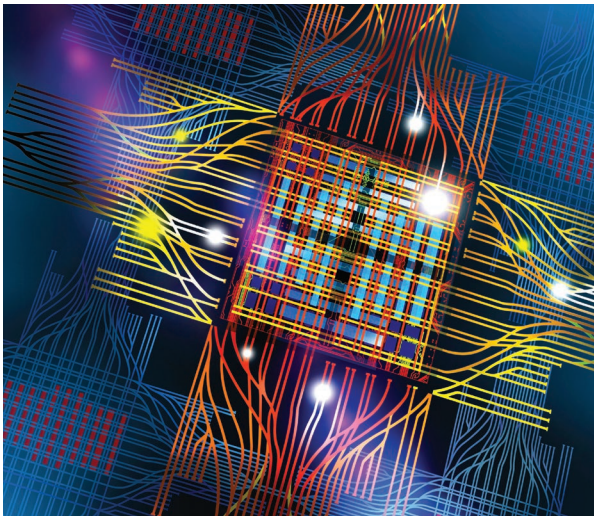

The Impact of Dynamic Directories on Multicore Interconnects

Matthew Schuchhardt, *Northwestern University*
Abhishek Das, *Intel*
Nikos Hardavellas, Gokhan Memik, and Alok Choudhary, *Northwestern University*

A large fraction of on-chip multicore interconnect traffic originates not from actual data transfers but from communication between the cores to maintain data coherence. Co-locating directories near their shared data eliminates many interconnect traversals, substantially reducing power and energy consumption.

To combat increasing on-chip wire delays as core counts and cache sizes grow, multicore architectures have become more distributed. For example, in Intel Xeon Phi and Tilera TILE-Gx processors, the on-chip last-level cache is divided into multiple cache slices that are spread across the die area along with the cores.[1] To facilitate data transfers and communication among the cores, such processors employ elaborate on-chip interconnection networks that consume 10 to 28 percent of a multicore chip's power,[2,3] stressing an already limited resource. As core counts continue to scale, the on-chip interconnect's power consumption is expected to rise even higher.

To minimize this power consumption, researchers have recently proposed circuit-level techniques to improve the power efficiency of the link circuitry and router microarchitecture,[4] dynamic voltage scaling and power management,[5] and thermal-aware routing.[6] However, these efforts fail to consider that a significant fraction of the on-chip interconnect traffic stems from packets that facilitate data coherence, rather than from packets that transfer shared data.

The coherence requirement is a consequence of performance optimizations for on-chip data. To enable faster data accesses, the distributed cache slices are typically treated as private caches for the nearby cores, forming tiles with a core and a cache slice in each tile.[1] These caches maintain data coherence through a directory-based coherence protocol, wherein a distributed directory is address interleaved among the tiles.[1] However, address interleaving is oblivious to data access and sharing patterns; it is often the case that a cache block maps to a directory in a tile physically located far from the accessing cores. To share a cache block, the sharing cores must traverse the on-chip interconnect multiple times to communicate with the directory, instead of communicating directly with each other. These unnecessary network traversals increase traffic and consume power and energy.

In recognition of this effect, Tilera's TILEPro64 implements a mechanism that aims to reduce directory coherence traffic by allowing the software to designate a page's home node.[7] This technique is similar to *dynamic directories*, a technique that cooperates with the operating system (OS) to eliminate the need to place directory entries on a predetermined tile.[8] Although the TILEPro64 provides the placement mechanism, Tilera does not advocate a specific placement policy, nor does the literature contain an evaluation of this technique's efficacy.

 Published by the IEEE Computer Society

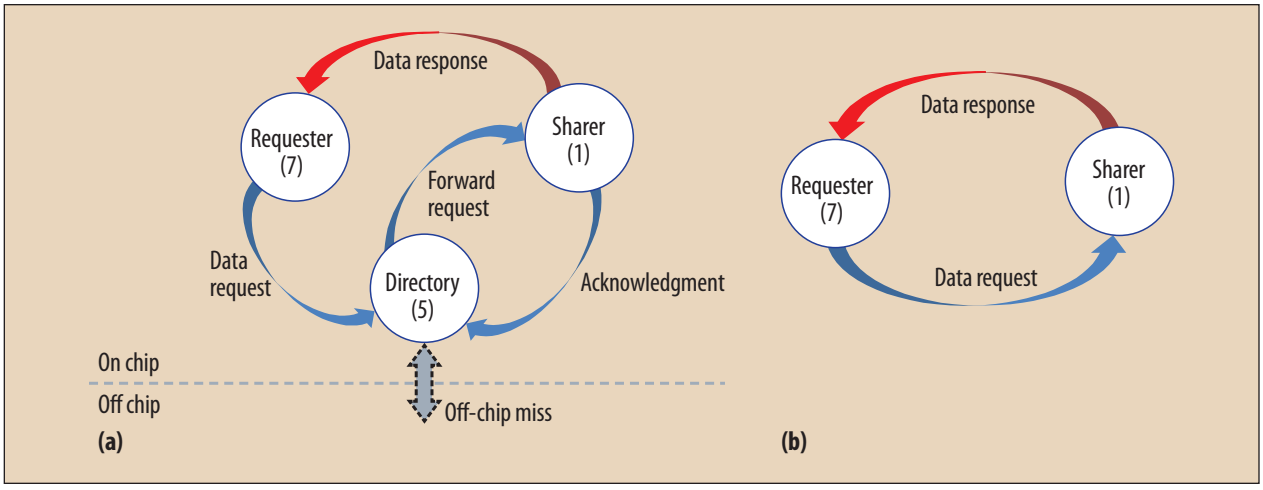

Figure 1. Data-sharing traffic patterns in the baseline multicore architecture. (a) Packet flow when tile 7 requests data from a block owned by tile 1, with the directory at tile 5. (b) Packet flow when the directory is co-located with the data.

Here, we assess the impact of dynamic directories on a multicore processor's performance, power, and energy consumption. We demonstrate this method's effectiveness under a simple directory placement policy, which locates directory entries close to the most active sharer cores of the corresponding cache blocks, and compare it to both *virtual hierarchies*,[9] a previously proposed technique that also helps with directory placement, and a scheme developed by Blas Cuesta and his colleagues[10] that deactivates coherence tracking for cache blocks within a private page—for simplicity, we refer to this scheme as *private coherence deactivation* (PCD).

Compared to the baseline architecture, dynamic directories reduce interconnect power and energy by up to 37.3 percent (22.9 percent on average for scientific workloads, 8.0 percent for MapReduce workloads), with negligible performance impact and hardware overhead, and save four times more interconnect energy than virtual hierarchies. Dynamic directories exhibit the same gains on private pages as PCD.

ARCHITECTURE OVERVIEW

The baseline architecture is a tiled multicore wherein each tile consists of a processing core, private split instruction and data first-level caches, a private second-level cache (L2), a slice of the distributed directory, and a router.[1] The tiles are connected through a two-dimensional folded torus. Without loss of generality, and similarly to most relevant studies, we consider a full-map distributed directory that is address interleaved among the tiles.

Address interleaving does not require a lookup to extract the directory location; all nodes can independently calculate it by the cache block address. However, address-interleaved placement statically distributes the directories without regard to the accessing cores' locations, leading to unnecessary on-chip interconnect traversals.

Figure 1a shows a typical data-sharing traffic pattern, in which the directory is placed at an arbitrary node (tile 5) oblivious to the sharing cores' locations (tiles 1 and 7). In this case, with tile 7 requesting data from tile 1, as Figure 1b shows, the directory ideally would be co-located with the sharer core at tile 1: this would eliminate two unnecessary network messages and let the sharing cores communicate directly rather than through an intermediate node. Such placement is the goal of dynamic directories. Note that even if the data are core-private, the private tile still routes L2 misses to memory through the directory, thereby unnecessarily involving an intermediate node.

DYNAMIC DIRECTORIES

Dynamic directories, similar to the TILEPro64, reduce unnecessary on-chip interconnect traffic by placing directory entries on tiles with cores that share the corresponding data. To achieve this, for every page, dynamic directories designate an owner tile of the directory entries for the blocks in that page and store the owner ID in the page table. By utilizing the existing virtual-to-physical address translation mechanism, dynamic directories propagate the directory location to all cores touching the page through the *translation lookaside buffer* (TLB).

OS support and directory placement

To access the L2 cache, a core translates the virtual address of the data to a physical address through the TLB. Upon a TLB miss—for example, the first time a core accesses a page, or if the TLB entry has been evicted—the system locates the corresponding OS page-table entry and loads the address translation into the TLB.

Dynamic directories modify this process slightly. The first time any of the cores access a page, the OS declares the page private to that core—we refer to that core as the *first accessor*. This information is stored in the page table.

No directory entries need to be allocated for a private page, as there is no need to maintain coherence without sharers.

If another core accesses the page, that core will also miss in its TLB as it has no valid entry. Upon the TLB miss, the OS (or the hardware page-walk mechanism) discovers that a core is already accessing this page and reclassifies it as shared. At the same time, the first accessor becomes the owner of the page's directory entries, and the system allocates the necessary directory entries in its tile. The OS records the directory location in the page table and communicates this to the core through the TLB fill. Thus, any subsequent accessor of the page is also notified of the location of the directory for the page's blocks. This guarantees that the directory is co-located with one of the page's sharer cores, and at the same time provides a simple mechanism to locate the directory entries.

Dynamic directories allow cache coherence at the granularity of individual cache blocks.

When it reclassifies a private page as shared, the system must allocate directory entries for the page's cached blocks. However, it does not allocate directory entries for private pages, so when a page becomes shared, the system does not know which of its blocks are cached and what state they are in. The only certainty is that the first accessor has any of the page's cached blocks.

The least complex solution is to flush all of that page's blocks from the first accessor's cache, eliminating the need to allocate directory entries for previously cached blocks. From that point on, the system can allocate directory entries on demand at the first accessor's directory, as the first accessor remains the directory owner. Although simple, this solution increases the cache miss rate as well as the number of cache accesses to perform the cache flush, both of which have potentially detrimental effects on performance, power, and energy consumption.

Alternatively, the system can conservatively allocate directory entries for all the page's blocks and declare the first accessor's cache as the owner. The system will then forward all requests for this page to the first accessor, which can easily resolve the state of the requested blocks locally (both cached blocks and directory entries reside in the same tile).

In either method, the first accessor remains the owner of the directory entries, and the OS invalidates the corresponding TLB entry at the first accessor to remove the stale private page state (the page is now shared). For our evaluation, we chose the latter option: the system conservatively allocates directory entries for all the page's blocks. This method has negligible power and performance impact, and the only downside is that it might allocate more directory entries than absolutely necessary during the relatively rare reclassification events.

Dynamic directories allow cache coherence at the granularity of individual cache blocks; the OS only places the corresponding directory entries at some tile. The placement occurs at page granularity—the system places the directory entries for all blocks within a page at the same tile. Finer-grain placement is possible but incurs significant overhead. To support placement at granularities smaller than a page, the TLB and page-table entries would need to store multiple directory owners (one per placement grain), and each subsection of the page would have to generate a separate TLB trap to extract the directory location for it. However, complex fine-grain techniques are not justified, as granularities smaller than a page provide negligible energy benefits. Nonetheless, for completeness, we also present the energy savings of such a hypothetical approach.

Thread migration

When threads migrate, the corresponding directory entries could either stay in the original tile and be accessed remotely by the migrating thread or move along with the migrating thread. Dynamic directories could also be turned off.

The simplest choice is to leave the directory entries in place and let the migrating thread access them remotely. This does not require any new mechanisms or structures, nor does it change any existing ones. Having directories at a remote node and accessing them via the interconnect is similar to the baseline architecture. The downside is that directory placement might no longer be optimal, and the interconnect could suffer from hot spots as all the directory entries frequently accessed by this thread are concentrated at the same remote node. Thus, under this scheme, thread migration becomes even more expensive than it already is, thereby raising the importance of affinity scheduling. This would be the best option if threads migrate away from their core only for short periods of time, relatively infrequently, and quickly return due to affinity scheduling.

The second alternative—migrate the directory entries together with the thread—would preserve the affinity of the directory entries and the thread at the cost of identifying the entries to migrate, performing the actual migration, and resolving any races occurring during the directory migration. Identifying the directory entries to migrate is easy when a core executes only a single thread: all entries owned by that tile should migrate. However, a core that executes multiple threads through multithreading or time sharing needs to tag the directory entries with the process/thread IDs, so the system knows exactly which entries to move. Performing the migration would

require TLB shootdowns, as well as bandwidth and energy consumption on the interconnect for the actual transfer. Races could easily appear, as requests could arrive while the directory migration is in flight, and the OS would have to handle them successfully. Overall, this is an expensive proposition, akin to traditional directory migration, and likely would not provide enough additional benefits to justify its implementation unless threads migrate to a new core and stay there for a very long time before migrating again.

Dynamic directories can be turned off for pathological cases: one bit per page could indicate whether its directory entries are managed by dynamic directories or by traditional address interleaving. However, the moment that a page's dynamic directories are turned off, the page would need to go through a process similar to reclassification: the OS has to flush cached blocks from the caches and shoot down the corresponding TLB entries so they can be updated. Thus, while easy to implement, this solution also comes with potentially significant overhead. To avoid excessive overhead, dynamic directories should remain switched off for the entire duration of high thread-migration rates. In that case, the overhead would be a one-time event amortized over billions of accesses, and the scheme would perform almost identically to the baseline. While this would prevent dynamic directories from providing any benefit over the baseline, it would also guarantee that they do no harm.

We did not observe any significant migration rates in our workloads, hence we did not separately evaluate these options. However, each solution is best suited for different conditions of migration frequency and length of stay. A sophisticated system could monitor these metrics and employ the best option each time.

Overhead

Dynamic directories extend each TLB entry by $\log_2 N + 1$ bits, where N is the number of tiles. The Intel Core i7 (Sandy Bridge) has two-level TLBs, with 64 L1-TLB entries and 512 L2-TLB entries for a 4-Kbyte page size. For 16-core chip multiprocessors (CMPs), dynamic directories add 4 bits for the directory owner and 1 bit for the shared/private state. This amounts to only 5.63 Kbytes additional static RAM (SRAM). We modeled the TLB with CACTI 6.5[11] and found that the energy overhead is negligible (0.7 percent of the TLB read energy).

On the software side, Linux on Core i7 typically uses 64-bit page-table entries, in which bits 9-11 and 48-62 are unused. These extra bits can accommodate up to 131,072 cores and hence dynamic directories do not add any overhead in the page table. The only software overhead incurred is a few extra lines of kernel code to initialize the directory owner and state bits and to orchestrate page reclassifications when necessary.

Mechanism justification

Instead of utilizing unused page-table entry bits to encode the owner ID for a page's directory entries, dynamic directories could designate $\log_2 N$ bits of the physical address to encode the owner tile ID. The selection of a physical frame for a virtual page would then be limited only to addresses that assign the correct tile ID to these bits.

However, this would couple the memory allocation with the directory placement. Limiting physical addresses to specific address ranges for each tile would cause memory fragmentation, degrade performance, and complicate other optimizations that pose conflicting address translation requests—for example, page coloring for L1 caches. Overloading the address bits within the cache index for directory placement would result in these bits being all the same for pages assigned to the same tile, thereby utilizing only a subset of the available L2 cache sets. Similarly, using higher address bits for directory placement might conflict with the bits used for dynamic RAM bank, channel, or column selection, leading to underutilization of the DRAM banks, memory channels, or row buffer. Moreover, if address bits are used for directory placement, each core in an N-core CMP will be able to allocate locally only $1/N$ of the overall physical pages, limiting the performance potential of dynamic directories if more pages are private to (or accessed mostly by) that core.

Dynamic directories fully decouple page allocation from directory placement.

Dynamic directories avoid all these problems by fully decoupling page allocation from directory placement.

EXPERIMENTAL RESULTS

We evaluated dynamic directories by simulating a 16-core tiled CMP running scientific workloads (a mixture of compute-intensive applications and computational kernels) and Phoenix MapReduce workloads. We simulated the CMP on Flexus[12] and followed the SimFlex multiprocessor sampling methodology.[13] The full details of our simulation methodology, the power model, and details on the simulated architecture and workloads appear elsewhere.[8] We simulated the entire execution of the Map phase for Phoenix applications, which constitutes the majority of execution time, and three complete iterations for the scientific applications.

Directory placement policy

Ideally, in the absence of directory migration, the directory entries for a page are placed at the tile that issues the

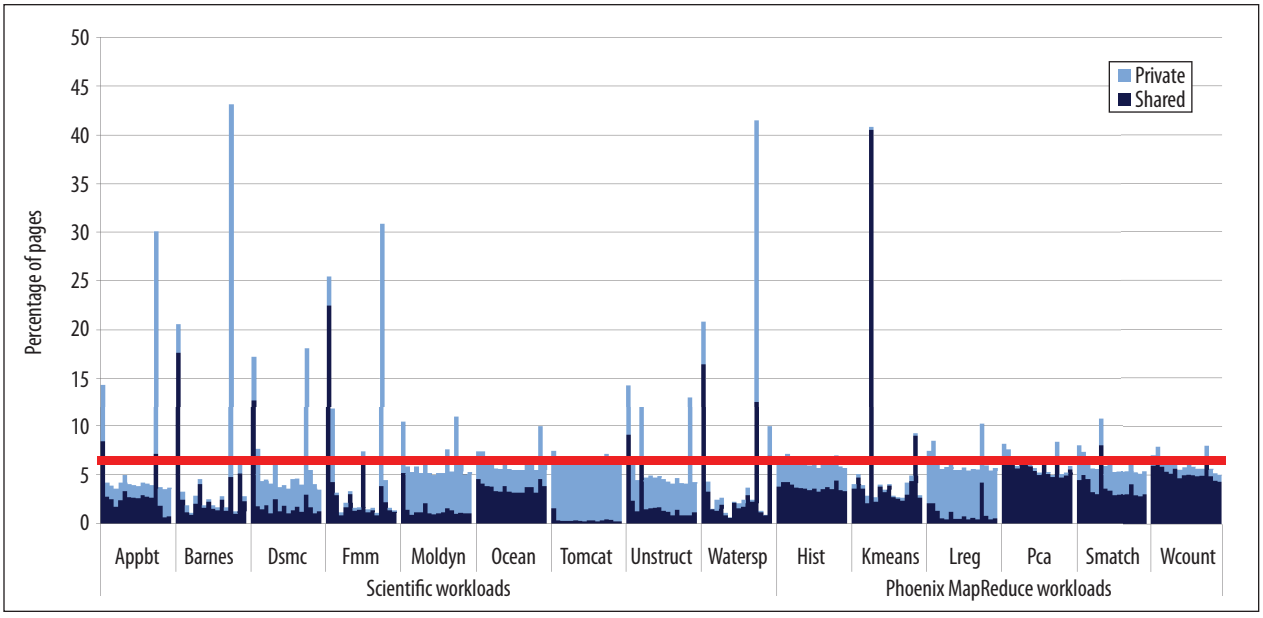

Figure 2. Distribution of directory entries across 16 tiles for private and shared pages in simulations of scientific and Phoenix MapReduce workloads. The red line indicates a hypothetical uniform distribution. Flatter benchmarks indicate that the directory entries are relatively evenly distributed across the tiles.

most accesses to them. Identifying this tile, however, requires complex techniques. Fortunately, the first accessor of a page issues on average only 6 percent fewer accesses than the top accessor, and it is trivial to identify. Therefore, dynamic directories choose to allocate a page's directory entries to its first accessor.

Distribution of directory entries across tiles

Dynamic directories could skew the distribution of directory entries to tiles, in contrast to the uniform distribution of traditional address interleaving. If some tiles allocate vastly more directory entries than others, they might require a disproportionately large area for the directory or cause traffic hot spots that degrade performance.

Figure 2 shows the distribution of directory entries across the tiles for private and shared pages. There is a band of 16 bars for each workload, with each bar's height designating how many pages have their directories allocated at that tile. The red line indicates a hypothetical uniform distribution. The figure yields three key insights.

First, while the distribution of directory entries is sometimes skewed, the imbalance could be minimized. Only one core accesses private data, obviating the need for a directory. Thus, dynamic directories allocate directory entries only for shared pages, reducing the on-chip directory capacity requirements by 48 percent on average.

Second, while the directory entries for shared pages are mostly evenly distributed, oversubscribed tiles still exist (excluding Kmeans and Fmm, a tile gets at most 18 percent of the directories). With dynamic directories, the system has the flexibility to allocate the directories to nearby tiles or to another sharer core when the first accessor is overloaded, thus spreading the load while still guaranteeing directory proximity to the accessing cores.

Finally, the uneven distribution of directory entries is not a direct indicator of increased traffic hot spots. The baseline might already exhibit imbalanced traffic, some pages are colder than others, and dynamic directories reduce the number of messages traversing the interconnect by 22.7 percent on average, easing traffic congestion. While dynamic directories on Kmeans, Fmm, and Dsmc exacerbate existing traffic imbalances, these hot spots have a negligible performance impact, as the applications spend only a small fraction of execution time on the distributed L2 cache. In the remaining applications, a tile could receive on average 8 percent more directory accesses from remote cores compared to the baseline, but these imbalances are relatively small and do not impact performance. Also, as indicated earlier, in pathological cases it is simple to turn off dynamic directories.

For completeness, we evaluated Kmeans and Fmm with a modified dynamic directories scheme, wherein the directories for the shared pages are address interleaved at block granularity similar to Reactive NUCA.[1] This directory placement removes the hot spots that Kmeans and Fmm experience.

Energy savings

We evaluated the impact of dynamic directories on energy and compared the results against three schemes: the baseline architecture, virtual hierarchies,[9] and PCD.[10]

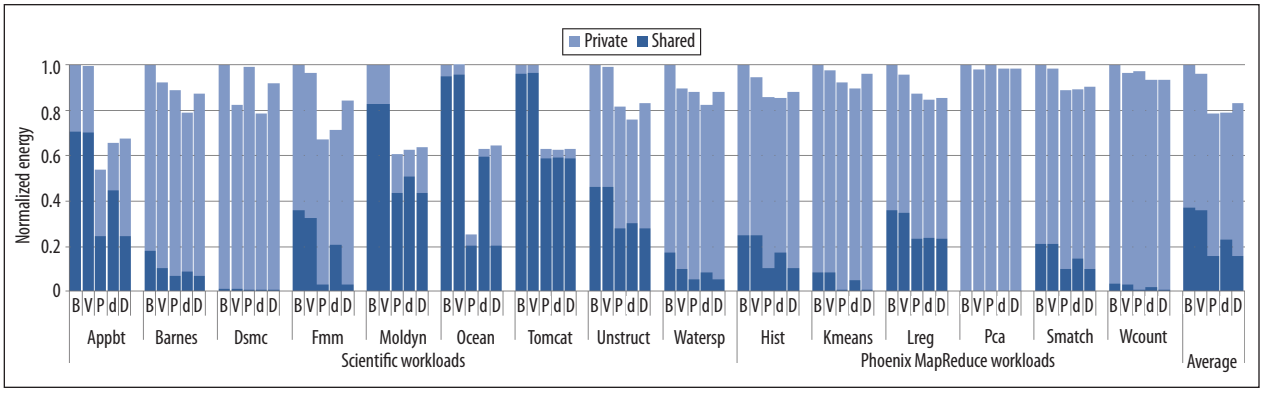

Figure 3. Network energy consumption of multiple directory placement schemes in simulations of scientific and Phoenix MapReduce workloads, normalized to the baseline architecture. The evaluated schemes are, from left to right: the baseline architecture (B), virtual hierarchies (V), private coherence deactivation (P), dynamic directories at 64-byte cache-block granularity (d), and dynamic directories at 8-Kbyte page granularity (D). The energy consumption for private and shared data, classified at page granularity, is shown separately.

Virtual hierarchies utilize a coherence protocol that dynamically assigns home tiles to cache blocks. An access that misses on the local tile finds the corresponding directory location by indexing a table using the bits of the physical address above the block offset. Thus, all pages with the same $\log_2 N$ bits in the physical address have their directory entries at the same tile. For our evaluation, we implemented a "perfect" virtual hierarchies scheme wherein the tile with the most accesses to a memory region becomes the home tile for the entire region.

PCD deactivates coherence tracking for cache blocks within a private page, while the placement of the directories for the shared pages is orthogonal to PCD. The original PCD proposal was evaluated on the AMD Magny-Cours processor architecture, where all the blocks within a page map to the same directory node, essentially interleaving the directories at page granularity. To isolate the effect of PCD from the choice of a coherence-tracking mechanism for shared data, we implemented an improved version of PCD in which the directories of shared blocks are address interleaved at block granularity, thereby relieving the hot spots generated in the original Magny-Cours-based PCD scheme.

Figure 3 illustrates the on-chip interconnect energy consumption of dynamic directories at cache-block granularity (DynDir-BLK, indicated by the letter d) and page granularity (DynDir-8K, indicated by the letter D), virtual hierarchies (V), and PCD (P), normalized to the energy consumption of the baseline architecture (B). Dynamic directories reduce network energy on average by 21.2 percent at block granularity and 16.9 percent at page granularity. Virtual hierarchies save only 3.9 percent of network energy on average, as the system allocates directory entries for an entire memory region rather than individual pages.

PCD exhibits the same gains as DynDir-8K for private pages, as both PCD and directory homing eliminate interconnect traversals to the directory for private data. For shared data, PCD follows the baseline coherence and address interleaves the directories at block granularity, thereby placing the directories at arbitrary nodes. DynDir-8K places the directories together with the sharers, but placement at the page granularity could increase false sharing. Overall, DynDir-8K attains higher energy savings for shared data in some applications (for example, Dsmc), while block interleaving the directories of shared data is a better choice for other applications (for example, Ocean).

To separate the impact of directory homing from the granularity of directory placement, we compared page-grain address interleaving of shared directory entries—identical to the original Magny-Cours-based PCD scheme—with page-grain directory homing—identical to DynDir-8K. We found that DynDir-8K provides significant benefits over the original Magny-Cours-based PCD scheme, achieving 35 percent energy savings in Ocean, 29 percent in Appbt, 27 percent in Moldyn, and 9 percent in Fmm. However, compared to our improved version of PCD, the benefits of page-grain homing are balanced by the increased false sharing of page-grain directory placement. As a result, the block interleaving of PCD and the directory-page homing of DynDir-8K even out across applications, resulting in energy savings within only 4 percent of each other on average.

The scientific applications attained higher energy savings (22.9 percent on average, 37.3 percent maximum) compared to the Phoenix MapReduce applications (8 percent on average). Phoenix applications exhibit a higher fraction of shared data accesses, rendering dynamic directories less useful. The energy savings for all schemes are largely achieved through a reduction of control messages. As Figure 4 shows, on average dynamic directories eliminated 22.7 percent of the control messages on the on-chip

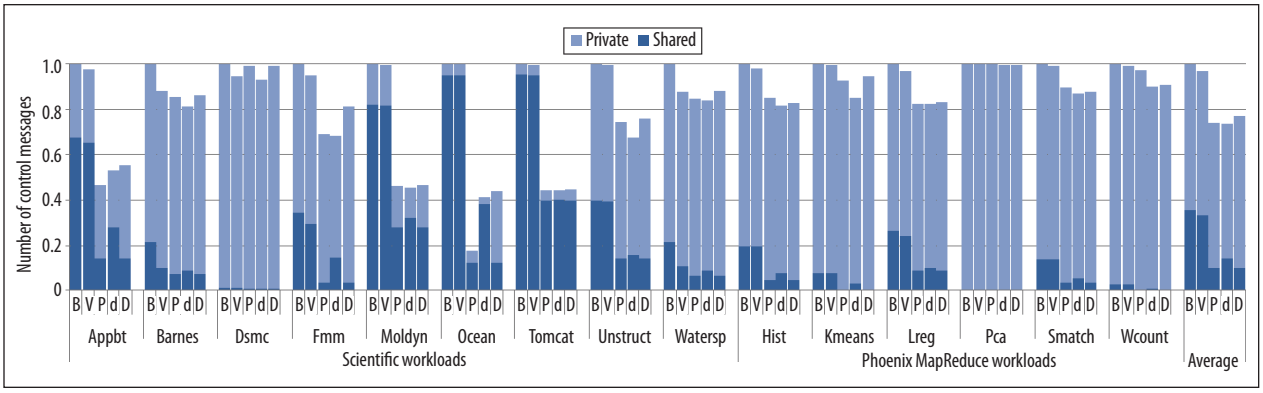

Figure 4. Number of interconnect control messages (all messages except data replies) of scientific and Phoenix MapReduce workloads, normalized to the baseline architecture (B). Data is analyzed at 64-byte cache-block (d) and 8,000-page (D) granularities, and compared to virtual hierarchies (V) and private coherence deactivation (P). The number of messages for private and shared data, classified at page granularity, is shown separately. The nonglobal sharing characteristics of the scientific workloads lead to generally higher savings.

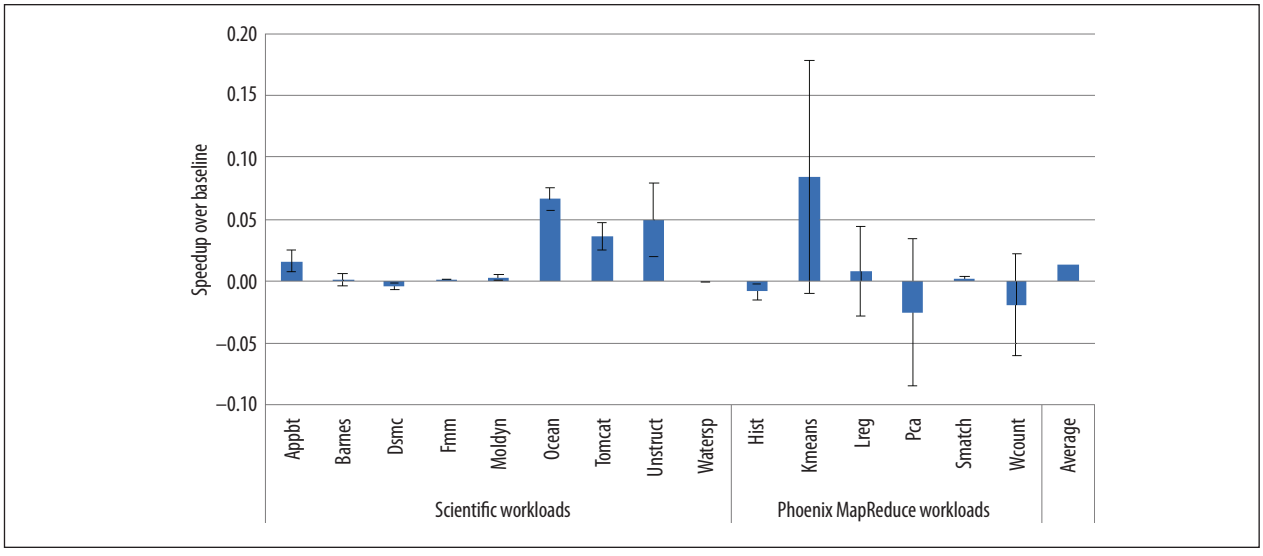

Figure 5. Speedup of dynamic directories over the baseline architecture for scientific and Phoenix MapReduce workloads. Performance is slightly improved, with a 1.4 percent average speedup.

interconnect while PCD eliminated 25.7 percent and virtual hierarchies eliminated 3.1 percent of the control messages.

Performance impact

Figure 5 shows the overall speedup achieved using dynamic directories compared to the baseline architecture. Dynamic directories slightly increased performance in 7 out of 15 applications and decreased performance in 2. They improved performance by up to 7 percent (Ocean) and by 1.4 percent on average, while the maximum performance slowdown was 1.3 percent (Pca).

Performance improved because

- dynamic directories reduce the number of network packets, which eliminates congestion and hence reduces the effective interconnect latency, and
- data transfers (on-chip and off-chip) are faster because many accesses to remote directories are eliminated.

Because the working set of most applications is large, dynamic directories' savings are realized mostly by off-chip memory accesses. As off-chip memory access latency is already large, saving a few cycles does not improve performance considerably.

We attribute the slowdown exhibited by Pca and Wcount to the page granularity at which dynamic directories are assigned, as this assignment can cause contention and hot spots. Especially for universally shared pages, it is likely that different cores access cache blocks in nearly consecutive cycles, causing contention in the directory tile and increasing the directory's response time. On average, the positive and negative forces canceled each other out, and

dynamic directories had only a small performance impact (1.4 percent on average).

A large fraction of on-chip interconnect traffic on multicore processors stems from placing directory entries without regard to data access and sharing patterns. Future architectures should consider the implications of allocating directory entries close to the cores accessing the corresponding data. The primary impact will be a reduction in on-chip interconnect power and energy consumption; in our evaluation, power and energy consumption was reduced by up to 37.3 percent, with negligible hardware overhead and without any adverse performance impact—in fact, performance slightly improved. As the importance of on-chip interconnects rises with future process technologies, dynamic directories will effortlessly scale and provide even greater power savings.

Acknowledgments

This work was partially supported by National Science Foundation grants CCF-1218768, CCF-0916746, CCF-0747201, and CNS-083092; by Northwestern University through an ISEN booster award; and by the June and Donald Brewer Chair in EECS at Northwestern University (and other funds from Northwestern University).

References

1. N. Hardavellas et al., "Near-Optimal Cache Block Placement with Reactive Nonuniform Cache Architectures," *IEEE Micro*, vol. 30, no. 1, 2010, pp. 20-28.
2. E. Totoni et al., "Comparing the Power and Performance of Intel's SCC to State-of-the-Art CPUs and GPUs," *Proc. 2012 IEEE Int'l Symp. Performance Analysis of Systems & Software* (ISPASS 12), IEEE CS, 2012, pp. 78-87.
3. S. Borkar, "The Exascale Challenge," *Proc. 2010 Int'l Symp. VLSI Design Automation and Test* (VLSI-DAT 10), IEEE, 2010, pp. 2-3.
4. H. Wang, L.-S. Peh, and S. Malik, "Power-Driven Design of Router Microarchitectures in On-Chip Networks," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO-36), IEEE CS, 2003, pp. 105-116.
5. L. Shang, L.-S. Peh, and N.K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," *Proc. 9th Int'l Symp. High-Performance Computer Architecture* (HPCA 03), IEEE CS, 2003, pp. 91-102.
6. L. Shang et al., "Thermal Modeling, Characterization and Management of On-Chip Networks," *Proc. 37th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO-37), IEEE CS, 2004, pp. 67-78.
7. Tilera Corp., *Tile Processor User Architecture Manual*, rel. 2.4, doc. no. UG101, Nov. 2011; www.tilera.com/scm/docs/UG101-User-Architecture-Reference.pdf.
8. A. Das et al., "Dynamic Directories: A Mechanism for Reducing On-Chip Interconnect Power in Multicores," *Proc. 2012 Conf. Design, Automation, and Test in Europe* (DATE 12), EDA Consortium, 2012, pp. 479-484.
9. M.R. Marty and M.D. Hill, "Virtual Hierarchies to Support Server Consolidation," *Proc. 34th Ann. Int'l Symp. Computer Architecture* (ISCA 07), ACM, 2007, pp. 46-56.
10. B.A. Cuesta et al., "Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks," *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), ACM, 2011, pp. 93-104.
11. N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi, *CACTI 6.0: A Tool to Model Large Caches*, tech. report HPL-2009-85, HP Labs, 2009; www.hpl.hp.com/techreports/2009/HPL-2009-85.html.
12. N. Hardavellas et al., "SimFlex: A Fast, Accurate, Flexible Full-System Simulation Framework for Performance Evaluation of Server Architecture," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 31, no. 4, 2004, pp. 31-34.
13. T.F. Wenisch et al., "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, no. 4, 2006, pp. 18-31.

Matthew Schuchhardt *is a PhD student in the Department of Electrical Engineering and Computer Science at Northwestern University as well as an intern in the Strategic CAD Labs group at Intel in Hillsboro, Oregon. His research interests include parallel computer architectures, empathic systems, and affective computing. Contact him at matthew.schuchhardt@intel.com.*

Abhishek Das *is a security architect in the Data Center and Connected Systems group at Intel in Hillsboro, Oregon. His research focuses on security architecture and the design of future-generation Intel Xeon platforms. Das received a PhD in electrical engineering and computer science from Northwestern University. Contact him at abhishek.das@intel.com.*

Nikos Hardavellas *is an assistant professor in the Department of Electrical Engineering and Computer Science at Northwestern University. His research interests include parallel computer architecture, memory systems, optical interconnects, elastic fidelity computing, and design for dark silicon. Hardavellas received a PhD in computer science from Carnegie Mellon University. He is a member of IEEE and ACM. Contact him at nikos@northwestern.edu.*

Gokhan Memik *is an associate professor in the Department of Electrical Engineering and Computer Science at Northwestern University. His research interests include computer architecture, embedded systems, and mobile computing. Memik received a PhD in electrical engineering from the University of California, Los Angeles. Contact him at memik@eecs.northwestern.edu.*

Alok Choudhary *is the John G. Searle Professor of Electrical Engineering and Computer Science and a professor in the Kellogg School of Management at Northwestern University, where he is also director of the Center for Ultrascale Computing and Information Security (CUCIS). His research interests include high-performance computing, data-intensive computing, scalable data mining, computer architecture, and high-performance I/O systems and software. Choudhary received a PhD in computer engineering from the University of Illinois at Urbana-Champaign. He is a Fellow of IEEE, ACM, and the American Association for the Advancement of Science. Contact him at choudhar@eecs.northwestern.edu.*