Embedded Systems and Exascale Computing

What do the architectures of a future exascale computing system and a future battery-operated embedded system have in common? At first glance, their requirements and challenges seem unrelated. However, discussions and collaboration on the projects revealed not only similar requirements, but many common power and packaging issues as well.

Developers of exa-sized computing systems are concerned with their unrealistically high projections for power consumption circa 2015. Implementation and operating costs of these large systems are an important concern. Also, many problems won't scale to billions of processors; therefore individual compute node performance must be increased. Such systems also need smaller packages to reduce communication and networking costs.

Developers of embedded systems often face high-performance computing (HPC) and battery power requirements. Near-future applications will require hundreds of gigaops in the short term, and teraops not long after. These applications are expected to be in small, nearly invisible body-worn packages. And, of course, commercial competition is driving the cost of these devices lower and lower.

Although the two application domains are quite different, as we discuss here, we found that both consider the same approaches to address power, performance, packaging, and cost concerns.

1521-9615/10/$26.00 © 2010 IEEE
Copublished by the IEEE CS and the AIP

David W. Jensen
Rockwell Collins
Arun F. Rodrigues
Sandia National Laboratories

Exascale and Terascale Computing

Exascale computers perform a million-trillion (10^{18}) calculations per second. The world's fastest supercomputers today recently achieved a petaflops, or a thousand-trillion (10^{15}) point operations per second. With the HPC community's goal of reaching an exascale by 2020, researchers must improve performance by three orders of magnitude in a short time. To do so, they must confront challenges in power, packaging, efficiency, and communication.

The recent DARPA-sponsored *Exascale Computing Study*[1] envisions a 2015 system with three categories of exascale systems:

- exa-sized data center systems,
- peta-sized departmental computing systems, and
- tera-sized embedded systems.

Here, we describe progress on the high and low end of these categories. One of us (Rodrigues) is researching the architecture for a future exa-sized computing system; the other (Jensen) is designing the architecture for a future battery-operated tera-sized embedded system. Our collaboration revealed that our projects share similar requirements and challenges. To achieve terascale performance, embedded systems must borrow from high-end HPC systems; likewise, to achieve exascale performance, high-end HPC systems must borrow from embedded systems.[1]

An embedded computer system typically performs a few dedicated functions with real-time operating constraints, often with unique hardware, interface, and mechanical requirements. The limitations make it possible to optimize the embedded computer and system to improve the size, cost, reliability, power, and performance. The total manufacturing volume of embedded system electronics today dwarfs the volume for personal computers.

Embedded microprocessors are used in everything from one-dollar toys to desktop computers. In recent years, the explosion of portable battery-powered applications levied significant energy-efficiency requirements on embedded processors. Historically, embedded systems haven't needed supercomputer performance, but increased application sophistication mandates more processing performance. Image processing is becoming ubiquitous in handheld embedded systems, and video processing dramatically increases the performance requirements.

We've identified multiple military and commercial applications that could use hundreds to thousands of gigaops if such performance were available in a low-power, easily portable system. Moore's law and miniaturization trends predict these capabilities won't be available for at least a decade; the recent focus on exascale systems and our own research suggests that this availability could come much sooner.

Exascale Challenges

Exascale computing faces many challenges, including power, cost, reliability, programmability, and performance. The most significant challenge today for exa-sized computers is meeting power and energy requirements. With computing, networking, cooling, and infrastructure power, total power demand would be hundreds of megawatts—that is, a significant part of an entire power plant's output—using today's technologies.

The cost of powering and cooling a machine is also a major factor in its overall cost. At a rate of six to 12 cents per kW-hr, a megawatt-year of electricity costs roughly US$500,000 to $1 million. Thus, machines with projected power consumptions in the tens or hundreds of megawatts will have electricity costs that rival their purchase cost every year. This would have a massive impact on global power consumption. Already server power consumption is estimated to consume roughly 1 percent of the world's power usage, and is growing at a rate of 11 percent—far faster than other power consumers.[2]

Exa-sized systems will use millions of processors that must be physically distributed across a hierarchy of systems ranging from chips to modules, boards, racks, systems, buildings, and (potentially) states. Although thousands of these processors might be available to work on a problem, until the data or assignment is distributed to the processors, they can't perform the work. Communication and maintaining data locality will be critical for these systems, as accessing local data will be orders of magnitude faster than accessing remote data.

Any complex system entails reliability and resiliency issues. Individual processors work for years without failure, but the mean time between failures (MTBF) for millions of processors could be only a few minutes. Many conventional redundancy approaches dramatically add costs or degrade performance.

Finally, exascale systems all have concerns regarding development, integration, and recurring expenses. The broad set of exascale system challenges implies multiple years of research and development. Integrating millions of processors will require large capital investments. The power, maintenance, and cooling for such systems could cost many millions of dollars.

The broad set of exascale system challenges implies multiple years of research and development.

The Gap

Currently, there's no clear roadmap to power-efficient exascale computing. If we look at a future supercomputer as being composed of a processor, memory system, and interconnect, we see that each of these areas requires major improvements if we're to build an exascale computer with reasonable power consumption.

If we extrapolate the current double-data-rate-based dynamic RAM technology to the 2016–2018 time frame, DRAM access would consume more than 31 picojoules (pJ) per bit.[1] A reasonable exascale memory system of 48 petabytes with 500-petabytes/second bandwidth would then consume more than 125 MW of power. Adding in another 25 MW for cooling, our sample memory system alone would cost $75 million per year to operate. This is similar to the purchase price of a large supercomputer today.

Similarly, off-chip electrical interconnect power is projected to be in the 4 to 8 pJ/bit range

Table 1. Embedded processor energy efficiency.

Processor	Performance (million instructions per second)	Power	MIPS/Watt	Picojoules per operation
General purpose processor (GPP)	20,000	90 W	222	4,500
Embedded GPP	1,000	2.5 W	400	2,500
Research GPP multicore	64,000	30 W	2,200	450
Low-power embedded processor	400	90 mW	4,400	225
Digital signal processor (DSP)	1,200	100 mW	12,000	83
Research multicore DSP	50,000	800 mW	64,000	11
Rockwell Collins (RCI) MicroCore	1,000	4.0 mW	250,000	4

(including routing). Assuming a 500 petabytes/second aggregate bandwidth, this would result in 24-MW consumption for the interconnect bandwidth. Adding in cooling would increase operating costs by another $15 million per year.

Table 1 shows a compilation of embedded and conventional processing solutions, with their key metric of pJ per operation (pJ/op). Modern processors require thousands of pJ for every operation executed. Low-power embedded processors use an order of magnitude less energy for each operation. The DARPA-funded research multicore using general-purpose processors apparently uses hundreds of pJ for each operation. Another DARPA-funded multicore using digital signal processors (DSPs) is targeted to operate at an order of magnitude lower and use only 11 pJ per operation. The table also includes early results from our Rockwell Collins (RCI) MicroCore processor. The MicroCore processor (with minimal memory and I/O) uses only four pJ per operation.

Assuming Moore's law will provide an "automatic" factor of 16 improvement between current processors and those of 2018, we can expect traditional server and HPC processors to consume between 150 and 280 pJ/op. Floating point operations, traditionally important in HPC, might add an additional seven to 11 pJ. If we assume our exaop-per-second machine is derived from conventional processors, it will consume 150–300 MW for processing. With cooling factored in, this amounts to $90 to $180 million per year in operating costs.

Putting these pieces together, if we assume an exascale computer is built using "business as usual" techniques of DDR-based memory, conventional server-derived processors, and electrical interconnect, we could reasonably expect a power bill in the range of $250 to $400 million per year. It's doubtful that the HPC community could justify nearly a half-billion dollars a year to keep a single machine powered on and kept from melting.

Clearly, HPC must depart from business as usual to reach the exascale. Luckily for HPC, there's a massive technology community that has been extremely power-conscious and unconstrained by business as usual—that is, the embedded-computing community. While HPC has been largely content to build machines based on commodity server parts, the embedded world has traditionally used a more diverse set of tools, techniques, and components to meet stringent power requirements.

DARPA's *Exascale Computing Study* suggests that the tera-size exascale embedded system will be based on today's exascale supercomputers. The entire supercomputer would be reduced to a few chips and a few tens of watts. The DARPA study recognizes that a terascale chip set for embedded applications would be relevant to US Department of Defense missions.[1]

As we now describe, both existing and planned techniques and technologies could make this tera-size exascale embedded system possible and form the building block for exascale HPC systems.

Techniques and Technologies

As we discovered, our teams are using a common set of techniques and technologies to address HPC and embedded power and packaging issues. Both groups are considering methods to simplify the processor to reduce its energy consumption. A side benefit of simplification is that it reduces the processor's size and cost. Both groups recognize that migrating key software algorithms to a hardware implementation provides significant performance improvements and power reductions. Embedded and supercomputing applications tend to use sets of standard, well-understood routines that lend themselves to this hardware–software codesign process. Both groups have also identified parallelism as critical to attaining power and performance goals. Finally, both teams determined that data movement and access is a bottleneck that system design must address.

Simplify Processors and Processing

In HPC applications, the processors typically result from a historical path that emphasizes backward compatibility, single-thread performance, and commonality with the desktop and laptop PC market. This lets them benefit from the economies of scale in the commodity PC market. However, what's best for a laptop computer optimized for word processing isn't always best for a supercomputer. Legacy instruction sets (such as x86) add significant complexity to hide memory hierarchies and to maximize single-thread performance. This complexity adds significant power overhead.

To reach exascale, we must rethink how processing is done and relentlessly simplify existing designs for maximum efficiency.

Simple microprocessor engine. Rockwell Collins has been making microprocessors for embedded systems for more than 30 years. When possible, we typically use commercial microprocessors to exploit the available tools, libraries, and knowledge pool. When customer requirements can't be met with conventional microprocessors, we build a proprietary solution. Recently, we were presented with a requirement to provide hundreds of gigaops with 1 watt of power. We already knew that our family of proprietary processors was energy efficient, but we were surprised when our analysis showed that it provides world-class performance.

Our MicroCore microprocessor is based on our advanced-architecture microprocessors (AAMP) family. The MicroCore architecture is simple and small, resulting in intrinsic energy efficiency and computing performance. The basic 16-bit MicroCore uses approximately 10,000 gates, 32-bit on-chip microprogram control storage, and 32 Kbits of RAM. We often include a basic 1,000-word local RAM that's extensible to 64,000 words. The base design nominally requires 0.12 square millimeters of chip area in a 90-nanometer complimentary metal-oxide semiconductor (CMOS) application-specific integrated circuit (ASIC) process. The design is easily extensible with multiple arithmetic logic units, multipliers, custom operations, and larger memories.

The MicroCore is radically simpler and more energy efficient than most existing cores. Its die size is two to three orders of magnitude smaller than a conventional general-purpose processor (GPP) server core. At an estimated 4 milliwatts, it uses four to five orders of magnitude less power than a GPP. Analysis of the power and performance of the MicroCore microprocessor using commercial ASIC analysis tools shows good correlation of our instruction and memory power values to historic results and to other low-power microprocessor projects.[3] We achieve this through a simple design and a philosophy of seeking total application efficiency rather than simple performance.

Fine-grain parallelism (microcode). Microcoded machine architectures have been implemented in many computer systems and microprocessor designs over the past 50 years, including the Rockwell Collins AAMP microprocessors. The computing unit executes microcode instructions from a stored microprogram, often from ROM or loaded from external memory. MicroCore's native execution language is fully decoded microcode, which enables implementation of efficient, fine-grained logic and control. Although these instructions are more primitive than assembly-level programming, microcode gives the programmer and tools access to hardware signals controlling the design's busses, registers, and other circuit elements.

Each line of microcode has full data manipulation, arithmetic, and conditional branching capability. This approach implements explicit fine-grain parallelism; in our observations, each microcoded instruction is comparable to two or three reduced-instruction-set computing (RISC) operations.

Table 2 shows some results from an implementation of a JPEG2000 compression algorithm. This MicroCore version used multiple processors and included hardware acceleration for several algorithms in the JPEG2000 software. With hardware coprocessing and with the microcode efficiency, each line of microcode executes the equivalent of more than 14 RISC operations. The power reduction is substantial using the simple processor and hardware acceleration.

To minimize the microprocessor complexity and power, we don't provide hardware support for a shared address space. This forces the software developer (or tools) to define and maintain the data locality and coherence, simplifying the architecture and moving complexity to the programmer or compiler. Microcode programming is mainly a manual operation today; however, microcode compilers do exist. It's possible to combine the manual programming expertise and

Table 2. JPEG2000 video compression performance comparison with an image frame size of 1,600 × 1,200 pixels and a video frame rate of 60 Hz.

	Instructions (clocks per pixels)	Energy per instruction (picojoules)	Clock speed (MHz)	Frame time (seconds)	Processors	Frame power (joules per frame)	Real-time power (watts)	Power reduction (scale)
GPP	1,123	4,500	3,000	.7187	43.12	9.70	582.16	1
Embedded GPP	1,123	225	1,800	1.1979	71.87	.49	29.11	20
MicroCore	79	29.1	250	.6067	36.40	.0044	.26	2,198

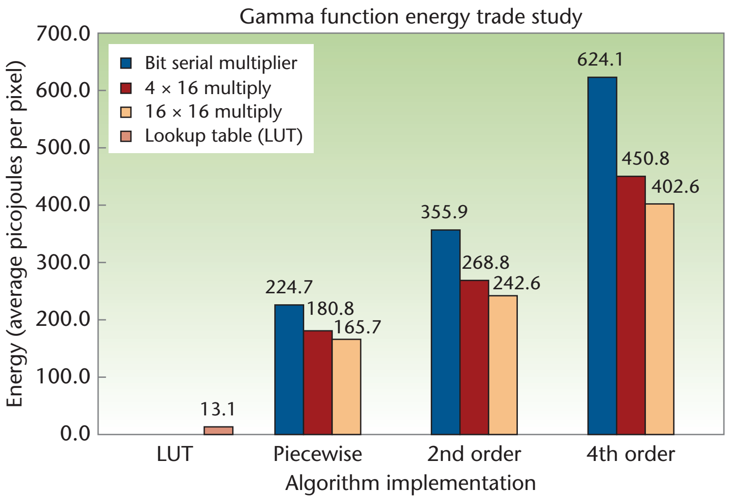

Figure 1. Energy trade study for MicroCore gamma-correction algorithm implementations. The four gamma-function implementations used the pixel value as an index to a precomputed lookup table (LUT); a piecewise linear approximation of the gamma function; a second-order and fourth-order polynomial approximation of the gamma function; and variants using different types of multipliers.

compiler expertise to create a compiler for multiple microcoded machines. We recently funded a successful research and proof-of-concept effort to extend the GCC compiler and produce microcode for our MicroCore processor.

Floating point. Embedded systems designers often avoid the use of floating point representations. Historically, integer operations executed faster and use less power. The number representation is generally more compact using integers (16 bits versus 24 to 32 bits). Smaller representations result in less memory and storage and lower cost and power to access those numbers. Often a part of the embedded system design eliminates the application's floating point portions. We use a toolbox of approaches to eliminate the floating point numbers: fixed-point representation, fractional representations, and replacing formulas with piecewise linear approximations or table-lookups. All of these approaches result in lower power and higher performance for the applications. Of course, designers must be cautious, and evaluate the approximations' numeric precision.

We ran a power analysis study on a gamma-correction algorithm, where floating point exponential functions were replaced with integer solutions. The gamma function adjusts the display's intensity and contrast. Each pixel value is used as an input to the gamma function. Display quality variations can be calibrated and used as constants in the gamma function. We investigated four implementations of the gamma function using

- the pixel value as an index to a precomputed lookup table (LUT),
- a piecewise linear approximation of the gamma function,
- a second-order and fourth-order polynomial approximation of the gamma function, and
- variants using different types of multipliers.

We considered a bit serial, four sets of 4 × 16 multipliers, and a 16 × 16 multiplier, and analyzed each variant's power and performance. Figure 1 shows the results.

The power advantage of using a lookup table (LUT) is clear from analysis. A LUT-based algorithm reduces power consumption by more than an order of magnitude compared to the next closest competitor. This methodology can be applied to a range of algorithms where the input values can be bounded or where careful precision analysis is possible.

We can make similar trade-offs in HPC. Traditionally, most numeric algorithms use double-precision floating point representations, but there are cases where single precision is adequate or where combining double and single can provide better performance. Single-precision computations are lower power, often faster, and require half the memory bandwidth.

Hardware-Software Codesign

Perhaps the largest change to "business as usual" HPC is a social, rather than technological, change. HPC has been narrowly confined to commercial off-the-shelf (COTS)-based hardware and legacy code that's often decades old. These constraints on both hardware and software design have produced a feedback cycle that's hard to break: supercomputers are built from COTS components, so code is written for COTS components, so we buy computers based on COTS components. The new requirements of exascale systems—most notably power—require a departure from this methodology.

In contrast, the embedded design space has traditionally embraced a codesign methodology. With codesign, problem requirements are analyzed and the solution is partitioned between hardware and software, which are specially designed to meet the problem requirements. This methodology requires a greater understanding of both the problem to be solved and the hardware constraints, but it lets developers design optimal solutions.

Specialized coprocessors. We've done significant research on the benefits of widely used hardware coprocessors for low-power, high-performance embedded microprocessor systems.[4] Chronological examples include math coprocessors, graphics engines, network processors, and TCP/IP offload engines. Integrated cores are commercially available to provide acceleration for specific functions such as discrete cosine transform (DCT), Advanced Encryption Standard (AES), and Reed-Solomon.

Designing these offload engines is nearly as difficult as developing a small microprocessor. Historically, a team of engineers would define the interfaces and architecture, and implement the design in schematics or a hardware definition language. We investigated a new approach for building core parts of these offload engines. We found that we can convert the software algorithms directly to hardware, which is an approach that became feasible just a decade ago. Advanced tools and ASICs with billions of transistors are now available to support these functions.

Modern tools let us rapidly develop hardware solutions for multiple software algorithms. In a few weeks, we were able to develop hardware solutions for 10 key software algorithms. The tool automatically generated very high-speed integrated circuit hardware-description language (VHDL) that could target both field-programmable gate array (FPGA) and service-oriented computing (SOC) designs. We achieved the performance results of this study without manual design or "tweaking" of the hardware circuitry or the original software. In contrast, an engineering co-op student spent a summer profiling the MPEG4 software, converting five critical lines of C code to schematic design, and testing, validating, and documenting results. The solutions provided performance gains of seven to 70 times that of the original software.[4] We found an average performance gain of 16.8 times the original with these routines. The conversion of a broad set of 10 software routines has shown an average of 270 gates for every line of C source code.

The MicroCore design is easily extended with multiple arithmetic logic units, multipliers, custom operations, and larger memories. The multicore MicroCore Vision Processor includes duplicate arithmetic units to manage image addressing and data processing and specialized hardware for image processing, compression, and encryption algorithms. This specialized hardware is directly integrated in the MicroCore design; the communication overhead typical of coprocessors is drastically reduced. We recently implemented six image-processing algorithms in microcode on this MicroCore. The average microcoded instruction with this hardware support is comparable to more than 18 RISC instructions.

Software tools. Simulation is a key aspect of codesign. We have Simulink models of a basic MicroCore and of a networked array of MicroCore processors. As an example, we programmed test routines to evaluate the processing, memory, logic, and communication subsystems. We implemented the core AES encryption algorithm for a grid of 4×4 MicroCores in Simulink. The AES algorithm uses more than 300 lines of C source code; the same algorithm has been implemented in 14 lines of microcode. This microcoded architecture performs a 128-bit encryption in approximately 100 microcode clock cycles, which would result in encrypting at 600 megabits per second using only 60 milliwatts. By developing algorithms in simulation, we can optimize both the software and hardware implementations in a way that would be impossible to develop on finished hardware alone.

Similarly, understanding the power and performance of large HPC systems requires extensive simulation. Many performance effects become evident only when thousands or tens of thousands of nodes act (or fail to act) in concert.

Yet, many HPC systems are designed without detailed simulation at scale. Modular, scalable, full-system simulations that consider all major system interactions must be performed at multiple levels of detail to be effective. To address this, Sandia has recently developed the Structural Simulation Toolkit,[5] a parallel simulator of large parallel systems. This simulator includes both detailed and abstract models of processors, network components, and memory—and allows simulations to capture the often subtle challenges of scaling.

Exploit More Levels of Parallelism

The MicroCore Vision Processor exploits many levels of parallelism in an embedded multicore multiprocessor. It provides intrinsic fine-grain parallelism in the microcode, hardware coprocessing for special functions, and coarse-grain parallelism with multiple cores. Similar trade-offs can be made in HPC.

Fine-grain communication. Each MicroCore processor performs independent operations and can share data and processing results with neighboring processors in a multiprocessor array. We have provided fast communication between the microcoded processors by using registers to transfer small data packets. Exchanges between processors can occur under direct microcode control on a cycle-by-cycle basis. We recognize that high-speed, low-latency communication is critical for good parallel computing performance and that multiple communications are needed. We currently offer a synchronized exchange of packets between MicroCores; we can also offer packet routing between arbitrary MicroCores in an array. This range of communication provides efficient and fast exchanges of single packets, long messages, and external network traffic.

We currently use four register first-in, first-outs (FIFOs) to provide data exchange to the four nearest-neighbor MicroCores. This implements a simple 2D mesh network for a large group of microcoded machines. Hardware control lines and microcode control store bits are added to read from or write to one of the four neighbors. We're finding that this simple, low-power connectivity addresses many on-chip network research challenges.[6] Our early results from models, simulations, and FPGA implementations show that this fine-grain communication supports many image-processing and communication algorithms efficiently.

The fine-grain processing and communication of our MicroCore Vision Processor system in some aspects is a throwback to systolic arrays or legacy mainframe programming. It also provides a programmable system that's tightly coupled to the hardware and addresses many of the current issues for the exascale supercomputer. Multiple groups have recently advocated such changes as well as the need for radical change to the computer architecture.[7]

Coarse-grain parallelism. Previous studies have shown that multiple small microprocessors can outperform a single complex microprocessor and use less power.[8] To provide the performance of some envisioned multifunction embedded products, multiple microcoded engines will be needed. We're thus designing a version of the Vision Processor to use up to 256 MicroCores in an array.

Table 3 shows the algorithm performance summary for a broad set of image-processing algorithms. From this analysis we found that the average speedup from commercial processor software to microcode is about 18 times, the speedup of commercial processor software to a coprocessor tool is about 17 times, and the speedup of commercial software to a hand-built (manual) coprocessor is about 150 times. These performance gains and the coarse-grain parallelism provide a unique approach to provide highly power-efficient computing for embedded systems.

Message passing communication. The message passing interface (MPI) that forms the foundation for most parallel HPC codes is an example of coarse-grain parallelism. While extremely successful at expressing high-level parallelism, MPI incurs significant overhead, which makes it difficult to use for expressing fine-grain parallelism, such as short exchanges of data between tasks. As a result, many applications are pushed toward large-message communication patterns. A typical algorithm has each task compute a time step and then exchange boundary values with its neighbors. While conceptually simple, this has the drawback of performing only computation or communication at a given time. This means that the network is either idle or overworked and requires all tasks to be closely synchronized.

The MPI standard currently requires that an entire message buffer be present before any of that data can be used. If it were possible to send partial messages or to let a task begin reading from an

Table 3. Early software performance results for various image-processing algorithms.

		Average clocks per pixel				Multicore		
Function	Major algorithms	Software (tool)	Microcode (manual)	Coprocessor (tool)	Coprocessor (manual)	4 Micro-Cores	16 Micro-Cores	64 Micro-Cores
Compression	Discrete wavelet transformation (DWT), embedded block coding with optimized truncation (EBCOT)	1,123	79	216	2.5	26	8.6	2.8
Dynamic range compression	Beta algorithm (linear)	328	34	121	2	8.5	2.1	0.5
Dynamic range compression	Beta algorithm (table)	19	2	12	2	0.5	0.1	0.0
Automatic dynamic range	Histogram	90	2	13	2	0.5	0.1	0.0
Hands-free focus	DWT	326	11	5	2	4	1.4	0.5
Dynamic range enhancement	Linear scale	400	50	148	2	12.5	3.1	0.8
Noise reduction	Two neighbors	40	4	15	2	1.5	0.6.	0.2

incoming message buffer before the entire message arrived, it would be possible to overlap communication and computation. This would reduce the demands on the interconnection network and make processing more efficient, but it would require hardware support. Analysis of several Sandia applications shows that this computation model is possible, as most applications consume data at a much lower rate than the network can provide—often by a factor of two to five times.[9] If processors and memory systems were designed to allow this low-level locking, performance could be improved and networks could be designed with less bandwidth.

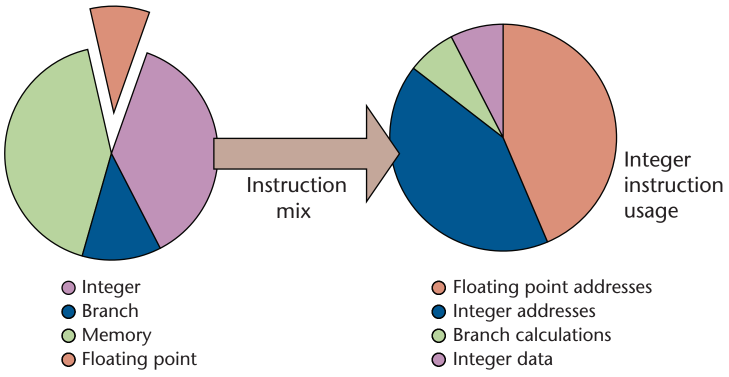

Figure 2. Instruction breakdown and integer usage. In terms of frequency, integer instructions are second only to memory access instructions; of these integer instructions, the bulk are used to compute addresses for future load or store operations.

Improve Data Access

An analysis of several Sandia codes reveals that less than 10 percent of executed instructions are floating point operations—even in ostensibly "floating point intensive" applications. Memory access instructions (loads and stores) are the most frequent class of instructions, followed by integer instructions. Of these integer instructions, the bulk are used to compute addresses for future load or store operations (see Figure 2). It seems that "floating point intensive" codes are really "memory intensive codes" or "address computation intensive" codes.

Proximity to memory. To demonstrate the importance of the memory bottleneck, we simulated three Sandia codes: depth-first search (irregular graph traversal), Cube3D (dense floating point), and KMetis (graph partitioning) on a range of "ideal" processors. These "ideal" processors included a processor with perfect branch prediction (allowing it to "guess" which program paths would be taken and execute them earlier), a processor with an infinite number of fast functional units (allowing many instructions to be executed in parallel), and processors with the latency to main memory reduced by 50 and 75 percent. For all of the codes, the performance improvement over the baseline processor was minimal (0 to 8 percent) except for the processors with lower memory latency, which improved 80 to 200 percent. Clearly, for these codes, memory access is the limiting factor.

Recent developments in electronics packaging might make it possible to dramatically reduce the latency between processors and memory.[1] Computer chips are currently encased in a package and connected to other chips by conductors in a printed circuit board. Accessing this physically distant memory requires substantial power. In future systems, advanced packaging will let processors and memory exist in the same package, by either stacking them on top of each other or by placing them side-by-side, allowing signals to travel much shorter distances.

For example, a modern memory chip consumes about 225 to 250 pJ per word transferred.[1] By comparison, a close-proximity DRAM would only expend only five to 10 pJ per word—a 95 to 98 percent reduction. Additionally, close integration might enable architectural changes—such as offloading some DRAM logic from the DRAM chip or reducing the DRAM's active page size.

Close proximity between processor and memory addresses many of the performance and power concerns. Unlike hardware solutions—such as larger caches, longer memory queues, and prefetching—that make memory "appear" closer yet require additional power, close proximity memory actually reduces power requirements. This approach is already being adopted by many embedded systems through the use of eDRAM. Many processor-in-memory (PIM) projects, such as the Execube and data-intensive architecture (DIVA), PIMLite, and Intelligent RAM (IRAM) have shown the approach's power and performance benefits. Because much of a conventional processor is devoted to dealing with the great distance to memory, a shorter memory access lets simpler processor cores perform better than the current complex cores.

Explicit data management. To minimize microprocessor complexity and power, MicroCore doesn't provide hardware support for a shared address. This forces the software developer (or tools) to define and maintain the data locality and coherence. Although this adds complexity to the programmer's task, it isn't unheard of in HPC. For example, the MPI that dominates current HPC workloads doesn't assume a shared address space and instead relies on the programmer to partition the problem and coordinate communication.

Forcing the programmer to manage an application's data layout allows much simpler hardware and significant power savings. Many modern processors use complicated prefetching units in an attempt to guess what data a program will need next. Although this can improve performance in some cases, in most cases it results in extra data accesses (extra energy) and can even harm performance by using up limited bandwidth. Indeed, programmers often have better knowledge of the structure of memory accesses, and can do a better job than the hardware. Similarly, most modern HPC processors contain several levels of cache that try to guess which data will be reused. Again, this guessing adds hardware complexity and consumes power. Embedded processors such as the MicroCore and the IBM Cell BE processor avoid this by using user-controlled scratchpad memories—relying on programmer knowledge to reduce power and complexity.

To reach the goal of a tera-sized embedded or exa-sized HPC system, major challenges must be overcome. However, by cross-applying the experiences of both the embedded and HPC communities, such challenges need not be insurmountable. The community must embrace fundamental shifts in design methodology, such as relentlessly exploiting parallelism at all levels and adopting hardware-software codesign. We must re-examine assumptions, from the use of floating point in applications to the complexity of microprocessors. And we must develop new compilers and simulators that optimize for both power and performance.

As Table 4 summarizes, combining the techniques we outlined here could have a dramatic impact on future systems. Simplified processors like the MicroCore could reduce power consumption by 95 percent from the baseline of current microprocessors. Reducing the use of floating point operations, or using smaller floating point representations, could offer additional improvement. Bringing memory closer to the processor through advanced packaging would reduce communication power and enable architectural changes,

Table 4. Projected effects of power-reduction techniques on an exascale system in three areas of power consumption.

	2018 estimate	Reduction techniques	Resulting power consumption	Percentage of reduction
Processing	224 MW	Simpler processor, reduce floating point	11.2 MW	95%
Memory	125 MW	Closer proximity	37.5 MW	70%
Interconnect	24 MW	Message overlap	12 MW	50%
Total	373 MW	—	60.7 MW	84%

potentially reducing memory system power by 70 percent. Explicit data management, combined with better overlap of communication and computation, could reduce bandwidth requirements by 50 percent or more.

These enhancements can enable tera-size computing for handheld embedded systems. New sets of capabilities will be enabled with hundreds of gigaops of computing performance running on batteries for long missions. Miniaturization enables many of the power savings and also supports the concept of a disappearing computer. Applying these techniques to the hypothetical HPC system we described in the introduction could reduce power by 84 percent. This would transform a wholly impractical machine—consuming hundreds of megawatts and costing significant fractions of a billion dollars per year to operate—into a much more attainable machine consuming tens of megawatts. More aggressive application of the techniques, along with technology improvements, could reduce it even further.

Acknowledgments

The authors acknowledge countless discussions and interactions with fellow employees at Rockwell Collins and at Sandia National Laboratories, with our US Department of Defense sponsors, and with Notre Dame University staff.

References

1. P. Kogge, ed., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, report, US Defense Advanced Research Projects Agency, 28 Sept. 2008.
2. J. Scaramella, *Worldwide Server Power and Cooling Expense 2006–2010 Forecast*, report #203598, IDC Market Analysis, Sept. 2006.
3. W. Dally et al., "Efficient Embedded Computing," *Computer*, vol. 41, no. 7, 2008, pp. 27–32.
4. D. Jensen, "Developing System-on-Chips with Moore, Amdahl, Pareto, and Ohm," *Proc. Int'l Conf. IEEE Electro/Information Technology*, IEEE Press, 2008, pp. 13–18.
5. A. Rodrigues, *The Structural Simulation Toolkit*, Sandia Nat'l Laboratories, 18 Feb. 2010; www.cs.sandia.gov/sst.
6. J.D. Owens et al., "Research Challenges for On-Chip Interconnection Networks," *IEEE Micro*, vol. 27, no. 5, 2007, pp. 96–108.
7. D. Donofrio et al., "Energy-Efficient Computing for Extreme-Scale Science," *Computer*, vol. 42, no. 11, 2009, pp. 62–71.
8. A. Gontmakher et al., "Using Fine Grain Multithreading for Energy Efficient Computing," *Proc. 12th ACM Symp. Principles and Practice of Parallel Programming*, ACM Press, 2007, pp. 259–269.
9. A. Rodrigues, *Dusty Decks, Memory Walls, and the Speed of Light*, doctoral thesis, Computer Science and Eng. Dept., Univ. of Notre Dame, 2006.

***David W. Jensen** is a principal engineer in the Advanced Technology Center at Rockwell Collins in Cedar Rapids, Iowa. He has been issued eight diverse patents for innovations in advanced computing hardware and software systems. His research interests focus on many aspects of power-efficient computing systems. Jensen has a PhD in computer science from the University of Illinois at Urbana Champaign and an MBA from the University of Iowa. He is a member of IEEE and the ACM. Contact him at dwjensen@rockwellcollins.com.*

***Arun F. Rodrigues** is a senior member of the technical staff in the Scalable Computer Architecture group at Sandia National Laboratories in Albuquerque, New Mexico. His research interests include advanced architectures, non-silicon architecture, advanced packaging, and system simulation. Rodrigues has a PhD in computer science and engineering from the University of Notre Dame. He is a member of IEEE. Contact him at afrodri@sandia.gov.*