PAD: Power-Aware Directory Placement in Distributed Caches

Technical Report NWU-EECS-10-11
Electrical Engineering and Computer Science (EECS)
Northwestern University

Authors

Abhishek Das
Matt Schuchhardt
Nikos Hardavellas
Gokhan Memik
Alok Choudhary

Document created by:
Abhishek Das

December 2010

ABSTRACT

On-chip interconnection networks already consume a large fraction of the chip's power, and the rapidly increasing core counts in future technologies further aggravate the problem. The interconnect facilitates the active sharing of data on chip. However, 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 data coherence in the presence of conflicting updates. We observe that the majority of such traffic is unnecessary, and stems from the uncoordinated and data-access-oblivious placement of the directory structures on chip. In this paper, we propose Power-Aware Directory Placement (PAD), a novel distributed cache architecture that co-locates directories together with highly active sharers of the corresponding data, thereby eliminating a large fraction of the on-chip interconnect traversals. Through trace-driven and cycle-accurate simulation in a range of scientific and Map-Reduce applications, we show that PAD 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 (1.4% on average).

1. 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 [11, 34]. To facilitate data transfers and communication among the cores, such processors employ elaborate on-chip interconnection networks. The on-chip interconnection networks are typically optimized to deliver high bandwidth and low latency.

However, such on-chip interconnects come at a steep cost. Recent studies show that on-chip networks consume between 20% to 36% of the power of a multicore chip [16, 19]and significantly raise the chip temperature [25] 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.

The flurry of recent research to minimize the power consumption of on-chip interconnects is indicative of the importance of the problem. Circuit-level techniques to improve the power efficiency of the link

circuitry and the router microarchitecture [29], dynamic voltage scaling [23] and power management [16, 24], and thermal-aware routing [25] promise to offer a respite, at the cost of extensive re-engineering of the interconnect circuitry and routing protocols. Yet, 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 for fast data accesses, the distributed cache slices are typically treated as private caches to the nearby cores [4, 5, 34], forming tiles with a core and a cache slice in each tile [2, 11]. Private caches allow the replication of shared data, which, in turn, require a mechanism to keep the data coherent in the presence of updates. To allow scaling to high core counts and facilitate coherent data sharing, modern multicores employ a directory structure, which is typically address-interleaved among the tiles [4, 11, 34].

However, 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. Ideally, from a power-optimization standpoint, the directory entry for a cache block would be co-located with the most active sharing core of the block, rather than a seemingly random one.

In this paper, we observe that a large fraction of the on-chip interconnect traffic stems from the data-access-oblivious placement of directory entries. Based on this observation, we propose a distributed cache 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, they are easy and practical to implement, and can even slightly improve performance. In summary, the 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 Power-Aware Directory placement (PAD), 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 (including SPLASH-2 [31]) and Map-Reduce [20] workloads, we show that PAD reduces the interconnect energy and power by up to 37% (22% on average for the scientific 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 2 presents related work. Section 3 provides background information on distributed caches. Section 4 introduces Power-Aware Directory co-location (PAD), followed by the details of the proposed architecture in Section 5. The results of our evaluation are presented in Section 6. Finally, we conclude the paper with a short summary in Section 7.

2. RELATED WORK

To mitigate the access latency of large on-chip caches, Kim *et al.* propose Non-Uniform Cache Architectures (NUCA) [15], showing that a network of cache banks can be used to reduce average access latency. NUCA caches and directory management in chip multiprocessors (CMPs) have been explored by several researchers in the past. We classify some of the most notable works in the following areas:

Cache block placement: To improve cache performance, Dynamic NUCA [15] attracts cache blocks to the requesting cores, but requires complex lookup algorithms. CMP-NuRAPID [7] decouples the physical placement of a cache block from the cache's logical organization to allow the migration and replication of blocks in a NUCA design, but requires expensive hardware pointers for lookup, and coherence protocol modifications. Cooperative Caching [5] proposes the statistical allocation of cache blocks in the local cache to strike a balance between capacity and access speed, but requires centralized structures that do not scale, and the allocation bias is statically defined by the user for each workload. ASR [4] allows the allocation bias to adapt to workload behavior, but requires complex hardware tables and protocol modifications. R-NUCA [11] avoids cache block migration in favor of intelligent block placement, but distributes shared data across the entire die. Huh advocates NUCA organizations with

static block-mapping and a small sharing degree [12], but the mapping is based on the block's address and is oblivious to the data access pattern. Kandemir proposes migration algorithms for the placement of cache blocks [14] and Ricci proposes smart lookup mechanisms for migrating blocks in NUCA caches using Bloom filters [21]. PDAS [32] and SP-NUCA [18] propose coarse-grain approaches of splitting the cache into private and shared slices. However, none of these works optimize for power and energy. Nahalal [10] builds separate shared and private regions of the cache, but the block placement in the shared cache is statically determined by the block's address. Finally, Page-NUCA [6] dynamically migrates data pages to different nodes whenever the system deems it necessary, but requires hundreds of KB to MB of extra storage which scales linearly to the number of cores and cache banks, and complicated hardware mechanisms and protocols. Overall, all these schemes place data blocks and optimize for performance; PAD places directories and optimizes for power and energy. Thus, PAD is orthogonal to them and can be used synergistically. Moreover, PAD does not require complex protocols or hardware.

Software-driven schemes: OS-driven cache placement has been studied in a number of contexts. Sherwood proposes to guide cache placement in software [26], suggesting the use of the TLB to map addresses to cache regions. Tam uses similar techniques to reduce destructive interference for multi-programmed workloads [27]. Cho advocates the use of the OS to control partitioning and cache placement in a shared NUCA cache [13]. PAD leverages these works to guide the placement of directories using OS mechanisms, but, unlike prior proposals, places directories orthogonally to the placement of data.

Coherenceschemes: Several proposals suggest novel coherence mechanisms to increase the cache performance. Dico-CMP [22] extends the cache tags to keep sharer information. Zebchuk proposes a bloom-filter mechanism for maintaining tagless cache coherency [33]. These proposals are orthogonal to PAD,which co-locates the directory with a sharer, as opposed to changing the cache coherence protocol.

Distributed shared memory (DSM) optimizations: Zhang observes that different classes of accesses benefit from either a private or shared system organization [35] in multi-chip multiprocessors. Reactive NUMA [9] dynamically switches between private and shared cache organizations at page granularity. Marchetti proposes first-touch page placement to reduce the cost of cache fills in DSM systems [36].

Overall, these techniques optimize performance in DSM systems. In contrast, PAD introduces a mechanism that decouples a block's address from its directory location, allowing the directory to be placed anywhere on chip, without space or performance overhead, and without complicating lookup. PAD uses this mechanism to minimize power and energy in CMPs. Finally, the first-touch directory placement policy is only used because it is simple and effective, and is orthogonal to the PAD mechanism.

On-chip power: Several studies optimize power for on-chip interconnects. Balfour optimizes router concentration for higher energy efficiency [3]. Wang proposes circuit-level techniques to improve the power efficiency of the link circuitry and the router microarchitecture [29]. Shang proposes dynamic voltage and frequency scaling of interconnection links [23], dynamic power management [24], and thermal-aware routing [25] to lower the power and thermal envelope of on-chip interconnects. All these techniques are orthogonal to PAD, which focuses on reducing the overall hop count and number of network messages by changing the directory placement. PAD can work synergistically with prior proposals to lower power and enhance the energy efficiency even further.

3. BACKGROUND

We assume a tiled multicore, where each tile consists 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. Figure 1 depicts a typical tiled multicore architecture. We assume a private NUCA organization of the distributed L2 cache [11], where each L2 slice is treated as a private L2 cache to the local core within the tile.

On-chip distributed caches in tiled multicores typically use a directory-based mechanism to maintain coherence [4, 11, 34]. To scale to high core counts, the directory is also 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 the ideal case, the directory has the capacity to hold coherence information for all the cache blocks across all the tiles in all cases (i.e., a full-map directory). Techniques like sparse directories reduce the capacity requirements of full-map ones. However, the investigation of directory capacity-management mechanisms is beyond the scope of this paper. Without loss of generality, and similarly to most relevant works, we assume a full-map directory for the baseline and PAD architectures. In general, PAD uses the same directory mechanisms as the baseline architecture.

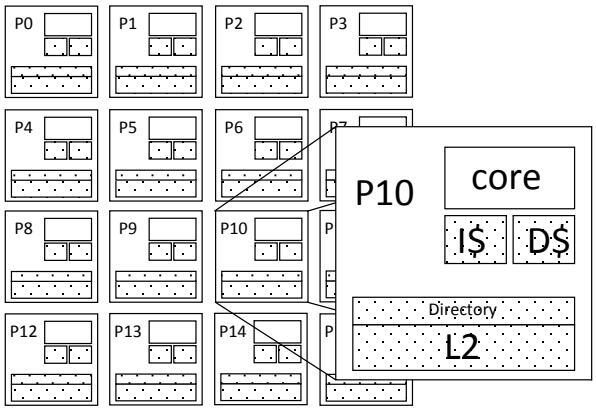

Figure 1.Baseline tiled architecture of a 16-core CMP. Each tile has core, split I/D L1, L2 and directory slice.

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-a 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. 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, the directory would be co-located with the sharer at Tile 1 (Figure 2-b), which would eliminate two unnecessary network messages and result in reduced power consumption and faster data access. Such placement is the goal of PAD.

A similar message sequence is generated upon requests than result in off-chip misses. In such a case, even if the accessed data are private, the requesting core first contacts the corresponding directory, which then sends a message to the memory controller. When the data is available, the memory controller sends a message to the directory node and the data to the requestor. If the directory is co-located with the requesting core using PAD, one or two messages are eliminated (in an aggressive design, the data and directory replies may be combined into a single message, as they both go to the same destination tile).

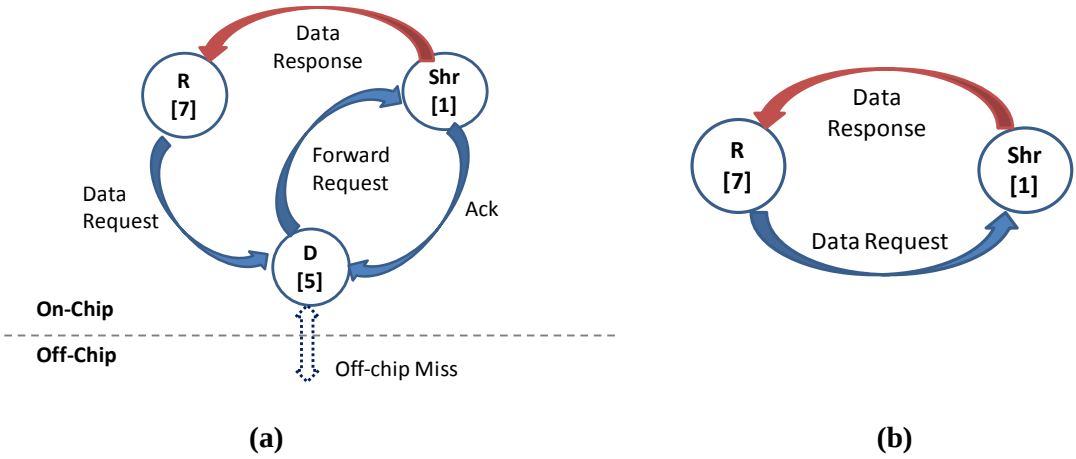

Figure 2. (a) Sequence of on-chip network messages following a request by tile 7 for a block owned by tile 1, with its directory at tile 5. (b) The same when the owner tile 1 also holds the directory entry.

The fewer the sharers of a block, the higher the impact of intelligent directory placement. A block that is private to a core and has its directory entry within the same tile can be accessed without any intermediate directory nodes participating in resolving local cache misses. A block with a couple of sharers and its directory co-located with one of them can be shared through direct communication among the sharers, also without the need to access an intermediate node. At the far end of the sharing spectrum, a universally-shared block that all cores access with similar frequency cannot benefit from the intelligent placement of its directory entry, because any location is as good as any other. Thus, applications with a large fraction of data with one sharer (private) or with few sharers (2-4) will benefit the most from PAD. In Section 4 we show that there is a considerable fraction of accesses to blocks with one or few sharers in a variety of parallel applications, rendering traditional address-interleaved directory placement inefficient.

Note that in an *N*-tile multicore 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 the core counts grow. This has a twofold impact, both on the power and the performance of the chip. The extra hops to a remote directory increase the on-chip network power usage, and the extra hop latency increases the access penalty for a block that missed locally. Our proposal, **Power-Aware Directory placement (PAD)**, promises to mitigate both effects.

Table 1.Description of workloads.

Benchmark		Application	Description
Scientific	NAS	*appbt*	Solves multiple independent systems of equations
	SPEC Parallel	*tomcatv*	A parallelized mesh generation program
	Other Scientific	*dsmc*	Simulates the movement and collision of gas particles
		moldyn	Molecular dynamics simulation
		unstructured	Computational fluid dynamics application
	SPLASH-2	*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	*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

Table 2.System parameters for the simulated framework.

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 [8,28], 32-byte links, 1-cycle link latency, 2-cycle router
Cache Coherence Protocol	Four-state MOSI modeled after Piranha [17]

4. OVERVIEW OF POWER-AWARE DIRECTORY PLACEMENT (PAD)

At a high level, PAD utilizes the virtual address translation mechanism to assign directories to tiles at page granularity. The first time a page is accessed, the tile of the accessing core becomes the owner of the directories for that page (directory information is still maintained at cache block granularity; only the placement of the entries to tiles is done at the page level). This information is stored in the page table and propagated to the TLB. If another core accesses the same page, the directory location is provided along with the physical address of the page. Therefore, a second core accessing the same page can directly contact the directory entries for blocks in that page. Thus, PAD decouples the address of a block from the physical location of its directory, allowing the directory to be placed anywhere on chip without

complicating lookup. In the remainder of this section, we motivate the deployment of PAD through an analysis of the sharing patterns of scientific and Map-Reduce applications.

4.1 Experimental Methodology

We evaluate PAD on two different categories of benchmark suites: 'scientific' and Phoenix [20], which are described in more detail in Table 1. The scientific benchmark suites consist 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 SimFlex [30], a full-system cycle-accurate simulator of multicores with distributed 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 supported by a 2D folded torus interconnect similar to [11]. The architectural parameters for our baseline configuration are depicted in Table 2.

4.2 Analysis of Sharing 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 shows the percentage of local L2 misses (i.e., directory accesses) 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).

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.

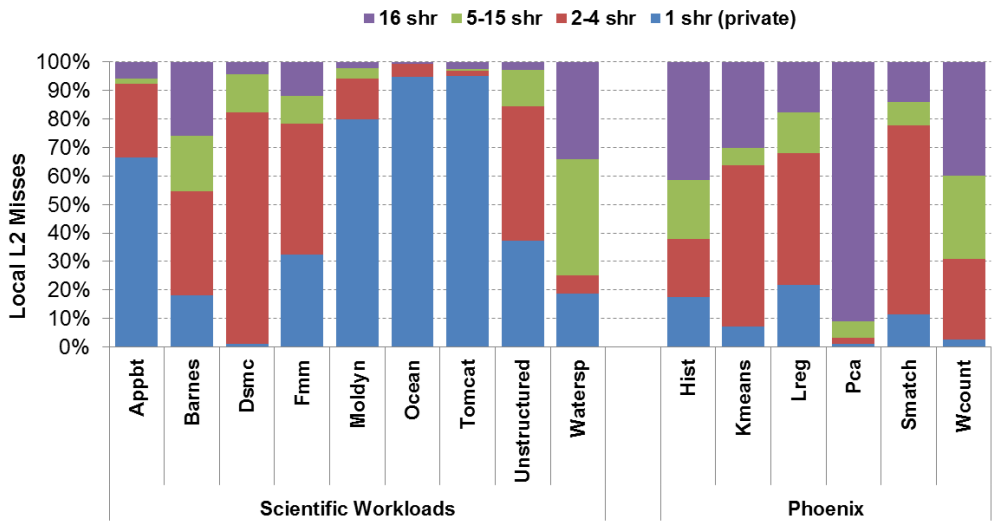

Figure 3. Access sharing pattern at the block level based on number of sharers per block.

Figure 3 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* exhibits a large fraction of universally shared data. Nevertheless, our analysis suggests that in a large majority of applications, the most accessed directories are either for private data or for data with a few sharers, motivating the use of PAD.

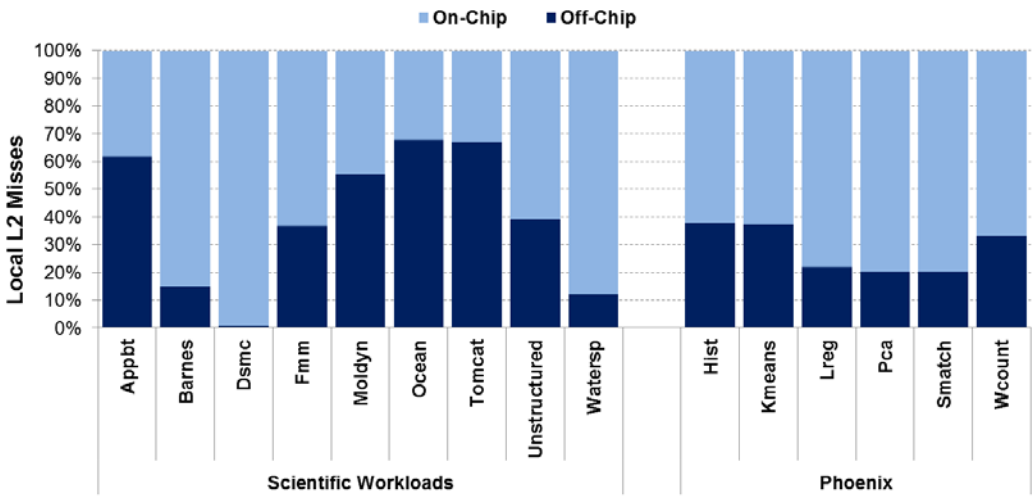

Figure 4. Accesses breakdown by off-chip and on-chip accesses.

Figure 4 illustrates the breakdown of local L2 misses based on whether the access is made to an off-chip block or to a block that resides in some remote tile on chip. Both types of accesses have to access the directory first. In the case of an off-chip miss, the directory sends a message to the memory controller to fetch the block. In an aggressive protocol, the memory controller sends the data directly to the requestor, and a reply acknowledgement to the directory. In a conservative protocol, the memory controller sends the data to the directory, which then forwards it to the requestor. In either case, the directory is informed about the cache fill. If the requestor and the directory entry are in different tiles, four messages are generated; PAD could eliminate two of them, by placing the directory entry together with the requestor.

In the case of a local miss to a block that resides on chip, the directory sends a request to the owner of the block, which then sends the data to the requestor and an acknowledgement to the directory, so that it can update the coherence state of the block and finalize the transaction. Hence, for cache-to-cache transfers, a total of four messages are generated (one of which will be avoided if the directory resides with the requesting core, and two will be avoided if the directory resides with the owner of the block).

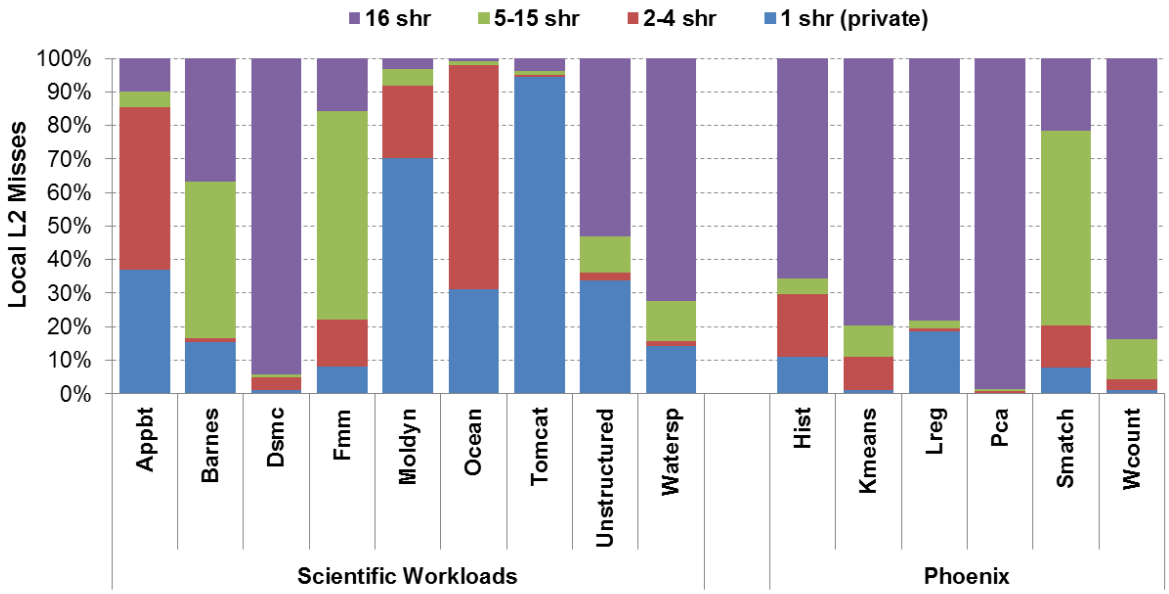

Figure 5. Access sharing pattern at the page level based on number of sharers per page.

PAD determines the placement of a directory at the page granularity (i.e., all the directory entries for the blocks within in 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 3, Figure 5 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, although working at the page granularity introduces false sharing, the change is not drastic as compared to block granularity.

5. POWER-AWARE DIRECTORY PLACEMENT (PAD)

Power-Aware Directory placement (PAD) 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, PAD designates an owner of the directory entries for the blocks in that page, and stores the owner ID in the page table. By utilizing the already existing virtual-to-physical address translation mechanism, PAD propagates the directory owner location to all cores touching the page. There are three important aspects of this scheme: the classification of pages by the OS, the directory placement mechanism, and the distribution of directory owners among cores. We investigate these aspects in the following sections.

5.1 Operating System Support

To categorize pages and communicate their directory location to the cores, PAD piggybacks 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 PAD by slightly modifying this process. When a page is accessed for the first time ever by any of the cores, the first accessor becomes the owner of the corresponding directory entries (this is called *first-touch directory placement* and we discuss its effects in the next section). This information is stored in the page table. Upon a TLB fill, the OS (or the hardware page walk mechanism) provide this owner information to the core along with the translation, and store it in its TLB. 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. If the page is privately accessed, the tile of the accessing core will hold the directory entries for all the blocks in the page.

5.2 Discussion

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 that employing finer-grain techniques is unjustified. Nevertheless, in the next section, we provide hypothetical energy savings of such an approach; our results indicate that, for most applications, finer granularity provides negligible benefits.

Directory placement 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 in performance, and may complicate other optimizations (e.g., page coloring for L1) that pose conflicting address translation requests. PAD avoids these problems by fully decoupling page allocation from directory placement.

While pathological cases 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 [11], so the pages should distribute evenly. It is important to note here that it is simple to turn off PAD in pathological cases: one bit per page could indicate whether these entries are managed by PAD or a traditional method. Finally, in the case of heavily-migrating threads, the corresponding directory entries could either stay in the original tile and be accessed remotely by the migrating thread (similar to the baseline), or move along with it, or we could simply turn off PAD as described above. While dynamic directory migration is possible under our scheme, the complexities it entails may overshadow its benefits. Hence, we leave the investigation of on-chip directory migration schemes to future work.

6. EXPERIMENTAL RESULTS

6.1 Methodology

We evaluate PAD using the SimFlex multiprocessor sampling methodology [30]. 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 [30]. The architectural parameters are described in Section 4.1.

6.2 First Touch Directory Placement

To evaluate the effectiveness of the first-touch 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 6 shows, first-access directory placement is a good approximation of the ideal scheme: the number of accesses issued by the first accessor is very close to the same issued by the maximum accessor.

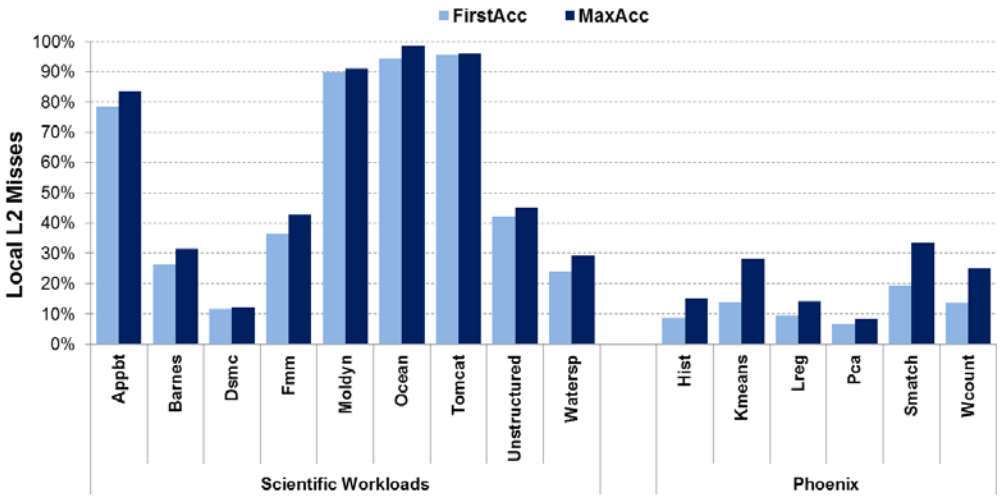

Figure 6. Effectiveness of the first-touch directory placement policy.

6.3 Distribution of Directory Entries Across Tiles

The first-touch directory placement policy may result in an imbalanced distribution of directory entries to tiles, in contrast to the almost even distribution under traditional address interleaving. If some tiles are assigned vastly more directory entries than others, they would require a disproportionately large area for the directory, or could result in traffic hotspots that degrade performance.

Figure 7 presents the distribution of directory entries under the first-touch placement policy across tiles for private and shared pages. The red line indicates the level of a hypothetical uniform distribution. The uneven distribution of directory entries is an artifact of the first-touch directory placement policy and could be minimized. First, only the shared pages matter; private data are accessed by only one core so they are always coherent, obviating the need for a directory. Thus, PAD can defer the directory entry allocation until a page is accessed by a second core, ensuring that directories exist only for shared pages and conserving directory area. With the exception of Kmeans and Fmm, shared entries for the remaining applications are mostly evenly distributed (in the remaining applications, a tile gets at most 18% of the

total entries). Second, PAD can minimize the uneven distribution by utilizing a "second-touch" placement policy (i.e., the second sharer being assigned the entries) when the first-touching tile is overloaded. Third, the uneven distribution of pages is not a direct indicator of increased traffic hotspots, as some pages are colder than others, and the baseline may also exhibit imbalanced traffic.

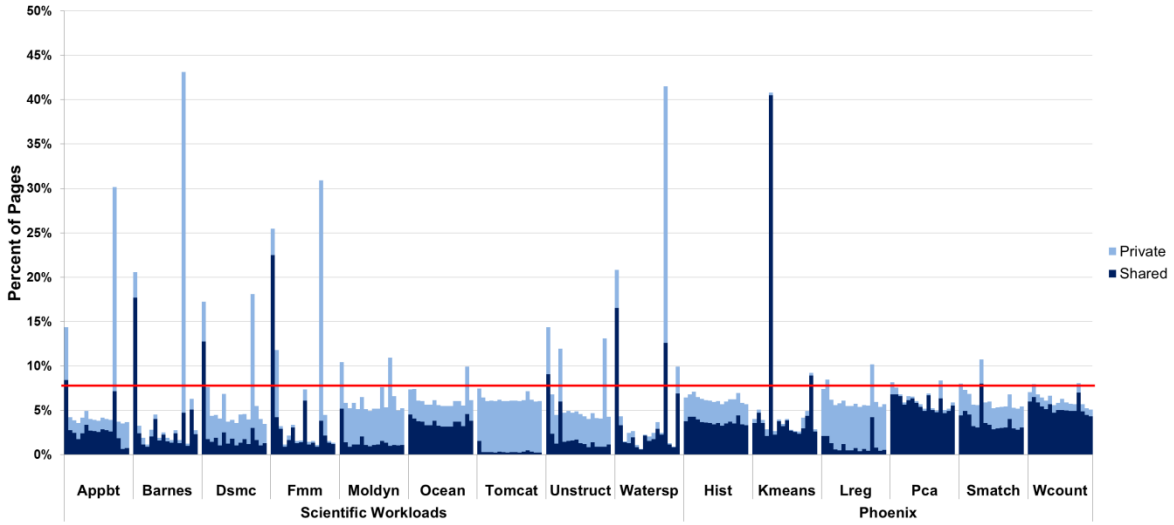

Figure 7. Distribution of directory entries for pages across tiles under the first-touch placement policy.

To investigate hotspots, we analyzed the accesses to each directory tile under PAD and baseline[1]. Our results indicate that PAD reduces the number of network messages for most applications. In some cases, it even cuts the number of control messages by almost half (appbt, moldyn, ocean, and tomcatv; see Figure 9). With the exception of Kmeans, Fmm and Dsmc, the remaining applications exhibit a slightly higher imbalance than baseline, with a tile receiving at most 16% more directory accesses from remote cores (8% more on average). These imbalances are relatively small and do not impact the overall performance (Figure 10). Applying PAD on Kmeans, Fmm, and Dsmc exacerbates already existing traffic imbalances. However, we find that even these hotspots have a negligible performance impact, due to the already small fraction of execution time these applications spend on the distributed L2 cache.

6.4 Energy Savings

Figure 8 presents the fraction of network energy saved by PAD. For each application, the left bar indicates the energy savings attained by PAD at cache-block granularity, while the right bar presentsPAD

[1] We omit the graph due to space constraints, and instead present our findings in the text.

for 8KB pages. PAD reduces the network energy by 20.4% and 16.1% on average for block- and page-granularity, respectively, mainly by reducing the network messages. 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).

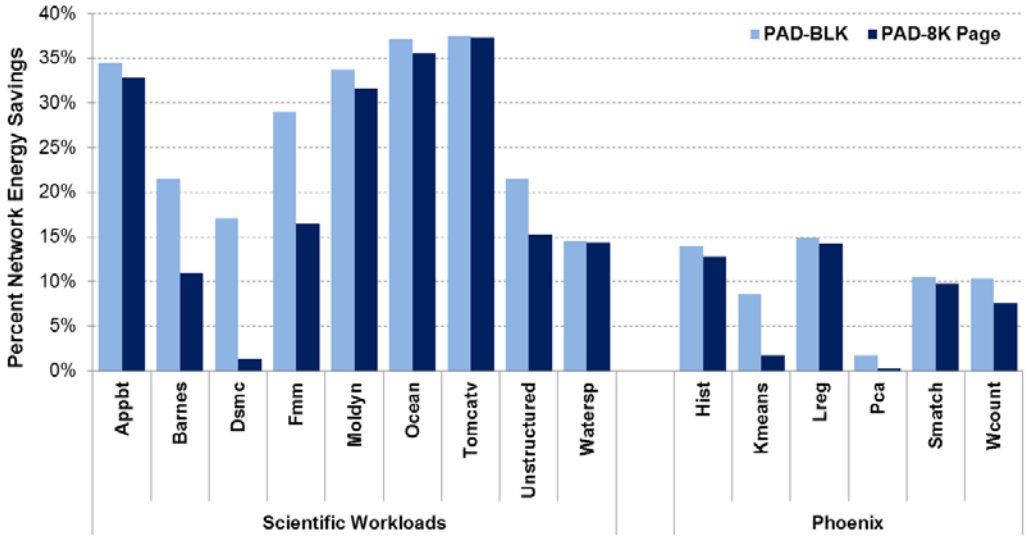

Figure 8. On-chip network energy savings obtained by block-grain and page-grain PAD.

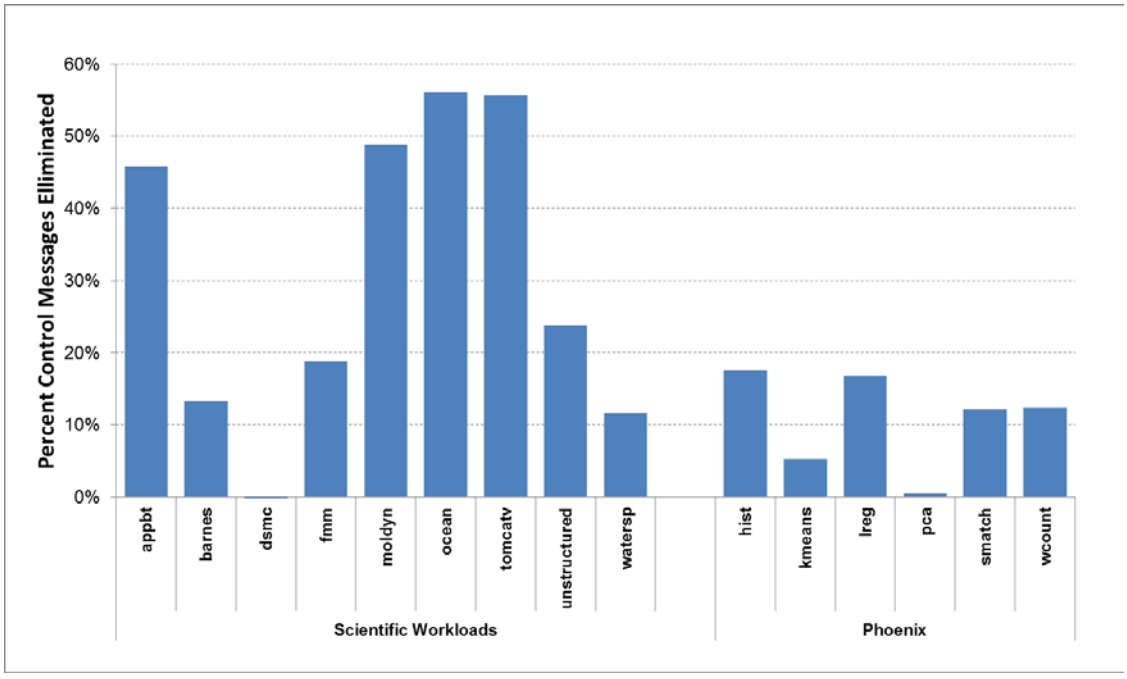

Figure 9. Reduction of network control messages attained by PAD with respect to Baseline.

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, our schemes are more useful for the scientific workloads. In fact, we observe a strong correlation between the sharing distribution (Figure 5) and the energy reduction (Figure 8) for each of the studied applications.

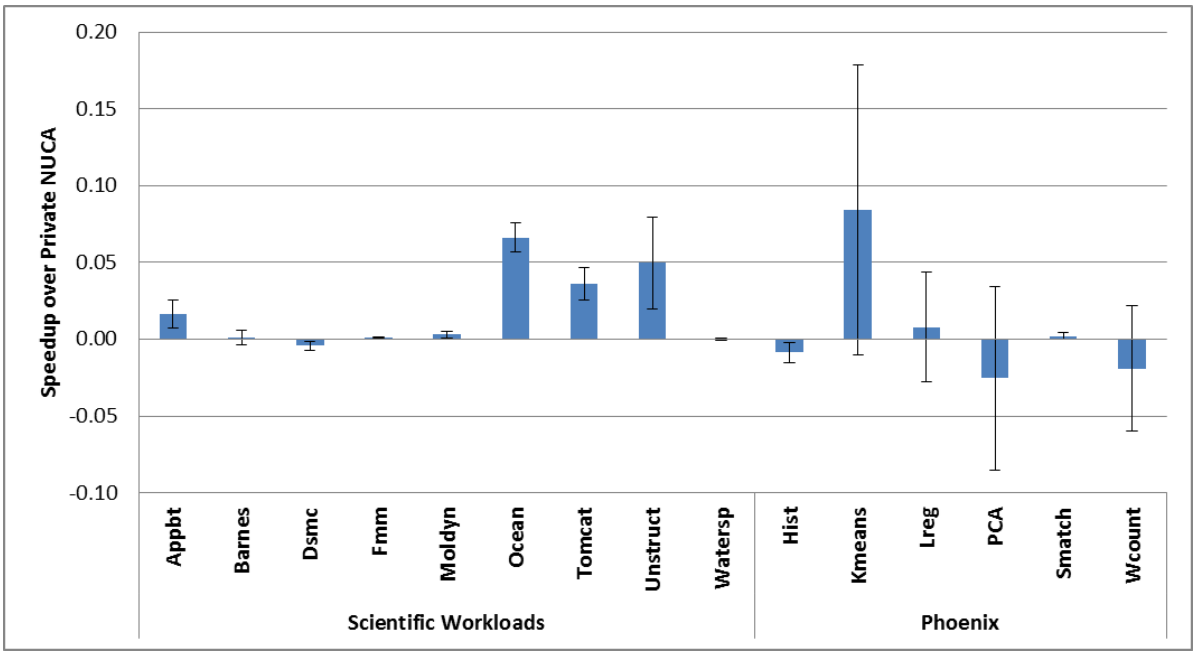

Figure 10.Speedup of PAD over the baseline private NUCA architecture.

6.5 Performance Impact

Figure 10 shows the overall speedup of PAD as compared to a baseline private NUCA architecture. Interestingly enough, we observe that PAD slightly increases performance in 7 out of 15 applications, and decreases performance in 2. PAD improves 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, PAD reduces 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, PAD's 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.

The reason for the slowdown exhibited by a couple of the applications is attributed to the fact that PAD assigns directories for a whole page to one node. 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 PAD has only a negligible overall performance impact.

7. CONCLUSIONS

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 traffic stems from placing directory entries on chip without regards to the data access and sharing patterns. Based on this observation, we propose Power-Aware Directory placement (PAD), a distributed cache 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 PAD reduces the power and energy expended by the on-chip network by up to 37% (16.4% on average) while attaining a small improvement in performance (1.3% on average). Thus, we believe PAD is an appealing technique that shows great promise in reducing the power and energy of the on-chip interconnect, with negligible overheads.

REFERENCES

[1] Virtutech Simics. Available: http://www.virtutech.com/

[2] M. Azimi, N. Cherukuri, D. N. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, A. S. Vaidya, "Integration challenges and trade-offs for tera-scale architectures," Intel Technology Journal, vol. 11, pp. 173-184, 2007.

[3] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," ICS, pp. 198, 2006.

[4] B. M. Beckmann, M. R. Marty, D. A. Wood, "ASR: Adaptive selective replication for CMP caches," MICRO, pp. 443-454, 2006.

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

[6] M. Chaudhuri, "PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," HPCA, pp. 227-238, 2009.

[7] Z. Chishti, M. D. Powell, T. N. Vijaykumar,"Optimizing replication, communication, and capacity allocation in CMPs," ISCA, pp. 357-368, 2005.

[9] B. Falsafi and D. A. Wood, "Reactive NUMA: a design for unifying S-COMA and CC-NUMA," SIGARCH, vol. 25, pp. 240, 1997.

[10] Z. Guz, I. Keidar, A. Kolodny, U. C. Weiser, "Utilizing shared data in chip multiprocessors with the Nahalal architecture,"SPAA, pp. 1-10, 2008.

[11] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches" ISCA, 2009.

[12] J. Huh,C. Kim, H. Shafi, L. Zhang, D. Burger, S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing,"TPDS, vol. 18, 2007.

[13] S. Cho and L. Jin, "Managing distributed, shared L2 caches through OS-level page allocation," MICRO, 2006.

[14] M. Kandemir, F. Li, M. J. Irwin, S.W. Son, "A novel migration-based NUCA design for chip multiprocessors," SC, pp. 1-12, 2008.

[15] C. Kim, D. Burger, S.W. Keckler, "An Adaptive, Non-uniform Cache Structure for Wire-delay Dominated On-chip Caches,"ASPLOS, pp. 211-222, 2002.

[16] J. S. Kim, M. B. Taylor, J. Miller, D. Wentzlaff, "Energy characterization of a tiled architecture processor with on-chip networks," ISPLED, 2003.

[17] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer,B. Sano, S. Smith, R. Stets, B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," ISCA, pp. 282-282, 2000.

[18] J. Merino, V.Puente, P. Prieto, J. Á. Gregorio, "Sp-nuca: a cost effective dynamic non-uniform cache architecture," SIGARCH, vol. 36, pp. 64-71, 2008.

[19] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, D. Webb, "The Alpha 21364 network architecture," MICRO, vol. 22, pp. 26-35, 2002.

[20] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," HPCA, vol. 7, pp. 13–24, 2007.

[21] R. Ricci, S. Barrus, D. Gebhardt, R. Balasubramonian, "Leveraging bloom filters for smart search within NUCA caches," ISCA, 2006.

[22] A. Ros, M. E. Acacio, J. M. García, "DiCo-CMP: Efficient cache coherency in tiled CMP architectures," IPDPS, pp. 1–11, 2008.

[23] L. Shang, L. S. Peh, N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," HPCA, 2003.

[24] L. Shang, L. S. Peh, N. K. Jha, "PowerHerd: dynamic satisfaction of peak power constraints in interconnection networks," ICS, p. 108, 2003.

[25] L. Shang, L. S. Peh, A. Kumar, N. K. Jha, "Thermal modeling, characterization and management of on-chip networks," in MICRO, pp. 67-78, 2004.

[26] T. Sherwood, B. Calder, J. Emer, "Reducing cache misses using hardware and software page placement," ICS, pp. 164, 1999.

[27] D. Tam, R. Azimi, L. Soares, M. Stumm, "Managing shared L2 caches on multicore systems in software,"WIOSCA 2007.

[28] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks.," DAC, 2001.

[29] H. Wang, L. S. Peh, S. Malik, "Power-driven design of router microarchitectures in on-chip networks," MICRO, 2003.

[30] T. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, J. C. Hoe, "SimFlex: statistical sampling of computer system simulation," MICRO, vol. 26, pp. 18, 2006.

[31] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," ISCA, 1995.

[32] T. Yeh and G. Reinman, "Fast and fair: data-stream quality of service," CASES, pp. 248, 2005.

[33] J. Zebchuk, V. Srinivasan, M. K. Qureshi, A. Moshovos, "A tagless coherence directory," MICRO, pp. 423-434, 2009.

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

[35] Z. Zhang and J. Torrellas, "Reducing remote conflict misses: NUMA with remote cache versus COMA," HPCA, pp. 272, 1997.

[36] M. Marchetti, L. Kontothanassis, R. Bianchini, M. Scott, "Using simple page placement schemes to reduce the cost of cache fills in coherent shared-memory systems," IPPS, pp. 380–385, 1995.