

Back to Thin-Core Massively Parallel Processors

Ami Marowka, *Bar-Ilan University, Israel*

Examination of the innovations of the past three decades that brought chips to the point at which many-core processors are possible reveals that there are multiple roads ahead, and each is full of challenges.

Multicore processors are inside every desktop computer, supercomputer, and mobile device. Unfortunately, general-purpose parallel applications that fully exploit these processors are still rare. Furthermore, although parallel architectures are advancing rapidly, the development of parallel software is lagging behind.

Chipmakers believe they can resolve the fundamental technical impediments to building a 1,000-core processor and are studying the efficiency of novel advanced parallel architectures for many-core processors.¹ However, software researchers still struggle to achieve new technology breakthroughs in the design of software architecture to make the programming of future many-core processors feasible.

Software architects are looking for new approaches to dealing with various pitfalls and issues such as data dependency, race conditions, load balancing, nondeterminism, and deadlocks. In addition, they are researching advanced parallel programming paradigms that will allow the construction of easy-to-use, productive, energy-efficient, scalable, and portable parallel applications.² These challenges are difficult to overcome, particularly because the design of new parallel architectures is characterized by a

high degree of architectural diversity between successive multicore processor generations.

Sequential computing has not suffered from this deficiency because it has a universal computation model—the von Neumann architecture. This architecture's stability over the past six decades fostered the prosperity of the software industry as we know it today. Unfortunately, parallel computing does not have a universal computation model to bridge the hardware-software gap, which brings us to the first crisis in computing since the birth of the industry.

THE PARALLEL REVOLUTION

The paradigm shift from sequential to parallel computing is a revolutionary leap, not an evolutionary step,³ because it changes almost every area in computer science—architectures, operating systems, algorithms, languages, data structures, databases, and so on. However, the parallel revolution is a revolution in progress, and as such it will probably stumble upon anomalies, measures, or indicators that conflict with the existing scientific paradigm. The appearance of anomalies might lead to the conclusion that the existing paradigm does not solve the current scientific problems and that a viable alternative is needed. Many

researchers believe that cloud computing will be the new paradigm if the parallel revolution fails.

More than any other new architectural trend, the shift from complex multicore processors to simplified many-core processors symbolizes the fact that we are at a technology crossroads. The transition from single-core to multicore processors was driven by technology constraints such as energy efficiency and the performance limitations of instruction-level parallelism (ILP). Superscalar CPU architectures enhance performance by adding execution units and sophisticated optimization techniques such as dynamic branch prediction, out-of-order execution, and deep pipelines for higher clock frequencies. Unfortunately, this approach is hitting a wall and can no longer guarantee linear gains. Furthermore, using extra chip space to boost performance by superscalar aggressive optimization decreases power efficiency and increases heat generation. However, using the same space for more but simpler cores can achieve better performance per watt.¹

The source of inspiration for the Connection Machine design was the human brain—a massively parallel network of simple processing units (neurons).

The first decades of parallel computing

Parallel computing was already in the minds of the first computer designers. Computers such as ENIAC and IAS were actually parallel machines: ENIAC was a parallel dataflow machine in which program memory was located in each unit, and IAS was an asynchronous machine with parallel memory and parallel arithmetic. However, using these machines' parallel features was very complicated, so their architects concluded that serial operation was preferable in electronic machines, given fast enough components.⁴ Moreover, they claimed that serial machines were preferable to parallel machines because of their reduced hardware requirements, increased reliability, and ease of programming.

During the 1960s and 1970s, the most notable research effort to build a large-scale parallel machine was the Illiac series, which was constructed by the University of Illinois and Burroughs Corporation over a decade. The last machine in the series, the ILLIAC IV, was a vector machine with up to 256 processors. Unfortunately, the project failed, mainly because it did not deliver real-world performance. Nevertheless, the architectural concept behind the ILLIAC IV, which was very efficient for solving many scientific and engineering problems, inspired researchers to develop new supercomputer architectures.

The Cray vector machines, which appeared in the 1980s, were the first commercially successful supercomputers, using more sophisticated pipeline techniques than their predecessor vector machines and improving performance on a wide range of application domains. However, the Cray architects had to overcome problems similar to those that challenge many-core processor designers today: developing the electrical circuitry for the desired computation speed and power consumption. To solve this problem, the architects constructed the Cray machines using new integrated circuits in a C-shape chassis, which became their identifying mark. To cope with the heat the machine generated, the designers built a special liquid cooling system that enveloped the entire machine.

From the mid-1980s to the mid-1990s, leading high-performance computing (HPC) vendors put many new parallel processors and parallel machines on the market. New start-ups appeared with many ideas for supercomputing innovations. Among them were Kendall Square Research's KSR-1 and KSR-2 supercomputers; Meiko Scientific's INMOS transputer microprocessor; the Intel Paragon; Larry Ellison's nCube parallel machine; and supercomputers from NEC, Fujitsu, and Hitachi. Most of these vendors failed to find customers that were willing to buy their supercomputers, and most of the start-ups disappeared—not because they offered unsatisfactory products, but because the markets were not ready for them. However, their technology legacy pushed the supercomputing industry to higher levels.

The rise of massive parallelism

The CM-1 Connection Machine, the first massively parallel supercomputer, introduced a radical new interconnection network design philosophy that inspired many parallel supercomputer architects in the subsequent two decades.⁵

During the 1980s, supercomputers had reached a performance barrier because of the limit on the speed of signal transmission in wires. Therefore, researchers looked to parallelism, despite knowing that parallel machines were complex to construct and difficult to program. The source of inspiration for the Connection Machine design was the human brain—a massively parallel network of simple processing units (neurons). Indeed, the CM-1's basic processing unit was a 1-bit processor. However, harnessing 65,536 of these together and having them simultaneously perform the same computation, each on a different dataset, tremendously increased performance. Furthermore, this single-instruction, multiple-data (SIMD) style of programming was relatively easy.

However, the engineering breakthrough that made the CM-1 unique was its novel interconnection network. The challenge was to find a sparse, flexible, and fast network topology to connect 65,536 processors. The solution, proposed by Richard Feynman,⁶ was a *hypercube topology*. In the

case of the CM-1, 4,096 chips, each with 16 1-bit processors, formed a 12D hypercube. In this way, any two processors could communicate with each other in 12 or fewer hops. Today, inspired by network topologies developed in the past three decades, the architects of future many-core processors are searching in new directions to effectively connect hundreds of simplified cores on-chip rather than off-chip.

The Connection Machine also introduced a rich and advanced development environment. This environment included parallel languages derived from serial languages such as C*, which aimed to create a standard for a parallel C language, and CM Fortran, which was inspired by Fortran 90 and evolved into High-Performance Fortran (HPF) and the CM Scientific Software Library (CMSSL), a set of 250 functions covering a wide range of problems in scientific computation.

The commoditization of supercomputers

During the 1990s, new computer networking technologies made it possible to harness tens, and later hundreds, of workstations together to form a do-it-yourself Beowulf cluster of workstations. These commercial off-the-shelf clusters appeared to be a cost-effective replacement for monolithic supercomputers. They can be built from inexpensive commodity components at a significantly lower price. Moreover, a Beowulf cluster is simple to construct and can easily be scaled up later by adding more computer nodes.

Beowulf clusters represent the fastest-growing choice for building clusters for HPC. As of November 2010, 415 systems (83 percent) in the TOP500 supercomputer list were classified as clusters. Moreover, 392 systems use the Intel EM64T processor family (78 percent), 214 systems use the InfiniBand family interconnect (43 percent), and 228 systems use Gigabit Ethernet (46 percent).

Now history is repeating itself, but in smaller formats. In other words, today, the massively parallel paradigms that were developed for off-chip systems (clusters and supercomputers) are being integrated into on-chip parallel systems (many-core processors).

Energy consumption is also impacting modern supercomputer design as reflected in the Green500 list, which ranks supercomputers based on their performance per watt. For example, as of November 2010, the 1.75-Pflops Jaguar Cray XT5 supercomputer is ranked number 2 in the TOP500 list, but only 81 in the Green500 list. This illustrates the current trend to construct accelerator-based supercomputers for better energy efficiency. As an example, the 2.5-Pflops Tianhe-1A Chinese supercomputer (ranked number 1 in the TOP500 list) powered by GPUs consumes 4 megawatts (ranked number 11 in the Green500 list) compared to the CPU-only 1.75-Pflops Jaguar machine, which consumes 7 MWs.

The appearance of commodity clusters marked a new milestone in the history of supercomputing, but the mas-

sive breakthrough toward mainstream parallel computing that many have been waiting for has not happened.

The shift to multicore processors

During the past five decades, the performance of single-CPU processors has grown exponentially as the CPU clock frequency increased and as more transistors were integrated on a single silicon die to support more sophisticated superscalar optimization techniques in hardware and in replicated execution units. However, the laws of physics limit the improvement in performance that can be achieved by a higher clock frequency because of voltage leakage and heat dissipation. Therefore, hardware designers began using simplified multicore CPUs without further increasing processor frequency.

In the early 2000s, chipmakers Sun Microsystems and IBM introduced the first dual-core processors: Sun's Micro-processor Architecture for Java Computing and IBM's Power 4 microprocessor. These new processors were expensive and targeted special-purpose servers that run computation-

Because chipmakers can still scale up chip size to keep pace with Moore's law, increases in the core count on chips will continue.

intensive tasks and demand HPC. Five years later, AMD and Intel launched the first dual-core processors for desktop computers and changed the computing ecosystem forever.

Eight-core processors are now the norm for desktop workstations and 12-core processors for high-end servers. Because chipmakers can still scale up chip size to keep pace with Moore's law, increases in the core count on chips will continue, leading to multicore processors with 16 or more cores in the foreseeable future. However, beyond 16 cores, the cores will likely be simpler and more heterogeneous.

ENERGY CONSTRAINTS

Power will be a major limiting factor in future processor architectures, so optimizing for performance per watt will be a key driver for massively scalable many-core architectures. The main technique to achieve this goal is to simplify the cores by reducing ILP mechanisms such as out-of-order execution (Intel Atom) and dynamic branch prediction (IBM Cell Broadband Engine) or by removing floating-point units (Tilera TILE64). Designers use the transistor real estate made available by these circuitry reductions to accommodate additional simplified cores and the on-chip interconnection network. However, although an additional core on a single CPU increases energy efficiency, it also expands the interconnection networks and thus increases the power budget.

The on-chip interconnection network is another heavy energy consumer. Contemporary multicore microprocessors use interconnection networks such as Crossbar (AMD Opteron, Intel Core i7, Sun UltraSPARC), Ringbus (IBM Cell BE), and Mesh (Tilera TILE64) to reduce latency and contention. A comprehensive study of the codesign issues of on-chip interconnection architectures must analyze the fine-tuning required among different resources to achieve performance objectives such as bandwidth, latency, power, and area budgets. Under optimistic modeling assumptions, the interconnection network of an eight-core microprocessor might consume as much energy as one core and as much area as three cores.⁷

Using only hardware-based power management techniques to increase power savings is not enough; the software must be power aware as well.

Silicon photonic technology is catching the attention of microprocessor architects as a possible optical communication solution for low-power, highly scalable many-core processors. Researchers have shown that an on-chip hybrid photonic-electronic 64-core mesh network can improve energy efficiency up to 37 times for real-world applications compared to the energy efficiency of the electronic mesh.⁸

The IBM Blue Gene/P supercomputer (1 Pflop) is an example of a contemporary large-scale system design (294,912 processors) based on a many-thin (850 MHz, 1 Gbyte) processing-core approach. The cores are not cache coherent, and each node uses a thin version of the Posix operating system. The communication system uses a fast optical network between racks, three low-latency parallel communications networks between computing nodes, and two Ethernet networks for I/O nodes and maintenance. As of November 2010, Blue Gene/P has achieved high energy efficiency of 378.77 Mflops per watt while ranking 20 on the Green500 list and nine on the TOP500 list.

Using only hardware-based power-management techniques to increase power savings is not enough; the software must be power aware as well. Reducing the parallelism overheads of software systems by developing efficient load-balancing algorithms, low-latency collective communications, fine-grained primitive synchronization, and sophisticated performance optimization techniques directly impacts power consumption.⁹ Moreover, smart algorithms that exploit data locality, perform loop unrolling, eliminate iterative loops and recursive algorithms, and use idle-power-friendly programming languages and librar-

ies as well as auto-tuning based on multiversion algorithms can achieve higher-energy-efficiency applications.¹⁰

GENERAL-PURPOSE GPU COMPUTING

General-purpose GPU computing (GPGPU) adopts the processor-simplification approach.¹¹ Contemporary programmable GPGPU processors offer fine-grained massive data parallelism on-chip alongside graphic processing capabilities that challenge the energy efficiency of mainstream multicore processors.

A typical GPGPU processor has hundreds of simplified cores with small shared-cache memories, each optimized for SIMD data processing via thousands of lightweight threads per core. This approach achieves high scalability with simple, low-overhead thread management and no cache-coherence hardware requirements. Moreover, each core is a massively threaded, in-order, single-instruction-stream processor that shares its control and instruction cache with other cores.

In recent years, GPGPU computing has introduced advanced fine-grained programming modeling and development tools that exhibit real performance gains on a wide range of applications. Experiments have shown that well-tuned and optimized GPGPU applications can achieve a more than 100-fold speedup over sequential execution while delivering attractive energy savings.¹² For that reason, as of November 2010, eight of the 10 highest-ranking machines in the TOP500 list are powered by GPGPU accelerators. However, it is not obvious how to achieve a 100-fold speedup. Other performance studies of well-tuned applications have shown only a threefold speedup over sequential execution.¹³

HETEROGENEOUS COMPUTING

The energy-efficiency advantage of combined CPU/GPU systems has motivated chipmakers to reconsider the benefits of heterogeneous parallel computing.

Integrating CPU and digital signal processing cores on a single chip is an attractive solution in the mobile and embedded market segments,¹⁴ so a similar direction for CPU/GPU computing is an obvious step. Integrating thin cores and fat cores onto a single processor can achieve a better performance gain per watt. For example, a study of analytical models of various heterogeneous many-core processor configurations found that integrating many simplified cores with one complex core achieves greater speedup and energy efficiency than using homogeneous simplified cores.¹⁵

Intel researchers have shown that integrating general-purpose cores alongside special-purpose hardware accelerators can improve energy efficiency by an order of magnitude.¹⁶ Based on their view, future terascale processors will contain a few dedicated hardware accelerators, such as speech recognition, GPU, and encryption accelera-

tors, that operate at ultra-low voltage (as low as 320 mV) and ultra-low frequency (as low as 23 MHz) and consume ultra-low power (as low as 56 microwatts).

Moreover, the Intel researchers offer a unified programming model for a CPU/GPU platform that improves the programmability and throughput of such heterogeneous systems.¹⁷ For example, CPU/GPU memory architectures currently use separate memory address spaces for the CPU and the GPU, with the programmer responsible for communication between them using a relatively slow communication channel. This communication bottleneck limits the peak throughput that can be obtained from these systems. The proposed programming model offers a global memory address space that supports shared pointers and data structures. A shared memory model enables concurrent processing of the same data on the CPU and on the GPU without needing to manually transfer the data back and forth. Nvidia's Fermi architecture and its CUDA (Compute Unified Device Architecture) programming model support a similar enhancement.

Future many-core processors pose great challenges for software architects. Software development will need to be revolutionized to achieve widespread use of heterogeneous many-core systems. Not only will new, highly abstract parallel programming models and high-level languages be needed to make parallel programming as simple as serial programming, but new creative solutions for many parallel programming pitfalls and issues also must be found.¹⁸ Moreover, because redesigning and rewriting applications is time-consuming and expensive, most legacy applications are still waiting to be parallelized.

MANY-CORE PROGRAMMING CHALLENGES

The software research community has not kept pace with the accelerating development of new hardware architectures for multicore processors. Parallel programming is considered a tedious and error-prone task. Writing correct and efficient parallel applications calls for sophisticated development and debugging tools to make multicore programming safer and more convenient. Several other difficulties exist.

Thinking in parallel

Many-core programming demands new programming skills and a different kind of problem-solving thinking. Sequential programming is a deterministic and predictable process, so it arises intuitively from the way programmers solve problems using algorithms. In contrast, many-core programming is intrinsically a nondeterministic process. Parallelism requires the programmer to think in a way that humans find difficult. However, allowing nondeterminism in parallel programs is the key to achieving high-performance and scalable programs. A parallel program's nondeterministic behavior is controllable, but must be used only when necessary. Moreover, it is possible to

learn how to think in parallel and how to write bug-free parallel programs.

Bridging the software-hardware gap

Many-core programming demands a close familiarity with the underlying details of the many-core processor architecture. Programmers need this knowledge to match the program's logical structure to the processor's physical architecture. They must be aware of details such as the number of cores, the main memory layout, and the cache memory hierarchy. An efficient match increases performance and achieves the desired scalability. In sequential programming, the programmer is free from this annoying task.

Because redesigning and rewriting applications is time-consuming and expensive, most legacy applications are still waiting to be parallelized.

Programmability issues

Many-core programming demands an understanding of new programming issues that are not a factor in sequential programming. These issues include the relationship between logical threads and the underlying physical cores, thread communication and synchronization, performance measurement in parallel environments, and the sources of load unbalancing. The programmer must verify that the program is free from problems such as data and control dependencies, deadlocks, conflicts, and race conditions.

Implicit problems

Many-core programming bugs are difficult to avoid because they cannot be detected just from inspecting the source code. Actually, What-You-See (in the source code) is not What-You-Get (after optimization). For example, aggressive optimization procedures such as branch prediction and reordering instructions for out-of-order execution are performed "under the hood" and can change a program's functioning. Moreover, these techniques are often the main cause of race conditions, which are hard to detect.

Parallel debugging

Debugging a many-core program is tedious and difficult. Although researchers continue to enhance parallel debuggers and visual profiling analyzers, finding a bug in a parallel program is like finding a needle in a haystack. The complexity of parallel debugging is due to the hidden nature of problems and the nondeterministic timing of the program's multiple execution threads. Parallel debugging becomes even more difficult in the case of temporary bugs whose appearance cannot be predicted.


MANY-CORE TASK FORCE

Many parallel computing research centers around the world are working on making parallel programming synonymous with programming. In the past five years, more research laboratories have joined the many-core worldwide task force while recruiting young researchers and obtaining funding from industry. This mission has become crucial for the computer industry and for computerized society overall as well.

Currently, the responsibility for bridging the gap between hardware and software to write better parallel programs might ultimately lie with developers. Many programmers are not up-to-speed on the latest developments in hardware design. They need to study chip architectures to understand how their code can perform better. This is not a desirable situation; parallel programming should be as simple and intuitive as sequential programming.

Efforts are focusing on making the art of many-core programming a productive task that produces high-performance applications that are also energy efficient, scalable, and portable with a high degree of correctness. To realize this vision, scientists are studying highly abstract programming frameworks in which the target hardware platform is assumed to be composed of hundreds of heterogeneous cores simultaneously executing hundreds of threads.

The frameworks under construction are usually two-level infrastructures. Their upper level will allow domain experts to use powerful abstractions to design parallel programs using intuitive APIs that will hide the underlying system complexity. Their lower level will be designed by well-trained developers and will consist of optimized software modules. This software layer will be responsible for achieving the best possible match of the software abstractions to the complexity of the underlying hardware architecture. Such framework partitioning aims to improve portability and increase productivity.

A 1,000-core microprocessor will be an impressive engineering milestone in the history of computing, but incorporating it inside a laptop with less than a 100-W power requirement will be an outstanding achievement. Beyond this point, we must wait and see whether 65,536-bit-core massively parallel processors will be back, but this time entirely on a single chip. 

References

1. S. Borkar, "Thousand Core Chips—A Technology Perspective," *Proc. 44th Design Automation Conf. (DAC 07)*, ACM, 2007, pp. 746-749.
2. B. Catanzaro et al., "Ubiquitous Parallel Computing from Berkeley, Illinois, and Stanford," *IEEE Micro*, vol. 30, no. 2, 2010, pp. 41-55.
3. S.H. Fuller and L.I. Millett, "Computing Performance: Game Over or Next Level?," *Computer*, Jan. 2011, pp. 31-38.

4. J. von Neumann, "First Draft of a Report on EDVAC," Moore School of Electrical Eng., Univ. of Pennsylvania, 1945.
5. D.W. Hillis, *The Connection Machine*, MIT Press, 1985.
6. D.W. Hillis, "Richard Feynman and the Connection Machine," *Physics Today*, vol. 42, no. 2, 1989, p. 78.
7. R. Kumar, V. Zyuban, and D.M. Tullsen, "Interconnections in Multicore Architectures: Understanding Mechanisms, Overheads, and Scaling," *Proc. 32nd Ann. Int'l Symp. Computer Architecture (ISCA 05)*, ACM, 2005, pp. 408-419.
8. G. Hendry et al., "Analysis of Photonic Networks for a Chip Multiprocessor Using Scientific Applications," *Proc. 2009 3rd ACM/IEEE Int'l Symp. Networks-on-Chip (NOCs 09)*, IEEE CS, 2009, pp. 104-113.
9. B. Steigerwald and A. Agrawal, "Developing Green Software," white paper, Intel, 2010; <http://software.intel.com/en-us/articles/developing-green-software>.
10. "Energy Smart Software," white paper, Microsoft, 2010; <http://msdn.microsoft.com/en-us/windows/hardware/gg463226.aspx>.
11. D. John et al., "GPU Computing," *Proc. IEEE*, vol. 96, no. 5, 2008, pp. 879-899.
12. J. Tolke and M. Krafczyk, "TeraFLOP Computing on a Desktop PC with GPUs for 3D CFD," *Int'l J. Computational Fluid Dynamics*, vol. 22, no. 7, 2008, pp. 443-456.
13. V.W. Lee et al., "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," *Proc. 37th Ann. Int'l Symp. Computer Architecture (ISCA 10)*, ACM, 2010, pp. 451-460.
14. K. Berkel, "Multi-core for Mobile Phones," *Proc. Design, Automation, and Test in Europe (DATE 09)*, IEEE CS, 2009, pp. 1260-1265.
15. D.H. Woo and H.S. Lee, "Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era," *Computer*, Dec. 2008, pp. 24-31.
16. R.K. Krishnamurthy and H. Kaul, "Ultra-Low Voltage Technologies for Energy-Efficient Special-Purpose Hardware Accelerators," *Intel Technology J.*, vol. 13, no. 4, 2009, pp. 100-117.
17. B. Saha et al., "A Programming Model for Heterogeneous Intel X86 Platforms," *Intel Technology J.*, vol. 13, no. 4, 2009, pp. 42-61.
18. A. Marowka, "Pitfalls and Issues of Many-Core Programming," *Advances in Computers*, vol. 79, M.J. Zelkowitz, ed., Elsevier, 2010, pp. 71-117.

Ami Marowka is an adjunct assistant professor in the Department of Computer Science at Bar-Ilan University, Israel. His research interests include the portability of high-performance applications, parallel computing, advanced computer architectures, and tools for parallel computers and ad hoc networks. Marowka received a PhD in computer science and engineering from the Hebrew University of Jerusalem. Contact him at amimar2@yahoo.com.



Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.