RIGEL: A 1,024-CORE SINGLE-CHIP ACCELERATOR ARCHITECTURE

RIGEL IS A SINGLE-CHIP ACCELERATOR ARCHITECTURE WITH 1,024 INDEPENDENT PROCESSING CORES TARGETED AT A BROAD CLASS OF DATA- AND TASK-PARALLEL COMPUTATION. THIS ARTICLE DISCUSSES RIGEL'S MOTIVATION, EVALUATES ITS PERFORMANCE SCALABILITY AS WELL AS POWER AND AREA REQUIREMENTS, AND EXPLORES MEMORY SYSTEMS IN THE CONTEXT OF 1,024-CORE SINGLE-CHIP ACCELERATORS. THE AUTHORS ALSO CONSIDER FUTURE OPPORTUNITIES AND CHALLENGES FOR LARGE-SCALE DESIGNS.

Daniel R. Johnson
Matthew R. Johnson
John H. Kelm
William Tuohy
Steven S. Lumetta
Sanjay J. Patel
University of Illinois at Urbana-Champaign

Increasing demand for performance on data-intensive parallel workloads has driven the design of throughput-oriented parallel *compute accelerators*. For this work, we consider programmable accelerators in contrast to fixed-function or hardwired application-specific accelerator units. Current programmable accelerators generally expose restricted programming models that yield high performance for data-parallel applications with regular computation and memory-access patterns, but present a more difficult target for less-regular parallel applications. Generally, existing compute accelerators provide higher throughput via architectural choices that compromise the generality of the programming model. For instance, accelerators commonly achieve high throughput by using wide single-instruction, multiple-data (SIMD) processing elements, as opposed to the multiple-instruction, multiple-data (MIMD) model used in general-purpose processors. For dense or regular data-parallel computations, SIMD hardware reduces the cost of performing many computations by amortizing costs such as control and instruction fetch across many processing elements. However, when applications don't naturally map to the SIMD execution model, programmers must adapt their algorithms or suffer reduced efficiency. SIMD then limits the scope of applications that can achieve the hardware's peak performance. The memory system is another area where accelerators commonly favor hardware efficiency over programmability. Programmer-managed scratchpad memories yield denser hardware and tighter access-latency guarantees and consume less power than caches; however, they impose an additional burden on either the programmer or software tools. Additionally, the multiple address spaces often associated with scratchpad memories require copy operations and explicit memory management.

We conceived of the Rigel architecture in 2007 to address some shortcomings of parallel computation accelerators while pushing the envelope on throughput-oriented designs. (For more information on throughput-oriented designs, see the "Throughput Processors" sidebar.) Broadly, Rigel's goals

Throughput Processors

We consider two broad classes of throughput-oriented architectures: general-purpose chip multiprocessors (CMPs) and specialized accelerators. For this work, we consider programmable accelerators in contrast to fixed-function or hardwired application-specific logic. Contemporary general-purpose CMP development is driven by the need to increase performance while supporting a vast ecosystem of existing multitasking operating systems, programming models, applications, and development tools. Increasingly, CMPs are integrating additional functionality such as memory controllers and peripheral controllers on die, converging on an SoC model. Accelerators are designed to maximize performance for a specific class of workloads by exploiting characteristics of the target domain, and are optimized for a narrower class of workloads and programming styles. Although general-purpose processors tend to employ additional transistors to decrease latency along a single execution thread, accelerators are architected to maximize aggregate system throughput, often with increased latency for any particular operation.

Throughput-oriented CMPs such as Sun's UltraSPARC T3[1] target server workloads with a moderate number of simple, highly multithreaded cores. Such CMPs achieve relatively high throughput but are limited by low memory bandwidth relative to contemporary graphics processors and the per-core features and latency-reduction techniques required to meet the needs of general-purpose workloads.

Tilera's most recent processors,[2] based on earlier work on RAW,[3] feature up to 100 cache-coherent cores in a tiled design with a mesh interconnect optimized for message passing and streaming applications. Although they're Linux capable, Tilera's chips have poor floating-point capability. Intel has developed several experimental mesh-based throughput processors, including the 1-Tflop 80-core chip[4] and the 48-core single-chip cloud computer.

The Cell processor,[5] introduced with the PlayStation 3 and also used in high-performance computing, uses a heterogeneous model with a multithreaded PowerPC processor and up to eight synergistic processing elements (SPEs) as coprocessors. SPEs use a programmer-managed scratchpad for both instruction and data access and implement a SIMD instruction set.

Stream processors are programmable accelerators targeted at media and signal processing workloads with regular, predictable data-access patterns. Imagine pioneered the stream processing concept,[6] and stream processing has influenced modern graphics processor designs.

GPUs are the most prominent class of programmable accelerators at present. GPUs from Nvidia[7] and AMD are targeted primarily at the graphics rendering pipeline, but have exposed an increasingly flexible substrate for more general-purpose data-parallel computations. For instance, compared with the first CUDA-capable GPUs, Nvidia's latest Fermi GPUs include a cached memory hierarchy and accelerated atomic operations, support execution of multiple concurrent kernels, and reduce the performance penalty for memory-gather operations.

Both Nvidia and AMD GPUs utilize single-instruction, multiple-thread (SIMT)-style architectures, whereby a single instruction simultaneously executes across multiple pipelines with different data. A key aspect of SIMT designs is their ability to allow control divergence for branching code, whereby only a subset of a SIMT unit's pipelines are active. Such designs enable dense hardware with high peak throughput but work best for regular computations with infrequent divergence. Both Nvidia and AMD rely on programmer-managed scratchpad memories and explicit data transfers for high performance, though modern variants do implement small noncoherent caches. GPUs also rely on high memory bandwidth and thousands of hardware threads to hide memory latency and maximize throughput.

Intel's Larrabee project approached the accelerator design point with a fully programmable many-core x86 design, cached memory hierarchy, and hardware cache coherence.[8] Like Rigel, Larrabee was intended for more general-purpose parallel programming. Larrabee and Intel's AVX extensions support wide SIMD vectors for parallel processing. These wide vectors represent a vastly different design point than the independent scalar cores of Rigel or the SIMT units of GPUs, requiring additional programmer or compiler effort for packing and alignment.

References

1. J. Shin et al., ''A 40 nm 16-Core 128-Thread CMT SPARC SoC Processor,'' *Proc. IEEE Int'l Solid-State Circuits Conf.,* IEEE Press, 2010, pp. 98-99.
2. S. Bell et al., ''Tile64 Processor: A 64-Core SoC with Mesh Interconnect,'' *Proc. IEEE Int'l Solid-State Circuits Conf.,* IEEE Press, 2008, pp. 88-598.
3. M.B. Taylor et al., ''The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs,'' *IEEE Micro,* vol. 22, no. 2, 2002, pp. 25-35.
4. S. Vangal et al., ''An 80-Tile 1.28 Tflops Network-on-Chip in 65 nm CMOS,'' *Proc. IEEE Int'l Solid-State Circuits Conf.,* IEEE Press, 2007, pp. 98-99, 589.
5. M. Gschwind, ''Chip Multiprocessing and the Cell Broadband Engine,'' *Proc. 3rd Conf. Computing Frontiers,* ACM Press, 2006, pp. 1-8.
6. S. Rixner et al., ''A Bandwidth-Efficient Architecture for Media Processing,'' *Proc. 31st Ann. ACM/IEEE Int'l Symp. Microarchitecture,* IEEE CS Press, 1998, pp. 3-13.
7. ''Nvidia's Next Generation CUDA Compute Architecture: Fermi,'' white paper, Nvidia, 2009.
8. L. Seiler et al., ''Larrabee: A Many-Core x86 Architecture for Visual Computing,'' *ACM Trans. Graphics,* vol. 27, no. 3, 2008, article 18.

are to demonstrate the feasibility of a single-chip, massively parallel MIMD accelerator architecture; to achieve high computation density, or throughput, in terms of $\left(\frac{operations/sec}{mm^2}\right)$; and to determine how to organize such a device to be programmer-friendly, presenting a more general target to developers and enabling a broader range of parallel applications to target the design.

These goals drove our development of Rigel, a 1,024-core single-chip accelerator architecture that targets a broad class of data- and task-parallel computations, especially visual computing workloads.[1] With the Rigel design, we aim to strike a balance between raw performance and ease of programmability by adopting programming interface elements from general-purpose processors. Rigel comprises 1,024 independent, hierarchically organized cores that use a fine-grained, dynamically scheduled single-program, multiple-data (SPMD) execution model. Rigel adopts a single global address space and a fully cached memory hierarchy. Parallel work is expressed in a task-centric, bulk-synchronized manner using minimal hardware support. Compared to existing accelerators—which contain domain-specific hardware, specialized memories, and restrictive programming models—Rigel is more flexible and provides a more straightforward target for a broader set of applications.

The design of Rigel's memory system, particularly cache coherence, shaped many other aspects of the architecture. We observed that we could leverage data sharing and communication patterns in parallel workloads in designing memory systems for future many-core accelerators. Using these insights, we developed software and hardware mechanisms to manage coherence on parallel accelerator processors. First, we developed the Task-Centric Memory Model (TCMM), a software protocol that works in concert with hardware caches to maintain a coherent, single-address-space view of memory without the need for hardware coherence.[2] Although we originally developed Rigel without global cache coherence, we ultimately found that we could implement hardware-managed cache coherence (HW_{cc}) with low overhead for thousand-core processors. Thus, we developed WayPoint, a scalable hardware coherence solution for Rigel.[3] Finally, we developed Cohesion as a bridge enabling effective use of both hardware and software coherence mechanisms, simplifying the integration of multiple memory models in heterogeneous or accelerator-based systems.[4]

Rigel system architecture

Rigel is a MIMD compute accelerator developed to target task- and data-parallel visual computing workloads that scale up to thousands of concurrent tasks.[1] Rigel's design objective is to provide high-compute density while enabling an easily targeted conventional programming model. Figure 1 shows a block diagram of Rigel.

Rigel's basic processing element is an area-optimized, dual-issue, in-order core with a 32-bit reduced-instruction-set computing architecture, a single-precision floating-point unit (FPU), and an independent fetch unit. Eight independent cores and a shared cluster cache compose a single Rigel cluster. Clusters allow efficient communication among their cores via the shared cluster cache. Clusters are grouped logically into a tile using a bidirectional tree-structured interconnect. We selected a tree-structured interconnect based on our use model; the interconnect links the cores to memory but does not enable arbitrary core-to-core communication. Eight tiles of 16 clusters each are distributed across the chip, attached to global cache banks via a multistage switch interconnect. The last-level globally shared cache provides buffering for multiple high-bandwidth memory controllers. Our baseline 1,024-core design incorporates eight 32-bit Graphics Double Data Rate (GDDR5) memory controllers and 32 global cache banks. Table 1 summarizes our design parameters.

Memory and cache management

All cores on Rigel share a single global address space. Cores within a cluster have the same view of memory via the shared cluster cache, while cluster caches aren't explicitly kept coherent with one another in our baseline architecture. When serialization of accesses is necessary between clusters, the global cache is the point of coherence. Rigel implements two classes of memory operations: local and global.

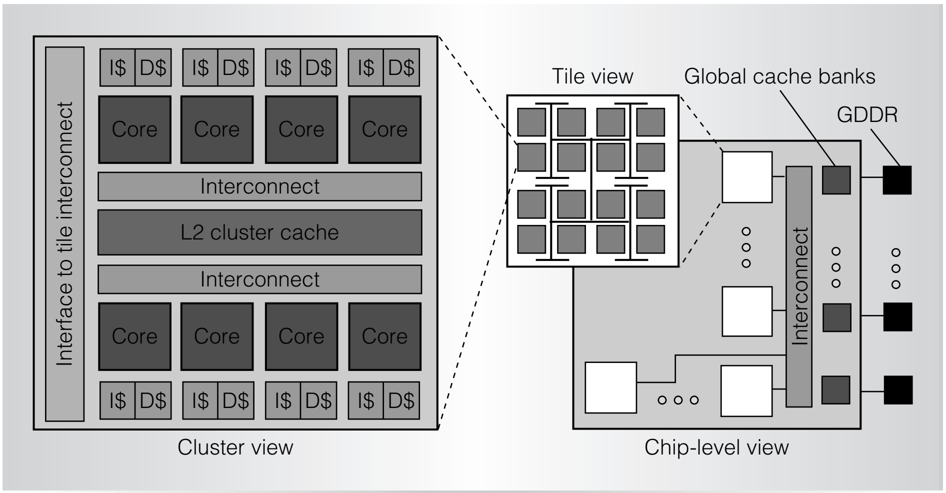

Figure 1. Block diagram of the Rigel accelerator processor: cluster view (a) and chip-level view (b). The Rigel cluster is a collection of cores with a shared cache. Clusters are organized into tiles to share interconnect, and tiles are connected to a last-level global cache. (GDDR: Graphics Double Data Rate.)

Local memory operations constitute the majority of loads and stores and are the default memory-operation class the compiler generates. Memory locations accessed by local operations are cacheable at the cluster cache but aren't kept coherent by hardware between clusters. Local memory operations are used for accessing read-only data, private data, and data shared within the same cluster.

Global loads and stores on Rigel always bypass cluster-level caches and complete at the global cache. Memory locations operated solely by global memory operations are thus kept trivially coherent across the chip. Global operations are key for supporting system resource management, synchronization, and fine-grained intercluster communication. However, because of increased latency, the cost of global memory operations is high relative to local operations. Rigel also implements a set of atomic operations (arithmetic, bitwise, min/max, and exchange) that complete at the global cache.

In the baseline Rigel architecture, software must enforce coherence when intercluster read-write sharing exists. We can do this by colocating sharers within a single coherent cluster, by using global memory accesses for shared data, or by forcing the writer to explicitly flush shared data before allowing the reader to access it. Rigel provides explicit instructions for actions such as flushing and eviction for cache management. We explore the topic of coherence in more detail later.

Table 1. Simulated parameters for the Rigel architecture.

Component	Characteristics
Cores	1,024, two-wide issue, in order
DRAM	Eight 32-bit channels, GDDR5, 6 Gbyte/s/pin, 192 Gbyte/s total
Level 1 (L1) instruction cache	2 Kbytes, two-way associative
L1 data cache	1 Kbyte, two-way associative Kbyte
L2 cluster cache	Unified, one per 8-core cluster, 64 Kbytes, 16-way associative
L3 global cache	Unified, globally shared, 4 Mbytes total, 32 banks, 8-way associative

Area and power

To demonstrate Rigel's feasibility on current process technology, we provide area and power estimates on a commercial 45-nm process. Our estimates are derived from synthesized Verilog, compiled static RAM (SRAM) arrays, IP components, and die plot analysis of other 45-nm designs. Figure 2 shows a breakdown of preliminary area estimates for the Rigel design. Cluster caches are 64 Kbytes each, and global cache banks total 4 Mbytes. "Other logic" encompasses

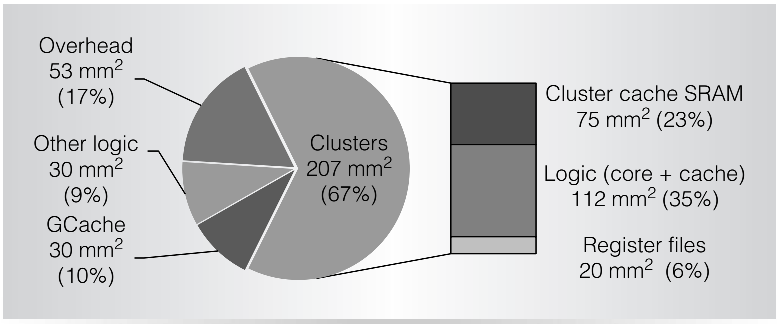

Figure 2. Area estimates for the Rigel design. Our estimate includes cluster cache, logic, register files, overhead, other logic, and global cache.

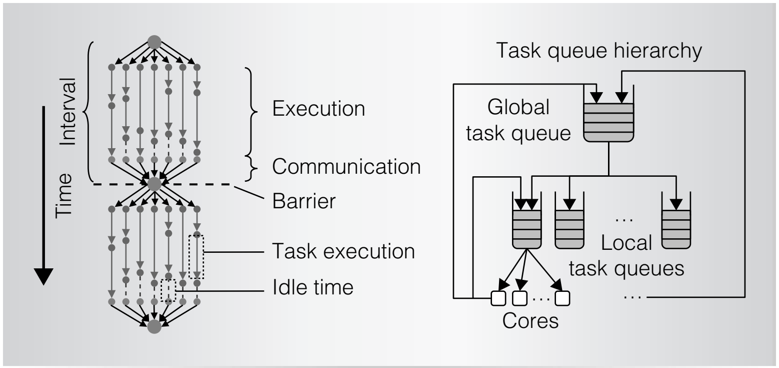

Figure 3. The bulk-synchronous parallel (BSP) execution model (a) and hierarchical task queues (b) that constitute the Rigel Task Model.

Table 2. Data- and task-parallel workloads.

Benchmark	Description
`cg`	Conjugate gradient linear solver
`dmm`	Blocked dense-matrix multiplication
`fft`	2D complex-to-complex radix-2 fast Fourier transform
`gjk`	Gilbert-Johnson-Keerthi 3D collision detection
`heat`	2D 5-point iterative, out-of-place stencil computation
`kmeans`	*K*-means clustering
`march`	Marching cubes polygonization of 3D volumetric data
`mri`	Magnetic resonance image reconstruction ($F^H D$ matrix)
`sobel`	Sobel edge detection
`stencil`	3D 7-point iterative, out-of-place stencil computation

interconnect as well as memory controller and global cache controller logic. For a conservative estimate, we include a 20-percent charge for additional overhead. The resulting 320 mm^2 is reasonable for implementation in current process technologies and leaves space for additional SRAM cache or more aggressive memory controllers. Table 1 summarizes the chip parameters.

Typical power consumption of the design with realistic activity factors for all components at 1.2 GHz is expected to be in the range of 99 W, though peak power consumption beyond 100 W is possible. Our estimate is based on power consumption data for compiled SRAMs, postsynthesis power reports for logic, leakage, and a clock tree of cluster components, estimates for interconnect and I/O pin power, and the 20-percent charge for additional power overhead. The figure is similar to modern GPUs from Nvidia that consume around 150 W,[5] while modern high-end CPUs can consume nearly as much.

Programming Rigel

Rigel isn't restricted to running software written in a particular hardware-specific paradigm, but instead has the ability to run standard C code. We target Rigel using the Low-Level Virtual Machine (LLVM) compiler framework and a custom back end. Rigel applications are developed using the Rigel Task Model (RTM), a simple bulk-synchronous parallel (BSP), task-based work distribution library that we developed. Applications are written in RTM using an SPMD execution paradigm, where all cores share an application binary with arbitrary control flow per core. The programmer defines parallel work units, referred to as *tasks*, that are managed via queues by the RTM runtime. RTM task queues can act as barriers when empty to provide global synchronization points. Figure 3 illustrates the BSP model we implement along with our hierarchical task queues.

Scalability

We evaluate Rigel using various parallel applications and kernels drawn from visual and scientific computing. Most benchmarks are written in a bulk-synchronous style using RTM for dynamic work distribution, but `stencil` statically allocates work to threads. While all of our applications exhibit abundant data parallelism, the structure varies from dense (`dmm`, `sobel`) to sparse (`cg`) to irregular task-parallel (`gjk`) and includes diverse communication patterns (`kmeans`, `fft`, `heat`, `stencil`). Table 2 describes our set of benchmark codes. Figure 4

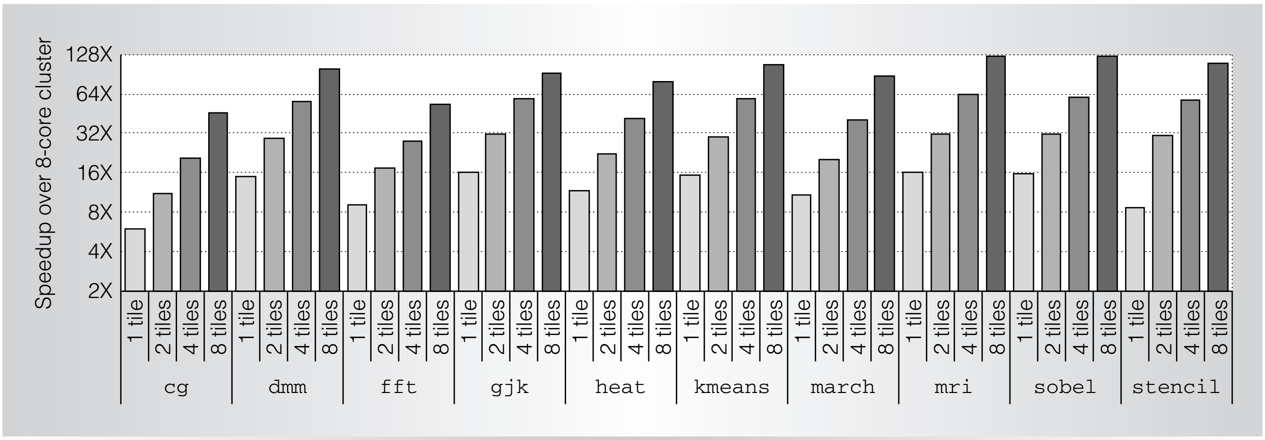

Figure 4. Benchmark scalability on Rigel with one, two, four, and eight 128-core tiles (128 to 1,024 cores). Speedups are relative to one eight-core cluster. The 128× represents linear scaling at 1,024 cores. Benchmark binaries and data sets are identical across all system sizes; global cache capacity and memory bandwidth are scaled with the number of tiles.

illustrates kernel scalability for various parallel applications up to 1,024 cores. Across our selection of benchmarks, we observe an average speedup of 84× (harmonic mean) at 1,024 cores compared to one eight-core cluster (128× speedup is ideal). For a more thorough evaluation of the baseline Rigel architecture, see our previous work.[1]

Coherence and memory system

A high-performance accelerator requires efficient mechanisms for safely sharing data between multiple caches. Scalable multiprocessor hardware coherence schemes[6] exist, but were designed for machines with a different mix of computation, communication, and storage resources than chip multiprocessors (CMPs). Indeed, modern general-purpose CMPs and multisocket systems generally use much simpler protocols that work well for small systems but are cost-prohibitive for a 1,024-core accelerator. (For more information on cache coherence, see the "Related Work in Coherence" sidebar.)

Additionally, our target applications exhibit data-sharing patterns that are more structured than those targeted by traditional distributed machines and CMPs. While our initial design goal for Rigel was to achieve good performance and programmability without HW_{cc}, we have since examined ways to achieve the benefits of hardware coherence with reduced overhead by leveraging our target workloads' sharing characteristics.

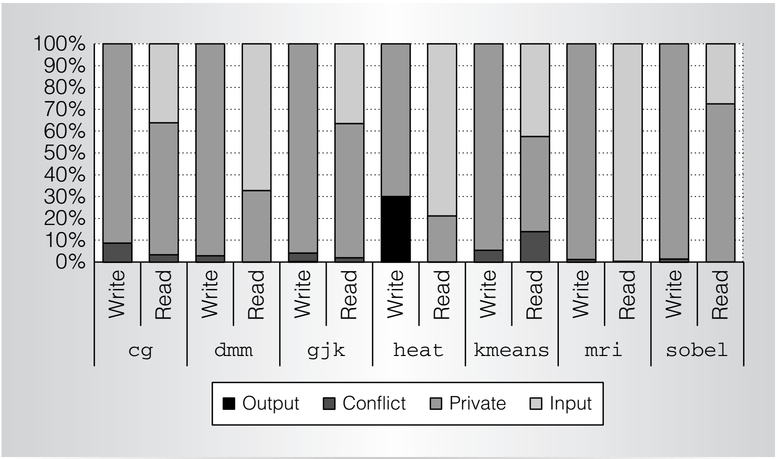

Figure 5. Characterizing memory accesses in task-based BSP applications. Input reads and output writes communicate data across barriers. The majority of memory accesses are to data that is private to a task. Conflict accesses share data between two tasks in the same barrier interval, requiring hardware coherence or synchronization mechanisms such as atomic operations to maintain correctness, but these are rare in the applications we examine.

The design of Rigel's memory system is informed by the sharing and communication patterns of the parallel workloads targeted by accelerators. In studying several such applications written for two different platforms (x86/pthreads and Rigel/RTM), we found that structured, coarse-grained sharing patterns are common, that most sharing occurs across global synchronization points (*barriers*), and that fine-grained data sharing between barriers

Related Work in Coherence

A rich and diverse set of related work exists on the topic of cache coherence.[1] Researchers developed distributed shared memory as a scalable way to provide the illusion of a single, coherent address space across multiple disjoint memories.[2] Various mechanisms such as that by Zebchuk et al.[3] have been proposed for reducing directory overhead on CMPs, and novel optical networks have been proposed for 1,000-core cache-coherent chips.[4] The Smart Memories project has examined programmable controllers that can implement various on-chip memory models including cache coherence,[5] while Cohesion enables the management of multiple coherence domains simultaneously. A more complete treatment of related work on memory models and cache coherence can be found in our earlier work.[6-8]

References

1. D.J. Lilja, ''Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons,'' *ACM Computing Surveys,* vol. 25, no. 3, 1993, pp. 303-338.
2. J. Hennessy, M. Heinrich, and A. Gupta, ''Cache-Coherent Distributed Shared Memory: Perspectives on Its Development and Future Challenges,'' *Proc. IEEE,* vol. 87, no. 3, 1999, pp. 418-429.
3. J. Zebchuk et al., ''A Tagless Coherence Directory,'' *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture,* ACM Press, 2009, pp. 423-434.
4. G. Kurian et al., ''ATAC: A 1000-Core Cache-Coherent Processor with On-Chip Optical Network,'' *Proc. 19th Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, pp. 477-488.
5. A. Firoozshahian et al., ''A Memory System Design Framework: Creating Smart Memories,'' *Proc. 36th Ann. Int'l Symp. Computer Architecture,* ACM Press, 2009, pp. 406-417.
6. J.H. Kelm et al., ''A Task-Centric Memory Model for Scalable Accelerator Architectures,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* IEEE CS Press, 2009, pp. 77-87.
7. J.H. Kelm et al., ''Cohesion: A Hybrid Memory Model for Accelerators,'' *Proc. Int'l Symp. Computer Architecture,* ACM Press, 2010, pp. 429-440.
8. J.H. Kelm et al., ''Waypoint: Scaling Coherence to 1000-Core Architectures,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, pp. 99-110.

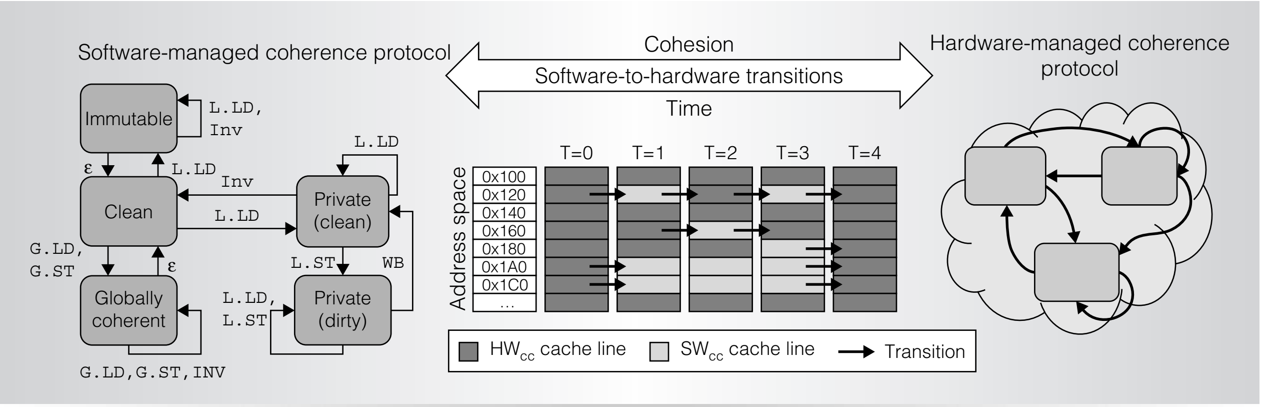

Figure 6. Cohesion is a hybrid memory model for accelerators that enables hardware-managed cache coherence (HW_{cc}) and software-managed cache coherence (SW_{cc}) to coexist, with data migrating between the two domains dynamically. On the left, state transitions for a cache line under control of the Task-Centric Memory Model (TCMM) protocol. (G: global operations; L: local operations; WB: writebacks; INV: invalidates; ∈: no sharers; LD: loads; and ST: stores.)

is uncommon.[2] Figure 5 illustrates the sharing patterns in several of our workloads.

Software-managed coherence

Adopting a structured programming model lets us implement software-managed cache coherence (SW_{cc}) efficiently. We developed TCMM as a contract describing the software actions necessary to ensure correctness in task-based BSP programs in the absence of hardware-enforced coherence.[2] The left side of Figure 6 illustrates the state transitions for a cache line in our protocol. The coherence state of a block is implicit and must be tracked by the programmer or runtime system. All blocks start in the `clean` state with no sharers or cached copies and can transition to `immutable` (read-only), shared as `globally coherent`, or `private`. Cached *local* memory operations may operate on `private` or `immutable` data, whereas uncached *global* operations are required for `globally coherent` data. Transitioning data between states requires first moving through the `clean` state.

SW_{cc} requires minimal hardware support in the form of instructions for explicitly writing back and invalidating data in private caches. We found that a small number of additional hardware mechanisms, such as broadcast support to accelerate global barriers and global atomic operations to facilitate infrequent intrabarrier sharing, greatly improved scalability over a naive design. With these relatively inexpensive mechanisms, SW_{cc} could achieve performance within a few percent of idealized hardware coherence at 1,024 cores. Future accelerators might improve upon TCMM by automating coherence actions in the compiler and scheduling coherence actions to maximize cache locality. Our earlier work describes the TCMM protocol and provides a detailed performance analysis.[2]

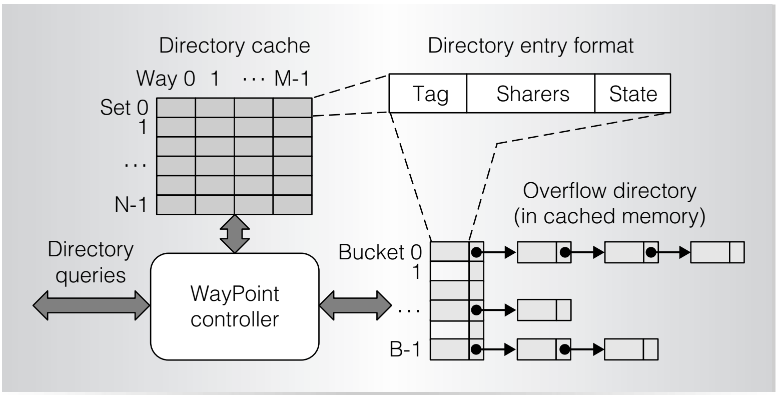

Figure 7. WayPoint consists of a directory cache bank, a hardware controller, and a cached, in-memory hash table of directory entries associated with each global cache bank. The hash table, known as the overflow directory, acts as a backing store for the directory cache, providing increased directory associativity when needed with lower storage requirements than a cached full directory.

Hardware-managed coherence

Though SW_{cc} can provide high performance and hardware efficiency for many parallel applications, two important benefits motivate the investigation of hardware coherence. First, applications with fine-grained or unpredictable sharing benefit from hardware coherence because of the removal of software coherence instructions. Second, when acceptable performance can be achieved, hardware coherence decreases the programmer's burden by implicitly handling data movement between caches. We developed WayPoint,[3] a directory-based hardware coherence scheme that exploits the characteristics of our architecture and applications to achieve scalable performance and low overhead for up to 1,024 cores.

Through our analysis of several existing coherence schemes on Rigel, we determined that set-associative on-chip directory caches are more scalable than duplicate tag, snoopy, or in-cache directory approaches for our workloads. When the directory cache overflows, either complete sharing information must be preserved or the system must act conservatively the next time the evicted line is accessed. The most common methods of dealing with directory cache overflows are to immediately invalidate all sharers of the line, or to simply drop the directory entry and send a broadcast invalidation to all caches the next time the evicted line is accessed (also known as *probe filtering*). For applications with widely shared data, both approaches result in on-chip coherence traffic that scales roughly as the square of the number of coherent caches, impeding overall scalability.

WayPoint vastly decreases the cost of a directory cache eviction by spilling the evicted entry to a hardware-managed in-memory hash table, eliminating the need for eviction-related broadcasts. Figure 7 illustrates the WayPoint microarchitecture. Directory entries in the hash table can be held on-die along with application data in the last-level cache. WayPoint dynamically provides the illusion of increased associativity for those directory sets that require it. WayPoint is more efficient than statically provisioning the directory cache for worst-case demand because the group of directory cache sets that require high associativity varies widely across applications and over time. We find that WayPoint with a four-way associative directory caches

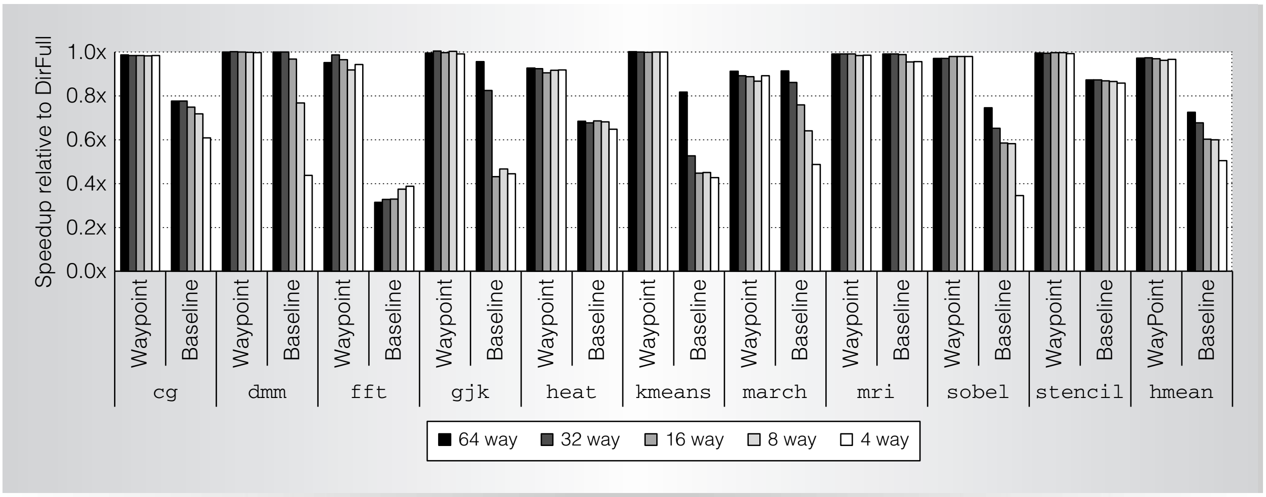

Figure 8. Performance of WayPoint and a conventional sparse directory cache (*Dir*$_4$*B*) versus directory cache associativity. All results are for a 1,024-core system with 2,048 directory entries per global cache bank. Results are normalized to an infinite on-die directory. By avoiding expensive global broadcasts on directory insertion or eviction, WayPoint provides robust performance without requiring a highly associative directory cache, reducing area and power costs.

outperforms a 64-way conventional directory cache (see Figure 8). Across our benchmarks, WayPoint achieves performance within 4 percent of an infinitely large on-die directory while adding less than 3 percent to the total die area.

Hybrid coherence

Memory models in use today are either fully hardware coherent or fully software coherent. In systems that include both models, the two models are strictly separated by using disjoint address spaces or physical memories. As systems on chip (SoCs) and other heterogeneous platforms become more prevalent, the ability to seamlessly manage data across different memory models will become increasingly important.

SW_{cc} removes the area, power, and interconnect traffic overhead of cache coherence for structured data sharing patterns and allows experienced application developers to achieve high performance. Hardware coherence avoids the instruction overhead of software coherence, performs well with unstructured sharing patterns, and provides correct data sharing with low programmer effort. To achieve the combined benefits of these two models, we have developed Cohesion, a hybrid memory model.

Cohesion includes a hardware coherence implementation which tracks the entire address space by default. The developer can selectively remove cache lines from the HW_{cc} domain at runtime and manage them using software to improve performance. Because data can move back and forth between the SW_{cc} and HW_{cc} domains at will, Cohesion can be used to dynamically adapt to the sharing needs of applications and runtimes and does not require multiple address spaces nor explicit copy operations. Cohesion can also enable the integration of multiple memory models in heterogeneous or accelerator-based systems such as SoCs. Figure 6 illustrates the high-level operation of Cohesion. We implement SW_{cc} using TCMM and use an MSI-based hardware coherence protocol, but any hardware and software protocols could be used so long as the necessary state transitions are enforced.

A developer can instruct the hardware coherence machinery to defer to software management for a particular cache line by updating a software-accessible table in memory. For instance, hardware coherence management is inefficient when data is private or when a large amount of data can be managed as a unit by software. Handling read-mostly and private data outside the scope of the hardware coherence protocol can increase performance and reduce the load on the coherence hardware, increasing the effective directory size for data managed under

HW_{cc} (see Figure 9). Ultimately, Cohesion allows explicit coherence management to be an optional optimization opportunity, rather than necessary for correctness.

Looking forward

When we conceived of the Rigel accelerator architecture in 2007, we concerned ourselves with the technology parameters and constraints of that time. Although large dies with billions of transistors were possible in 45-nm technology, our target of 1,024 cores was very aggressive and led to sacrifices in our design. We developed Rigel as a coprocessor for parallel computation, and as an alternative to GPUs, rather than as a complete system. We limited ourselves to 32-bit data paths and single-precision FPUs to limit the die area. As process technology marches on, these limitations can be addressed, though ultimately at the expense of throughput.

Rigel's architecture achieves its scale in part because of what it omits compared to GPUs. While GPUs expend substantial die area on graphics-specific hardware, Rigel repurposes this die area for features such as caches and independent processing cores. (Nvidia's Tesla die is approximately 25 percent stream processing units, while roughly half the die area is dedicated to graphics-related hardware such as texture and raster operations.[5]) Ultimately, Rigel makes choices that aim to increase generality. Further work is merited to drive down the cost of Rigel's MIMD hardware compared to efficient SIMD hardware.

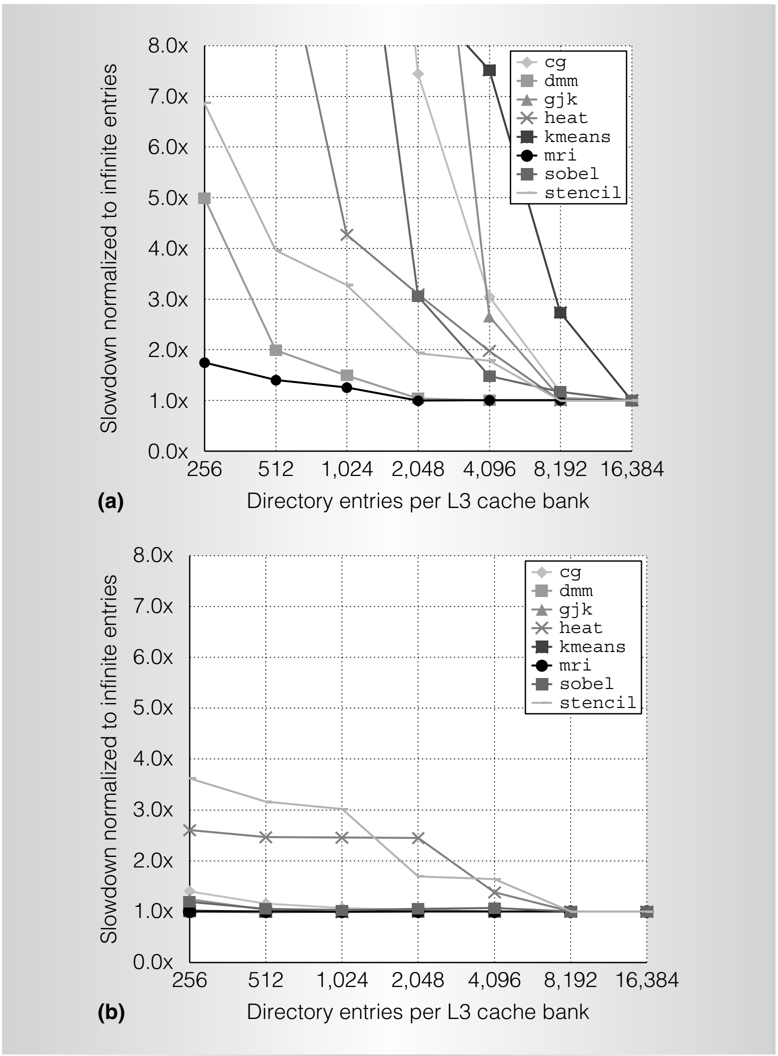

Figure 9. Performance versus directory cache size for HW_{cc} alone (a) and for Cohesion (b). Cohesion amplifies effective directory size by removing data easily tracked by software from the hardware coherence protocol.

Power

We chose an aggressive design point for Rigel and show that it's feasible in current process technology within a reasonable power budget, on par with GPUs of similar peak throughput.[5] We made architectural choices generally favorable to power efficiency, including the use of small, nonspeculative in-order cores and moderate clock targets. Although industry researchers have advocated future thousand-core chips,[7] power consumption for future large-scale processors nevertheless remains a concern.[8] Although some studies portend challenges for scaling multicore processors in a power-constrained world of dark silicon, others find that thousands of cores are reasonable for highly parallel workloads such as ray tracing. Future massively parallel processors such as Rigel will likely need to conserve power through various techniques at multiple levels of the technology stack, including process technology, circuits, architecture, and software.

Off-chip bandwidth

Although on-chip transistor and bandwidth budgets will likely increase along with Moore's law, off-chip memory bandwidth will increase more slowly,[8] becoming a scalability bottleneck for many applications.

High-bandwidth off-chip memory systems are also a major source of power consumption in high-performance systems, requiring more than 30 W to meet modern GPU systems' bandwidth demands. Future accelerators will require careful consideration of data locality at all levels of the memory hierarchy to make optimal use of limited off-chip bandwidth. Emerging technologies such as optical off-chip interconnect or 3D die stacking with through-silicon vias could provide additional bandwidth to future designs.

System software support

Like GPUs, Rigel was originally conceived as a coprocessor, complementary to a general-purpose CPU. Both omit system-level support required for features such as resource virtualization, multiprogramming, process isolation, and resource management. Recent GPUs implement a form of virtual memory and allow execution of multiple concurrent kernels in space, although they generally can't be time- or space-multiplexed efficiently among multiple applications. When required, a host processor must emulate unimplemented functionality at a significant cost. Future accelerators will need to integrate into systems as first-class entities, manage their own resources when possible, and provide the safety and portability guarantees that enhance programmer productivity and robustness on general-purpose processors. Providing this additional functionality while maintaining an accelerator's performance characteristics is an important problem.

A tale of two laws

Amdahl's law states that overall system performance is ultimately limited by the serial portion of a problem. As additional processing elements are added, performance levels off. Gustafson's law[9] is a counterpoint to Amdahl's, arguing that as more parallel resources become available, larger problem sizes become feasible. Rather than becoming dominated by the serial portion of a problem, the parallel portion expands to exploit additional processing capability.

One possible future for designs such as Rigel is as a parallel computation fabric for heterogeneous systems. Sufficient die area is now available such that multiple high-performance CPUs can be integrated onto the same die as Rigel. This strategy allows a few latency-optimized cores to handle operating systems, serial code, and latency-sensitive code while offloading parallel work to a more efficient computational substrate. Evidence of such an approach can already be seen in the embedded SoC space, where consumer-oriented visual computing applications are driving the demand for increased performance. AMD's Fusion and Intel's on-die integration of GPUs are also a step in this direction.

With the transistor count afforded by modern manufacturing processes and large available die sizes, a 1,024-core processor is feasible in today's technology. Exploiting sharing patterns in our target applications and co-optimizing hardware, system software, and applications has enabled the Rigel architecture to scale to 1,024 cores. Our evaluation of accelerator memory systems has given rise to software-managed, hardware-managed, and hybrid memory models that enable accelerator architects to match the memory model to their needs. MICRO

References

1. J.H. Kelm et al., ''Rigel: An Architecture and Scalable Programming Interface for a 1,000-Core Accelerator,'' *Proc. 36th Ann. Int'l Symp. Computer Architecture,* ACM Press, 2009, pp. 140-151.
2. J.H. Kelm et al., ''A Task-Centric Memory Model for Scalable Accelerator Architectures,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* IEEE CS Press, 2009, pp. 77-87.
3. J.H. Kelm et al., ''Waypoint: Scaling Coherence to 1,000-Core Architectures,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* ACM Press, 2010, pp. 99-110.
4. J.H. Kelm et al., ''Cohesion: A Hybrid Memory Model for Accelerators,'' *Proc. Int'l Symp. Computer Architecture,* ACM Press, 2010, pp. 429-440.
5. E. Lindholm et al., ''Nvidia Tesla: A Unified Graphics and Computing Architecture,'' *IEEE Micro,* vol. 28, no. 2, 2008, pp. 39-55.

6. J. Laudon and D. Lenoski, "The SGI Origin: A CCNUMA Highly Scalable Server," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* ACM Press, 1997, pp. 241-251.
7. S. Borkar, "Thousand Core Chips: A Technology Perspective," *Proc. 44th Ann. Design Automation Conf.,* ACM Press, 2007, pp. 746-749.
8. *International Technology Roadmap for Semiconductors,* tech. report, ITRS, 2009.
9. J.L. Gustafson, "Reevaluating Amdahl's Law," *Comm. ACM,* vol. 31, no. 5, 1988, pp. 532-533.

Daniel R. Johnson is a PhD candidate in electrical and computer engineering at the University of Illinois at Urbana-Champaign. His research interests include parallel accelerators and domain-specific architectures. Johnson has an MS in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He's a member of the ACM and IEEE.

Matthew R. Johnson is a PhD candidate in electrical and computer engineering at the University of Illinois at Urbana-Champaign. His research interests include memory systems, energy-efficient architectures, and hardware/software codesign. Johnson has a BS in electrical and computer engineering and computer science from Duke University. He's a member of the ACM and IEEE.

John H. Kelm is a software engineer at Intel. His research interests include parallel architectures, memory system design, and cache-coherence protocols. Kelm received his PhD in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He's a member of the ACM and IEEE.

William Tuohy is a PhD student in electrical and computer engineering at the University of Illinois at Urbana-Champaign. His research interests include parallel architectures, compilers, and memory systems. Tuohy has a BS in electrical engineering and computer science from the University of California, Berkeley. He's a member of the ACM and IEEE.

Steven S. Lumetta is an associate professor in the Electrical and Computer Engineering Department at the University of Illinois at Urbana-Champaign. His interests center on high-performance networking and computing, hierarchical systems, and parallel runtime software. Lumetta has a PhD in computer science from the University of California, Berkeley. He's a member of the ACM and IEEE.

Sanjay J. Patel is an associate professor in the Electrical and Computer Engineering Department and a Sony Faculty Scholar at the University of Illinois at Urbana-Champaign. His research interests include high-throughput chip architectures and visual computing. Patel has a PhD in computer science and engineering from the University of Michigan, Ann Arbor. He's a member of IEEE.

Direct questions or comments about this article to Daniel R. Johnson, University of Illinois at Urbana-Champaign, Coordinated Sciences Laboratory, 1306 W. Main St., Urbana, IL 61801; djohns53@illinois.edu.