Multicore Processors: Status Quo and Future Directions

Mohamed Zahran
Computer Science Department
New York University
New York, NY 10012
Email: mzahran@cs.nyu.edu

***Abstract*—Words like multicore, manycore, Moore's law ending, have been around for more than a decade. How do these words describe the current status quo of computer architecture? How do they give a glimpse of the future? In this paper, we will present the status quo of the current multicore/manycore processors, and the expected future directions in light of several advances both in process technology and in system software.**

I. Introduction

Computers are designed to solve problems. This makes *correctness* the initial and main goal. This has been the case from the dawn of computer systems when computers were scarce and used only in niche applications and at very few organizations. As more sophisticated problems arise, another criteria emerges: speed (or *performance*). With higher performance we can solve bigger and more sophisticated problems, or solve the same problems in shorter time. To achieve this higher performance, computer architects need to make use of the increasing number of transistors given to us by Moore's law [1]. Doing so was successful for a while till architects faced a big physical constraints: *power* [2].

For big machines, such as data centers and supercomputers, power consumption and dissipation are translated into huge electricity bill and cooling cost. For portable devices, like tablets, smartphones, laptops, etc, power consumption and dissipation mean packaging cost and battery life. This brings power-constraint as a primary goal as opposed to being just a secondary concern. Power and performance continue to be the two main goal, for a while, until another constraint arises.

As transistors are getting smaller and smaller and we reach the sub-micron era, transistors become less reliable(i.e. can be switched on/off adversely) and they leak (i.e. cannot be totally switched off). This brings another factor to the front-line of computer design: *reliability*. Reliability-aware architecture research strives to answer the following question: how can we build a reliable machine with unreliable components?

Performance, power, reliability, are related to a single machine. What happens when we connect several machines? *Security* becomes an issue. Security is not only a software issue. Hardware Trojan horses are big threats. How about if your machines gets stolen? Secure hardware with the least performance impact is a very hard problem.

For the rest of this paper, we will discuss all the above issues in more details, shedding some light on the different research directions.

II. Performance

In the early days of computers, performance depended mainly on computations. We try to make machines that compute fast. The big-oh notation, at the algorithmic level, has thus a very precise prediction of the speed by which a machine can execute a specific algorithm. However, technology gave rise to a different trend. Processor's speed increased at a much faster rate than memory speed. Since processors need to be fed with instructions and data from the memory (mostly DRAM till now), memory access became the bottleneck of performance. This is known in computer architecture community as the memory-wall [3]. This sparked several research directions in the past that continued till the present is expected to continue at least for the near future. How to improve DRAM's performance [4], [5]? How to manage the cache memory [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]?

The memory-wall becomes harder as we move from single-core to multicore processors [17]. Our inability to sustain frequency increase in processors forced the designers to adopt a different strategy: instead of increasing the frequency, keep the frequency intact (or even reduce it) but increase the number of cores (aka CPUs) on-chip. A core is now defined as a CPU + level 1 instruction cache and level 1 data cache. We have now a more sophisticated memory hierarchy. The last-level cache (LLC) is usually at level 3, with level 2 being shared or private. How to manage this hierarchy of caches (SRAM) and memory (DRAM) [18], [19], [20], [21], [22], [23], [24]? A question looking for an answer is: as the number of cores increases, how will the cache hierarchy look like? One possible answer is to use tiling for large caches in a way similar to Tilera processors [25]. In this design a tile (a tile is a CPU + private level 1 caches + part of the shared cache + router) contains part of the shared cache. So the LLC is really a distributed shared cache. However, as the number of cores increases, so does the number of tiles and the interconnection among tiles becomes another bottleneck. This brings another factor that affects performance: the interconnection.

If the memory is a major bottleneck of performance, the interconnection among cores and caches is the other bottleneck. This makes NoC (network-on-chip) a very hot topic of research [26], [27], [28], [29]. As the number of cores increases, how will the interconnect evolve [30]? What type of routing algorithm to use? How to deal with coherence overhead [31]? We can confidently say that we are facing a bandwidth-wall [32]. All the above questions are research topics worth exploring.

With memory access and interconnect becoming the main performance bottlenecks, we need to rethink a bit the big-oh notation. Given a specific problem to solve, we may have several candidate algorithms. The big-oh notation tells us, for each algorithm, as the problem size increases how will the number of computations increases. We usually pick the algorithm whose number of computations increases the slowest. That is no longer enough. Computations are no longer the most expensive part. Memory access and communication (among cores and between cores and memory hierarchy) are now more expensive. So we better pick the algorithm that can potentially lead to less memory access and less communication, even if it leads to more computations. If the amount the amount of computations goes beyond the capability of the current processors (i.e. intractable problem) then it is time to move to a another level of parallelism: multiprocessors (supercomputers).

Single-core processors (e.g. Intel Pentium 4) exploits parallelism among instructions [33] through techniques such as superscalar capability citeomondi99. Then simultaneous multithreading capability (known as hyperthreading technology in industrial lingo) was added to cores to exploit small number of parallel threads [34]. Now, with multicore processor, another level of parallelism is added: task-level parallelism. There is a different type of parallelism that exists in some application types: single-thread multiple data (STMD). In this type, few instructions are executed on large amount of data. The best architecture to exploit this type of parallelism is the graphics-processing unit (GPU), sometimes called manycore processors. A general purpose core outsource the part of the code that exhibits STMD to GPUs [35]. From an architectural point of view, we have two schemes: discrete GPU (i.e. a separate GPU chip on-board) and embedded-GPU (the GPU is on-chip together with traditional cores, like Intel Sandy-Bridge architecture). Discrete GPUs are stronger than the embedded ones due to lack of area on-chip. However, a lot of performance is lost in CPU-GPU communication [36]. Embedded GPUs do not suffer from this communication overhead, yet they are not as powerful. What is the best memory hierarchy when GPU and CPU share the LLC? GPU as general-purpose processor is still in its infancy [37], so there are a lot of open questions. Shall we have coherence or not in GPUs? Shall we have speculative execution in GPU [38]? What to expose and what not to expose to the programmer? How to manage the complicated GPU memory hierarchy [39], [40], [41], [42]?

We have touched upon some research topics related to performance. But performance is not very useful if it comes with high power consumption/dissipation.

III. Power

Power is divided into power consumed and power dissipated. Although there is no agreed-upon definition, we can say that power consumed is the one used to do useful work. Power dissipated is a wasted one. Whether consumed or dissipated, power can be divided into two types: dynamic and static. Dynamic power is related to activities in the processor, this is why it is related, among other things, to clock frequency. Static power is related to several reasons, the major one is leakage. Leakage power is related to the fact that as transistor get smaller, they cannot be completely turned-off. The main reason power is becoming a serious problem is that even though Moore's law is still working, but its enabling technology (Dennard Scaling [43]) stopped working since around 2004. This means that the transistor is getting smaller but the power it consumes/dissipates no longer scales down with the transistor's dimensions.

The major source for leakage power is the cache memory. This is why reducing leakage in caches is a worth pursing research problem [44], [45], [46], [47].

As for dynamic power, almost all the techniques we mentioned in the performance section above can be made power-aware, for example power-aware interconnection [48], [49]. The main technique is dynamic voltage and frequency scaling (DVFS). The challenges related to DVFS are: Who makes the decision (programmer, compiler, OS, the hardware)? What triggers the decision? What are the configurations we have for the frequency and voltage? How much to expose to the programmer? Although the literature has tens of papers about power-aware design [50], there are still room for improvement: from the programming-level, compiler-level, operating system level, architectural-level, circuits-level, and VLSI level. With the number of transistors on-chip increasing, we can no longer turn them on at the same time. Which part of the chip shall we turn-off (called dark silicon) with minimal performance impact [51]? With transistors getting smaller, power is not the only problem designers face.

IV. Reliability

As transistors get smaller, they also become less reliable. Yet, we want to design reliable machines with these unreliable transistors. The straightforward solutions are replication: in time and in space. Replication in-time means repeating the computation. Replication in space means replicating the hardware structure and check any discrepancy in results. For storage elements (caches, memory, disks) error-detection and correction codes are used. However, replication comes at cost: in performance and/or in area. In multicore processors, we can make use of idle processors for reliability check, with some cost of power. Some important questions are: how often to check for reliability?

It is important to note that there are two type of structures inside the processor. The first type consists of structures needed for correctness, for example, the execution unit. The second type consists of structures needed for performance, for example, the branch predictor. If structures of the second type fail, the execution is still correct, yet at worse performance. This means we need to check for reliability of the first type to ensure correctness of execution.

Another interesting question: what to do when a faulty structure is detected after deployment? The straightforward solution is to turn-it off with performance loss, assuming we have other structures that do the same job. A more challenging solution is to use the faulty part to give some hints to the non-faulty part [52].

V. Security

As computing devices become widespread, interconnected, and handle sensitive/personal information, the need for security is becoming more crucial. When a platform performs a single

task, never changes from that task and never shares that task over a network, securing such a platform is not challenging. Conventional approaches like anti-virus and anti-spyware tools and OS patches are not totally effective in preventing security attacks. To design a secure computing system, security has to be systematically incorporated into the various stages during the design of such systems: including architecture and hardware implementation [53], [54], [55], [56]. The main challenge, from a hardware perspective, is how to detect both hardware and software malicious activities and deal with them with minimal performance overhead.

VI. Then What?

In this paper, we took a quick look at the different areas of computer architecture status quo and possible future directions. But this paper is by no means exhaustive and we cannot easily predict future directions. At the short-term, we are likely to see evolutionary advances, for example:

- increase in the number of on-chip cores
- larger cache sizes
- bigger GPUs
- reconfigurable interconnects
- non-volatile memory system (e.g. PCM, STT-RAM, MRAM, ...)
- more on-chip heterogeneouity: GPUs + FPGAs + cores of different capabilities
- Anything that is *more of the same with some tweaks!*

At the long term, we may see some revolutionary ideas. These are the hardest to predict, but here are some examples:

- biologically inspired machines
- non-CMOS circuit (Moore's law will eventually ends in less than a decade.)
- quantum computing
- new programming-paradigm: Functional programming and transactional memories may or may not be the answer, we do not know yet!

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, pp. 114–117, April 1965.

[2] N. Leavitt, "Will power problems curtail processor progress?" *Computer*, vol. 45, no. 5, pp. 15–17, May 2012. [Online]. Available: http://dx.doi.org/10.1109/MC.2012.184

[3] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: ACM, 2004, pp. 162–. [Online]. Available: http://doi.acm.org/10.1145/977091.977115

[4] B. L. Jacob and T. N. Mudge, "A look at several memory management units, tlb-refill mechanisms, and page table organizations," in *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VIII. ACM, 1998, pp. 295–306. [Online]. Available: http://doi.acm.org/10.1145/291069.291065

[5] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010, pp. 1–11.

[6] A. Agarwal and S. D. Pudar, "Column-associative cache: A technique for reducing the miss rate of direct-mapped caches," in *Proc. 20th International Symposium on Computer Architecture*, 1993, pp. 179–190.

[7] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite," in *Proc. 42nd ACM Southeast Conference*, 2004.

[8] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proc. 2nd International Symposium on High Performance Computer Architecture*, 1996.

[9] J. N. Chame, "A compiler analysis of cache interference and its applications to compiler optimizations," Ph.D. dissertation, Los Angeles, CA, USA, 1997, aAI9835071.

[10] S. M. Günther and J. Weidendorfer, "Assessing cache false sharing effects by dynamic binary instrumentation," in *Proceedings of the Workshop on Binary Instrumentation and Applications*, ser. WBIA '09, 2009, pp. 26–33.

[11] S. Ghosh, M. Martonosi, and S. Malik, "Cache miss equations: a compiler framework for analyzing and tuning memory behavior," *ACM Trans. Program. Lang. Syst.*, vol. 21, no. 4, Jul. 1999.

[12] M. Kharbutli and Y. Solihin, "Counter-based cache replacement algorithms," in *Proc. International Conference on Computer Design*, Oct 2005, pp. 61–68.

[13] T. Karkhanis and J. E. Smith, "A day in the life of a data cache miss," in *Proc. 2nd Annual Workshop on Memory Performance Issues (WMPI)*, Jun. 2002.

[14] N. Megiddo and D. Modha, "Outperforming lru with an adaptive replacement cache," 2004.

[15] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer, "Adaptive insertion policies for managing shared caches," in *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 208–219.

[16] M. Qureshi, D. Lynch, O. Mutlu, and Y. Patt, "A case for mlp-aware cache replacement," in *Proc. 33rd International Symposium on Computer Architecture (ISCA)*, Jun. 2006.

[17] L. Hammond, B. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *IEEE Computer*, 1997.

[18] B. M. Beckmann and D. A. Wood, "Managing wire delay in large chip-multiprocessor caches," in *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, 2004.

[19] C.-Y. Chang, J.-P. Sheu, and H.-C. Chen, "Reducing cache conflicts by multi-level cache partitioning and array elements mapping," *J. Supercomput.*, vol. 22, no. 2, pp. 197–219, Jun. 2002.

[20] Y. Guo, Q. Zhuge, J. Hu, J. Yi, M. Qiu, and E. H.-M. Sha, "Data placement and duplication for embedded multicore systems with scratch pad memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 6, June 2013.

[21] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004, pp. 111–122.

[22] M. Zahran, K. Albayraktaroglu, and M. Franklin, "Non-inclusion property in multi-level caches revisited," *the International Journal of Computers and Their Applications Special Issue on Techniques and Architectures for High Performance and Energy Efficient Computing Systems*, vol. 14, no. 2, jun 2007.

[23] M. Zahran and S. A. McKee, "Global management of cache hierarchies," in *Proceedings of the 7th ACM international conference on Computing frontiers*, ser. CF '10, 2010, pp. 131–140.

[24] Y. Zheng, B. T. Davis, and M. Jordan, "Performance evaluation of exclusive cache hierarchies," in *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2004, pp. 89–96.

[25] "http://www.tilera.com/."

[26] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnection in multi-core architectures: Understanding mechanisms, overheads, and scaling," in *Int' Symposium on Computer Architecture*, June 2005.

[27] J. Xue, A. Garg, B. Ciftcioglu, J. Hu, S. Wang, I. Savidis, M. Jain, R. Berman, P. Liu, M. Huang, H. Wu, E. Friedman, G. Wicks, and D. Moore, "An intra-chip free-space optical interconnect," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 94–105. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815975

[28] Y. Katayama, A. Okazaki, and N. Ohba, "Software-defined massive-core networking via freespace optical interconnect," in *The ACM International Conference on Computing Frontiers (CF)*, 2013.

[29] M. J. Kobrinsky and et. al., "On-chip optical interconnects," *Intel Technology Journal*, vol. 8, no. 2, pp. 129–142, May 2004.

[30] A. Y. Weldezion, M. Grange, D. Pamunuwa, Z. Lu, A. Jantsch, R. Weerasekera, and H. Tenhunen, "Scalability of network-on-chip communication architecture for 3-d meshes," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '09, 2009, pp. 114–123.

[31] M. M. K. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *Commun. ACM*, vol. 55, no. 7, pp. 78–89, Jul. 2012.

[32] H.-H. S. Lee, G. S. Tyson, and M. T. Farrens, "Eager writeback- a technique for improving bandwith utilization," in *33rd annual IEEE/ACM international symposium on Microarchitecture*, December 2000.

[33] M. A. Postiff, D. A. Green, G. S. Tyson, and T. N. Mudge, "The limits of instruction level parallelism in spec95 applications," in *Proc. 3rd Workshop on Interaction Between Compilers and Computer Architecture (Interact-3)*, 1998.

[34] D. M. Tullsen, S. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proc. 22th Int'l Symposium on Computer Architecture*, 1995.

[35] M. Arora, S. Nath, S. Mazumdar, S. B. Baden, and D. M. Tullsen, "Redefining the role of the cpu in the era of cpu-gpu integration," *Micro, IEEE*, vol. 32, no. 6, pp. 4–16, 2012.

[36] T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R. Beard, and D. I. August, "Automatic cpu-gpu communication management and optimization," in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '11, 2011, pp. 142–151.

[37] E. Blem, M. Sinclair, and K. Sankaralingam, "Challenge benchmarks that must be conquered to sustain the gpu revolution," in *4th Workshop on Emerging Applications for Manycore Architecture*, 2011.

[38] J. Menon, M. de Kruijf, and K. Sankaralingam, "igpu: Exception support and speculative execution on gpus," in *Proceedings of 39th International Symposium on Computer Architecture (ISCA)*, ser. ISCA'12, June 2012.

[39] N. B Lakshminarayana, J. Lee, H. Kim, and J. Shin, "Dram scheduling policy for gpgpu architectures based on a potential function," *Computer Architecture Letters*, vol. PP, p. 1, 2011.

[40] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09, 2009, pp. 152–163.

[41] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated scheduling and prefetching for gpgpus," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[42] K. S. Lee, "Characterization and exploitation of gpu memory systems," Master's thesis, Virginia Polytechnic Institute and State University, 2012.

[43] "Discussing dram and cmos scaling with inventor bob dennard," *IEEE Des. Test*, vol. 25, no. 2, pp. 188–191, Mar. 2008.

[44] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *Proc. International Symp. on Low Power Electronics and Design*, Jul. 2000, pp. 90–95.

[45] S. Roy and K. Chakraborty, "Microarchitecture aware gate sizing: A framework for circuit-architecture co-optimization," in *Proceedings of the Int'l Conference on Computer Design (ICCD)*, October 2010, the paper argues that DVFS techniques are becoming less effective as Vt decreases and concerns in reliability and leakage arise. They propose to design some component with slower technology with as little impact on performance as possible. Their main argument is that components are designed for best power-performance at highest utilization. But this high utlization is rarely reached, and reducing V and F dynamically makes these component less perfect in terms of power-performance.

[46] A. Youssef, M. Anis, and M. I. Elmasry, "Dynamic standby prediction for leakage tolerant microprocessor functional units," in *39th annual IEEE/ACM international symposium on Microarchitecture*, December 2006, pp. 371–384.

[47] W. Zhang, M. Kandemir, M. Karakoy, and G. Chen, "Reducing data cache leakage energy using a compiler-based approach," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 3, pp. 652–678, Aug. 2005.

[48] W. Kdouh, "On the power management of multi-core processors with network on chip," Ph.D. dissertation, Dallas, TX, USA, 2010, aAI3409220.

[49] A. K. Kodi and A. Louri, "Power-aware bandwidth-reconfigurable optical interconnects for high-performance computing (hpc) systems," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, march 2007, pp. 1 –10.

[50] M. Sjlander, M. Martonosi, and S. Kaxiras, *Power-Efficient Computer Architectures: Recent Advances*. Morgan and Claypool Publishers, 2014.

[51] M. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013. [Online]. Available: http://dx.doi.org/10.1109/MM.2013.90

[52] A. Durytskyy, M. Zahran, and R. Karri, "Improving robustness of gpus by making use of faulty parts," in *Proc. International Conference on Computer Design (ICCD11)*, 2011.

[53] A. Waksman and S. Sethumadhavan, "Tamper evident microprocessors," may 2010, pp. 173 –188.

[54] L. Uhsadel, A. Georges, and I. Verbauwhede, "Exploiting hardware performance counters," in *5th Workshop on Fault Diagnosis and Tolerance in Cryptography, 2008 (FDTC '08)*, aug. 2008, pp. 59 –67, * This paper proposes the usage of hardware performance counters for microarchitectural side-channel attacks.

[55] G. Kornaros and D. Pnevmatikatos, "A survey and taxonomy of on-chip monitoring of multicore systems-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp. 17:1–17:38, Apr. 2013. [Online]. Available: http://doi.acm.org/10.1145/2442087.2442088

[56] A. M. Fiskiran and R. B. Lee, "Runtime execution monitoring (rem) to detect and prevent malicious code execution," in *ICCD '04: Proceedings of the IEEE International Conference on Computer Design*, 2004, pp. 452–457.