THE Next Wave

The National Security Agency's review of emerging technologies

Vol. 20 | No. 2 | 2013

Globe at a Glance | Pointers | Spinouts

THE
Next Wave
The National Security Agency's review of emerging technologies

Editor's column

Managing Editor

"End of the road for Roadrunner" Los Alamos National Laboratory, March 29, 2013. [a]

Five years after becoming the fastest supercomputer in the world, Roadrunner was decommissioned by the Los Alamos National Lab on March 31, 2013. It was the first supercomputer to reach the petaflop barrier—one million billion calculations per second. In addition, Roadrunner's unique design combined two different kinds of processors, making it the first "hybrid" supercomputer. And it still held the number 22 spot on the TOP500 list when it was turned off.

Essentially, Roadrunner became too power inefficient for Los Alamos to keep running. As of November 2012, Roadrunner required 2,345 kilowatts to hit 1.042 petaflops or 444 megaflops per watt. In contrast, Oak Ridge National Laboratory's Titan, which was number one on the November 2012 TOP500 list, was 18 times faster yet five times more efficient.

In addition, data-intensive applications for supercomputers are becoming increasingly important. According to the developers of the Graph500 benchmarks, these data-intensive applications are "ill-suited for platforms designed for 3D physics simulations," the very purpose for which Roadrunner was designed. New supercomputer architectures and software systems must be designed to support such applications.

These questions of power efficiency and changing computational models are at the core of moving supercomputers toward exascale computing, which industry experts estimate will occur sometime between 2020 and 2030. They are also the questions that are addressed in this issue of *The Next Wave (TNW)*.

Look for articles on emerging technologies in supercomputing centers and the development of new supercomputer architectures, as well as a brief introduction to quantum computing. While this column takes the reader to the recent past of supercomputing, the remainder of the issue will propel you "beyond digital" to the future of advanced computing systems.

Managing Editor, *The Next Wave*

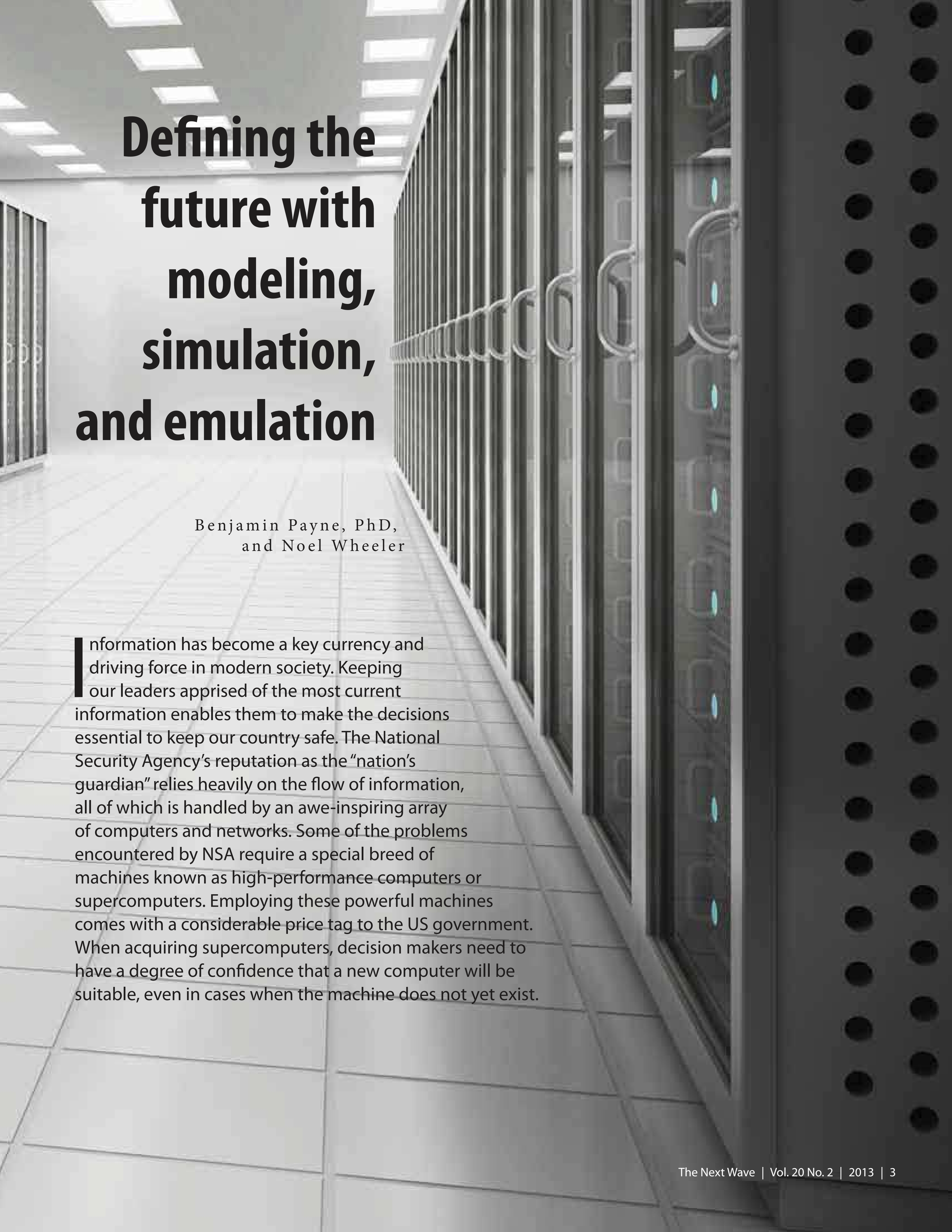
a. Press release, Los Alamos National Laboratory. 29 March 2013. Available at: http://www.lanl.gov/newsroom/news-releases/2013/March/03.29-end-of-roadrunner.php.

Contents

The Next Wave is published to disseminate technical advancements and research activities in telecommunications and information technologies. Mentions of company names or commercial products do not imply endorsement by the US Government.

This publication is available online at http://www.nsa.gov/research/tnw/tnw202/article1.shtml. For more information, please contact us at TNW@tycho.ncsc.mil.

Vol. 20 | No. 2 | 2013



Defining the future with modeling, simulation, and emulation

Benjamin Payne, PhD,
and Noel Wheeler

Information has become a key currency and driving force in modern society. Keeping our leaders apprised of the most current information enables them to make the decisions essential to keep our country safe. The National Security Agency's reputation as the "nation's guardian" relies heavily on the flow of information, all of which is handled by an awe-inspiring array of computers and networks. Some of the problems encountered by NSA require a special breed of machines known as high-performance computers or supercomputers. Employing these powerful machines comes with a considerable price tag to the US government. When acquiring supercomputers, decision makers need to have a degree of confidence that a new computer will be suitable, even in cases when the machine does not yet exist.

What is a supercomputer?

Although supercomputers are unique, custom-built machines, they fundamentally share the design of the computers you use at home—a processor (i.e., a central processing unit or CPU), small and fast memory (i.e., random-access memory or RAM), storage (i.e., hard disk drive/CD/DVD), and a network to communicate with other computers. A typical high-performance computing (HPC) system could be considered a personal computer on a much grander scale, with tens of thousands of processors, terabytes (i.e., trillions of bytes) of memory, and petabytes (i.e., quadrillions of bytes) of storage (see figure 1). High-performance computers can readily fill a large room, if not a whole building, have customized cooling infrastructure, use enough electricity to power a small town, and take an act of Congress to purchase. Such an investment is not made without a great deal of study and thought.

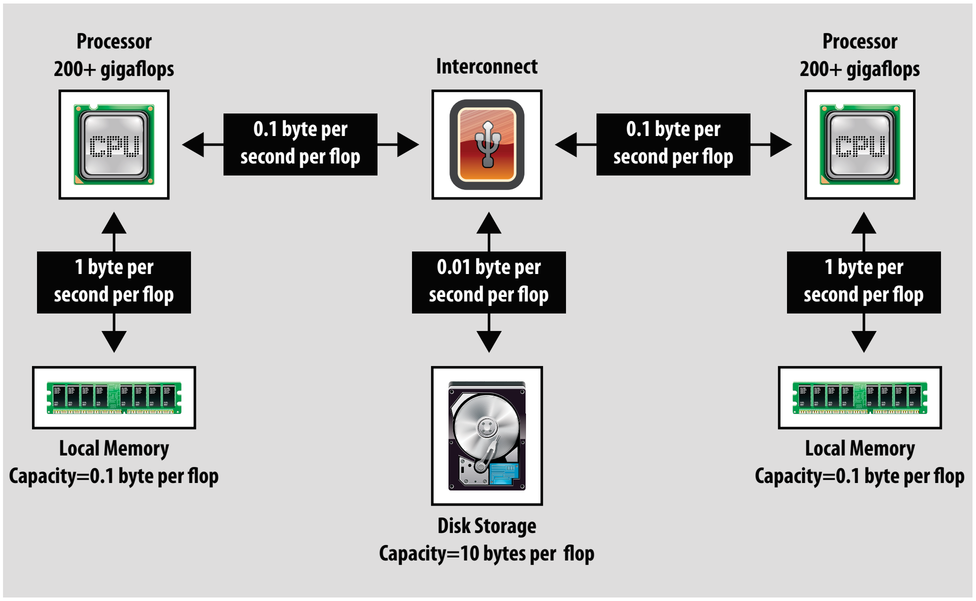

Processor
200+ gigaflops
Interconnect
Processor
200+ gigaflops
CPU
0.1 byte per second per flop
0.1 byte per second per flop
CPU
1 byte per second per flop
0.01 byte per second per flop
1 byte per second per flop
Local Memory
Capacity=0.1 byte per flop
Disk Storage
Capacity=10 bytes per flop
Local Memory
Capacity=0.1 byte per flop

FIGURE 1. A high-performance computer is like a personal computer on a much grander scale—it has tens of thousands of processors, terabytes of memory, and petabytes of storage.

Simulating a supercomputer

Although HPC technology is not unique to NSA, the specialized problems faced by the Agency can necessitate unique customizations. Because NSA's applications and software are often classified, they cannot be shared with the architects and engineers developing supercomputers. At the same time, an investment of this magnitude requires confidence that a proposed system will offer the performance sought.

Currently, benchmarks, simplified unclassified software that exercises important attributes of a computer system, are developed and used to evaluate the performance of potential computing system hardware. However, these benchmarks may not paint the complete picture. To better understand this problem, there is substantial value to the construction of a model. Architects, engineers, and scientists have a long history of building models to study complex objects, such as buildings, bridges, and aircrafts.

A new team—the Modeling, Simulation, and Emulation (MSE) team—within the Laboratory of Physical Science's Advanced Computing Systems Research Program [1] has been assembled to address this gap between classified software, which cannot be distributed to vendors, and the vendors' hardware systems, which have not been purchased by NSA. As an additional twist, the proposed hardware may be built from prototype components such as the hybrid memory cube (HMC; see figure 2), a three dimensional stacked memory device designed by a consortium of industry leaders and researchers [2]. The core objectives of the MSE team include exploration of system architectures, analysis of emerging technologies, and analysis of optimization techniques.

Owners of HPC systems desire a computer that is infinitely fast, has infinite memory, takes up no space, and requires no energy. None of these attributes are truly realizable, and when considering a practical HPC system, trade-offs must be considered. When analyzing a prospective HPC system, four primary metrics are customarily considered: financial cost, system resilience, time-to-solution, and energy efficiency. These metrics are interdependent. For example, increasing the speed of an HPC system will increase the amount of power it consumes and ultimately increase the cost necessary to operate it. In order to measure these metrics, one could build the system and test it. However, this would be extremely expensive and difficult to optimize. A model simulating the computer can be developed in far less time, and design parameters can be adjusted in software to achieve the desired balance of power, performance, reliability, and cost.

Any simulation or model of a computer should address the metrics listed above. If any are not addressed, then the model could yield incomplete results because optimizing for fewer than all relevant variables potentially leads to non-global extrema. Many scalar benchmarks, for example the TOP500 and the Graph500, focus exclusively on one characteristic, like time-to-solution, to the neglect of the other parameters of interest. The MSE team is collaborating to evangelize a more balanced approach to multiple facets of HPC system characterization, assuring an optimal solution to the Agency's needs.

The use of benchmarking software allows for a more comprehensive evaluation of a proposed computer architecture's performance. This enables HPC system architects to better target their designs to serve NSA's needs. Simply stated, NSA has to work within budgetary and power (i.e., electricity) constraints, and it is vital to maximize the return on investment of money and time.

While this description is somewhat generic to all HPC system purchasers, NSA is willing to build special purpose hardware devices and to employ specially developed programming languages if a cost-benefit analysis demonstrates noteworthy benefits. Unlike developers in the scientific community whose expertise usually does not span science, computer programming, and computer architecture, developers at NSA access and understand the full software and hardware stack—algorithm, source code, processors, memory, network topology, and system architecture. Compute efficiency is often lost in the process of separating these abstraction layers; as a result, NSA makes an effort to comprehend the full solution.

This approach to mission work is reflected in the work of the MSE team. A simulation or model should

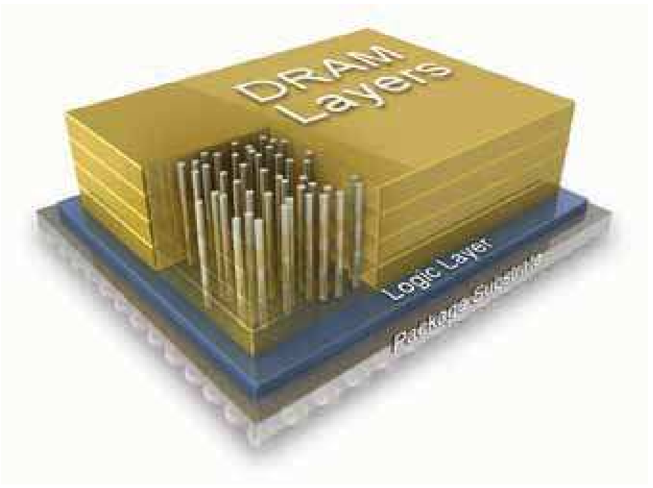


FIGURE 2. NSA collaborated with the University of Maryland and Micron to develop a simulation tool for Micron's Hybrid Memory Cube that is helping to advance supercomputing applications. Micron now is sampling the three-dimensional package that combines logic and memory functions onto a single chip.

take a holistic approach, targeting the network, CPU, memory hierarchy, and accelerators (e.g., a graphics processing unit or field-programmable gate array). Multiple levels of detail for a simulation are required to accomplish this. A simulation may be compute-cycle or functionally accurate; it may range from an abstract model to a hardware simulation including device physics.

Simulation technology

To accomplish the objective of enabling HPC system simulation within NSA, the MSE group carried out a survey of existing simulators from academia, industry, and national labs. Although many simulators exist for HPC systems, few attempt to model a complete architecture. There have been previous efforts like University of California, Los Angeles's POEMS and Hewlett Packard's COTSon, but these projects are no longer actively supported. Two simulation frameworks, the Structural Simulation Toolkit (SST; see figure 3) [3] from Sandia National Laboratories and Manifold from Georgia Institute of Technology represent today's most promising candidates. Additionally, NSA researchers have been crafting simulation tools which are also being considered for application in the HPC problem space.

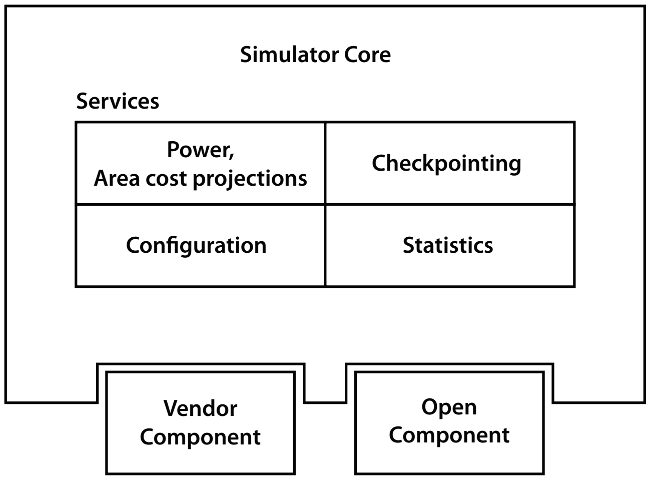


FIGURE 3. Sandia National Laboratories' Structural Simulation Toolkit is one of today's most promising high-performance computing system simulators.

Both SST and Manifold use component simulators to construct a larger-scale system. For example, SST can use the gem5 [4] CPU simulator along with the University of Maryland's DRAMSim2 to capture performance characterization of processor to memory latency and bandwidth. Since simulating a full-scale HPC system would require an even larger supercomputer to run in a reasonable time, SST breaks the simulation into two components: SST/micro, a node-level simulation (e.g., CPU, memory), and SST/macro, which handles network communication between nodes. With the emerging HMC memory technology, the MSE team is making plans to employ the SST family of tools to extensively model an HPC system and gain perspective on its potential capabilities. This will place NSA's HPC programs on the leading edge in understanding the application potential for this new technology.

At this time, SST/micro is capable of simulating the execution of programs in a single processor core and of monitoring the application's use of the simulated CPU and memory. By 2014, the development team at Sandia plans on parallelizing the simulation, enabling multiple processor cores to be simultaneously simulated. This would allow parallel applications (i.e., software designed to simultaneously run on multiple processor cores) to be run in a realistic compute node configuration (i.e., multiple cores concurrently accessing the same memory hierarchy) while potentially reducing the time needed to complete a simulation.

SST/macro, combined with NSA's benchmarking software, has already been used to demonstrate how different network topologies, used to connect an HPC system's processing cores, can affect the time-to-solution metric. SST/macro allowed researchers to specify data-routing algorithms used in the network configuration and to study how a modified network topology serves to optimize the performance of a system. The clear benefit of this research is in the ability to enable application and network codesign to create an optimal and cost-effective architecture.

The SST and its counterpart, Manifold, are being actively developed and are useful for research, but they are not yet ready for use as decision-making tools by NSA. The MSE team is actively collaborating with Sandia and the Georgia Institute of Technology, providing feedback, guidance, and assistance to the simulation framework developers. Multiple other national labs, academic researchers, and vendors are also participating in the effort driven by the MSE team. Other potential applications for simulation techniques could be codesign of software before the actual hardware is available, software performance analysis/optimization, and debugging of software.

About the authors

Noel Wheeler is the lead of the Modeling, Simulation, and Emulation (MSE) team in the Advanced Computing Systems group at NSA's Laboratory of Physical Sciences. **Ben Payne,** PhD, is a physicist working as a postdoctoral researcher for the MSE team.

References

[1] Klomparens W. "50 Years of research at the Laboratory for Physical Sciences." *The Next Wave*. 2007;16(1):4–5.

[2] Hybrid Memory Cube Consortium [homepage; updated 2013]. Available at: http://www.hybridmemorycube.org.

[3] Sandia Corporation. SST: The Structural Simulation Toolkit [homepage; accessed 4 Apr 2013]. Available at: http://sst.sandia.gov.

[4] Binkert N, Beckman G, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, et. al. "The gem5 simulator." *ACM SIGARCH Computer Architecture News*. 2011;39(2):1–7. doi: 10.1145/2024716.2024718.

Predicting the performance of extreme-scale supercomputer networks |

Scott Pakin, Xin Yuan, and Michael Lang

A modern supercomputer is the Internet in a microcosm, with tens of thousands of nodes—computers not much different from the one you may be using to read this article—all hooked together via a high-speed network. However, while computers on the Internet operate largely independently of each other, supercomputers regularly harness the power of thousands to many tens of thousands of nodes at once to run a single application significantly faster than any lone computer could. Coordinating the efforts of so many nodes requires massive amounts of communication, making the design of the interconnection network critical to the performance of the supercomputer as a whole.

In this article we present technology we are developing to predict the impact of various network-design alternatives on the overall performance of supercomputing applications *before* the supercomputer is even built. This is important because a large supercomputer can easily cost tens to hundreds of millions of dollars (and in the case of Japan's K supercomputer, over a billion dollars). Being able to evaluate network technologies during their design phase helps ensure that the supercomputer will provide as much performance as possible to applications.

Network topologies

Supercomputers gain their performance edge from *parallelism*, the ability to perform many pieces of work at the same time. Taking advantage of a supercomputer consequently requires an application to divide up the work it has to perform into small chunks that can be spread over a supercomputer's nodes. In practice, some of these chunks of work depend on other chunks.

Consider, for example, an arithmetic expression such as (5+5)×(6+8). The two sums can be computed concurrently, but the product cannot be computed until after both sums have been computed. This necessitates communication, commonly taking the form of inter-node *messages* sent over a communication network. The node computing one sum has to tell the other node when it has finished and what sum it computed so the latter node can perform the multiplication. (Alternatively, both nodes can communicate their sum to a third node, which can multiply the two sums.) Network speed is critical to application performance. If the network is too slow relative to the time spent in computation, which is likely the case for our simple arithmetic example, there will be no performance gain to be had from parallelism, and the supercomputer's performance capabilities will be wasted.

While the Internet is composed of a motley connection of subnetworks haphazardly linked together, supercomputer networks gain some of their speed advantage by exploiting homogeneous hardware arranged into regular patterns. This avoids some nodes lying in the boondocks of the network and slowing down the entire application whenever distant nodes need to communicate with them. Figure 1 illustrates three topologies out of endless possibilities. Contrast the irregular structure of figure 1(a), which illustrates the graph nature of the Internet's topology, with the symmetry in each of figures 1(b) and (c), which illustrate two common supercomputer topologies: a *fat tree* and a *three-dimensional torus* (i.e., *3-D torus*).

Nodes in the figure are shown as blue spheres. (Each of a modern supercomputer's nodes typically contains 10–100 processor cores, making a node a powerful computer in its own right.) Network links are portrayed in the figure as lavender tubes and switches are portrayed as salmon-colored boxes. A *switch* receives data on one link and, based on where the data is to be delivered, sends it out on another link.

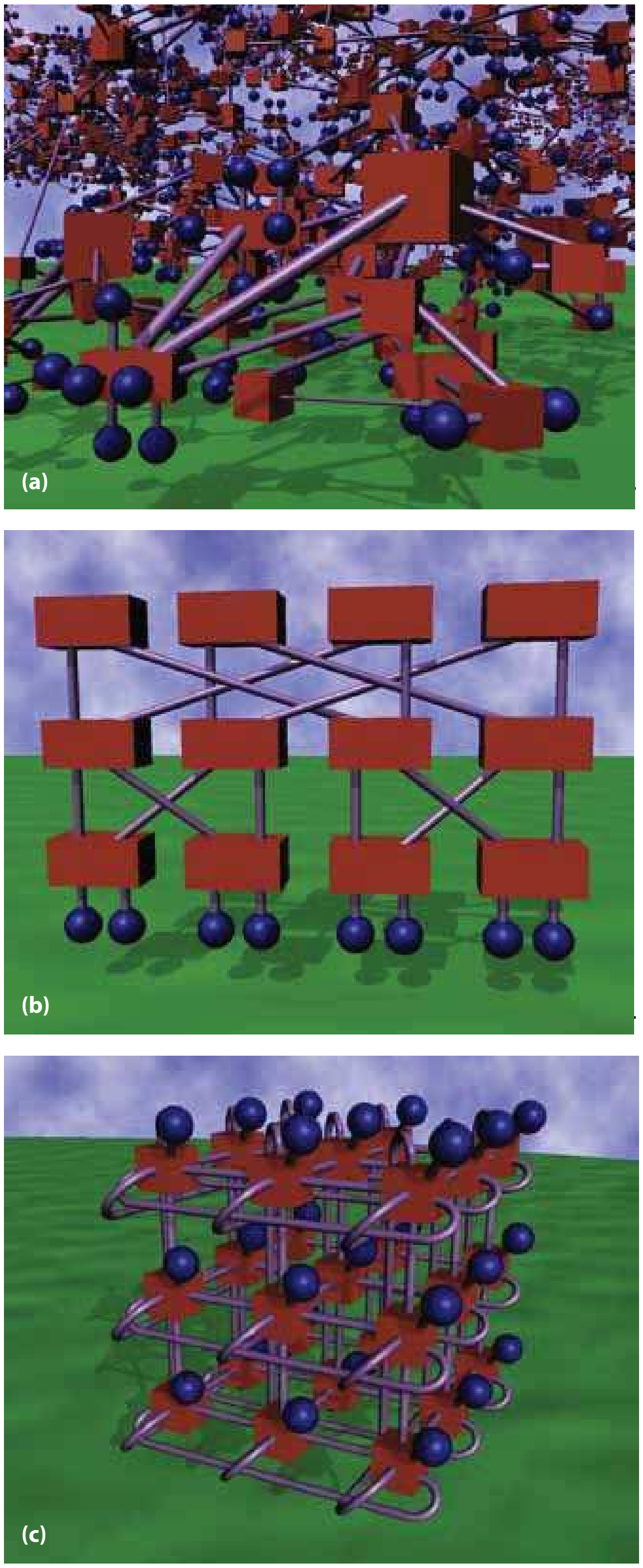


FIGURE 1. Three examples of network topologies. Figure (a) shows an example of a small-world network topology. Figure (b), which depicts a fat tree, and figure (c), which depicts a 3-D torus, are two common supercomputer network topologies.

For example, if the leftmost node in the fat tree depicted by figure 1(b) needed to communicate with the rightmost node, it could send data to the switch above it, which could forward the data to the switch above it and then to the switch above it. The topmost switch could then forward the data diagonally down and to the right, then diagonally down and to the right again, and finally down to the destination node.

An alternative route would be to start with a couple of diagonally upward-and-rightward hops followed by vertically downward hops. (As an exercise, see if you can find a third path from the leftmost node to the rightmost node. Are there any more routes?) We use the term *routing algorithm* to describe the process by which switches select one route among the alternatives.

The importance of a topology such as a fat tree is that there *are* multiple ways to get from any node to any other node. Hence, if one route is congested, data can proceed along a different route. Consider an analogy to cars and roads, with cars representing data, roads representing links, and intersections representing switches. The more roads connecting a residential neighborhood to a commercial district, the less traffic is likely to appear on any given road. At the extreme, one could connect every node to every other node in a supercomputer to eliminate all congestion. In practice, this is not done for the same reason that there are not private roads connecting every house to every other house in a town—cost. Switches and links are expensive; hence, a network designer must simultaneously minimize the number of switches and links while maximizing the number of alternative routes between pairs of nodes. A 10,000 node supercomputer with all-to-all connectivity would require one hundred million links. At even a dollar apiece (an unrealistically small amount), this would dominate the cost of the supercomputer.

Figure 1(c) illustrates a 3-D torus, another common supercomputer network topology and one that makes different trade-offs from a fat tree with respect to switch and link count and alternative paths. In this topology, nodes and switches are arranged in a cube (or rather, rectangular cuboid) formation, and wraparound links enable data sent out one side of the network to re-enter on the other side. For example, if the node in the lower left of figure 1(c) needed to communicate with the node in the upper right, the long way would be to travel up-up-up-right-right-right-back-back-back. However, the wraparound links enable the data to travel down to the topmost position, then left to the rightmost position, and finally forward to the backmost position, taking three hops instead of nine.

Putting cost arguments aside for the moment and assuming the same node count in both networks, could a fat tree be expected to outperform a 3-D torus, or would the 3-D torus likely be the faster network? In the next section, we discuss how to answer this question.

Simulating networks

As creating a new network is expensive and time-consuming, we want to be able to gauge how well a given network might perform in advance of its construction. This is commonly done via *network simulation*—mimicking hardware's behavior with slower but vastly more malleable software. We again turn to a car-and-road analogy. Consider the situation of bumper-to-bumper traffic on two single-lane roads that merge into one single-lane road, as shown in figure 2. It would be prohibitively expensive to construct the roads and hire drivers to drive in the specified pattern just to determine the speed at which traffic can move. Instead, one could write a computer program that moves virtual cars on virtual roads and measures how much time elapses in this virtual world. In networking terms, this approach is called *flit-level simulation* because it tracks every flit (a unit of data, typically a byte) as it moves from switch to switch throughout the network.

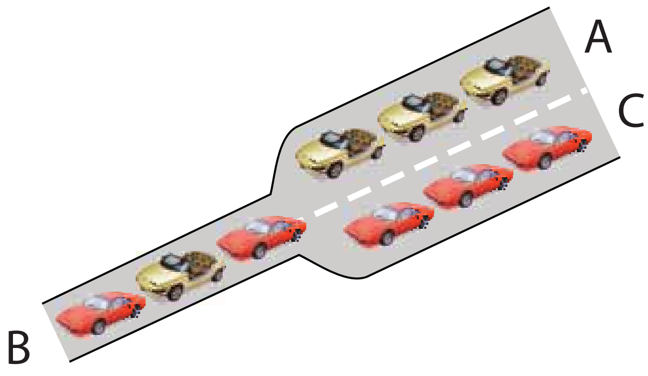


FIGURE 2. To determine the speed at which vehicles can move in bumper-to-bumper traffic on two single-lane roads that merge into one single-lane road, one can simulate this "network" using a computer program. There are different approaches to network simulation, which vary in speed and degree of realism.

At each point in (virtual) time, the simulator considers the current location of each flit in the network; the routing algorithm, which is used to decide where each flit should go next; the internal switch architecture, which controls how link contention is resolved (for example, a simple alternating of flits as illustrated by the cars in figure 2); and all the myriad other characteristics that determine performance. With regard to figure 2, a simulator would need to take into consideration not only the speed limits and layout of the road system but also the decision-making process of each driver on the road to know where the driver wants to go and how he will negotiate with other drivers as to who gets to go first when lanes merge.

There are two main problems with flit-level simulation, one inherent and one artificial. The inherent problem is that simulating a large network at such a fine level of detail is necessarily slow—vastly slower than real network hardware could run. Thousandfold slowdowns are not uncommon. In other words, the simulator might need to run for an hour to report how a network might behave over the course of a single second of execution. To put that slowdown in perspective, consider that many of the scientific applications commonly run on supercomputers at Los Alamos National Laboratory take hours to days to run; a few even require months to over a year to complete. Dilating such times by a factor of a thousand clearly limits the practicality of simulating such applications. Consequently, flit-level simulations must by necessity whittle down their inputs to a more manageable size, simulating only small networks and for only brief periods of time, which limits realism.

The artificial problem is that for simplicity of operation, simulators are typically fed synthetic communication patterns rather than communication patterns derived from actual supercomputing applications. For example, two common test patterns are uniformly random traffic in which each node sends data to some number of other nodes selected at random, and hot-spot traffic in which all nodes send data to a small subset of the nodes selected at random. Second, almost all simulation studies presented in the supercomputer-network literature assume that communication begins at fixed points in time, typically exclusively at the start of the simulation. Third, computation time is almost universally ignored, even though this can greatly affect the severity and impact of link contention.

Returning to our car-and-road metaphor, typical simulator usage would be analogous to gauging the quality of a layout of a city street under assumptions like the following:

1. People drive randomly from one place to another as opposed to, say, a bias to drive to the kids' school at the beginning of the day, then to the office, then to the kids' school again, and finally back home.
2. Everyone leaves home at exactly 9:00 a.m., drives directly to his destination, and leaves the car there. A less-common variation on this assumption is that Alice picks up Bob at exactly 8:15 a.m., Carol at exactly 8:30 a.m., and Dave at exactly 8:45 a.m. for their carpool to work—all regardless of how heavy or light the traffic happened to be at the time or whether a new highway had just been built to speed up their commute.
3. No one stops to work, shop, or relax; all anyone in the city does is drive.

It would be hard to lend much credence to any result of such a study, yet this is very much how supercomputer networks are analyzed today. Again, this is an artificial problem. There is no fundamental reason that such assumptions must be made; they are merely a convenience to simplify the simulation effort. In the next section, we describe how we are improving the state of the art in network-simulation technology, both in terms of simulation speed and simulation realism.

A new approach to network simulation

Our goal is to address all of the shortcomings discussed above; in particular, our aim is to simulate all of the following:

- Full-sized applications, not synthetic communication patterns;
- Hours of application-execution time, not seconds;
- Tens of thousands of nodes, not hundreds to low thousands;
- Communication interleaved with computation, not treated as independent; and
- Communication beginning when prior communication or computation ends, not at fixed points in time.

The two mechanisms that underlie our approach are *flow-based simulation* and *logical clocks.* We now describe each of these in turn.

Flow-based simulation

The reason that flit-based simulation is so slow is that supercomputer networks contain a massive number of components, and each of these must be simulated individually. Logically, if one were to simulate large groups of components as single entities, this would greatly reduce the amount of work, and therefore time, required to run the simulation. We therefore choose to consider a complete, end-to-end communication operation as a single unit of simulation rather than the numerous flits that get transmitted as part of that operation.

Before we explain the details precisely, we present the intuition behind our approach in terms of our running car-and-road analogy. Assuming a 40 mph speed limit and that the distance from the front of one car to the front of the next is 29 feet, the math works out to two cars per second passing any particular point on the road. Hence, if we knew that 100 cars wanted to go from point *A* to point *B* and that there was no other traffic on the road, the first car in that sequence would arrive after some given length of time (i.e., however long it takes to drive from point *A* to point *B* on an empty road, say three minutes), and the last car would arrive 100 ÷ 2 = 50 seconds later.

We now consider the variation indicated by figure 2: 100 yellow cars want to go from point *A* to point *B* at the same time that 100 red cars want to go from point *C* to point *B.* What impact does the shared segment of road have on the time it takes each of those two flows of cars to reach their destination? As before, two cars per second are reaching point *B,* but because the two flows are interleaved, only one yellow car per second and one red car per second can reach that location. The first car in each flow is not delayed, so it still takes our assumed three minutes to arrive at point *B,* but the last car in each flow arrives not 50 seconds later but 100 ÷ 1 = 100 seconds later.

The point of this exercise is to demonstrate that, unlike with flit-level simulation, we do not have to consider each individual car's behavior. Instead, we can analyze an entire sequence of cars at once, regardless of whether there are a hundred cars in each flow or a million. Furthermore, we do not need to consider how the drivers negotiate the merge. All that matters is that there is an even 50–50 split between red and yellow cars on the merged segment of road, not that it went red–yellow–red–yellow versus red–red–yellow–yellow.

Our approach to network simulation works in very much the same way as the preceding analysis of traffic speeds. As in the above instance, instead of working with communication times directly, we work with communication *rates,* which we can easily relate back to time by noting that time = latency + (data size ÷ communication rate), where *latency* is the time it would take a single flit to move from the source node to the destination node in the absence of any other traffic. For example, suppose that the latency between node *A* and node *B* is 0.6 seconds and that all of the links between node *A* and node *B* are capable of transmitting 5 gigabytes per second. If node *A* were to transmit 1 gigabyte of data to node *B,* this communication would take a total of 0.6 + (1.0 ÷ 5.0) = 0.8 seconds.

While latencies are essentially constant and data sizes can be extracted from an application (as we will discuss further when we discuss logical clocks), communication rates vary dynamically based on the amount of *link contention,* the number of communications sharing a network link at any given time. Consider the network topology shown in figure 3 (i.e., a 2-D mesh).

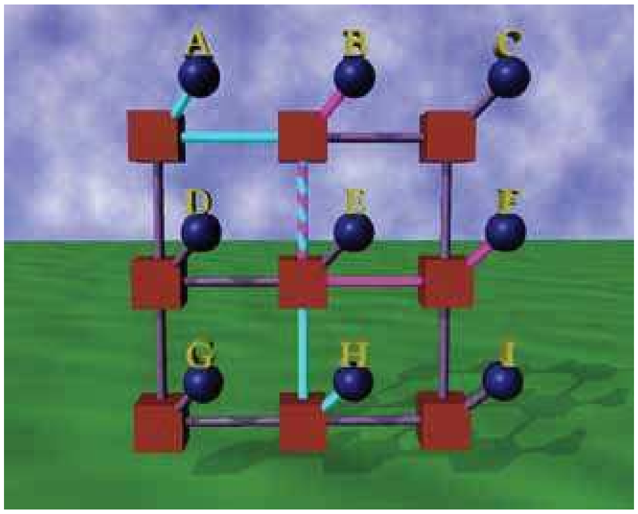


FIGURE 3. An illustration of link contention on a 2-D mesh network topology.

If node *A* sends data to node *H* via the route *A*–*B*–*E*–*H* (cyan links) at the same time as node *B* sends data to node *F* via the route *B*–*E*–*F* (magenta links), the *B*–*E* link will be shared by the two routes. Supposing the link is capable of transmitting at a rate of 5 gigabytes per second, 2.5 gigabytes per second will be allocated to each of the two communications. Because data cannot enter a link faster than it can exit, this slow link then exerts back-pressure all the way to the source nodes, slowing down the *entire* communication to 2.5 gigabytes per second. Using the previously mentioned sample numbers from each of the two communications will now take 0.6 + (1.0 ÷ 2.5) = 1.0 seconds instead of the contention-free 0.8 seconds computed earlier—slower but notably not twice as slow, even though the link speed effectively halved.

Logical clocks

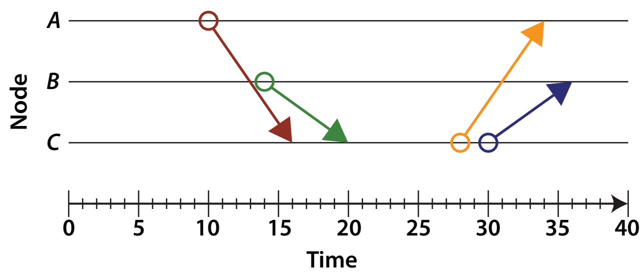
We criticized prior simulation efforts for relying on synthetic communication patterns instead of actual communication patterns derived from supercomputing applications. Our question is therefore how we can acquire an application's communication pattern so that it can be analyzed by a simulator. The enumeration of all communication that an application performs during its execution—which node sent how many bytes to whom when—is called a *communication trace*. Fortunately, intercepting and logging an application's communication operations is fairly straightforward, and there exist numerous tools for collecting communication traces.

The issue is not with *collecting* the trace but with *interpreting* it. Figure 4 helps clarify the problem. Figure 4(a) presents a trace of a communication pattern in which node *A* sent a message to node *C*, then node *B* sent a message to node *C*, then, after a brief interlude, node *C* sent a message to node *A*, and finally, node *C* sent a message to node *B*. A graphical view of this trace is shown in figure 4(b). Send and receive times are reported from the perspective of each node's clock. For example, the first line of the table in figure 4(a) indicates that node *A* reported that it sent a message to node *C* at time 10 and that node *C* reported that it received node *A*'s message at time 16.

The first problem with this type of communication trace is that supercomputer nodes seldom include per-node clocks that are globally synchronized to within half a message latency (i.e., the tolerance needed to

Source Node	Destination Node	Sent Time	Received Time
A	C	10	16
B	C	14	20
C	A	28	34
C	B	30	36

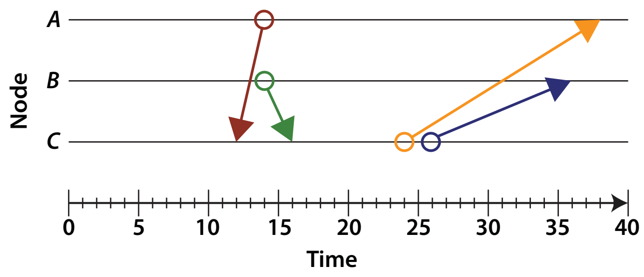
(a) Communication trace



(b) Timeline view of the trace

Source Node	Destination Node	Sent Time	Received Time
A	C	14	12
B	C	14	16
C	A	24	38
C	B	26	36

(c) Communication trace with poorly synchronized clocks



(d) Timeline view of the trace with poorly synchronized clocks

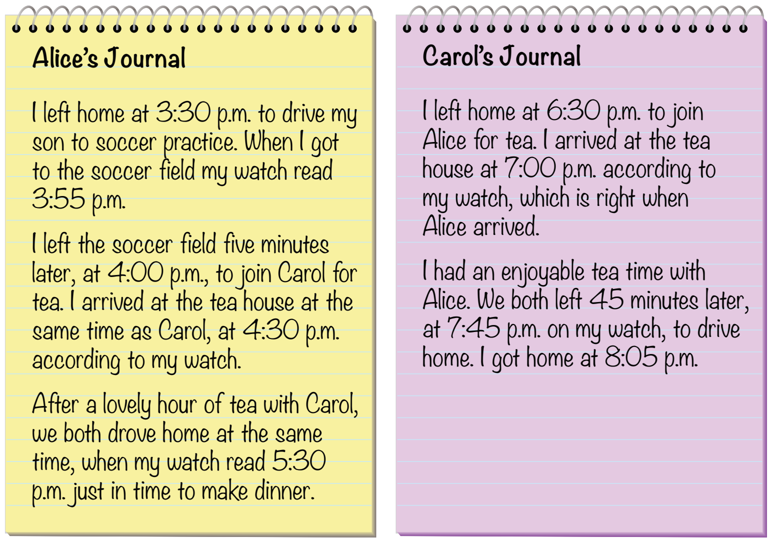
FIGURE 4. Example of a communication pattern. Figure (a) and (b) illustrate a communication trace of nodes with perfectly synchronized clocks (an unrealistic condition). Figure (c) and (d) illustrate a communication trace of nodes with poorly synchronized clocks.

avoid erroneous readings, as discussed below). These would represent a costly but rarely useful expense. Furthermore, access to a single, centralized clock would be devastating to performance—imagine figure 2 with tens of thousands of lanes merging into one. Hence, some node clocks may run slightly ahead or behind others, and even worse, some node clocks may run slightly faster or slower than others. This is called *clock drift.* Although clock-synchronization algorithms exist, software implementations are unable to synchronize clocks to a granularity fine enough to measure network-communication time.

Figure 4(c) represents the same trace as figure 4(a) but as measured with node *A*'s clock running four time units late and node *C*'s clock running four time units early. As the graphical depiction of this trace in figure 4(d) clarifies, the faulty clocks make the first message appear to have been received before it was sent, a physical impossibility. Furthermore, instead of each message taking a constant six time units to get from source to destination as indicated by the "perfect" trace in figure 4(a), the *B*–*C* communication in figure 4(c) appears to take only two time units while the *C*–*B* communication appears to take ten.

The second problem with using figure 4-style communication traces involves how the simulator replays the traced communication pattern. Suppose we wanted to simulate a network that runs twice as fast as the one on which the communication trace was acquired or perhaps the same network attached to processors running three times as fast as on the measurement system. It would be unreasonable in either case to expect all of the messages to be sent at the same times shown in figure 4(a). A node that receives a message sooner or finishes some computation faster may then be able to send a message earlier. We therefore do not want our simulator necessarily to simulate messages being sent at the times listed in the input trace but rather at the times that the simulated supercomputer would actually send them.

The solution to both of the preceding problems is an abstraction called a *logical clock,* first proposed by Lamport in 1978 [2] and sometimes called a *Lamport clock* after its inventor. A logical clock is a simple, integer counter that "ticks" as follows:


Alice's Journal

I left home at 3:30 p.m. to drive my son to soccer practice. When I got to the soccer field my watch read 3:55 p.m.

I left the soccer field five minutes later, at 4:00 p.m., to join Carol for tea. I arrived at the tea house at the same time as Carol, at 4:30 p.m. according to my watch.

After a lovely hour of tea with Carol, we both drove home at the same time, when my watch read 5:30 p.m. just in time to make dinner.

Carol's Journal

I left home at 6:30 p.m. to join Alice for tea. I arrived at the tea house at 7:00 p.m. according to my watch, which is right when Alice arrived.

I had an enjoyable tea time with Alice. We both left 45 minutes later, at 7:45 p.m. on my watch, to drive home. I got home at 8:05 p.m.

FIGURE 5. The ordering of events from Alice's and Carol's perspective.

1. When a node performs any operation (communication or computation), its clock advances its logical time by one.
2. When a node sends a message, its clock includes the current logical time along with the normal data.
3. When a node receives a message, it sets its logical clock to the maximum of its current logical time and one plus the logical time included in the message.

These rules help define a "happened before" relation (mathematically, a partial ordering) on communication operations. If one operation occurred at a smaller logical time than another, then the simulator cannot perform the second operation until the first one finishes. In contrast, if two operations occur at the same logical time, the simulator has no restrictions on the order it performs them: it can run *A* then *B*, *B* then *A*, or both simultaneously. In essence, a logical clock provides a way to globally *order* communication operations regardless of the locally observed *time* at which each operation may appear to have occurred.

To clarify using yet another driving analogy, consider Alice's and Carol's sequences of events, presented in figure 5.

In what order did those events happen? It would be incorrect to sort them by the times listed in the event descriptions because Alice and Carol may not have synchronized their watches beforehand and because

Event	Logical time spent at various locations			
	Alice's House	Carol's House	Soccer Field	Tea House
(Our story begins)	1	1	1	1
Alice's → Soccer	1	1	**2**	1
Soccer → Tea	1	1	2	**3**
Tea → Alice's	**4**	1	2	3
Carol's → Tea	4	1	2	3
Tea → Carol's	4	1	2	**4**

Logical Time	Observable Events
1	Alice and Carol are both at home.
2	Alice is at the soccer field. Carol may be either at home or en route to the tea house. We have insufficient information to determine which.
3	Alice and Carol are both at the tea house.
4	Alice and Carol are both at home. We have insufficient information to determine who arrived first.

FIGURE 6. The ordering of events based on logical time.

either watch may run faster or slower than the other. Nevertheless, we can intuitively rely on what makes sense to order the specified events. Specifically, we know that Alice must have driven *to* the soccer field before driving *from* the soccer field; we know that both Alice and Carol were at the tea house at the same time; and we know that both Alice and Carol left the tea house at the same time after having tea together.

Figure 6 shows how to express that "what makes sense" intuition as formal statements of changes to logical time. The table assigns one logical clock to each *location* (as opposed to each person) mentioned and lists the events that Alice observed, in order, followed by the events that Carol observed, in order. (The results would be the same if we swapped or even interleaved Alice's and Carol's journals, as long as the events were not reordered relative to how they appear in either journal.) In our network-simulation framework, locations correspond to nodes, and a person driving from location to location corresponds to communication.

At the beginning, all locations are at logical time 1, and Alice and Carol are both in their respective home. When Alice drives to the soccer field, she must arrive some time after she was at home. The soccer field therefore increments its logical time to 2, the maximum of its current time (1) and one plus the time at Alice's house (1 + 1). When Alice drives to the tea house, she must arrive some time after she left the soccer field. The tea house therefore increments its logical time to 3, the maximum of its current time (1) and one plus the time at the soccer field (1 + 2). When Alice drives home, she must arrive both after the last time she was there (1) and after she left the tea house (3), that is to say, at time 4.

Turning our attention to Carol, Carol must arrive at the tea house at a time later than when she was at home. However, the tea house's clock does not change because the maximum of its current time (3) and one plus the time at Carol's house (1 + 1) is already 3. Finally, when Carol drives home, she must arrive both after the last time she was there (1) and after she left the tea house (3), that is to say, at time 4.

For clarity, the bottom part of figure 6 re-sorts the data by logical time, showing which events happened at each time. From this presentation, one can infer that despite the physical times stated in the event descriptions, Alice could not possibly have returned home before Carol arrived at the tea house (time 4 versus time 3). However, the logical-clock readings in figure 6 say nothing about whether Alice arrived back at her home before Carol arrived back at her home (time 4 for both events). More subtly, the readings do not indicate which of Alice or Carol arrived first at the tea house (as both soccer → tea and Carol's → tea completed at time 3); we know only that neither left (time 4) before both arrived (time 3).

Logical clocks provide an important mechanism for fulfilling the goals stated at the beginning of this section in that they enable a network simulator to reason about communication *dependencies*—what must happen before what—rather than physical times. One additional innovation of our network-simulation methodology is that we record computation time in physical time. In figure 6's analogy, this would be like a waiter at the tea house reporting how long Alice and Carol spent there. Maintaining this information enables the simulator to honor computation time, which may have substantial impact on communication time. Consider, for example, how much faster the cars in figure 2 would move if the yellow cars were on the road only in the morning and the red cars were on the road only in the afternoon.

Even without perfectly synchronized, drift-free node clocks, combining physical computation time with logical communication time enables us to accurately reproduce application timing measurements and provide some confidence that varying hardware parameters will lead to accurate predictions of performance. In the following section we quantify how well this works by presenting an early evaluation of our simulation methodology.

Initial results

Our simulation project is still in its early stages. However, the logical-time trace acquisition software and the simulator itself are operational and support a sufficient set of features for an initial evaluation of our approach.

As a sample application, we use a hydrodynamics code developed at Los Alamos National Laboratory called PAGOSA. PAGOSA is designed to simulate high-speed fluid flow and high-rate material deformation [1]. The application comprises approximately 67,000 lines of code (about 1,000 printed pages), mostly written in Fortran but with some C. PAGOSA's constituent processes are logically arranged in a three-dimensional layout and communicate primarily with their immediate north, south, east, west, front, and back neighbors. This is an ideal structure for a three-dimensional network such as the one shown in figure 1(c) *if* the application's coordinates directly map to the network's coordinates. For example, mapping a 6 × 6 × 6 PAGOSA layout onto a 6 × 6 × 6 network could be expected to perform well. In contrast, mapping it onto a 6 × 4 × 9 network would in fact make some "neighbors" not adjacent to each other, leading to link contention. In practice, users are rarely given control over the set of nodes allocated to their applications.

We ran PAGOSA on 1,000 nodes of a 1,600-node supercomputer called Mustang. Mustang is based on a fat-tree network such as the one shown in figure 1(b), but 200 times larger. More precisely, figure 1(b) represents what is often called a *2-ary 3-tree,* because each switch connects to two switches in each adjacent row and there are three rows of switches. Mustang uses an 18-ary 3-tree[a] so each switch connects to 18 switches in each adjacent row, but there are still only three rows in the network, just as in figure 1(b). As of June 2013, Mustang was rated the 137th fastest supercomputer in the world [3].

Full-application simulation at scale

PAGOSA was configured to execute a canonical hydrodynamics test problem, the simulation of a spherical shell of beryllium being subjected from all directions to a given amount of kinetic energy, which compresses the shell. Figure 7 presents the results of simulating this PAGOSA execution using the sets of network parameters listed in table 1.

The first bar, labeled *Fat tree, measured,* indicates that the PAGOSA test problem normally takes an hour and a half to complete on Mustang. The second bar, *Fat tree,* demonstrates that our simulator is quite accurate, being only 6.4% above the correct value. Recall that our work is still in its early stages; we hope in the near future to improve simulation accuracy. The second and subsequent bars each represent between 14½ and 15½ hours of time running the simulator on a single desktop computer. This is a noteworthy success: Even though we used a thousandth of the number of nodes as in the real execution, our simulator took only tenfold the time to run. And, unlike real execution,

a. Mustang in fact contains an incomplete 18-ary 3-tree—an XGFT (3; 18, 6, 16; 1 6 18) in Öhring's notation [4]—and this is what we simulate.

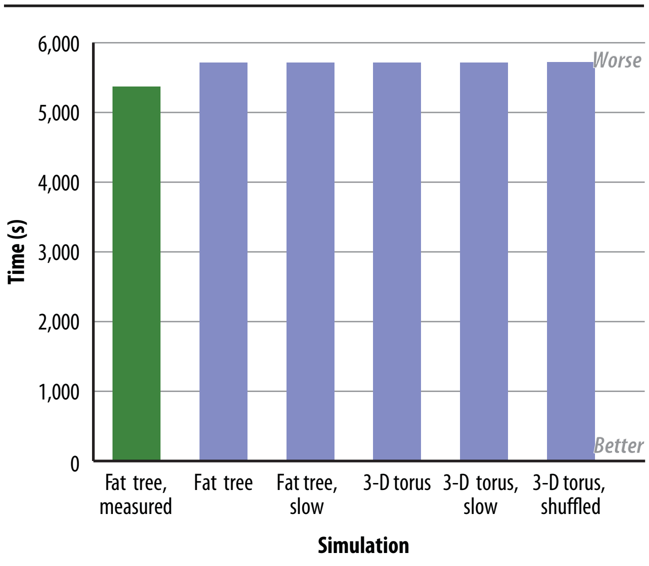


FIGURE 7. Simulated PAGOSA execution time using the sets of network parameters in table 1.

our simulator enables limitless "what if" experimentation with different network topologies and network performance characteristics.

As a demonstration of that capability, the remaining bars in figure 7 show the results of simulating different networks from Mustang's actual network. As detailed in table 1, *Fat tree, slow* represents a substantially slower network than *Fat tree. 3-D torus* uses the same network speeds as *Fat tree* but with a 3-D torus topology instead of a fat tree. Likewise, *3-D torus, slow* uses the same network speeds as *Fat tree, slow* but with a 3-D torus topology instead of a fat tree. *3-D torus, shuffled* represents the same topology and network speeds as *3-D torus* but randomly shuffles the mapping of PAGOSA processes to torus nodes. Torus networks are notoriously sensitive to process placement, and we can use our simulation technology to evaluate how sensitive a given application is to the placement of its constituent processes.

The clear implication of figure 7 is that PAGOSA's overall performance is almost completely oblivious to network performance. Despite the simulated variations in network topologies and speeds, the difference in execution time from one network to another is a tiny fraction of a percent. Although the 1,000-node run of PAGOSA communicated an aggregate of two billion messages comprising a total of 14 terabytes of data, communication time is so dominated by computation time that network speed is largely inconsequential.

Comparison with simplistic simulators

We have shown that flow-based simulation delivers simulation speed and that logical clocks provide high fidelity to actual application execution time. The next question to consider is how our approach compares to the more simplistic approach employed by most network-simulation studies. While our simulator honors both communication dependencies and computation time, it is far more typical in the simulation literature to pretend that all messages are sent simultaneously at time 0 and to simulate the time it takes all messages to reach their destination in the absence of computation.

We configured our simulator to disregard communication dependencies and computation time, in essence dumbing down our simulator to the capabilities of a more traditional network simulator. The results, shown in figure 8, paint a very different picture of performance from figure 7.

The total height of each bar represents the time for the last message in the corresponding simulation to complete. The light purple region represents the average time for a message to complete. While figure 7 indicates that PAGOSA's total execution time is almost completely independent of communication time, figure 8 exaggerates the differences. Specifically,

TABLE 1. Simulation parameters

Simulation	Topology	Link Speed (Gbps)	Switch Latency (ns)	Software Overhead (ns)
Fat tree	18-ary 3-tree	40	100	1,500
Fat tree, slow	18-ary 3-tree	10	400	4,000
3-D torus	8 x 16 x 16 torus	40	100	1,500
3-D torus, slow	8 x 16 x 16 torus	10	400	4,000
3-D torus, shuffled	8 x 16 x 16 torus	40	100	1,500

the 3-D torus requires 70% more time than the fat tree to transfer PAGOSA's two billion messages. For both network topologies, quartering the bandwidth exactly quadruples the communication time.

This study demonstrates that it is critical to include communication dependencies and computation time in a network simulation. Otherwise, differences in network topology and basic performance characteristics appear more significant than they really are. This misleading information could persuade a supercomputing center to pay extra for a faster network when a slower, less expensive network may deliver almost exactly the same performance to applications.

Conclusions

Modern supercomputers are architected as vast aggregations of processors interconnected with high-speed networks. Because scientific applications are generally composed of myriad processes working together to simulate natural phenomena, communication speed is critical for efficiently coordinating all of those processes. However, engineering a high-speed network involves inevitable cost/performance trade-offs. Furthermore, all applications use the network differently, contraindicating a one-size-fits-all solution. Some applications transmit a large number of small messages; others transmit a small number of large messages. In some applications, each node communicates with only a small set of other nodes; in others, all nodes communicate with all of the others. Some applications communicate continuously throughout their execution; others alternate communication and computation phases.

Supercomputing centers want to maximize the overall performance delivered to the applications they expect to run but without overpaying for unnecessary network performance. One way to predict how well a given application will perform on a particular network in advance of its procurement is via a technique called *network simulation*. With simulation, one mimics hardware's behavior and performance characteristics using a software test bed. Simulating hardware is slower—typically many thousands of times slower—than running on true hardware but is cheap to deploy and easy to modify to investigate different design alternatives.

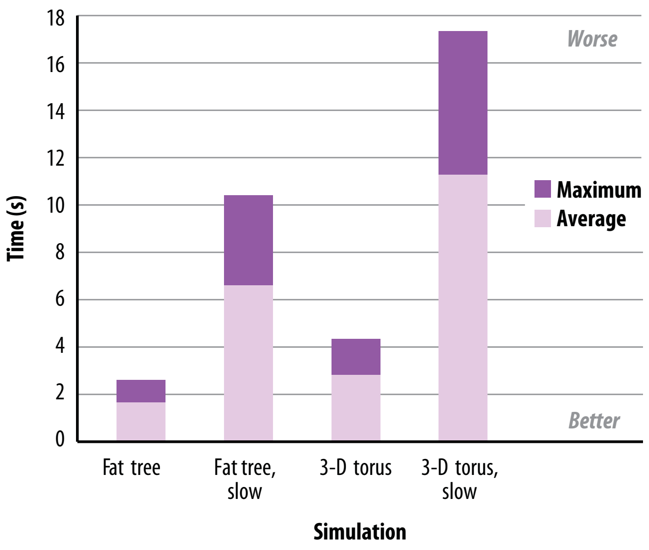


FIGURE 8. Differences in simulated communication time only.

The problem with existing network simulators and simulation studies is that they tend to incorporate so much detail that they cannot handle large numbers of nodes or substantial lengths of time. Furthermore, for simplicity of implementation they ignore the juxtaposition of communication with computation and with other communication, unrealistically assuming that all messages are initiated in a single burst. In this article we proposed addressing the speed issue with flow-based simulation and the realistic-usage issue with logical clocks that are augmented with physical computation time. To demonstrate the potential of this approach we implemented a tool to derive logical-time traces from parallel applications and a flow-based simulator to replay those traces on different simulated network topologies and with different network performance characteristics.

One can draw the following conclusions from the experimental data we presented. First, our approach accurately simulates real execution time. Although our implementation is in its nascent stages, we already saw less than 7% error when simulating a scientific application, PAGOSA, running for an hour and a half across a 1,000-node network. Second, flow-based simulation runs at reasonable speeds. We replayed that 1,000-node, hour-and-a-half run on different simulated networks using only a single node, and it ran only 10 times slower than real time, not thousands or tens of thousands, which is what is typical for a more traditional network simulator. Third, the common simplification of ignoring communication dependencies and computation time in network simulations exaggerates the pressure the application applies to the network and leads to incorrect assessments of network performance.

In our experiments, we found that PAGOSA performs so much computation relative to communication that the network topology and basic performance characteristics are largely inconsequential. In contrast, a more traditional network simulator would incorrectly claim 70% more performance for a fat-tree topology than for a 3-D torus topology when replaying PAGOSA's communication pattern.

In summary, combining logical time with flow-based simulation opens up new avenues for evaluating how fast applications will run on different supercomputer networks, most notably supercomputer networks that have not yet been built. This capability can inform network design decisions—or even simply a selection from multiple existing networks—to help provide applications with the best communication performance that the supercomputer budget allows.

About the authors

Scott Pakin is a research scientist at Los Alamos National Laboratory. He has been actively working in the area of high-performance network research for over 15 years, beginning with the development of Fast Messages, one of the first high-speed messaging layers for a commodity supercomputing network, Myrinet; and more recently including the Cell Messaging Layer, which makes it practical for computational accelerators to communicate directly across a deep, heterogeneous network hierarchy; and coNCePTuaL, a domain-specific language, compiler, and run-time system that facilitate the rapid generation of custom network speed tests with repeatable results.

Dr. Pakin has served on numerous network-related national and international conference and workshop program committees, including the position of area cochair for the Architecture and Networks track of this year's annual Supercomputing Conference (SC'13) and continuing cochair service for the annual Communication Architecture for Scalable Systems (CASS) workshop. He also served as a guest editor for the November 2012 special issue of Elsevier's *Journal of Parallel and Distributed Computing,* which focused on interconnection networks. Dr. Pakin received a BS in mathematics/computer science with research honors from Carnegie Mellon University in 1992, an MS in computer science from the University of Illinois at Urbana-Champaign in 1995, and a PhD in computer science from the University of Illinois at Urbana-Champaign in 2001.

Xin Yuan is a full professor in the Department of Computer Science at Florida State University and recently took a research sabbatical at Los Alamos National Laboratory. His research interests include parallel and distributed systems, interconnection networks, communication optimizations, and networking. He obtained his BS and MS degrees in computer science from Shanghai Jiaotong University in 1989 and 1992, respectively. He earned his PhD degree in computer science from the University of Pittsburgh in 1998. He publishes extensively on interconnection networks and communication-library implementation and optimizations.

The Self-Tuned Adaptive Routines for Message Passing Interface (STAR-MPI) software package that he and his students developed has been incorporated into the software stack of IBM's Blue Gene/P supercomputer. Professor Yuan is currently serving on the editorial boards of several international journals. He has also served as the program chair and vice chair for several international conferences and workshops, such as the International Conference on Parallel Processing (ICPP) and the Institute of Electrical and Electronics Engineers (IEEE) International Conference on High Performance Computing (HiPC), and as a program committee member for many international conferences and workshops. He is a senior member of both the Association for Computing Machinery (ACM) and IEEE.

Michael Lang is the team leader of the Ultrascale Systems Research at Los Alamos National Laboratory. His research interests include distributed services, performance of large-scale systems, operating-system and run-time issues for supercomputers, and interconnects for large-scale systems. He has published work on the application-specific optimization of routing on InfiniBand interconnects for large-scale systems. Notably, this algorithm is currently included in OpenSM in the OpenFabrics software stack. Lang was formerly a member of Los Alamos National Laboratory's Performance and Architecture team, involved in performance analysis of new large-scale systems for the US Department of Energy. He received a BS in computer engineering and an MS in electrical engineering in 1988 and 1993 respectively, both from the University of New Mexico.

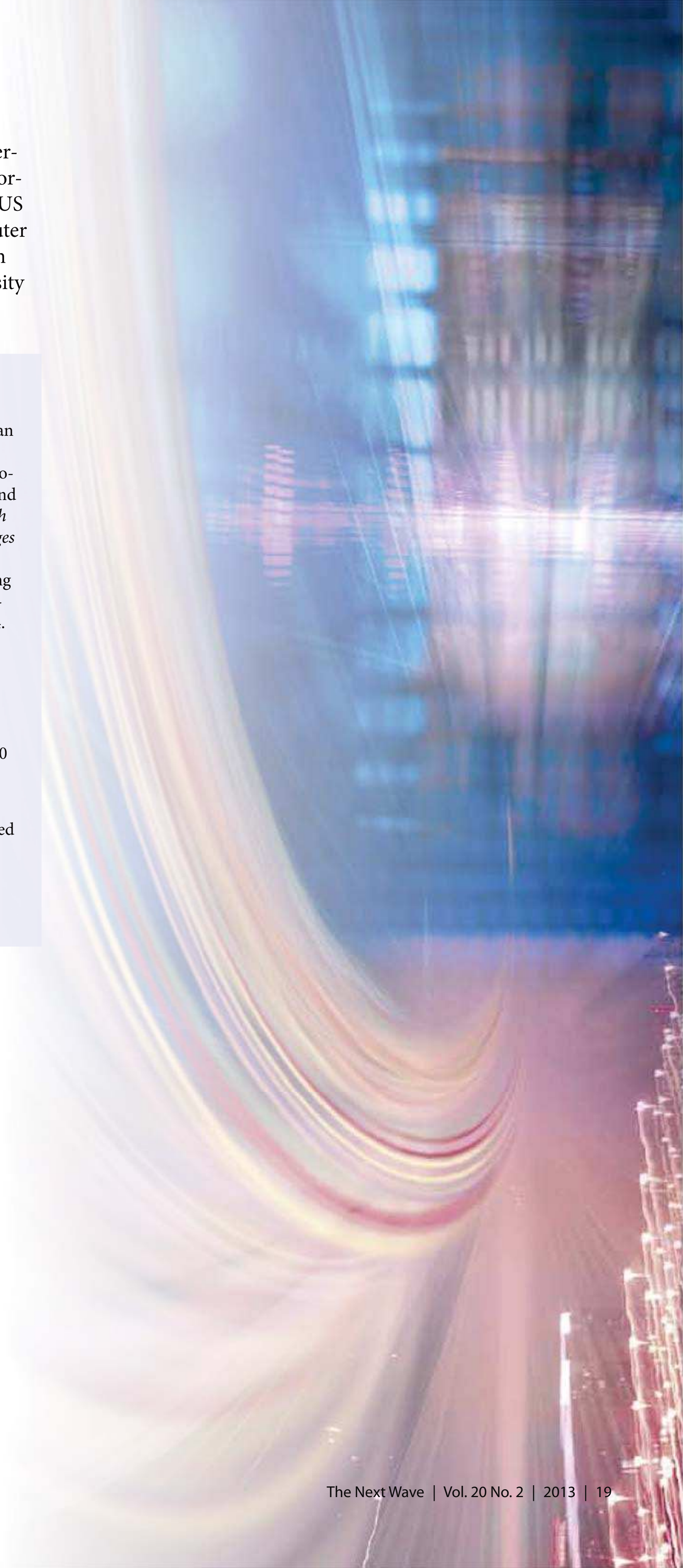
References

[1] Kothe DB, Baumgardner JR, Cerutti JH, Daly BJ, Holian KS, Kober EM, Mosso SJ, Painter JW, Smith RD, Torrey MD. "PAGOSA: A massively parallel, multi-material hydrodynamics model for three-dimensional high-speed flow and high-rate material deformation. In: Tentner A, editor. *High Performance Computing Symposium 1993: Grand Challenges in Computer Simulation* (Proceedings of the 1993 Simulation Multiconference on the High Performance Computing Symposium; Mar 29–Apr 1, 1993, Arlington, VA). San Diego (CA): Society for Computer Simulation; 1993. p. 9–14. ISBN: 978-1565550520.

[2] Lamport L. "Time, clocks, and the ordering of events in a distributed system." *Communications of the ACM.* 1978;21(7):558–565. doi: 10.1145/359545.359563.

[3] Meuer H, Strohmaier E, Dongarra J, Simon H. TOP500 Supercomputer Sites: June 2013 [accessed 2013 Aug 2]. Available at: http://www.top500.org/lists/2013/06.

[4] Öhring SR, Ibel M, Das SK, Mohan J K. "On generalized fat trees." In: *Proceedings of the 9th International Parallel Processing Symposium;* Apr 1995, Santa Barbara, (CA). p. 37–44. doi: 10.1109/IPPS.1995.395911.

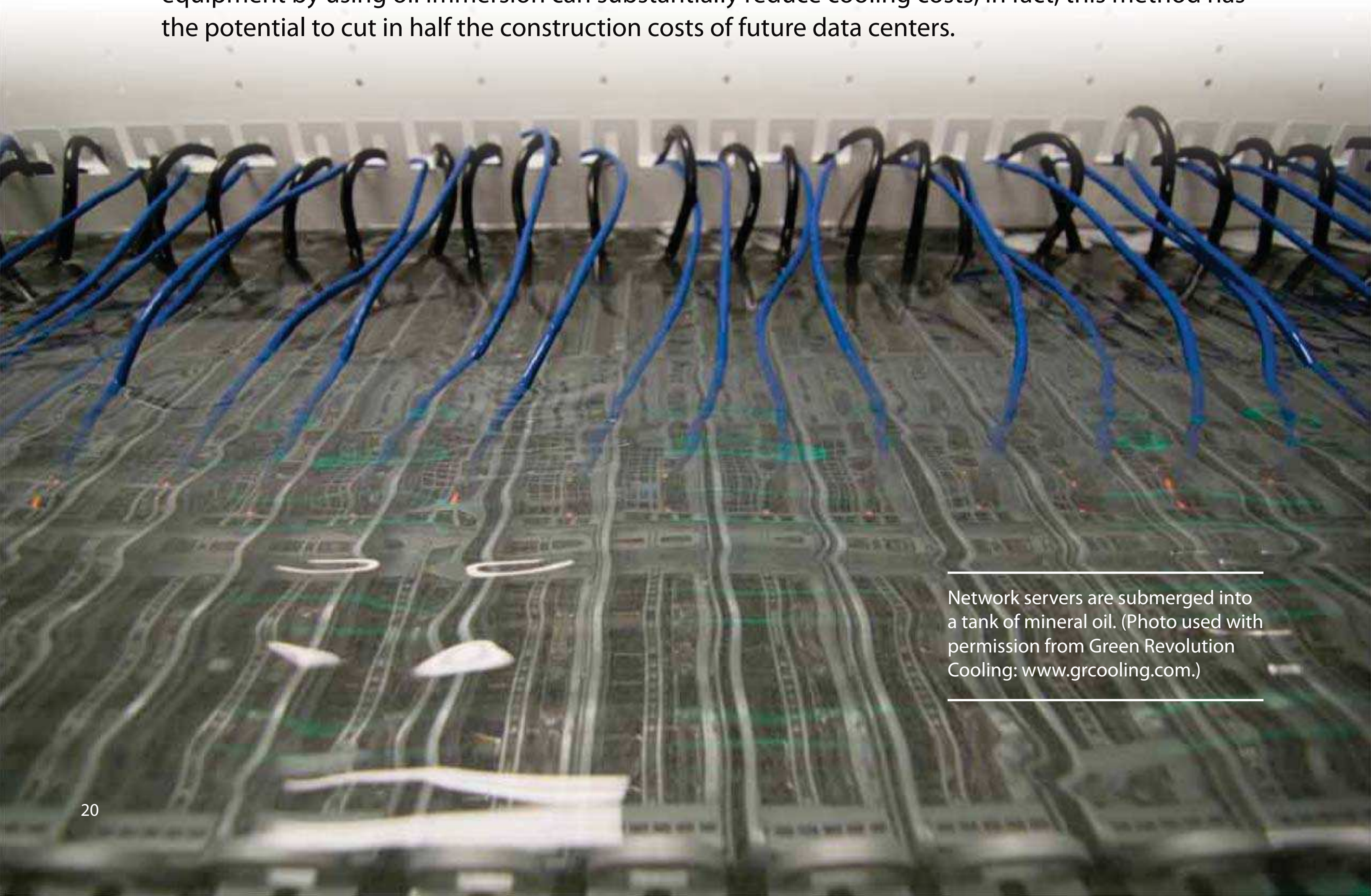


Doing more with less: Cooling computers with oil pays off

David Prucnal, PE

A consequence of doing useful work with computers is the production of heat. Every watt of energy that goes into a computer is converted to a watt of heat that needs to be removed, or else the computer will melt, burst into flames, or meet some other undesirable end. Most computer systems in data centers are cooled with air conditioning, while some high-performance systems use contained liquid cooling systems where cooling fluid is typically piped into a cold plate or some other heat exchanger.

Immersion cooling works by directly immersing IT equipment into a bath of cooling fluid. The National Security Agency's Laboratory for Physical Sciences (LPS) acquired and installed an oil-immersion cooling system in 2012 and has evaluated its pros and cons. Cooling computer equipment by using oil immersion can substantially reduce cooling costs; in fact, this method has the potential to cut in half the construction costs of future data centers.


Network servers are submerged into a tank of mineral oil. (Photo used with permission from Green Revolution Cooling: www.grcooling.com.)

The fundamental problem

Before getting into the details of immersion cooling, let's talk about the production of heat by computers and the challenge of effectively moving that heat from a data center to the atmosphere or somewhere else where the heat can be reused.

In order for computers to do useful work, they require energy. The efficiency of the work that they do can be measured as the ratio of the number of operations that they perform to the amount of energy that they consume. There are quite a few metrics used to measure computer energy efficiency, but the most basic is operations per watt (OPS/W). Optimizing this metric has been the topic of many PhD theses and will continue to be the subject of future dissertations. Over the years, there has been progress against this metric, but that progress has slowed because much of the low-hanging fruit has been harvested and some of the key drivers, Moore's Law and Denard scaling, have approached the limits of their benefit. Improvements to the OPS/W metric can still be made, but they usually come at the expense of performance.

The problem is not unlike miles per gallon for cars. The internal combustion engine is well understood and has been optimized to the *n*th degree. For a given engine, car weight, and frontal area, the gas mileage is essentially fixed. The only way to improve the miles per gallon is to reduce the performance or exploit external benefits. In other words, drive slower, accelerate less, drift down hills, find a tailwind, etc. Even after doing all of these things, the improvement in gas mileage is only marginal. So it is, too, with computers. Processor clock frequencies and voltages can be reduced, sleep modes can be used, memory accesses and communications can be juggled to amortize their energy costs, but even with all of this, the improvement in OPS/W is limited.

A natural consequence of doing useful work with computers is the production of heat. Every watt of energy that goes into a computer is converted into a watt of heat that needs to be removed from the computer, or else it will melt, burst into flames, or meet some other undesirable end. Another metric, which until recently was less researched than OPS/W, is kilowatts per ton (kW/ton), which has nothing to do with the weight of the computer system that is using up the energy. Here, ton refers to an amount of air conditioning; hence, kW/ton has to do with the amount of energy used to expel the heat that the computer generates by consuming energy (see figure 1).

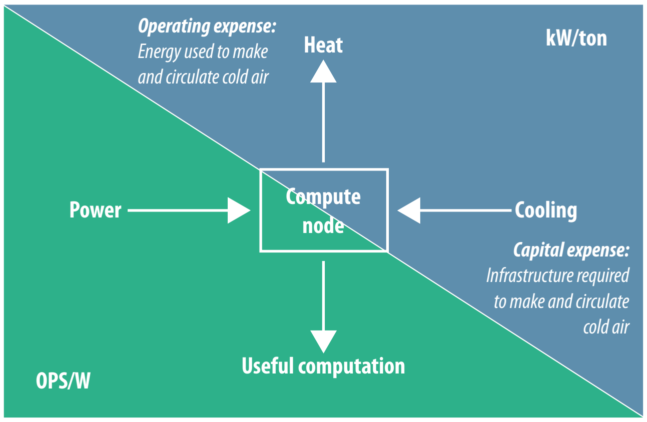

Operating expense:
Energy used to make and circulate cold air
Heat
kW/ton
Power
Compute node
Cooling
Capital expense:
Infrastructure required to make and circulate cold air
Useful computation
OPS/W

FIGURE 1. There are two halves to the computer power efficiency problem: efficiency of the actual computation (green sector) and efficiency of the cooling infrastructure (blue sector).

In fact, many traditional data centers consume as much energy expelling heat as they do performing useful computation. This is reflected in a common data center metric called power usage effectiveness (PUE), which in its simplest form is the ratio of the power coming into a data center to the power used to run the computers inside. A data center with a PUE of 2.0 uses as much power to support cooling, lighting, and miscellaneous loads as it does powering the computers. Of these other loads, cooling is by far the dominant component. So, another way to improve data center efficiency is to improve cooling efficiency. The best case scenario would be to achieve a PUE of 1.0. One way to achieve this would be to build a data center in a location where the environmental conditions allow for free cooling. Some commercial companies have taken this approach and built data centers in northern latitudes with walls that can be opened to let in outside air to cool the computers when the outside temperature and humidity are within allowable limits. However, for those of us who are tied to the mid-Atlantic region where summers are typically hot and humid, year-round free cooling is not a viable option. How can data centers in this type of environment improve their kW/ton and PUE?

How computers are cooled

There are many different ways that computers are kept cool in data centers today; however, the most common method is to circulate cool air through the chassis of the computer. Anyone who has ever turned on a computer knows that the computer makes noise when it is powered on. Central processing units (CPUs), memory, and any other solid-state components are completely silent, so what makes the noise? Spinning disk drives can make a little noise, but by far, the dominant noisemakers are the fans that are used to keep air moving across the solid-state devices that are all busily doing work and converting electrical power input into computation and heat. Even the power supply in a computer has a fan because the simple act of converting the incoming alternating current (ac) power to usable direct current (dc) power and stepping that power down to a voltage that is usable by the computer creates heat. All of the fans in a computer require power to run, and because they are not perfectly efficient, they too create a little heat when they run. The power used to run these fans is usually counted as computer load, so it ends up in the denominator of the PUE calculation, even though it does nothing toward actual computation.

But how do all of these fans actually cool the computer? Think of cooling as heat transfer. In other words, when an object is cooled, heat is transferred away from that object. What do people do when they burn their finger? They blow on it, and if they are near a sink, they run cold water on it. In both cases they are actually transferring heat away from their burnt finger. By blowing, they are using air to push heat away from their finger, and by running water, they are immersing their hot finger in a cool fluid that is absorbing and carrying the heat away. Anyone who has burned a finger knows that cold water brings much more relief than hot breath. But why? The answer depends on principles like thermal conductivity and heat capacity of fluids. It also helps to understand how heat moves.

Heat on the go: Radiation, conduction, convection, and advection

Imagine a campfire on a cool evening. The heat from the fire can be used to keep warm and to roast marshmallows, but how does the heat move from the fire? There are three modes of heat transfer at work around a campfire: radiation, conduction, and convection (see figure 2). As you sit around the fire, the heat that moves out laterally is primarily *radiant* heat. Now, assume you have a metal poker for stirring the coals and moving logs on the fire. If you hold the poker in the fire too long it will start to get hot in your hand. This is because the metal is *conducting* heat from the fire to your hand. To a much lesser extent the air around the fire is also conducting heat from the fire to you. If you place your hands over the fire, you will feel very warm air rising up from the fire. This heat transfer, which results from the heated air rising, is *convection*. Now, if an external source, such as a breeze, blows across the fire in your direction, in addition to getting smoke in your eyes, you will feel heat in the air blowing towards you. This is *advection*. In a computer, a CPU creates heat that is typically conducted through a heat spreader and then into the surrounding air. Convection causes the air to rise from the heat spreader, where it is then blown, or advected, away by the computer's cooling fan.

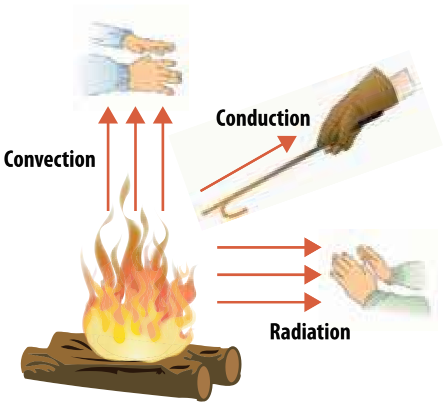


FIGURE 2. There are three modes of heat transfer at work around a campfire: radiation, conduction, and convection.

Now that we know how heat moves, why is it that it feels so much better to dunk a burnt finger in water than to blow on it? This is where thermal conductivity and heat capacity of the cooling fluid come into play. First, a few definitions:

- **Thermal conductivity** is the ability of a material to conduct heat; it is measured in watts per meter degree Celsius, or W/(m·°C).
- **Heat capacity** is the amount of heat required to change a substance's temperature by a given amount or the amount of heat that a substance

can absorb for a given temperature increase; it is measured in joules per degree Celsius (J/°C).

- **Specific heat capacity** is the heat capacity per unit mass or volume; it is typically given per unit mass and simply called specific heat (C_p); it is measured in joules per gram degree Celsius, or J/(g·°C).

The answer to why it feels so much better to dunk a burnt finger into water than to blow on it can be found in table 1. First, water is a much better conductor of heat than air, by a factor of 24. Think of it as having 24 times more bandwidth for moving heat. Second, water can hold far more heat than air. In fact, 3,200 times more. So, water provides 24 times more heat transfer bandwidth and 3,200 times more heat storage than air. No wonder the finger feels so much better in the water.

One more thing to consider about heat transfer; heat naturally flows from hot to cold, and the rate of heat transfer is proportional to the temperature difference. This is why the colder the water, the better that burnt finger is going to feel.

Cooling computers

By now it should be apparent that the fans in a computer are there to advect (i.e., move) a cooling fluid (e.g., air) across the heat producing parts (e.g., CPUs, memories, and peripheral component interconnect cards) so that the cooling fluid can absorb heat through conduction and carry it away. This can be described by the following mass flow heat transfer equation: $\dot{Q} = \dot{m}\, c_p \Delta T$

In this equation, $\dot{Q}$ is the rate of heat transfer in watts, $\dot{m}$ is the mass flow rate of the cooling fluid in grams per second, c_p is the specific heat of the cooling fluid, and ΔT is the change in temperature of the cooling fluid. What it says is that the cooling depends on production of the amount of coolant flowing over the heat source, the ability of the coolant to hold heat, and the temperature rise in the coolant as it flows across the heat source.

How much air does it take to keep a computer cool? There is a rule of thumb used in the data center design world that 400 cubic feet per minute (CFM) of air is required to provide 1 ton of refrigeration. One ton of refrigeration is defined as 12,000 British thermal units per hour (Btu/h). Given that 1 kilowatt-hour is equivalent to 3,412 British thermal units, it can be seen that a ton of refrigeration will cool a load of 3,517 W, or approximately 3.5 kW. The mass flow heat transfer equation can be used to confirm the rule of thumb. Air is supplied from a computer room air conditioning (CRAC) unit in a typical data center at about 18°C (64°F). Now, 400 CFM of air at 18°C is equivalent to 228 grams per second, and the specific heat of air is equivalent to 1 J/(g·°C). Solving the mass flow heat transfer equation above with this information yields a change in temperature of 15°C. What all this confirms (in Fahrenheit) is that when 64°F cooling air is supplied at a rate of 400 CFM per 3.5 kW of computer load, the exhaust air from the computers is 91°F. Anyone who has stood in the "hot aisle" directly behind a rack of servers will know that this rule of thumb is confirmed.

It is not unusual for a server rack to consume over 10 kW. Using the rule of thumb above, a 10.5 kW server rack requires 1,200 cubic feet of cooling air—enough air to fill a 150 square foot office space with an 8 foot ceiling—per minute. That's a whole lot of air! Simply moving all of that air requires a significant amount of energy. In fact, for racks of typical one-unit (1U) servers, the energy required to move cooling air from the CRAC units and through the servers is on the order of 15% of the total energy consumed by the computers. Remember—this is just the energy to move the cooling air, it does not include the energy required to make the cold air.

TABLE 1. Thermal conductivity and heat capacity of common substances

	Thermal Conductivity, W/(m·°C) at 25°C	Specific Heat (C_p), J/(g·°C)	Volumetric Heat Capacity (Cv), J/(cm^3·°C)
Air	0.024	1	0.001297
Water	0.58	4.20	4.20
Mineral Oil	0.138	1.67	1.34
Aluminum	205	0.91	2.42
Copper	401	0.39	3.45

If there was a way to cool computers without moving exorbitant quantities of air, it could reduce energy consumption by up to 15%. This may not seem like much, but consider that a 15% improvement in OPS/W is almost unheard of, and for a moderate 10 megawatt (MW) data center, a 15% reduction in energy consumption translates into a savings of $1.5 million per year.

Pumping oil versus blowing air

Unfortunately, we cannot dunk a computer in water like a burnt finger since electricity and water do not play well together. Mineral oil, on the other hand, has been used by electric utilities to cool electrical power distribution equipment, such as transformers and circuit breakers, for over 100 years. Mineral oil only has about 40% of the heat holding capacity and about one quarter the thermal conductivity of water, but it has one huge advantage over water—it is an electrical insulator. This means that electrical devices can operate while submerged in oil without shorting out.

While mineral oil does not have the heat capacity of water, it still holds over 1,000 times more heat than air. This means that the server rack discussed earlier that needed 1,200 CFM of air to keep from burning up could be kept cool with just about 1 CFM of oil. The energy required to pump 1 CFM of oil is dramatically less than the energy required to blow 1,200 CFM of air. In a perfectly designed data center, where the amount of air blown or oil pumped is matched exactly to the heat load, the energy required to blow air is five times that required to pump oil for the same amount of heat removed. In reality, the amount of air moved through a data center is far more than that required to satisfy the load. This is due to the fact that not all of the air blown into a data center passes through a computer before it returns to the CRAC unit. Since the air is not ducted directly to the computers' air intakes, it is free to find its own path back to the CRAC unit, which is frequently over, around, or otherwise not through a server rack. As we will soon see, it is much easier to direct the path of oil and to pump just the right amount of oil to satisfy a given computer heat load. Thus, the energy required to circulate oil can be more than 10 times less than the energy required to circulate air.

Immersion cooling system

Now that we have established that mineral oil would be a far more efficient fluid to use for removing heat from computers, let's look at how a system could be built to take advantage of this fact.

Imagine a rack of servers. Now imagine that the rack is tipped over onto its face. Now convert the rack into a tub full of servers. Now fill the tub with mineral oil.

Figures 3 and 4 show the system that LPS acquired and is using in its Research Park facility. The system is comprised of a tank filled with mineral oil that holds the servers and a pump module that contains an oil-to-water heat exchanger and oil circulation pump. In this installation, the heat exchanger is tied to the facility's chilled water loop; however, this is not a necessity, as will be discussed later. The oil is circulated between the tank and the heat exchanger by a small pump. The pump speed is modulated to maintain a constant temperature in the tank. This matches the cooling fluid supply directly to the load. The design of the tank interior is such that the cool oil coming from the heat exchanger is directed so that most of it must pass through the servers before returning to the heat exchanger. The combination of pump speed modulation and oil ducting means that the cooling fluid is used very efficiently. The system only pumps the amount needed to satisfy the load, and almost all of what is pumped passes through the load.

There are three interesting side benefits to immersion cooling in addition to its efficiency. The first is due to the fact that the system is designed to maintain a constant temperature inside the tank. Because the pump is modulated to maintain a set point temperature regardless of changes in server workload, the servers live in an isothermal environment. One of the causes of circuit board failures is due to the mismatch in the coefficients of thermal expansion, or CTEs. The CTEs for the silicon, metal, solder, plastic, and fiberglass used in a circuit board are all different, which means that these materials expand and contract at different rates in response to temperature changes. In an environment where the temperature is changing frequently due to load changes, this difference in CTEs can eventually lead to mechanical failures on the

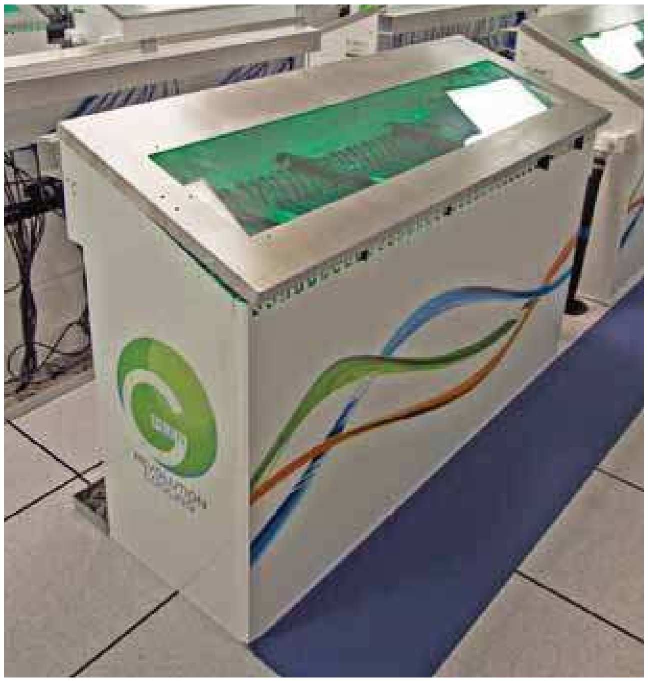


FIGURE 3. The immersion cooling system at the Laboratory for Physical Sciences, like the one pictured above, uses mineral oil to cool IT equipment. (Photo used with permission from Green Revolution Cooling: www.grcooling.com.)

circuit board. Oil immersion reduces this problem by creating a temperature-stable environment.

The second side benefit is server cleanliness. Air-cooled servers are essentially data center air cleaners. While data centers are relatively clean environments, there is still some dust and dirt present. Remember, a typical server rack is drawing in a large office space full of air every minute. Any dust or dirt in that air tends to accumulate in the chassis of the servers.

The final side benefit of immersion cooling is silence. Immersion cooling systems make virtually no noise. This is not an insignificant benefit, as many modern air-cooled data centers operate near or above the Occupational Safety and Health Administration's allowable limits for hearing protection.

In addition to efficient use of cooling fluid and the side benefits mentioned above, there is another advantage to immersion cooling—server density. As mentioned earlier, a typical air-cooled server rack consumes about 10 kW. In some carefully engineered HPC racks, 15–20 kW of load can be cooled with air.

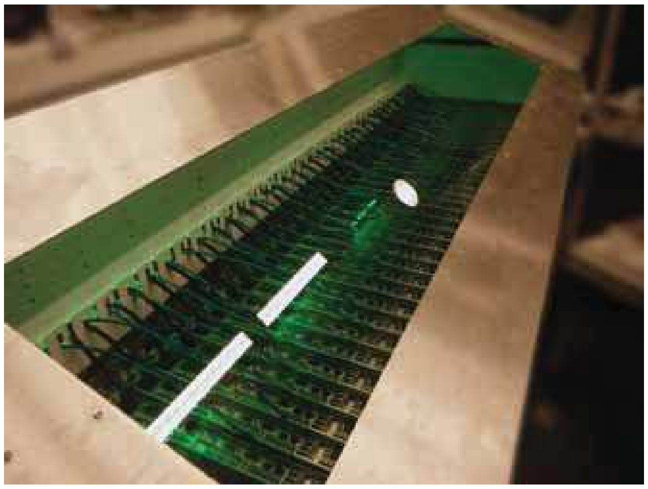


FIGURE 4. Network servers are submerged into a tank of mineral oil and hooked up to a pump that circulates the oil. (Photo used with permission from Green Revolution Cooling: www.grcooling.com.)

In comparison, the standard off-the-shelf immersion cooling system shown in figure 3 is rated to hold 30 kW of server load with no special engineering or operating considerations.

Doing more with less

Let's take a look at how immersion cooling can enable more computation using less energy and infrastructure.

Air cooling infrastructure

Cooling air is typically supplied in a computer room with CRAC units. CRAC units sit on the computer room raised floor and blow cold air into the underfloor plenum. This cold air then enters the computer room through perforated floor tiles that are placed in front of racks of computers. Warm exhaust air from the computers then travels back to the top of the CRAC units where it is drawn in, cooled, and blown back under the floor. In order to cool the air, CRAC units typically use a chilled-water coil, which means that the computer room needs a source of chilled water. The chilled water (usually 45–55°F) is supplied by the data center chiller plant. Finally, the computer room heat is exhausted to the atmosphere outside usually via evaporative cooling towers.

Oil-immersion systems also need to expel heat, and one way is through the use of an oil-to-water heat exchanger; this means that oil-immersion systems, like CRAC units, need a source of cooling water. The big difference however is that CRAC units need 45–55°F water; whereas, oil-immersion systems can operate with cooling water as warm as 85°F. Cooling towers alone, even in August in the mid-Atlantic area, can supply 85°F water without using power-hungry chillers. Because oil-immersion systems can function with warm cooling water, they can take advantage of various passive heat sinks, including radiators, geothermal wells, or nearby bodies of water.

The takeaway here is that there is a significant amount of expensive, energy-hungry infrastructure required to make and distribute cold air to keep computers in a data center cool. Much of this infrastructure is not required for immersion cooling.

Fan power

One of the primary benefits of immersion cooling is the removal of cooling fans from the data center. Not only are the energy savings that result from the removal of cooling fans significant, they are compounded by potentially removing the necessity for CRAC units and chillers.

Cooling fans in a typical 1U rack-mounted server consume roughly 10% of the power used by the server. Servers that are cooled in an oil-immersion system do not require cooling fans. This fact alone means that immersion cooling requires approximately 10% less energy than air cooling. Internal server fans, however, are not the only fans required for air-cooled computers. CRAC unit fans are also necessary in order to distribute cold air throughout the data center and present it to the inlet side of the server racks.

This CRAC unit fan power must be considered when determining the actual fan-power savings that can be realized by immersion cooling systems. Table 2 compares the power required to move 1 W of exhaust heat into a data center's chilled water loop for fan-blown air cooling versus pump-driven oil-immersion cooling. The third column shows this power as a percentage of IT technical load. It shows that the power required to run all fans in an air-cooled system is equal to 13% of the technical load that is being cooled. This is contrasted with the power required to run pumps in an oil-immersion cooling system, which is equal to 2.5% of the technical load that is being cooled. The difference, 10.5%, represents the net fan-power savings achieved by switching from an air-cooled to immersion-cooled data center. This analysis assumes that in both the air-cooled and immersion-cooled cases, the cooling infrastructure is matched exactly to the load. The last column in table 2 uses a similar analysis but assumes that the cooling infrastructure capacity is provisioned at twice the load. It shows that overprovisioned fan power grows faster than overprovisioned pump power. This is further illustrated in figure 5.

Lower operating expenses

Table 3 compares the fan power versus pump power required to serve a 1 MW technical load, assuming the cooling infrastructure is sized to serve 150% of the load. It shows that the fan power to circulate cold air exceeds the pump power to circulate oil by 158 kW per megawatt of technical load. At one million dollars per megawatt-year, this equates to $158,000 a year in additional cooling energy operating expense. This represents the savings due solely to circulating cooling fluid. When the cost of making cold air is considered, the energy savings of immersion cooling becomes much more significant.

Table 4 summarizes the energy required for air cooling that is not needed for immersion cooling. The values in Table 4 are typical for reasonably efficient data centers.

TABLE 2. **Power usage for air-cooled versus immersion-cooled data centers**

Method of Cooling	Power Required to Move 1W of Waste Heat into Chilled Water Loop (W)	Percentage of Technical Load to Power Fan or Pump (at 100%)	Percentage of Technical Load to Power Fan or Pump (at 200%)
Fan-Powered Air	0.13 W	13%	26%
Pump-Powered Oil Immersion	0.025 W	2.5%	5%
Net savings due to fan removal		*10.5%*	*21%*

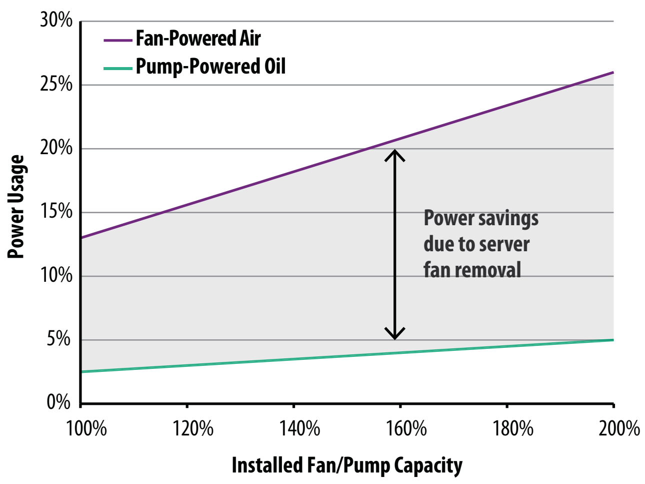


FIGURE 5. The power required to run the fans in an air-cooled data center (purple line) accounts for about 13% of the center's technical load (26% if run at twice the technical load); whereas, the power required to run the pumps in an immersion-cooled data center (green line) accounts for about 2.5% of the center's technical load (5% if run at twice the technical load). As is illustrated, overprovisioned fan power grows faster than overprovisioned pump power.

One ton of refrigeration will cool approximately 3,500 W of technical load; therefore, 1 MW of technical load requires a minimum of 285 tons of refrigeration. At 2.1 kW/ton, the air-cooled data center cooling infrastructure consumes about 600 kW to cool 1 MW worth of technical load. This equates to $600,000 per year per megawatt of technical load. Almost all of this energy cost can be eliminated by immersion cooling since chillers, CRAC units, and server fans are not required.

Lower capital expenses

Immersion cooling requires far less infrastructure than air cooling; therefore, building data centers dedicated to immersion cooling is substantially less expensive.

TABLE 4. Power usage of cooling equipment in air-cooled data centers

Cooling Equipment	Power Usage (kW/ton)
Chillers	0.7 kW/ton
CRAC Units	1.1 kW/ton
Server Fans	0.2 kW/ton
Total	*2.1 kW/ton*

Cooling infrastructure accounts for a major portion of data center construction costs. In high reliability/availability data centers, it is not uncommon for the cooling infrastructure to account for half of the overall construction cost. According to the American Power Conversion Data Center Capital Cost Calculator, cooling infrastructure accounts for **at least 43%** of data center construction cost.

For large data centers, where the technical load is in the neighborhood of 60 MW, construction costs can approach one billion dollars. This means that about 500 million dollars is being spent on cooling infrastructure per data center. Since immersion-cooled systems do not require chillers, CRAC units, raised flooring, and temperature and humidity controls, etc., they offer a substantial reduction in capital expenditures over air-cooled systems.

Immersion cooling FAQs

Several recurring questions have emerged over the many tours and demonstrations of the LPS immersion cooling system. Here are answers to these frequently asked questions.

Q. What server modifications are required for immersion?

There are three modifications that are typically required including:

1. Removing the cooling fans. Since some power supplies will shut down upon loss of cooling,

TABLE 3. Power usage for air-cooled versus immersion-cooled data centers with 1 MW of technical load

Method of Cooling	Fan or Pump Power as a Percentage of Technical Load (at 150% capacity)	Total Power (at 150% capacity)
Fan-Powered Air	19.5%	1.195 MW
Pump-Powered Oil Immersion	3.75%	1.0375 MW
Delta		*158 kW*

a small emulator is installed to trick the power supply into thinking the fan is still there.

2. Sealing the hard drives. This step is not required for solid-state drives or for newer sealed helium-filled drives.
3. Replacing the thermal interface paste between chips and heat spreaders with indium foil.

Some server vendors are already looking at providing immersion-ready servers which will be shipped with these modifications already made.

Q. Are there hazards associated with the oil? (e.g., fire, health, spillage)

With regard to flammability, the mineral oil is a Class IIIB liquid with a flammability rating of 1 on a scale of 4. Accordingly, immersion cooling does not require any supplemental fire suppression systems beyond what is normally used in a data center. The health effects are negligible. The oil is essentially the same as baby oil.

Spills and leaks are considered a low probability; however, for large installations, some form of spill containment is recommended. Spill decks, berms, curbs, or some other form of perimeter containment is sufficient.

Q: How much does the system weigh?

A 42U tank fully loaded with servers and oil weighs about 3,300 pounds, of which the oil accounts for about 1,700 pounds. This weight is spread over a footprint of approximately 13 square feet for a floor loading of approximately 250 pounds per square foot. A comparably loaded air-cooled server rack weighs about 1,600 pounds with a footprint of 6 square feet, which also translates to a floor loading of about 250 pounds per square foot.

Q: How is the equipment serviced or repaired?

Basic services such as device and board-level replacements are not significantly different than for air-cooled equipment. Hot-swaps can be done in the oil. For services requiring internal access, the server can be lifted out of the tank and placed on drainage rails above the surface of the oil. After the oil drains, component replacement is carried out the same way as for air-cooled servers.

For rework at the circuit board level that requires removal of the oil, there are simple methods available to ultrasonically remove oil from circuit boards and components.

Q: Are there other types of immersion-cooling systems besides oil immersion?

Yes. What this article has covered is called single-phase immersion. That is, the oil remains in the liquid phase throughout the cooling cycle. There are some people looking into two-phase immersion-cooling systems. In a two-phase cooling process, the cooling liquid is boiled off. The resulting vapor is captured and condensed before being recirculated. The phase change from liquid to gas allows for higher heat removal but adds to the complexity of the system. Also, the liquid used in two-phase systems is extremely expensive compared to mineral oil. At this time, there are no two-phase immersion-cooling systems commercially available.

Conclusion

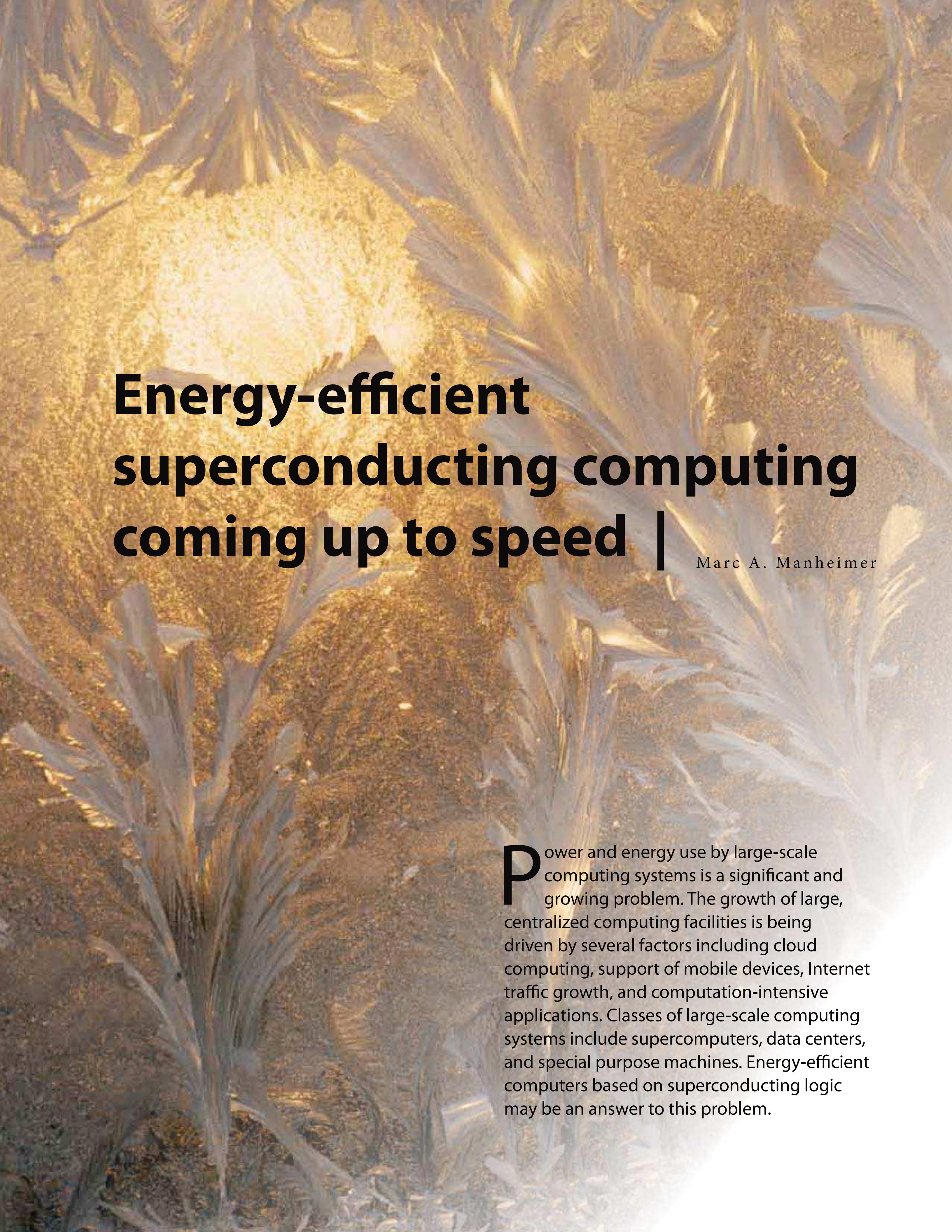
Computers consume energy and produce computation and heat. In many data centers, the energy required to remove the heat produced by the computers can be nearly the same as the energy consumed performing useful computation. Energy efficiency in the data center can therefore be improved either by making computation more energy efficient or by making heat removal more efficient.

Immersion cooling is one way to dramatically improve the energy efficiency of the heat removal process. The operating energy required for immersion cooling can be over 15% less than that of air cooling. Immersion cooling can eliminate the need for infrastructure that can account for half of the construction cost of a data center. In addition, immersion cooling can reduce server failures and is cleaner and quieter than air cooling.

Immersion cooling can enable more computation using less energy and infrastructure, and in these times of fiscal uncertainty, the path to success is all about finding ways to do more with less.

About the author

David Prucnal has been active as a Professional Engineer in the field of power engineering for over 25 years. Prior to joining NSA, he was involved with designing, building, and optimizing high-reliability data centers. He joined the Agency as a power systems engineer 10 years ago and was one of the first to recognize the power, space, and cooling problem in high-performance computing (HPC). He moved from facilities engineering to research to pursue solutions to the HPC power problem from the demand side versus the infrastructure supply side. Prucnal leads the energy efficiency thrust within the Agency's Advanced Computing Research team at the Laboratory for Physical Sciences. His current work includes power-aware data center operation and immersion cooling. He also oversees projects investigating single/few electron transistors, three-dimensional chip packaging, low-power electrical and optical interconnects, and power efficiency through enhanced data locality.



Energy-efficient superconducting computing coming up to speed

Marc A. Manheimer

Power and energy use by large-scale computing systems is a significant and growing problem. The growth of large, centralized computing facilities is being driven by several factors including cloud computing, support of mobile devices, Internet traffic growth, and computation-intensive applications. Classes of large-scale computing systems include supercomputers, data centers, and special purpose machines. Energy-efficient computers based on superconducting logic may be an answer to this problem.

Introduction

Supercomputers are also known as high-performance or high-end systems. Information about the supercomputers on the TOP500 list is readily available [1, 2]. The cumulative power demand of the TOP500 supercomputers was about 0.25 gigawatts (GW) in 2012. The Defense Advanced Research Projects Agency and the Department of Energy have both put forth efforts to improve the energy efficiency of supercomputers with the goal of reaching 1 exaflops for 20 megawatts (MW) by 2020. The flops metric (i.e., floating point operations per second) is based on Linpack, which uses double-precision floating point operations, and 1 exaflops is equivalent to 10^{18} flops.

Data centers numbered roughly 500,000 worldwide in 2011 and drew an estimated 31 GW of electric power [3–5]. Information about data centers is harder to find than that of supercomputers, as there is no comprehensive list and much of the information is not public. Exceptions include colocation data centers [6], which are available for hire and include about 5% of data centers by number, and the Open Compute Project led by Facebook [7]. Part of the Open Compute Project, Facebook's first European data center under construction in Lulea, Sweden will be three times the size of its Prineville, Oregon data center, which has been using an average of 28 MW of power [8, 9]. Facebook has been a leader in efforts to reduce power consumption in data centers and Lulea's location just below the Arctic Circle with an average temperature of 1.3°C helps with cooling, but average power usage is still expected to exceed 50 MW.

A 2010 study by Bronk et al. projected that US data center energy use would rise from 72 to 176 terawatt hours (TWh) between 2009 and 2020, assuming no constraints on energy availability [10]. The potential benefit to the US of technology that reduces energy requirements by a factor of 10 is on the order of $15 billion annually by the year 2020, assuming an energy cost of $0.10 per kilowatt hour (kWh). Note that this counts only the benefit of energy savings and does not include the potential economic benefits resulting from increased data center operation or savings due to reduced construction costs.

Conventional computing technology based on semiconductor switching devices and normal metal interconnects may not be able to increase energy efficiency fast enough to keep up with increasing demand for computing. Superconducting computing is an alternative that makes use of low temperature phenomena with potential advantages. Superconducting switches based on the Josephson effect switch quickly (i.e., ~1 picosecond), dissipate very little energy per switch (i.e., less than 10^{-19} joules), and produce small current pulses which travel along superconducting passive transmission lines at about one third the speed of light with very low loss. Superconducting computing circuits typically operate in the 4–10 kelvin temperature range.

Earlier technologies for superconducting computing were not competitive due to the lack of adequate cryogenic memory, interconnects between the cryogenic and room temperature environments capable of high data transmission rates, and fabrication capability for superconducting electronic circuits.

Superconducting computing

Recent developments in superconducting computing circuits include variants with greatly improved energy efficiency [11]. Prospects for cryogenic memories have also improved with the discovery of memory elements which combine some of the features of Josephson junctions and magnetic random access memory (MRAM). The ability to operate both logic and memory within the cold environment, rather than with the main memory out at room temperature, decreases demands on the interconnects to room temperature to the point that engineering solutions can be found.

Superconducting computers are being evaluated for potential energy efficiency benefits relative to conventional technology. The total benefit of such an energy-saving technology would scale as the number of systems multiplied by the energy savings per system.

My group at NSA's Laboratory for Physical Sciences conducted a feasibility study of a range of superconducting computer systems from petascale to exascale (10^{15}–10^{18} flops) for performance, computation efficiency, and architecture. Our results indicate that a superconducting processor might be competitive for supercomputing [11]. Figure 1 shows a conventional computer in comparison with a conceptual superconducting computer with the same computing performance, but much better energy efficiency. On the left is Jaguar, the supercomputer that held the

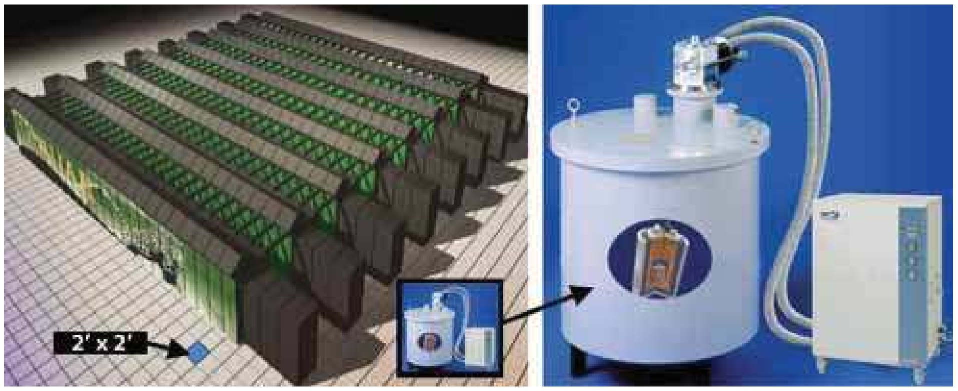


FIGURE 1. The Jaguar XT5 supercomputer at Oak Ridge National Laboratory (on left) and the conceptual superconducting supercomputer (on right) both perform at 1.76 petaflops, but the Jaguar XT5 consumes over 7 MW; whereas, the superconducting one consumes 25 kW. (Jaguar XT5 image credit: Cray Inc.)

performance record on the TOP500 list from 2009 to 2010. The conceptual superconducting supercomputer shown on the right is much smaller and uses much less power (i.e., 25 kW versus over 7 MW).

Conclusion

Superconducting computing shows promise for large-scale applications. The technologies required to build such computers are under development in the areas of memories, circuit density, computer architecture, fabrication, packaging, testing, and system integration. The Intelligence Advanced Research Projects Activity (IARPA) recently initiated the Cryogenic Computing Complexity (C3) Program with the goal of demonstrating a scalable, energy-efficient superconducting computer [12]. The results of this program should tell us if superconducting computing can live up to its promise.

About the author

Marc Manheimer is a physicist at NSA's Laboratory for Physical Sciences. His research interests include magnetic materials and devices, and cryogenic phenomena, devices, and systems. He recently became interested in superconducting computing as a solution to the power-space-cooling problem facing supercomputing. Manheimer is currently serving as the program manager for the new C3 program at IARPA.

References

[1] TOP500. Available at: http://www.top500.org.

[2] The Green500. Available at: http://www.green500.org.

[3] Koomey JG. "World-wide electricity used in data centers." *Environmental Research Letters*. 2008;3(3). doi: 101.1088/1748-9326/3/3/034008.

[4] Koomey JG, Belady C, Patterson M, Santos A, Lange K-D. "Assessing trends over time in performance, costs, and energy use for servers." 2009 Aug 17. Final report to Microsoft Corporation and Intel Corporation. Available at: http://download.intel.com/pressroom/pdf/servertrendsrelease.pdf.

[5] Koomey JG. "Growth in data center electricity use 2005 to 2010." 2011 Aug 1. Oakland, CA: Analytics Press. Available at: http://www.analyticspress.com/datacenters.html.

[6] Information on collocation data centers is available at: http://www.datacentermap.com/datacenters.html.

[7] Open Compute Project. Available at: http://www.opencompute.org.

[8] Ritter K. "Facebook data center to be built in Sweden." *The Huffington Post*. 2011 Oct 27. Available at: http://www.huffingtonpost.com/2011/10/27/facebook-data-center-sweden_n_1034780.html.

[9] McDougall D. "Facebook keeps your photos in the freezer: Arctic town now world data hub." *The Sun*. 2013 Jan 24. Available at: http://www.thesun.co.uk/sol/homepage/features/4759932/New-Facebook-data-hub-freezing-Swedish-town-Lulea.html .

[10] Bronk C, Lingamneni A, Palem K. "Innovation for sustainability in information and communication technologies (ICT)." James A. Baker III Institute for Public Policy, Rice University. 2010 Oct 26. Available at: http://bakerinstitute.org/publications/ITP-pub-Sustainabilityin ICT-102510.pdf.

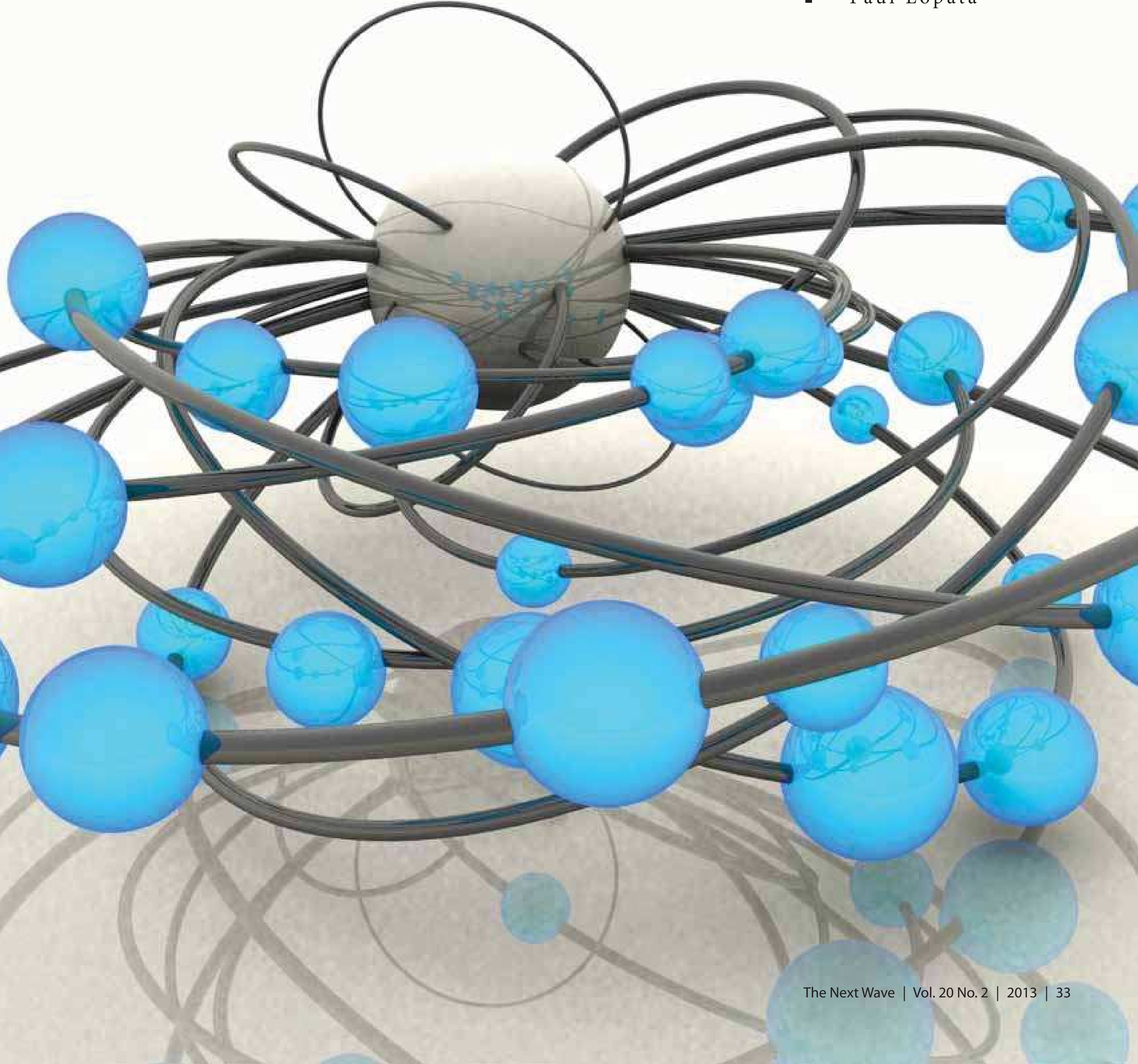
[11] Holmes DS, Ripple AL, Manheimer MA. "Energy-efficient superconducting computing—power budgets and requirements." *IEEE Transactions on Applied Superconductivity*. 2013;23(3). doi: 10.1109/TASC.2013.2244634.

[12] IARPA. Cryogenic Computing Complexity (C3) Program. Available at: http://www.iarpa.gov/Programs/sso/C3/c3.html.

Beyond digital

A brief introduction to quantum computing |

Paul Lopata



Introduction

Computers are based on logic. These fundamental rules of logic dictate the types of problems that can be solved on a computing machine. These rules of logic also determine the resources required to complete a calculation. From the early years of computing machines to the present, the most successful computer designs have utilized two-level digital logic. The amazing success of modern-day computing technology is based on the algorithmic strengths of digital logic paired with the stunning technological advances of silicon complementary metal-oxide semiconductor (CMOS) chip technology. Processor chips and memory chips built out of silicon CMOS technology have provided a continually improving platform on which to perform digital logic.

Despite the well-known successes of computing machines based on digital logic, some algorithms continue to be difficult to perform—and some problems are intractable not only on existing machines but on any practical digital-logic machine in the foreseeable future! These intractable problems serve as both a curse and a blessing: A curse because solutions to many of these intractable problems have significant scientific and practical interest. A blessing because the computational difficulty of these intractable problems can serve as a safeguard for secure data storage and secure data transmission through the use of modern encryption schemes.

It is clear that the only algorithmic way to solve these intractable problems is to utilize a computing machine that is based on something other than standard digital logic.

One such path toward developing a "beyond-digital logic" machine is in the field of quantum computing. Quantum computing is still in the early stages of its development, and most of its advances are being reported from universities and basic research labs. Three major insights have led to the current understanding that quantum computing technology may have a significant potential for solving some of these algorithmically intractable problems:

1. Specific algorithms have been developed to solve mathematical problems on a (yet-to-be-developed) quantum computer that are otherwise intractable using standard digital logic;
2. Physical systems exist that can be used as the basic building blocks for a machine to implement these quantum algorithms;
3. There are ways to effectively handle errors that will inevitably occur when running an algorithm on one of these quantum computing machines.

This article introduces quantum computing through a discussion of these three insights and the technical literature that underpins this exciting and fast-moving field.

Quantum algorithm discoveries

When discussing the speed of an algorithm, it is useful to break the algorithm down to a basic set of steps, or gate operations, that can be repeated over and over again to complete the calculation. Once an algorithm has been written down in terms of a fixed-gate set, all that remains is counting up the number of gates required for a particular problem size to determine how many resources are required to finish the calculation. When a problem is said to be intractable, it is because the number of gates required to complete the calculation is so overwhelmingly large that the algorithm will not finish in a practical amount of time.

The first quantum algorithm discovered to have a speedup over algorithms based on digital logic came from David Deutsch in the first of a series of two papers in the *Proceedings of the Royal Society of London A* (from 1985 and 1989). The algorithm Deutsch devised to demonstrate this speedup over digital logic is something of a toy problem—it involves two narrowly defined classes of functions and tries to determine whether a function falls into one or the other of these two classes. While this toy problem has extremely limited practical interest, it was very useful in demonstrating that there is potential for a quantum computer, based on its beyond-digital logic, to solve problems faster than computers based on digital logic. This algorithmic discovery, along with the quantum circuit formalism spelled out by David Deutsch, spurred on further algorithm development.

Whereas Deutch's algorithm had extremely limited utility beyond a first demonstration, an algorithm later developed by Peter Shor proved to be of more widespread interest. What became known as Shor's Algorithm provides a speedup for finding the unique prime factors of a number—a problem of historic interest that gets extremely difficult as the number to be factored gets larger and larger. Shor's Algorithm for factoring remains one of the best-known examples of a seemingly intractable problem that is potentially solved using a quantum computer. Many other quantum algorithms have been invented, each for tackling some difficult mathematics problem. (See the further reading section for further details on the Deutsch algorithm and Shor's Algorithm, as well as details on the many other algorithmic discoveries and their advantages.)

It must be noted that not all algorithms achieve a speedup when tackling the problem with a quantum computer. That is, a quantum computer will provide an improvement on solving *some* problems, but will not provide an improvement on solving *all* problems. As with the other aspects of quantum computing described below, quantum algorithm development remains a vigorous open field of investigation.

Physical implementations of a quantum computer

Building and operating devices to implement the beyond-digital logic of quantum computing has been the focus of intense effort since the early quantum algorithm discoveries. A great deal of progress has been made in several different technologies toward these goals, with many impressive early demonstrations. This includes demonstrating some small algorithms with a handful of logic operations.

Demonstrating the basics of beyond-digital logic requires exquisite control over the tiniest parts of a physical system. At this small scale, the behavior of these systems is described by the laws of quantum physics. By utilizing a system governed by the laws of quantum physics, beyond-digital logic becomes possible.

Exquisite control is needed to prevent the introduction of damaging noise into the system during the control process because as noise is introduced the rules of quantum physics that describe the behavior of these small systems get washed out. (This is, in some sense, why the broader world around us is seen to obey the everyday rules of classical physics rather than quantum rules that dominate the behavior of very small systems. The jostling of the many small systems against one another contributes to the overall noise that washes out the quantum effects at a large scale.) The term *coherence time* is used in the field of quantum computing to describe how long the regular behavior of a quantum system survives before an irreversible connection to the outside world sets in and the quantum effects required for beyond-digital logic are washed out.

The first step in operating on a system capable of going beyond digital logic is to identify a suitable small subsystem that is isolated enough to have a long coherence time but, at the same time, can be fully controlled without introducing too much extra noise. These contradictory system requirements—isolation (for a long coherence time) and connection to the outside world (for full control)—make the demonstration of beyond-digital logic such a challenge. (See the further reading section for more details on additional requirements on physical systems to perform quantum logic.)

Several physical systems have been used for early demonstrations of beyond-digital quantum logic. These include the following:

- Optical and microwave operations on the electronic and motional states of ionized atoms trapped in radio-frequency electric traps,
- Microwave operations on superconducting resonator circuits,
- Microwave and direct-current operations on the spin of a single electron isolated in a semiconductor,
- Linear and nonlinear optical operations on single photons,
- Optical operations on electrons in quantum dots grown into semiconductors, and
- Nuclear magnetic resonance operations on various states of a molecular ensemble.

Each of these technologies has different setup, control, and measurement techniques. Furthermore, each technology is at a different level of development, and the outlooks for future development vary wildly between technologies. While impressive strides have been made, no technology has successfully implemented more than a handful of quantum logic operations before succumbing to its limited coherence time. (See the further reading section for information about recent review articles in *Science Magazine* that describe several of these technologies in more detail.)

Dealing with errors in a quantum machine

Every complex machine demonstrates unexpected behavior. The challenge for an engineer designing any computing machine is to design it so that the final answer at the end of an algorithm will not be wrong, even if errors creep in along the way.

The beyond-digital logic of quantum algorithms requires the data to remain isolated from external disturbances through the course of a calculation. This requirement imposes a difficult restriction on any error protection scheme that is implemented on a quantum computing machine: How do you check for errors in a way that does not impose a disturbance on the system that is too great to allow for the beyond-digital logic to be performed?

The key insight into this problem is to couple the small subsystem being used to perform the calculation to another small subsystem that also is capable of performing beyond-digital quantum logic. These two subsystems together can be used to cleverly encode the data for the algorithm so that tests can be performed on the second subsystem to check for errors on the original subsystem. And, if done correctly, these tests on the second subsystem will not disturb the original subsystem too much. Furthermore, if an error is detected, there are ways to correct this error on the original subsystem to allow the algorithm to finish without corrupting the final result.

For this quantum error protection protocol to work: 1) the two subsystems must be encoded so that the data remains intact while encoded, and 2) a subroutine algorithm to perform on these two subsystems must be devised that will robustly correct errors on the original subsystem—even if an error occurs while running this subroutine.

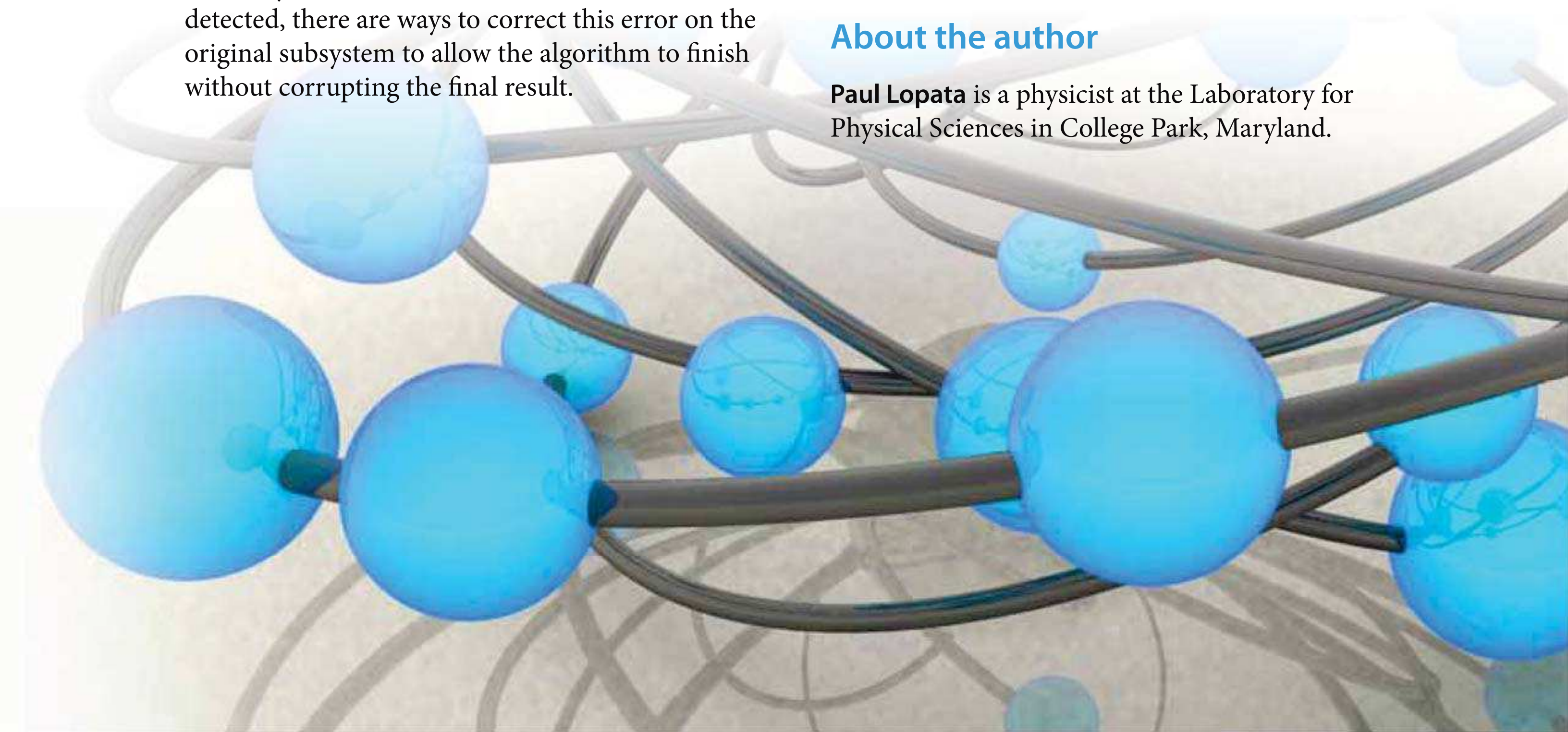
Several different schemes have been developed that accomplish these two requirements of quantum error protection, some of which are the most interesting and elegant results within the field of quantum computing. A serious challenge is the significant overhead required for encoding the data and performing these subroutines. There is also typically a very low threshold in error rates required before these schemes become effective. No experimental groups have yet demonstrated a quantum computing system of sufficient size and quality that can successfully demonstrate the full power of these quantum error protection schemes. Research continues to develop encodings that fix more errors while requiring fewer resources.

Conclusion

The rapid growth in the field of quantum computing has been a result of three key insights: discoveries of novel quantum algorithms based on beyond-digital logic, demonstration of physical systems capable of implementing beyond-digital logic, and discovery of quantum error correction. And the field of quantum computing based on its beyond-digital logic remains a fast-moving and exciting field of study.

About the author

Paul Lopata is a physicist at the Laboratory for Physical Sciences in College Park, Maryland.



Further reading

Introduction to quantum computing

- Nielsen MA , Chuang IL. *Quantum Computation and Quantum Information.* Cambridge (UK): Cambridge University Press; 2000. ISBN-13: 978-0521635035.

  This is a comprehensive and classic reference in quantum computing. It includes an introduction to the mathematics involved, algorithms, error correction, and other topics in quantum information theory. Chapter 7 on physical realizations is out of date, but the book clearly lays out the physical requirements needed for operating beyond-digital logic on a physical system.

- Mermin ND. *Quantum Computer Science.* New York: Cambridge University Press; 2007. ISBN-13: 978-0521876582.

  This is a readable, high-quality introduction and reference. It is not as comprehensive as Nielsen and Chuang, but the choice of topics is well considered.

- Kitaev AY, Shen AH, Vyali MN. *Classical and Quantum Computation.* Providence (RI): American Mathematical Society; 2002.

  This is another nice introduction to major results in the field of quantum computing. More mathematical sophistication is expected from the reader.

Algorithmic developments

- All three books in the Introduction to quantum computing section of this list contain introductions to quantum algorithms.
- Jordan S. Quantum Algorithm Zoo [updated 2013 May 23]. Available at: http://math.nist.gov/quantum/zoo/.

  Stephen Jordan at the National Institute of Standards and Technology maintains a comprehensive online catalog of quantum algorithms. This useful resource includes original references along with descriptions of the algorithms.

Experimental progress

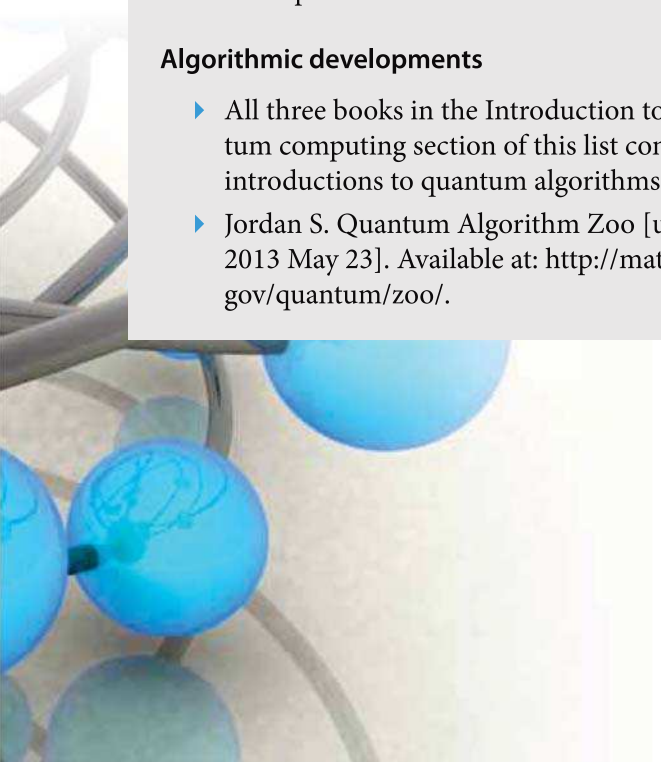
- Special feature: Quantum information processing. *Science Magazine.* 2013;339(6124):1163–84.

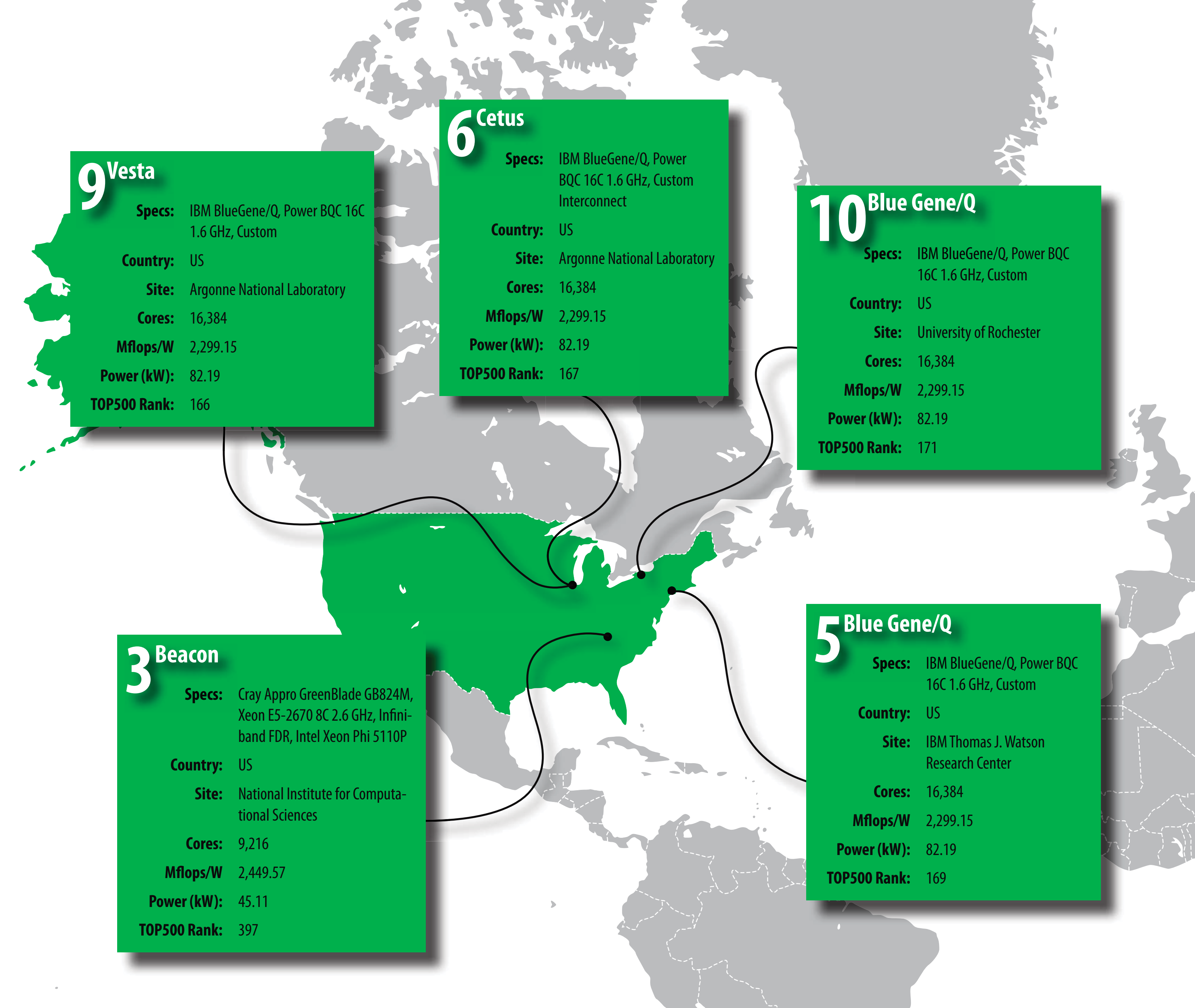
  This recent special section in *Science Magazine* covers several technologies in the field of experimental quantum computing. It includes review articles on ion traps, superconducting circuits, spins in semiconductors, and topological quantum computation. All of the articles are written by leaders in their respective subfield and include brief insights into the history, current state of art, and outlook on future developments.

Quantum error correction

- All three books in the Introduction to quantum computing section of this list contain introductions to quantum error correction.
- Gaitan F. *Quantum Error Correction and Fault Tolerant Quantum Computing.* Boca Raton (FL): CRC Press; 2008. ISBN: 978-0-8493-7199-8.

  This book provides a comprehensive discussion of many of the major results in the field, with a focus on stabilizer codes and their fault tolerant operation.





9 Vesta

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom
Country:	US
Site:	Argonne National Laboratory
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	166

6 Cetus

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom Interconnect
Country:	US
Site:	Argonne National Laboratory
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	167

10 Blue Gene/Q

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom
Country:	US
Site:	University of Rochester
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	171

3 Beacon

Specs:	Cray Appro GreenBlade GB824M, Xeon E5-2670 8C 2.6 GHz, Infiniband FDR, Intel Xeon Phi 5110P
Country:	US
Site:	National Institute for Computational Sciences
Cores:	9,216
Mflops/W	2,449.57
Power (kW):	45.11
TOP500 Rank:	397

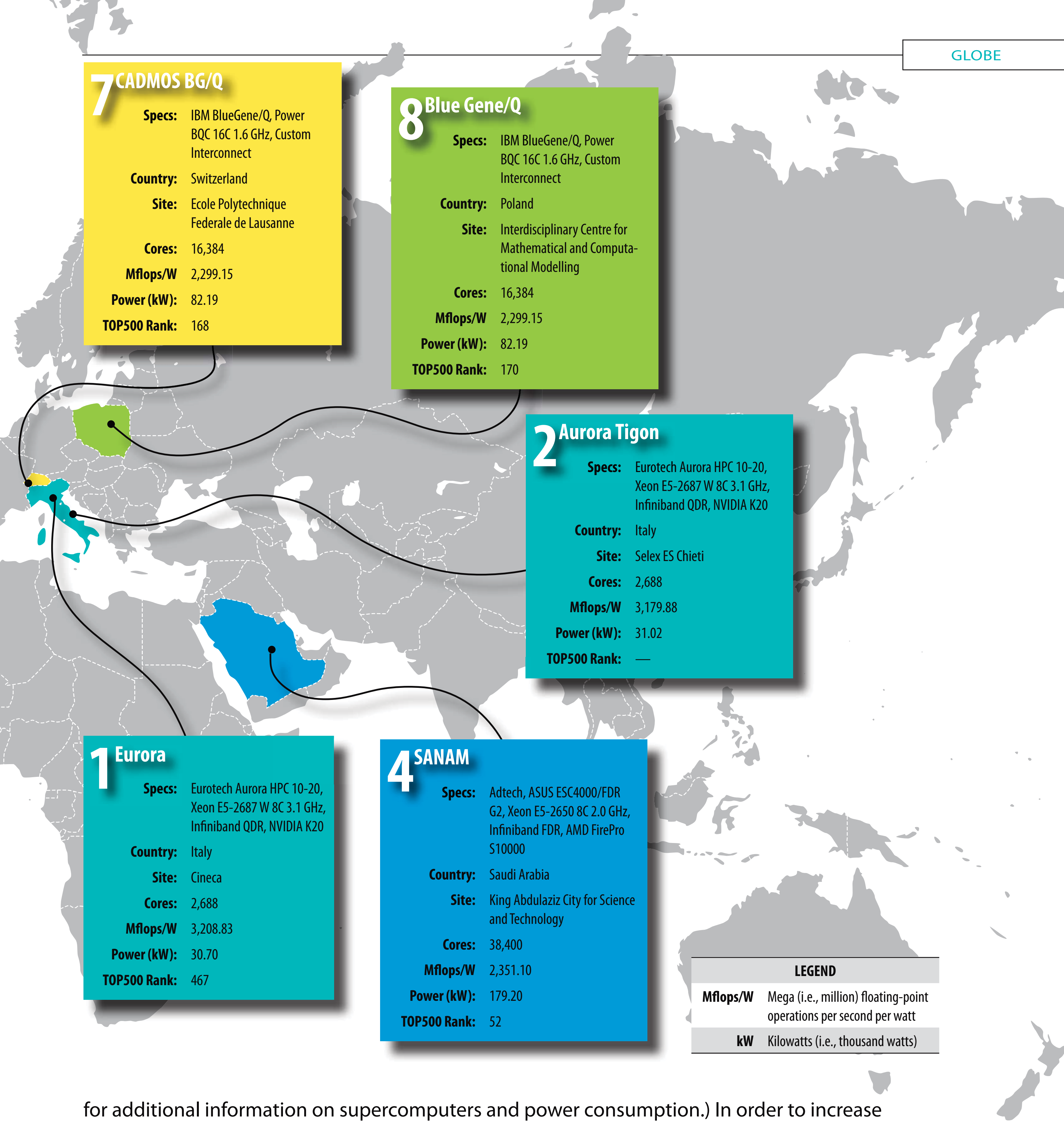
5 Blue Gene/Q

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom
Country:	US
Site:	IBM Thomas J. Watson Research Center
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	169

GLOBE AT A GLANCE

The Green500 top 10 supercomputers

The Green500 provides a ranking of the most energy-efficient supercomputers in the world. For decades, supercomputer performance has been synonymous with speed as measured in floating-point operations per second. This particular focus has led to supercomputers that consume enormous amounts of power and require complex cooling facilities to operate. The rising cost of power consumption has caused an extraordinary increase in the total cost of ownership of a supercomputer. (See "Doing more with less: Cooling supercomputers with oil pays off"



7 CADMOS BG/Q

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom Interconnect
Country:	Switzerland
Site:	Ecole Polytechnique Federale de Lausanne
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	168

8 Blue Gene/Q

Specs:	IBM BlueGene/Q, Power BQC 16C 1.6 GHz, Custom Interconnect
Country:	Poland
Site:	Interdisciplinary Centre for Mathematical and Computational Modelling
Cores:	16,384
Mflops/W	2,299.15
Power (kW):	82.19
TOP500 Rank:	170

2 Aurora Tigon

Specs:	Eurotech Aurora HPC 10-20, Xeon E5-2687 W 8C 3.1 GHz, Infiniband QDR, NVIDIA K20
Country:	Italy
Site:	Selex ES Chieti
Cores:	2,688
Mflops/W	3,179.88
Power (kW):	31.02
TOP500 Rank:	—

1 Eurora

Specs:	Eurotech Aurora HPC 10-20, Xeon E5-2687 W 8C 3.1 GHz, Infiniband QDR, NVIDIA K20
Country:	Italy
Site:	Cineca
Cores:	2,688
Mflops/W	3,208.83
Power (kW):	30.70
TOP500 Rank:	467

4 SANAM

Specs:	Adtech, ASUS ESC4000/FDR G2, Xeon E5-2650 8C 2.0 GHz, Infiniband FDR, AMD FirePro S10000
Country:	Saudi Arabia
Site:	King Abdulaziz City for Science and Technology
Cores:	38,400
Mflops/W	2,351.10
Power (kW):	179.20
TOP500 Rank:	52

LEGEND

Mflops/W	Mega (i.e., million) floating-point operations per second per watt
kW	Kilowatts (i.e., thousand watts)

for additional information on supercomputers and power consumption.) In order to increase awareness and use of supercomputer performance metrics based on efficiency and reliability, the Green500 list puts a premium on energy-efficient performance for sustainable supercomputing. The following ranking is from June 2013; the list in its entirety as well as information about the measurement methodology is available at www.green500.org.

GREEN SUPERCOMPUTERS

The Green500 announces the most energy-efficient supercomputers of June 2013

The Green500 list of June 2013 is dominated by heterogeneous supercomputers—those that combine two or more types of processing elements together, such as a traditional central processing unit (CPU) combined with a graphical processing unit (GPU) or a coprocessor.

Eurotech manufactured the top two supercomputers on the list—Eurora and Aurora Tigon. Eurora, located in Italy at Cineca, performs at 3.21 gigaflops per watt, while Aura Tigon, located in Italy at Selex ES Chieti, performs at 3.18 gigaflops per watt. These supercomputers are nearly 30% more energy efficient than the previous top supercomputer on the Green500 list. The fastest supercomputer of June 2013—Tianhe-2—performed at 1.9 gigaflops per watt, placing it in the number 32 spot on the Green500 list.

"Overall, the performance of machines on the Green500 List has increased at a higher rate than their power consumption. That's why the machines' efficiencies are going up," says Wu Feng, founder of the Green500. For machines built with off-the-shelf components, a great deal of their efficiency gains can be attributed to heterogeneous designs; such a design allows these systems to keep pace and in some cases even outpace custom systems (e.g., IBM's Blue Gene/Q).

"While the gains at the top end of the Green500 appear impressive, overall the improvements have been much more modest," says Feng (see figure 1). "This clearly indicates that there is still work to be done."

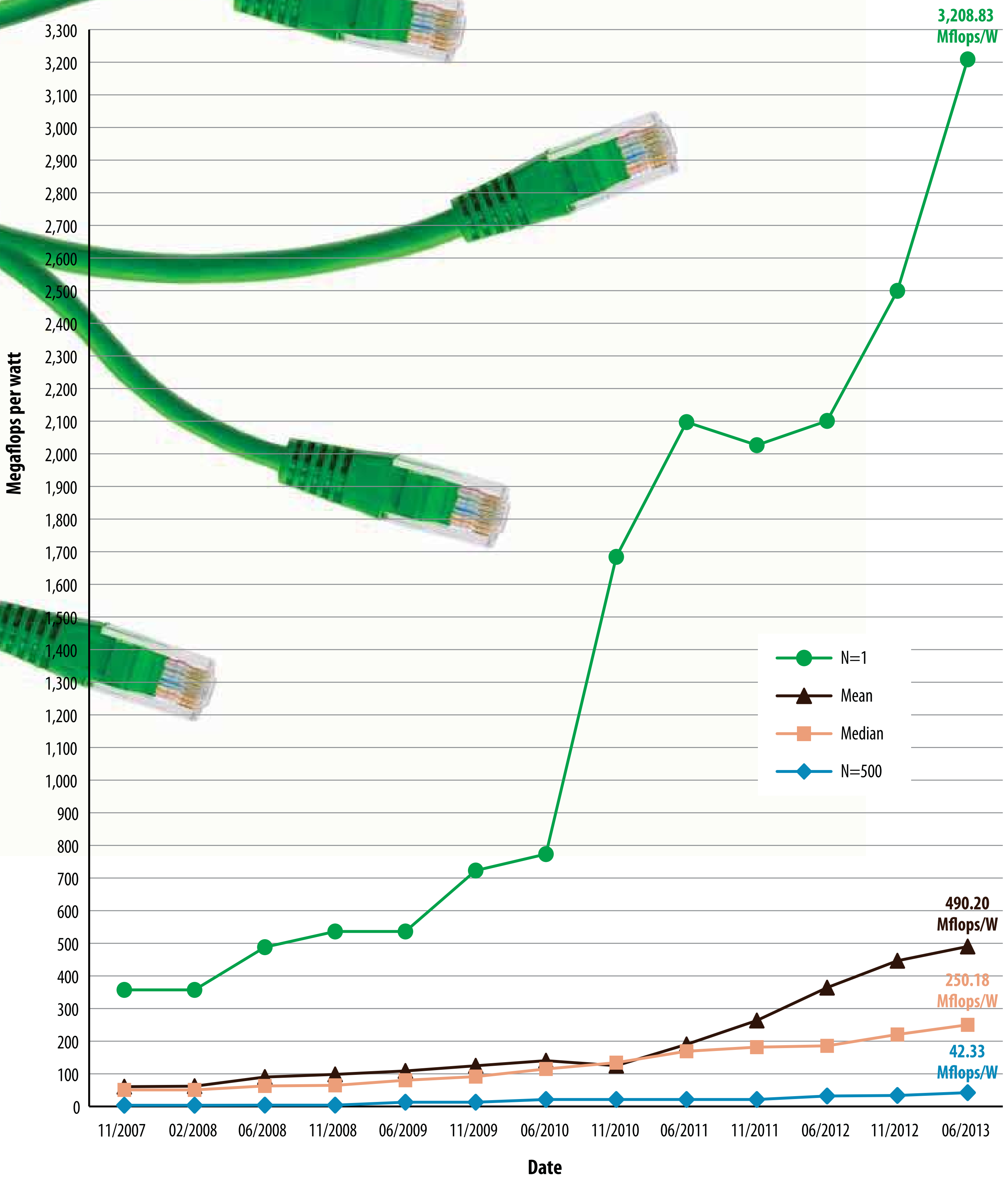


FIGURE 1. The energy efficiency of the highest-ranked supercomputers on the Green500 list (green circles) has been improving much more rapidly than the mean (brown triangles) and the median (pink squares). For instance, while the energy efficiency of the greenest supercomputer improved by nearly 30%, the median improved by only about 14%, and the mean by only about 11%.

Novel methods for manufacturing photonic logic devices

Traditional integrated electronic components have not kept pace with the performance demands of applications such as high-speed encryption, video on demand, and broadband television. Because optoelectronics (integrated photonics) promise to deliver greater speed and bandwidth than their electronic counterparts, some experts anticipate that they could one day make traditional electronic semiconductors obsolete. The widespread adoption of optical switching has increased pressure on industry to create fully photonic components to replace traditional electronic devices.

Even though research on photonic logic devices has been ongoing for several years, an integrated photonic device that could rival today's integrated electronic circuit does not yet exist. In particular, the ability to easily manufacture laser based devices and waveguides at the microcircuit level has presented challenges in the wafer fabrication stage. One specific issue has been the ability to develop optical interfaces, such as laser to waveguide, that do not have impedance mismatches.

NSA engineers within the Trusted Systems Research group in the Research Directorate took on the challenge. Their research resulted in methods to precisely manufacture photonics devices that use air gaps to tune the reflectance between optical devices (e.g., figure 1). These air gaps are formed by making a wafer mask with very precise regions that allow the deposition of sacrificial material onto the wafer forming spacers. This material is removed later by chemical etching processes. Engineers can now adjust the reflectance by varying the sacrificial spacer layers.

Another challenge facing photonic device developers is the specialized equipment required to manufacture sacrificial layers within a wafer. Working with the Laboratory for Physical Sciences (LPS), NSA engineers were able to develop methods of producing photonic devices using standard wafer manufacturing equipment such as Plasma Enhanced Chemical Vapor Deposition (PECVD) and later Biased Target Ion Beam Deposition (BTIBD). These novel methods opened up the potential for even more advanced devices since custom or highly specialized manufacturing equipment is not required. Another key to this technology is LPS's Projection Lithography Stepper tool (see figure 2) which projects the circuit image onto the wafer.

In late 2011, NSA's Technology Transfer Program (TTP) licensed 16 patented photonics manufacturing methods to industry. This technology transfer was one of the largest bundled patent deals in the history of TTP and reemphasized the commitment of NSA to return taxpayer-funded research and technology back to private industry.

FIGURE 1. An image of a fabricated mode transition-discrimination (MTD) photonic logic device with semiconductor laser edged facets and etched waveguide trenches.

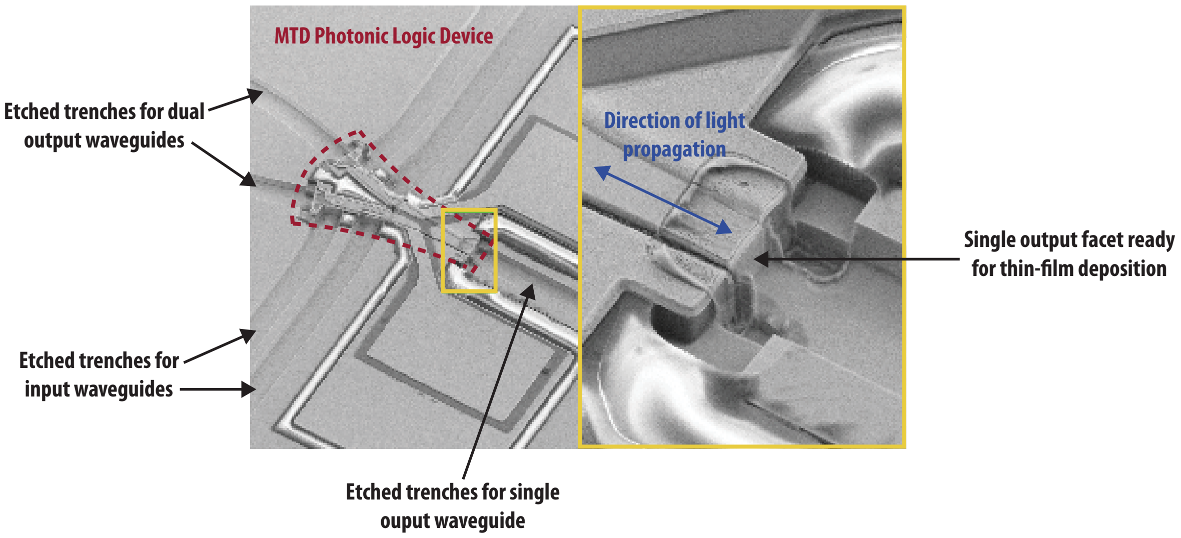
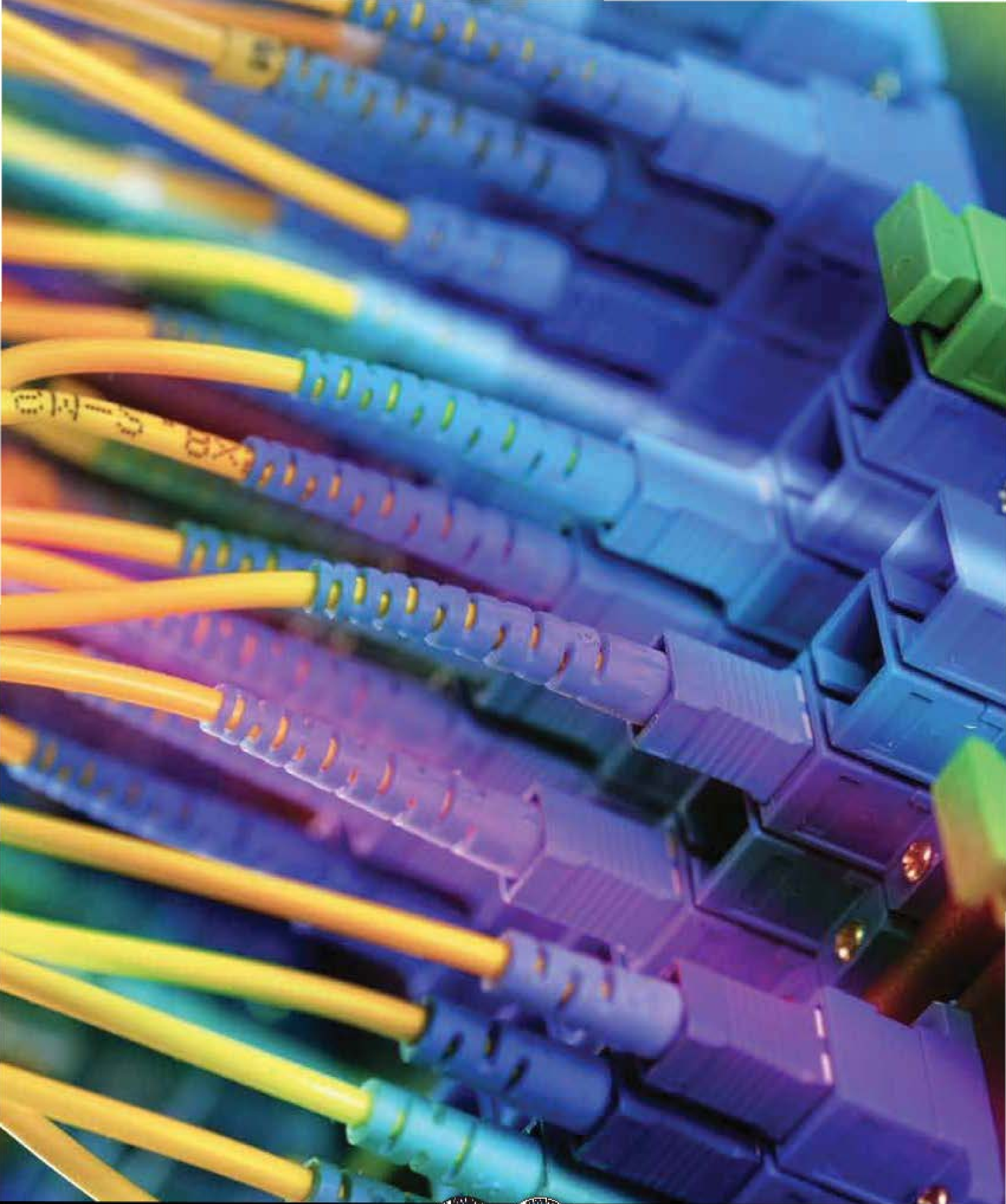

MTD Photonic Logic Device
Etched trenches for dual output waveguides
Etched trenches for input waveguides
Etched trenches for single ouput waveguide
Direction of light propagation
Single output facet ready for thin-film deposition

FIGURE 2. LPS's Projection Lithography Stepper tool.



National Security Agency Central Security Service

Defending Our Nation. Securing The Future