Effective Execution on Multicore Processor Platform

Priyanka Saxena[1] Aman Ansari[2]
[1, 2]Niet, Nims University, Jaipur

Abstract—*In the last few years we have seen the rise of the Chip Multiprocessor (CMP). With clock speeds staying static but feature sizes still shrinking exploiting thread level parallelism (TLP) along with instruction level parallelism (ILP) is the natural way to gain further performance. Microprocessors have revolutionized the world we live in and continuous efforts are being made to manufacture not only faster chips but also smarter ones. A number of techniques such as data level parallelism, instruction level parallelism and hyper threading (Intel's HT) already exists which have dramatically improved the performance of microprocessor cores [1, 2]. This paper briefs on evolution of multi-core processors followed by introducing the technology and its advantages in today's world. In this paper we also propose Tomb, an architecture designed for multi-core systems. Tomb has two major aims: (i) make on-chip communication explicit to the programmer so they can optimize for it and (ii) support many threads and supply very lightweight communication and synchronization primitives for them. These aims are based on the observations that: (i) as feature sizes shrink, on-chip communication becomes relatively more expensive than computation and (ii) as we go increasingly multi-core we need highly scalable approaches to inter-thread communication and synchronization. The paper concludes by detailing on the challenges currently faced by multi-core processors and how the industry is trying to address these issues.*

I. Introduction

Driven by a performance hungry market, microprocessors have always been designed keeping performance and cost in mind. Gordon Moore, founder of Intel Corporation predicted that the number of transistors on a chip will double once in every 18 months to meet this ever growing demand which is popularly known as Moore's Law in the semiconductor industry [3,4]. Advanced chip fabrication technology alongside with integrated circuit processing technology offers increasing integration density which has made it possible to integrate one billion transistors on a chip to improve performance [5, 6]. However, the performance increase by micro-architecture governed by Pollack's rule is roughly proportional to square root of increase in complexity [7]. This would mean that doubling the logic on a processor core would only improve the performance by 40%. With advanced chip fabrication techniques comes along another major bottleneck, power dissipation issue. Studies have shown that transistor leakage current increases as the chip size shrinks further and further which increases static power dissipation to large values as shown in Figure 1 [7, 8] .One alternate means of improving performance is to increase the frequency of operation which enables faster execution of programs [3,9]. However the frequency is again limited to 4GHz currently as any increase beyond this frequency increases power dissipation again [18]. "Battery life and system cost constraints drive the design team to consider power over performance in such a scenario" [10]. Power consumption has increased to such high levels that traditional air-cooled microprocessor server boxes may require budgets for liquid-cooling or refrigeration hardware [10]. Designers eventually hit what is referred to as the power wall, the limit on the amount of power a microprocessor could dissipate [11].

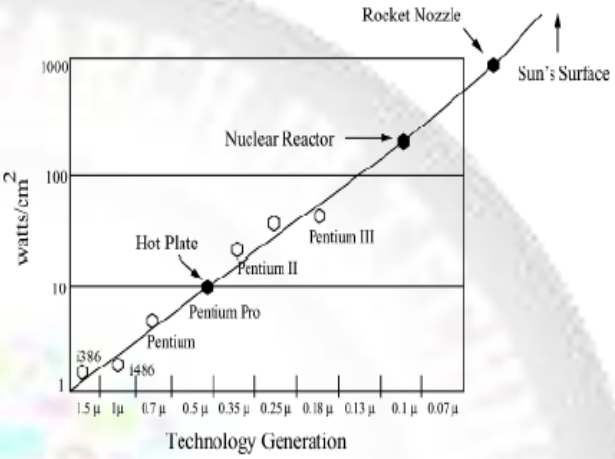

Fig. 1: Power Density Rising

II. Multicore Processors

"A Multi-core processor is typically a single processor which contains several cores on a chip" [2]. The cores are functional units made up of computation units and caches [12]. These multiple cores on a single chip combine to replicate the performance of a single faster processor. The individual cores on a multi-core processor don't necessarily run as fast as the highest performing single-core processors, but they improve overall performance by handling more tasks in parallel [13].The multiple cores inside the chip are not clocked at a higher frequency, but instead their capability to execute programs in parallel is what ultimately contributes to the overall performance making them more energy efficient and low power cores as shown in the figure below[14] . Multi-core processors are generally designed partitioned so that the unused cores can be powered down or powered up as and when needed by the application contributing to overall power dissipation savings.

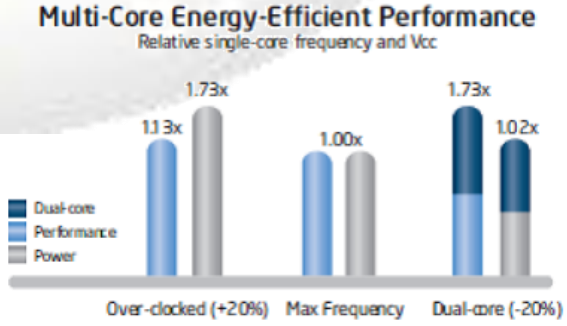

Fig. 3: Dual core processor at 20% reduced clock frequency effectively delivers 73% more performance while

approximately using the same power as a single-core processor at maximum frequency [14].

Given a constant die size with shrinking feature size, the chip area reachable in a single cycle also shrinks. This complicates the design of a single large core that uses the increasing availability of gates to extract more ILP. Using that area to instead have a multitude of smaller, simpler cores, increasing core number, rather than core complexity, is preferable and the costlier global cross-chip communications can be made architecturally explicit, allowing software to optimize for it [7,8].

Stream programming language targets to achieve two goals at the same time - efficiency and convinience in writing a streaming application. Since communication overhead dominates over that of computation in a streaming program, a stream processor exposes the communication explicitly into the upper layer so that software handles it to maximize performance.

There are some examples of stream programming languages that have been developed and/or are being developed.

- StreamC/KernelC A language specific to a single machine - Imagine at Stanford
- StreaMIT A language intended for the MIT RAW machine but not machine specific
- Brook 2 generation language based on the Imagine concept of streams, but machine independent

III. Major Challenges Faced By Multi-Core Processors

In spite of the many advantages that multi-core processors come with, there are a few major challenges the technology is facing. One main issue seen is with regard to software programs which run slower on multi-core processors when compared to single core processors. It has been correctly pointed out that "Applications on multi-core systems don't get faster automatically as cores are increased" [9]. Programmers must write applications that exploit the increasing number of processors in a multi-core environment without stretching the time needed to develop software [11]. Majority of applications used today were written to run on only a single processor, failing to use the capability of multi-core processors [4]. Although software firms can develop software programs capable of utilizing the multi-core processor to the fullest, the grave challenge the industry faces is how to port legacy software programs developed years ago to multi-core aware software programs [15]. Redesigning programs although sounds possible, it's really not a technological decision in today's environment. It's more of a business decision where in companies have to decide whether to go ahead redesigning software programs keeping in mind key parameters such as time to market, customer satisfaction and cost reduction[16] .

The industry is addressing this problem by designing compilers which can port legacy single core software programs to 'multi-core aware' programs which will be capable of utilizing the power of multi-core processors. The compilers could perform "code reordering", where in compilers will generate code, reordering instructions such that instructions that can be executed in parallel are close to each other [10]. This would enable us to execute instructions in parallel improving performance. Also compilers are being developed to generate parallel threads or processes automatically for a given application so that these processes can be executed in parallel [10]. Intel released major updates for C++ and Fortran tools which aimed at programmers exploiting parallelism in multi-core processors. Also alongside OpenMP (Open Multiprocessing), an application programming interface which supports multiprocessing programming in C, C++ and Fortran provides directives for efficient multithreaded codes [15]. It has however been correctly pointed out that "The throughput, energy efficiency and multitasking performance of multi-core processors will all be fully realized when application code is multi-core ready" [6].

Secondly, on-chip interconnects are becoming a critical bottle-neck in meeting performance of multi-core chips [17]. With increasing number of cores comes along the huge interconnect delays (wire delays) when data has to be moved across the multi-core chip from memories in particular [1]. The performance of the processor truly depends on how fast a CPU can fetch data rather than how fast it can operate on it to avoid data starvation scenario [16]. Buffering and smarter integration of memory and processors are a few classic techniques which have attempted to address this issue. Network on a chip (NoCs) are IPs (Intellectual Property) being developed and researched upon which are capable of routing data on a SoC in a much more efficient manner ensuring less interconnect delay [15].

Increased design complexity due to possible race conditions as the number of cores increase in a multi-core environment. "Multiple threads accessing shared data simultaneously may lead to a timing dependent

error known as data race condition" [13]. In a multi-core environment data structure is open to access to all other cores when one core is updating it. In the event of a secondary core accessing data even before the first core finishes updating the memory, the secondary core faults in some manner. Race conditions are especially difficult to debug and cannot be detected by inspecting the code, because they occur randomly. Special hardware requirement implementing mutually exclusion techniques have to be implemented for avoiding race conditions [16].

Another important feature which impacts multi-core performance is the interaction between on chip components viz. cores, memory controllers and shared components viz. cache and memories [18] where bus contention and latency are the key areas of concern. Special crossbars or mesh techniques have been implemented on hardware to address this issue [9].

IV. Proposed Tomb Architecture

A system designed to encourage the creation of programs that will scale over a large number of cores without the need to tailor a program to a specific number of cores. This is accomplished by provided a simple, lightweight synchronization mechanism in the form of presence bits combined with a hardware scheduler.

A Tomb system consists of a network of nodes. Each node contains a simple in-order RISC core, a cache and a network-on-chip (NoC) router. Each node is allocated an area of global physical address space which is termed that node's local address space; any other addresses are in a

remote address space. When a node's core accesses local address space it goes to the local cache. Remote address space is accessed from remote cache via the network. Provided the network retains ordering of messages between nodes we can ensure sequential consistency within a particular local address space.

Every 64-bit memory word in a Tomb system has an associated presence-bit (also known as a full/empty bit). This is used as a basic primitive for thread to thread communication and synchronization. A thread attempting to read a non-present location will be stalled and descheduled until that location becomes present. Each register also has an associated presence bit, when a thread attempts to use a non-present register it stalls until the register becomes present. Presence bits are stored in main memory at the top of each node's local address space, this leaves a gap in the address space that cannot be used for data storage, however this could be hidden by a virtual memory system.

A node supports hardware threading and scheduling. The register file (RF) in a node's core can hold the registers of eight separate threads. Every cycle it will issue an instruction from a different thread, scheduling the threads within the register file in a round robin manner. A thread is represented in memory by an activation frame (AF), this is simply the contents of the thread's registers, its program counter (PC) and a status word. It fits in a 256-byte (32 x 64-bit words) block.

There is a special store instruction (store doubleword to AF, SDA) that causes a node to check the presence bits of the AF being stored to and if they are all set it places the AFon a queue of ready nodes. The node has a simple round-robin scheduler that switches contexts from the RF back into memory and switches an AF from the ready queue from memory into the RF. A context switch occurs when an active thread's quantum expires. Each node has three separate caches. An instruction cache, a data cache and a presence bit cache. The instruction cache does not obey the restriction that a particular address can only live in a particular cache, so code may be replicated across instruction caches. A seperate presence bit cache is used as it allows checking of an entire AFs worth of presence bits without looking at multiple cache lines (which would be the case if presence bits were stored along with words in the data cache).

When a core executes a load or a store instruction it generates a memory request, this will be sent directly to the local cache or out to the network depending upon the address. Upon executing a load a core will clear the presence bit of the register which is the load's destination, so if a thread attempts to use that register before the load has completed it will stall.

There are two major request types, load and store:

Load (Figure A.3) has an address (A) to load from and a return address (R). The return address is where the response from the load should be sent. As each thread is represented by an AF, each register has a memory address so the return address is the address of the register that was the load's destination. When a load request is received the presence bit of the corresponding word is checked, if it is:

Present — An immediate data response is sent to the return address with the contents of the word.

Non Present — The contents of the word are checked, if it is a sentinel value, the return address of the load is written into the word and nothing else is done. If the sentinel value is not there (The sentinel value is a particular invalid return address), then some other load request has already written its return address into this word and an exception response is immediately sent

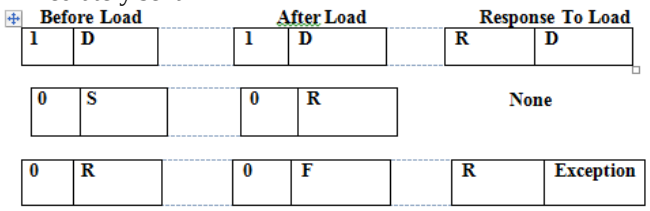

Figure A.3: Detail of LD and LDNR operation. D is the data stored at the word being loaded, S is the sentinel value, R is the return address of the load and F is an already existing forwarding address. 0 refers to a non-present word, 1 refers to a present word

Store (Figure A.4) has an address (A) to store to and the data to store (D′). When a store request is received the presence bit of the corresponding word is checked, if it is:

Present — The contents of the word are overwritten with the new data, nothing else is done.

Non Present — The presence bit is set and the current word contents are checked. If the sentinel value is there, nothing further is done. Otherwise a load return address is there and a load response with the store data is generated to that address.

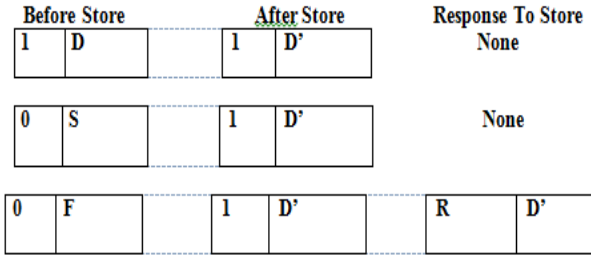

Figure A.4: Detail of store operation. D is the data stored at the word being loaded, D′ is the new data being stored, S is the sentinel value, R is a forwarding address. 0 refersto a non-present word, 1 refers to a present word

Upon receiving a data response, the AF portion of the address is checked. If the AF is in the RF then the corresponding register is updated with the data from the response and the register's presence bit is set. If the AF has been swapped out to memory then the corresponding word within the AF is updated and its presence bit set. The presence bits of all words within the AF are checked and if they are all set the AF is placed at the back of the ready queue.

The mechanism above allows many communication styles. For simple producer single consumer communication the presence bit of a particular word can be cleared, at some point a thread (the consumer) loads from that word. When the thread tries to use the result of that load it will stall (as the word was non-present so no response is received and thus the register holding the result is still marked as non-present). At a later point another thread (the producer) will write to the non present word causing a data

response to be sent to the consumer's AF. Either the consumer will still be in the register file so upon receiving the response can begin execution again immediately or the scheduler will have swapped it out in which case the data response will cause all words within the AF to be present so the consumer will be placed at the back of the ready queue.

V. CONCLUSIONS

Power and frequency limitations observed on single core implementations have paved the gateway for multi-core technology and will be the trend in the industry moving forward. However the complete performance throughput can be realized only when the challenges multi-core processors facing today are fully addressed. A lot of technological breakthroughs are expected in this area of technology including a new multi-core programming language, software to port legacy software to "multi-core aware" software programs. Although it has been one of the most challenging technologies to adopt to, there is considerable amount of research going on in the field to utilize multi-core processors more efficiently.

This paper also presented Tomb, architecture that

- Gives explicit control over communication, so software can be optimized to reduce it
- Provides a light-weight threading and synchronization model.

REFERENCES

[1] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," Micro, IEEE, vol. 27, pp. 51-61, 2007.

[2] Ami Marowka, "A Study of the Usability of Multicore Threading Tools", International Journal of Software Engineering and Its Applications, Vol. 4, No.3, 2010

[3] J. Yan and W. Zhang. "Hybrid multicore architecture for boosting single-threaded performance", In ACM SIGARCH Computer Architecture News, Vol 35, No.1, pp. 141-148, March, 2007.

[4] Dempsey, Paul. "Monsters, Incapacitated [Multicore Processors]." Engineering & Technology (17509637) 4.16 (2009): 38-41. [9] Cass, S. (2010). Multicore Processors Create Software Headaches. Technology Review, 113(3), 74-75.

[5] Lizhe Wang, Jie Tao, Gregor von Laszewski, Holger Marten. 2010, "Multicores in Cloud Computing: Research Challenges for Applications", Journal of Computers, Vol 5, No 6 (2010)

[6] Lance Hammond, Basem A. Nayfeh, KunleOlukotun, "A Single-Chip Multiprocessor," Computer, vol. 30, no. 9, pp. 79-85, Sept. 1997

[7] S. Borkar. "Thousand core chips – A technology perspective".DAC'07: Proc. of the 44th annual Conf. on Design Automation, pages 746–749, 2007

[8] Roy, A., Jingye Xu &Chowdhury, M.H. 2008, "Multi-core processors: A new way forward and challenges", Microelectronics, 2008. ICM 2008. International Conference on, pp. 454.

[9] AssafShacham, Keren Bergman, Senior Member, and Luca P. Carloni. Photonic Networks-On-Chip for Future Generations of Chip Multiprocessors. *IEEE Trans. Computing*, page 1260, 2008.

[10] Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J., Zyuban, V., Gupta, M. & Cook, P.W. 2000, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors", Micro, IEEE, vol. 20, no. 6, pp. 26-44.

[11] Patterson, D. 2010, "The trouble with multi-core", Spectrum, IEEE, vol. 47, no. 7, pp. 28-32, 53.

[12] Blake, G., Dreslinski, R.G. &Mudge, T. 2009, "A survey of multicore processors", Signal Processing Magazine, IEEE, vol. 26, no. 6, pp. 26-37.

[13] Lowney, G. "Why Intel is designing multi-core processors". In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 113-113, 2006

[14] R. Ramanathan. Intel multi-core processors: Making the move to quad-core and beyond. Technology@Intel Magazine, Dec 2006.

[15] Xinmin Tian, M. Girkar, S. Shah, D. Armstrong, E. Su and P. Petersen, "Compiler and runtime support for running OpenMP programs on pentium- and itanium-architectures," in High-Level Parallel Programming Models andSupportive Environments, 2003. Proceedings. Eighth International Workshop on, 2003, pp. 47-55.

[16] Moore, S.K. 2008, "Multicore is bad news for supercomputers", Spectrum, IEEE, vol. 45, no. 11, pp. 15-15.

[17] Jongman Kim, Dongkook Park, Theocharides, T., Vijaykrishnan, N. & Das, C.R. 2005, "A low latency router supporting adaptivity for on-chip interconnects", Design Automation Conference, 2005. Proceedings. 42nd, pp. 559

[18] Schaller, R.R. 1997, "Moore's law: past, present and future", Spectrum, IEEE, vol. 34, no. 6, pp. 52-59.