Appears in the 2nd USENIX Workshop on Hot Topics in Parallelism (HotPar '10)

Get the Parallelism out of My Cloud

Karthikeyan Sankaralingam and Remzi H. Arpaci-Dusseau
University of Wisconsin-Madison

Abstract

The hardware trend toward multicore processors has so far been driven by technology limitations of wire delays, power efficiency, and limited capability to exploit instruction-level parallelism. Software evolution has lead to the rise of the cloud. This multicore + cloud evolution provides several challenges and has led to a call for parallelism.

In this paper, we first examine the drivers behind these trends to address three fallacies: software is driven by hardware, multicores will be everywhere, and multicore hardware implies parallelism is exposed to all developers. We first address these fallacies and then present our simple view of the future cloud-based ecosystem, based on what we refer to as data-centric concurrency.

1 Introduction

Two trends dominate the landscape of computing today (and likely will for the next many years). The first is a hardware trend: the rise of multicore processors [18]. The second is a software trend: the coming storm of cloud computing [5].

Trend 1: The Rise of Multicore

For years, architects worked steadily towards the goal of achieving the highest single-thread performance possible. Various techniques, including caching [17], speculation [23], and other aggressive optimizations [21, 9] sustained this movement for many years. But, like all good things, this too came to an end.

Microprocessors have been recently forced into multicore designs because of fundamental energy efficiency of transistors and wire-delay properties [15, 14]. In multicore chips, multiple simpler processors are deployed instead of a single large one; parallelism, which used to be hidden within the microarchitecture of a processor, becomes exposed to programmers.

The move to multicore seemingly represents a sea-change for developers. In the past, the best way to improve performance was simply to *wait*; the next-generation CPU would arrive, and the inevitable performance bump would be achieved. With multicore, this performance-improvement is no longer available; adding cores does not improve single-thread performance.

Trend 2: The Rise of the Cloud

The dominant software trend has been the advent of "cloud computing" [5], an idea that has been articulated many times in the past (e.g., the utility computing vision behind Multics [10]) but only now is becoming a standard approach to computing systems.

This world consists of two important (and radically different) computing platforms. The platform most directly relevant to users is the "device", by which we mean the computer system that the user primarily interacts with in order to obtain services from the cloud or otherwise run applications. The platform that exists to support devices is what most people now refer to as the "cloud", which are the thousands of servers that combine to provide back-end service of varying types.

The cloud has a tremendous impact on software development. As an extreme point, consider Google's Chrome OS, which is a stripped-down Linux platform meant to run a single application: a web browser. In this world, virtually all applications are true "web" applications, where substantial server-side components and client-side code (e.g., javascript) combine to form what the user perceives as a single application.

2 The Algebra of the Future: Multicore + Cloud = ?

We believe that viewing these trends in isolation has led our field astray in articulating a vision for future research. We now discuss three critical fallacies that have arisen.

Fallacy #1: Software Trends Will Be Dictated By Hardware Trends

The focus of software development has changed dramatically over the past decades. We first review the past, examine current trends, and conclude with our future projection.

In the past few decades, hardware used to be prohibitively expensive, to the extent that proper utilization of every cycle was of the utmost concern. For example, the original developers of UNIX were finally able to obtain a machine to work upon (in the early 1970's) when the cost dropped to $65,000 [20]; adjusted for inflation, the amazing cost of the machine is $360,000.[1]

As Jim Gray once related: "When I was growing up, the only metric was *cycles*; what mattered was how efficient your algorithm was." [12]. An examination of software technology of the time confirms this. Developers for many years were forced to use C and other low-level languages to deliver efficient, lean applications to users.

However, hardware costs have consistently dropped and performance has risen so dramatically, that counting cycles is no longer the developer's primary worry. Today's software ecosystem is different in two ways. First, significant amount of today's software is web-deployed software. Second, today's web developer, uses modern frameworks where much of the code is interpreted. The clear goal of said frameworks is to enable the developer to write as little code as possible to implement their ideas, and efficiency is left on the back-burner. As a result, as Gray further elaborated, the key metric is *time-to-market*. What matters is how quickly you can develop an application and make it available to users.

An excellent example of this new philosophy is found in Django, one of many relevant web development systems. As stated on the Django web page: "Django focuses on automating as much as possible and adhering to the DRY principle: Don't Repeat Yourself." [22] The DRY principle quite simply states: "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." The one and only goal is to write as little code as possible.

Undoubtedly hardware *played* a role in this evolution[2]. The relative abundance of computing capability has made computation a commodity thus enabling such other metrics becoming primary constraints and these new frameworks (and their philosophies). Developers no longer think of efficiency, they simply think of how to get what they want done in as few lines of code as possible. Minimalism is the dominant software trend[3].

Thus, it no longer makes sense to think of hardware trends first when trying to understand where software is headed; at best, the opposite is needed. The bottom line: *Software trends are now largely independent of hardware trends. If they do relate, it is in the opposite direction: the demands of software will drive hardware, and not vice-versa.*

Fallacy #2: Multicore Everywhere

The observation that multicores will be ubiquitous assumes high-performance computing must be performed locally (i.e., on the client). While 10 TFLOPS at 1 Watt for x86 applications would be great, it is not realistic; neither is it necessary or useful, as we outline below.

Instead of examining microprocessor trends alone, we view the entire information technology ecosystem. We see an obvious inflection point. We observe that the conventional world of the personal high-performance computation device is disruptively transforming into highly energy-efficient and mobile devices wirelessly tethered to remote clouds and hence providing orders of magnitude increases in performance, reliability, and maintainability than personal desktops. This transformation is driven by usability trends, microprocessor trends, and technology trends, as we now discuss.

Usability trend: Highly-mobile devices ("converged mobile devices" in industry parlance) are projected to have explosive growth, reaching 390 million units in 2013 [2]. In contrast world-wide sales of desktop, notebooks, and servers is projected to total 440 million in 2013. These devices will soon outnumber conventional desktops. We observe that many sharing and data-mining related tasks require access to the cloud and must offload computation thus making high-performance on the client moot.

Microprocessor trend: As transistors have gotten smaller and faster overall, microprocessor performance has improved over the years. We contend that 2006/2007 is a subtle inflection point where a "powerful" microprocessor consuming only a few watts could be built. In terms of quantitative benchmarks, SPECINT scores of 700 to 1000 could be obtained. While the number itself is inconsequential, with such a device several common tasks could be easily executed; more importantly, a web browser could execute without creating interaction problems. From a specific processor-design standpoint, the Intel Atom chip (or designs from ARM, Via, or Centaur) can be viewed as the inflection point, perhaps portending a second "attack of the killer micros". As the RISC

[1] Humorously, the high cost of the machine didn't prevent Ken Thompson from playing Space Travel at $75 a game!

[2] We concur with the conventional wisdom and recognize the empirical evidence that software trends were dictated by hardware in the past.

[3] One of the synergistic mechanisms in these web frameworks is that they have abundant concurrency and effectively hide this from the programmer, which we elaborate more in our third fallacy.

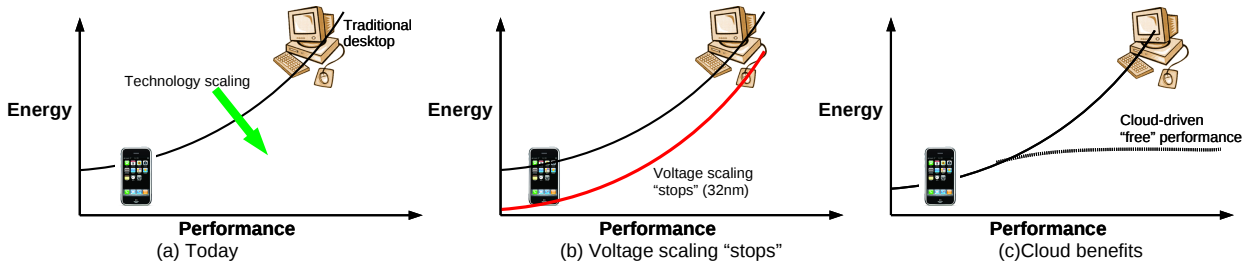

Figure 1: Microprocessor technology driven trends

	Number of cores				
	(45 nm) Current	**(32nm) 2012**	**(24 nm) 2014**	**(19 nm) 2016**	**(15 nm) 2018**
Mobile-devices	1	2	3-4	5-6	9-10
***Complex* servers**	8	16	28	44	72
***Simple* servers**	64	128	225	359	576

Projected core counts for different types of processors based on ITRS scaling.

and x86 micros became dominant in HPC by focusing on energy and cost, these smaller devices are poised to take over personal computing using those same metrics.

Technology trends: The energy efficiency of transistors is no longer increasing from one generation to the next. Figure 1 shows a Pareto-optimal curve of energy vs. performance. For a given technology node (e.g., a transistor size of 45nm), different microprocessor design techniques, including multicore vs. unicore, simply move a design from a given energy/performance point higher or lower. In previous years, technology scaling lowered this curve every year, thus providing additional performance in every generation. Going forward, voltage scaling faces fundamental challenges and little or no energy benefit will be seen from technology scaling. However, sufficient network bandwidth and efficient network processing provide practically free performance to a hand-held device as shown in Figure 1c. There is essentially no requirement for a high-energy high performance conventional desktop, which is a fundamental driver behind cloud-computing.

The table below the figure shows an optimistic (and simple) view of microprocessor scaling in terms of number of cores. These projections are based on the ITRS projections of technology scaling. Even assuming multicores will become dominant everywhere, a realistic assessment shows the number of cores in mobile devices will be limited to a handful through the end of this decade and beyond. Furthermore, providing multiple programmable general-purpose cores on a chip does not fundamentally alter the energy-efficiency problem. Thus, in the long-term, devices are more likely to use specialization to provide energy-efficient functionality. The number of "cores" in a device is likely to be less than even the 9-10 range projected from technology scaling. Core counts for servers will likely grow as shown in the table depending on the complexity of the core itself.

Based on these trends, we argue that the two dominant types of compute systems will be lightweight devices and remote clouds, with personal high-performance desktops becoming irrelevant. We argue that the only utility of high-performance local computation is for "games" which are following a natural evolutionary path toward immersive reality. Examples of this trend include the Wii remote, Microsoft's Project Natal, and Sony's motion controller. In this paper, we do not address the challenges or propose solutions for this segment of computation. Usability and economic trends clearly show that this segment will be a separate discrete device and will not be a general-purpose PC. Game-console software sales and handheld game software sales totaled $11.3 billion, while PC game sales totaled less than $701 million for 2009. High-performance desktops are no longer computationally superior or well-suited as gaming platforms.

Summarizing: *Multicore will (likely) dominate in the server cloud but not on the devices used to access the cloud. Thus, how we use multicore chips in server clouds is the main issue we face.*

Fallacy #3: The Hardware Is Parallel; Thus, Everyone Must Be a Parallel Programmer

Some have concluded from industry announcements about multicore that we all should become parallel programmers. For example, the recent standard curriculum proposed by ACM has a strong focus on parallel (concurrent) programming [3]. Quoting the authors:

> "The development of multicore processors has been a significant recent architectural development. To exploit this fully, software needs to exhibit concurrent behavior; this places greater emphasis on the principles, techniques and technologies of concurrency."

If parallelism in software is what is needed, then the IT industry may indeed be in trouble [6]. As John Hennessy has been quoted saying: "...when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. ...I would be panicked if I were in industry." [6]

Because the goal of parallel software has been extant for so long, perhaps we should take that as good evidence that is generally to be avoided rather than embraced.

Before addressing the parallelism problem and our strategy on how to avoid it, let us summarize our analysis thus far: multicore processors will be found within the server cloud and probably less so on devices accessing said cloud. The direct implication: a large amount of parallel hardware will exist that server-side software must exploit.

Fortunately, in the cloud, true software parallelism will rarely be needed to utilize the many cores found across servers. To understand why, we first define a few terms. We label two tasks *concurrent* if they can be (or are) executing at the same time, but are unrelated (i.e., not working on the same problem). Some examples include separate applications running on your phone, or even threads within a web-server processing different requests.

We label two tasks as *parallel* if they are executing at the same time but working on the same larger task. For example, a search for a keyword across many machines (each of which contains a small portion of the search database), or a parallel matrix multiply.

To utilize "parallel hardware", *either* concurrency or parallel software is needed. Indeed this is fortunate, because concurrency is ubiquitous within server-side cloud software and generally easy to come by. Imagine thousands of users simultaneously editing documents within Google Docs; the backend support for this will certainly execute many (unrelated) concurrent tasks, thus being fully capable of utilizing parallel hardware without great effort. For most server software on the cloud, concurrency is the solution, not parallelism.

This is not to say that parallelism will not have a place within the cloud; indeed, services that access massive amounts of data will always need to execute in parallel across many nodes (e.g., a Google search). However, such parallelism, whether specialized as in the Google example or more general as in an SQL query, will be the rare case. This is good news as it leaves parallelism where it has always been in computer systems: hidden behind the scenes, left to a few experts in particular application domains where it is required[4]. Web-development frameworks particularly excel at hiding and abstracting such details away from the programmer. Second, by construction, web applications are themselves written as a set of fine-grained concurrent tasks.

Thus, our final point: *Concurrency of unrelated tasks will naturally utilize the many cores of cloud servers; there will be no great need to parallelize all software in order for the cloud to be successful.*

3 Future Directions

As the device may be largely unaffected by multicore chips, we focus on challenges for the future on the server side of the cloud. We discuss both hardware and software research directions.

Our overarching theme is the need to design *data-centric concurrent servers (DCC servers)*. This new class of machine will be built to service the needs of the cloud running many concurrent (and thus largely unrelated) tasks.

3.1 Hardware Challenges

Technology Revisited: The primary technology constraint for this decade and beyond is the decreasing energy efficiency of transistors. While the number of devices is expected to double every generation, the power efficiency of devices is growing slowly. The main reason behind this trend is that classical voltage scaling has effectively ended and capacitance of transistors is reducing slowly from one generation to another [1]. While the number of transistors will increase sixteen-fold from now through 2020, due to limitations in device efficiency, power will increase. Applying a simple model of 5% reduction in voltage per generation, first-order models show an approximate three-fold increase in power.

While this problem is daunting, understanding the synergy with application provides hope. In the context of designing DCCs, several simplifications come to our rescue: (a) global coherence can be significantly relaxed, (b) managing the concurrency is simpler because the tasks are independent, and (c) extracting the concurrency is purely a software problem. Thus DCCs will be simple to design. In fact, this design is synergistic with future game console (graphics) platforms as well where there is a requirement for simplicity and the ability to handle concurrency [11]. Interesting research directions to explore are:

- **Locality.** The main problem is to extract and exploit locality at unprecedented levels beyond conventional caching techniques. Fundamental abstractions are required to directly map data structures

[4]Indeed, almost every example of successful parallelism in computer systems has come in a form hidden from most programmers: Google search, extracting ILP at a microarchitectural level, and RAID disk arrays [19] are just a few good examples

to hardware. For example, staged databases [4] attempt to exploit instruction-locality by dedicating different cores to different types of database queries. PLUG processors take the extreme approach of building a programmable on-chip memory that spatially lays out data-structures and provides a high-level abstraction to software [7].

- **Specialization:** New types of specialization, heterogeneity, and run-time reconfiguration are required. The main opportunity is provided by the fact that most software executed will be written in interpreted high-level languages. Energy efficiency can be obtained by specializing hardware to the most-common tasks. For example, DVD encoding in hardware using a specialized chip is 10000 times more energy efficient that software using even a media-enhanced ISA like SSE/MMX [13]. Computation spreading [8] attempts to dynamically specialize different cores in a processor to different types of functionality. In the past, such specialization in hardware was impractical because of the heterogeneity in executed code. However, with web applications, optimizations are possible to the runtime based on analysis of the most frequent regions.

Issues not addressed above include deterministic debugging of back-end servers, performance optimizations, instrumentation, and reliability. We believe, all these issues pose challenges as well and provide interesting avenues for further exploration.

3.2 Software Challenges

The systems software challenges will also be plentiful in the coming era of DCC servers. Here we highlight a few directions for systems work within these servers:

- **OS Support for Hardware Non-Coherence.** Memory in future DCC servers need not be coherent; hence, OS support, and in some cases, radical re-design, is required. As a simple example, the OS will have to explicitly move memory when migrating threads or processes across different cores.

- **Concurrency Management.** Server-side operating systems will need to multiplex thousands of largely-independent tasks across a few cores; managing such concurrency efficiently and with higher-level objectives in mind (e.g., users who paid for the service should receive priority) will thus be a primary OS concern. Extremely scalable task management will be needed. Current schedulers likely do not scale to the levels required by DCCs.

- **Global Resource Management.** As clouds are comprised of large collections of multicores connected via high-performance networks, traditional OS issues of scheduling and resource management at a global scale become increasingly relevant. Reducing peak usage of resources may be a primary goal, as peak drives cost and power usage [16]; thus, clever techniques to better offload data movement and computation to non-peak hours is needed.

- **Low-latency storage.** As the cloud becomes the last tier in the storage hierarchy, and as more data moves to the cloud, building and deploying low-latency storage clouds will become of the utmost importance. Flash and other solid-state storage technologies are likely to be used; redesign of file storage software within DCCs is needed to realize the full benefits of these emerging hardware platforms. Better integration between volatile memories and persistent storage devices will likely be of great importance.

We believe this list represents only a small fraction of many possible interesting software directions for DCC-based systems, and it is certainly biased by our own interests and perspective. Other interesting directions undoubtedly exist, particular in the domain of compiler support and other related areas of research.

4 Conclusions

Our analysis of current and future hardware and software trends bears some similarity to the classic tale of the horse and the cart; which pulls which? In the old world, the horse (hardware) clearly pulled the cart (software), and thus writing efficient software was needed to extract the most out of the precious few cycles available. In the new world, advances in hardware have led us to an abundance of cycles, and a modern emphasis on ease of development and deployment; the cart (software) seems to be now driving itself, without much attention paid to the horse (hardware). Put simply: where the cart goes, the horse must now follow.

The software cart tells us that the cloud is the platform of the future; it has many innate advantages over traditional software development models and a huge amount of momentum behind it. Thus, how should the systems horse follow? We suggest in this paper an approach based on simplification and specialization; by building innately parallel hardware, but using it to largely execute concurrent (unrelated) tasks, we believe that many of the challenges in performance, energy-efficiency, and cost can be overcome. Many research issues remain, and only through broad efforts of a large and focused research community will they be adequately addressed.

Acknowledgments

We thank the anonymous reviewers for their tremendous feedback and insightful comments, which have substantially improved the content and presentation of this paper.

This material is based upon work supported in part by the National Science Foundation under the following grants: CNS-0509474, CCF-0621487, CCF-0811657, CNS-0834392, CCF-0937959, CCF-0845751, CCF-0917238, CNS-0917213 as well as by generous donations from Google, NetApp, HP, and Samsung.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other institutions.

References

[1] Semiconductor Industry Association (SIA), Process Integration, Devices, and Structures, International Roadmap for Semiconductors, 2005 edition. Int. SEMATECH, 2008.

[2] Smartphone Forecast for 2009, IDC Forecast.

[3] ACM. Computer Science Curriculum 2008: An Interim Revision of CS 2001. www.acm.org/education/curricula/ComputerScience2008.pdf, 2008.

[4] Stavros Harizopoulos Anastassia and Anastassia Ailamaki. Stageddb: Designing database servers for modern hardware. In *In IEEE Data*, pages 11–16, 2005.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, February 2009.

[6] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A View of the Parallel Computing Landscape. *CACM*, 52(10):56–67, 2009.

[7] Lorenzo De Carli, Yi Pan, Amit Kumar, Cristian Estan, and Karthikeyan Sankaralingam. Plug: Flexible lookup modules for rapid deployment of new protocols in high-speed routers. In *SIGCOMM '09*, 2009.

[8] Koushik Chakraborty, Philip M. Wells, and Gurindar S. Sohi. Computation spreading: employing hardware migration to specialize cmp cores on-the-fly. In John Paul Shen and Margaret Martonosi, editors, *ASPLOS*, pages 283–292. ACM, 2006.

[9] George Z. Chrysos and Joel S. Emer. Memory dependence prediction using store sets. In *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, pages 142–153, Washington, DC, USA, 1998. IEEE Computer Society.

[10] F. J. Corbata and V. A. Vyssotsky. Introduction and overview of the Multics system. In *AFIPS Conference 27*, pages 185–196, 1965.

[11] Venkatraman Govindaraju, Peter Djeu, Karthikeyan Sankaralingam, Mary Vernon, and William R. Mark. Toward a Multicore Architecture for Real-time Ray-tracing. In *Proceedings of the 41st Annual International Symposium on Microarchitecture*, November 2008.

[12] Jim Gray. Personal Communication, June 2002.

[13] Mark Hempstead, Gu-Yeon Wei, and David Brooks. Navigo: An early-stage model to study power-constrained architectures and specialization. In *Workshop on Modeling, Benchmarking, and Simulation*, 2009.

[14] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[15] Mark Horowitz, Elad Alon, and Dinesh Patil. Scaling, Power and the Future of CMOS. In *Electron Devices Meeting, 2005. IEDM Technical Digest*, 2005.

[16] James Hamilton. Where Does the Power Go in High-Scale Data Centers? Keynote at USENIX ATC, June 2009.

[17] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers. In *25 Years ISCA: Retrospectives and Reprints*, pages 388–397, 1998.

[18] Kunle Olukotun Lance Hammond, Basem A. Nayfeh. A Single-Chip Multiprocessor. IEEE Computer Special Issue on Billion-Transistor Processors, September 1997.

[19] David Patterson, Garth Gibson, and Randy Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD88*, pages 109–116.

[20] Dennis M. Ritchie. The Evolution of the Unix Time-sharing System. In *Language Design and Programming Methodology Conference*, September 1979.

[21] Gurindar S. Sohi, Scott E. Breach, and T. N. Vijaykumar. Multiscalar processors. In *ISCA '95: Proceedings of the 22nd annual international symposium on Computer architecture*, pages 414–425, New York, NY, USA, 1995. ACM.

[22] The Many Django Coders. The Django Project. http://www.djangoproject.com/, 2010.

[23] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive training branch prediction. In *International Symposium on Microarchitecture*, pages 51–61, 1991.