Microprocessors in the Era of Terascale Integration

Shekhar Borkar
Intel Corporation
2111 NE 25th Ave,
Hillsboro, OR, 97124
Shekhar.Y.Borkar@intel.com

Norman P. Jouppi
Hewlett Packard
1501 Page Mill Road
Palo Alto, CA 90304
norm.jouppi@hp.com

Per Stenstrom
Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Goteborg, Sweden
pers@ce.chalmers.se

Abstract

Moore's Law will soon deliver tera-scale level transistor integration capacity. Power, variability, reliability, aging, and testing will pose as barriers and challenges to harness this integration capacity. Advances in microarchitecture and programming systems discussed in this paper are potential solutions.

1. Introduction

Microprocessors have evolved all the way from the early 4004 with a few thousand transistors to today's high performance microprocessors with hundreds of millions to billions of transistors. Microprocessor clock frequency has risen too, from kilo-hertz in the 70's to multi giga-hertz today. Increases in both transistor count and circuit frequency have yielded the more than five orders of magnitude increase in compute performance which we enjoy today. Moore's Law is here to stay, and in the future we expect a capacity of trillions of transistors integrated on a single die. Of course, this will not be easy, and there will be challenges that we will face in all disciplines. We will discuss these challenges, opportunities, and potential solutions in microarchitecture and programming systems.

2 Technology Trends

The technology scaling treadmill will continue to follow Moore's Law, providing integration capacity of billions, even trillions of transistors, improving transistor performance, providing an abundance of interconnections to realize complex architectures, and reducing transistor integration cost by half every generation. However, power, variability and reliability will be the barriers to take advantage of technology scaling.

2.1 Power and Energy

Die size, chip yields, and design productivity used to limit transistor integration in a microprocessor design. Now the focus has shifted to energy consumption, power dissipation and power delivery. As a transistor scales down in dimension, supply voltage must scale accordingly to keep electric field constant [5]. Supply voltage scaling has been rewarding since it reduces active power quadratically, allowing us to use the transistor budget to employ complex microprocessor architectures to deliver performance. But the threshold voltage of the transistor too has to scale to deliver circuit performance, and as the threshold voltage scales, the transistor sub-threshold leakage increase exponentially. Today, this sub-threshold leakage power amounts to almost 40% of the total power of a microprocessor, and the threshold voltage scaling will have to stop or slow down. Although leakage avoidance, leakage tolerance, and leakage control circuit techniques have been devised and employed, they are not enough; threshold voltage scaling will either stop or slow down considerably.

2.2 Reliability

As technology scales further we will face even new challenges, such as variability, single event upsets (soft errors), and device (transistor performance) degradation – these effects manifesting as inherent unreliability of the components, posing design and test challenges.

Single event upsets (soft-errors) could be a source of concern for present and future high performance microprocessors [9]. These errors are caused by alpha particles and more importantly cosmic rays (neutrons), hitting silicon chips, creating charge on the nodes to flip a memory cell or a logic latch. These errors are transient and random. It is relatively easy to detect these errors in memories by protecting them with parity, and correcting these errors in memory is also relatively straight forward with error correcting codes. However, if such a single event upset occurs in logic state,

978-3-9810801-2-4/DATE07 © 2007 EDAA

such as a flip-flop, then it is difficult to detect and correct. Although soft-error rate per logic state bit may not increase much each generation, the logic state bits in a microprocessor will double each technology generation (following Moore's Law). The aggregate effect on soft-error rate FIT (failure in time) of a microprocessor could be almost two orders of magnitude higher in the Tera-scale era.

2.3 Variability

Random dopant fluctuations cause variability in the threshold voltages of transistors, resulting from the discreteness of the dopant atoms in the channel of a transistor [25]. As transistor area scales, the number of dopant atoms in the channel reduce exponentially. As a result, two transistors sitting side by side will have different electrical characteristics due to randomness in small number of dopant atoms, resulting in variability. We see the impact of this effect even today in static memory (cache) stability.

Sub-wavelength lithography will continue until EUV (Extreme Ultra-Violet) technology becomes available, resulting in line edge roughness, and thus variability in transistors. Increasing power density increases heat flux, resulting in more demand on the power distribution system causing voltage variations across a microprocessor die. It also causes hot-spots on the die with increased subthreshold leakage power consumption. We are facing static (process technology) and dynamic (circuit operation) variability today and it could get worse.

Designs must comprehend variability from day one as part of the design problem. Today's design methodology optimizes performance and power, but test and yield in the presence of variability is missing–what we need is a multivariate design optimization capability for probabilistic and statistical design. Circuit design techniques such as body biasing will help, but the effect of body bias diminishes with technology scaling.

2.4 Aging

Aging has had significant impact on transistor performance. Studies have shown that transistor saturation current degrades over years due to oxide wear-out and hot carrier degradation effects. So far the degradation is small enough such that it can be accounted for as an upfront testing margin in the specification of a component. We expect this degradation to become worse as we continue to scale transistor geometries. It may become so bad that it would be impractical to absorb degradation effects upfront in the test margin, and you may have to deal with it at the system level throughout the lifetime of the microprocessor.

2.5 Test and Burn-in

Gate dielectric (gate oxide) thickness scales down too, as the transistor size scales, to improve transistor performance and to keep short-channel effects under control. But the gate leakage current (and power) increases exponentially due to tunneling. This leakage power is especially worse at high voltages during burn-in, and we fear that burn-in power will become prohibitive, making burn-in testing obsolete.

High-K alternative is on the horizon, where gate oxide will get replaced by a material with high dielectric constant to provide the same capacitance as silicon dioxide, but with much higher thickness. Since this dielectric is thicker it will have less leakage. However, this dielectric thickness too will have to scale down over time and ultimately will face the same gate leakage problem.

Then how do you screen for defects and infant mortalities in microprocessor chips? One time factory testing will be insufficient, and what you need is the test hardware embedded in the design, to dynamically detect errors, isolate and confine the faults, reconfigure using spare hardware, and recover on the fly.

2.6 Putting it all together

All these problems are not new; even today we design systems to comprehend variability and reliability issues. For example, error correcting codes are commonly used in memories to detect and correct soft errors. Careful design, and testing for frequency binning, copes with variability in transistor performance. What is new is that as technology scaling continues, the impact of these issues keeps increasing, and we need to devise techniques to deal with them effectively.

Technology will continue to scale towards Tera-scale integration capacity. Supply voltage scaling will slow down, may even stop, due to sub-threshold leakage and lack of gate oxide scaling. Voltage scaling was a powerful lever supplied by the process technology to employ more transistors towards complex microarchitecture, and it will no longer be available. Microarchitectures and programming systems will have to carefully manage the power with innovations in their respective disciplines, and with fine grain power management to continue to deliver unprecedented compute performance, but not at any cost – it has to fit in the power envelope and has to be reliable. Variability and reliability could get worse, which will need an interdisciplinary solution, with proper trade-offs, from process technology, design, microarchitecture, programming systems, and applications.

3 Architectural Trends

Changes in base technology create three key problems for architects to solve: the power wall, the memory wall, and system reliability wall. These problems are best alleviated with an interdisciplinary approach, with architects working in partnership with technologists and software developers.

3.1 The Power Wall

The most important "wall" is the power wall. What can architecture do to reduce increasing power dissipation? There are several techniques that have been proposed and used recently, but more work needs to be done.

Heterogeneous multiprocessing (also called asymmetric multiprocessing) is one architectural technique that can significantly reduce power dissipation[18]. These heterogeneous cores can run different instruction set architectures, or run the same instruction set architecture (i.e., single ISA) for ease of programming. The key reason why heterogeneous processing can help is that for a given technology smaller cores are more power efficient than larger cores. By using a smaller core, power efficiency gains of 2–4X can be obtained, in some cases with very little loss in performance. The Cell processor[11] is a recent example of a heterogeneous chip multiprocessor. However, smaller cores can have lower performance for some applications or phases of applications. Thus it is important when using heterogeneous chip multiprocessors to achieve a good assignment of programs and phases of programs to cores, in order to maximize power savings and minimize performance loss.

An interesting application of heterogeneous multiprocessors is in mitigating Amdahl's law by running serial portions of an algorithm on a large, fast and relatively power-inefficient core while executing parallel portions of the algorithm on multiple small power-efficient cores[2]. This can simultaneously provide both a program speedup and power reduction!

Even though smaller cores are more power efficient, paradoxically the use of smaller cores can result in higher chip power densities. This is because a large complex core usually makes extensive use of clock gating. Hence many (but not all) of the unused units in a large core may not dissipate much dynamic switching power. Of course the wires connecting units in a large core will still be longer and require more power to be driven. And units that are not clocked still dissipate leakage power, which is fast increasing to become on the same order as dynamic switching power. In contrast, many smaller cores will fit in the die area of a larger core. Because they are smaller, a higher percentage of transistors of each core switch to execute an instruction. Thus a large number of smaller cores can have many more switching transistors each cycle than a single larger core. Thus while the computation may be more power efficient, the overall die power will still likely be large.

Luckily cache memories dissipate relatively little power in comparison to processor cores of the same area. However, in chip multiprocessors with large numbers of processors, it will be important to keep data in caches close by the cores accessing the data, in order to reduce the power required to move cache lines across the die. Non-uniform cache architectures (NUCA) are an interesting approach to this problem[17]. In the scientific computing domain, the memory hierarchy of stream processors can also improve performance while being power efficient[15].

3.2 The Memory Wall

The "memory wall" is a well-known problem in computer architecture that has only been getting worse with time and scaling[10]. The access time of main memory in computing systems has only been improving very slowly, while the speed of microprocessors has been increasing dramatically for decades. This gap has increased from a factor of two in the late 1970's to a factor of over one hundred today.

During the last decade, microprocessors were increasing in performance roughly as the square root of the number of transistors available plus another factor proportional to lithography from device speed improvements[14]. Cores were only speeding up as the square root of the number of transistors because many structures used in the core design were getting larger, and these yielded less-than linear performance improvements with increasing size. Examples of these structures include on-chip caches, instruction issue widths, and pipeline depths. Quadrupling a cache size often results in a halving of the miss rate, not a reduction to one quarter. Quarupling the issue rate from 1 to 4 instructions per cycle or doubling the pipeline depth from 8 to 32 stages is also more likely to result in only a doubling in performance rather than quadrupling. All these are examples where the performance obtained from more transistors deployed in a single core results in only an improvement in performance by $\sqrt{N}$ for the use of N more transistors.

In contrast, the multicore era is returning to simpler cores which are more power efficient, but doubling the number of cores at each generation. In this scenario, the "computational bandwidth" of a multicore system scales linearly with the number of transistors, not as their square root. This will put even more pressure on the memory wall.

What can be done to reduce the gap between memory and processor speeds? Two promising techniques have recently been proposed: processor and memory die stacking and optical interconnects.

Die stacking can be used to mate one or more high-density DRAM chips with processor chips[16], and has

been included in recent ITRS roadmaps[12]. Although stacking of logic circuits has also been proposed, stacking one logic layer with multiple memory layers has multiple advantages. First, the memory chips dissipate much less power than the processor core. By having just one logic chip in direct contact with a heat sink and other lower power layers stacked underneath it, the temperature of the die stack is minimized. Also, microprocessors and DRAMs are manufactured with different technologies. Integrating diverse technologies is expensive and usually compromises each technology. By stacking a die made with a high performance microprocessor process and other die made with a DRAM process, the benefits of both high-density DRAM and high-performance logic can be obtained without the difficulties, expense, and compromises of process integration.

In the longer term, optical interconnects may play a role in mitigating the memory wall. Although optical interconnects have the potential for very high bandwidths, their latency is within integer multiples of electrical communication. If integrated optics[7] is successful and can be scaled, the bandwidths between processor chips and memory chips or other processor chips may be dramatically increased. However extensive caching, presumably using 3D die stacking, would still be required to reduce the average latency to access memory. In cases where caching is not as effective, multithreading would be useful as well.

3.3 The Reliability Wall

Of the three "walls", the reliability wall is the furthest out and probably the one most easily addressed. The basic issue is that as more and more components are integrated on chip with lithographic scaling, each component is more likely to suffer from soft and hard errors due to their extremely small feature sizes. Hence, at some point conventionally-designed chips will have a failure rate that is on average much more frequent than the expected service lifetime of the device. Currently chips are designed with some redundancy and fault tolerance (e.g., ECC) in memory structures, but assume that logic is all good and remains good over time. These assumptions will need to change in the future.

Thankfully there is a long history of systems designed for high-availability applications, such as NonStop systems[4]. There is also much recent research at providing increased reliability without the overheads of triple-modular-redundant (TMR) or dual-modular-redundant (DMR) architectures[3, 22]. Initially these more recent approaches that require less overhead should be sufficient for most applications, but as scaling continues classical DMR or TMR techniques may be required. Even so, two interesting areas for research remain.

First, flexible approaches that allow "availability on demand" would be interesting in a number of applications, especially given the convergence of hardware platforms to use of a small number of core designs. For example, reliability and availability might be more important for a bank accounting application while performance could be more important for a gamer. Future chip multiprocessors should be able to support a range of availabilities given a single core design.

Second, trends in chip multiprocessors are moving in the direction of increased sharing. For example, some second-generation multicore processors share a level of cache while the first generation had separate caches. As more and more hardware becomes shared, maintaining fault isolation, a key component of reliable and high availability systems, becomes more difficult. Techniques for maintaining isolation in the presence of sharing are also an interesting area of research.

4. Programming Trends

Moore's Law will continue to predict a doubling of the transistor count every eighteen months for the next 10 years. At the same time, the two main sources of performance growth of processing performance – frequency scaling and exploitation of instruction-level parallelism – are offering diminishing returns on investments. Instead, industry decided a few years back to launch multi-core roadmaps that in ten years from now will lead to the integration of as many as hundreds of cores, where each core is capable of issuing a handful of instructions from one or many threads.

This route appears promising under the premise that a doubling of cores yields a doubling of performance. Unfortunately, this is hard to achieve in general. Most existing software is single-threaded. Although research in parallelizing compilers has been an ongoing activity since the 70s, limitations in static analysis and/or lack of information at compile-time make it difficult to ascertain that independent threads are indeed independent. As a result, only a small fraction of codes can be automatically parallelized.

4.1 Automatic Approaches

More recently, there have been lots of efforts to extend the scope of codes that can be parallelized automatically. Thread-level Speculation(see, e.g., [24, 8, 26, 23, 19]) (TLS) attempts to postpone dependency checks to runtime by providing a hardware/software substrate that checks memory dependences on-the-fly. While this approach has broadened the scope of codes that can be automatically parallelized, the amount of thread-level parallelism (TLP) that can be exposed through TLS is in general quite limited. For example, in a recent study based on the SPEC 2000, Prabhu and Olukotun [21] found that even after significant analysis and code modification, it was not possible to uncover a

sufficient amount of TLP to keep more than a handful cores busy.

Research is certainly warranted in how to more efficiently combine static and dynamic techniques to uncover TLP. However, it is not realistic to believe that a fully automatic approach will uncover TLP at the scale that will be needed to keep hundreds of cores busy as we foresee in the next decade. Fortunately, as multi-core microprocessors will become main-stream, there will be less reluctance to manually convert codes to uncover TLP. However, to cut down on development time, *high-productivity programming models* together with tools infrastructures to reduce the manual efforts to uncover thread-level parallelism are needed more than ever.

4.2 Manual Approaches

Research into parallel processing took off some three decades back as a pro-active means for the case that performance growth of conventional architectural approaches would cease. It is interesting to note that single-core performance growth has just recently slowed down. In the creative era of parallel processing in the 70s and 80s, programming model/language research was a key focus. Several programming language models and there suitability as to extract TLP were scrutinized heavily. It is fair to say that time is ripe to lift up the experiences from that time to question whether predominant programming languages as of today are the right vehicles for uncovering thread-level parallelism at the scale needed. It may very well be the right time for a paradigm shift in programming languages with a focus on high-productivity parallel program design. Now, what are the hurdles in parallel program design?

Parallel programming is intrinsically difficult. The task of converting a single-threaded program into a highly-efficient parallel code involves the following steps [6]:

1. Decomposition
2. Assignment
3. Orchestration

In the decomposition step, concurrent units-of-execution or *tasks* must be identified and exposed. In a loop nest, for example, the goal is to expose as many independent threads as possible – a daunting task to say the least. In the assignment step, tasks must be bundled to trade *task management overhead* and *inter-processor communication (memory locality)* against *load-balance*. This step is again non-trivial and alerts the programmer/compiler to make informed decisions based on intricate performance properties of the underlying architecture and the computation itself. In the orchestration step, synchronization among tasks must be set up to respect dependences between task. The whole analysis involved is complex and error-prone.

Architecture research in the 80s and 90s with a focus on cache coherency took us a long way to shield programmers from low-level architectural memory locality issues and also demonstrated that cache coherent multiprocessors can indeed scale to hundreds of processors(see, e.g., [20]. Needless to say, that research forms a good base for scalability issues for multi-cores. So it is fair to say that much of the difficulties involved in the assignment phase were removed. Also the discovery of self-scheduling policies greatly simplified load balancing issues. However, the more difficult problem of dealing with dependency issues remained unsolved.

4.3 Semi-Automatic Approaches

Recently, research into transactional memory [1] has gained a lot of attention. At its core, transactional memory enables the programmer to not have to deal with dependences between tasks – this is taken care of by the implementation of the architecture. If two tasks are dependent, they will be forced to execute after each other to respect the dependences. Transactional memory consequently simplifies the decomposition as well as the orchestration steps. However, this is traded against inefficiencies due to resolving dependences at run-time, much like TLS, and manifests itself as performance losses due to re-execution of tasks. As a result, while transactional memory may take us some strides on the way, there are still many hurdles left to bring down the programming efforts.

4.4 Architectural Opportunities

It is clear that there is yet no suitable architectural model that can drastically simplify parallel programming. At the same time, time seems ripe to seriously scrutinize the architectural interface to provide a productive abstraction for parallel software developments. With the emergence of multi-core architectures come quite different operating conditions for making a breakthrough in new programming abstractions.

First, the comparably low communication latencies on-chip as compared to the quite significant inter-processor latencies in multiprocessors a decade back will open up for lower performance-sensitivity of the on-chip memory hierarchy. Second, there is an opportunity to use the enormous transistor resources to build new structures that can remove the efforts to design parallel programs. For example, load balancing remains a problem. It is conceivable to support efficient task management structures on-chip that off-load programmers/compilers for elaborate load-balancing trade-offs.

While we are standing in front of a gigantic programming dilemma to unleash the computational power from multi-cores, there appears to be many fruitful research directions. One direction is to understand how the hardware/software interface can be enriched with functional primitives to off-load programmers/compilers from some of the challenges involved in parallel programming. Second, another direction is to look into the big bag of language concepts that were researched in the 70s and 80s and possibly enrich todays prevailing programming languages with them. Third, while not untouched ground, research into better programming tools infrastructures is certainly warranted. Overall, it is more important than ever to cross-fertilize across computing disciplines to accelerate progress. At the end, parallel processing appears to be our only hope.

5 Concluding Remarks

The era of tera-scale integration is quickly approaching, delivering an abundance of transistors. Power, variability, reliability, aging, and testing will pose technological challenges. This paper has discussed some promising solutions in microarchitecture and programming systems to deliver unprecedented compute performance in the era of tera-scale integration.

References

[1] On-line bibliography on transactional memory. http://www.cs.wisc.edu/trans-memory/biblio/.

[2] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proc. of the International Symposium on Computer Architecture*, pages 298–309, June 2005.

[3] T. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of the 32nd Annual IEEE/ACM Symposium on Microarchitecture (Micro-32)*, pages 196–207, November 1999.

[4] D. Bernick et al. NonStop Advanced Architecture. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 12–21, June 2005.

[5] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, July-August 1999.

[6] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishing Inc., 1998.

[7] C. Gunn. CMOS Photonics for High-Speed Interconnects. *IEEE Micro*, pages 58–66, March–April 2006.

[8] L. Hammond, M. Willey, and K. Olukotun. Data Speculation Support for a Chip Multiprocessor. In *Proc. of the Eigth ACM Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[9] P. Hazucha. *Neutron Soft Error Rate Measurements in a 90-nm*. CMOS Press.

[10] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 4th edition, 2006.

[11] P. Hofstee. Power Efficient Processor Architecture and the Cell Processor. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, pages 258–262, February 2005.

[12] ITRS. International Technology Roadmap for Semiconductors. http://public.itrs.net.

[13] N. Jouppi. The Future Evolution of High-Performance Microprocessors. In *Proceedings of 38th Annual IEEE/ACM Symposium on Microarchitecture (Micro-38)*, page 155, November 2005. http://pcsostres.ac.upc.edu/micro38/01_keynote2.pdf.

[14] U. Kapasi et al. Programmable Stream Processors. *IEEE Computer*, pages 54–62, August 2003.

[15] T. Kgil et al. PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 117–128, Oct. 2006.

[16] C. Kim, D. Burger, and S. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 211–222, Oct. 2002.

[17] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan. Heterogeneous Chip Multiprocessors. *IEEE Computer*, November 2006.

[18] P. Marcuello, A. Gonzalez, and J. Tubella. Speculative Multithreaded Processors. In *Proc. Of ACM International Conference on Supercomputing*, pages 77–84, July 1998.

[19] V. Milutinovic and P. Stenstrom. Opportunities and Challenges for Distributed Shared-Memory Multiprocessors. *Proceedings of the IEEE*, 87(3):399–404, March 1999.

[20] M. Prabhu and K. Olukotun. Exposing Speculative Thread Parallelism in SPEC2000. In *Proceedings of the 2005 Principles and Practices of Parallel Programming*, June 2005.

[21] S. Reinhardt and S. Mukherjee. Transient Fault Detection via Simultaneous Redundant Multitreading. In *Proc. of the International Symposium on Computer Architecture*, pages 25–36, June 2000.

[22] P. Rundberg and P. Stenstrom. All-Software Thread-Level Data Dependence Speculation System for Multiprocessors . *Journal of Instruction-Level Parallelism*, 3, October 2002.

[23] G. Steffan and T. Mowry. The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization. In *Proc. of Fourth IEEE Int. Symp. on High-Performance Computer Architecture*, pages 2–13, February 1998.

[24] Xinghai et al. Intrinsic MOSFET parameter fluctuations due to random dopant placement. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, December 1997.

[25] Y. Zhang, L. Rauchwerger, and J. Torrellas. Hardware for Speculative Run-Time Parallelization in Distributed Shared-Memory Multiprocessors. In *Proc. of Fourth IEEE Int. Symp. on High-Performance Computer Architecture*, February 1998.