

Advanced Architectures and Execution Models to Support Green Computing

Creating the next generation of power-efficient parallel computers requires a rethink of the mechanisms and methodology for building parallel applications. Energy constraints have pushed us into a regime where parallelism will be ubiquitous rather than limited to highly specialized high-end supercomputers. New execution models are required to span all scales, from desktop to supercomputer.

Over the next decade, computer architecture will change dramatically to adapt to increasingly challenging power envelope constraints. Thus far, the initial changes have been a reactive transition to multicore architectures. However, as supercomputers transition from petascale to exascale systems, we'll have to dramatically rethink the underlying way in which computation is performed. This reevaluation could trickle down to all scales of computers and facilitate truly high-efficiency, lower-power operation. The previous high-end transition (from terascale to petascale) didn't face fundamental limits in the underlying implementation technology (the complementary metal-oxide semiconductor, or CMOS). Instead, the principal concern related to fabrication processes and manufacturability. Today, the limits are based on fundamental physics, as exhibited by the dynamic power equation:

$$P = CV^2f$$

where P is power, C is capacitance, V is voltage, and f is frequency. Historically, CMOS scaled by decreasing the capacitance per device, while increasing the total number of devices with Moore's law. Successive generations also experienced increases in clock frequency. Despite the exponential growth derived by increasing capacitance and clock frequency, decreasing supply voltages held the entire equation in balance.

The current transition to multicore architectures is driven by the power equation's fundamental requirements: supply voltages won't decrease significantly due to switching noise, requiring a flattening in the clock frequency. In fact, there's concern about whether it's even possible to further decrease individual device capacitance. Consequently, performance gains are realized by increasing the number of cores (as the number of devices still grows with Moore's law in every generation), but clock frequencies are essentially flat—or in some cases declining—while core architecture complexity also remains relatively constant.

Unlike prior generations, the CMOS roadmap's physical limits require fundamental changes in the architecture. However, at the high end, simply moving to multicore computers won't achieve an additional three orders of magnitude of performance within a constrained power budget by the decade's end. As a result, supercomputers must

1521-9615/10/\$26.00 © 2010 IEEE
COPUBLISHED BY THE IEEE CS AND THE AIP

RICHARD MURPHY

Sandia National Laboratories

THOMAS STERLING AND CHIRAG DEKATE

Louisiana State University

be designed to provide significantly better power performance. This contrasts sharply with previous decades, when reliance on commodity parts drove supercomputer evolution at the cost of power and device efficiency.

Today, there's a real opportunity to redesign computers to be highly power efficient. The key is to enable highly parallel fine-grained systems and let the architecture exploit explicitly defined work—rather than “guess” when information isn't yet available, using prediction or speculation to advance program execution. This approach contrasts with most computer architectures since the microprocessor's invention. As we describe here, the ParalleX execution model^{1,2} can enable the transition to more power-efficient computing. Not only does it fundamentally expose parallelism, it lets the architecture take advantage of that parallelism explicitly, rather than implicitly, and reduces critical communication overheads prevalent in today's models of computation.

New Concepts in HPC Architecture

Over the next 10 years, we'll see dramatic changes in how we structure and operate our computer systems—from the smallest handhelds, embedded, and mobile computers to the largest supercomputers, which will deliver a thousand times the capabilities of today's most powerful computing complexes. These changes, which initially will be almost imperceptible, will accelerate such that within this decade, new systems and their applications will be based on almost entirely different concepts than conventional practices today.

Such developments will result from the imperative to achieve exaflops sustained performance before the year 2020 through innovative technology advances. These changes—including a possible paradigm shift—are further necessitated by the severe challenges that are already impeding continued progress in sustained performance. Future system architectures will embrace new or at least untraditional concepts so as to exploit the performance opportunities afforded by raw technology advances, while also providing innovative approaches to circumvent constraining factors. But, as we now discuss, system architecture must change in several key ways if performance gains are to meet the goal of practical exaflops capability by the decade's end.

The Multicore World

All projections support the contention that Moore's law, in its basic sense of increased transistor density on semiconductor dies, will continue

to quadruple every three years. By 2018, components using 11-nanometer feature size should be commercially available. Such sustained device density improvements will continue to drive performance advances. However, two other common methodologies critical to performance gains of the past can no longer be relied upon as a means to future performance.

The first is clock rate, which—along with device density—has increased continuously by a factor of a thousand over the last three decades from a few megahertz in the 1970s to a few gigahertz in 2000. This has been a key factor contributing to sustained improvements in overall performance gain. However, clock-rate increases have resulted in increases in per-processor power consumption, which have now reached the upper limits of practical acceptance, even with the mitigating contributions of reduced logic voltage levels. Clock rate shouldn't increase over

Within this decade, new systems and their applications will be based on almost entirely different concepts than conventional practices today.

the coming decade as much as it has over the previous two, which will largely eliminate an important contributing factor to performance increase. Also, voltage levels will diminish only slightly because we're now reaching the limits of reliable circuit switching. Thus, we can expect little future power advantage from this other source of past improvements.

Design complexity is another methodology previously exploited for performance gain that's unlikely to significantly contribute in the future. In the past, microprocessor design was primarily constrained by the number of available transistors and the transistor switching rate. As transistor density increased with semiconductor technology advances, the complexity of microprocessor design could increase as well, instilling such processors with important architecture improvements often previously developed for and adopted from earlier-generation mainframe computers. But with increased design complexity came diminishing returns in performance achieved per transistor. Eventually, the inflationary period of performance advantage through complexity

reached a point of diminishing returns. Indeed, the use of hardware-supported speculative computing was one factor in increasing power consumption. The current era in processor core size has largely eliminated processor complexity as the second important source of performance gain.

Impact of Energy Limits on Computer Architecture

Future performance gains, still exploiting device density increases through the end of this decade, will be achieved through system and core architectures. System architectures will represent organizations of ever-increasing numbers of processor cores through multicore (increases in cores per die), stacked dies to provide more cores per socket, and increased sockets per node and even nodes per system. Upwards of a billion cores might be integrated as a single system to achieve delivered exaflops performance by 2020. To realize this performance, we'll need unprecedented levels of application-program parallelism of multibillion-way concurrency. Core architecture will change—not to add complexity, but rather to make their

Exploiting the efficiency gap through new methods and structures can bring cost and power within practical limitations and ensure future availability of multi-exaflops systems into the next decade.

mutual integration and interoperability in large distributed ensembles more efficient.

A second contributing factor to achieving required performance (at a continued rate of growth) is *efficiency*. Although some benchmarks (such as high-performance Linpack³) and a few applications exhibit efficiencies—that is, a fraction of sustained floating point performance with respect to peak floating point performance—more than half of many large-scale, mission-critical applications deliver less than 10 percent, with 3 to 5 percent being common (and some unfortunate cases of less than 1 percent). With power concerns dominating future system concepts, energy and temporal efficiency are essential. Jaguar, today's most powerful system (as measured by the high-performance Linpack benchmark), can consume up to 7 megawatts (MW) of power, more than an

order of magnitude greater than that of supercomputers a decade ago. The threshold of pain according to some experts is in the range of 10 to 25 MW. But current estimates of 2020 exascale systems are 120 MW (+/- 50 percent); this being sensitive to many assumptions of form and function. Exploiting the efficiency gap through new methods and structures can bring cost and power within practical limitations and ensure future availability of multi-exaflops systems into the next decade.

A New Path Forward

To achieve the necessary scalability and efficiencies required, a break from conventional practices in structure, operation, and usage is essential. We've identified four key efficiency factors—represented by the acronym SLOW—that together contribute to most performance degradation in today's systems; these factors must be addressed if we're to realize practical exaflops-scale systems.

- **Starvation**—insufficient work to keep all processing units working; this is either because of inadequate work or improper balance of work allocation across system resources. Starvation is addressed through *task parallelism*.
- **Latency**—the distance measured in time (often cycles) for remote accesses and services, such as those to memory or other processors. Latency is mitigated by *avoidance through locality management*, including caching and hiding via multithreading, and *future message-driven computing techniques*, such as with active messages.
- **Overhead**—the critical path extra work that systems must perform to manage parallel resources and the computing that wouldn't be necessary for pure, sequential implementations of the same algorithms. Overhead is addressed through *new hardware mechanisms* to support global parallel computing and efficient runtime software techniques.
- **Waiting for contention**—delays to computation further experienced because of task blocking resulting from contention for shared resources, including memory bank conflicts, communication networks, or guarded local control objects for synchronization (such as semaphore synchronization variables). Contention is addressed by *augmenting bandwidth* to communication channels and concurrent hardware function units. (This might be counterintuitive as computer designers can typically adjust bandwidth easily, and thus don't enumerate it

explicitly. However, latency is the dominant performance parameter.^{4,5})

To address these four critical performance factors, future architectures of both processor cores and complete systems might vary significantly from both past-generation massively parallel processors (MPPs) and commodity clusters in structure and semantic mechanisms.

First, it's likely that processor cores will become simpler to reduce the average energy per operation. Although this might reduce the throughput of useful operations slightly, it will greatly improve the power consumption per processor core and also require less die real estate to improve spatial efficiency. Such cores will become closer to those of embedded computing today, although they'll be optimized for a different application space.

Second, processor cores will be designed explicitly to work synergistically with a billion other cores throughout the largest systems. This is quite different from current processors, which are designed either as standalone or to work with a few other processors in a cache-coherent node and rely on slow I/O channels and system software to increase scalability.

Third, future processor cores will support global address space (GAS) to permit efficient and immediate access to remote virtual objects anywhere in the system. While not cache coherent—which can be costly in time and power—such direct access to global system resources, both logical and physical, can greatly reduce overhead and latency. One such model is partitioned GAS (PGAS).⁶

Fourth, we'll shift from message-passing methods using very long packets to message-driven methods that will sometimes use relatively short packets. Although known for decades, message-driven communication and coordination hasn't been widely employed, except in some wide-area semantics like remote procedure calls (RPC). Message-driven computation, such as University of California, Berkeley's active messages,⁷ lets work move to data anywhere in the system rather than always requiring data to be gathered and delivered to statically located work. Average latency can be significantly reduced and energy per remote operation can be dramatically decreased. Using this method might also permit data-directed execution methods to dynamically expose the parallelism found in the metadata of complex global data structures, such as the directed graphs of adaptive mesh refinement (AMR) algorithms

and knowledge-management problems to address the starvation challenge.

Fifth, hardware support for efficient synchronization objects—such as the *futures*⁸ construct—will greatly reduce overhead costs and let us effectively exploit finer-grained parallelism and thus address starvation as well. Dataflow synchronization objects are fine-grained enough to enable the elimination of global barrier synchronization.^{9,10} Freeing ourselves from the bulk synchronous model (which is the only one that today's architectures efficiently support) is the only path to the parallelism levels required for future-generation architectures.

Sixth, we'll replace conventional static methods of task resource assignment and scheduling, as well as static distribution of largely regular data structures (such as dense matrix partitioning), with dynamic adaptive methods for best operation as runtime conditions vary. Thread context switching, load balancing, asynchronous communication, and local control objects will exploit runtime information. Processor architecture features supporting dynamic control will address all four sources of performance degradation.

Finally, active power management and fault-tolerance mechanisms will be included within the new architectures. Where core logic is used only lightly, clock rates will be lowered to further reduce power consumption. Fault detection, isolation, and reconfiguration mechanisms will be incorporated for fault tolerance. The mean time between failures of future systems comprising hundreds of millions of cores will be too short to support conventional check-pointing and restart methods.

Not all of these points are obvious; conventional practices often successfully rely on more pervasive static binary space partitioning (BSP)-like methods. The direct exploitation of runtime information is valuable only if operation is unpredictable because of data-value dependencies, as in the solution to highly nonlinear equations in inner loops. Although execution-time variation among different instances of such loops can be dramatic, global barrier synchronization always proceeds at the slowest loop's rate. Using lightweight mechanisms such as futures (our fifth factor) provides dynamic control of execution flow, permitting not only overlap of computation with communication but also of multiple computation phases adapting to natural flow rates of the computation itself. These and other features will emerge as new design structures and operational

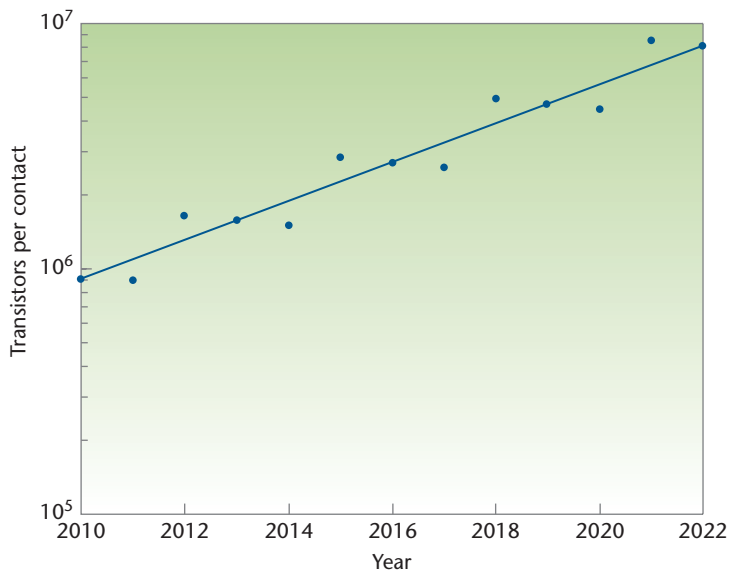


Figure 1. Growth in transistors relative to off-chip communication. The dots show individual points in time; the line extrapolates the trend.

mechanisms incorporated in the new core architectures of future exascale computer systems.

Why Technology Trends Demand Improved Efficiency

Technology evolution, including recent trends in CMOS technology, drives computer and system architectures to advance and change to better address scalability, efficiency, and user productivity. The architectural trends we described earlier are the result of long-term CMOS trends that have led to the creation of supremely unbalanced architectures. Unfortunately, these trends show no sign of abating, and the lack of balance leads to machines that are difficult to scale, particularly at the high end. Worse, the applications supported by those machines are restricted to those that map well onto the hardware, and might not reflect the kinds of applications required in the future.

Currently, relatively structured physics applications are well supported by supercomputers, but highly unstructured codes are not. With technology also pushing toward requiring structured memory accesses to save energy, the community might simply choose to leave many problems unsolved.

The New Memory Wall

Historically, the disparity between CPU and memory performance has been expressed by the memory wall, which resulted from the trend toward increased relative memory latency in every processor generation (basically, processor clock

rates were increasing much faster than memory clock rates).⁴ Because the power dissipation of higher clock rates is no longer tolerable, we might expect the memory wall to improve. Unfortunately, poor memory performance can't entirely be accounted for by latency. Little's law (from economics) provides a better intellectual model for the memory system, dictating that

$$\text{Throughput} = \text{Concurrency} / \text{Latency}.$$

With relative latency no longer increasing with Moore's law, concurrency might be the primary influence on memory system performance. Unfortunately, the underlying technology simply will not keep pace. Today, increased memory concurrency is achieved by adding additional memory channels, requiring off-chip communication paths. These paths are supported by a series of off-chip contacts or pins that support wires between two packaged chips. With the number of cores still projected to increase with Moore's law, maintaining a constant number of memory channels per core would require the same exponential increase, which simply doesn't exist.

Figure 1 shows the disparity between a Moore's law growth in transistors and the expected growth in contacts as projected by the International Technology Roadmap for Semiconductors (ITRS) roadmap.¹¹ By 2022, each chip's off-chip communication contact will be required to support nearly an order of magnitude more transistors than today. Moving to high-speed serialized communication channels with a packetized interface will almost certainly improve the situation, and more advanced packaging technologies, such as 3D integration, could radically change how we build computers. However, as things stand today, future technology will provide significantly less potential communication on a per-core basis. This is the modern expression of the memory wall, and continues to plague computer architecture. It seems likely that, without fundamentally changing how we build computers, a tremendous excess of compute capability will exist for local operations only. Given this, contention will likely replace latency as the most severe component of the future memory wall.

From the supercomputing community's standpoint, the technology trends supporting the drive to exascale are very different from those that supported the drive to petascale. The primary constraint on achieving the next three orders of magnitude in performance is power. Projecting today's systems into exascale performance, even

adjusting for technology improvements, will result in supercomputers that exceed 100-MW power budgets.¹² In a sense, the drive to peta was more about potentially accelerating a known roadmap's delivery date, and the drive to exa about the feasibility of producing such a system at all. Aggravating this are the additional constraint factors of parallelism and resiliency in the presence of faults.

Finally, we'll likely near the end of the CMOS roadmap and require a transition to a new device technology somewhere within the lifetime of the next decade's trans-exascale performance regime. Indeed, limits on CMOS devices are now dictated more by fundamental physics than past predictions of "the end" were by manufacturing challenges. Speed of light, Boltzmann's Constant, and atomic granularity all contribute to the limitations of fundamental physics for future device technologies.

Future Architecture Research Directions

To address the combined communication (technology) and power (architecture) problems, research into future architectures has shifted from realizing "compute" to more efficiently managing communications, including memory access.

There are three key areas of investment:

- Improved communication channel bandwidth and power, with optical communication the most likely candidate for a technological solution. Thus far, the cost of integrating optical systems (from a different fabrication process) with logic has proven the most challenging impediment to wide-spread adoption.
- Improved packaging to let local structures communicate more efficiently and, more importantly, to allow the creation of heterogeneous structures. Efforts to combine heterogeneous elements, such as processing-in-memory (PIM), typically focus on combining two heterogeneous elements—in this case, logic and dynamic RAM (DRAM)—when in fact combining multiple elements will be required in the future. Examples include logic, DRAM, and optics. 3D integration appears the most promising solution to this problem.
- Innovation in communication interfaces (processor-to-memory and system-level interconnect) and topologies to take advantage of the technologies developed by solving the first two problems. Additionally, by closely matching the interconnect's communication semantics with those of the system's programming models, we

can make greater energy efficiency gains by reducing communication overheads.

This investment, combined with architectures supporting improved efficiency, will better support a solution.

New Applications

In addition to technological and architecture changes, we must also transition the high-end machine application base from 3D physics simulations—which have a clear spatial decomposition, well-established programming models, and relatively intensive flops—to large, combinatorial, data analytics problems that tend to have no clear decomposition. Such problems also require new programming models and are primarily integer applications. Examples of these problems include

- large-scale graph problems, including social network analysis;
- semantic networks, representing linguistic and neural-network data;
- combinatorial analysis of genomic and other biological data;
- modeling of biological networks, such as pandemic flu outbreaks; and,
- other complex network modeling, such as the smart power grid.

These problems start as large-scale analytics problems, and can be solved only with large-scale systems today. They tend to be more data intensive (performing more unique accesses to memory) and exhibit less locality (the basis for modern cache systems).^{5,13}

In any programming model, the challenges posed by such applications require architecture enhancements, particularly to the data movement system, which dominates any computer's power. Specifically, they need

- increased capability to support numerous small messages in flight,
- access to smaller data items in a unit (words instead of cache lines), and
- enhanced synchronization mechanisms to manage increased parallelism.

Given such architectural enhancements, these applications hold tremendous promise for providing sufficient parallelism to keep large-scale parallel systems continuously busy doing useful work, which helps the energy budget. In cases where

this parallelism doesn't exist, hardware speculation and other complex, energy-inefficient mechanisms can achieve higher performance. However, because these applications are parallel by nature, they could lead to more energy-efficient programming models.

Implications for Programming Models and System Software

A high-performance computing system is more than just computer architecture. To achieve true efficiency, scalability, and programmability in the exascale era, we'll need new system software and programming models as well as to support the innovative computer architectures we described earlier.

Historically, when architecture responded to technology advances with dramatically new forms and function, programming models changed to support them. Typically, corresponding paradigm shifts inspired such changes and provided the conceptual framework for codesign of all system layers (including architecture) and their joint operation. Unique computation models were used in vector processing; single instruction, multiple data (SIMD); array processing; systolic arrays; experimental dataflow systems; and conventional MPPs and commodity clusters using the communicating sequential processing model. We believe that a next-generation system capable of delivering exaflops performance using novel architectures will be organized, coordinated, and programmed by a new execution model that will open up new opportunities for operations. Such operations will address the challenges previously described to achieve exaflops-sustained performance within practical constraints.

Current Research Directions

Ongoing research is exploring one possible computation model to govern development of programming models and supporting system software. Such a model must

- expose a far greater abundance of parallelism,
- intrinsically hide latency effects,
- dynamically work around bottlenecks caused by resource contention, and
- recover from faults.

This work, from multiple sources, suggests that such a model will provide the conceptual framework for future systems, incorporating several key logical elements:

- parallel processes that are ephemeral (can be created and destroyed) and span multiple (possibly overlapping) nodes in the global address space;
- dynamic multithreading (some potentially lightweight) that are also ephemeral and supported by rapid context switching for low overhead and latency hiding;
- message-driven computation that moves work to remote sites to work on local data;
- lightweight control objects for synchronizing the global computation and eliminating global barriers;
- dynamic resource management, scheduling, and allocation; and
- self-aware operational status to respond to faults, security threats, and power usage.

Figure 2 shows one possible model—the ParalleX model^{1,2}—that could replace the conventional communicating sequential-processes model¹⁴ that's widely employed on scalable distributed memory systems. The ParalleX experimental framework explores the synthesis of key concepts that address exascale computing challenges. It borrows heavily from selected prior art, synthesizing them into a new starting point for exploring HPC. ParalleX also addresses the scaling challenges experienced by some problem classes, including adaptive mesh refinement and molecular dynamics.

ParalleX comprises four major classes of constructs: parallel processes, first-class threads, local control objects (LCOs), and parcels. ParalleX processes offer contexts for data, code objects, threads, LCOs, and child processes. They're parallel in that, within their context, they can simultaneously execute multiple threads and child processes. They also differ from MPI¹⁵ processes in that any one process might employ multiple hardware nodes, possibly with many cores in each. Such processes might even share one or more such nodes.

Threads are bits of executing operations that share intermediate values and local control; they're first class in that they're named in the same address space as common global data, and other threads can manipulate them directly if necessary. A thread's internal state control need not be sequential, and using single assignment operators assumes a form similar to static dataflow, except that global mutable variables are also accessible and easily manipulated. LCOs are small objects, each residing within a single locality (a kind of node) that unifies synchronization's disparate

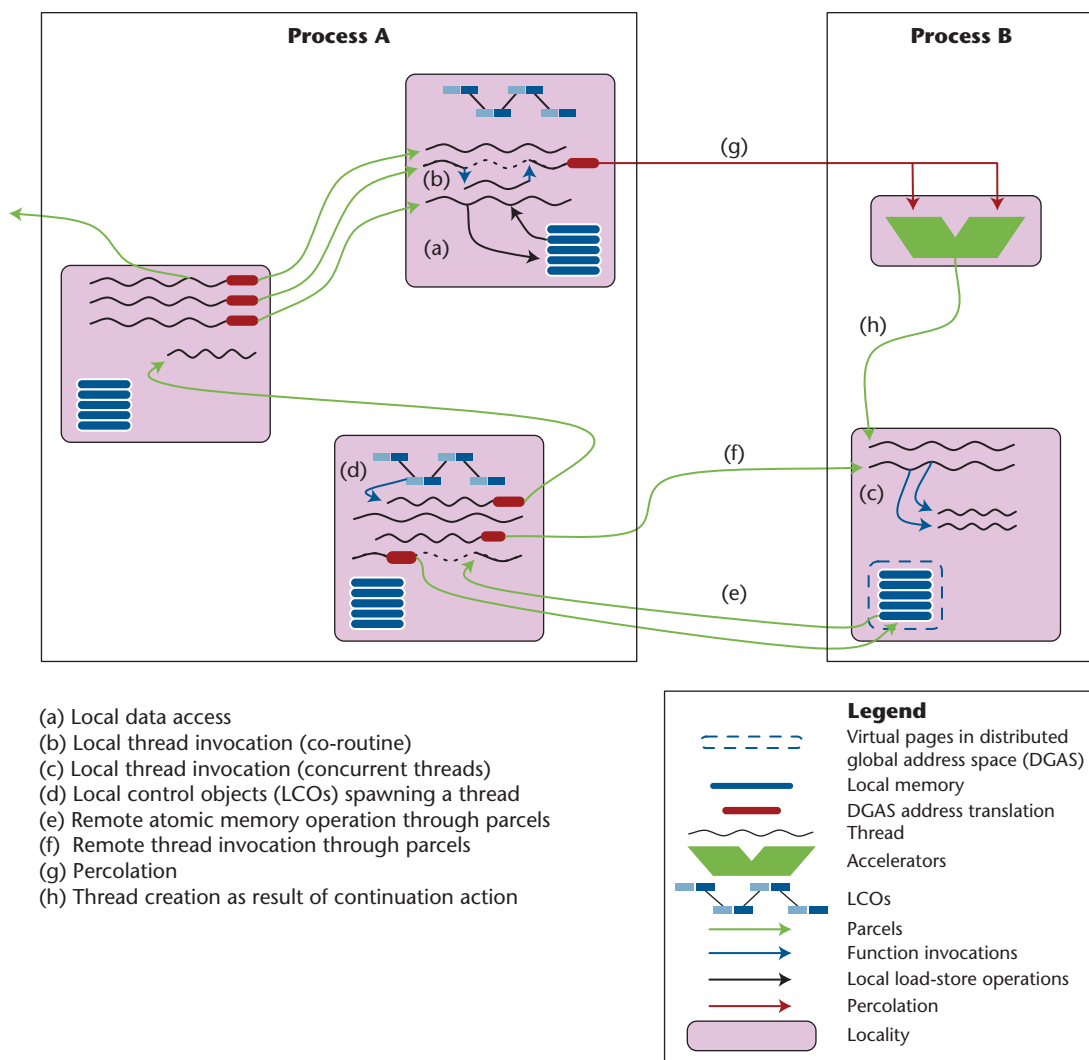


Figure 2. The ParalleX model. ParalleX could replace the conventional communicating sequential processes model that's widely employed on scalable distributed memory systems. (a) Local data access, (b) local thread invocation (co-routine), (c) local thread invocation (concurrent threads), (d) local control object spawning a thread, (e) remote atomic memory operation through parcels, (f) remote thread invocation through parcels, (g) percolation, and (h) thread creation as a result of continuation action.

principles, including such common forms as mutexes, semaphores, and barriers, but also more atypical but powerful forms such as dataflow (templates),^{9,10} producer-consumer, and futures. LCOs even include suspended threads.

This powerful super-construct is used in a unique way to manage asynchrony and allow flow control through continuations that migrate across the abstract computation and the physical distributed machine. Parcels are a form of active messages⁷ that enable message-driven computation for reasons we described earlier. Together, these powerful semantic mechanisms supported by compilation, a runtime system, and hardware architecture open up a vast array of parallelism

while mitigating the effects of latency, contention, and overhead. The result is potentially dramatic improvements in efficiency, scalability, and energy consumption.

Application and System Software Impact

Developers will create new system software incorporating this new execution model's principles to support future new architectures and their applications. The most significant change will be the emergence of user-level runtime system importance. The runtime system will likely take over many task management responsibilities from the conventional operating system (OS) and will offer far greater efficiency. The OS is distinct

from the runtime system in that it's persistent and owns the architecture hardware resources; in contrast, the runtime is ephemeral: a new one is invoked for each user application and terminates when the application is completed.

The runtime system operates within the application context and can support rapid user-thread context switching and global application synchronization. It also manipulates the hardware resources allocated to it by the OS—usually much faster than the OS itself could. Essentially, because of its much narrower domain of responsibility, a user runtime system can be far more lightweight and efficient than an OS, which must consider all the issues related to the entire system and its application workload. The runtime system will have a close association with the compiler that understands the requirements of the specific application they're both serving. The compiler will optimize the runtime system use to minimize runtime overheads for the specific application needs, even as it exploits runtime knowledge to better manage resources and scheduling than would be possible with static methods alone.

Nonetheless, the OS will continue to serve as a critical component of the future exascale system. Originally, conventional OS implementations ran on a single processor. In the future, the OS will be responsible for upwards of a billion processors. This dramatic distinction will drive the new relationships and OS function definitions. Several approaches are now under consideration. Perhaps the most radical is one in which the OS is a single system comprising a number of coexisting distributed functionalities. This is quite different from the generally accepted practice of each node having its own OS and being coordinated by some higher-level middleware.


In the alternative revolutionary strategy, every major OS distributed function operates autonomously, managing a specific class of resources. For example, the system of a billion cores will have a single memory system comprising potentially hundreds of petabytes. Instead of each node OS managing its own block of data, a global memory supervisor will manage all system memory. Similarly—and perhaps most importantly—a global address manager will control address translation throughout the entire system. Yet another distributed supervisor will manage hardware thread-execution elements that feed off runtime task thread queues. Most significantly, the OS functionality will differ from today's Unix-like systems in that every mechanism must be scalable—that is, it must operate in a constant time with

respect to system scale. Otherwise, the OS will become the system bottleneck.

But how will we program such systems? The programming model is a logical interface between the application algorithm above and the system architecture below. It responds to the application's expected semantic needs, gives access to the system resources for flexibility, and yet abstracts away the myriad details to make the programming task manageable. Future programming models must be more effective at both than existing programming methodologies. It's quite possible that domain-specific programming models will emerge that are optimized to serve particular problem classes. But we'll need more general models, perhaps of a "lower" nature, from which we can develop such specialized programming interfaces.

There are two key attributes here that aren't properties of today's APIs. First, the semantics of local and remote processing must be the same—that is, symmetric in both local synchronous and global asynchronous domains of operation. Just as Newton made humanity realize that the laws of physics were the same as they applied to Cambridge or to Jupiter, so too should the rules of programming (the model's semantics) be the same whether they apply to the local thread or to some far-off system node. Second, threads should be event driven; the criteria governing when to invoke a thread should be definable.

Combining these two attributes automates message management for message-driven computation and eliminates over-constraining global barriers. That is, it lets continuations (global control state) migrate throughout the distributed system application state, without requiring users to explicitly direct them, thus allowing a true scalable parallel computer and programming model.

Many of the new DARPA UHPC efforts are working to address the issues discussed in this article, including the Sandia X-caliber project, which we're actively investigating. Ultimately, creating a new model of computation will require wide-spread community adoption. If successful, a model addressing the challenges discussed here will usher in a new era of green computing. 

Acknowledgments

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for

the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References


1. T. Sterling and C. Dekate, "Enabling Exascale through ParalleX Paradigm," *Parallel Computational Fluid Dynamics*, NASA Ames Research Center, 2010, pp. 3–18.
2. H. Kaiser, M. Brodowicz, and T. Sterling, "ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications," *Int'l Conf. Parallel Processing Workshops*, IEEE Press, 2009, pp. 394–401.
3. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: Past, Present and Future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, 2003, pp. 803–820.
4. W.A. Wulf and S.A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *Computer Architecture News*, vol. 23, no. 1, 1995, pp. 20–24.
5. R.C. Murphy, "On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance," *IEEE Int'l Symp. Workload Characterization (IISWC07)*, IEEE Press, 2007, pp. 35–43.
6. D. Bonachea, *GASNet Specification*, tech. report UCB/CSD-02-12072002, Electrical Eng. and Computer Science Dept., Univ. California, Berkeley, Oct. 2002; <http://gasnet.cs.berkeley.edu>.
7. T. von Eicken et al., "Active Messages: A Mechanism for Integrated Communication and Computation," *Computer Architecture News*, vol. 20, no. 2, 1992, pp. 256–266.
8. C. Hewitt and H. Baker, "Laws for Communicating Parallel Processes," *IFIP Congress Proc.*, Massachusetts Inst. Tech. Artificial Intelligence Lab, 1977; <http://hdl.handle.net/1721.1/41962>.
9. K. Arvind and R.S. Nikhil, "Executing a Program on the MIT Tagged-Token Dataflow Architecture," *IEEE Trans. Computing*, vol. 39, no. 3, 1990, pp. 300–318.
10. J.B. Dennis and D.P. Misunas, "A Preliminary Architecture for a Basic Data-Flow Processor," *Computer Architecture News*, vol. 3, no. 4, 1974, pp. 126–132.
11. ITRS Committee, *International Technology Roadmap for Semiconductors 2008*, Int'l Tech. Roadmap for Semiconductors, 2008.
12. P.M. Kogge, ed., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, tech. report TR-2008-13, Computer Science and Eng. Dept., Univ. Notre Dame, 28 Sept. 2008.
13. R.C. Murphy and P.M. Kogge, "On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications," *IEEE Trans. Computers*, vol. 56, no. 7, 2007, pp. 937–945.

14. C.A. Hoare, "Communicating Sequential Processes," *Comm. ACM*, vol. 21, no. 8, 1978, pp. 666–677.
15. W. Gropp et al., "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, 1996, pp. 789–828.

Richard Murphy is a computer architect in the scalable systems group at Sandia National Laboratories, where he leads research projects in memory systems and advanced computer architectures, including the Sandia X-caliber exascale architecture effort. His research interests include computer architecture, with a focus on memory systems and processing-in-memory, very large scale integration, and massively parallel architectures, programming languages, and runtime systems. Murphy has a PhD in computer engineering from the University of Notre Dame. He is a senior member of IEEE. Contact him at rcmurph@sandia.gov.

Thomas Sterling is a professor of computer science at Louisiana State University, a faculty associate at the California Institute of Technology, a distinguished visiting scientist at Oak Ridge National Laboratory, and a fellow of Sandia's Computer Science Research Institute. He is probably best known as the "father" of Beowulf clusters and for his research on petaflops computing architecture. He currently leads the MIND memory accelerator architecture project for scalable data-intensive computing and is an investigator on the US Department of Energy's Fast-OS Project to develop a new generation of configurable lightweight parallel runtime software systems. Thomas has a PhD in electrical engineering from the Massachusetts Institute of Technology, where he was a Hertz Fellow. Contact him at tron@cct.lsu.edu.

Chirag Dekate is a doctoral student and HPC researcher at Louisiana State University's Center for Computation and Technology, where he's focusing on thread management policies and dynamic graph applications on innovative HPC systems. His research interests include studying and developing exascale HPC software systems, specifically runtime management systems for new innovative HPC applications. Chirag has an MS in computer science from Louisiana State University. Contact him at cdekate@cct.lsu.edu.

 Selected articles and columns from IEEE Computer Society publications are also available for free at <http://ComputingNow.computer.org>.