Dynamic Directories: A Mechanism for Reducing On-Chip Interconnect Power in Multicores

Abhishek Das*[1], Matt Schuchhardt, Nikos Hardavellas, Gokhan Memik, and Alok Choudhary

Electrical Engineering and Computer Science Department
Northwestern University
Evanston, IL USA

*Datacenter and Connected Systems Group
Intel Corporation
Hillsboro, OR USA

***Abstract*—On-chip interconnection networks consume a significant fraction of the chip's power, and the rapidly increasing core counts in future technologies is going to further aggravate their impact on the chip's overall power consumption. A large fraction of the traffic originates not from data messages exchanged between sharing cores, but from the communication between the cores and intermediate hardware structures (i.e., directories) for the purpose of maintaining coherence in the presence of conflicting updates. In this paper, we propose Dynamic Directories, a method allowing the directories to be placed arbitrarily in the chip by piggy-backing the virtual to physical address translation. This eliminates a large fraction of the on-chip interconnect traversals, hence reducing the power consumption. Through trace-driven and cycle-accurate simulation in a range of scientific and Map-Reduce applications, we show that our technique reduces the power and energy expended by the on-chip interconnect by up to 37% (16.4% on average) with negligible hardware overhead and a small improvement in performance (1.3% on average).[2]**

Keywords-On-chip networks; Non-uniform caches; Multicore architecture

I. INTRODUCTION

Advances in process technology enable exponentially more cores on a single die with each new process generation, leading to a commensurate increase in cache sizes to supply all these cores with data. To combat the increasing on-chip wire delays as the core counts and cache sizes grow, future multicore architectures become distributed: the last-level on-chip cache (LLC) is divided into multiple cache slices, which are distributed across the die area along with the cores [5, 21]. To facilitate data transfers and communication among the cores, such processors employ elaborate on-chip interconnection networks. However, recent studies show that such on-chip networks consume between 20% to 36% of the power of a multicore chip [7, 10] and significantly raise the chip temperature [13] leading to hot spots, thermal emergencies, and degraded performance. As core counts continue to scale, the impact of the on-chip interconnect is expected to grow even higher in the future.

To minimize the power consumption of on-chip interconnects, recent research proposes circuit-level techniques to improve the power efficiency of the link circuitry and the router microarchitecture [17], dynamic voltage scaling [13] and power management [13, 14], and thermal-aware routing [15]. However, these prior works miss one crucial observation: a large fraction of the on-chip interconnect traffic stems from packets sent to enforce data coherence, rather than from packets absolutely required to facilitate data sharing.

The coherence requirement is a consequence of performance optimizations for on-chip data. To allow faster data accesses, the distributed cache slices are typically treated as private caches to the nearby cores [2, 21], forming tiles with a core and a cache slice in each tile [1, 3, 5]. Private caches allow the replication of shared data, which, in turn, employ a directory-based coherence mechanism where a directory structure is typically address-interleaved among the tiles [5, 21]. However, this address interleaving is oblivious to the data access and sharing patterns; it is often the case that a cache block maps to a directory in a tile physically located far away from the accessing cores. To share a cache block, the sharing cores need to traverse the on-chip interconnect multiple times to communicate with the directory, instead of communicating directly between them. These unnecessary network traversals increase traffic, consume power, and raise the operational temperature with detrimental consequences.

In this paper, we propose Dynamic Directories, a distributed directory architecture that cooperates with the operating system to eliminate the need to place directory entries on a predetermined tile. We utilize this capability to place directory entries close to the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power. The principal contributions of this paper are:

1. We observe that a large fraction of the on-chip interconnect traffic stems from the data-access-oblivious placement of directory entries.
2. We propose *Dynamic Directories*, a mechanism to co-locate directory entries with the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power.
3. Through trace-driven and cycle-accurate simulation of large scale multicore processors running a range of scientific and Map-Reduce workloads, we show that Dynamic Directories reduce the interconnect energy and power by up to 37% (22% on average for the scientific

[1] This work was performed while Abhishek Das was affiliated with Northwestern University.

[2] This work was supported in part by NSF grants CCF-0916746, CCF-0747201, CNS-0720691, CNS-0830927, CNS-0551639, IIS-0536994, HECURA CCF-0621443, OCI 0956311, OCI 0724599, and SDCI OCI-0724599.

978-3-9810801-8-6/DATE12/©2012 EDAA

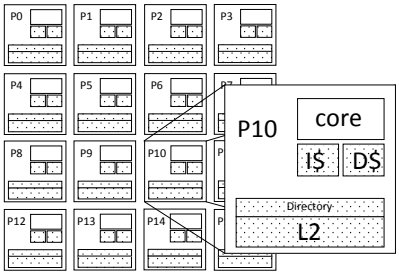


Figure 1. Baseline tiled architecture of a 16-core CMP. Each tile has a core, split I/D L1 caches, an L2 cache slice, and a directory slice.

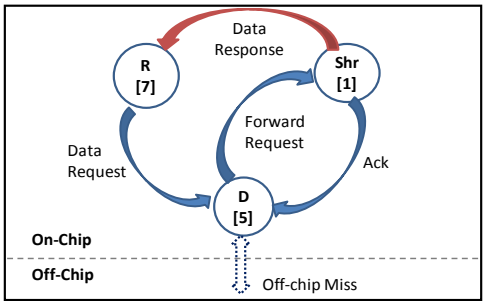


Figure 2. Sequence of messages following a request by tile 7 for a block owned by tile 1, with its directory at tile 5.

workloads and 8% on average for Map-Reduce) with a 1.4% performance improvement on average.

The rest of the paper is organized as follows. Section II presents background and related work. Section III describes the details of our Dynamic Directory scheme, followed by the evaluation methodology in Section IV. The results of our evaluation are presented in Section V. Finally, we conclude the paper with a short summary in Section VI.

II. Background and Related Work

A. Baseline Architecture

This section describes the basics of our tiled architecture. Figure 1 shows each tile consisting of a processing core, a private split I/D first-level cache (L1), a slice of the second-level cache (L2), and a slice of the distributed directory. To scale high core counts, the directory is distributed among the tiles in an address interleaved fashion, i.e., the address of a block modulo the number of tiles determines the directory location for this block. In this work we assume a full-map directory for the baseline and Dynamic Directory architectures, i.e., the directory has the capacity to hold coherence information for all the cache blocks across all the tiles in all cases.

Address interleaving does not require a lookup to extract the directory location; all nodes can independently calculate it using only the address of the requested block. However, address-interleaved placement statically distributes the directories without regards to the location of the accessing cores, leading to unnecessary on-chip interconnect traversals. Figure 2 shows an example of the drawbacks of static address-interleaved directory placement. Tile 7 requests a data block, currently owned by Tile 1, with its directory entry located at Tile 5 as determined by address interleaving. To access the block, Tile 7 first has to access the directory at Tile 5, which forwards the request to the owner Tile 1, which then sends the data to Tile 7 and an acknowledgement to the directory at Tile 5. As the directory placement is oblivious to the location of the sharing cores, most on-chip data transfers will require similar 3-hop messages. Ideally, if the directory is co-located with the sharer at Tile 1, it could eliminate two unnecessary network messages. Such placement is the goal of Dynamic Directories.

Note that in an N-tile multicore system with address-interleaved distributed directory, the probability of a particular tile holding the directory entry for a block is 1/N. This is the probability with which a requesting core can access a directory within its own tile. As the number of tiles increases, the probability of hitting a local directory diminishes. Thus, traditional address-interleaved directory placement becomes increasingly inefficient in future technologies, as it increases the on-chip interconnect power.

B. Related Work

Several previous proposals suggest new coherence mechanisms and cache architectures to reduce multicore cache energy and enhance performance. Ros et al., proposed Dico-CMP [12] which extend cache tags to keep sharer information. Zebchuk et al. [20] proposes bloom filter mechanism for tag-less cache coherency. Finally, Cuesta et al. [4] use protocol deactivation for private block accesses to reduce directory accesses. All these schemes are orthogonal to our scheme which allows directories to be located at any tile by cooperating with the OS.

III. Dynamic Directories Scheme

The Dynamic Directory mechanism reduces the 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 already existing virtual-to-physical address translation mechanism, Dynamic Directories propagate the directory owner location to all cores touching the page. There are two important aspects of this scheme: the classification of pages by the OS, and the directory placement and distribution among the cores. We describe these aspects in the following sections.

A. Operating System Support

To categorize pages and communicate their directory location to the cores, Dynamic Directories piggyback on the virtual-to-physical address translation mechanism. In modern systems, almost all L2 caches are physically accessed. Thus, for all data and instruction accesses, a core translates the virtual address to a physical one through the TLB before accessing L2. Upon a TLB miss (e.g., 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.

We implement Dynamic Directories by slightly modifying this process. When a page is accessed for the first time ever by any of the cores, the page is declared private to the accessing

core. 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 this page is already accessed by a core, and reclassifies the page as shared. At the same time, the first accessor core becomes the owner of the page's directory entries and the directory entries are allocated in its tile. The directory location is recorded in the page table, and communicated to the core through the TLB fill. Thus, any subsequent accessor of the page is also notified of the directory location for the blocks in the page. This mechanism guarantees that the directory is co-located with one of the sharers of the page, and at the same time provides a simple mechanism to locate the directory entries.

To allocate the directory entries when the page is reclassified from private to shared, the system can either issue a purge request to the old owner of the page similar to [5], which shoots down the TLB entries for that page and flushes the corresponding cache blocks, or it can allocate directory entries for all the blocks in the page and declare the old owner's cache as the owner of the blocks. In the latter case, requests for blocks that are not in the old owner's cache can be easily detected and translated into off-chip misses. In either case, such events happen only once per shared page for the lifetime of the execution of the program, and it has been shown that they have a negligible performance impact [5]. Upon reclassifying a page from private to shared, the page table entry is placed in a special poison state that holds off all requests for TLB fills in a FIFO queue until the reclassification is complete, similar to [5].

We modeled the TLB structure with CACTI 6.5 and found that the energy overhead for accessing the TLB is negligible (0.7% per read access). Note that when a TLB entry is evicted, the system does not need to take additional actions. The page table is already up-to-date, and there is no need to flush the cache for that address.

TABLE I. BENCHMARKS USED

Benchmark		Application	Description
Scientific	NAS	*appbt*	Solves multiple independent systems of equations
	SPEC-CPU	*tomcatv*	Vectorized mesh generation; parallel version of 101.tomcatv from SPEC-FP
	Other Scientific	*dsmc*	Simulates the movement and collision of gas particles
		moldyn	Molecular dynamics simulation
		unstructured	Computational fluid dynamics application
	SPLASH-2 [19]	*barnes*	Barnes-Hut hierarchical N-body simulation
		fmm	Simulates particle interactions using the Adaptive Fast Multipole Method
		ocean	Simulates large-scale ocean movements based on eddy and boundary currents
		watersp	Simulates the interactions of a system of water molecules
Map-Reduce	Phoenix [11]	*lreg*	Linear regression to find best fit line for a set of points
		hist	Histogram plot over a bitmap image file
		kmeans	K-Means clustering over random cluster points and cluster centers
		pca	Principal component analysis over a 2D-matrix
		smatch	String matching in a large text file
		wcount	Word count in a large text file

B. *Directory Placement Mechanism*

Directory placement can be done at different granularities. For example, instead of designating one tile as the owner for the directory entries of all the blocks in the page, we could designate different owners for the directory entry of each block individually (or any granularity in between). Such a fine-grain placement would require considerable changes in the overall system operation. First, each TLB entry would have to store multiple directory owners (one per placement-grain). In turn, this would require a separate TLB trap for each sub-section of the page that is accessed to extract the directory location for it. Our results indicate that the system behaves well enough at the page granularity; thus, employing finer-grain techniques is unjustified.

An alternative implementation of Dynamic Directories would be to reuse address bits for directory placement. This could be achieved by simply guiding the selection of physical addresses for each virtual page (i.e., some bits of the physical address will also designate the directory owner). However, such a technique would couple the memory allocation with the directory placement. As a result, forcing the use of specific address ranges could lead to address space fragmentation with detrimental consequences to performance, and may complicate other optimizations (e.g., page coloring for L1) that pose conflicting address translation requests. Overloading address bits could result in underutilizing the cache if the directory placement bits overlap with the cache index, or similarly could underutilize the DRAM banks, or the DRAM row buffer, or the memory channels. Dynamic Directories avoid all these problems by fully decoupling page allocation from directory placement.

While pathological cases of uneven directory distribution are possible, we didn't see any in our workloads, and we don't expect to see any in commercial workloads either: their data are typically universally shared with finely interleaved accesses [5], so the pages should distribute evenly. It is important to note here that it is simple to turn off Dynamic Directories in pathological cases: one bit per page could indicate whether its

TABLE II. ARCHITECTURAL CONFIGURATION

CMP Size	16 cores
Processing Cores	UltraSPARC III ISA; 2GHz, in-order cores, 8-stage pipeline, 4-way superscalar
L1 Caches	split I/D, 16KB 2-way set-associative, 2-cycle load-to-use, 3 ports, 64-byte blocks, 32 MSHRs, 16-entry victim cache
L2 NUCA Cache	private 512KB per core, 16-way set-associative, 14-cycle hit
Main Memory	4 GB memory, 8KB pages, 45 ns access latency
Memory Controllers	one controller per 4 cores, round-robin page interleaving
Interconnect	2D folded torus [16], 32-byte links, 1-cycle link latency, 2-cycle router, 1-flit control packets, 4-flit data packets
Cache Coherence Protocol	Four-state MOSI modeled after Piranha [8]

directory entries are managed by Dynamic Directories or by a traditional method.

IV. Evaluation Methodology

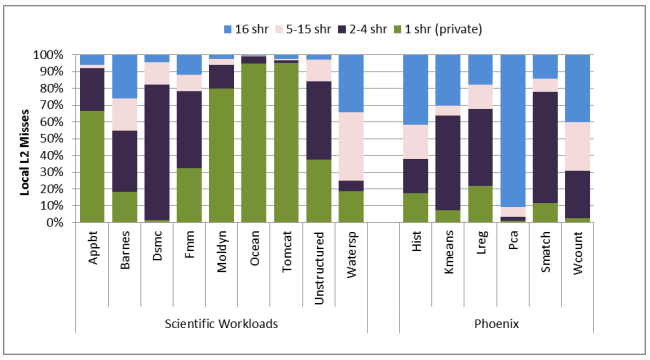
We motivate the deployment of Dynamic Directories through an analysis of the sharing patterns of two different categories of benchmark suites: Scientific and Map-Reduce, which are described in Table I. The scientific benchmark suite consists of a mixture of compute-intensive applications and computational kernels. Phoenix consists of data-intensive applications that use Map-Reduce. We analyze the data sharing patterns across our application suite by collecting execution traces of each workload using Flexus [6], a full-system cycle-accurate simulator of multicores with non-uniform caches. The traces cover 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. The workloads execute on a 16-core tiled CMP similar to Figure 1. The architectural parameters of the simulated CMP are depicted in Table II.

A. Analysis of Access Patterns

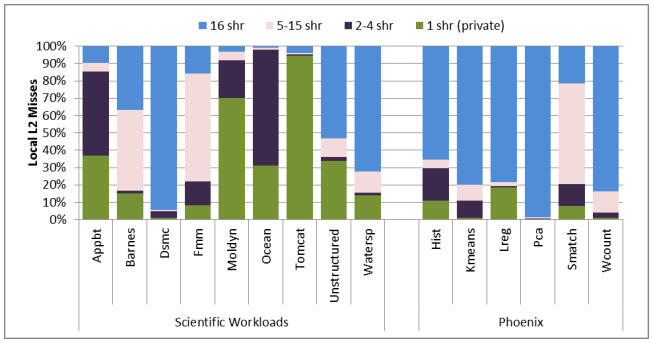
A core first searches for data in its local L2 cache. If it misses, then a directory access for the corresponding block follows. For each workload, Figure 3a shows the percentage of misses to the local L2 cache on blocks that are accessed by only one core during the execution of the program (1 shr, i.e., private blocks), accessed by few cores (2-4 shr), accessed by a large number of cores (5-15 shr), and blocks that are universally shared (16 shr). Each local L2 miss results in a directory access.

As described in Section 3, placing the directory of private blocks in the same tile with the core accessing these blocks will eliminate two control messages for every local L2 miss. In contrast, conventional address-interleaved directory placement will co-locate the directory and the requestor only a small fraction of the time. For the cases where the accesses are to blocks with a few sharers (2-4), co-locating the directory with one of the requesting cores will significantly increase the probability that the directory and the requester are in the same tile, which will also lead to the elimination of two messages. As the number of sharers increases, this probability decreases; in the case of universal sharing (16 shr), conventional address-interleaved directory placement will always co-locate the directory with one of the sharers, hence our proposed scheme will provide no additional benefit.

Figure 3a shows that the scientific and Phoenix applications exhibit a significant fraction of directory accesses for blocks that are private or have a few sharers. Averaged across all 15 workloads, 35% of the directory accesses are for private data and 33% of the accesses are for data shared among 2-4 cores. However, there are some exceptions to this behavior: pca has a large fraction of universally shared data. Nevertheless, our analysis suggests that in a large majority of applications, the most frequently accessed directories are either for private data or for data with a few sharers, motivating the use of Dynamic Directories.



(a)



(b)

Figure 3. (a) Access sharing pattern based on the number of sharers per cache block. (b) Access sharing pattern based on the number of sharers per page.

Dynamic Directories determine the placement of a directory at the page granularity (i.e., all the directory entries for the blocks within a page are located in the same tile). Hence, the sharing pattern at the page granularity determines the overall performance of our scheme. Similar to Figure 3a, Figure 3b shows the percentage of local L2 misses (i.e., directory accesses) on blocks that are within pages accessed by some number of cores during the execution of the workload. Averaged across all 15 applications, 23% of the accesses are on pages that are private and 13% of the accesses are on pages with 2-4 sharers. Thus, operating at page granularity does not introduce drastically more false sharing.

V. Experimental Results

A. Simulation Framework

We evaluate Dynamic Directories using the SimFlex multiprocessor sampling methodology [18]. Our samples are drawn over an entire parallel execution (Map phase) of the Phoenix workloads, and three iterations of the scientific applications. We launch measurements from checkpoints with warmed caches, branch predictors, TLBs, on-chip directories, and OS page tables, then warm queue and interconnect state for 100,000 cycles prior to measuring performance for 200,000 cycles. We use the aggregate number of user instructions committed per cycle as our performance metric, which is proportional to overall system throughput [18].

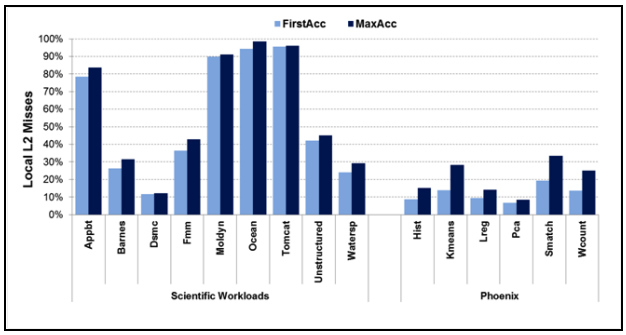


Figure 4. Effectiveness of the Dynamic Directories placement policy.

B. *Power Model*

In this work we have modeled the on-chip interconnection power as opposed to the chip power dissipation. Hence, our power savings are relative to the base power dissipation in the interconnection network. The power-model consists of two parts: modeling on-chip interconnect energy dissipation and calculation of overall execution time. With this the power dissipation for a given application (A) can be defined as:

$$Power_A = Interconnect\_Energy_A \Big/ Execution\_Time_A$$

The on-chip interconnect of our base architecture is a 2D folded torus. To calculate energy dissipation in the interconnection fabric, we first calculate the total number of hops (H) for each transaction over the network. The total number of hops (H) of a transaction (T) is the sum of the hops of all flits transmitted for the transaction. In our base architecture, control packets are 1-flit wide whereas a data packet consists of 4 flits (Table II). Next, we multiply the hop count (H) by the energy dissipation per flit per hop (α) to calculate the overall hop energy for each transaction (T). This is summed up over all memory transactions (control and data) to calculate the total interconnect energy.

$$Energy_A = \sum_T H.\alpha$$

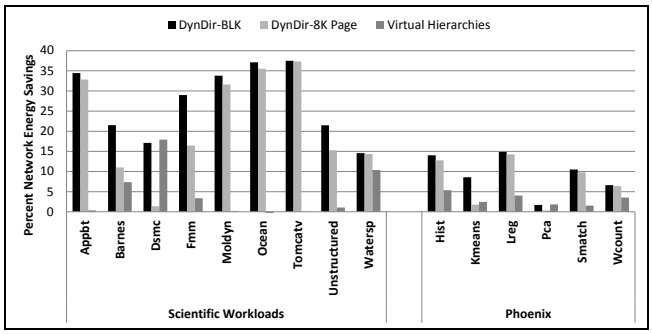
We do not need to estimate the lump energy constant (α), as we are interested in the relative power of Dynamic Directories over the baseline, so the constant (α) cancels out.

C. *Directory Placement Policy*

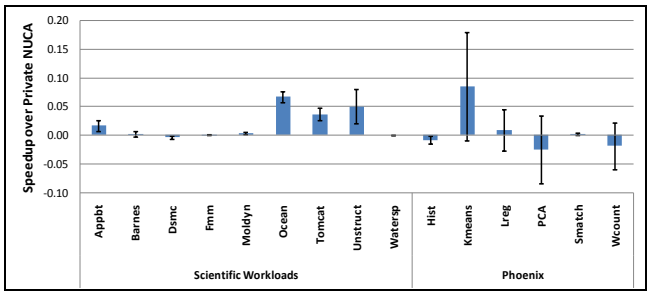
To evaluate the effectiveness of the directory placement policy, we compute the number of page accesses by the core that was the first ever to access the page (FirstAcc), and compare it against the accesses issued by the most frequent accessor for the same page (MaxAcc). From a power optimization standpoint and in the absence of directory migration, MaxAcc would be the ideal directory location for that page. As Figure 4 shows, allocating directory entries at the tile of the first accessor is a good approximation of the ideal scheme: the number of accesses issued by the first accessor is very close to the number of accesses issued by the most frequent accessor.

D. *Comparison with Alternative Schemes*

In addition to evaluating the power and performance impact of our scheme, we also compare against Virtual Hierarchies



(a)



(b)

Figure 5. (a) Network energy savings with Dynamic Directories at cache-block (DynDir-BLK) and 8K-page (DynDir-8k) granularities, and Virtual Hierarchies. (b) Speedup of Dynamic Directories over the baseline private NUCA architecture.

(VH), a directory migration technique proposed for server consolidation by Marty et al. [9]. The VH technique implements a two-level data coherence policy using a hypervisor. VH assigns specific home tiles for each memory region that have the same last bits in their block address; in other words, all accesses that miss a local tile are directed towards a home tile which is determined from a table indexed by the last bits of the block address. The home tile then sends/broadcasts the request to the appropriate owners. Since the home tile assignment policy used by the authors is not clearly indicated, we implement VH optimistically assuming perfect home tile placement, where the tile that makes the most accesses to a memory region is assigned to be the home tile for that specific region of the addresses. This way, any tile that accesses a block with the last few bits mapping to the memory region will be directing its request to the home tile for that region (upon a miss).

E. *Energy Savings*

Figure 5a presents the fraction of network energy saved by Dynamic Directories and VH respectively. For each application, the left and the middle bars indicate the energy savings attained by Dynamic Directories at the granularity of cache blocks and 8KB pages respectively. Dynamic Directories reduce the network energy by 20.4% and 16.1% on average for block- and page-granularity, respectively. As expected, the block granularity shows higher energy savings compared to the page granularity. However, as we describe in Section 5.2, such an implementation would complicate the design considerably (and will incur performance costs). The rightmost bar for each application presents the results for VH; VH saves 3.9% network energy on average. The power savings for both

Dynamic Directories and VH are largely achieved through a reduction of control messages in the network.

In general, we note that the scientific applications attain higher energy savings compared to Phoenix. Phoenix applications exhibit a higher fraction of shared data accesses (Section 4). As a result, Dynamic Directories are more useful for the scientific workloads. In fact, we observe a strong correlation between the sharing distribution (Figure 3b) and the energy reduction (Figure 5a) for each of the studied applications.

F. Performance Impact

Figure 5b shows the overall speedup of Dynamic Directories compared to a baseline private NUCA architecture. Interestingly enough, we observe that Dynamic Directories slightly increase performance in 7 out of 15 applications, and decrease performance in 2. Dynamic Directories improve performance by up to 7% (Ocean), and by 1.3% on average, while the maximum performance slowdown is 1.3% (PCA). The performance is improved due to two reasons. First, Dynamic Directories reduce the number of network packets, which may eliminate congestion and hence reduce the overall latency of network operations. Second, data transfers (on-chip and off-chip) are faster because the access to a remote directory is eliminated in many cases. Because the working set is large, Dynamic Directories' savings are realized mostly by off-chip memory accesses. As the off-chip memory access latency is already large, saving a small number of cycles does not improve the performance considerably.

We attribute the slowdown exhibited by two of the applications (PCA and Wcount) to the fact that Dynamic Directories assign directories for a whole page to one tile. If it fails to reduce the number of network packets, this assignment can cause contention and hotspots. Especially for universally-shared pages, it is likely that blocks are accessed by different cores in nearly consecutive cycles, causing contention in the directory tile, and increasing the directory's response time. On average, we observe that the positive and negative forces cancel each other out, and Dynamic Directories have only a negligible overall performance impact.

VI. CONCLUSION

As processor manufacturers strive to deliver higher performance within the power and cooling constraints of modern chips, they struggle to reduce the power and energy consumption of the most insatiable hardware components. Recent research shows that on-chip interconnection networks consume 20% to 36% of a chip's power, and their importance is expected to rise with future process technologies. In this paper, we observe that a large fraction of the on-chip interconnect traffic stems from placing directory entries on chip without regards to the data access and sharing patterns. Based on this observation, we propose Dynamic Directories, a distributed directory architecture that cooperates with the operating system to place directory entries close to the most active requestors of the corresponding cache blocks, eliminating unnecessary network traversals and conserving energy and power. The mechanisms we propose exploit already existing hardware and operating system structures and events, have negligible overhead, and are easy and practical to implement. Through trace-driven and cycle-accurate simulation on a range of scientific and Map-Reduce applications, we show that Dynamic Directories reduce the power and energy expended by the on-chip interconnect by up to 37% (16.4% on average) while attaining a small improvement in performance (1.3% on average).

REFERENCES

[1] M. Azimi, et al., "Integration challenges and trade-offs for tera-scale architectures," Intel Technology Journal, vol. 11, pp. 173-184, 2007.

[2] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," SIGARCH Comput. Archit. News, vol. 34, pp. 264-276, 2006.

[3] S. Cho and L. Jin, "Managing distributed, shared L2 caches through OS-level page allocation," 2006, pp. 455-468.

[4] B. Cuesta, et al., "Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks," in ISCA, San Jose, CA, 2011, pp. 93-103.

[5] N. Hardavellas, et al., "Reactive NUCA: near-optimal block placement and replication in distributed caches," in Proceedings of the 36th annual international symposium on Computer architecture Austin, TX, 2009.

[6] N. Hardavellas, et al., "Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," ACM SIGMETRICS Performance Evaluation Review, vol. 31, pp. 31-34, 2004.

[7] J. S. Kim, et al., "Energy characterization of a tiled architecture processor with on-chip networks," in International Symposium on Low Power Electronics and Design Seoul, Korea 2003.

[8] G. Kourosh, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," ACM SIGARCH Computer Architecture News, vol. 28, pp. 282-293, 2000.

[9] M. Marty and M. Hill, "Virtual hierarchies to support server consolidation," ACM SIGARCH Computer Architecture News, vol. 35, pp. 46-56, 2007.

[10] S. Mukherjee, et al., "The Alpha 21364 network architecture," in IEEE MICRO, 2002, pp. 26-35.

[11] C. Ranger, et al., "Evaluating mapreduce for multi-core and multiprocessor systems," in HPCA, pp. 13–24.

[12] A. Ros, et al., "DiCo-CMP: Efficient cache coherency in tiled CMP architectures," in IPDPS, 2008, pp. 1–11.

[13] L. Shang, et al., "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," in HPCA, Anaheim, CA, 2003.

[14] L. Shang, et al., "PowerHerd: dynamic satisfaction of peak power constraints in interconnection networks," in ICS, 2003, p. 108.

[15] L. Shang, et al., "Thermal modeling, characterization and management of on-chip networks," in IEEE MICRO, 2004, pp. 67-78.

[16] B. Towles and W. J. Dally, "Route packets, not wires: On-chip interconnection networks.," presented at the 38th Design Automation Conference (DAC), 2001.

[17] H. Wang, et al., "Power-driven design of router microarchitectures in on-chip networks," in IEEE MICRO, 2003.

[18] T. Wenisch, et al., "SimFlex: statistical sampling of computer system simulation," in IEEE MICRO, 2006, p. 18.

[19] S. C. Woo, et al., "The SPLASH-2 programs: characterization and methodological considerations," in ISCA, S. Margherita Ligure, Italy, 1995, pp. 24-36.

[20] J. Zebchuk, et al., "A tagless coherence directory," in IEEE MICRO, 2009, pp. 423-434.

[21] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in ISCA, 2005, pp. 336-345.