PARALLELISM VIA MULTITHREADED AND MULTICORE CPUS

Angela C. Sodan, Jacob Machina, Arash Deshmeh, Kevin Macnaughton, and Bryan Esbaugh, *University of Windsor, Canada*

Multicore and multithreaded CPUs have become the new approach to obtaining increases in CPU performance. Numeric applications mostly benefit from a large number of computationally powerful cores. Servers typically benefit more if chip circuitry is used for maximizing throughput via multiple threads per core.

This survey compares multicore and multithreaded CPUs currently on the market and examines the underlying design decisions, performance, power efficiency, and software concerns in relation to application and workload characteristics.

Traditionally, CPUs have doubled in performance roughly every 18 months because designs grew more complex and CPU clock speeds increased with advances in chip fabrication technology. However, there are barriers to further significant improvements in operating frequency due to voltage leakage across internal chip components and heat dissipation limits.

Moore's law—which projects that the density of circuits on chip will double every 18 months—still applies and provides hardware designers with the ability to add more complexity to a chip.[1] This will remain true until CPUs reach the hard physical limits of circuit density.

In the past, developers used additional capacity to develop superscalar CPUs with replicated execution units and deep pipelines to exploit instruction-level parallelism. However, they only harvested about 25 percent of the additional chip space that became available per year by adding new architectural features.[2] Moreover, the performance gap between processors and memory limits the gains possible from further increasing processor frequency. Thus, the design direction currently employed for performance increases uses available chip space for multithreaded and multicore CPUs. These designs support multitasking via parallel programs or running several applications concurrently.

Designers first introduced multithreaded CPUs, which employ hardware-level context switching between threads, to reduce the idle time of resources in complex superscalar processors. Shortly after this, designers integrated more than one processor core onto a single chip, and we now have eight-core processors. Assuming that Moore's law holds, we expect a doubling of the number of cores on chip every two years, leading to CPUs of 16 or more cores in the near future.

DESIGN SPECTRUM

Multithreaded and multicore CPUs both exploit concurrency by executing multiple threads, although their designs target different objectives. Multithreaded CPUs support concurrent thread execution at the more fine-grained instruction level, aiming at better utilizing the resources of CPUs by issuing instructions from multiple

 Published by the IEEE Computer Society

threads. Multicore CPUs achieve thread concurrency at a higher level, focusing less on utilization per core and aiming at scalability via replicating cores. These CPUs are often called *chip multiprocessors* (CMPs). Most recent CPU and graphics processing unit designs, like the Sun UltraSPARC T2, IBM Power6, Intel Xeon, ATI RV770, and Nvidia GT200 combine both options and have multiple multithreaded cores.

Multithreaded cores

All multithreaded cores keep multiple hardware threads on-chip and ready for execution. This is necessary to make fine-grained switching between threads feasible and to minimize context-switch costs by hardware-level multiplexing.

Each on-chip thread needs its own state components, such as the instruction pointer and other control registers. Thus, the number of on-chip threads determines the number of required replications of state components and subsequently the maximum degree of hardware-supported concurrency and execution-unit saturation. More threads also improve the possibilities for hiding memory-access latencies or stalls from branch mispredictions. The Intel Xeon only needs 5 percent more chip space to support a second hardware thread.[3] Cost growth is approximately linear up to at least eight threads, but it is clearly superlinear thereafter.[4]

The number of on-chip threads per core typically supported by commercial processors ranges from two in Intel's Xeon to eight in Sun's UltraSPARC T2. One extreme example, the 128 threads in the Tera/Cray MTA, represented one of the first practical but not commercially successful designs. The processor needed the large number of threads to hide memory-access latencies and compensate for the lack of a cache in its architecture. Designers used the same technique in the massively multithreaded Nvidia GT200 GPU, which opts for minimal caches in favor of additional computational resources.

Table 1 shows that manufacturers use a variety of approaches to switching between threads per core, which range from alternating between the threads to actually issuing instructions from several threads each cycle.[5] Most current CPUs employ the latter approach, which usually is called *simultaneous multithreading* (SMT), and which Intel calls *hyperthreading technology* (HTT).[3] SMT dispatches predecoded instructions from only a subset of the on-chip threads per cycle. The number of threads in this subset also impacts the execution units' utilization, particularly if the threads complement each other's use of these units. However, at present, no commercially available CPU issues from more than two threads per core and per cycle.

Multicore CPUs

Hardware multithreading per core has limited scalability—bound by the saturation point of the execution units and the cost of additional threads—whereas multicore CPUs promise more potential for scalability. For a summary of current multicore CPUs, see Table 1 (more detailed version available at http://cs.windsor.ca/~acsodan/cpu-tables.htm). Most early multicore chips were constructed as a simple pairing of existing single-core chip designs, as in the Itanium dual-core. These chips retained much of their predecessors' architecture, replicating only the control and execution units and sharing the remaining units per chip: cache, memory controller, secondary processing units like floating-point units (FPUs), cooling components, and off-chip pins. However, sharing has disadvantages regarding contention of the shared resources.[6]

The number and selection of integrated components on-chip is an important design decision.

Development trends indicate a move toward replicating additional on-chip components—such as memory controllers and caches—which may be private or shared. For example, the IBM Power6[9] and AMD Opteron each have private L2 caches and share multiple memory controllers.

Component integration

The number and selection of integrated components on-chip is an important design decision. Possible components to include on-chip are memory controllers, communication interfaces, and memory. Placing the memory controller on-chip increases bandwidth and decreases latency, which explains the recent trend toward integrating this component.

Some designs support multiple integrated memory controllers to make memory-access bandwidth scalable with the number of cores, including both the IBM Power6 and Sun UltraSPARC T2. Integrating a GPU core on-chip is another option announced for next-generation CPUs. A similar approach is already used in the embedded and mobile markets, which frequently combine both a general-purpose core and digital-signal processor core on a single chip.

IBM's Blue Gene/P[10] system relies on a highly integrated system-on-a-chip design that features four cores, five network interfaces, two memory controllers, and 8 Mbytes of L3 cache, allowing the system to scale to hundreds of thousands of processors. As another example, the UltraSPARC T2 integrates memory controllers, I/O, security functions, and an advanced network interface.

Table 1. Comparison of features for current commercial multicore CPUs.

Vendor	Product	Cores	Threads		Clock (GHz)	Power (watts per CPU)	Special features	On-chip interconnect	L2 size per chip (Mbytes)❍	L2 allocation	L3 size (Mbytes)❑
			Per core on-chip / executing	Switching approach							
AMD	Opteron (3rd generation)	4, 6	N/A	N/A	1.7-3.1	40-105	IMC, 128-bit FPU per core, dual PM*	Crossbar	2, 3	Private	2, 6
AMD	Phenom II	3, 4	N/A	N/A	2.4-3.2	65-125	IMC, 128-bit FPU per core, dual PM*	Crossbar	1.5, 2	Private	6
AMD	Turion X2	2	N/A	N/A	1.6-2.4	18-35	IMC	Crossbar	1, 2	Private	N/A
Intel	Pentium Dual Core	2	N/A	N/A	1.7-2.7	65	IR 4	Front-side bus**	2, 4	Dynamic	N/A
Intel	Core 2 Duo Family	2	N/A	N/A	1.8-3.3	65	IR 4	On-chip bus	2, 3, 4, 6	Shared	N/A
Intel	Core 2 Quad	4	N/A	N/A	2.0-3.0	95-105	IR 4, dynamic PM	On-chip bus / front-side bus**	4, 6, 8, 12	Shared per 2 cores	N/A
Intel	Itanium (9000 series)	2	2 /1	Blocked+	1.4-1.66	75-104	VLIW, IR 6	Direct pathways	I: 2 D: 0.5	Private	4, 6, 9, 12 per core, private
Intel	Xeon (7400 series)	4, 6	2 /2	SMT	2.13-2.66	50-130	IR 4, dynamic PM	On-chip bus	6, 9	Shared	8, 12, 16
Intel	Core i7	4	2/2	SMT	2.66-3.33	130	Triple channel IMC	Crossbar	1	Private	8
IBM	Power5	2	2/2	SMT	1.5-1.9	Unpub-lished	IR 5, IMC	Crossbar	1.875	Shared	36 off-chip
IBM	Power6	2	2/2	SMT	4.7-5	Unpub-lished	IR 7, 1 decimal, 2 binary FPUs per core	On-chip bus	8	Private	32 off-chip
IBM	Cell BE, PPE	1	2/2	SMT	3.2	110+	General purpose	Ring bus	0.5	N/A	N/A
IBM	Cell BE, SPE	8	N/A	N/A	3.2	110+	Simplified for SIMD support	Ring bus	N/A	N/A	N/A
Sun	UltraSPARC T1	4, 6, 8	4/1	Interleaved++	1.0-1.2	72-79	1 FPU per chip, IMC	Crossbar	3	Shared	N/A
Sun	UltraSPARC T2	4, 6, 8	8/2	Parallel interleaved△	1.0-1.6	95-123	IMC and INC, crypto unit (per core), SOC, 1 FPU per core	Crossbar	4	Shared	N/A
Sun	UltraSPARC IV+	2	N/A	N/A	1.5-2.1	90	IR 4, IMC	On-chip bus	2	Shared	32 off-chip
Sun	Sun/Fujitsu SPARC64 VII	4	2/2	SMT	2.5	135	IR 4, hardware barrier	On-chip bus	6	Shared	N/A
Sun	Rock	16	2/2	SMT	2.1	250	4-core clusters, IR 4, aggressive speculation, HTM, 2 FPUs per cluster	Direct pathways / crossbar (among clusters)	2	Shared	16 off-chip
Specialized	Tilera TILE 64	64	N/A	N/A	0.5-0.9	15-22	Simple cores, no FPUs	Multilink mesh	4†	Shared†	N/A
Specialized	ARM Cortex-A9 MPCore	2, 4	N/A	N/A	1	<1	Ultrasmall, SOC, ultra-low-power	Multilevel bus	2	Shared	N/A
Specialized	ATI RV770††7	10	> 1,000‡ / 10	Interleaved	0.75	160	Simplified for SIMD, 80 FPUs per core	Crossbar	>256 Kbytes‡	Shared	N/A
Specialized	Nvidia GT200††8	30	1,024 / 8-16	Interleaved	1.295	236	Simplified for SIMD, 10 FPUs per core	Crossbar	256 Kbytes	Shared	N/A

IMC = integrated memory controller, IR *n* = issue rate up to *n* instructions per cycle, PM = power management, VLIW = very long instruction word, INC = integrated network controller, SOC= system on a chip, HTM = hardware transactional memory

❍ The L1 cache is private per core in all processors (size ranges from 16 Kbytes to 128 Kbytes)
For private L2 caches, the total size is obtained by multiplying the size per core by the number of cores

❑ The L3 cache, if present, is shared

* Separate power management for cores and memory controllers

** For data exchange; otherwise not relevant since no integration of memory controller and network controller

\+ Blocked multithreading switches to another thread only if the currently executing thread stalls

++ Interleaved multithreading switches among ready-to-run threads every cycle

△ Two execution pipelines per core, each serving one thread per cycle

† L2 caches of other cores can be aggregated per application, accessible at L2-like speed

†† The ATI RV770 is used on the Radeon HD 4870, and the NVIDIA GT200 on the GeForce GTX280

‡ Estimates, no vendor specifications available

Shared versus private caches

Aside from concurrency, caches are the most important feature for enhancing modern CPU performance because of the gap between CPU speed and memory-access times. The dominant approach to mitigating this gap exploits available chip space to provide more on-chip cache memory. Some CPU architectures choose a completely different path and do not employ a cache at all, hiding memory-access latencies via multithreading, as in the Tera/Cray MTA, or by using high-speed direct-addressed memory, as in the Cell SPE.

The organization of the cache memory is a major consideration. Most current multicore chip designs have a private L1 cache per core to reduce the amount of contention for this critical cache level. If the core supports multiple hardware threads, the L1 cache is shared among the threads per core. The assignment of the L2 cache in multicore designs varies. The L2 cache may be either private and dedicated to each core, or shared between cores. The L3 cache was historically off-chip and shared, but newer designs such as the Intel Itanium and quad-core AMD Opteron feature on-chip L3 caches.

Whether shared or private caches are more beneficial depends not only on tradeoffs regarding the use of chip space but also on the application characteristics. Shared caches are important if threads of the same application execute on multiple cores and share a significant amount of data. In this case, a shared cache is more economical because it avoids multiple copies of data and cache-to-cache transfers. However, shared caches can impose high demands on the interconnect.[6]

Software threads that do not share much data might compete for the cache. This makes it difficult to predict the service to each thread as it depends on details of memory-access patterns and memory-access locality as well as on the system load. Private caches constitute an easy solution to isolating performance and guaranteeing predictable service.

As a more flexible approach, a hybrid design provides different numbers of cache banks that can be allocated as shared or private, depending on the cache needs of the currently running threads. This approach can support threads that share data and threads that do not. The hybrid design can be refined to dynamic proportional partitioning, as proposed in recent research.[11] This makes it possible to provide a level of service for each core equal to that of a single-core chip with the corresponding amount of provisioned cache resources.

Shared versus private hardware-thread resources

In contrast to multicore designs that tend to replicate most resources, sharing is the dominant approach in hardware multithreading. However, some level of replication and partitioning is still necessary.[3] Replication is essential for execution units that might be subject to high contention. Static or dynamic partitioning of a resource guarantees each thread exclusive access to its share, which constitutes a simple solution to provide fair and independent progress of thread execution.

For example, some designs apply partitioning to instruction buffers. Static partitioning creates strict boundaries, whereas dynamic partitioning can choose boundaries flexibly, while keeping a minimum share for each of the executing threads. Sharing allows greater flexibility in resource usage, but adds extra potential for contention and may need some mechanism to prevent monopolization. Most multithreaded designs use a combination of sharing, replication, and partitioning. The design decision is based on the degree of contention among threads for a particular resource, fairness considerations, and cost.

A hybrid design provides different numbers of cache banks that can be allocated as shared or private, depending on the cache needs of the currently running threads.

Fault tolerance

Dynamic partitioning of cache or other resources can be extended to deal with hardware faults more likely to occur with higher circuit density.[12] Faults can result in electrical noise or minor permanent defects in silicon, potentially spreading from individual components and resulting in the entire chip failing. Some CPUs disable faulty cores at fabrication time to increase yields, as a form of static partitioning. Additionally, fault tolerance may comprise dynamic configurability and partitioning of replicated and separable units, such as multiple interchip interconnects and memory controllers in addition to multiple cache banks. This leads to supporting different degrees of isolation versus sharing and separation of working components from faulty ones. Such solutions greatly increase overall availability and provide graceful performance degradation in case of faults.[12]

Interconnects

Another important feature that impacts multicore chip performance is the communication among different on-chip components: cores, caches, and—if integrated—memory controllers and network controllers. Initial designs used a bus as in traditional multiple-CPU systems. The trend has now shifted to a crossbar or other advanced mechanisms to reduce latency and contention. For instance, AMD CPUs employ a crossbar, and the Tilera

TILE64 implements a fast nonblocking multilink mesh. However, the interconnect can become expensive: An 8 × 8 crossbar on-chip can consume as much area as five cores and as much power as two cores.[6]

With only private caches on-chip, data exchange between threads running on different cores historically necessitated using the off-chip interconnect. Shared on-chip caches naturally support data exchange among threads running on different cores. Thus, introducing a level of shared cache on-chip—commonly L2, or in the more recent trend, L3—or supporting data-exchange shortcuts such as cache-to-cache transfer helped reduce off-chip traffic. However, more on-chip cache levels force the on-chip interconnect to support even greater complexity and bandwidth requirements.

An important consideration for per-application performance is that serial programs cannot exploit chip concurrency.

As data processing increases with more thread-level parallelism, demands also typically increase on the off-chip communication fabric for memory accesses, I/O, or CPU-to-CPU communication. To address these requirements, off-chip communication is trending from bus-based to packet-based, point-to-point interconnects. AMD first implemented this concept as HyperTransport, followed by Intel's QuickPath Interconnect. The off-chip interconnect and data-coherency support also impact the scalability of multiple CPU servers.

Specialized designs

Some multicore processors are tailored to very specific workloads. The Azul Vega series of compute appliances uses multicore chips with up to 48 cores, each including special execution units designed to increase performance of Java operations. Designers optimized the Tilera TILE64's CPU for data processing with 64 low-powered simple processing cores. Its increased dataflow capacity makes it well suited for embedded systems, such as telecommunications routers. Although the IBM Cell was originally designed for gaming, it also proved useful for data processing applications in bioinformatics and astrophysics.

GPUs represent an extreme example of specialized multicore design. Modern GPUs have 10 or more cores, each optimized for SIMD data processing done via hundreds or thousands of simplified threads per core. This makes them suitable for highly numeric processing such as video rendering, genomics, scientific modeling, or cryptography.

Core complexity versus number of cores

Traditional CPU optimizations sought to increase the serial execution speed of a single thread, adopting techniques such as out-of-order execution, dynamic branch prediction, and longer pipelines for higher clock rates. The availability of thread-level parallelism in addition to instruction-level parallelism raises the major design decision of the extent necessary to simplify traditional CPU designs to allow the dedication of more circuitry to concurrency.

Examples include Sun's UltraSPARC T1, which reduces the number of on-chip FPUs, and Intel's Atom, which removes out-of-order execution. In the extreme case of IBM's Cell, this leads to a greatly reduced instruction set and no dynamic branch prediction or instruction reordering.

Other chips increase complexity to maximize the per-core performance, such as the POWER6 chip, which offers highly optimized integer units and FPUs, including a decimal FPU. Mainframe processors additionally need to support heavy transaction processing; thus, IBM's z10 extends the POWER6 architecture with advanced branch prediction and cache management.

Greater issue-width also increases peak performance, with the POWER5 architecture issuing five instructions per cycle and AMD chips issuing only three. CPUs that focus on per-thread performance also generally have much higher clock rates than those that focus on many-threaded support. This contrast can be seen in the 5.0-GHz clock rate of the IBM POWER6 and the 3.73-GHz rate of the Pentium Extreme Edition, compared to the highly multithreaded UltraSPARC T2, which has a core frequency of 1.6 GHz.

However, using extra chip space to enhance per-thread performance results in nonlinear gains, with experience suggesting that performance only doubles when complexity is quadrupled.[1] The "Performance of Multithreaded and Multicore CPUs" sidebar provides performance comparisons for different designs using standard benchmarks.

An important consideration for per-application performance is that serial programs cannot exploit chip concurrency. Even in parallel programs, some parts of the algorithm must run sequentially, and Amdahl's law states that the maximum speed of an algorithm is determined by the percentage of its sequential part. Balancing core complexity and number of cores, while considering diminishing returns from higher per-thread performance, can be formalized as an extension of this law,[1,13] and that, along with other considerations, leads to the following conclusions:

- Larger numbers of simple cores are preferable as long as the application's serial part is very small; otherwise, more complex cores have proven beneficial.[13]

- Benefits can shift toward more complex cores due to growing chip-space demands for the interconnect among larger numbers of cores and limited application scalability due to lack of sufficient parallelism, synchronization overhead, or load imbalance.
- Applications that can exploit much of the theoretical peak performance—such as floating-point-intensive or highly instruction-parallel numeric applications—might experience higher returns from added complexity than typically expected.

However, in addition to per-application performance, the overall workload must be considered as well. Per-application performance is important if the load consists of only a few applications or if there are performance-critical applications. Otherwise, good utilization can easily be obtained from workloads with many serial jobs and parallel applications scaled to only a fraction of the number of cores.[1] This can lead to high throughput, and—via reduced waiting times—also to good turnaround times, which are commercial servers' design goals, such as database and webservers.

To strike a balance between per-thread performance and throughput, the former might be enhanced if more chip resources can be allocated dynamically, such as for speculative execution. This is likely to benefit applications with many data dependencies and cache misses.

Sun's preproduction Rock processor implements this idea by optionally using the two hardware threads per core to execute one application thread.[14] A simpler approach, already applied in some multithreaded CPUs, allocates partitioned resources to one thread if run in single-task mode, as implemented in the Intel Xeon and Pentium Extreme Edition.[3]

As another possibility, chip designs can incorporate some diversity regarding the cores' complexity, such as in IBM's Cell processor. A few higher-complexity cores might run sequential parts of demanding applications. Though they are not yet commercially available, research

PERFORMANCE OF MULTITHREADED AND MULTICORE CPUS

Single-thread performance: Systems based on Intel Core i7 (ASUSTeK i7-965 results of Feb. 2009) processors rank highest in the SPECfp2006 and SPECint2006 benchmarks (http://spec.org), achieving 68 percent higher integer speed and 84 percent higher floating-point speed (supported by its memory controller), compared to AMD's Opteron (HP Opteron 2384 results of Dec. 2008). Additionally, Intel's Core i7 currently has the best score for SPECfp and SPECint throughput.

Regarding throughput for the numeric SPECfp applications, the Opteron is only slightly better using highly optimized code than Sun's UltraSPARC T2 (Fujitsu T5120 results of Jan. 2009), but has a 26 percent advantage using standard optimizations. However, the UltraSPARC T2 (Sun T5440 results of Oct. 2009) outperforms the Opteron (HP Opteron 8393 results of May 2009) in the multithreaded SPECweb2005 benchmark for webserver throughput and response times, by serving 36 percent more Web requests over the same time span.

Considering different loads, simulation studies with database applications,[1] specifically OLTP and DSS, showed up to 40 percent shorter response times for the POWER5 compared to the UltraSPARC T1 if serving an unsaturated load. However, for saturated loads, the UltraSPARC T1 achieved up to 70 percent greater throughput.

Multiple-thread performance: The benefits from multiple hardware threads partially depend on whether the application employs multithreading or multiprocessing. Enabling dual-thread hyperthreading on the Intel Xeon processor resulted in a 33 percent performance gain versus single-thread execution[2] for the OpenMP version of the NAS[3] CG benchmark. By comparison, decreases in performance were observed for the multiprocess version of the NAS FT benchmark. The OpenMP version of the FT benchmark suffered 8 percent performance loss versus single-thread execution,[2] whereas the standard multiple-process version of FT suffered a larger loss at 50 percent, which was mostly attributed to memory contention from intensive interprocess communication.[4]

Multiple-core performance: The AMD Opteron dual-core processors example demonstrated performance gains from multiple cores, showing 37 percent improved performance utilizing the second core when measured by the standard multiple-process NAS CG and FT benchmarks.[5] The same study also showed that one dual-core chip performed only 5.8 percent slower in the CG benchmark, and only 9 percent slower in the FT benchmark, than two chips using a single core, while being much more power- and cost-efficient. Another study with pure multiprocess applications running on large clusters with up to 4,096 CPUs obtained benefits of between 20 and 50 percent from using a second core.[6]

References

1. N. Hardavellas et al., "Database Servers on Chip Multiprocessors: Limitations and Opportunities," *Proc. 3rd Biennial Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 79-87.
2. M. Curtis-Maury et al., "Integrating Multiple Forms of Multithreaded Execution on Multi-SMT Systems: A Study with Scientific Applications," *Proc. Int'l Conf. Quantitative Evaluation of Systems* (QUEST 05), IEEE Press, 2005, pp. 199-208.
3. D. Bailey et al., *The NAS Parallel Benchmarks 2.0*, tech. report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995; www.nas.nasa.gov/News/Techreports/1995/1995.html.
4. T. Leng et al., "An Empirical Study of Hyper-Threading in High Performance Computing Clusters," *Linux HPC Revolution*, 2002.
5. S.R. Alam et al., "Characterization of Scientific Workloads on Systems with Multi-Core Processors," *Proc. IEEE Int'l Symp. Workload Characterization* (IISWC 06), IEEE Press, 2006, pp. 225-236.
6. R. Brightwell, K.D. Underwood, and C. Vaughan, "An Evaluation of the Impacts of Network Bandwidth and Dual-Core Processors on Scalability," *Proc. Int'l Supercomputing Conf.* (ISC 07), 2007, pp. 1-12.

indicates that processors comprising many simplified cores and a few high-performance cores could provide the greatest total processing power for a given chip space and power budget.[13,15] However, these CPUs may be more difficult to design and program.

Cost and power consumption

Performance no longer dominates design objectives: Chip fabrication costs and fault tolerance, power efficiency, and heat dissipation have all become critical considerations.

As cores are simplified, power consumption decreases linearly,[1] which is a major advantage of multicore CPUs. Increased power efficiency and reduced heat generation permit the integration of more cores into a single CPU, with the tradeoff that the power budget for the interconnect increases with the number of cores.[6] Power usage affects the choice between multicore designs and single-core multithreaded designs: The former are more power-efficient, but hybrid designs with multiple SMT cores achieve nearly the same performance per watt as pure CMP designs.[4] Multicore CPUs also provide more options for power management because CMP cores can be individually power-tuned by being powered off or run at a lower frequency when system load is light.[1] Power tuning is critical in mobile computing, but servers can also benefit greatly.

In the future, exponential growth in CPU performance will primarily be obtainable from more hardware threads and cores.

From an overall system perspective, increasing electricity costs demand more power efficiency from processors and other system components, with the additional benefit of reduced cooling costs. Though the CPU accounts for only 25 to 45 percent of the power a server consumes,[16] projected electricity costs for a four-year term approach the system's purchase price. In the case of high-performance computing machines, building customized cooling solutions can cost as much as the computer itself.[17] Additionally, the reduced power consumption permits higher rack density in server rooms.

Optimizing for performance per watt and per dollar also enables massively scalable architectures. An extreme example is IBM's Blue Gene/L or the Blue Gene/P, which runs at 850 Mhz. Designed for simplicity, low fabrication cost, high integration, and scalability, the Blue Gene/P architecture reached 450 teraflops by employing 40,960 CPUs. The fastest, most power-efficient[17] architecture—according to the Top500 (http://top500.org) list of June 2009—per parallel application is IBM's hybrid QS22/LS21, used in the Roadrunner supercomputer at Los Alamos National Labs. This architecture also held the highest rank in the Green500 (http://green500.org) list of November 2008. These top rankings were made possible by employing the power- and cost-efficient Cell as the main compute processor.

THE SOFTWARE CHALLENGE

In the future, exponential growth in CPU performance will primarily be obtainable from more hardware threads and cores. However, hardware concurrency can only be exploited with multiple serial programs or with parallelized applications.[18] Because of the limited opportunities for further per-thread performance enhancements, serial code should be carefully optimized. Throughput can be improved even on personal computers with serial programs, if the additional cores run operating system or background tasks such as security software or virus scans, or are used to support virtualization. However, these arguments only hold for small numbers of cores, whereas the trend is toward many-core CPUs.

Server software may already be multithreaded for higher throughput by interleaving requests and potentially exploiting multiple CPUs. However, most commodity software is not prepared for concurrency.[18,19] Possibilities for automatically extracting parallelism are currently limited, and parallelism typically must be expressed explicitly. Thus, Herb Sutter considers changing toward parallel programming for commodity machines to be the next revolution after the introduction of object-oriented programming.[18] Writing correct and efficient parallel programs is a major challenge that calls for better tools and more abstract programming models to make thread programming safer and more convenient. Solutions can draw upon experiences obtained in high-performance computing, with the greatly enlarged market providing the stimulus for further improvement in HPC techniques.[20]

Widely used in HPC, commodity software developers could adopt the OpenMP shared-memory programming model (http://openmp.org/wp). Another promising direction is transactional memory,[21] which borrows the transaction concept from databases and simplifies data-access coordination through automatic checkpointing and rollback mechanisms. Sun's Rock[14] is the first CPU that supports this model in hardware for common cases. Rather than investing additional time for parallel-software development, a more economical approach uses preparallelized compilers and libraries like the Basic Linear Algebra Subprograms (BLAS) library.[22]

The need for better tools and programming models also affects HPC. Currently, even if data could be shared, many parallel programmers exclusively use processes,

despite the performance benefits of employing software multithreading on shared memory symmetric multiprocessing (SMP) nodes. HPC clusters with many-core nodes may require using hybrid thread/process programming models for higher efficiency and scalability. Fortunately, users tend to prefer multithreading and may find the additional step toward incorporating it easier than the initial step taken from serial to multiprocessing.[23] Moreover, HPC applications will need to exhibit a higher degree of parallelism than before to exploit hardware concurrency offered by multicore CPUs. This may only be possible to a certain extent as application scalability is limited unless problem sizes are increased.[20] Another limiting factor is that the performance benefit of additional cores is less than that of additional CPUs, except when threads share data. Thus, multicore CPUs are not the new SMP.[20]

The software challenge also affects commodity compilers that may need to address simplified or specialized cores like in the Cell or a GPU. Whereas in the past, the hardware itself to a large extent extracted instruction-level parallelism, simplified cores now demand more compiler effort for reordering instructions, inserting static branch prediction hints, and vectorizing data processing to exploit SIMD instructions.

Regarding the operating system, traditional CPU schedulers needed modifications to accommodate the heterogeneity and performance differences in the hierarchy of CPUs, cores, and hardware threads. Additionally, research has shown that scheduling with the goal of minimizing resource contention is important if the machine is fully loaded.

The challenge then is to match applications with complementary resource needs whenever resources are shared—such as moderately cache-intensive applications if caches are shared or integer- and floating-point-dominant applications if FPUs are shared.[24,25] Since threads of the same application are likely homogeneous, better options for matchmaking may be obtainable with threads of different applications. For HPC clusters, this option has not been used much to date since contention effects among programs with large numbers of interdependent processes are hard to estimate and need to be predicted before jobs are launched onto the machine. Recent research showed acceptably low contention effects for most program combinations on 64-cluster nodes with potential to obtain high prediction accuracy.[26,27]

PROSPECTIVE DIRECTIONS FOR MULTITHREADED AND MULTICORE CPUS

We expect that chip designs, according to Moore's law, will grow to large numbers of cores and hardware threads. However, off-chip communication and pin limits put significant constraints on the scalability and programmability of multicore/multithreaded chips, as they impact the transfer rate of data to and from the cores. There is currently no technology in sight to drastically increase the pin count. New transport technologies—such as HyperTransport and QuickPath Interconnect—increase the effective throughput per pin, but cannot keep pace with exponential core growth. Since more cores must be kept busy with instructions and data, the future for many-core designs may be limited. These concerns can be mitigated by hiding memory-access latencies via hardware multithreading and increasing the amount of memory on-chip.

In regard to software limits, relatively few applications can use very high concurrency for performance increases. Throughput increases by executing many serial jobs or several moderately parallel applications can be obtained on servers, though commodity machines may not benefit. Otherwise, the success of many-core designs highly depends on proper programming tools, libraries, and models becoming available.

Currently available CPUs incorporate different choices in regard to their design and use of chip space. Design considerations not only include determining the number of cores and threads but also the core complexity, interconnect, cache sizes, and the degree to which components are shared. Since design choices involve tradeoffs, holistic design is necessary, driven by target applications and additional optimization criteria like power consumption, heat dissipation, failure tolerance, and cost.[6]

In regard to the decision between cores and hardware threads, for commodity computing the sweet spot seems to lie in hybrid designs. A small number of on-chip threads can be added for relatively little additional circuitry and can significantly increase throughput. However, diminishing returns in performance and increasing circuitry costs limit the gain from hardware threads.[4] Thus, chip space beyond a few hardware threads is generally better exploited for more cores, cache, or other components. Hybrid CPUs have also been shown to be almost as energy efficient as pure multicore designs.[28]

The balance between cores and hardware threads shifts for servers, which demand maximized throughput and benefit more from larger numbers of hardware threads per core. Servers generally run a large set of nonnumeric programs, typically involving more latency that can be hidden using multithreading. Conversely, numeric applications rarely benefit from hardware multithreading, instead performing better on many-core designs.

Considering that each CPU has made different optimization choices, the consumer is left to decide which is best suited to a specific application mix.

Acknowledgments

We thank Tracy Carver of AMD, Jaime Moreno of IBM, Denis Sheahan of Sun, and Xinmin Tian of Intel for their helpful feedback and for validation of our CPU/GPU data.

References

1. S. Borkar, "Thousand Core Chips—A Technology Perspective," *Proc. 44th Design Automation Conference* (DAC 07), ACM Press, 2007, pp. 746-749.
2. T. Duff, "A Conversation with Kurt Akeley and Pat Hanrahan," *ACM Queue*, Mar./Apr. 2008, pp. 11-17.
3. D. Marr et al., "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology J.*, vol. 6, no. 1, 2002, pp. 4-15.
4. J. Burns and J.L. Gaudiot, "SMT Layout Overhead and Scalability," *IEEE Trans. Parallel and Distributed Systems*, Feb. 2002, pp.142-155.
5. T. Ungerer, B. Robic, and J. Šilc, "A Survey of Processors with Explicit Multithreading," *ACM Computing Surveys*, Mar. 2003, pp. 29-63.
6. R. Kumar, V. Zyuban, and D.M. Tullsen, "Interconnections in Multicore Architectures: Understanding Mechanisms, Overheads, and Scaling," *Proc. 32nd Ann. Int'l Symp. Computer Architecture* (ISCA 05), ACM Press, 2005, pp. 408-419.
7. M. Mantor, "Entering the Golden Age of Heterogeneous Computing"; http://ati.amd.com/technology/streamcomputing/IUCAA_Pune_PEEP_2008.pdf.
8. D. Kanter, "NVIDIA's GT200: Inside a Parallel Processor"; www.realworldtech.com/page.cfm?ArticleID=RWT090808195242.
9. H.Q. Le et al., "IBM POWER6 Microarchitecture," *IBM J. Research and Development*, vol. 51, no. 6, 2007, pp. 639-662.
10. IBM Blue Gene Team, "Overview of the IBM Blue Gene/P Project," *IBM J. Research and Development*, Jan.-Mar. 2008, pp. 199-220.
11. K.J. Nesbit, J. Laudon, and J.E. Smith, "Virtual Private Caches," *Proc. Int'l Symp. Computer Architecture* (ISCA 07), IEEE CS Press, 2007, pp. 57-68.
12. N. Aggarwal et al., "Isolation in Commodity Multicore Processors," *Computer*, June 2007, pp. 49-59.
13. M.D. Hill and M.R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, July 2008, pp. 33-38.
14. D. Dice et al., "Early Experiences with a Commercial Hardware Transactional Memory Implementation," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 09), ACM Press, 2009, pp. 157-168.
15. R. Kumar et al., "Heterogeneous Chip Multiprocessors," *Computer*, Nov. 2005, pp. 32-38.
16. L.A. Barroso and U. Hoelzle, "The Case for Energy-Proportional Computing," *Computer*, Dec. 2007, pp. 33-37.
17. W.-C. Feng and K.W. Cameron, "The Green500 List: Encouraging Sustainable Supercomputing," *Computer*, Dec. 2007, pp. 50-55.
18. H. Sutter, "The Free Lunch Is Over—A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's J.*, Mar. 2005; www.gotw.ca.
19. M. Creeger, "Multicore CPUs for the Masses," *ACM Queue*, Sept. 2005, pp. 64-ff.
20. J. Dongarra et al., "The Impact of Multicore on Computational Science Software," *CTWatch Quarterly*, Feb. 2007, pp. 3-10.
21. J. Larus and C. Kozyrakis, "Transactional Memory," *Comm. ACM*, July 2008, pp. 80-88.
22. L.S. Blackford et al., "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM Trans. Mathematical Software*, vol. 28, no. 2, 2002, pp. 135-151.
23. A.C. Sodan, "Message Passing vs. Shared-Data Programming—Wish vs. Reality," *Proc. 19th Int'l Symp. High-Performance Computing Systems* (HPCS 05), IEEE CS Press, 2005, pp. 131-139.
24. A. Snavely and D.M. Tullsen, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor," *Proc. 5th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 00), ACM Press, 2000, pp. 234-244.
25. J. Nakajima and V. Pallipadi, "Enhancements for Hyper-Threading Technology in the Operating System—Seeking the Optimal Scheduling," *Proc. Usenix 2nd Workshop on Industrial Experiences with Systems Software*, Usenix, Dec. 2002, pp. 25-38.
26. A.C. Sodan and L. Lan, "LOMARC—Lookahead Matchmaking for Multi-Resource Coscheduling on Hyperthreaded CPUs," *IEEE Trans. Parallel and Distributed Computing*, Nov. 2006, pp. 1360-1375.
27. A.C. Sodan et al., *Benefits of Semi Time Sharing and Trading Time vs. Space in Computational Grids*, tech. report 08-020, Univ. of Windsor, Dept. of Computer Science, May 2008.
28. R. Sasanka et al., "The Energy Efficiency of CMP vs. SMT for Multimedia Workloads," *Proc. 18th Ann. Int'l Conf. Supercomputing* (ICS 04), ACM Press, 2004, pp. 196-206.

Angela C. Sodan *is an associate professor in the Department of Computer Science at the University of Windsor, Canada. She received a PhD in computer science from the Technical University of Berlin, Germany. Sodan is a senior member of the IEEE. Contact her at acsodan@uwindsor.ca; http://cs.uwindsor.ca/~acsodan.*

Jacob Machina *is a graduate student in the Department of Computer Science at the University of Windsor. Contact him at machina@uwindsor.ca.*

Arash Deshmeh *is a graduate student in the Department of Computer Science at the University of Windsor. Contact him at deshmeh@uwindsor.ca.*

Kevin Macnaughton *is a graduate student in the Department of Computer Science at the University of Windsor and works as a systems programmer in IT Services at the University of Windsor. Contact him at macnaug@uwindsor.ca.*

Bryan Esbaugh *is a graduate student in the Department of Computer Science at the University of Windsor and works as integration and electronic warfare lead in the HCM Program at Lockheed Martin Canada. Contact him at esbaugh@uwindsor.ca.*