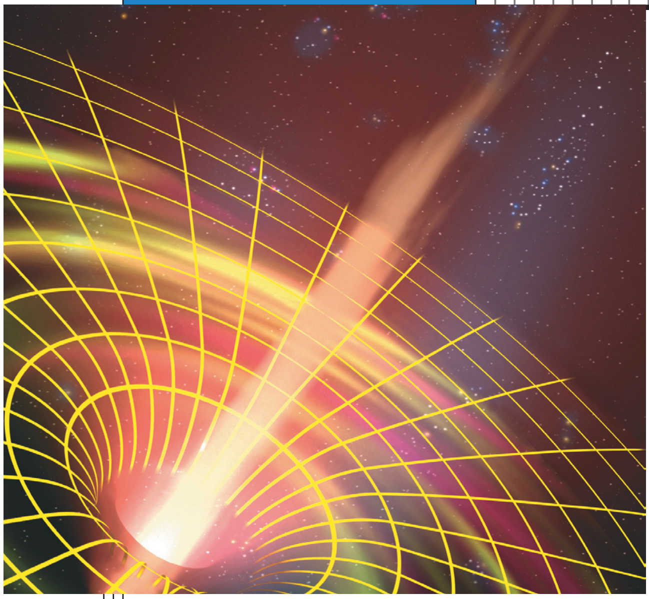


ARCHITECTURES FOR EXTREME-SCALE COMPUTING

Josep Torrellas, *University of Illinois at Urbana-Champaign*

Extreme-scale computers promise orders-of-magnitude improvement in performance over current high-end machines for the same machine power consumption and physical footprint. They also bring some important architectural challenges.

After many years of research and development, high-end computing has finally reached peta-op performance—that is, individual machines that can execute about 10^{15} operations per second (peta-ops). These machines attain such performance using extraordinary, highly concurrent architectures. They include Los Alamos National Laboratory's Roadrunner, which augments conventional processors with high-performance IBM Cell accelerators, providing a hybrid platform; or installations of IBM's Blue Gene/P, which use a massive number of simpler, conventional processors and rely on high chip and system integration. They will soon include the University of Illinois' Blue Waters system described in the "Blue Waters: Application-Driven System Design for Sustained Petascale Performance" sidebar, which uses high-performance processors and novel system components and packaging. With all of these architectures available, the next few years will bring breakthroughs in science and engineering.

However, these architectures are hardly scalable. They are simply too large and consume too much power. Indeed, petascale machines have a footprint of about 1/10th of a football field and consume several megawatts (MW). One megawatt costs about one million dollars per year. For these reasons, high-performance computer architects are focusing on extreme-scale computing.

Broadly speaking, an extreme-scale architecture is one thousand times more capable than a current architecture with the same power consumption and physical footprint. This means that a machine with the power consumption and physical footprint of a current petascale machine must be able to deliver exascale performance—namely, 10^{18} operations per second (exa-ops). It also means that, intuitively, the power consumption and physical footprint of a current departmental server should be enough to deliver petascale performance. Finally, a single commodity chip should deliver terascale performance—namely, 10^{12} operations per second (tera-ops).

Clearly, attaining such a general-purpose tera-op chip, peta-op departmental server, and exa-op data center would revolutionize computing. Conceiving and building such systems, however, poses technical challenges at all levels of the computing stack, including circuits, architecture, software systems, and applications. The sheer size of the challenges and opportunities of attaining such systems by the end of the next decade should act as a strong motivator for researchers.

ARCHITECTURAL CHALLENGES IN EXTREME-SCALE COMPUTING

To attain extreme-scale computing, researchers must address architectural challenges in energy and power efficiency, concurrency and locality, resiliency, and programmability.

Published by the IEEE Computer Society

BLUE WATERS: APPLICATION-DRIVEN SYSTEM DESIGN FOR SUSTAINED PETASCALE PERFORMANCE

Robert Fiedler, Robert Wilhelmson, William Kramer, and Brett Bode
National Center for Supercomputing Applications

Blue Waters (www.ncsa.illinois.edu/BlueWaters), a system designed to provide sustained petaflops performance on a wide range of applications, is being developed by IBM in collaboration with the National Center for Supercomputing Applications (NCSA) and the University of Illinois, and is funded by the National Science Foundation (NSF) and the state of Illinois.

The heart of this system is the Power7 chip, which features eight cores, each with 32-Kbyte L1 instruction and data caches and a 256-Kbyte L2 unified cache. The entire 32-Mbyte on-chip L3 cache can be accessed by any core with latency approximately three times lower than local memory. For each core, up to 4 Mbytes of data in the L3 cache is automatically migrated to a private region with latency approximately 15 times lower than local memory. Each core includes four floating-point units, a VSX (vector) unit that supports single- and double-precision operands, and two load/store units. Simultaneous multithreading (SMT) is supported, allowing one, two, or four SMT threads per core. Dual double data rate 3 (DDR3) memory controllers provide a sustained memory bandwidth of 100 Gbytes per chip.

The 200,000+ Power7 cores in the system communicate via a unique, integrated, high-speed, low-latency interconnection fabric. Remote direct memory access technology enables effective overlap of communication and I/O operations with computational work. The system also includes well over 10 Pbytes of user disk space in a general parallel file system (GPFS), and an archival high-performance storage system (HPSS) that will expand to 500 Pbytes. GHI, a software interface between GPFS and HPSS, will enable automatic migration of disk files to archival storage while presenting users with a simple, unified view of their files.

The Blue Waters packaging extends the use of water-cooled designs, leading to greater energy efficiency. In cold weather, an outdoor cooling tower will chill the water. In addition, energy losses due to AC/DC conversion associated with an uninterruptable power supply are eliminated by exploiting the university's highly reliable electricity supply.

High levels of reliability and automated system health monitoring will undoubtedly be critical for practical use of extreme-scale systems. Blue Waters includes numerous reliability, availability, and serviceability (RAS) features designed to ensure that the mean time between failures is more than a few days, and includes automated checkpoint/restart capabilities. Moreover, an integrated system console will assist operators in monitoring the system's health by automatically filtering status data collected by very large numbers of system components, enabling them to quickly pinpoint any problematic hardware.

The programming environment for application developers and users is an important aspect of the Blue Waters system, and one that is under active development. It not only accommodates established tools and parallel programming models such as the message passing interface (MPI) and OpenMP, but also encourages the use of new tools and models such as Unified Parallel C (UPC), Co-Array Fortran, and global shared memory by ensuring interoperability for all of these programming languages and models.

The Blue Waters programming environment also includes advanced software development tools, which are a key factor in programmer productivity. Developers will have the opportunity to move beyond the traditional command-line environment to an expanded Eclipse-based integrated development environment. The IDE will provide intuitive access via a common communication interface to many powerful capabilities on Blue Waters, including tools for basic code development and building, for remote debugging, and for automated/expert-system-guided performance tuning. Further, interfaces provided to the Blue Waters resource manager will facilitate science and engineering discovery, including input dataset creation/data staging, batch job submission and monitoring, in-line data analysis using coscheduled processors, data reduction/postprocessing, remote scientific visualization, and archival data storage.

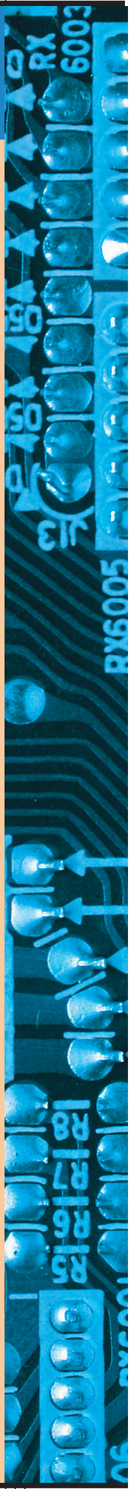
The Blue Waters system design was chosen to deliver sustained petascale performance for a broad range of science and engineering applications. A wide range of applications selected by the NSF through the Petascale Computing Resource Allocations (PRAC) program are currently being prepared for Blue Waters with the help of a team of application specialists at NCSA. The PRAC teams span many areas of investigation, including quantum chromodynamics (fundamental properties of matter), astrophysics (cosmology, galaxy formation, gamma ray bursts, turbulent stellar dynamics), chemistry (biomolecular dynamics, materials science, superconductors), turbulent fluid flows, geosciences (earthquakes), weather and climate modeling (tornadoes, global warming), social sciences (contagion), and evolutionary biology.

***Robert Fiedler** is a technical program manager at the National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign. Contact him at rfiedler@ncsa.uiuc.edu.*

***Robert Wilhelmson** is a chief science officer at NCSA. Contact him at bw@ncsa.uiuc.edu.*

***William Kramer** is a deputy project director at NCSA. Contact him at wkramer@ncsa.uiuc.edu.*

***Brett Bode** is a technical program manager at NCSA. Contact him at bbode@ncsa.uiuc.edu.*



Increasing energy and power efficiency

As Peter Kogge and his colleagues indicate in their DARPA-sponsored study on exascale computing, improving energy and power efficiency is the most formidable challenge facing designers of high-end systems.[1] Because extreme-scale machines must be three orders of magnitude more energy efficient than current machines, a possible target for extreme-scale computing is an exa-op data center that consumes 20 MW, a peta-op departmental server that consumes 20 kilowatts (KW), and a tera-op chip multiprocessor that consumes 20 watts (W). These numbers imply that the machine must deliver 50 giga operations (or 50×10^9 operations) per watt. Because these operations must be performed in a second, each operation can only consume, on average, an energy of 20 pico-Joules (pJ). For reference, consider Intel's Core Duo mobile processor circa 2006, which consumed, on average, more than 10,000 pJ per instruction.[2] Our target is even harder to attain than these numbers suggest. This is because large machines spend most of the energy transferring data from or to remote caches, memories, and disks. Minimizing data transport energy, rather than arithmetic logic unit (ALU) energy, is the real challenge.

Several evolutionary approaches to attaining more energy-efficient architectures exist. At the circuit level, these approaches emphasize designing circuits for energy and power efficiency, rather than for speed, as in most current approaches. Such designs include on-chip interconnection network circuits for low swing, or new memory layouts and bank organizations that minimize the amount of capacitance switched per access. The latter is important because current memory designs focus on providing cost-effective bandwidth, and are wasteful when activating portions of the memory. Future designs must minimize the energy spent charging and discharging lines, possibly through memory designs that include hierarchical bit-line organizations.

Phase change memory's main attraction is its scalability with process technology.

At the microarchitecture level, evolutionary approaches involve simplifying the cores, making their pipelines shallower and their execution engines less speculative. Finally, at the machine architecture level, a popular approach is to augment the processing nodes with accelerators that are energy-efficient for some operations.

Unfortunately, attaining three orders of magnitude higher efficiency in energy and power requires all of this and much more. In particular, it calls for the maturity of several technologies that are now being developed or investigated.

Near-threshold voltage operation. One of the most effective approaches for energy-efficient operation is to reduce the supply voltage (V_{dd}) to a value only slightly higher than the transistor threshold voltage (V_{th}). This is called near-threshold voltage (NTV) operation. It corresponds to a V_{dd} value of around 0.4 V, compared to a V_{dd} of around 1 V for current designs.

Broadly speaking, operation under NTV can reduce the gates' power consumption by about 100× while increasing their delay by 10×. The result is a total energy savings of one order of magnitude.[3] In addition to the 10× increase in circuit delay, the close proximity of V_{dd} and V_{th} induces a 5× increase in gate delay variation due to process variation, and a several orders-of-magnitude increase in logic failures—especially in memory structures, which are less variation tolerant.

Effective use of NTV in extreme-scale architectures will require solving these challenges. For example, increasing the number of cores in the machine can help make up for the cores' lower speed. Moreover, the optimal NTV for memory structures is slightly higher than the optimal NTV for logic. Thus, caches can cycle at a few times higher frequency than cores, suggesting novel architectural designs in which a group of cores might share a cache.[3]

Aggressive use of circuit or architectural techniques that minimize or tolerate process variation can address the higher-variation shortcoming. This includes techniques such as body biasing and variation-aware job scheduling. Finally, novel designs of memory cells and other logic can solve the problem of higher probability of logic failure. Overall, NTV operation is a promising direction that several research groups are pursuing.

Nonsilicon memory. Nonsilicon memory is another relevant technology. Phase change memory (PCM), which is currently receiving much attention, is one type of nonsilicon memory. PCM uses a storage element composed of two electrodes separated by a resistor and phase-change material such as $Ge_2Sb_2Te_5$.[4] A current through the resistor heats the phase-change material, which, depending on the temperature conditions, changes between a crystalline (low-resistivity) state and an amorphous (high-resistivity) one—hence recording one of the two values of a bit.

PCM's main attraction is its scalability with process technology. Indeed, both the heating contact areas and the required heating current shrink with each technology generation. Therefore, PCM will enable denser, larger, and very energy-efficient main memories. DRAM, on the other hand, is largely a nonscalable technology, which needs sizable capacitors to store charge and, therefore, requires sizable transistors.

Currently, PCM has longer access latencies than DRAM, higher energy per access (especially for writes), and limited lifetime in the number of writes. However, advances in circuits and memory architectures will hopefully deliver advances in all these axes while retaining PCM scalability.

Finally, because PCM is nonvolatile, it can potentially support novel, inexpensive checkpointing schemes for extreme-scale architectures. Researchers can also use it to design interesting, hybrid main memory organizations by combining it with plain DRAM modules.

Photonic interconnects. Optics have several key properties that can be used for interconnects. They include low-loss communication, very large message bandwidths enabled by wavelength parallelism, and low transport latencies, as given by the speed of light.[5] Consequently, they are especially good substrates for long-range communication. Indeed, when used for long-haul data communication, they deliver substantial end-to-end reductions in energy per bit and time per access. We therefore expect extreme-scale machines to use photonic interconnects extensively, especially to support communication between far-away nodes in larger machines. Some researchers are also proposing the use of photonics for on-chip interconnects,[6] targeting the technology to large, high-bandwidth message transfers on chip. An area of intense current research is efficient interfaces between electronic and photonic signaling.

Other system technologies. Several other technologies will likely significantly impact energy and power efficiency. An obvious one is 3D die stacking, which will reduce memory access power. A 3D stack might contain a processor die and memory dies, or it might contain only memory dies. The resulting compact design eliminates energy-expensive data transfers, but introduces manufacturing challenges, such as the interconnection between stacked dies through vias. Interestingly, such designs, by enabling high-bandwidth connections between memories and processors, might also induce a reorganization of the processor's memory hierarchy. Very high bandwidth caches near the cores are possible.

Efficient on-chip voltage conversion is another enabling system technology. The goal here is for the machine to be able to change the voltage of small groups of cores in tens of nanoseconds, so they can adapt their power to the threads running on them or to environmental conditions. A voltage controller in each group of cores can regulate the group's voltage. Hopefully, the next few years will see advances in this area.

Enabling concurrency and locality

In general, the performance of future energy-efficient extreme-scale machines will not be attained through high frequency—dynamic power consumption is roughly proportional to the cube of the frequency. Instead, circuits will likely be designed for low voltages and modest frequencies. Consequently, we will have to rely on more threads running concurrently.

Assume, for example, 1-GHz cores, each completing one operation per cycle. In this case, a chip will need 1,024 cores to attain one tera-op, a server will need about 1 million cores to attain one peta-op, and a data center will need about 1 billion cores to attain one exa-op. In reality, a thread will often stall waiting for data, rather than committing one operation per cycle. Consequently, to hide the stall time and attain the desired performance level, the system will have to support several times more threads. Extreme-scale architectures will need memory hierarchy organizations, synchronization primitives, and network links that support these concurrency levels. Moreover, designers cannot optimize such structures in a way that ignores, let alone penalizes, locality. Exploiting high degrees of spatial and temporal data locality is the only way to attain the desired performance at the target power budget.

We suggest two architectural supports to enable the fine-grained parallelism that extreme-scale machines require. The first support is efficient, scalable synchronization and communication primitives—especially for dynamic and irregular parallelism. These primitives must provide efficient point-to-point synchronization between two cores and collective operations.[7] Examples include low-overhead dynamic hierarchical barriers, producer-consumer synchronization through full-empty bits, and broadcast updates.

The second architectural support for fine-grained parallelism is low-overhead primitives for the creation, commit, and migration of lightweight tasks. Such lightweight tasks are likely to be created by the compiler, spawned with a single instruction, and managed with scalable queuing structures that minimize overheads and stall times.

In addition, several hardware architecture structures can help minimize data movement and exploit locality.

A processing-in-memory unit typically performs memory-intensive operations on arrays or sets of data.

The most obvious one is a many-core chip organization based on clusters. A cluster is a set of cores that have physical proximity and share some cache or other storage structure. The compiler can break a task into smaller subtasks and assign them to the cores in a cluster.

A second structure for locality is simple compute engines in the memory controllers or in the L3 cache controllers that perform certain memory-intensive computations. Such an approach—often known as processing-in-memory (PIM)—seeks to avoid transferring large amounts of data between the memory and the main cores and then back to perform a simple computation. PIM can be embodied in a variety of hardware—from simple functional units to specialized compute engines. A PIM unit typically performs memory-intensive operations on arrays or sets of data, such as element-by-element operations, reductions of various sorts, and recurrences.

Finally, although conventional cache hierarchies seek to minimize data movement, it is important to note that they sometimes end up moving sizable amounts of data unnecessarily. They do this through their use of cache lines and automatic mapping of lines in the cache. Mechanisms to prevent such data movements are needed.

Bolstering resiliency

Resiliency will be another key challenge in extreme-scale machines due to a combination of several effects:

- Spatial variations in process, voltage, and temperature, as well as logic wearout (aging), will likely become relatively more acute as semiconductor feature sizes decrease.
- Smaller feature sizes imply less charge in storage elements, making these elements more vulnerable to soft

Table 1. Characteristics of several popular techniques for resiliency.

Level	Detection and isolation	Recovery
Hardware	Error-correcting codes (ECC) for on- and off-chip memory structures; parity for processor data path and network; testing circuitry, sensors, detectors, and watchdog timers; hardware replication; checker engines or cores	Low-overhead checkpointing and rollback; hardware reconfiguration, including disabling or salvaging the faulty component
Operating system/ runtime	Resiliency module to diagnose and isolate	Job scheduling around faulty components; virtualization
Compiler	Augmenting the code with checks	Compiler support for checkpointing
Application/ programming system	Applications that check themselves	Applications that checkpoint themselves; transactional model

errors induced by particle impacts.

- The use of V_{dd} values that are close to V_{th} will increase process variation.

All these effects will increase the chances of transient and permanent faults. At the same time, the largest extreme-scale machines will have many components, which will also increase the chance of faults. For example, an exa-op supercomputer might have 10 to 100 petabytes of memory, requiring tens or hundreds of millions of memory chips. The machine might also have hundreds of exabytes of secondary storage, requiring millions of disk drives.

No single solution can fully address the resiliency challenge. Instead, acceptable resilience in extreme-scale architectures might only be attainable through a combination of techniques at different levels of the computing stack. Each of these techniques has advantages and shortcomings, making the most cost-effective combination a complex function depending on, among other things, the workload executed.

Table 1 lists some of the most popular techniques for resiliency. We classify them based on whether they detect and isolate (and sometimes also correct) errors or recover from errors. Within each group, we classify the techniques based on the level at which they operate—namely, hardware, operating system and runtime, compiler, or application and programming system.

Several popular techniques support error detection and isolation in hardware. They include error-correcting codes (ECCs)—which also correct errors—for on- and off-chip memory structures and parity for processor data path and network links. Extreme-scale architectures might need to augment single-bit parity in the data path and network with more expensive detection and perhaps even correction codes. Testing circuitry (exercised right after manufacture and periodically in the field), various sensors and detectors (wearout, power, and temperature), and watchdog timers to detect timeouts will likely also be needed. Widespread hardware replication or using engines or cores to check the work of other cores are probably too expensive for these machines.

Software can also help detect and isolate errors. A resiliency module can diagnose and isolate (and even correct) the error at runtime, while a compiler can augment the code with checks on code control flow or data accesses. Finally, some applications can check themselves.

Error recovery is typically based on checkpointing and rollback. Checkpointing can be implemented in different layers. For example, as Table 1 shows, it can be supported in hardware. One hardware-based implementation with especially low overhead is ReVive,[8] which uses in-memory, incremental checkpointing—possibly coupled with nonvolatile memory.

Checkpointing can also be compiler driven. In this case, the compiler, fully aware of the program state at any point, chooses to checkpoint when the program has the least state. Finally, the application itself can decide when to checkpoint, again based on the size of the program state. In general, checkpointing should induce only minor overhead during error-free operation and, in a rollback, result in little work loss and in low error-recovery latency.

Unfortunately, conventional checkpointing consumes substantial time and power, and requires high disk bandwidth and capacity. As the processor core count increases to more than one billion for exa-op machines, conventional checkpointing can consume most of the execution time and power, hence becoming infeasible. Thus, effective checkpointing support in extreme-scale architectures is a major challenge. These machines will require novel, highly energy-efficient checkpointing and rollback mechanisms that combine techniques from multiple layers of the stack.

Other recovery mechanisms rely on reconfigurable or redundant hardware, such as spare cores. On an error, the hardware can be reconfigured and the faulty component can be disabled or salvaged. The operating system or runtime system can schedule jobs around faulty components, and even rely on virtualization to transparently mask away the faulty component. Finally, at the application and programming system level, the application can be written using intrinsically resilient programming models. For example, in the transactional model, the transaction is rolled back if an error occurs.

Designing for programmability

Programming highly concurrent machines has traditionally required heroic efforts. Extreme-scale architectures, with their emphasis on power efficiency, can

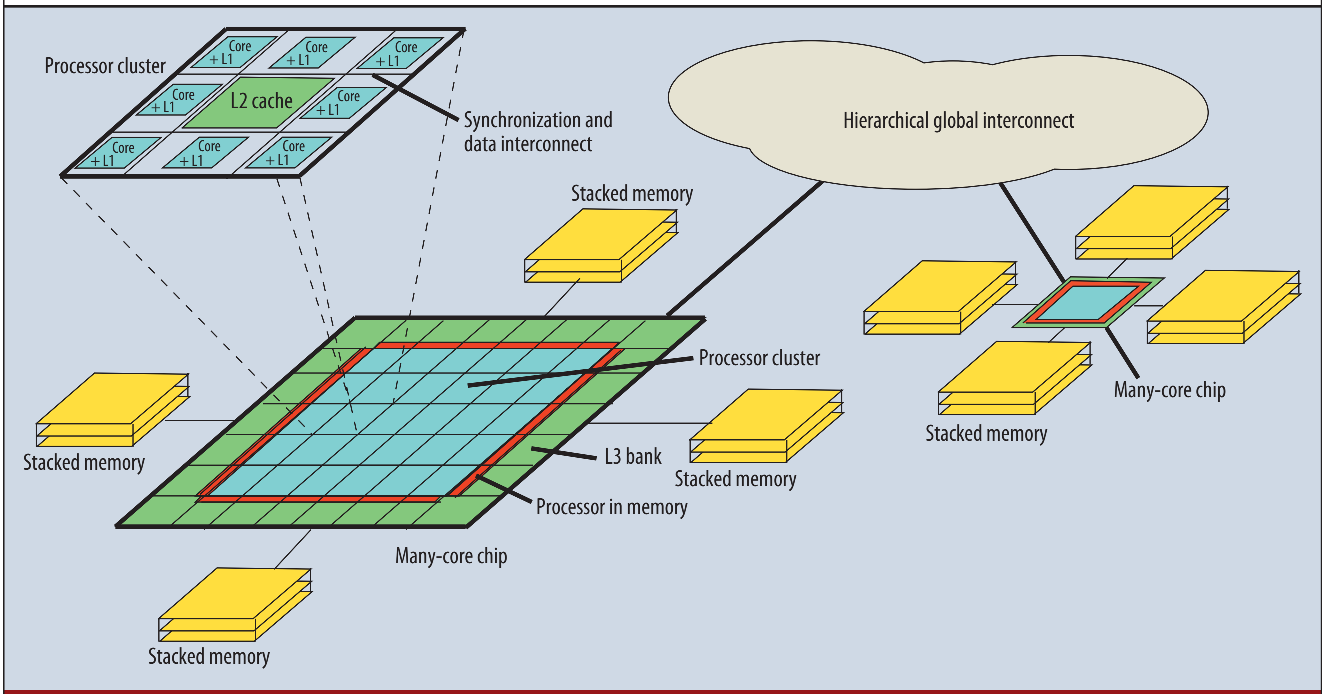


Figure 1. Overview of the Thrifty architecture. The architecture comprises 1,024-core many-core chips.

make the task even more difficult. Indeed, by keeping V_{dd} low, they need several times more concurrency to attain the same performance level. Moreover, power efficiency mandates carefully managing locality and minimizing communication. All of these requirements can easily increase programming complexity.

Programmers of extreme-scale machines must be able to express a high degree of parallelism in a way that does not preclude careful locality management and communication minimization. An appealing approach is to program the machine using a high-level programming model, and then rely on intelligent static and dynamic compilation layers to efficiently map the code to the hardware. The most appropriate high-level model will likely be a data-parallel one, where programmers apply high-level operations to data aggregates.[9] This model can compactly express high degrees of parallelism. With this model, execution appears as computation segments, in which the cores compute largely independently, logically separated by barriers, with some data shuffling between the segments.

A compiler takes the computation assigned to individual elements of the data aggregate and, possibly driven by programmer annotations, breaks it into fine-grain tasks. Then, the compiler maps these tasks to cores, trying to leverage the different locality levels provided by the hardware, such as core cluster, chip, and node. Because the architecture is complex, this technique requires a level of code autotuning or adaptation. Autotuning can be provided through libraries that generate multiple versions of code with different parameters and choose the best one, or through continuous optimization, an approach that relies on a slow adaptation of the program as it runs.

Several architectural features can help support the execution environment described. For example, the machine can provide a single address space to the software. With such a capability, programmers developing irregular codes have a substantially simpler task. Hardware mechanisms that support compiler optimizations and high-level languages are also helpful. For example, such mechanisms could detect dependences within and across threads inexpensively[10] or manage the caches in software. The architecture can also provide features to detect data transfer patterns and eliminate, minimize, or hide data movement. For example, this includes efficient primitives for prefetching, multicast-update of the copies of a datum, and movement of computation to the data's location. It also includes efficient implementations of synchronization primitives—in particular, various forms of barriers.

Finally, autotuning libraries and continuous optimization software will benefit from tight coupling with performance or energy-monitoring hardware structures, such as counters, signatures, and trace buffers. Ideally, these structures should be programmable by the user software. They should also enable a low-latency feedback loop to the application, so program adaptation can be effective.

OUTLINE OF AN EXTREME-SCALE ARCHITECTURE

Figure 1 gives an overview of the Thrifty extreme-scale architecture concept, which we are developing. The

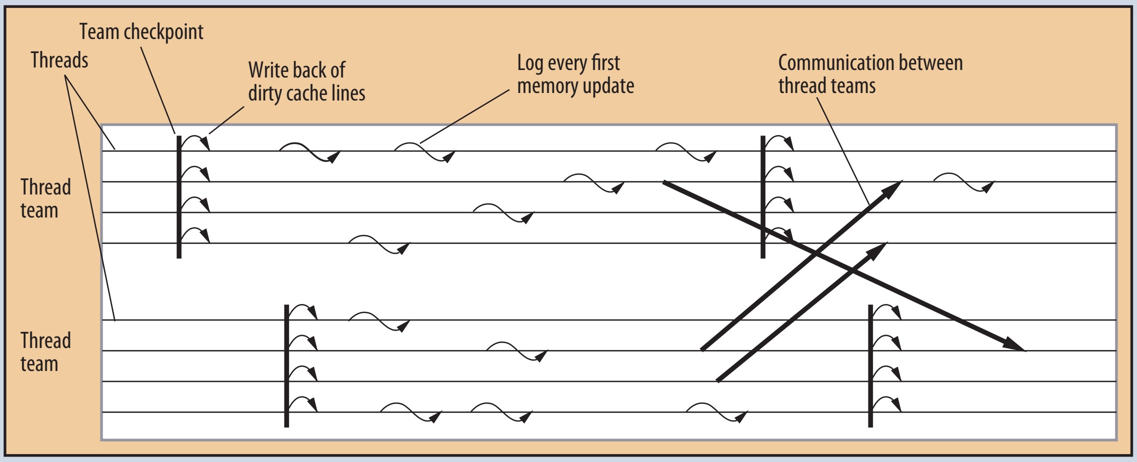

Team checkpoint
Threads
Write back of dirty cache lines
Log every first memory update
Communication between thread teams
Thread team
Thread team

Figure 2. Hierarchical operation of ReVive. Threads are organized in teams, and each team independently follows the ReVive operation. This involves performing regular team checkpoints, in which processors write their registers to memory and caches write their dirty lines to memory. Between checkpoints, a controller logs the value in memory before every first memory update. Communication between teams is explicitly recorded and causes additional checkpoints.

architecture comprises 1,024-core many-core chips, and supports a shared address space. Because Thrifty has no hardware cache coherence, the software must maintain data coherence.

The chips' cores are homogeneous, two-issue, in-order, and single-threaded. They have floating-point support and a reconfigurable vector unit. To enable locality optimizations, groups of eight cores are organized in a cluster, sharing an L2 cache. Each group of four clusters is an independent voltage and frequency domain, because Thrifty relies heavily on voltage and frequency changes to control its power-performance operation. All the clusters share an on-chip L3 cache, which is banked and distributed across the chip. Each L3 bank has a simple PIM engine that performs reductions and other simple memory-intensive operations. The chip has efficient hardware primitives for synchronization and communication (including hierarchical barrier and multicast update). A wide hierarchical network connects the cores.

The machine can connect thousands of these chips hierarchically into boards and cabinets, using a network organized in a fat tree. Memories are organized in 3D stacks and associated with PIM engines. A low-overhead, in-memory incremental checkpointing system based on a hierarchical form of ReVive provides recovery for applications that are explicitly willing to pay for it.[8] The scheme is outlined in Figure 2.

The architecture is programmed with high-level, data-parallel operators on data aggregates. Tasks are mapped to processor clusters. A back-end compiler further divides these tasks into small subtasks, which it then maps to the cores in a cluster for locality.

Substantial advances in architecture and hardware technologies should appear in the next few years. For extreme-scale computing to become a reality, we need to revamp most of the subsystems of current multiprocessors. Many aspects remain wide open, including effective NTV many-core design and operation; highly energy-efficient checkpointing; rearchitecting the memory and disk subsystems for low energy and fewer parts; incorporating high-impact technologies such as nonvolatile memory, optics, and 3D die stacking; and developing cost-effective cooling technologies. A new generation of computer designers will deliver these advances. C

Acknowledgments

I thank Peter Kogge and the writers of the ExaScale report. I also thank Rishi Agarwal, Pranav Garg, David Padua, Maria Garzaran, Brian Greskamp, Ulya Karpuzcu, Shekhar Borkar, David Dunning, Allan Snavely, and William Harrod.

References

1. P. Kogge et al., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, DARPA Information Processing Techniques Office (IPTO) sponsored study, 2008; www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.
2. E. Grochowski and M. Annavaram, “Energy per Instruction Trends in Intel Microprocessors,” *Technology@Intel Magazine*, Mar. 2006, pp. 1-8.
3. R. Dreslinski et al., “Near Threshold Computing: Overcoming Performance Degradation from Aggressive Voltage Scaling,” *Proc. Workshop Energy-Efficient Design*, 2009, pp. 44-49.

4. B. Lee et al., "Architecting Phase Change Memory as a Scalable DRAM Alternative," *Proc. Int'l Symp. Computer Architecture* (ISCA 09), ACM Press, 2009, pp. 2-13.
5. R. Ramaswami and K. Sivarajan, *Optical Networks: A Practical Perspective*, Morgan Kaufmann, 2nd ed., 2002.
6. D. Vantrease et al., "Corona: System Implications of Emerging Nanophotonic Technology," *Proc. Int'l Symp. Computer Architecture* (ISCA 08), IEEE Press, 2008, pp. 153-164.
7. J. Shirako et al., "Phasers: a Unified Deadlock-Free Construct for Collective and Point-to-Point Synchronization," *Proc. Int'l Conf. Supercomputing* (SC 08), ACM Press, 2008, pp. 277-288.
8. M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors," *Proc. Int'l Symp. Computer Architecture* (ISCA 02), ACM Press, 2002, pp. 111-122.
9. J. Brodman et al., "New Abstractions for Data Parallel Programming," *Proc. First Usenix Workshop on Hot Topics in Parallelism* (HotPar), Usenix Assoc., 2009; www.usenix.org/event/hotpar09/tech/full_papers/brodman/brodman.pdf.
10. J. Tuck et al., "SoftSig: Software-Exposed Hardware Signatures for Code Analysis and Optimization," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 08), ACM Press, 2008, pp. 145-156.

***Josep Torrellas** is a professor and Willett faculty scholar in the Department of Computer Science at the University of Illinois at Urbana-Champaign. His research interests include computer architectures, parallel computing, hardware and software reliability, and low-power design. Torrellas received a PhD in electrical engineering from Stanford University. He is an IEEE Fellow and a member of the ACM. Contact him at torrellas@cs.uiuc.edu.*