International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 4, Issue 4, April 2015

Architecture of Hyper-Threading on Intel Xenon Processor

L.Jyothsna, IV/IV B.Tech, Dept. of Electronics and Computer
Dr. K. Kiran kumar, Professor, Dept. of Electronics and Computer

Abstract— Over the years, the computer architects were continuously devising new hard ware technologies to improve the system performance and to compete with the advancements in software. Simultaneous Multithreading Technology (SMT) was one of the techniques where multiple threads can be simultaneously executed by a single processor without doing a context switch. Hyper-Threading, the Intel's way of providing Simultaneous Multithreading Technology, adds a virtual thread along the side of a physical thread to increase the processing power of a processor. This paper combines the concepts of architecture of hyper-threading and its implementation on Intel's xenon processor. The experimentation is done on Intel's enterprise product line with wide variety of products.

Keywords—Hyperthreading, Simultaneous Multithreading Technology, Virtual thread, physical thread, Intel's xenon processor.

I. INTRODUCTION

The astonishing technology of the Internet and telecommunications made vast tremendous changes in various fields. Ever-speed systems demands for high level processors which runs with the technology. So to process the high speed processors, we should not only depend entirely on traditional approaches but also with the new technology approaches. Micro-architecture techniques are used to attain target of processor performance. The improvement of super pipelining, out of order execution and caches performance leads the microprocessors more complex. These changes in the past architecture raise the number of transistors and usage of power than speeding up the processor performance. Therefore Processor architects are searching for improvements to overcome the above mentioned problems. Intel Xenon processor Hyper-threading technology is one of the solution to this problem.

Considering the past design approaches, the architects mainly focused on instruction level parallelism and clock speeds and caches performance. Pipelining technology is used to overcome the high speed clock in design of processor architecture. This performance is increased based on the number of instructions. That is when we increase the number of instructions the clock frequencies increased and executed each second. Due to far more instructions in a super pipelining, micro architecture which manages the events that wrap the pipelining such as cache misses and interrupts.

A. Hyper threading Technology Architecture:

Normally a CPU can execute one instruction at a time by maintaining a single instruction pointer . And the processor can execute multiple threads by switching between the threads. In Simultaneous Multithreading Technology (SMT), a single processor could simultaneously execute multiple threads without context switching [7]. Intel called SMT as Hyper-Threading (HT) technology [8].

From the architectural point of view, a hyperthreaded processor contains two logical processors each with its own processor architectural state which consists of all the processor registers like data, segment, control, debug and other model specific registers [9,10].Each processor separately has an independent instruction stream and programmable interface controller. They share the system bus and cache memory. [7,11,12]. Fig.1 shows processor architecture (a) without and (b) with Hyper-Threading respectively [13].

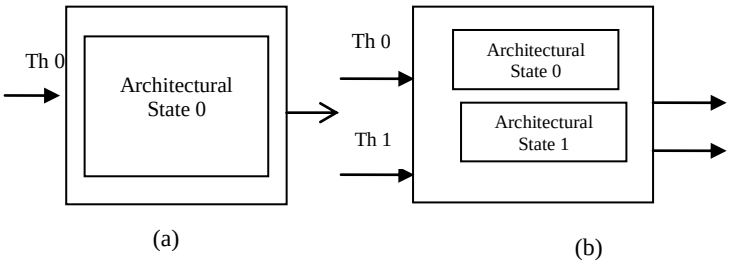


Fig 1. (a) non hyperthreaded vs (b) hyperthreaded processor architecture

B. Implementation on Intel's xenon processor:

There are many objectives to design the architecture of Intel Xeon processor MP implementation of Hyper-Threading Technology. First objective is to reduce the die area cost of hyper-threading implementation in Intel xenon processor because the logical processors partitioned large area and

ISSN: 2278 – 909X

resources and also the small structures were duplicated. As we implemented for the first time measuring the die area cost, the result at that time was less than 5% of total area.

The second objective is to run the other logical processors when the previous processor was stopped running. If one logical processor is executing and got obstructed due to various reasons then other logical processors should continue executing. The logical processors stopped due to many reasons such as servicing Trace cache misses, handling branch mispredictions, or waiting for the results of previous instructions. These stops execution temporarily and these interrupts can overcome by managing buffering queues, so that any of logical processor cannot occupy the entire space when the sequential of two threads are executing. Finally we have to allocate maximum entries required to each thread, so that there will be no problem of interruption.

A third objective is by using hyper-threading technology, when the thread is active then it should allow to run the active thread at the same speed when used hyper-threading. That is the shared resources are combined when the active thread is executing.

C. Architecture in front-view

The front end of the pipeline architecture which performs operations in Intel xenon processor is processed to transport the instructions for further pipeline phases.

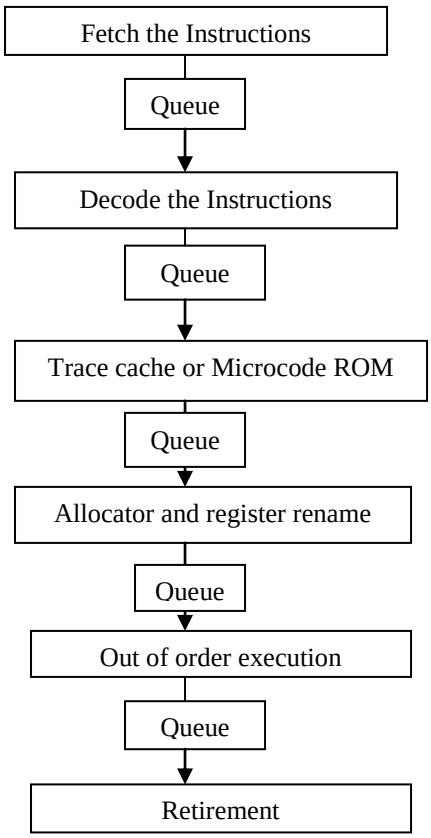


Fig.2: Front view of hyper-threading on Intel Xenon processor

In this process there are two levels which are defined as execution trace cache and integrated trace cache.
Firstly instructions are delivered from primary level which is known as execution trace cache. That is the level 1 instruction cache fetches the instructions for execution purpose.

The secondary level is used when there is trace cache miss then the instructions are delivered and decoded from integrated trace cache. That is the level 2 integrated trace cache consists of the missed instruction which are in trace cache. This decoded the instructions are stored in microcode ROM, which can therefore handle IA-32 instructions and prolonged instructions.

Level 1 Instruction Cache:

As described above, most of the instructions in the program which was given by user is processed and executed from trace cache. This TC reserves the decoded instructions in the form of micro-operations or can be called as "uops". The main purpose of instruction-pointers is to monitor the advancement of processing threads. When the two software threads are in execution process then the next set of instruction-pointer individually instruct the process of thread execution. The access can be provided by logical processors to each thread for every clock cycle. When two logical processors want to access the trace cache at the same time then there arises a problem. The problem can be solved by judging the access to TC for each cycle. Now consider the problem that both logical processors request for TC, then one logical processor get access to TC during first cycle. One cycle fetches the instructions to TC and next cycle is allotted to the other logical processor. If the logical processor which is executing the instruction of respective thread is interrupted or unable to fetch TC then it is stopped and the other logical processor would use the entire range of TC. When one logical processor needs more entries then TC would allocate to that particular processor than other processors.

Level-2 integrated trace cache:

If the complex IA-32 instructions are accomplished then these can be handled by Microcode ROM. As the access of logical processor to TC is same as the processors access to microcode ROM. The microcode ROM entries are shared by both logical processors. The complex instructions are processed by creating microcode instruction pointer. The TC transfer the microcode instruction pointer to microcode ROM. The microcode ROM controller encodes the uops that are required for process and return the acknowledgement to TC. These microcode instruction pointers are used to control the instructions individually respective to the logical processors.

Speaking about IA-32 instructions, they are complex enough to decode the instructions due to memory size and various requirements. Different instructions specify different

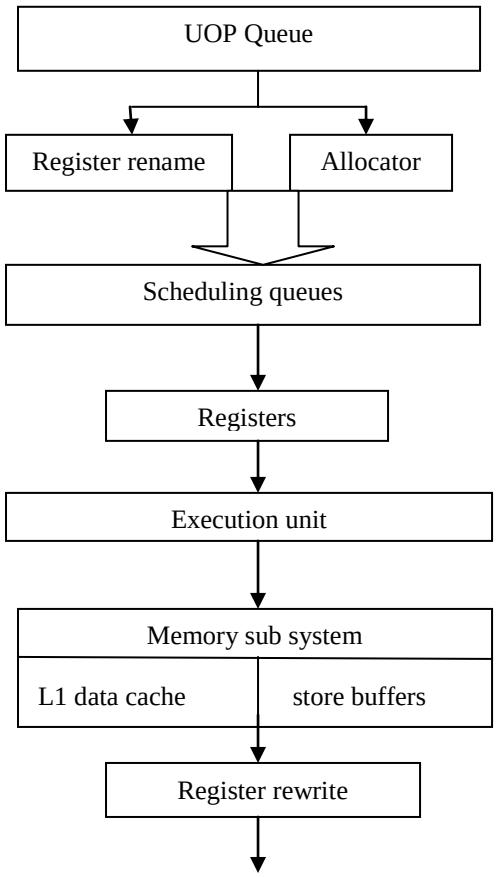
ISSN: 2278 – 909X

size of bytes and various logical operations. To decode these type of instructions we require a middleware state with sufficient amount of logic. The missed instructions in trace cache are decoded into uops by taking them from streaming buffers. When the decoder logic is executing by both the threads at a time, the streaming buffers are interspersed between the threads, so that they can share the same decoder logic. Although decoder logic is same for both the logical processor but it executes only the IA-32 instruction for that logical processor. Firstly, it decodes most of the instructions that belong to one logical processor and then switch to next logical processor. The switching between processors depend on the byte size and it is used to minimize the complicated problem. These all decoded instructions are send to uop queue from TC.

The instructions that are fetched from microcode ROM are send to decoder logic at first and then from decoder logic the decoded instructions are further processed to uop queue. The uop queue is used to dissociate the front-end from the back-end execution process in the architecture. Uop queue stores the half of entries of each and every logical processor.

D. Back-end of the Execution

In the back-end of the execution is used to reorder the instructions and execute them on the basis of inputs. If the inputs are ready, it executes without considering the order of program. It consists of register rename, register write, allocator, queue scheduling and so on.



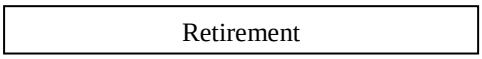


Fig.3: Back-end view of hyper-threading on Intel Xenon

In the rename process the allocator logic is processed parallel to the register rename. The register rename purpose is to rename the IA-32 registers on to the physical registers. By using these 8 IA-32 registers are flexible to use already existing physical registers. This renaming register instructs the next instructions where to obtain the inputs, so that it can update with all the latest architectural registers. For this purpose it creates register alias table(RAT). RAT maintains the architectural state of every logical processor. This process is executed along with the allocator which is the part of register rename.

In back-end execution there are many buffers which performs several operations and trace the instructions. From uop queue, the uops are assigned to allocator and then allocator allocates these uops to physical buffers. These buffers execute each uop which specify the 126 reorder buffer entries,128 integer and float point buffers, 48 load buffers and 24 store buffer entries. In these buffer entries, each logical processor use maximum of 63 re-order, 24 load and 12 store buffer entries. As in the TC clock cycle assignment, the allocator also assigns the resources by uops from the logical processors for every clock cycle when both the logical processors contains uops in uop queue. If the logical processor has used its required number of resources then the allocator stops the logical processor and then give chance to other logical processor to assign the resources such as load buffer entries or store buffer entries. By using this allocator process of resource usage, we can prevent from occurring deadlocks and have simple architectural states. The register rename and the allocator logic works on the same uops for resource allocation. After the completion of uops to allocate and rename they are sent to queues which are of two sets. One set of queue is used store store and load operations which is named as memory instruction queue and other for all type of remaining operations which is named as general instruction queue.

The above mentioned instruction queues are very fast to send the uops to scheduler queues. The scheduler queue is used to execute uops on the basis of input register operand and the availability of execution units. For each clock cycle these uops are alternatively sent between two logical processors. Five uop schedulers are used to schedule different uops to execution units. For each clock cycle they can be executed to maximum of six uops. As mentioned above the deadlock problem occurs when the two logical processors sends the uops during same clock cycle. For example three uops are send by one logical processor and two uops are send by other logical processor at one clock cycle then it can be solved by the scheduler queue. Each schedule has its own scheduler queue and thus queue has limited number of entries and the limit depends on size of the queue.

Execution units are the basic need for the uops which first access the physical registers to execute the instructions

ISSN: 2278 – 909X

and then send back the results to the register file. The physical register consists of all the renamed registers of source and destination registers in a registered pool. After execution units process the uops are sent to reorder buffer where the reorder buffer separate the execution unit stage from retirement stage. This separation advantages the logical processor to use half of the entries in reorder logic buffer.

Next step is level1 data cache. It is located in the memory sub system which includes level2 unified cache and level 3 unified cache. Level3 unified cache is present in only Intel Xenon processor. The memory sub system manages the uops in order that comes from the scheduler queue where it is unaware to logical processor. the access to memory sub system also has nothing to do with logical processors. The L1 data cache is also named as write through cache. It is used to write the instruction that are executing in registers. These writes are send to L2 unified cache. The L1 data cache structure is defined as the four-way set link with the 64byte lines. The data cache is addressed effectively and labelled physically. The L2 and L3 unified cache structure is double the L1 cache. L2 and L3 cache consists of eight-way set link with 128 byte lines. These both caches are addressed physically. The two logical processors already insert the data in the cache and these 3 level caches can use all entries. As there is resource sharing in the cache, all logical processors share the data without regard of uops.

The final stage for back-end process is retirement. The main purpose of retirement logic is to monitor the uops and execute the uops from logical processor when they are ready for retirement. Uop retirement logic perform the operation in program order for architecture state. First the retirement logic will retire the uops that belongs to one logical processor and then go to other logical processor. If one logical processor does not give uops readily to retire then the retirement logic allocates all the memory size to other logical processor. after the retire of uops of one logical processor then those are sent back to the L1 data cache which in turn commit the data in cache by the selection logic.

E. Operating system and applications

The logical processors that is executed using hyper threading technology in a operating system. In the hyper threading technology system the application software have twice the number of logical processors than it usually have. Operating system operates the logical processors in the way of scheduling tasks and thread execution to logical processor. the operating system should execute two functionalities which is considered as best performance. One functionality is about when one logical processor is agile and other logical processor is idle then we have to use HALT instruction. HALT instruction allows the processor to transmit from single task logical processor mode 0 or single task logical processor mode 1. If the operating system does not use this functionality, then it execute on idle logical processor which search repeatedly for instructions to do operations.

The second functionality is programming software threads to logical processors. The best result can be observed after the operating system programmes the multiple threads to same physical processor, then programme the threads to logical processor on various physical processors. This functionality allows the software threads to use various resources in the execution unit whenever it is needed.

II. EXPERIMENTAL RESULTS

The Intel® Xeon™ processor family transports the highest server system performance of any IA-32 Intel architecture processor till date. Using hyper-threading, the opening standard tests display up to 65 percent growth on high-end server applications when compared to Initial benchmark tests shows up to a 65 percent performance increase on high-end server applications when compared to Pentium® III Xeon™ processor on 4-way server platforms.

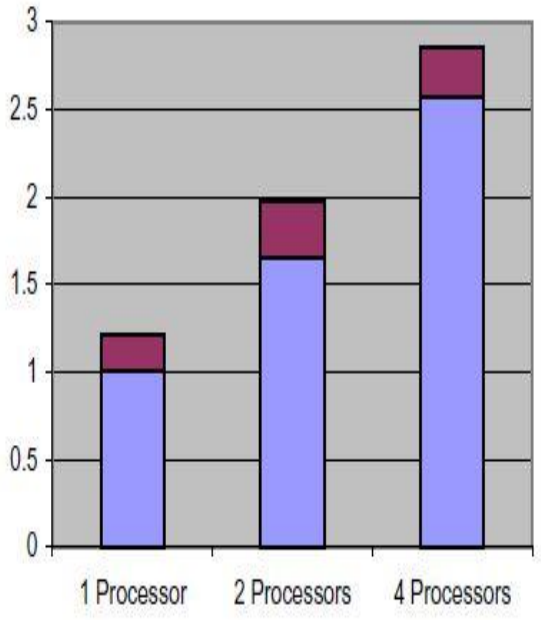


Fig.4: OLTP functioning on Intel Xenon processor

The figure shows the performance of online transaction processing on the basis of using number of processors. The hyper threading technology enable from single processor result to the 4 processors result. As shown in the figure the result concludes the growth attributable to hyper-threading technology, 21 percent in the case of single and dual processor systems.

ISSN: 2278 – 909X

III. CONCLUSIONS

Experiments were conducted to see the effect of hyper-threading on Intel Xenon processor architecture. Our experiments proved that the process executed faster on hyper threaded processor and obtained a benefit of 30% approximately.

REFERENCES

[1] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In: Proceedings of the 20th International conference on very large data bases (VLDB'94), Santiago, Chile 1215: 487–499

[2] Han, J. and Kamber, M. (2006). Data mining: concepts and techniques, Second Edition Morgan Kaufmann.

[3] Tevanian, A., Rashid, R. F., Young, M., Golub, D. B., Thompson, M. R., Bolosky, W. J. and Sanzi, R. (1987). A UNIX Interface for Shared Memory and Memory Mapped Files Under Mach. In USENIX Summer：53-68.

[4] Krieger, O., Reid, K., and Stumm, M. (1995). Exploiting Mapped Files for Parallel I/O. In SPDP Workshop on Modeling and Specification of I/O: 1-11

[5] Rao, S. T., Prasad, E. V., and Venkateswarlu, N. (2010). A critical performance study of memory mapping on multi-core processors: An experiment with k-means algorithm with large data mining data sets. International Journal of Computer Applications 1(9): 90-98.

[6] Robert Love. (2007). Linux System Programming, 2nd Edition O'Reilly Media, Inc

[7] Tullsen, D. M., Eggers, S. J. and Levy, H. M. (1995). Simultaneous Multithreading: Maximizing On-Chip Parallelism. In: Proc. of International Symposium on Computer Architecture : 392-403.

[8] Deborah, T. M., Frank, B., David, L. H., Glenn, H., David A. K., Miller, J. A. and Michael, U. (2002). Hyper-Threading Technology Architecture and Microarchitecture. Intel Technology Journal 6, I issue (1) :1-12.

[9] Tian, X., Chen, Y. K., Girkar, M., Ge, S., Lienhart, R., and Shah, S. (2003). Exploring the use of Hyper-Threading technology for multimedia applications with Intel® OpenMP compiler. In:Proceedings of the IEEE International Symposium on Parallel and Distributed Processing:1- 8

[10] Martinez, T. and Parikh, S. (2005). Understanding dual processors, hyper-threading technology, and multi-core systems. Intel Optimizing Center © Intel Corporation. 1-5

[11] Intel (2003) Intel® Hyper-Threading Technology Technical user's guide Copyright©IntelCorporation
www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf

[12] Akhter, S., and Roberts, J. (2006). Multi-core programming : 33. Intel Press.

[13] Drysdale, G., Valles, A. C., and Gillespie, M. (2009). Performance insights to Intel R Hyper-Threading technology. Intel® Software Network.

[14] Bach, M. J. (2004). The design of Unix operating system. PHI publications, New Delhi 110 001, India.

[15] Keringhan, B. W. and.Ritchie, D. M. (2004). "C Programming Language", 2nd edition, PHI publications, New Delhi, India.

[16] Venkateswarlu, N.B. (2005). Advanced UNIX Programming, BS publications, Hyderabad

[17] Vahalia.U, (1996). UNIX Internals The New Frontiers, Pearson education, New Delhi 110 017, India.

[18] MathWorks—Overview of Memory-Mapping http://www.mathworks.in/help/matlab/import_export/overview-of-memory-mapping.html [Accessed on February 18 2014]

[19] Heidemann, J. (1997). Performance interactions between P-HTTP and TCP implementations. ACM SIGCOMM Computer Communication Review 27 (2): 65-73

[20] Bodon, F. (2005). A trie-based APRIORI implementation for mining frequent item sequences. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, ACM : 56-65.

[21] Geurts, K., Wets, G., Brijs, T. and Vanhoof, K. (2003). Profiling High Frequency Accident Locations Using Association Rules. Electronic Proceedings of the 82th Annual Meeting of the Transportation Research Board, Washington, January 12-16, USA :18.

[22] Anuradha,T., Satya Prasad, R. and Tirumalarao, S. N. (2012 a). Parallelizing Apriori on Dual Core using OpenMP. International Journal of Computer Applications 43 (24): 33-39.

[23] Anuradha, T, Satya Prasad, R. and Tirumala Rao, S.N. (2013). Performance evaluation of apriori with memory mapped files. International Journal of Computer Science Issues 10, Issue 1(1) :162-169.

[24] Anuradha, T and Satya Prasad, R. S.N. (2013). Parallelizing Apriori on Hyper-Threaded Multi-Core Processor. International Journal of Advanced Research in Computer Science and Software Engineering 3(6) :1072-1082.

ISSN: 2278 – 909X