Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

THE EDITORIAL VIEW

The Future of Multicore Processors

By Tom R. Halfhill {12/31/07-02}

With the multicore era undeniably upon us, more talk is turning to the future implications of multicore processors. Of course, software development remains a big challenge, even provoking a recent article in *The New York Times*, of all places. (See *NYT 12/17/07*, "Faster Chips Are Leaving Programmers In Their Dust," by John Markoff.) But the discussion is equally spirited on the hardware side.

One debate is about symmetric versus asymmetric multiprocessing. Should all the cores on a multicore chip be identical, or should some be specialized for different tasks? Another debate questions the value of core-level multithreading. How many threads make sense? In many ways, these debates echo the classic RISC versus CISC arguments of the 1990s—simplicity versus complexity, efficiency versus expediency.

Let's review the symmetric-asymmetric debate first. With quad-core processors now penetrating the PC market, it's not hard to envision PC processors with 8, 12, 16, or more cores in the near future. (Some embedded processors passed those milestones years ago.) Already, the term "manycore" has been coined to describe chips that rise above the mere multicore category. The boundary between multicore and manycore isn't well defined, but in most discussions we've heard and read, "manycore" seems to describe chips with tens of cores. Beyond manycore, the next category is "massively parallel," which seems to describe chips with at least 64 cores. Naturally, these definitions are approximations.

Some CPU architects believe that as the number of processor cores increases, symmetry is the practical approach to large-scale designs. It's easier to slap down multiple copies of the same basic core than to create several different cores, even if specialized cores are better suited for their tasks. This argument foresees processor cores becoming general-purpose processing elements in the same way that ALUs or even NAND gates are today. Many superscalar processors use multiple copies of the same ALU, even when different ALUs might be more efficient. Logic-synthesis compilers often generate NAND gates in circuits where simpler gates might suffice. The general principle is that finishing an adequate design more quickly is better than crafting a highly optimized design more slowly. (Software engineers settled their version of this debate a long time ago, when they traded their assemblers for compilers.)

Symmetric-multiprocessor advocates have many facts on their side. They can cite time-to-market pressures, rising nonrecurring engineering (NRE) costs, and the ever-present desire to simplify an already complex design project. In addition, a symmetric multiprocessor is usually easier to program than an asymmetric multiprocessor, because all the cores are identical. And the advantages may extend beyond the processor cores themselves. For instance, if all the cores are symmetrical, their on-chip interconnects can be more uniform.

In contrast, an asymmetric multiprocessor may require more design effort, on-chip interconnects that are more complex, different software-development tools for the different cores, and perhaps even different teams of programmers with different skills. So, for numerous reasons, the cookie-cutter approach to multicore design is certainly attractive.

Make the Cores Fit the Chores

On the other hand, why settle for mediocrity? An asymmetric multicore processor can deliver superior throughput while conserving silicon and power. For example, network connectivity is almost universal in computing devices of all kinds, from desktop PCs to mobile phones. Why assign a suboptimal general-purpose core to packet processing when an optimized core can do the job better?

Some other specialized but commonplace tasks that come to mind are cryptography, digital-rights management, data compression/decompression, audio processing, video processing, media transcoding, and scanning a bitstream for telltale signs of malware (basically, a pattern-matching task). Sure, general-purpose cores can do all these things, but specialized cores can do them much faster, with fewer transistors, while burning less power.

Berkeley Design Technology's Jeff Bier made a similar observation in a recent edition of his online newsletter, Impulse Response. Bier notes that the architectures of CPUs and DSPs have been converging for several years, as general-purpose CPUs have added DSP extensions and DSPs have added general-purpose extensions. But in the multicore era, Bier foresees a divergence. CPU and DSP cores may revert to their original differentiation, because an asymmetric multicore chip can integrate both types of cores. No longer must one core do it all.

There is plenty of historical precedence for task specialization in microprocessors, even in "general-purpose" microprocessors. For instance, although an ALU can execute floating-point operations entirely in software, a hardware FPU is much faster. More to the point, many network processors have specialized engines for networking tasks in addition to their general-purpose cores. The highly integrated processors from Cavium Networks and Freescale Semiconductor are good examples. (See *MPR 8/27/07-01*, "Freescale's Multicore Strategy," and *MPR 7/16/07-01*, "Cavium Stalks Storage.")

Another factor in favor of asymmetric multiprocessing is that the processor cores in multicore chips tend to be simpler than those in single-core chips, so differentiation is easier to achieve. This is particularly true of manycore and massively parallel designs. Indeed, some massively parallel chips we've covered in are built with dozens or hundreds of very simple four-bit, eight-bit, or 16-bit cores. Designing a small, specialized core is much easier than designing a large, complex, general-purpose core that must be a jack of all trades.

Different Cores, Common Architecture

Tensilica CEO Chris Rowen makes another argument in favor of asymmetric multiprocessing: small special-purpose cores are even easier to create when engineers use highly automated design tools. Of course, Rowen's company happens to make such tools, but his argument is sound nevertheless.

Tensilica's Xtensa configurable processors let developers customize and extend the instruction-set architecture for application-specific tasks, greatly improving performance. Better yet, Tensilica's XPRES (Xtensa PRocessor Extension Synthesis) tool automatically generates custom instructions by analyzing ordinary C/C++ code. (See *MPR 7/12/04-01*, "Tensilica's Automaton Arrives.")

Rowen predicts that future multicore chips will integrate numerous specialized engines for data-plane processing. These engines will supplement, not replace, the general-purpose cores dedicated to control-plane processing. Although the data engines will be programmable—and therefore adaptable to changing conditions—their optimized microarchitectures will deliver performance on a par with hard-wired logic. To make these asymmetric multicore chips easily programmable, Rowen says, the various cores should share a common CPU architecture. This commonality will allow programmers to use the same software-development tools for all the cores.

Rowen's vision is compelling. Yes, it meshes with Tensilica's marketing, but asymmetric multiprocessing can work just as well with the x86 or any other CPU architecture. We won't be surprised if Intel's upcoming low-power x86 core (Silverthorne) someday appears on the same die with Intel's larger Core 2 microarchitecture. Looking further ahead, we can imagine a day when all chip designers use highly automated tools like Tensilica's to rapidly generate specialized microarchitectures based on a common architecture.

However, we can also imagine a day when most chip designers care no more for optimizing processor cores than most software programmers care for optimizing subroutines in assembly language. If replicating the same general-purpose core delivers adequate performance and finishes the project faster, then the greater expediency of symmetry may trump the greater efficiency of asymmetry. Remember, everyone pretty much agreed that RISC was more efficient than CISC, but CISC still thrives, partly because using an established software base is expedient.

A lot will depend on the evolution of processor-design tools. Rowen doesn't like my assembly-language analogy, because he believes that designing a specialized processor core with automated tools will be even easier than writing high-level software code. In some cases, Tensilica's XPRES tool already achieves this goal. It can automatically convert C/C++ application code into custom instructions for Tensilica's Xtensa processors, using Tensilica Instruction Extension (TIE) language. However, XPRES is a unique tool that works only with Xtensa configurable processors. A one-company, one-architecture solution from a company as small as Tensilica won't change the direction of the whole industry. Similar technology must become available for every CPU architecture. Design tools are moving in the same general direction as Tensilica's tools, but the progress is slow.

The Rise of Core-Level Multithreading

A similar debate rages about single threading versus multithreading at the hardware level. Single-threaded processor cores are simpler, smaller, and easier to replicate on chip, but multithreaded cores are superior for some applications. Adding multithreading to a core usually requires much less logic than replicating an entire core. The processor needs duplicate registers for each additional thread, plus control logic to select the appropriate register set.

Hardware multithreading was pioneered by companies as disparate as DEC (in Alpha server processors) and Ubicom (in embedded packet processors). (See *MPR 12/6/99-01*, "Compaq Chooses SMT for Alpha," and *MPR 4/21/03-01*, "Ubicom's New NPU Stays Small.") Intel has flirted with the same technology on a smaller scale under the brand name Hyper-Threading. (See *MPR 12/2/02-01*, "Intel's Hyper-Threading Takes Off," and *MPR 9/17/01-01*, "Intel Embraces Multithreading.")

Today, the leading flag-bearer for hardware multithreading is Sun Microsystems. Sun's UltraSPARC-T2 (Niagara 2) server processor has eight processor cores, each with eight-way multithreading, for a total of 64 simultaneous threads. Using virtualization, an UltraSPARC-T2 can run 64 different instances of an operating system at the same time. (At last, a processor that can simultaneously run every version of Unix!)

In the opposite corner, to some degree, is IBM. The POWER6 processor is a reincarnation of the classic RISC speed demon, faster than a speeding locomotive. IBM hopes to use its world-class fabrication technology to push the POWER6 to clock frequencies in the 6.0GHz range. Although IBM is not completely averse to hardware multithreading—the POWER6 core is dual threaded—IBM disdains the larger-scale threading that Sun espouses. In IBM's view, large-scale threading is suitable for web servers and other applications in which many users intermittently clamor for attention. But IBM says the UltraSPARC-T2 will be less effective than POWER6 in computing applications requiring high single-thread performance. (See *MPR 12/10/07-01*, "Server Processors: Chapter 2007 [Part 2].")

Sun argues that hardware multithreading ameliorates the perennial memory-latency problem, because a thread waiting for data simply yields to another thread that's ready to go. Nary a clock cycle goes to waste. Cache misses become an opportunity, not a problem. In a brazen bid for the speed-demon crown, Sun multiplies the UltraSPARC-T2's 64 threads by the 1.4GHz maximum clock rate of each core to claim an aggregate clock frequency of 89.6GHz.

When Sun's multithreading strategy works, it's great. However, it assumes that another thread is always ready to go. In some applications, thread-level parallelism is as elusive as the instruction-level parallelism that mesmerized the architects of wide-issue superscalar processors in the 1990s. When even the best four- or five-way superscalar designs were found to average only about 1.5 instructions per cycle over time, the early enthusiasm for wider pipelining soon waned. Today, few processors venture beyond two- or three-way superscalar execution. The incremental performance improvement isn't worth the significant additional design complexity.

Nevertheless, Sun is onto something with Niagara. The UltraSPARC-T2 sets a new standard for integration in server processors. In addition to eight multithreaded SPARC cores, the UltraSPARC-T2 has eight cryptography accelerators, four FB-DIMM memory controllers, a dual-threaded 10Gb/s Ethernet controller, and an eight-lane PCI Express controller—all on one chip. That level of integration is rarely seen outside of system-on-chip embedded designs.

A Few Predictions

In my opinion, the UltraSPARC-T2 is the most visionary multicore design to date. Although *MPR* has covered massively parallel processors with as many as 4,096 cores, those chips are highly specialized. The UltraSPARC-T2 is a general-purpose microprocessor (within the realm of server processors), yet it embodies all the concepts discussed here. It's a large-scale multithreaded, multicore, system-on-chip microprocessor that does symmetric and asymmetric multiprocessing at the same time.

Microprocessors of the future will look a lot like the UltraSPARC-T2. They will have many general-purpose processor cores supplemented by special-purpose cores, on-chip peripherals, and integrated I/O interfaces. And they will be multithreaded at the core level, because hardware multithreading is relatively cheap and doesn't require application programmers to write explicitly multithreaded code. Instead, the microprocessor manages the threading transparently. (However, programmers may still choose to write explicitly multithreaded code for other reasons.)

Ideally, all chip designers of the future will use development tools like those championed by Tensilica. With relatively little effort—more akin to software programming than hardware design—engineers will rapidly generate dozens of general-purpose and special-purpose processor cores. These cores will share the same basic CPU architecture, but their microarchitectures will be optimized for specific tasks. Programmers will write code for all the cores in the same programming language.

For explicitly parallel processing, I think programmers will avoid explicit software-level multithreading because of the problems it entails. (See *MPR 4/30/07-02*, "The Dread of Threads.") Instead, they will likely use technology similar to that promoted by RapidMind. (See *MPR 11/26/07-01*, "Parallel Processing For the x86.")

If my predictions are wrong, CPU architects will take the easy way out. They will simply replicate the same basic general-purpose processor cores on their multicore chips, even if those cores aren't optimal for all tasks. Hardware design will be as sloppy as much software design is today. But even if that design philosophy is popular in the short run, I think it will eventually lose to the more optimal approach—especially when other avenues to higher performance reach their inevitable dead ends. ◇

Tom R. Halfhill