Parallelism via Multithreaded and Multicore CPUs

A.C. Sodan, Jacob Machina, Arash Deshmeh, Kevin Macnaughton, Bryan Esbaugh

Multicore and multithreaded CPUs have become the new approach to obtaining increases in CPU performance. This survey compares multicore and multithreaded CPUs currently on the market, and discusses 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 because CPU clock speeds have 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 eighteen months, still applies and is providing hardware designers with the ability to add more complexity to a chip.[1] This will be true for the immediate future until CPUs reach hard physical limits of circuit density. The additional capacity was used in the past for development of superscalar CPUs with replicated execution units and deep pipelines to exploit instruction-level parallelism. However, only 25% of the additional chip space that became available per year was actually harvested by new architectural features.[2] Additionally, the performance gap between processors and memory limits the gains possible from further raising processor frequency. Thus, the current design direction for performance increases is to utilize available chip space for multithreaded and multicore CPUs. Multithreaded and multicore designs support multitasking via parallel programs or running several applications concurrently.

First introduced were multithreaded CPUs, which employ hardware-level context switching between threads to reduce the idle time of resources in complex superscalar processors. Shortly after, designers integrated more than one processor core onto a single chip. We now have eight-core processors, with forecasts for CPUs with more cores becoming available in the near future. Assuming that Moore's Law holds, we expect a doubling of the number of cores on chip every two years, leading to many-core CPUs (16 or more cores) just over the horizon.

Design Spectrum

Multithreaded and multicore CPUs both exploit concurrency by executing multiple threads, though 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 threads. Multicore CPUs achieve thread concurrency at a higher level, caring less about utilization per core and rather aiming at scalability via

Digital Object Indentifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

replicating cores. These CPUs are often called CMP which stands for Chip Multi-Processing. Most recent CPU and GPU (Graphics Processing Unit) designs like the Sun UltraSPARC T2, IBM POWER6, Intel Xeon, ATI RV770, and NVIDIA GT200 combine both design options and have multiple multithreaded cores.

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

	Product	On-chip / executing threads (per core)	Multithreading type	Number of floating-point units (per core)	Monopolizing prevention (approach taken may depend on core resources)	Single task mode
Intel[3]	Itanium (9000 series)	2 / 1	Blocked*	2	N/A	No
	Xeon (7400 series)	2 / 2	Simultaneous	1	Static partitioning, minimum shares	Yes
	Core i7	2 / 2	Simultaneous	2	Static and dynamic partitioning, minimum shares	
IBM[4]	Power 5	2 / 2	Simultaneous	2	Decode rates managed via (hardware or software set) thread priorities; maximum shares	Yes
	Power 6	2 / 2	Simultaneous	3	POWER 5 capabilities and more replication	No
SUN[5]	SUN / Fujitsu SPARC64 VII	2 / 2	Simultaneous	2	Static Partitioning, minimum shares	Yes
	Ultra SPARC T1	4 / 1	Interleaved**	1 per chip	N/A	No
	Ultra SPARC T2	8 / 2	Parallel Interleaved+	1	2 groups of 4 threads, per group Least Recently Fetched/Executed and static partitioning	Semi++
	Rock[6]	2 / 2	Simultaneous	2 per 4-core cluster	Static partitioning, minimum shares	Yes†
GPU††‡	ATI RV770[7]	> 1000‡ / 10	Interleaved	80	N/A	No
	NVIDIA GT200[8]	1024 / 8 – 16	Interleaved	10	N/A	No

* Blocked multithreading switches to another thread only if the currently executing thread runs into a stall.[9]
** Interleaved multithreading switches among ready-to-run threads every cycle.[9]
+ Two execution pipelines per core, each serving one thread group.
++ A single thread exploits only one of the two integer pipelines, but has access to all other execution units.
† The two threads per core can execute one application thread via speculative execution and merging some partitioned resources.
†† The ATI RV770 is used on the Radeon HD 4870, and the NVIDIA GT200 on the GeForce GTX280.
‡ Estimates, no vendor specifications available.

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

Multithreaded Cores

All multithreaded cores keep multiple hardware threads "on-chip", ready-for-execution. This is necessary to make fine-grained switching between threads feasible and minimize context-switch cost 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. A second thread increases hardware costs by only a few percent. For example, the Intel Xeon only needs 5% extra chip space to support a second hardware thread.[10] Cost growth is approximately linear up to at least eight threads, but clearly superlinear thereafter.[11]

The number of on-chip threads per core typically supported by commercial processors ranges from 2 in Intel's Xeon to 8 in Sun UltraSPARC T2. An extreme example was the 128 threads in Tera/Cray MTA (one of the first practical but not commercially successful designs) which needed the large number of threads to hide memory-access latencies and to compensate for the lack of a cache in its architecture. The same technique is used in the massively multi-threaded NVIDIA GT200 GPU which opts for minimal caches in favor of additional computational resources.

There exist 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,[9] see Table 1. Most current CPUs employ the latter approach which is called Simultaneous Multi-Threading (abbreviated as SMT), named HyperThreading Technology[10] (HTT) by Intel.

SMT dispatches (pre-decoded) instructions from only a subset of the on-chip threads per cycle. The number of threads in this subset also impacts the utilization of the execution units, particularly if the threads complement each other's use of these units. However, at present time, 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 2. 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 the architecture of their predecessors, 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 on the shared resources.[12]

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

Table 2. Comparison of features for current commercial multicore CPUs.

	Product	Cores	Clock (Ghz)	Power (Watts) per CPU	Special features	On-chip interconnect	L1-I / L1-D size (KB) per core	L2 size (MB) per chip	L2 allocation	L3 Size (MB)
AMD[13]	Opteron (3rd gen.)	4, 6	1.7 – 3.1	40 – 105	IMC, 128-bit FPU per core, dual PM*	Crossbar	64 / 64	2, 3	Private	2, 6
	Phenom II	3, 4	2.4 – 3.2	65 – 125	IMC, 128-bit FPU per core, dual PM*	Crossbar	64 / 64	1.5, 2	Private	6
	Turion X2	2	1.6 – 2.4	18 – 35	IMC	Crossbar	64 / 64	1, 2	Private	-
Intel[3]	Pentium Dual Core	2	1.7 – 2.7	65	IR 4	Front side bus**	12 / 16	2, 4	Dynamic	-
	Core 2 Duo Family	2	1.8 – 3.3	65	IR 4	On-chip bus	32 / 32	2, 3, 4, 6	Shared	-
	Core 2 Quad	4	2.0 – 3.0	95 – 105	IR 4, dynamic PM	On-chip bus / front-side bus** (between pairs of cores)	32 / 32	4, 6, 8, 12	Shared per 2 cores	-
	Itanium (9000 series)	2	1.4 – 1.66	75 – 104	VLIW, IR 6	Direct pathways	16/ 16	I: 2 D: 0.5	Private	4, 6, 9, 12 per core, private
	Xeon (7400 series)	4, 6	2.13 – 2.66	50 – 130	IR 4, dynamic PM	On-chip bus	32 / 32	6, 9	Shared	8, 12, 16
	Core i7	4	2.66 – 3.33	130	Triple channel IMC	Crossbar	32 / 32	1	Private	8
IBM[4]	Power 5	2	1.5 – 1.9	Unpublished	IR 5, IMC	Crossbar	64 / 32	1.875	Shared	36 off-chip
	Power 6	2	4.7 – 5	Unpublished	IR 7, 1 decimal, 2 binary FPUs per core	On-chip bus	64 / 64	8	Private	32 off-chip
	Cell BE, PPE	1	3.2	110+	General purpose, SMT (2 threads)	Ring Bus	32 / 32	0.5	N/A	-
	Cell BE, SPE	8	3.2		Simplified for SIMD support	Ring Bus	256	-	N/A	-
SUN[5]	Ultra SPARC T1	4 ,6, 8	1.0 – 1.2	72 – 79	1 FPU per chip, IMC	Crossbar	16 / 8	3	Shared	-
	Ultra SPARC T2	4, 6, 8	1.0 – 1.6	95 – 123	IMC and INC, crypto unit (per core), SOC	Crossbar	16 / 8	4	Shared	-
	Ultra SPARC IV+	2	1.5 – 2.1	90	IR 4, IMC	On-chip bus	64 / 64	2	Shared	32 off-chip
	SUN / Fujitsu SPARC64 VII	4	2.5	135	IR 4, hardware barrier	On-chip buses	64 / 64	6	Shared	-
	Rock	16	2.1	250	4-core clusters, aggressive speculation, HTM, IR 4	Direct pathways / crossbar (among clusters)	32 / 64 (per cluster)	2	Shared	16 off-chip

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26

	Product	Cores	Clock (Ghz)	Power (Watts) per CPU	Special features	On-chip interconnect	L1-I / L1-D size (KB) per core	L2 size (MB) per chip	L2 allocation	L3 Size (MB)
Specialized	Tilera: TILE 64[14]	64	0.5 – 0.9	15 – 22	Large number of simple cores	Multi-link mesh	8 / 8	4†	Shared†	-
	ARM Cortex-A9 MPCore[15]	2, 4	1	< 1	Ultra small, SOC, ultra low power	Multi-level bus	16 – 64 / 16 – 64	2	Shared	-
	ATI RV770††	10	0.75	160	Simplified for SIMD support	Crossbar	Unpublished	>256 KB‡	Shared	-
	NVIDIA GT200††	30	1.295	236	Simplified for SIMD support	Crossbar	Unpublished	256 KB	Shared	-

L1 cache is private for each core in all processors which have L1 cache. L3 cache is shared for all CPUs which have L3 cache. In case of private L2 caches, the total size was obtained by multiplying the size per core by the number of cores. Dynamic L2 allocation stands for dynamic partitioning.
Abbreviations in table: 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.
* Separate power management for cores and memory controllers.
** For data exchange; otherwise not relevant since no integration of memory controller and network controller.
† 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.

The trend is toward replicating additional on-chip components—such as memory controllers and caches—which may be private or shared. For example, the IBM POWER6 and AMD Opteron have private L2 caches and share multiple memory controllers.

Integration of Components

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 towards integrating this component. Some designs support multiple integrated memory controllers to make memory-access bandwidth scalable with the number of cores, as done by the IBM POWER6 and the Sun UltraSPARC T2. Integrating a GPU core on chip is another possibility which is currently planned by several vendors for next-generation CPUs. A similar approach is already used in the embedded and mobile markets which frequently combine a general-purpose core and a digital signal processor core on a single chip.

IBM's Blue Gene/P[16] system relies on a highly integrated system-on-a-chip design which features four cores, five network interfaces, two memory controllers and 8MB of L3 cache. Because each Blue Gene compute node is so highly integrated, the system scales to hundreds of

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

thousands of processors. The UltraSPARC T2 is another example of system-on-a-chip design which integrates memory controllers, I/O, security functions, and an advanced network interface.

Shared vs. Private Caches

Aside from concurrency, caches are the most important feature of modern CPUs to enhance performance, due to the performance gap between CPU speed and memory-access times. The dominant approach to mitigating this gap is to exploit 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, mentioned earlier, or using high-speed direct-addressed memory as in the Cell SPE.

For multicore designs following the dominant approach of incorporating caches, a major consideration is the organization of cache memory. 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 L2 cache in multicore designs varies. The L2 cache may be either private, i.e. dedicated to each core, or shared between cores. 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 not only depends on tradeoffs regarding the usage of chip space but also on the application characteristics. Shared caches are important if threads of the same application are executing on multiple cores and share a significant amount of data. In this case, a shared cache is more economic as it avoids multiple copies of data and cache-to-cache transfers. However, shared caches can impose high demands on the interconnect.[12]

Software threads which do not share much data may 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 provide performance isolation and guarantee predictable service. A more flexible approach is a hybrid design, where different numbers of cache banks can be allocated as shared or private, depending on the cache needs of the currently running threads. This approach can support threads which share data and threads which do not. The hybrid design can be refined to dynamic proportional partitioning as proposed in recent research.[17] The latter makes it possible to provide for each core a level of service equal to that of a single-core chip with the corresponding amount of provisioned cache resources.

Shared vs. Private Hardware-Thread Resources

In contrast to multicore designs which tend to replicate most resources, sharing is the dominant approach in hardware multithreading. However, some level of replication and partitioning is still

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

necessary.[10] Replication is essential for execution units that may 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 though also extra potential for contention, and may need some mechanism to prevent monopolization (see Table 1). 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.

Fault Tolerance

Dynamic partitioning of cache or other resources can be extended to deal with hardware faults which are more likely to occur with higher circuit density.[18] Faults can result in electrical noise or minor permanent defects in silicon, potentially spreading from individual components and resulting in failure of the entire chip. 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 inter-chip interconnects and memory controllers in addition to multiple cache banks. This leads to supporting different degrees of isolation vs. sharing and separation of working components from faulty ones. Such solutions greatly increase overall availability and provide graceful performance degradation in case of faults.[18]

Interconnect

Another important feature with impact on performance of multicore chips 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 multi-CPU systems. The trend now is to use a crossbar or other advanced mechanisms to reduce latency and contention (see Table 2, *On-chip interconnect*). For instance, AMD CPUs employ a crossbar, and the Tilera TILE64 implements a fast non-blocking multi-link mesh. However, the interconnect can become expensive: an 8 x 8 crossbar on-chip can consume as much area as five cores and as much power as two cores.[12]

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, as the more recent trend, L3—or supporting data-exchange short-cuts such as cache-to-cache transfer helped reduce the off-chip traffic. However, more on-

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

chip cache levels put additional requirements on the on-chip interconnect in terms of complexity and bandwidth.

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, the new trend in off-chip communication is from bus-based towards packet-based, point-to-point interconnects. This concept was first implemented in HyperTransport for AMD CPUs and later in Intel's QuickPath Interconnect. The off-chip interconnect and data-coherency support also impact 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 with special execution units designed to increase the performance of Java operations.[19] The Tilera TILE64's CPU is optimized for data processing, composed of 64 low-powered simple processing cores, and its increased data-flow capacity makes it well suited for embedded systems, such as telecommunications routers. The IBM Cell was originally designed for gaming though it also turned out to be very useful for data-processing applications in bioinformatics and astrophysics.

GPUs represent an extreme example of specialized multicore design. Modern GPUs have ten 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.

Performance Gain: Core Complexity vs. Number of Cores

Traditional CPU optimizations were aimed at increasing 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 as to what extent to simplify traditional CPU designs to allow more circuitry to be dedicated to concurrency. An example is Sun's UltraSPARC T1 which reduces the number of FPUs on chip, or the Intel Atom which removed 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 and floating-point units, 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.

Digital Object Indentifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

A greater issue-width also increases peak performance, with the POWER5 architecture issuing five instructions per cycle and AMD chips issuing three. CPUs that focus on per-thread performance also generally have much higher clock rates than those that focus on many-threaded support, as seen in the 5.0 Ghz clock rate of the IBM POWER6 and the 3.73 Ghz 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 if complexity is quadrupled.[1] See sidebar for performance comparisons of different designs using standard benchmarks.

An important consideration for per-application performance is that chip concurrency cannot be exploited by serial programs. Even in parallel programs, some parts of the algorithm must run sequentially, and Amdahl's Law tells us 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 formulized as an extension of Amdahl's Law.[1,20] This, along with other considerations gives the following conclusions:

- Larger numbers of simple cores are preferable as long as the serial part of the application is very small; otherwise more complex cores are beneficial.[20]
- Benefits may shift towards 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, may experience higher returns from added complexity than typically expected.

However, not only per-application performance but also the overall workload must be considered. Per-application performance is important if the load consists only of 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 design goals of commercial servers, such as database and web servers.

To strike a balance between per-thread performance and throughput, the former may 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 pre-production Rock processor implements this idea by optionally using the two hardware threads per core to execute one application thread.[6] A simpler approach, already applied in some

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

multithreaded CPUs, is to allocate partitioned resources to one thread if run in single-task mode, as implemented in the Intel Xeon and Pentium Extreme Edition.[3]

Another possibility are chip designs which incorporate some diversity regarding the complexity of the cores, such as in IBM's Cell processor. A few higher-complexity cores may run sequential parts of applications or demanding applications. Though not yet commercially available, research indicates that a processor comprised of many simplified cores and a few high performance cores could provide the greatest total processing power for a given chip space and power budget.[20,21] However, these CPUs may be more difficult to design and program.

Cost and Power Consumption

Performance no longer dominates the design objectives: chip fabrication costs and fault tolerance (as discussed above), power efficiency, and heat dissipation have 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.[12] 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.[11] Multicore CPUs also provide greater options for power management as 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 greatly benefit.

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% of the power consumed by a server,[22] projected electricity costs for a four-year term approach the purchase price of the system. In the case of High Performance Computing (HPC) machines, building customized cooling solutions can cost as much as the computer itself.[23] 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 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[23] architecture per parallel application is IBM's QS22/LS21 hybrid architecture used in the Roadrunner supercomputer at Los Alamos National Labs. This architecture is the highest ranked in the Top500 (http://top500.org) list of June 2009 and Green500 (http://green500.org) list

Digital Object Indentifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

of November 2008. These top rankings were made possible by employing the power and cost efficient Cell as main compute processor.

The Software Challenge

In the future, exponential growth in CPU performance will mostly be obtainable from more hardware threads and cores. However, hardware concurrency can only be exploited with multiple serial programs or with parallelized applications.[24] 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 (running multiple virtual machines with potentially different operating systems). 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.[24,25] Possibilities to automatically extract parallelism are currently limited, and parallelism typically needs to be expressed explicitly. Changing toward parallel programming for commodity machines is considered by Sutter as the next revolution after the introduction of object-oriented programming.[24] Writing correct and efficient parallel programs is a major challenge which calls for better tools and more abstract programming models that make thread programming safer and more convenient. Solutions may draw upon experiences obtained in High Performance Computing, with stimulus for further improvement in HPC techniques generated by the greatly enlarged market.[26]

Widely used in HPC, the OpenMP shared-memory programming model[27] could be adopted by commodity software developers. Another promising direction is transactional memory,[28] which borrows the transaction concept from databases and simplifies data-access coordination through automatic checkpointing and rollback mechanisms. The first CPU which supports this model in hardware (for common cases) is the pre-production Sun Rock.[6] Rather than investing additional time for parallel-software development, a more economical approach is using pre-parallelized compilers and libraries like the BLAS (Basic Linear Algebra Subprograms) library. [29]

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 Multi-Processing (SMP) nodes, see sidebar. 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.[30] Moreover, HPC applications will need to

Digital Object Indentifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

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.[26] Another limiting factor is that the performance benefit of additional cores is less than that of additional CPUs, except when threads share data, see sidebar. Thus, multicore CPUs are—as pointed out by Dongarra et al.[26]—not the new SMP.

The software challenge also affects commodity compilers which may need to address simplified or specialized cores like in the Cell or a GPU. Whereas in the past, instruction-level parallelism was to a large extent extracted by the hardware itself, 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. [31,32] 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 inter-dependent 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 and potential to obtain high prediction accuracy.[33,34]

Prospective Directions for Multithreaded and Multicore CPUs

Hardware and Software Limits

We expect chip designs, according to Moore's Law, to 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, though they cannot keep pace with exponential core growth. Since more cores need to 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.

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

In regards to software limits, there are relatively few applications which can make use of 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.

Design Choices

Currently available CPUs have made different choices in regards to CPU design and use of chip space. Design considerations not only include deciding 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.[12]

In regards 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.[11] 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.[35]

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 latencies 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 CPU is best suited to a specific application mix.

Acknowledgments

We thank (alphabetically) 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.

Sidebar--Performance of Multithreaded and Multicore CPUs

Highest single-thread performance in the SPECfp2006[1] and SPECint2006 benchmarks is provided by systems based on Intel Core i7[*] processors, achieving 68% higher integer speed and 84% higher floating-point speed (supported by its memory controller) compared to AMD's Opteron[+]. 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[†], but has a 26% advantage using standard optimizations. However, the UltraSPARC T2[††] outperforms the Opteron[++] in the multithreaded SPECweb2005 benchmark for webserver throughput and response times, by serving 36% more web requests over the same time span. Considering different loads, simulation studies with database applications[2], specifically OLTP and DSS, showed up to 40% shorter response times for the POWER5 compared to the UltraSPARC T1 if serving an unsaturated load. However, for saturated loads the UltraSPARC T1 was able to achieve up to 70% greater throughput.

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% performance gain vs. single-thread execution[3] for the OpenMP version of the NAS[4] 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% performance loss vs. single-thread execution[3], whereas the standard multiple-process version of FT suffered a larger loss at 50%, which was mostly attributed to memory contention due to intensive inter-process communication.[5]

Performance gains from multiple cores were demonstrated by the example of the AMD Opteron dual-core processors showing 37% improved performance utilizing the second core when measured by the standard multiple-process NAS CG and FT benchmarks.[6] The same study also showed that one dual-core chip performed only 5.8% slower in the CG benchmark and only 9% slower in the FT benchmark than two chips using a single core, while being much more power and cost efficient. Another study with pure multi-process applications running on large clusters with up to 4,096 CPUs obtained benefits of between 20% and 50% from using a second core. [7]

[*] ASUSTeK i7-965 results of Feb. 2009
[+] HP Opteron 2384 results of Dec. 2008
[++] HP Opteron 8393 results of May 2009
[†] Fujitsu T5120 results of Jan. 2009
[††] SUN T5440 results of Oct. 2009

1. SPEC—Standard Performance Evaluation Corporation, http://spec.org.
2. N. Hardavellas, I. Pandis, R. Johnson, N.G. Mancheril, A. Ailamaki, and B. Falsafi, "Database Servers on Chip Multiprocessors: Limitations and Opportunities", *Proc. 3rd Biennial Conf. on Innovative Data Systems Research (CIDR)*, Jan. 2007.
3. M. Curtis-Maury, T. Wang, C. Antonopoulos, and D. Nikolopoulos, "Integrating Multiple Forms of Multithreaded Execution on multi-SMT Systems: A Study with Scienitific Applications", *Proc. Int'l Conf. on the Quantitative Evaluation of Systems (QUEST05)*, IEEE, 2005.
4. D. Bailey, T. Harris, W. Saphir, R. V. der Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," Technical Report NAS-95-020,NASA Ames Research Center, Moffett Field, CA, 1995; benchmark suite available at http://www.nas.nasa.gov/Resources/Software/npb.html.
5. T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An Empirical Study of Hyper-Threading in High Performance Computing Clusters", *Linux HPC Revolution*, 2002.
6. S.R. Alam, R.F. Barrett, J.A. Kuehn, P.C. Roth, and J.S. Vetter, "Characterization of Scientific Workloads on Systems with Multi-Core Processors", *Proc. IEEE Int.l Symp. on Workload Characterization (IISWC06),* San Jose / CA, 2006.
7. 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)*, Dresden, Germany, June 2007.

1. S. Borkar, “Thousand Core Chips—A Technology Perspective”, Proc. DAC, San Diego, CA, June 2007.
2. T. Duff, “A Conversation with Kurt Akeley and Pat Hanrathan”, ACM Queue, 6(2), March/April 2008.
3. Intel Corportation, “Intel® Processors”, available at http://www.intel.com/products/processor/, retrieved June 2008.
4. H.Q. Le, W.J. Starke, J.S. Fields, F.P. O’Connell, D.Q. Nguyen, B.J. Ronchetti, W.M. Sauer, E.M. Schwarz, and M.T. Vaden, “IBM Power6 Microarchitecture”, *IBM Journal of Research and Development*, 51(6), 2007.
5. Sun Corporation, “Sun Microelectronics—Products”, available at http://www.sun.com/processors/, retrieved June 2008.
6. D. Dice, Y. Lev, M. Moir, and D. Nussbaum, “Early Experiences with a Commercial Hardware Transactional Memory Implementation”, *Proc. ASPLOS*, ACM, Washington, DC, March 2009.
7. M. Mantor, “Entering the Golden Age of Heterogeneous Computing”, available at http://ati.amd.com/technology/streamcomputing/IUCAA_Pune_PEEP_2008.pdf, retrieved April 2009.
8. D. Kanter, “NVIDIA's GT200: Inside a Parallel Processor”, available at http://www.realworldtech.com/page.cfm?ArticleID=RWT090808195242, retrieved March 2009.
9. D. Culler and J.P Singh with A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaumann Publishers, August 1998.
10. D. Marr D, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, and M.Upton, “Hyper-Threading Technology Architecture and Microarchitecture”, *Intel Technology Journal* Q1, 6(1), 2002.
11. J. Burns, J.L. Gaudiot, “SMT layout overhead and scalability”, *IEEE Trans. on Parallel and Distributed Systems,*13(2), pp.142-155, Feb 2002.
12. R. Kumar, V. Zyuban, and D.M. Tullsen, “Interconnections in Multicore Architectures: Understanding Mechanisms, Overheads, and Scaling”, *Proc. Int’l Symp. on Computer Architecture (ISCA05)*, 2005.
13. AMD Corporation, “Advanced Micro Devices, AMD–Global Provider of Innovative Microprocessor Solutions”, available at www.amd.com/us-en/processors/, retrieved June 2008.
14. Tilera Corporation, “Tilera Corporation”, available at http://www.tilera.com/products/processors.php, retrieved June 2008.
15. ARM Corporation, The ARM Cortex-A9 Processors, available at http://www.arm.com/pdfs/ARMCortexA-9Processors.pdf, Sept. 2007.
16. IBM Blue Gene team, “Overview of the IBM Blue Gene/P project”, *IBM Journal of Research and Development*, 52(1-2), January-March 2008, pp. 199-220.
17. K.J. Nesbit, J. Laudon, and J.E. Smith, “Virtual Private Caches”, *Proc. Int’l Symp. Computer Architecture (ISCA 07)*, IEEE CS Press, 2007.
18. N. Aggarwal, P. Ranganathan, N.P. Jouppi, and J.E. Smith, “Isolation in Commodity Multicore Processors”, *IEEE Computer*, June 2007, pp. 49-59.
19. Azul Systems, “Azul Compute Appliances”, available at http://www.azulsystems.com/products/compute_appliance.htm, retrieved June 2008.
20. M.D. Hill and M.R. Marty, “Amdahl’s Law in the Multicore Era”, *IEEE Computer*, July 2008, pp. 33-38.
21. R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan, “Heterogeneous Chip Multiprocessors”, *IEEE Computer*, Nov. 2005, pp. 32-38.
22. L.A. Barroso and U. Hoelzle, “The Case for Energy-Proportional Computing”, *IEEE Computer*, December 2007.

Digital Object Identifier 10.1109/MC.2009.377 0018-9162/$26.00 © 2009 IEEE

23. W.-C. Feng and K.W. Cameron, "The Green500 List: Encouraging Sustainable Supercomputing", *IEEE Computer*, December 2007.
24. H. Sutter, "The Free Lunch is Over—A Fundamental Turn Toward Concurrency in Software", *Dr. Dobb's Journal*, 30(3), March 2005.
25. M. Creeger, "Multicore CPUs for the Masses", *ACM Queue*, 3(7), September 2005.
26. J. Dongarra, D. Gannon, G. Fox, and K. Kennedy, "The Impact of Multicore on Computational Science Software", *CTWatch Quarterly*, 3(1), Feb. 2007.
27. OpenMP Official Web Site at http://openmp.org/wp/, retrieved June 2008.
28. J. Larus and C. Kozyrakis, "Transactional Memory", *Communications of the ACM*, 51(7), July 2008.
29. L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, "An Updated Set of Basic Linear Algebra Subprograms (BLAS)", *ACM Trans. on Mathematical Software*, 28(2), 2002, pp. 135–151.
30. A.C. Sodan, "Message Passing vs. Shared-Data Programming—Wish vs. Reality", *Proc. Int'l Symp. High Performance Computing Systems (HPCS 05)*, Guelph / Canada, IEEE CS Press, May 2005.
31. D.M. Tullsen and A. Snavely, "Symbiotic Jobscheduling for a Simultaneous Multithreading Processor", Proc. ASPLOS, Nov. 2000.
32. 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*, Boston/MA, USA, Dec. 2002.
33. A.C. Sodan and L. Lan, "LOMARC—Lookahead Matchmaking for Multi-Resource Coscheduling on Hyperthreaded CPUs", *IEEE Trans. on Parallel and Distributed Computing*, 17(11), Nov. 2006, pp. 1360-1375.
34. A.C. Sodan, G. Gupta, A. Deshmeh, and X. Zeng, "Benefits of Semi Time Sharing and Trading Time vs. Space in Computational Grids", *Technical Report 08-020*, University of Windsor, Computer Science, May 2008.
35. R. Sasanka, S.V. Adve, Y. Chen, and E. Debes, "The energy efficiency of CMP vs. SMT for multimedia workloads", *Proc. 18th Ann. Iinternat. Conf. on Supercomputing (ICS),* ACM, New York, NY, 2004, pp.196-206.