Challenges and Opportunities for Extremely Energy-Efficient Processors

In this point-counterpoint discussion, Trevor Mudge argues for the combination of near-threshold voltage processors with techniques such as boosting to address the needs of datacenter workloads. Urs Hölzle offers a cautionary note on the wisdom of giving up too much single-threaded performance to achieve energy-efficiency in large Internet service applications.

Extremely energy-efficient processors

Trevor Mudge, University of Michigan

Here, I make the case for improving the energy efficiency of processors suitable for data centers by operating the cores and caches at dramatically lower voltages—that is, at *near threshold*. Apart from dramatically reducing core and cache power, lowering core power has a multiplier effect that simplifies the use of other technologies. This, in turn, can significantly affect overall system power consumption. In particular, it simplifies 3D die stacking because of the reduced thermal design point (TDP). In addition, another source of energy savings, nonvolatile memory easily integrates into the stack to remove the need for local disks.

Near-threshold operation

Aggressively lowering supply voltage will be an important tool for building energy-efficient processors, particularly when sufficient parallelism is available. Over the past four decades, it has been possible to pack twice the number of transistors on a chip every two years without significantly increasing the cost to produce the chip—nonrecurring costs notwithstanding. Furthermore, Dennard's Scaling Theory says that power density should remain the same and device delay should decrease linearly with each new and smaller technology node. Since the 90-nm node, this free lunch has disappeared—supply voltages have hardly decreased. Consequently, power density is now growing almost exponentially. This creates a curious design dilemma: more gates can fit on a die, but they cannot actually be used because of strict power limits.

CMOS has no clear successor that might solve this dilemma and free up the dark silicon. Therefore, there is a strong case supporting the position that solutions to the power conundrum must come from new design styles, architectures, and enhanced devices, rather than the promise of radically new technologies becoming commercially viable.

The application of aggressive low-voltage operation fits these requirements. By lowering the supply voltage to a near-threshold level

Published by the IEEE Computer Society

(the turn-on voltage of a transistor), we can reduce the energy per operation tenfold. That's the good news. The bad news is that there is an accompanying tenfold performance loss and a decrease in reliable operation.[1]

Parallelism can help reclaim some of the performance. We see hope this will be a solution in the widespread emergence of multicore architectures. The limits imposed by power consumption have forced manufacturers to abandon frequency as a means of obtaining performance in favor of implied parallelism—multiple cores. However, parallelism only makes sense in those application domains where natural parallelism exists. It is not a universal solution. For example, most desktop applications have little parallelism, and where there is parallelism (for example, graphics), an accelerator makes more sense.[2] However, applications in Internet data centers are different from traditional benchmarks due to their scale and level of user involvement. Examples include Web search, Web mail, video hosting, and MapReduce, which emphasize unstructured data, user involvement, rich media types, and Web use as a platform, respectively. They are often distributed independent applications that emphasize throughput, or are parallelizeable.[3]

Caches

The near-threshold paradigm can reduce the core's power consumption, but the core is only part of the overall system that typically includes caches, memory, and disks. Examination of low-voltage operation of caches shows they reach their minimum energy level at a higher voltage and speed than the cores. The exact difference is a function of the workload. This observation leads to organizations with several cores (two to four) clustered on each L1 cache—the memory wall becomes much less daunting. As a secondary benefit, coherence traffic is reduced.

Boosting

Trading parallelism for power is effective, up to a point. In some situations, single-threaded performance might be necessary when a critical section blocks parallelism, or when a search deadline must be met. Proposed solutions have included switching from thin cores to a fat core when single-threaded performance is required. However, this solution requires more than one set of optimized binaries for the two distinct microarchitectures.

In boosting, a core operating at near threshold can be boosted to its nominal voltage, or even overclocked, to dramatically increase performance. The cluster architecture is a natural candidate, because, for example, one processor in a cluster of four can be boosted to four times its original clock rate while the others are turned off. The boosted machine sees a cache that can match its speed, is warmed up, and appears much bigger. There is also no need for separate binaries because the microarchitecture is unchanged. Figure 1 shows how boosting trades power for single-threaded performance.[4] Indeed, a subset of the cores can even be over clocked provided the TDP is not exceeded.

Device optimization

In addition to architectural techniques, near-threshold systems can be greatly improved through straightforward modifications and optimizations of the transistor structure and its fabrication process. This follows directly from the fact that commercially available CMOS processes are universally tailored to sustain the super-threshold trends forecasted by Moore's Law. For instance, simple modifications of a standard super-threshold device's channel-doping profile can reclaim almost a factor of two in performance.[1]

Of course, near-threshold operation only addresses part of the system—the processor and its immediate caches. It does not help with primary memory power consumption, which can be as much as 40 percent of a server. Much of this power loss is in the DRAM memory interface. An emerging solution is to use 3D die stacking, which allows low-latency, high-bandwidth interdie connections that consume very little power (milliwatts instead of watts).[5] The reduced power consumption of near-threshold cores and their caches works well with die stacking, because it reduces the TDP—a common concern for systems using 3D die stacking.

Enabling die stacking also lets us flatten cache hierarchies and avoid complex power-hungry out-of-order processors designed to

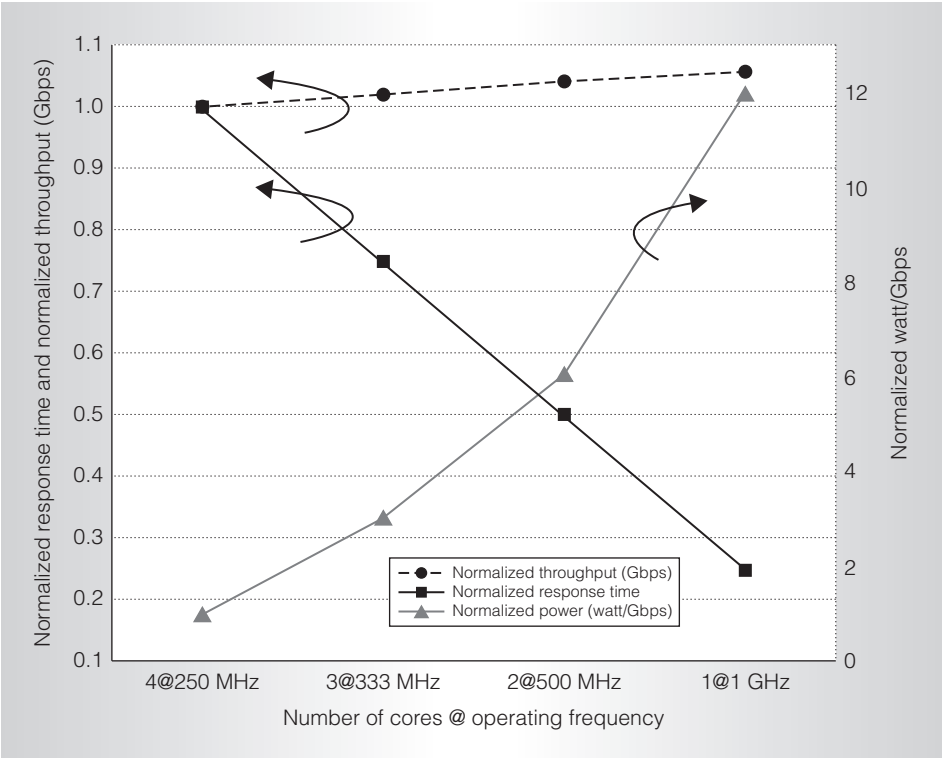

Figure 1. Trade-off of response time and power for a cluster-based architecture.

hide memory latencies. Finally, die stacking lets us add nonvolatile memories (Flash, phase change memories, or memristors) to the system. These are denser and cheaper than DRAM and can replace most of it, further reducing memory power.[6] Adopting nonvolatile memory at this point in the memory hierarchy opens up fresh opportunities for rethinking memory organization.

References

1. R. Dreslinski et al., ''Near-threshold Computing: Reclaiming Moore's Law through Energy Efficient Integrated Circuits,'' *Proc. IEEE,* vol. 98, no. 2, Feb. 2010, pp. 253-256.
2. G. Blake et al., ''Evolution of Thread-Level Parallelism in Desktop Applications,'' *Proc. IEEE/ACM 37th Int'l Symp. Computer Architecture,* ACM Press, 2010, pp. 302-313.
3. K. Lim et al., ''Server Designs for Warehouse Computing Environments,'' *IEEE Micro,* vol. 29, no. 1, Jan./Feb. 2009, pp. 41-49.
4. R. Dreslinski et al., ''Reconfigurable Multicore Server Processors for Low Power Operation,'' *Proc. Samos IX Workshop,* LNCS 5657, Springer, 2009, pp. 247-254.
5. T. Kgil et al., ''PicoServer: Using 3D Stacking Technology to Build Energy Efficient Servers,'' *ACM J. Emerging Technologies in Computing Systems* (JETC), vol. 4, no. 4, article 16, Oct. 2008.
6. D. Roberts, T. Kgil, and T. Mudge, ''Integrating NAND Flash Devices onto Servers,'' *Comm. ACM,* vol. 52, no. 4, Apr. 2009, pp. 98-103.

Trevor Mudge is the Bredt Family Professor of Engineering in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. His research interests include computer architecture, power aware computing, low-power DSP architectures, robust computing, and prototyping. Mudge has a PhD in computer science from the University of Illinois, Urbana-Champaign. He is a member of the ACM, Institution of Engineering and Technology, and British Computer Society, and is a Fellow of IEEE.

Direct questions and comments about this article to Trevor Mudge, tnm@eecs.umich.edu.

Brawny cores still beat wimpy cores, most of the time
Urs Hölzle, Google

At Google, we've been long-term proponents of multicore architectures and throughput-oriented computing. In warehouse-scale systems,[1] throughput is more important than single-threaded peak performance, because no single processor can handle the full workload. In addition, maximizing single-threaded performance costs power through larger die areas (for example, for larger reorder buffers or branch predictors) and higher clock frequencies. Multicore architectures are great for warehouse-scale systems because these systems provide ample parallelism in the request stream as well as data parallelism for search or analysis over petabyte data sets.

We classify multicore systems as *brawny-core systems*, whose single-core performance is fairly high, or *wimpy-core systems*, whose single-core performance is low. The latter are more power efficient. As a rule of thumb, CPU power decreases by approximately k^2 when CPU frequency decreases by k (the exact savings range from k up to a theoretical maximum of k^3 and depend, among other factors, on how much supply voltage can be reduced). Decreasing DRAM access speeds with core speeds can save additional power.

So why doesn't everyone want wimpy-core systems? Because in many corners of the real world, they're prohibited by law—Amdahl's law. Even though many Internet services benefit from seemingly unbounded request- and data-level parallelism, such systems aren't above the law. As the number of parallel threads increases, reducing serialization and communication overheads can become increasingly difficult. In a limit case, the amount of inherently serial work performed on behalf of a user request by slow single-threaded cores will dominate overall execution time.

Cost numbers used by wimpy-core evangelists always exclude software development costs. Unfortunately, wimpy-core systems can require applications to be explicitly parallelized or otherwise optimized for acceptable performance. For example, suppose a Web service runs with a latency of one second per user request, half of it caused by serial CPU time. If we switch to wimpy-core servers, whose single-threaded performance is three times slower, the response time doubles to two seconds and developers might have to spend a substantial amount of effort to optimize the code to get back to the one-second latency. Software development costs often dominate a company's overall technical expenses, so forcing programmers to parallelize more code can cost more than we'd save on the hardware side. Most application programmers prefer to think of an individual request as a single-threaded program, leaving the more difficult parallelization problem to middleware that exploits request-level parallelism (that is, it runs independent user requests in separate threads or on separate machines). Once cores become too slow to make this view practical for most applications, you know you're in trouble.

A few other effects splatter more mud on the shiny chrome of wimpy-core designs.

First, the more threads handling a parallelized request, the larger the overall response time. Often all parallel tasks must finish before a request is completed, and thus the overall response time becomes the maximum response time of any subtask, and more subtasks will push further into the long tail of subtask response times. With 10 subtasks, a one-in-a-thousand chance of suboptimal process scheduling will affect 1 percent of requests (recall that the request time is the maximum of all subrequests), but with 1,000 subtasks it will affect virtually all requests.

In addition, a larger number of smaller systems can increase the overall cluster cost if fixed non-CPU costs can't be scaled down accordingly. The cost of basic infrastructure (enclosures, cables, disks, power supplies, network ports, cables, and so on) must be shared across multiple wimpy-core servers, or these costs might offset any savings. More problematically, DRAM costs might increase if processes have a significant DRAM footprint that's unrelated to throughput. For example, the kernel and system processes consume more aggregate memory, and applications can use memory-resident data structures (say, a dictionary mapping words to their synonyms) that might need to be

loaded into memory on multiple wimpy-core machines instead of a single brawny-core machine.

Third, smaller servers can also lead to lower utilization. Consider the task of allocating a set of applications across a pool of servers as a bin-packing problem—each of the servers is a bin, and we try to fit as many applications as possible into each bin. Clearly that task is harder when the bins are small, because many applications might not completely fill a server and yet use too much of its CPU or RAM to allow a second application to coexist on the same server. Thus, larger bins (combined with resource containers or virtual machines to achieve performance isolation between individual applications) might offer a lower total cost to run a given workload.

Finally, even embarrassingly parallel algorithms are sometimes intrinsically less efficient when computation and data are partitioned into smaller pieces. That happens, for example, when the stop criterion for a parallel computation is based on global information. To avoid expensive global communication and global lock contention, local tasks can use heuristics that are based on their local progress only, and such heuristics are naturally more conservative. As a result, local subtasks might execute for longer than they would have if better hints about global progress were available. Naturally, when these computations are partitioned into smaller pieces, this overhead tends to increase.

So, although we're enthusiastic users of multicore systems, and believe that throughput-oriented designs generally beat peak-performance-oriented designs, smaller isn't always better. Once a chip's single-core performance lags by more than a factor two or so behind the higher end of current-generation commodity processors, making a business case for switching to the wimpy system becomes increasingly difficult because application programmers will see it as a significant performance regression: their single-threaded request handlers are no longer fast enough to meet latency targets. So go forth and multiply your cores, but do it in moderation, or the sea of wimpy cores will stick to your programmers' boots like clay. MICRO

Reference

1. L. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to Warehouse-Scale Machines,* Morgan Claypool, 2009.

Urs Hölzle is senior vice president of operations and Google Fellow at Google. His research interests include cluster computing, data centers, networking, and energy efficiency. Hölzle has a PhD in computer science from Stanford University. He is a Fellow of ACM.

Direct questions or comments about this article to Urs Hölzle, urs@google.com.