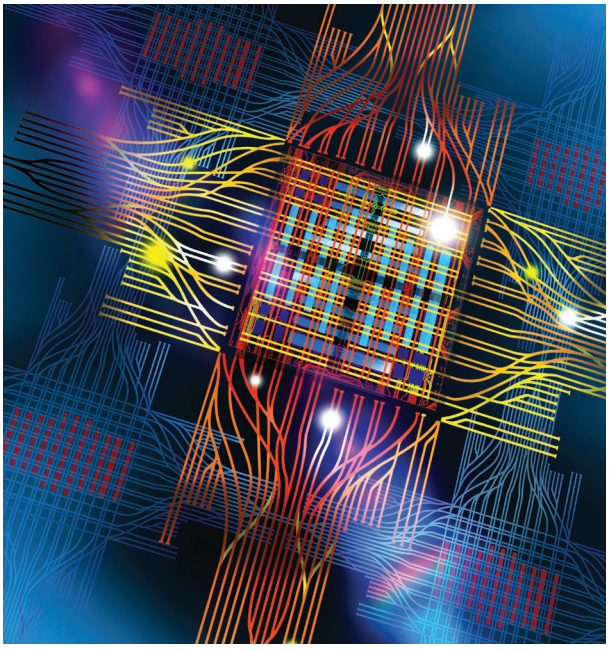

Toward a Coherent Multicore Memory Model

Srinivas Devadas, *MIT*

With exascale multicores, the question of how to efficiently support a shared memory model is of paramount importance. As programmers demand the convenience of coherent shared memory, ever-growing core counts place higher demands on memory subsystems, and increasing on-chip distances mean that interconnect delays exert a significant effect on memory access latencies.

Over the past decade, heat dissipation limits have halted the drive toward ever-higher core frequencies, but transistor density continues to grow,[1] and CPUs with eight or more cores are now common in the commodity- and server-class general-purpose processor markets.[2] To improve performance further and use available transistors more efficiently, architects are resorting to medium- and large-scale multicore processors, both in academia—Raw,[3] TRIPS,[4] and the Execution Migration Machine,[5] for example—and in industry—Tilera,[6,7] Intel TeraFLOPS,[8] and Intel Phi.[9] Pundits predict 1,000 or more cores within only a few years.[10]

How will these massively multicore chips be programmed? A shared memory abstraction stands out as a sine qua non for general-purpose programming. Although architectures with restricted memory models, most notably GPUs, are extremely successful in specific applications such as rendering graphics, most programmers prefer a shared memory model,[11] and commercial general-purpose multicores support this abstraction in hardware. The main question, then, is how to efficiently provide coherent shared memory on the scale of hundreds—or thousands—of cores.

These cores will typically contain per-core L1 and L2 caches because cache power requirements grow quadratically with size; therefore, the only practical option for implementing a large cache is to physically distribute it on the chip so that every core is near some portion of the cache.[7,9]

For programmability, these cores should present a unified addressing space, and, for efficiency, this space must be managed automatically at the hardware level. Cache coherence must be maintained at the L1 caches (and at L2 in case of private L2 caches) to support shared memory. Snooping cache coherence protocols, popular at small (such as four) core counts, are not viable in many-core architectures due to serialization overheads. Another difficulty arises from the fact that conventional bus-and-crossbar interconnects no longer scale due to bandwidth or area limitations. Many-core chip multiprocessors (CMPs), instead, tend toward a tiled architecture—where arrays of tiles are connected over a point-to-point on-chip interconnect.

 Published by the IEEE Computer Society

On scales in which bus-based mechanisms fail, the traditional solution for hardware-shared memory is directory-based cache coherence, where a logically central directory coordinates sharing among the per-core caches, and each core cache must negotiate shared (read-only) or exclusive (read/write) access to each cache line via a coherence protocol. However, the use of directories poses its own challenges. Coherence traffic can be significant, which increases interconnect delay, congestion, and power usage. Additionally, the performance of applications can suffer due to long latency between directories and requestors, especially for shared read/write data; directory sizes must equal a significant portion of the combined size of the per-core caches, or else directory evictions will limit performance.[12]

In this special issue, the first two articles address these directory-related challenges and provide novel techniques for improving power and performance of directory-based coherence.

In "The Impact of Dynamic Directories on Multicore Interconnects," Matthew Schuchhardt and his colleagues recognize that a large fraction of on-chip traffic originates not from actual data transfers, but from intercore communication to maintain data coherence. They describe how placing directories near the data sharers eliminates a large fraction of on-chip interconnect traversals.

In "An Application-Tailored Approach to Hardware Cache Coherence," Arrvindh Shriraman, Hongzhou Zhao, and Sandhya Dwarkadas propose tailoring coherence support to applications in many-core processors. They present a technique that reduces directory storage requirements by recognizing sharing patterns. Additionally, they present a protocol-level technique to exploit the spatial access granularity of an application and thereby avoid communication of unnecessary data.

In the third article, "Single-Cycle Multihop Asynchronous Repeated Traversal: A SMART Future for Reconfigurable On-Chip Networks," Tushar Krishna and his colleagues directly address the latency issue of the on-chip interconnect. The SMART (single-cycle multihop asynchronous repeated traversal) on-chip network presents a single-cycle path all the way from the source to the destination, irrespective of the physical number of hops between them. SMART potentially reduces the latency of coherence traffic significantly.

The fourth article, "Toward Holistic Soft-Error-Resilient Shared-Memory Multicores" by Qingchuan Shi and Omer Khan, presents an error-resilient shared-memory multicore architecture. Their distributed redundancy control mechanism operates in concert with the coherence protocol to enable a deterministic rolled program state for redundant execution at per-core granularity. Additionally, Shi and Khan combine their technique with a resilient cache coherence protocol to trade off performance and energy for soft-error coverage.[13]

There are, of course, other challenges in the design and implementation of many-core systems in addition to hardware shared memory—off-chip bandwidth requirements and on-chip power budgets, to name two. Enabling many-core shared memory will go a long way toward easing programmer burden, so the focus can be on application optimization to reduce bandwidth and energy requirements.

References

1. International Technology Roadmap for Semiconductors, "Assembly and Packaging," 2007; http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Assembly.pdf.
2. S. Rusu et al., "A 45nm 8-Core Enterprise Xeon Processor," *Proc. IEEE Asian Solid State Circuits Conf.* (A-SSCC 09), IEEE, 2009, pp. 9-12.
3. E. Waingold et al., "Baring It All to Software: Raw Machines," *Computer*, Sept. 1997, pp. 86-93.
4. K. Sankaralingam et al., "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," *Proc. IEEE/ACM Int'l Symp. Computer Architecture* (ISCA 03), IEEE CS, 2003, pp. 422-433.
5. K.S. Shim et al., "Design Tradeoffs for Simplicity and Efficient Verification in the Execution Migration Machine," *Proc. IEEE Int'l Conf. Computer Design* (ICCD 13), to be published Oct. 2013.
6. D. Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no, 5, Sept. 2007, pp. 15-31.
7. S. Bell et al., "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," *Proc. IEEE Int'l Solid State Circuits Conf.* (ISSCC 08), IEEE, 2008, pp. 88-89.
8. S.R. Vangal et al. "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, 2008, pp. 29-41.
9. R. Hazara, "The Explosion of Petascale in the Race to Exascale," presentation at the Int'l Supercomputing Conf., 2012; www.intel.com/newsroom/kits/isc/2012/pdfs/Intel_ISC_2012-Presentation.pdf.
10. S. Borkar, "Thousand-Core Chips: A Technology Perspective," *Proc. Design Automation Conf.* (DAC 07), ACM, 2007, pp. 746-749.
11. A.C. Sodan, "Message-Passing and Shared-Data Programming Models—Wish vs. Reality," *Proc. Int'l Symp. High Performance Computing Systems Applications* (HPCS 05), IEEE, 2005, pp. 131-139.
12. A. Gupta, W. Weber, and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," *Proc. Int'l Conf. Parallel Processing* (ICPP 90), Pennsylvania State Univ. Press, 1990, pp. 312-321.
13. K. Aisopos and L.-S. Peh, "A Systematic Methodology to Develop Resilient Cache Coherence Protocols," *Proc. 44th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 11), ACM, 2011, pp. 47-58.

Srinivas Devadas *is the Edwin Sibley Webster Professor of Electrical Engineering and Computer Science at MIT. His research interests include computer-aided design, computer architecture, and computer security. Devadas received a PhD in electrical engineering and computer science from the University of California, Berkeley. He is a Fellow of IEEE. Contact him at devadas@mit.edu.*