NoC-Aware Cache Design for Multithreaded Execution on Tiled Chip Multiprocessors

Ahmed K. Abousamra  
University of Pittsburgh  
Computer Science Department  
Pittsburgh, PA  
abousamra@cs.pitt.edu

Alex K. Jones  
University of Pittsburgh  
Electrical and Computer Engineering Department  
Pittsburgh, PA  
akjones@ece.pitt.edu

Rami G. Melhem  
University of Pittsburgh  
Computer Science Department  
Pittsburgh, PA  
melhem@cs.pitt.edu

ABSTRACT

In chip multiprocessors (CMPs), data access latency depends on the memory hierarchy organization, the on-chip interconnect (NoC), and the running workload. Reducing data access latency is vital to achieving performance improvements and scalability of threaded applications. Multithreaded applications generally exhibit sharing of data among the program threads, which generates coherence and data traffic on the NoC.

Many NoC designs exploit communication locality to reduce communication latency by configuring special fast paths on which communication is faster than the rest of the NoC. Communication patterns are directly affected by the cache organization. However, many cache organizations are designed in isolation of the underlying NoC or assume a simple NoC design, thus possibly missing optimization opportunities. In this work, we present a NoC-aware cache design that creates a symbiotic relationship between the NoC and cache to reduce data access latency, improve utilization of cache capacity, and improve overall system performance. Specifically, considering a NoC designed to exploit communication locality, we design a Unique Private caching scheme that promotes locality in communication patterns. In turn, the NoC exploits this locality to allow fast access to remote data, thus reducing the need for data replication and allowing better utilization of cache capacity. The Unique Private cache stores the data mostly used by a processor core in its locally accessible cache bank, while leveraging dedicated high speed circuits in the interconnect to provide remote cores with fast access to shared data. Simulations of a suite of scientific and commercial workloads show that our proposed design achieves a speedup of 14% and 16% on a 16-core and a 64-core CMP, respectively, over the state-of-the-art NoC-Cache co-designed system which also exploits communication locality in multithreaded applications.

Categories and Subject Descriptors

B.3.2 [Memory Structures]: Design Styles—Cache memories

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HIPEAC 2011 Heraklion, Crete, Greece
Copyright 2011 ACM 978-1-4503-0241-8/11/01 ...$10.00.

General Terms

Performance

Keywords

Network-on-chip, NoC, Cache, CMP

1. INTRODUCTION

Multicore processor performance is dependent on the data access latency, which is highly dependent on the design of the on-chip interconnect (NoC) and the organization of the memory caches. The cache organization affects the distance between where a data block is stored on chip and the core(s) accessing the data. The cache organization also affects the utilization of the cache capacity, which in turn affects the number of misses that require the costly off-chip accesses. As the number of cores in the system increases, the data access latency becomes an even greater bottleneck.

In the domain of network-on-chip (NoC), previous research has attempted to reduce communication latency by a variety of approaches. One approach is based on reducing the global hop count [4, 10, 16, 17, 26, 36]. Another approach provides fast delivery of high priority cache blocks [9]. A third approach configures fast paths or circuits which may be optical—through the NoC such that traffic traveling on these fast paths enjoys lower latency than regular traffic [3, 22, 28]. Communication locality is exploited in [1, 3, 22] such that the communication over a subset of source-destination node pairs is given higher priority than the rest of the traffic and explicit circuits connecting the source-destination pairs are configured to carry this higher priority traffic.

Static non-uniform cache architecture (S-NUCA) [25] and Private [11] caches represent the two ends of the cache organization spectrum. However, neither of them is a perfect solution for CMPs. S-NUCA caches have better utilization of cache capacity—given that only one copy of a data block is retained in the cache—but suffers from high data access latency since it interleaves data blocks across physically distributed cache banks, rarely associating the data with the core or cores that use it. Private caches allow fast access to on-chip data blocks but suffer from low cache capacity utilization due to data replication, thus resulting in many costly off-chip data accesses. Many researchers suggested hybrid cache organizations that attempt to keep the benefits of both S-NUCA and private caches while avoiding their shortcomings [6, 14, 15, 18, 20, 21, 37]. Most of these cache proposals assumed a simple 2-D packet-switched mesh interconnect. Such interconnects can be augmented with the ability to configure fast paths [1, 22, 28]. However, we do

∗We use the terms circuits and fast paths interchangeably
not expect all these cache organizations to equally benefit from an improved interconnect as the following motivating example shows.

In Fig. 1 we examine the performance of four systems running a set of parallel benchmarks on 16-core CMPs\(^2\). Two L2 cache organizations and two NoCs are used in constructing the 4 CMPs. The cache organizations are: (a) a private L2 cache and (b) A distributed shared L2 cache using an Origin 2000 based coherence protocol specifically designed to promote and exploit communication locality [22] (briefly described in Section 4.1). We refer to the earlier cache as Private and to the later as O2000P. The NoCs are: (a) CSP: A reconfigurable hybrid circuit/packet switched interconnect (a brief description can be found in Section 2) and (b) PKT: A regular packet switched interconnect. Because the CSP NoC exploits communication locality and O2000P is designed to promote this locality, the system with O2000P L2 cache benefits more from the CSP NoC since it achieves an average speedup of 10% over the same system with the PKT NoC. On the other hand, the system with the Private L2 cache - which does not promote communication locality and also suffers from capacity conflicts - does not show a significant change in performance with a CSP over a PKT NoC.

Therefore, in this work we present a NoC-aware cache design targeting the class of interconnects that exploit communication locality, which is a property both the cache and NoC affect and can exploit. We aim to create a symbiotic relationship between the cache and NoC that achieves our design goals, which are: (a) Improve data access latency and utilization of cache capacity and (b) Improve communication latency through decreasing NoC traffic volume and promoting communication locality. The remainder of this paper is organized as follows: Description of example NoCs from the targeted class of interconnects is presented in Section 2. Our proposed NoC-aware cache design is presented in Section 3. Evaluation and necessary background on the state-of-the-art NoC-Cache co-designed system we compare with is in Section 4. Section 5 describes related work, and finally, Section 6 concludes the paper.

2. BACKGROUND: NOCS THAT EXPLOIT COMMUNICATION LOCALITY

Our NoC-aware cache design targets a class of interconnects that improve communication latency through exploiting communication locality. We first briefly describe two such interconnects [1, 22], which are both reconfigurable hybrid circuit/packet switched NoCs and which we use in our evaluations.

Assuming a tiled CMP architecture, each tile has a network interface (NI). The NI is the interface point between the tile and the NoC. The NoC consists of one or more interconnect planes. The NI decides on which plane to send the next packet. Each plane is a hybrid circuit/packet-switched 2D mesh with a network router at each tile. On each plane, packet-switched (PS) and circuit-switched (CS) traffic are both intermingled on the same physical links and routers. In this context a circuit refers to a dedicated fast path that is established between the source and destination as explained below.

A packet may consist of one or more flits. In packet-switching, a flit goes through a 3-cycle router pipeline. In circuit-switching, packets are sent on pre-configured circuits (or fast paths) between source and destination tiles. A circuit from a source, \( S \), to a destination, \( D \), is configured by recording at each intermediate router the pair of input-output ports that should be connected by the router’s crossbar for the CS flit traversal. A CS flit takes one cycle to traverse a router. A lookahead signal is sent one cycle in advance of sending a CS flit so that the next router knows that there is an incoming CS flit and properly configures the crossbar. While a CS flit takes one cycle to traverse a router, a packet consisting of multiple flits cannot be delayed at any intermediate router. Therefore no conflicting circuits, i.e., circuits that use the same input or output port, can be simultaneously recorded at a router. Note, however, that routers have a timeout mechanism to prevent starvation of PS flits by temporarily disabling the circuit at the router until waiting PS packets are served. In this case, a CS packet becomes packet-switched at the first router where the circuit is disabled, and continues to be packet-switched until the packet reaches its destination. There is an additional virtual channel buffer at each of the router input ports, except the local port, to buffer a CS packet that becomes packet-switched at that input port (due to circuit reconfiguration while the CS packet is in flight, for example).

2.1 On-demand Circuit Configuration Policy

When a source, \( S \), wants to send a packet, \( p \), to a destination, \( D \), where there is already a circuit, \( C_{SD} \), configured from \( S \) to \( D \) on one of the NoC planes, then \( p \) is sent on that circuit. If there is no such circuit, \( S \) sends a circuit configuration request on a dedicated control interconnect plane. Configuring the requested circuit takes time because the configuration request is packet-switched. Thus, after sending the configuration request, \( S \) immediately sends \( p \) to \( D \) using regular packet-switching. \( S \) can use \( C_{SD} \) the next time it needs to send a packet to \( D \), assuming \( C_{SD} \) still exists. Due to limited resources, configuring a new circuit may tear down one or more other circuits. Routing on a partial circuit may occur only if the circuit is torn down while there is a packet in-flight on that circuit, in which case the packet becomes packet-switched for the remainder of the path.

2.2 Circuit Pinning Configuration Policy

A circuit pinning configuration policy is employed in [1] to exploit temporal communication locality. Temporal communication locality suggests that a processor will communicate most actively with a limited set of destinations and this set of destinations will be relatively small due to spatial locality [2, 27]. Time is divided into intervals, \( T_1, T_2, \ldots \) each of equal length, \( t_p \). During each interval \( T_i \), each tile, \( S \), gathers communication statistics on the number of packets it sends to every other tile, \( D \neq S \). At the beginning of the next interval, \( T_{i+1} \), the gathered statistics are used to configure circuits from each tile \( S \) to its most frequent destinations. The new circuit configuration is kept stable, i.e. pinned, for the duration of \( T_{i+1} \). A centralized and a distributed algorithm for configuring

---

\( ^2 \)Simulation parameters are described in Section 4.
the most important circuits based on the gathered statistics are presented in [1]. During the configuration time, only packet-switching can be used to send packets.

Stability of the circuit configuration is leveraged through partial circuit routing [1]. Specifically, a source $S$ may use partial circuit routing to send a packet, $p$, to a destination $D$ when there is no circuit from $S$ to $D$. $S$ may have a circuit, $C_{SE}$, to $E \neq D$ such that $p$ can be circuit-switched on part of $C_{SE}$ and then be packet switched until $D$.

3. UNIQUE PRIVATE: A NOC-AWARE CACHE

Parallelization and multithreaded programs harness the performance capabilities of CMPs. Our proposed Unique Private cache is designed to suit a workload consisting of a multithreaded program running on the CMP cores. As mentioned in the introduction, Unique Private is a NoC-aware cache organization targeting NoCs that exploit communication locality to reduce communication latency - however, our proposed design works correctly with any NoC. To reiterate, our design goals are: (a) Improve data access latency and utilization of cache capacity and (b) Improve communication latency through decreasing NoC traffic volume and promoting communication locality. These goals guide our design choices for the data placement and lookup, data migration, and even data replacement policies.

We assume a tiled CMP architecture with $n$ cores laid in a 2D mesh. Each tile has a processor core, a private level 1 (L1), and a level 2 (L2) cache banks. The Unique Private organization is proposed for the L2. We do not assume a third level of on-chip cache. We use the terms data block, cache block, and cache line, interchangeably.

We maximize the utilization of Unique Private’s cache capacity through keeping only a single - unique - copy of each cache block in L2. Controlled data replication has been shown to reduce data access latency [6, 14, 20, 31], particularly for read-only data, e.g., instructions. Support for replication can be added to our cache design. However, we do not study replication in this work. Similarly, data replication is not used in the last level cache of the state-of-the-art NoC-Cache co-designed system [22] which we compare with.

3.1 Data Placement and Lookup

Consolidating the working set of a thread in its locally accessible cache bank serves two important goals: (1) allows fast access by the thread to its working set, and (2) decreases the volume of traffic injected into the NoC due to increased hits in the local banks.

Further, prior research [5, 13, 32] showed that in parallel applications a thread often shares data with a small number of other threads. Hence, with the consolidation of the threads’ working sets, each thread (or equivalently core) would need to get most of its remote data from a small number of other cache banks, therefore creating locality of communication.

Often, a cache block is first accessed, or touched, by the core that is going to operate on that block. I.e., the first-touch accesses define most or all of the working set of each core. Thus, Unique Private employs first-touch as its data placement policy. Specifically, when a miss occurs in L2 for a data block, that block is brought from the off-chip memory and stored in the local L2 bank of the requesting core $P_i$. This policy allows any cache block to reside at any L2 bank (we refer to the L2 bank storing a cache block as the block’s host node). Consequently, there is a need for a method to lookup cache blocks in the L2. The Unique Private cache uses a distributed directory (i.e., there is a directory bank located at each tile of the CMP) for this purpose. For each cache block, there is exactly one directory bank, which we call the block’s home node, that keeps track of the block’s host node. The home node is determined based on the block’s physical address.

Example 1

We will use examples to explain how the directory is used during a data block access. First we need some terminology. Let $P_i$ denote the processor core located at tile $i, 1 \leq i \leq n$. Similarly, let $L_{1,i}, L_{2,i}$, and $D_i$ denote the L1 bank, L2 bank, and directory bank, located at tile $i$, respectively. Note that since $P_i, L_{1,i}, L_{2,i}$, and $D_i$, are all located at the same tile $i$, communication among them does not go over the NoC. Consider the example in Fig. 2(a). $P_i$ needs to read some data block $b_j$. $P_i$ first probes its local L1 bank, $L_{1,i}$, but misses, i.e., does not find $b_j$. $P_i$ next probes its local L2 bank, $L_{2,i}$, for $b_j$. Assume there is also a miss in $L_{2,i}$. The data read request is then sent to $b_j$’s home node, $D_i$. Assume $D_i$ does not have an entry for $b_j$. $D_i$ adds an entry for $b_j$ and records $L_{2,i}$ as the host node of $b_j$, and sends a reply to $P_i$ instructing it to retrieve $b_j$ from memory and store it locally in $L_{2,i}$. Note that the numbers in Fig. 2 are used to clarify the sequence of the example’s events.

Example 2

Consider the same example but assume $b_j$ already exists in some L2 bank, $L_{2,m}$ (Fig. 2(b)). Hence, $D_i$ already knows that $L_{2,m}$ is $b_j$’s host node. Upon receiving the data read request, $D_i$ forwards it to $L_{2,m}$. When $L_{2,m}$ receives the request, it sends a data reply
message to \( P_t \) containing a copy of \( b_j \), which will then be stored in \( L_1 \).

**Maintaining Data Coherence**

We may track the information necessary for maintaining coherence, i.e., each cache block’s status (e.g., shared, exclusive ...) and the L1 banks sharing it, in either the block’s *home node* or *host node*. Tracking this information in the *home node* requires that all data requests go through the directory to both update the requested blocks’ information and to properly order the requests before forwarding them to the blocks’ *host nodes*. We choose the other alternative, which is tracking the information in the block’s *host node*. This way, the *host node* orders and processes requests to the cache block similar to the way static non-uniform cache architectures (S-NUCA) [25] maintain data coherence. Consequently, the *home node* needs to only store the cache block’s tag and *host node*; effectively making the distributed directory act as a location service for finding cache blocks.

To reduce the number of lookup operations through the directory, we augment each L1 bank with a separate small cache for storing the ids of the remote hosts that previously sourced cache lines. This local cache of hosts is similar to the one proposed in [19] and we will refer to it as the *local directory cache* (LDC). Whenever \( P_t \) receives a data block, \( b_j \), from a remote host node, \( L_{2m}, m \neq i \), the LDC at tile \( i \) adds an entry containing \( b_j \)’s tag and the id of its *host node*, \( m \). The next time \( P_t \) needs to access \( b_j \) and misses in both its local L1 and L2 banks, the LDC is consulted and if an entry for \( b_j \) existed, the data access request is sent directly to the cached *host node*, \( L_{2m} \), instead of through the directory.

Note that due to data migration (explained below), the LDC may contain stale information – since it is only updated when a block is received from a remote host node. Thus, a request could be sent to a cached host node, \( L_{2m} \), that is no longer the host node of the cache block. This is remedied by having \( L_{2m} \) forward the request to the block’s *home node* as if the requester itself sent the request.

### 3.2 Data Migration

The *first-touch* data placement policy is based on the assumption that a cache block is first accessed by its owner thread, i.e., the block is part of the thread’s working set. However, this assumption is not always true, and data usage may change over time. For example, in a multithreaded program the main thread may first initialize all the data before spawning the other program threads. While the data is being initialized, it will be brought to the local L2 bank, \( L_{2j} \), of the core \( P_j \) on which the main thread is running. When the other threads are spawned, it would be beneficial to migrate the data blocks comprising the working set of each thread from \( L_{2j} \), to the corresponding locally accessible L2 bank of the core each thread runs on. Another example occurs in the producer-consumer and pipeline parallel programming models, where data may be passed from one thread to another. In such a case, it would also be beneficial to move the data to the local L2 bank accessible to the current thread manipulating the data. Thus, data migration is necessary for better data placement.

Prior research [7, 21, 23, 24] proposed and evaluated gradual migration policies and algorithms for near-optimal placement of cache blocks. A gradual block migration policy attempts to reduce a block’s access latency by gradually moving the block nearer to its frequent sharer(s). However, gradual data migration can possibly have negative effects such as: (1) Increased traffic volume due to the gradual movement of data blocks. (2) Decreased communication locality: frequent migrations may make it difficult for each tile to have an identifiable subset of other tiles with whom most or all of data sharing occurs. Additionally, sharers may already have configured circuits to where the block is located and may suffer from increased access time to the block if it is migrated. And (3) Reduced effectiveness of the *local directory caches*. Therefore, in addition to evaluating the gradual migration policy for the Unique Private cache, we propose and evaluate an alternate policy that migrates a block directly – instead of gradually - to its most frequent sharer.

**Direct Migration**

Specifically, the *direct migration* policy migrates a block, \( b_j \), to \( L_{2m} \), only if \( P_m \) accesses the block more frequently than other sharers. To determine to where \( b_j \) should be migrated, we would ideally augment the status of \( b_j \) in its *host node*, \( L_{2k}, \) with \( n \) counters, \( c_1, c_2, ... c_n \), where each counter \( c_k, 1 \leq k \leq n \), tracks the number of accesses of \( P_k \) to \( b_j \) in \( L_{2k} \). When a counter, \( c_m \), satisfies the condition \( c_m - c_i = th \), where \( th \) is a pre-specified migration threshold, and \( c_i \) is the counter for \( P_i \) (the local sharer), \( b_j \) is migrated to \( L_{2m} \) and a message is sent to \( b_j \)’s *home node* to notify it that \( L_{2m} \) is the new *host node* of \( b_j \).

Obviously, having \( n \) counters per cache block is a huge overhead and is not scalable. Hence, we propose a practical approximation of this migration scheme. We consider migrating a cache block if there is only one other sharer, \( P_m \), besides the local sharer, \( P_i \), otherwise migration of the cache block is not considered. This approximate scheme requires using only one counter, \( c \), per cache block. The migration mechanism works as follows: \( c \) is reset to 0 every time \( P \) accesses \( b_j \) in \( L_{2k} \), \( c \) is incremented by 1 whenever the only remote sharer, \( P_m \), accesses \( b_j \) in \( L_{2j} \). Migration of \( b_j \) to \( L_{2m} \) occurs when the condition \( c = th \) is satisfied, \( c \) can be implemented with a \( th \)-bit shift register. Our evaluation of the gradual and direct migration policies (Section 4) finds that the approximate direct migration scheme is the most appropriate one for our design.

**3.3 Data Replacement Policy**

When a cache block, \( b_j \), is brought from the off-chip memory to be stored in an L2 bank, \( L_{2j} \), an existing block \( b_j \) \( \in L_{2j} \) is chosen for replacement. We found that the *least recently used* (LRU) replacement policy is not always adequate for the Unique Private cache. We found it necessary to distinguish between shared and non-shared cache blocks (i.e., *private* blocks accessed only by the local core). Naturally, accesses to private blocks by the local core are faster than accesses of remote cores to shared blocks since the remote accesses have to go over the NoC. This difference in ac-

![Figure 3: Performance speedup relative to the system with O2000P cache + CSOD NoC](image-url)
cess latencies of private and shared blocks may result in biasing the LRU policy towards replacing shared blocks and retaining private blocks, especially in the case of poor initial placement of a shared block by the first-touch policy (i.e., if the local processor stops accessing the shared block).

Shared cache blocks are typically more “valuable” to retain in cache as they are accessed by more than one processor core. When a shared block, \( b_j \), is replaced and then later requested, the latency to service that miss could potentially affect more than one requester. Our intuition is supported by the work in [23], which showed that for the multitiered benchmarks they use, although the percentage of shared blocks to private blocks is small, shared blocks are accessed more than private blocks. Consequently, we propose to extend the LRU scheme to be biased towards replacing private cache blocks and retaining shared ones.

Specifically, we propose a Shared Biased LRU Policy (SBLRU) to select the cache line to evict from an associative set, \( S \). Depending on a parameter \( \alpha \), if the number of private cache lines, \( m \), within \( S \) satisfies \( m \geq \alpha \), then the LRU private cache line is selected for replacement. If \( m < \alpha \), then the LRU cache line, irrespective of being shared or private, is replaced. Note that SBLRU can be applied to any shared caching policy. Simulations (Section 4) show that SBLRU has a significant impact on the performance of the Unique Private cache.

4. EVALUATION

We first provide a brief background about the relevant state-of-the-art co-designed NoC-Cache scheme which we compare with. Then we describe our simulation environment, and finally present our simulation results.

4.1 Background: The Circuit-Switched Coherence co-designed scheme

Jerger [22] co-designed a NoC-Cache scheme that exploits communication locality in multithreaded applications to provide fast data access and improved system performance. They designed the hybrid circuit/packet switched NoC with the on-demand circuit configuration policy described in Section 2.1. We now describe their co-designed caching scheme. The on-chip cache is composed of three levels. The first two levels, i.e., L1 and L2, are private, while the third level, L3, is organized as a distributed shared cache. Data coherence is maintained through an adaptation of the Origin 2000 [29] protocol specifically co-designed with the NoC. A distributed directory is stored along-side the shared last level cache, L3. For each cache block, \( b_j \), L3 keeps track of the L2 banks that have copies of \( b_j \). The Origin 2000 protocol employs the request forwarding of the DASH protocol [30] for three party transactions, which target a cache block that is owned by another processor.

To promote communication locality on the NoC and reduce data access latency, the authors in [22], augment the base Origin 2000 [29] protocol with a scheme for predicting owners of requested cache blocks. A cache block can then be directly requested from the owner rather than experience an indirect through the home directory node. The prediction scheme is address-region-based; it assumes that if a tile \( D \) supplied a cache block \( b_j \), then \( D \) can probably supply other cache blocks with physical addresses close to the physical address of \( b_j \). Each tile is augmented with a local cache for predicting the owners of missed cache blocks. When a cache block, \( b_j \), is received from \( D \), an entry with the address of the memory region containing \( b_j \) and the id of \( D \) is cached in the local prediction cache. The prediction cache is checked on an L2 miss. If an entry for the memory region that the missed cache block belongs to is found, a request is sent to the L2 tile recorded in that entry. Otherwise, the request is sent to the cache block’s home directory bank. The distributed directory in L3 keeps the information for maintaining coherence of each cache block, including the sharer L2 banks. Thus, whenever a data request is sent directly to a predicted owner, the requester must also send a notification message to the cache block’s home directory bank. More details are provided in [22]. We call this cache organization Origin 2000 with Prediction, and refer to it as O2000P for short.

In our evaluation, we assume that the memory hierarchy on the simulated systems is composed of an on-chip level 1 and level 2 caches and an off-chip memory. Thus, to simulate O2000P, the private L1 and L2 of O2000P are lumped together in our assumed on-chip private level 1 cache, while the shared L3 of O2000P is represented by our assumed on-chip shared level 2 cache (note that there is no data replication in the level 2 cache).

4.2 Simulation Environment

We use Simics [33] for our simulation environment. We configured Simics to simulate a tiled CMP consisting of either 16 or 64 SPARC 2-way in-order processors, each clocked at 2 GHz, running the Solaris 10 operating system, and sharing a 4 GB main memory with 55 ns (110 cycles) access latency. The processors are laid out in a square mesh. Each processor has a 32 KB (divided equally between instruction and data) private 4-way L1 cache (access latency: 1 cycle). We simulate the following organizations of the L2 cache:

![Figure 4: Effect of Migration and LDC - Performance speedup relative to the system with O2000P cache + CSP NoC](image1)

![Figure 5: Comparing Migration Policies - Performance speedup relative to the system with O2000P cache + CSP NoC](image2)
Figure 6: Comparing Migration Policies - Percentage of the traffic volume used for migration messages

1 Regular Private cache which is only used in the motivating example of Fig. 1, (2) Origin 2000 with Prediction (O2000P) (Section 4.1), and (3) Unique Private cache (Section 3). We simulate 3 NoCs: (1) A purely packet switched NoC (used in the motivating example of Fig.1), (2) A hybrid packet/circuit switched NoC with an on-demand circuit configuration policy (Section 2.1), and (3) A hybrid packet/circuit switched NoC with a pinning circuit configuration policy and partial circuit routing (Section 2.2).

We use execution driven simulation, where we implemented cycle accurate simulators of the NoCs and cache schemes on top of Simics. We use benchmarks from the Splash-2 [35], Parsec [8], and SPECjbb2005 [34] suites. For the 16-core CMP, we simulate the parallel section of each benchmark. Benchmark input parameters are listed in Table 1. Due to the long simulation time on the 64-core CMP, we only simulate 400 Million instructions of each benchmark. We simulate the 64-core CMP to demonstrate scalability.

The Unique Private cache has an additional storage overhead due to its distributed directory, while O2000P does not since it is an S-NUCA based scheme where the directory and cache entries are located together and use the same tag. For a fair comparison, we account for that overhead by increasing the O2000P cache capacity. We use 64 byte cache blocks in the L1 and L2 cache banks. For 48-bit address space, we calculated the distributed directory’s overhead to be about $1/4th$ the size of the L2. We use 16-way directory banks (access latency: 2 cycles$^3$).

We use 16-way 1 MB banks for Private and Unique Private L2, and 20-way 1.25 MB banks for O2000P L2. L2 bank access latency is 6 cycles. O2000P uses a local prediction cache (we use regions of 512 bytes) and Unique Private uses a local directory cache (LDC). We set the number of entries of both of these local caches to be $1/2$ of the number of lines an L1 bank can cache. We calculated the size of each of these caches to be about $1/16th$ the size of an L1 cache bank. They are 4-way associative with 1 cycle access latency and are accessed in parallel with the L1 cache access.

For the simulated NoCs, we use parameters similar to those in [1]:

The packet switched NoC (PKT) is composed of one plane with a 64 byte link width. All control and data messages are one flit long. We simulate a 3-cycle router pipeline. Each router has 5 input and output ports. Each input port has 4 virtual channel buffers, with each buffer capable of storing 5 flits.

$^3$ We use CACTI [12] with 45 nm technology to estimate access latencies.

Hybrid packet/circuit switched NoC with an on-demand circuit configuration policy (CSOD) is composed of 4 planes, each with 16 byte links. Control and data packets are 1 and 4 flits long, respectively. The router is similar to that of the PKT NoC with the addition of: (1) Support for CS packets which traverse the router in one cycle and (2) one more virtual channel buffer per input port for buffering incoming CS packets if they become packet switched (due to circuit reconfiguration, for example).

Hybrid packet/circuit switched NoC with a pinning circuit configuration policy (CSP) is similar to CSOD but uses a circuit pinning configuration policy and partial circuit routing. The pinning time interval is $100\mu$sec while circuits configuration time is $8\mu$sec. During configuration time only packet switching is available.

All NoCs are clocked at 1 GHz and use X-Y routing. Private and O2000P use the LRU replacement policy. Unless specified otherwise, Unique Private uses the SBLRU replacement policy (Section 3.3) and the approximate direct migration policy (ADM) with a migration threshold of 3.

<table>
<thead>
<tr>
<th>Table 1: Benchmarks Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benchmark</strong></td>
</tr>
<tr>
<td>Parsec / Blacksholes</td>
</tr>
<tr>
<td>Parsec / Bodytrack</td>
</tr>
<tr>
<td>Parsec / Fluidanimate</td>
</tr>
<tr>
<td>Parsec / Swaptions</td>
</tr>
<tr>
<td>SPLASH-2 / Barnes</td>
</tr>
<tr>
<td>SPLASH-2 / LU</td>
</tr>
<tr>
<td>SPLASH-2 / Ocean</td>
</tr>
<tr>
<td>SPLASH-2 / Radiosity</td>
</tr>
<tr>
<td>SPLASH-2 / Raytrace</td>
</tr>
<tr>
<td>SPLASH-2 / Water Spatial</td>
</tr>
<tr>
<td>Specjbb</td>
</tr>
</tbody>
</table>

4.3 Simulation Results

Performance Comparison

We compare four 16-core CMP systems that promote and exploit communication locality in the on-chip cache and interconnect. We use the caches: Unique Private (UP) and O2000P, and the interconnects: CSOD and CSP. Fig. 3 shows the speedups of the 4 systems relative to the system with O2000P and CSOD. Using CSOD, the system with UP achieves a 15% speedup over the system with O2000P, on average. Using CSP, the system with UP achieves a 14% speedup over the system with O2000P, on average. Note that LU performs better with O2000P than with UP. This is due to the imbalance of the sizes of the threads’ working sets, in which O2000P’s uniform distribution of the memory space across the L2 banks allows better utilization of the aggregate cache capacity. Employing a cooperative scheme for managing cache capacity [14, 31] should improve the performance of workloads with imbalanced working sets.

Note that since CSP allows the systems to perform better than with a CSOD NoC, in the following evaluations we use CSP with all simulated systems.

Data Migration and the Local Directory Cache

We study the effect of migration and the local directory cache (LDC) on the performance of systems using the Unique Private cache. For this paper, we use the approximate direct migration policy (ADM) with $th = 3$ (we compare the gradual and direct migration policies later in this section). Fig. 4 shows the speedups of 5 systems rela-
similar to the baseline system which uses O2000P. This is mainly due to the big slowdown experienced by Raytrace and Swaptions. In fact, excluding Raytrace and Swaptions, the rest of the benchmarks show that the system with UP achieves an average speedup of 10.6% in this case. This speedup is due mainly to each thread having a large portion of its working set closely accessible in its local L2 bank.

**UP with LDC and No Migration (UP + LDC):** Second, we examine the effect of the LDC in the absence of migration. In this case, the system with UP achieves a 7.5% speedup, on average, over the baseline.

**UP with Migration and No LDC (UP + Migration):** Third, we examine the effect of migration in the absence of the local directory cache. The system with UP achieves an average speedup of 12.4% over the baseline.

**UP with Migration and LDC (UP + Migration + LDC):** Finally, we examine the effect of employing both the migration policy and the LDC. The system with UP achieves an average speedup of 14% over the baseline. From these results, we conclude that migration has a bigger effect than the local directory cache on the performance of the Unique Private cache, but using both of them allows an even better performance.

### Migration Policies

We study the effect of the gradual migration (GM), approximate direct migration (ADM), and exact direct migration (EDM) on the performance of Unique Private. Fig. 5 shows the speedup of 5 systems using UP relative to the baseline system which uses O2000P. Systems using UP and a GM policy with \( t_h = 5, 3 \) (denoted GM5 and GM3, respectively), achieve average speedups of 12% and 13%, respectively, over the baseline system. While systems using UP and an ADM policy with \( t_h = 5, 3 \) (denoted ADM5 and ADM3, respectively), both achieve a speedup of about 14%, on average, over the baseline system. The difference in performance between the GM and ADM policies is small; however, the two policies differ significantly with regard to the NoC traffic overhead. Fig. 6 shows the percentage of the NoC traffic that is used for migrating blocks. On average, systems applying GM5 and GM3 use 5.2% and 8.3%, respectively, of the NoC traffic for migrating blocks. On the other hand, the average percentage of NoC traffic for migrating blocks on the systems using ADM5 and ADM3 is 0.2% and 0.3%, respectively. In addition, direct migration requires less hardware resources: one counter per cache block versus 4 counters for gradual migration, for the 4 directions a block can be moved into. We conclude from these results that approximate direct migration better suits the Unique Private cache than gradual migration.

Further, we compare the performance of systems using the approximate and exact direct migration policies with \( t_h = 3 \) (denoted ADM3 and EDM3, respectively). The systems using ADM3 and EDM3 achieve an average speedup of 14% and 16.3%, respectively (Fig. 5). However, this difference in performance pales in comparison to the huge overhead of required hardware resources to implement the exact direct migration policy.

### Effect on the NoC

We study the effect of O2000P and UP on traffic volume, communication locality, and communication latency on the NoC.

**Traffic Volume:** Fig. 7 shows the traffic volume of 5 systems normalized to the system with O2000P. On average, systems with UP inject at least 40% less traffic than the baseline system. Specifically, UP with the GM policy injected 40% to 43% less traffic than the baseline, while UP with the ADM policy injected 50% less traffic than the baseline system. This is due to the first-touch data placement policy which allows many misses from the private L1 to be satisfied by the local L2 bank.

**Communication Locality:** We collected statistics on the average number of most important circuits that get created at the beginning of each pinning interval in the CSP NoC used in our simulated systems. At the beginning of the next pinning time interval, the circuits on the NoC are re-configured, and each NI at each tile has the list of the other 15 destinations in descending order of importance. Since in our design CSP consists of 4 interconnect planes, at most 4 circuits can originate from each tile. Therefore, when the circuits are re-configured each tile should ideally get circuits configured to its top 4 most important destinations. However, this is rarely possible since there must be no conflicts between configured circuits. Hence, some of the circuits that get configured are to less important destinations, but they all satisfy the no conflicts condition.
Figure 9: Performance speedup of systems using SBLRU relative to corresponding systems using LRU

We are interested in the number of the most important circuits that get created (there can be a maximum of $4 \times 16 = 64$ circuits) at the beginning of each network reconfiguration period (Fig. 8). On average, 31.8 and 36.8 most important circuits are configured for the system with O2000P and the two systems with UP, respectively. This 15.6% increase is due to the communication locality achieved by UP's data placement and migration policies.

Communication Latency: Communication should become faster with increased communication locality. Indeed, we found that systems with UP enjoyed a 11% reduction in flit latency, on average, compared to the system with O2000P.

SBLRU policy
We compare the speedup of a system with UP and SBLRU (Section 3.3) to the same system using the traditional LRU policy. We perform the same comparison for O2000P too. For the UP cache, six of the eleven benchmarks show more than 11% speedup with the SBLRU policy (Fig. 9). On average, the system with UP and SBLRU achieves a 19.6% speedup over the same system with the regular LRU policy. As explained in Section 3.3, SBLRU offsets the bias of the traditional LRU policy towards replacing poorly placed shared blocks. For the systems with O2000P, it does not matter whether LRU or SBLRU is applied; we get almost the same performance with both policies.

Scalability
We compare the performance of UP and O2000P on 64-core systems (Fig. 10). Again, the imbalance of the working sets of LU's threads caused it to show a 20% slowdown with UP. However, the system with UP achieves an average speedup of 16%, which demonstrates that larger systems would benefit from the Unique Private cache.

Applicability to purely packet-switched NoC
We simulated both O2000P and UP with a regular packet switching NoC (on a 16-core CMP). Since the placement and migration policies help UP reduce the injected traffic, UP showed an average performance speedup of 16.6% over O2000P.

5. RELATED WORK
Besides the NoC-Cache co-designed system [22] we compare with, other related work includes [7] which showed that performance can benefit from gradual block migration. D-NUCA cache for CMPs [21] allows dynamic mapping and gradual migration of blocks to cache banks but requires a search mechanism to find cache blocks. Kandemir [23] proposed a migration algorithm for near-optimal placement of cache blocks but requires the use of some of the cache lines for storing information necessary for making migration decisions. CMP-NuRapid [15] employs dynamic placement and replication of cache blocks. Locating cache blocks is done through per processor tag arrays which store pointers to the locations of the blocks accessed by each processor. CMP-NuRapid suffers from the storage requirements of the tag arrays and the use of a snooping bus for maintaining these arrays, which may not scale well with many cores on the chip.

R-NUCA [20] relies on the operating system (OS) to classify data pages as private or shared. The first access to a page classifies it as private and is therefore mapped to the local cache bank of the accessing core. A subsequent access to the same page that originates from another core re-classifies the page permanently as shared. The cache blocks of a shared page are mapped in the cache using the standard address interleaving and the rotational interleaving indexing schemes [20]. For multithreaded programs that initialize data in the main thread, all pages would be re-classified as shared once other threads start operating on them. In this case, the data placement of the R-NUCA becomes similar to that of the S-NUCA.

6. CONCLUSION
This work is based on the recognition that CMPs with different cache designs may not benefit equally from an improved interconnect design. We present a NoC-aware cache design, Unique Private, specifically targeting NoCs that exploit communication locality to optimize the NoC performance. Our goal is to create a positive interaction between the cache and NoC that results in reducing the traffic volume and promoting communication locality on the interconnect, consequently allowing the processing cores to enjoy faster on-chip communication and faster data access. We show how these requirements affect our design choices for the cache data placement and migration policies. Additionally, we show how our chosen data placement policy could negatively affect the traditional least-recently-used replacement policy and suggest the shared-biased-least-recently-used policy to mitigate such a negative effect. Finally, using simulation, we study the effects of the different design choices and demonstrate the merits and scalability of our NoC-aware cache design.

7. REFERENCES
pinning in NoC. In Proc. of Hot Interconnects (HOTI), 2009.


[34] "Simics". http://www.virtutech.com/.


