The Price of

Performance

LUIZ ANDRÉ BARROSO, GOOGLE

An Economic Case for Chip Multiprocessing

In the late 1990s, our research group at DEC was one of a growing number of teams advocating the CMP (chip multiprocessor) as an alternative to highly complex single-threaded CPUs. We were designing the Piranha system,[1] which was a radical point in the CMP design space in that we used very simple cores (similar to the early RISC designs of the late '80s) to provide a higher level of thread-level parallelism. Our main goal was to achieve the best commercial workload performance for a given silicon budget.

Today, in developing Google's computing infrastructure, our focus is broader than performance alone. The merits of a particular architecture are measured by answering the following question: Are you able to afford the computational capacity you need? The high-computational demands that are inherent in most of Google's services have led us to develop a deep understanding of the overall cost of computing, and continually to look for hardware/software designs that optimize performance per unit of cost.

This article addresses some of the cost trends in a large-scale Internet service infrastructure and highlights the challenges and opportunities for CMP-based systems to improve overall computing platform cost efficiency.

UNDERSTANDING SYSTEM COST

The systems community has developed an arsenal of tools to measure, model, predict, and optimize performance. The community's appreciation and understanding of cost factors, however, remain less developed. Without thorough consideration and understanding of cost, the true merits of any one technology or product remain unproven.

We can break down the TCO (total cost of ownership) of a large-scale computing cluster into four main components: price of the hardware, power (recurring and initial data-center investment), recurring data-center operations costs, and cost of the software infrastructure.

Often the major component of TCO for commercial deployments is software. A cursory inspection of the price breakdown for systems used in TPC-C benchmark filings shows that per-CPU costs of just operating systems and database engines can range from $4,000 to $20,000.[2] Once the license fees for other system software components, applications, and management software are added up, they can dwarf all other components of cost. This is especially true for deployments using mid- and low-end servers, since those tend to have larger numbers of less

The Price of Performance

expensive machines but can incur significant software costs because of still-commonplace per-CPU or per-server license-fee policies.

Google's choice to produce its own software infrastructure in-house and to work with the open source community changes that cost distribution by greatly reducing software costs (software development costs still exist, but are amortized over large CPU deployments). As a result, it needs to pay special attention to the remaining components of cost. Here I will focus on cost components that are more directly affected by system-design choice: hardware and power costs.

Figure 1 shows performance, performance-per-server price, and performance-per-watt trends from three successive generations of Google server platforms. Google's hardware solutions include the use of low-end servers.[3] Such systems are based on high-volume, PC-class components and thus deliver increasing performance for roughly the same cost over successive generations, resulting in the upward trend of the performance-per-server price curve. Google's fault-tolerant software design methodology enables it to deliver highly available services based on these relatively less-reliable building blocks.

Nevertheless, performance per watt has remained roughly flat over time, even after significant efforts to design for power efficiency. In other words, every gain in performance has been accompanied by a proportional inflation in overall platform power consumption. The result of these trends is that power-related costs are an increasing fraction of the TCO.

Such trends could have a significant impact on how computing costs are factored. The following analysis ignores other indirect power costs and focuses solely on the cost of energy. A typical low-end x86-based server today can cost about $3,000 and consume an average of 200 watts (peak consumption can reach over 300 watts). Typical power delivery inefficiencies and cooling overheads will easily double that energy budget. If we assume a base energy cost of nine cents per kilowatt hour and a four-year server lifecycle, the energy costs of that system today would already be more than 40 percent of the hardware costs.

And it gets worse. If performance per watt is to remain constant over the next few years, power costs could easily overtake hardware costs, possibly by a large margin. Figure 2 depicts this extrapolation assuming four different annual rates of performance and power growth. For the most aggressive scenario (50 percent annual growth rates), power costs by the end of the decade would dwarf server prices (note that this doesn't account for the likely increases in energy costs over the next few years). In this extreme situation, in which keeping machines powered up costs significantly more than the machines themselves, one could envision bizarre business models in which the power company will provide you with free hardware if you sign a long-term power contract.

The possibility of computer equipment power consumption spiraling out of control could have serious consequences for the overall affordability of computing, not to mention the overall health of the planet. It should be noted that although the CPUs are responsible for only

Three Successive Generations of Google Hardware

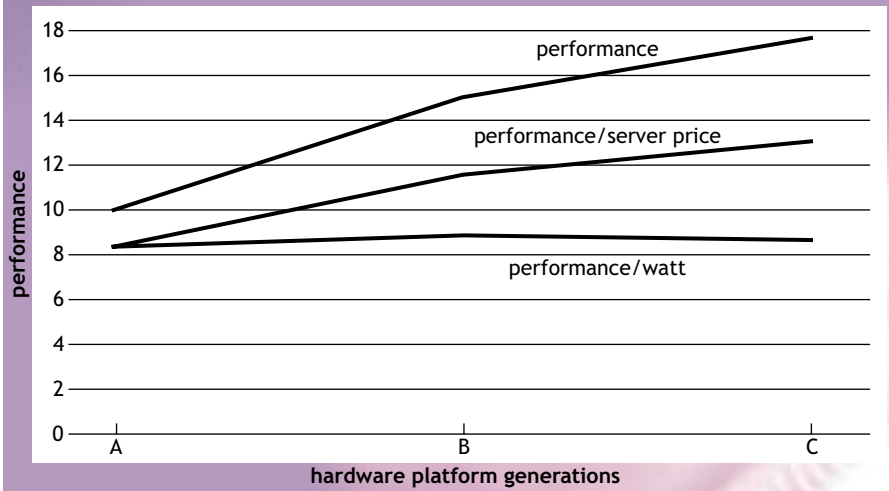


FIG 1

a fraction of the total system power budget, that fraction can easily reach 50 percent to 60 percent in low-end server platforms.

THE CMP AND COMPUTING EFFICIENCY

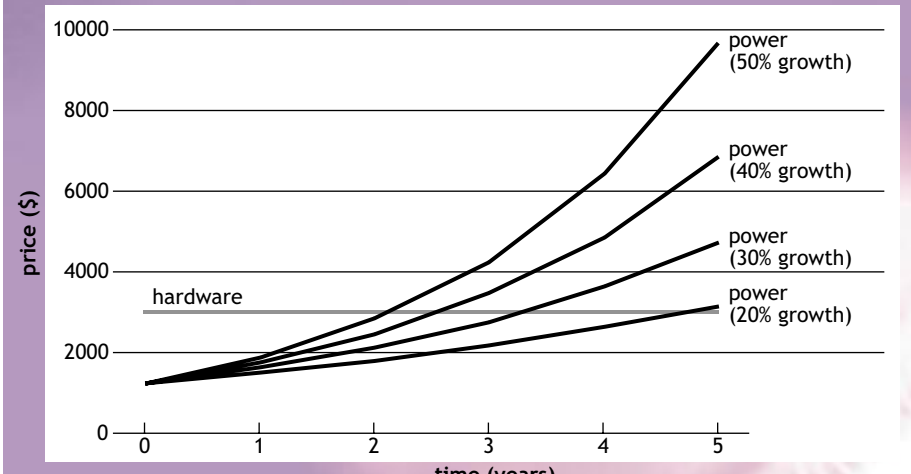
The eventual introduction of processors with CMP technology is the best (and perhaps only) chance to avoid the dire future envisioned above. As discussed in the opening article of this issue ("The Future of Microprocessors," by Kunle Olukotun and Lance Hammond), if thread-level parallelism is available, using the transistor and energy budget for additional cores is more likely to yield higher performance than any other techniques we are aware of. In such a thread-rich environment, prediction and speculation techniques need to be extremely accurate to justify the extra energy and real estate they require, as there will be nonspeculative instructions ready to execute from other threads. Unfortunately, many server-class workloads are known to exhibit poor instruction-level parallelism;[4] therefore, they are a poor match for the aggressive speculative out-of-order cores that are common today.

Some key workloads at Google share such behavior. Our index-serving application, for example, retires on average only one instruction every two CPU cycles on modern processors, badly underutilizing the multiple issue slots and functional units available. This is caused by the use of data structures that are too large for on-chip caches, and a data-dependent control flow that exposes the pipeline to large DRAM latencies. Such behavior also causes the memory system to be underutilized, since often a new memory access cannot be issued until the result of a previous one is available. There is enough unpredictability in both control flow and memory access streams to render speculation techniques relatively ineffective. This same workload, however, exhibits excellent thread-level speedup on traditional multiprocessors, simultaneous multithreaded systems, and CMPs.[5]

The Piranha implementation took the lessons from commercial workload behavior to heart: If there are enough threads (hardware and software), one should never have to speculate. The eight CPU cores were a throwback to early RISC designs: single-issue, in-order, nonspeculative. The first Piranha chip was expected to outperform state-of-the-art CPUs by more than a factor of two at nearly half the power consumption. What makes this especially significant is that this was achieved despite our team having completely ignored power efficiency as a design target. This is a good illustration of the inherent power-efficiency advantages of the CMP model.

Recent product announcements also provide insights into the power-efficiency potential of CMP microarchitectures. Both AMD and Intel are introducing CMP designs that stay within approximately the same power envelope of their previous-generation single-core offerings. For example, AMD reports that its dual-core Opteron 275 model outperforms its single-core equivalent (Opteron 248) by about 1.8 times on a series of benchmarks,[6] at a power envelope increase of less than 7 percent. Even if we pessimistically assume that the whole platform power increases by that same amount, the power efficiency of the dual-core platform (performance per watt) is still nearly 70 percent better than the single-core platform. Indeed, process technology improvements do play a large role in achieving this, but the fact remains that for the first time in many processor generations we are looking at dramatic power-efficiency improvements.

Extrapolation of Hardware and Power Costs for Low-End Servers*



*assumes constant performance/watt over the next five years

FIG 2

The Price of Performance

SLOW PACE

In our first Piranha paper published in 2000 we described chip multiprocessing as an inevitable next step in microarchitectural evolution. Although this is no longer a controversial view, it is nevertheless surprising that it has taken so long for this architecture to achieve widespread acceptance. I am particularly surprised that more aggressive CMP architectures—those (like Piranha) that trade single-threaded performance for additional thread-level parallelism—are only now beginning to appear in commercial products[7] and are unlikely to be widely available for quite some time.

The commercial introduction of CMPs seems to be following a more measured approach in which fairly complex cores are being slowly added to the die as the transistor budget increases every process generation. If CMPs have such compelling potential, why is it taking so long for that potential to be realized? There are four main reasons for this:

It's the power envelope, stupid. As it turned out, contrary to what we envisioned during the Piranha development, design complexity and performance alone were not compelling enough to trigger a switch to CMP architectures; power was. In order to steer away from expensive cooling technologies, chip developers had to stay within power density boundaries that became increasingly difficult to meet with conventional techniques.

Marketing matters. Megahertz is a performance metric that is easy to understand and communicate to consumers. Although it is a very poor indicator of application performance, the same can be said for most popular benchmarks. When given a choice between a bogus metric that sells and one that doesn't, the outcome is predictable. Unfortunately, the MHz competition has reinforced the direction toward larger and more complex single-threaded systems, and away from CMPs.

Execution matters. Many of us underestimated the incredible engineering effort that went into making conventional complex cores into very successful products. Seemingly suboptimal architectures can be made into winning solutions with the right combination of talent, drive, and execution.

Threads aren't everywhere yet. Although server-class workloads have been multithreaded for years, the same cannot be said yet for desktop workloads. Since desktop volume still largely subsidizes the enormous cost of server CPU development and fabrication, the lack of threads in the desktop has made CMPs less universally compelling. I will expand on this issue later in this article.

DREADING THREADING

Much of the industry's slowness in adopting CMP designs reflects a fear that the CMP opportunity depends on having enough threads to take advantage of that opportunity. Such fear seems to be based mainly on two factors: parallel programming complexity and the thread-level speedup potential of common applications.

The complexity of parallel software can slow down programmer productivity by making it more difficult to write correct and efficient programs. Computer science students' limited exposure to parallel programming, lack of popular languages with native support for parallelism, and the slow progress of automatic compiler parallelization technology all contribute to the fear that many applications will not be ready to take advantage of multithreaded chips.

There is reason for optimism, though. The ever-growing popularity of small multiprocessors is exposing more programmers to parallel hardware. More tools to spot correctness and performance problems are becoming available (e.g., thread checkers[8] and performance debuggers[9]). Also, a few expert programmers can write efficient threaded code that is in turn leveraged by many others. Fast-locking and thread-efficient memory allocation libraries are good examples of programming work that is highly leveraged. On a larger scale, libraries such as Google's MapReduce[10] can make it easier for programmers to write efficient applications that mine huge datasets using hundreds or thousands of threads.

While it's true that some algorithms are hard to parallelize efficiently, the majority of problem classes that demand the additional performance of CMPs are not. The general principle here is that, with few exceptions, the more data one has, the easier it is to obtain parallel speedup. That's one of the reasons why database applications have been run as parallel workloads successfully for well over a decade. At Google we have generally been able to tune our CPU-intensive workloads to scale to increasing numbers of hardware threads whenever needed—that is, whenever servers with higher numbers of hardware contexts become economically attractive.

The real challenge for CMPs is not at the server but the desktop level. Many popular desktop applications have not been parallelized yet, in part because they manipulate

modest datasets, and in part because multithreaded CPUs have only recently been introduced to that market segment. As more data-intensive workloads (such as speech recognition) become common at the desktop, CMP systems will become increasingly attractive for that segment.

It is important to note that CMPs are a friendly target platform for applications that don't parallelize well. Communication between concurrent threads in a CMP can be an order of magnitude faster than in traditional SMP systems, especially when using shared on-chip caches. Therefore, workloads that require significant communication or synchronization among threads will pay a smaller performance penalty. This characteristic of CMP architectures should ease the programming burden involved in initial parallelization of the established code base.

CMP HEADING FOR MAINSTREAM ACCEPTANCE

A highly cost-efficient distributed computing system is essential to large-scale services such as those offered by Google. For these systems, given the distributed nature of the workloads, single-threaded performance is much less important than the aggregate cost/performance ratio of an entire system. Chip multiprocessing is a good match for such requirements. When running these inherently parallel workloads, CMPs can better utilize on-chip resources and the memory system than traditional wide-issue single-core architectures, leading to higher performance for a given silicon budget. CMPs are also fundamentally more power-efficient than traditional CPU designs and therefore will help keep power costs under control over the next few years. Note, however, that CMPs cannot solve the power-efficiency challenge alone, but can simply mitigate it for the next two or three CPU generations. Fundamental circuit and architectural innovations are still needed to address the longer-term trends.

The computing industry is ready to embrace chip multiprocessing as the mainstream solution for the desktop and server markets, yet it appears to be doing so with some reluctance. CMP parallelism is being introduced only when it is absolutely necessary to remain within a safe thermal envelope. This approach minimizes any significant losses in single-threaded performance, but it is unlikely to realize the full cost-efficiency potential of chip multiprocessing. A riskier bet on slower cores could have a much larger positive impact on the affordability of high-performance systems. Q

REFERENCES

1. Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., and Verghese, B. 2000. Piranha: a scalable architecture based on single-chip multiprocessing. *Proceedings of the 27th ACM International Symposium on Computer Architecture* (June), Vancouver, BC.
2. Transaction Processing Performance Council. Executive summary reports for TPC-C benchmark filings; http://www.tpc.org.
3. Hoelzle, U., Dean, J., and Barroso, L. A. 2003. Web search for a planet: the architecture of the Google cluster. *IEEE Micro Magazine* (April).
4. Ranganathan, P., Gharachorloo, K., Adve, S., and Barroso, L.A. 1998. Performance of database workloads on shared memory systems with out-of-order processors. *Proceedings of the Eighth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS VIII)*, San Jose, CA.
5. See Reference 3.
6. AMD competitive server benchmarks; http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8796_8800~97051,00.html.
7. Kongetira, P., Aingaran, K., and Olukotun, K. 2005. Niagara: a 32-way multithreaded SPARC processor. *IEEE Micro Magazine* (March/April); http://www.computer.org/micro.
8. Intel Corporation. Intel thread checker; http://developer.intel.com/software/products/threading/tcwin.
9. Seward, J. Valgrind; http://valgrind.kde.org/.
10. Dean, J., and Ghemawat, S. 2004. MapReduce: simplified data processing on large clusters. *Proceedings of OSDI,* San Francisco, CA.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

ACKNOWLEDGMENTS

The author thanks Wolf-Dietrich Weber and Christopher Lyle Johnson for their careful review of the manuscript.

LUIZ ANDRÉ BARROSO is a principal engineer at Google, where he leads the platforms engineering group. He has worked on several aspects of Google's systems infrastructure, including load balancing, fault detection and recovery, communication libraries, performance optimization, and the computing platform design. Prior to Google he was on the research staff at Compaq and DEC, where he investigated processor and memory system architectures for commercial workloads and co-architected the Piranha system. Barroso holds a Ph.D. in computer engineering from USC, and a B.Sc. and M.S. in electrical engineering from PUC-Rio, Brazil.