GPUs and the Future of Parallel Computing

This article discusses the capabilities of state-of-the art GPU-based high-throughput computing systems and considers the challenges to scaling single-chip parallel-computing systems, highlighting high-impact areas that the computing research community can address. Nvidia Research is investigating an architecture for a heterogeneous high-performance computing system that seeks to address these challenges.

Stephen W. Keckler
William J. Dally
Brucek Khailany
Michael Garland
David Glasco
Nvidia

Although Moore's law has continued to provide smaller semiconductor devices, the effective end of clock rate scaling and aggressive uniprocessor performance scaling has instigated mainstream computing to adopt parallel hardware and software. Today's landscape includes various parallel chip architectures with a range of core counts, capability per core, and energy per core. Driven by graphics applications' enormous appetite for both computation and bandwidth, GPUs have emerged as the dominant massively parallel architecture available to the masses. GPUs are characterized by numerous simple yet energy-efficient computational cores, thousands of simultaneously active fine-grained threads, and large off-chip memory bandwidth.

Modern GPU computing lets application programmers exploit parallelism using new parallel programming languages such as CUDA[1] and OpenCL[2] and a growing set of familiar programming tools, leveraging the substantial investment in parallelism that high-resolution real-time graphics require. To date, more than 300 million CUDA-capable GPUs have been sold. Nvidia's current Fermi GPU architecture supports both high-performance graphics and high-performance computing.[3] In its highest-performing computing configuration, Fermi offers a peak throughput of more than 1 teraflop of single-precision floating-point performance (665 Gflops of double-precision) and 175 Gbytes/second of off-chip memory bandwidth.

GPUs' tremendous computing and memory bandwidth have motivated their deployment into a range of high-performance computing systems, including three of the top-five machines on the June 2011 Top 500 list (http://www.top500.org). An even greater motivator for GPUs in high-performance computing systems is energy efficiency. The top two production machines on the Green 500 list of the most energy-efficient supercomputers (http://www.green500.org) are based on GPUs, with the fourth-most-efficient machine overall (Tsubame 2.0 at the Tokyo Institute of Technology) achieving 958 Mflops/W and appearing at number 5 on the Top 500 at 1.2 petaflops. The potential for future GPU performance increases presents great opportunities for demanding applications, including computational graphics, computer vision, and a wide range of high-performance computing applications.

0272-1732/11/$26.00 © 2011 IEEE

However, continued scaling of performance of GPUs and multicore CPUs will not be easy. This article discusses the challenges facing the scaling of single-chip parallel-computing systems, highlighting high-impact areas that the research community can address. We then present an architecture of a heterogeneous high-performance computing system under development at Nvidia Research that seeks to addresses these challenges.

Challenges for parallel-computing chips

Scaling the performance and capabilities of all parallel-processor chips, including GPUs, is challenging. First, as power supply voltage scaling has diminished, future architectures must become more inherently energy efficient. Second, the road map for memory bandwidth improvements is slowing down and falling further behind the computational capabilities available on die. Third, even after 40 years of research, parallel programming is far from a solved problem. Addressing these challenges will require research innovations that depart from the evolutionary path of conventional architectures and programming systems.

Power and energy

Because of leakage constraints, power supply voltage scaling has largely stopped, causing energy per operation to now scale only linearly with process feature size. Meanwhile, the number of devices that can be placed on a chip continues to increase quadratically with decreasing feature size. The result is that all computers, from mobile devices to supercomputers, have or will become constrained by power and energy rather than area. Because we can place more processors on a chip than we can afford to power and cool, a chip's utility is largely determined by its performance at a particular power level, typically 3 W for a mobile device and 150 W for a desktop or server component.

Because power dominates lifetime system costs for supercomputers and servers, their utility is also determined by performance at the target system power level. For example, the reasonable power envelope for future supercomputers has been projected to be 20 MW.[4] The supercomputing community is now aiming to design exascale (10^{18} operations/second) systems. To build such a system within 20 MW requires an energy efficiency of approximately 20 picojoules (pJ) per floating-point operation. Future servers and mobile devices will require similar efficiencies.

Achieving this level of energy efficiency is a major challenge that will require considerable research and innovation. For comparison, a modern CPU (such as Intel's Westmere) consumes about 1.7 nanojoules (nJ) per floating-point operation, computed as the thermal design point (TDP) in watts divided by the peak double-precision floating-point performance (130 W and 77 Gflops). A modern Fermi GPU yields approximately 225 pJ per floating-point operation derived from 130 W for the GPU chip and 665 Gflops. The gap between current products and the goal of 20 pJ per floating-point operation is 85× and 11×, respectively. A linear scaling of transistor energy with technology feature size will provide at most a factor of 4 improvement in energy efficiency. The remainder must come from improved architecture and circuits.

Technology scaling is also nonuniform, improving the efficiency of logic more than communication, and placing a premium on locality in future architectures. Table 1 shows key parameters and microarchitecture primitives for contemporary logic technology with projections to 2017 using CV^2 scaling (linear capacitance scaling and limited voltage scaling with feature size). The 2017 projections are shown at two possible operating points: at 0.75 V for high-frequency operations and at 0.65 V for improved energy efficiency. Energy-efficiency improvements beyond what comes from device scaling will require reducing both instruction execution and data movement overheads.

Instruction overheads. Modern processors evolved in an environment where power was plentiful and absolute performance or performance per unit area was the important figure of merit. The resulting architectures were optimized for single-thread performance with features such as branch prediction, out-of-order execution, and large primary instruction and data caches. In such architectures, most of the energy is consumed in overheads of data supply,

Table 1. Technology and circuit projections for processor chip components.

	2010	2017	
Process technology	40 nm	10 nm, high frequency	10 nm, low voltage
V_{DD} (nominal)	0.9 V	0.75 V	0.65 V
Frequency target	1.6 GHz	2.5 GHz	2 GHz
Double-precision fused-multiply add (DFMA) energy	50 picojoules (pJ)	8.7 pJ	6.5 pJ
64-bit read from an 8-Kbyte static RAM (SRAM)	14 pJ	2.4 pJ	1.8 pJ
Wire energy (per transition)	240 femtojoules (fJ) per bit per mm	150 fJ/bit/mm	115 fJ/bit/mm
Wire energy (256 bits, 10 mm)	310 pJ	200 pJ	150 pJ

instruction supply, and control. Today, the energy of a reasonable standard-cell-based, double-precision fused-multiply add (DFMA) is around 50 pJ,[5] which would constitute less than 3 percent of the computational operation's energy per instruction. Future architectures must be leaner, delivering a larger fraction of their energy to useful work. Although researchers have proposed several promising techniques such as instruction registers, register caching, in-order cores, and shallower pipelines, many more are needed to achieve efficiency goals.

Locality. The scaling trends in Tables 1 and 2 suggest that data movement will dominate future systems' power dissipation. Reading three 64-bit source operands and writing one destination operand to an 8-Kbyte on-chip static RAM (SRAM)—ignoring interconnect and pipeline registers—requires approximately 56 pJ, similar to the cost of a DFMA. The cost of accessing these 256 bits of operand from a more distant memory 10 mm away is six times greater at 310 pJ, assuming random data with a 50 percent transition probability. The cost of accessing the operands from external DRAM is 200 times greater, at more than 10 nJ.

With the scaling projections to 10 nm, the ratios between DFMA, on-chip SRAM, and off-chip DRAM access energy stay relatively constant. However, the relative energy cost of 10-mm global wires goes up to 23 times the DFMA energy because wire capacitance per mm remains approximately constant across process generations. Because communication dominates energy, both within the chip and across the external memory interface, energy-efficient architectures must decrease the amount of state moved per instruction and must exploit locality to reduce the distance data must move.

Memory bandwidth and energy

The memory bandwidth bottleneck is a well-known computer system challenge that can limit application performance. Although GPUs provide more bandwidth than CPUs, the scaling trends of off-chip bandwidth relative to on-chip computing capability are not promising. Figure 1 shows the historical trends for single-precision performance, double-precision performance, and off-die memory bandwidth (indexed to the right *y*-axis) for high-end GPUs. Initially, memory bandwidth almost doubled every two years, but over the past few years, this trend has slowed significantly. At the same time, the GPU's computational performance continues to grow at about 45 percent per year for single precision and at a higher recent rate for double precision.

Power is another limiter for off-die memory. Assuming today's off-chip signaling technology provides 20 pJ/bit, 100-percent utilization of Fermi's 175 Gbytes/second of bandwidth costs approximately 28 W on the GPU die, plus another 21 W in the DRAMs for signaling and DRAM accesses. This 49 W accounts for about 20 percent of an integrated GPU card's thermal design power.

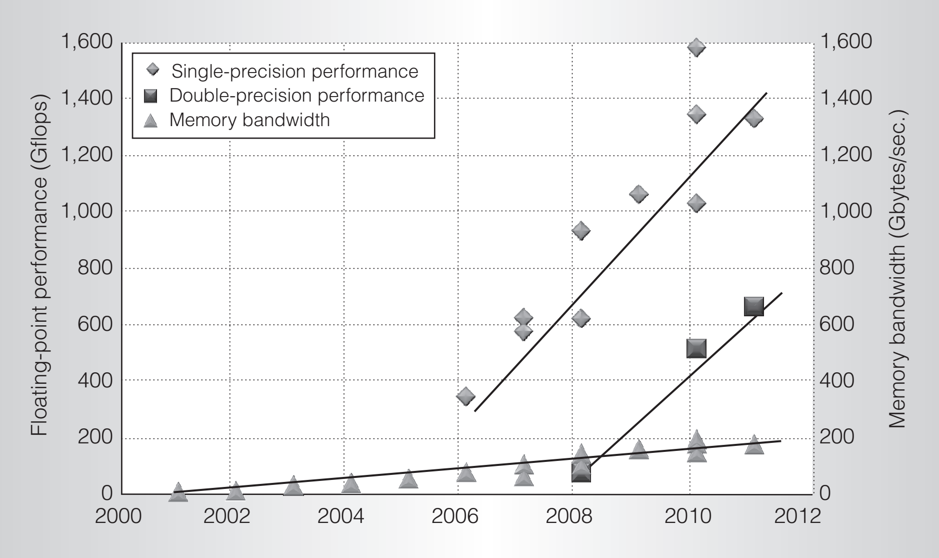

Figure 1. GPU processor and memory-system performance trends. Initially, memory bandwidth nearly doubled every two years, but this trend has slowed over the past few years. On-die GPU performance has continued to grow at about 45 percent per year for single precision, and at a higher rate for double precision.

Table 2. Projected bandwidth and energy for main-memory DRAM.

	2010	2017
DRAM process technology	**45 nm**	**16 nm**
DRAM interface pin bandwidth	4 Gbps	50 Gbps
DRAM interface energy	20 to 30 pJ/bit	2 pJ/bit
DRAM access energy[6]	8 to 15 pJ/bit	2.5 pJ/bit

In contrast, accessing an on-chip memory costs 250 times less energy per bit. For GPU performance to continue to grow, researchers and engineers must find methods of increasing off-die memory bandwidth, reducing chip-to-chip power, and improving data access efficiency. Table 2 shows projected bandwidth and energy for a main-memory DRAM interface. Dramatic circuit improvements in the DRAM interface are possible by evolving known techniques to provide 50 Gbps per pin at 4.5 pJ/bit. Although these improvements will alleviate some of the memory bandwidth pressure, further innovation in low-energy signaling, packaging, and bandwidth utilization will be critical to future system capability.

Packaging. Promising technologies include multichip modules (MCMs) and silicon interposers that would mount the GPU and DRAM die on the same substrate. Both technologies allow for finer pitch interconnects, and the shorter distances allow for smaller pin drivers. 3D chip stacking using through-silicon vias (TSVs) has potential in some deployments but is unlikely to provide the tens of gigabytes of capacity directly connected to a GPU that high-performance systems would require. 3D stacking is most promising for creating dense DRAM cubes, which could be stacked on an interposer or MCM. Also possible are systems that combine GPU and DRAM stacking with interposers to provide nonuniform latency and bandwidth to different DRAM regions.

Bandwidth utilization. Several strategies can potentially improve bandwidth utilization, such as improving locality in the data stream so that it can be captured on chip by either hardware- or software-managed caches; eliminating overfetch or the fetching of data from DRAM that is not used in the processor chip; and increasing the density of data transmitted through the DRAM interface using techniques such as data compression.

Programmability

The technological challenges we have outlined here and the architectural choices that they necessitate present several challenges for programmers. Current programming practice predominantly focuses on sequential, homogeneous machines with a flat view of memory. Contemporary machines have already moved away from this simple model, and future machines will only continue that trend. Driven by the massive concurrency of GPUs, these trends present many research challenges.

Locality. The increasing time and energy costs of distant memory accesses demands increasingly deep memory hierarchies. Performance-sensitive code must exercise some explicit control over the movement of data within this hierarchy. However, few programming systems provide any means for programs to express the programmer's knowledge about the data access patterns' structure or to explicitly control data placement within the memory hierarchy. Current architectures encourage a flat view of memory with implicit data caching occurring at multiple levels of the machine. As is abundantly clear in today's clusters, this approach is not sustainable for scalable machines.

Memory model. Scaling memory systems will require relaxing traditional notions of memory consistency and coherence to facilitate greater memory-level parallelism. Although full cache coherence is a convenient abstraction that can aid in programming productivity, the cost of coherence and its protocol traffic even at the chip level makes supporting it for all memory accesses unattractive. Programming systems must let programmers reason about memory accesses with different attributes and behaviors, including some that are kept coherent and others that are not.

Degree of parallelism. Memory bandwidth and access latency have not kept pace with per-socket computational throughput, and they will continue to fall further behind. Processors can mitigate the effects of increased latency by relying on increasing amounts of parallelism, particularly fine-grained thread and data parallelism. However, most programming systems don't provide a convenient means of managing tens, much less tens of thousands, of threads on a single die. Even more challenging is the billion-way parallelism that exascale-class machines will require.

Heterogeneity. Future machines will be increasingly heterogeneous. Individual processor chips will likely contain processing elements with different performance characteristics, memory hierarchy views, and levels of physical parallelism. Programmers must be able to reason about what kinds of processing cores their tasks are running on, and the programming system must be able to determine what type of core is best for a given task.

Echelon: A research GPU architecture

Our Nvidia Research team has embarked on the Echelon project to develop computing architectures that address energy-efficiency and memory-bandwidth challenges and provide features that facilitate programming of scalable parallel systems. Echelon is a general-purpose fine-grained parallel-computing system that performs well on a range of applications, including traditional and emerging computational graphics as well as data-intensive and high-performance computing. At a 10 nm process technology in 2017, the Echelon project's initial performance target is a peak double-precision throughput of 16 Tflops, a memory bandwidth of 1.6 terabytes/second, and a power budget of less than 150 W.

In this time frame, GPUs will no longer be an external accelerator to a CPU; instead, CPUs and GPUs will be integrated on the same die with a unified memory architecture. Such a system eliminates some of accelerator architectures' historical challenges, including requiring the programmer to manage multiple memory spaces, suffering from bandwidth limitations from an interface such as PCI Express for transfers between CPUs and GPUs, and the system-level energy

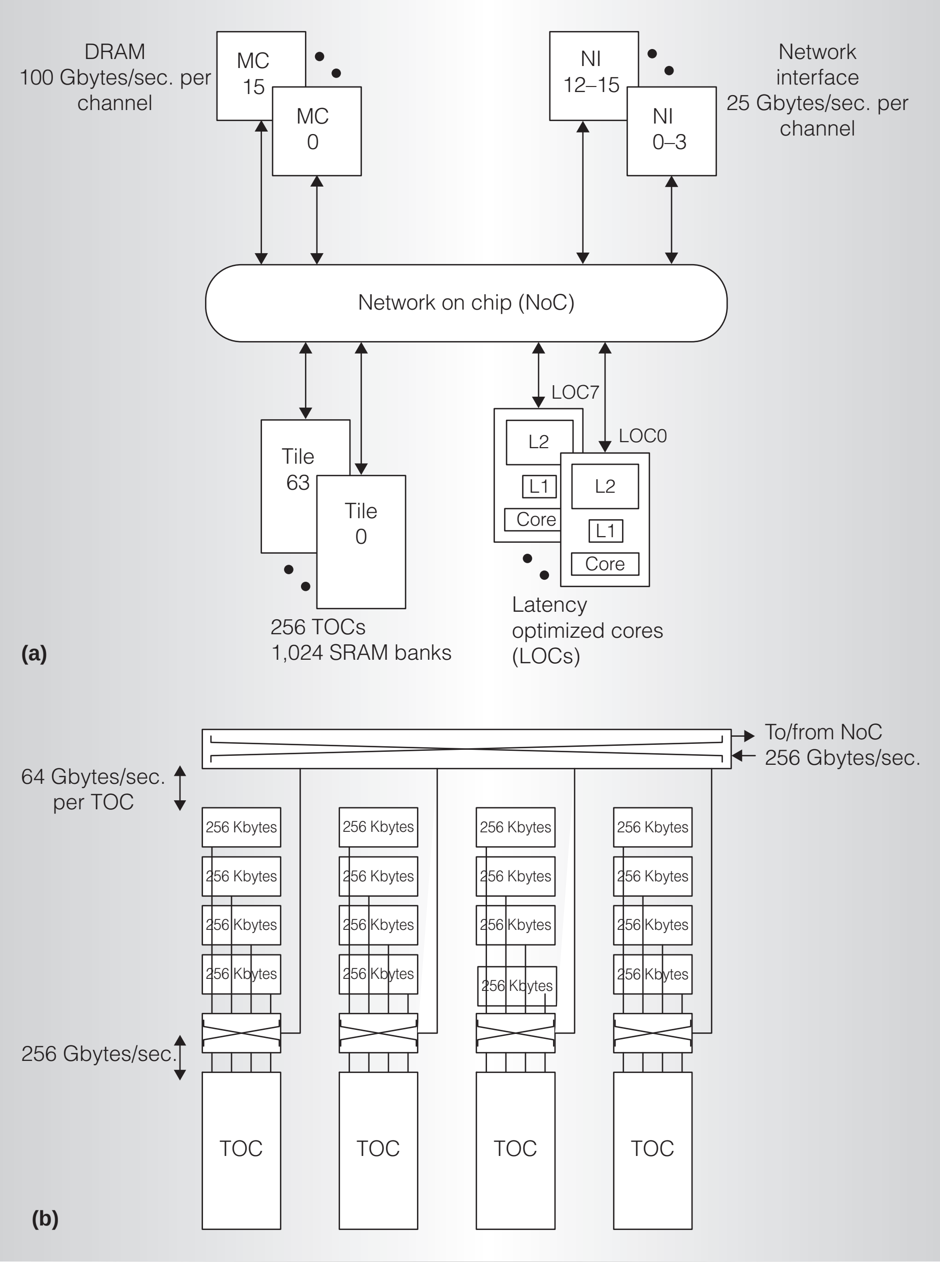

Figure 2. Echelon block diagrams: chip-level architecture (a); throughput tile architecture (b). The Echelon chip consists of 64 throughput computing tiles, eight latency-optimized cores, 16 DRAM controllers, and an on-chip network interface. Each throughput computing tile consists of four throughput-optimized cores (TOCs) with secondary on-chip storage. (L1: Level 1 cache; L2: Level 2 cache; MC: memory controller; NI: network interface.)

overheads for both chip crossings and replicated chip infrastructure. Echelon aims to achieve an average energy efficiency of 20 pJ per sustained floating-point instruction, including all memory accesses associated with program execution.

Echelon architecture

Figure 2a shows a block diagram of an Echelon chip, which consists of 64 throughput computing tiles, eight latency-optimized cores (CPUs), 16 DRAM memory controllers, a network on chip (NoC),

Table 3. Echelon's parameters.

Capability	Per lane	Per TOC	Per tile	Per chip
Instructions per clock	3	24	96	6,144
Instructions per second at 2 GHz	6	48	192	12,288
Double-precision floating-point operations per clock	4	32	128	8,192
Gflops at 2 GHz	8	64	256	16,384
Total threads	64	512	2,048	131,072
Active threads	4	32	128	8,192
Data SRAM (Kbytes)	32	256	5,120	327,680
L0 instruction cache (Kbytes)	1	8	32	2,048
L1 instruction cache (Kbytes)	—	32	128	8,192
DRAM bandwidth (Gbytes/second)	—	—	—	1,600
Network interface bandwidth (Gbytes/second)	—	—	—	400

and an on-chip network interface to facilitate aggregation of multiple Echelon chips into a larger system. Ultimately, such chips can serve a range of deployments, from single-node high-performance computers to exascale systems with tens of thousands of chips. Figure 2b shows that each throughput computing tile consists of four throughput-optimized cores (TOCs) with secondary on-chip storage that can be configured to support different memory hierarchies. We expect that an Echelon chip will be packaged in an MCM along with multiple DRAM stacks to provide both high bandwidth and high capacity. We anticipate that signaling technologies will provide 50 Gbits/second per pin at 2 pJ/bit in an MCM package.

The latency-optimized cores (LOCs) are CPUs with good single-thread performance, but they require more area and energy per operation than the simpler TOCs. These cores can serve various needs, including executing the operating system, application codes when few threads are active, and computation-dominated critical sections that can reduce concurrency. When an application has high concurrency, the TOCs will deliver better performance because of their higher aggregate throughput and greater energy efficiency. The exact balance between LOCs and TOCs is an open research question. Table 3 summarizes the Echelon chip's capabilities and capacities.

Energy-efficient throughput cores

The TOC architecture's goals include achieving

- extreme energy efficiency by eliminating as many instruction overheads as possible,
- memory locality at multiple levels, and
- efficient execution for instruction-level parallelism (ILP), data-level parallelism (DLP), and fine-grained task-level parallelism (TLP).

Figure 3 shows a diagram of the Echelon TOC consisting of eight computing lanes. Each lane is an independent multiple-instruction, multiple-data (MIMD) and highly multithreaded processor with up to 64 live threads. These eight lanes can be grouped in single-instruction, multiple-data (SIMD) fashion for greater efficiency on data-parallel (nondivergent) code. To adapt to the individual applications' locality types, each lane has local SRAM that can be configured as a combination of register files, a software-controlled scratch pad, or hardware-controlled caches. The Echelon hardware can configure each region's capacity on a per-thread group basis when the group launches onto a lane. The lane-based SRAM can be shared across the lanes, forming an L1 cache to facilitate locality optimizations across the TOC's threads.

Exploiting easily accessible instruction-level parallelism increases overall system

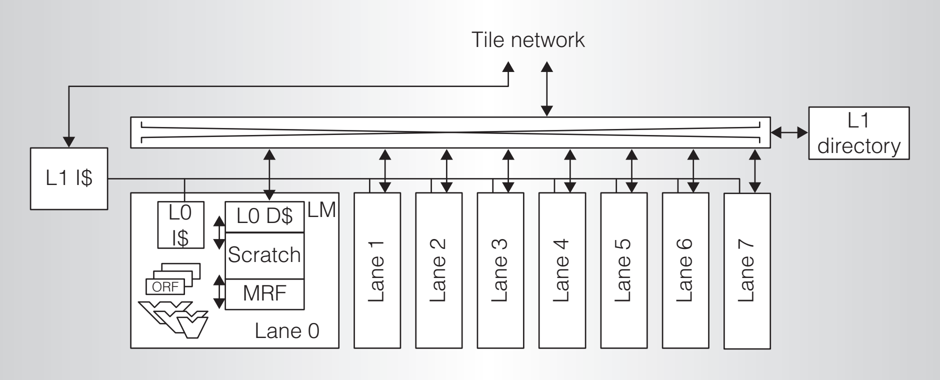

Figure 3. Echelon throughput optimized core. The TOC consists of eight computing lanes, which can be grouped in single-instruction, multiple-data (SIMD) fashion for greater efficiency on data-parallel code. (I$: instruction cache; D$: data cache; ORF: operand register file; MRF: main register file; LM: lane memory.)

concurrency without increasing thread count. However, CPUs' heroic efforts to dynamically extract ILP (such as speculation, out-of-order execution, register renaming, and prefetching) are beyond the needs of a highly concurrent throughput-oriented processor. Instead, Echelon implements a long-instruction-word (LIW) architecture with up to three instructions per cycle, which can include a mix of up to three integer operations, one memory operation, and two double-precision floating-point operations. The instruction-set architecture also supports subword parallelism, such as four single-precision floating-point operations within the two double-precision execution slots. The Echelon compiler statically packs the instructions for a thread, alleviating the hardware from the dynamic power burden. Each lane has a small Level 0 (L0) instruction cache to reduce the energy per instruction access. The lanes in the TOC share a larger L1 instruction cache. The Echelon lane architecture also has three additional features: register file caching, multilevel scheduling, and temporal single-instruction, multiple-thread (SIMT) execution.

Register file caching. If each of the 64 active threads needs 64 registers, the register file capacity per lane is 32 Kbytes. Even with aggressive banking to produce more register file bandwidth, accessing a 32-Kbyte structure as many as four times per cycle per instruction (three DFMA inputs and one output per cycle) presents a large and unnecessary energy burden. Values that are produced and then consumed quickly by subsequent instructions can be captured in a register file cache rather than being written back to a large register file.[7] Echelon employs a software-controlled register file cache called an *operand* register file (ORF) for each of the LIW units to capture producer and consumer operand locality. The Echelon compiler will determine which values to write to which ORF and which to write to the main register file (MRF). Our experiments show that a greedy ORF allocation algorithm implemented as a post-pass to the standard register allocator is a simple and effective way to exploit a multilevel register file hierarchy and reduce register file energy by as much as 55 percent.

Multilevel scheduling. The Echelon system can tolerate short interinstruction latencies from the pipeline and local memory accesses as well as long latencies to remote SRAM and DRAM. Keeping all threads active required maintaining state for all of them in the ORFs (making the ORFs large) and actively selecting among up to 64 threads every cycle. To reduce multithreading's energy burdens, Echelon employs a two-level scheduling system that partitions threads into *active* and *on-deck* sets.[7] In Echelon, four active threads are eligible for execution each cycle and are interleaved to cover short latencies. The remaining 60 on-deck threads have their state loaded on the machine but might be waiting for long-latency memory operations. An Echelon lane evicts a thread from the active set when it encounters the consumer of an instruction that could incur a long latency, such as the consumer of a load requiring a DRAM access. Such consumer instructions are identifiable and marked at compilation time. Promoting a thread to the active set requires only moving the program counter into the active pool. No hardware movement of the threads state is required. Reducing the active set to only four threads greatly simplifies the thread selection logic and can reduce each ORF's total size to 128 bytes, assuming four 64-bit entries per active thread.

Temporal SIMT. Contemporary GPUs organize 32 threads into groups called warps that execute in a SIMD fashion (called SIMT) across parallel execution units. Best performance occurs when threads are coalesced, meaning they execute the same instruction stream (same program counter), access scratch-pad memory without bank conflicts, and access the same L1 data cache line across all threads. When thread control flow or cache addresses diverge, performance drops precipitously. Echelon employs two strategies to make the performance curve smoother when threads diverge, with the goal of high performance and efficiency on both regular and irregular parallel applications. First, each lane can execute in MIMD fashion, effectively reducing the warp width to one thread to handle maximally divergent code.

Second, within a lane, threads can execute independently, or coalesced threads can execute in a SIMT fashion with instructions from different threads sharing the issue slots. When threads in a lane coalesce, an instruction is fetched only once and then used multiple times across multiple cycles for the coalesced threads. Temporal SIMT also enables optimizations in which threads can share common registers so that redundant work across multiple threads can be factored out and scalarized.[8] When threads diverge, a lane can still execute at full computing bandwidth, but with less energy efficiency than when exploiting temporal SIMT and scalarization.

Malleable memory system

Echelon aims to provide energy-efficient data access to applications with a range of characteristics. Applications with hierarchical data reuse, such as dense linear algebra, benefit from a deep caching hierarchy. Applications with a plateau in their working set benefit from a shallow caching hierarchy to capture the entire plateau on chip. When programmers can identify their working sets, they can obtain high performance and efficiency by placing them into physically mapped on-chip storage (scratch pads or shared memory in Nvidia terminology). Regardless of an application's access pattern, the key to both energy and bandwidth efficiency is to limit the volume of data transferred between memory hierarchy levels.

In Echelon, we are investigating a malleable memory system in which the organization of the on-chip SRAM can be configured to match an application's characteristics. The malleable storage consists of the secondary memory banks contained within each throughput computing tile. Figure 2 shows a total of 1,024 256-Kbyte SRAM banks arranged throughout the chip, in addition to the L1 and L2 cache hierarchy contained within the LOCs. Each bank is subdivided into eight 32-Kbyte partitions that can each be configured independently. The hardware can configure a partition to be cached or uncached by adjusting whether the corresponding tag array is accessed. Each partition also has a data structure used to compute a pointer to the partition's parent in the hierarchy. A partition's parent can be a specific cache bank or memory controller, or more than one cache bank or memory controller with the addresses interleaved across the range. These simple mechanisms let an Echelon memory system support various memory hierarchies, including private L2s for each TOC; a shared L2 that spans all the tiles; or a private L2 backed by a global L3, where the L3 is constructed from a subset of the partitions spanning the throughput tiles. The Echelon hardware and runtime system must balance this level of configurability with any optimization complexity burden added to the application programmer. We anticipate that strategies such as auto-tuning can automate the optimizations required to best use the memory hierarchy.[9]

The malleable memory system also enables construction of a hierarchy of scratch pads so that a programming system can control multiple hierarchy levels (as opposed to just the level nearest the processing cores in contemporary GPUs) without having to defeat a hardware-caching algorithm. Caches and scratch pads can also reside simultaneously within the malleable memory. Finally, Echelon provides fine-grained direct memory access to transfer data efficiently between different hierarchy levels without forcing the data through a processor's registers.

Programming-system features

An important and difficult goal of Echelon is to make writing an efficient parallel program as easy as writing an efficient sequential program. Although the strategies for achieving greater productivity in parallel programming are beyond this article's scope, we highlight some key Echelon mechanisms that should facilitate easier development of fast parallel programs.

Unified memory addressing provides an address space spanning LOCs and TOCs on a single chip, as well as across multiple Echelon chips. Removing the burden of managing multiple memory spaces will facilitate the creation of correct parallel programs that can then be successively refined for performance.

Selective memory coherence lets programmers initially place all shared data structures in a common coherence domain, and then subsequently optimize and remove some from the coherence domain. This lets programmers obtain better performance and energy efficiency as more data structures are managed more explicitly, without requiring programmer management of all data structures.

Hardware fine-grained thread creation mechanisms—such as those found in contemporary GPUs for thread array launch, work distribution, and memory allocation—let programmers easily exploit different parallelism structures, including the dynamic parallelism found in applications such as graph search. Reducing the thread management overhead enables parallelization of smaller pieces of work without artificially requiring aggregation into heavy-weight threads.

Fast communication and synchronization mechanisms such as active messages enable dynamic parallelism and thread migration, alleviating the burden of carefully aggregating communication into coarse-grained messages. Fast, flexible barriers among arbitrary groups of threads let programmers control different shapes of threads in a bulk-synchronous manner, rather than just threads that can fit on a single computation unit.

Nvidia's Echelon project is developing architectures and programming systems that aim specifically at the energy-efficiency and locality challenges for future systems. With a target performance of 16 Tflops per chip at 150 W in 10-nm technology, Echelon seeks an energy efficiency of 20 pJ per floating-point operation, which is 11 times better than today's GPU chips. We expect that efficient single-thread execution, coupled with fast interthread communication and synchronization mechanisms, will let an Echelon chip reach a range of applications that exhibit either thread- or data-level parallelism, or both. We are also developing programming systems that include features to exploit hierarchical and nested parallelism and enable an application to abstractly express the known locality between data and computation. These technologies will further facilitate the deployment of parallel systems into energy-sensitive environments such as embedded- and mobile-computing platforms, desktop supercomputers, and data centers. MICRO

Acknowledgments

This research was funded in part by the US government via DARPA contract HR0011-10- 9-0008. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US government. We thank the research staff at Nvidia for their feedback and contributions—in particular, James Balfour and Ronny Krashinsky.

References

1. *NVIDIA CUDA C Programming Guide,* version 4.0, Nvidia, 2011.
2. *The OpenCL Specification,* version 1.1, Khronos OpenCL Working Group, 2010.
3. J. Nickolls and W.J. Dally, ''The GPU Computing Era,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 56-69.
4. P. Kogge et al., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems,* tech. report TR-2008-13, Dept. of Computer Science and Eng., Univ. of Notre Dame, 2008.
5. S. Galal and M. Horowitz, ''Energy-Efficient Floating Point Unit Design,'' *IEEE Trans. Computers,* vol. 60, no. 7, 2011, pp. 913-922.
6. T. Vogelsang, ''Understanding the Energy Consumption of Dynamic Random Access Memories,'' *Proc. Int'l Symp. Microarchitecture,* IEEE CS Press, 2010, pp. 363-374.
7. M. Gebhart et al., ''Energy-Efficient Mechanisms for Managing Thread Context in Throughput Processors,'' *Proc. ACM/IEEE Int'l Symp. Computer Architecture,* ACM Press, 2011, pp. 235-246.
8. S. Collange, D. Defour, and Y. Zhang, ''Dynamic Detection of Uniform and Affine Vectors in GPGPU Computations,'' *Proc. Int'l Conf. Parallel Processing,* Springer-Verlag, 2010, pp. 46-55.
9. R.C. Whaley and J.J. Dongarra, ''Automatically Tuned Linear Algebra Software,'' *Proc. ACM/IEEE Conf. Supercomputing,* IEEE CS Press, 1998, pp. 1-27.

Stephen W. Keckler is the director of architecture research at Nvidia and a professor of computer science and electrical and computer engineering at the University of Texas at Austin. His research interests include parallel-computer architectures, memory systems, and interconnection networks. Keckler has a PhD in computer science from the Massachusetts Institute of Technology. He's a fellow of IEEE and a senior member of the ACM.

William J. Dally is the chief scientist at Nvidia and the Willard R. and Inez Kerr Bell Professor of Engineering at Stanford University. His research interests include parallel-computer architectures, interconnection networks, and high-speed signaling circuits. Dally has a PhD in computer science from the California Institute of Technology. He's a member of the National Academy of Engineering and a fellow of IEEE, the ACM, and the American Academy of Arts and Sciences.

Brucek Khailany is a senior research scientist at Nvidia. His research interests include energy-efficient architectures and circuits, cache and register file hierarchies, VLSI design methodology, and computer arithmetic. Khailany has a PhD in electrical engineering from Stanford University. He's a member of IEEE and the ACM.

Michael Garland is the senior manager of programming systems and applications research at Nvidia. His research interests include parallel-programming systems, languages, and algorithms. Garland has a PhD in computer science from Carnegie Mellon University. He's a member of the ACM.

David Glasco is a senior research manager at Nvidia. His research interests include high-performance computer systems, interconnection networks, memory system architectures, computer graphics, and cache-coherent protocols. Glasco has a PhD in electrical engineering from Stanford University. He's a member of IEEE.

Direct questions and comments about this article to Stephen W. Keckler, Nvidia, 2701 San Tomas Expressway, Santa Clara, CA 95050; skeckler@nvidia.com.