern region, south of the components on the motherboard. Both serve as bridges or connections between devices; they bridge components to make sure that data goes where it is supposed to go [6].

- ❖ The Northbridge, also called the Memory Controller Hub, communicates directly with the Central Processor via the Front Side Bus (FSB). It connects the CPUs with high - speed devices such as main memory. It also connects the CPUs with Peripheral Component Interconnect Express (PCI - E) slots and the Southbridge via an internal bus. Data is routed through the Northbridge first before it reaches the Southbridge.
- ❖ The Southbridge, also called the Input/Output (I/O) Controller, is slower than the Northbridge. Because it is not directly connected to the CPUs, it is responsible for the slower capabilities of the motherboard like the I/O devices such as audio, disk interfaces, and so on. The Southbridge is connected to BIOS support via the Serial Peripheral Interface (SPI), six PCI - E slots, and other I/O devices not shown on the diagram. The Serial Peripheral Interface enables the exchange of data (1 bit at a time) between the Southbridge and the BIOS support using a master - slave configuration. It also operates with a full duplex, meaning that data can be transferred in both directions.

2) Dual Core Cache:

Cache is memory placed between the processor and main system memory. While cache is not as fast as registers but, it is faster than Random Access Memory (RAM). It holds more than the registers but does not have the capacity of main memory. Cache increases the effective memory transfer rates and the overall processor performance. Cache is used to contain copies of recently used data or instruction by the processor. Small chunks of memory are fetched from main memory and stored in cache in anticipation that they will be needed by the processor. Programs tend to exhibit both temporal locality and spatial locality.

- ❖ Temporal locality is the tendency to reuse recently accessed instructions or data.
- ❖ Spatial locality is the tendency to access instructions or data that are physically close to items that were most recently accessed.

One of the primary functions of cache is to take advantage of this temporal and spatial locality characteristic of a program. Cache is often divided into two levels, they are: level 1 and level 2 (Patterson and Hennessy, 2007).

3) Level 1 Cache:

Level 1 cache is small in size sometimes as small as 16K. L1 cache is usually located inside the processor and is used to capture the most recently used bytes of instruction or data.

4) Level 2 Cache:

Level 2 cache is bigger and slower than L1 cache. Currently, it is stored on the motherboard (outside the processor), but this is slowly changing. L2 cache is currently measured in megabytes. L2 cache can hold an even bigger chunk of the most recently used instruction, data, and items that are in the near vicinity than L1 holds. Because L1 and L2 are faster than general - purpose RAM, the more correct the guesses of what the program is going to do next, the better the overall system performance because the right chunks of data will be located in either L1 or L2 cache. This saves a trip out to either Random Access Memory or Virtual Memory or, even worse, external storage.

A dual core processor design could provide the following for each physical processor:

- ❖ have its own on-die cache,
- ❖ it could provide for the on-die cache to be shared by the two processors,
- ❖ enable each processor could have a portion of on-die cache that is exclusive to a single processor and then have a portion of on-die cache that is shared between the two dual core processors

The two processors in a dual core package could have an on-die communication path between the processors so that putting snoops and requests out on the Front Side Bus (FSB) is not necessary. Both processors must have a communication path to the computer system front side bus.

VI. AMD DUAL CORE PROCESSOR ARCHITECTURE

Advanced Micro Devices (AMD), Inc., released its dual-core Opteron server/workstation processors on 22 April 2005, and its dual-core desktop processors, the Athlon 64 X2 family, was released on 31 May 2005. In terms of

architecture, AMD and Intel have quite different ways of dealing with this issue of multicore systems. Fig. 4 shows very simplified diagram of a dual-core Opteron designed by AMD. Each of the KB cores has its own, independent L2 cache on-board, but the two cores share a common system request queue. They also share a dual channel DDR memory controller and a set of Hyper-Transport links to the outside world. Access to these I/O resources is adjudicated via a crossbar, or switch, so that each CPU can talk directly to memory or I/O as efficiently as possible. In some respects, the dual-core Opteron acts very much like a sort of SMP system on a chip, passing data back and forth between the two cores internally. To the rest of the system I/O infrastructure, though, the dual-core Opteron looks more or less like the single-core version [8].

Each processor (whether dual core or single) has its own local dual channel DDR memory controller, and the processors talk to one another and to I/O chips via point-to-point Hyper-Transport links running at 1GHz. The total possible bandwidth flowing through the 940 pins of an Opteron 875 is 30.4GB/s. With HyperTransport link, the Opteron 275 can theoretically hit 22.4GB/s. AMD uses a cache coherency protocol called MOESI and CPU that owns certain data has the data in its cache, modified it, and yet makes it available to other CPUs. Data flagged in an Opteron cache can be delivered directly from the cache of CPU 0 into the cache of CPU 1 via a CPU-to-CPU HyperTransport link, without having to be written to main memory. This interface runs at the speed of the CPU, so transfers from the cache on core 0 into the cache on core 1 should happen very, very quickly.

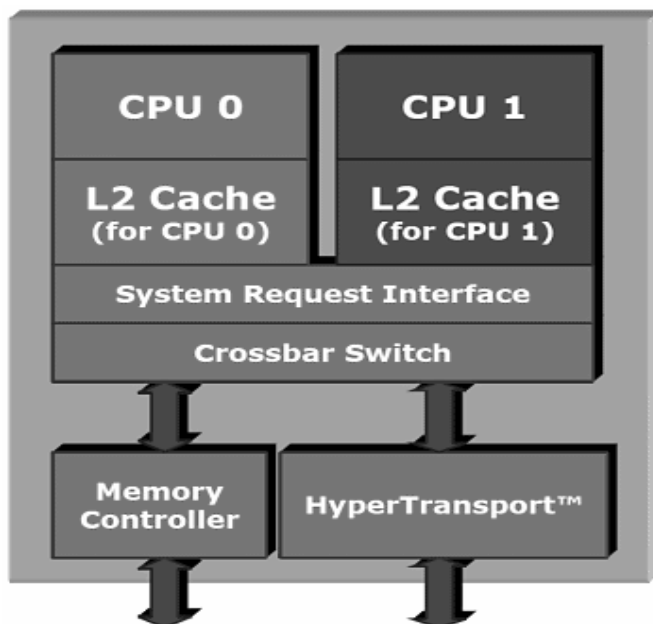


Fig. 4: AMD Dual Core Processor Architecture

This is in stark contrast to the Intel design where Modified, Exclusive, Shared and Invalid (MESI) updates are communicated over the front side bus [2].

VII. CONCLUSION

The requirements for successfully delivering hardware-enhanced threading and multicore processing capability go beyond critical silicon manufacturing capacity and technology. The promise of a better user experience also depends on software as well, unless we develop parallel user level applications, it will be difficult to harness the full power of multi core processor technology.

Multiple cores will provide easy benefits for multithreaded workloads, but many applications written for uniprocessors, will therefore not automatically benefit from CMP designs. The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used. The speedup that can be gained by using a particular feature is the factor of a parallel system as the ratio between the times taken by a single processor to solve a given problem to the time taken by a parallel system (consisting of n processors) to solve the same problem This is expressed in equation 1.1.

$$n = \frac{\text{Execution time of processor } Y}{\text{Execution time of processor } X} \text{ ----- 1.1}$$

It appears that the advent of these multi-core architectures will finally force a radical change in how applications are programmed. Specifically, developers must consider how to direct the collaboration of many concurrent threads of execution to solve a single problem.

REFERENCES

- [1] S. Akhter, and J. Roberts, (2006); "Multi-Core Programming (Increasing Performance through Software Multi-threading)" Published by Richard Bowles, Library of Congress Cataloging in Publication Data, Printed in the United States of America.
- [2] S. Anirban, (2006); Dual Core Processors – A brief overview1, uploaded: April 12th 2006, anirbans@cs.ubc.ca retrieved on April, 2011.
- [3] V. Saxena and M. Shrivastava, (2009); "UML Modeling and Performance Evaluation of Multithreaded Programs on Dual Core Processor", Ambedkar University (Central University), Lucknow, India, published by International Journal

- of Hybrid Information Technology Vol.2, No.3, July, 2009).
- [4] J. Emer, (2005); “Microprocessor Evolution: 4004 to Pentium-4” Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.
- [5] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker and P. Roussel, (2001); “The Microarchitecture of the Pentium® 4 Processor”, Desktop Platforms Group, Intel Corp., Intel Technology Journal Q.
- [6] C. Hughes and T. Hughes, (2008); “Professional Multicore Programming, (Design and Implementation for C++ Developers)” , Published by Wiley Publishing, Inc. 10475 Crosspoint Boulevard, Indianapolis, IN 46256, www.wiley.com
- [7] D.A. Patterson and J.L. Hennessy (2007); Computer Architecture A Quantitative Approach, 4th Edition, Published by Morgan Kaufmann publications, inc. San Francisco California, Printed in the United States of America.
- [8] A. I. Fasiku, (2012); Performance Evaluation of Multicore Processors, M.Tech Thesis, Federal University of Technology, Akure, Nigeria.