Nahalal: Memory Organization for Chip Multiprocessors

Zvika Guz, Idit Keidar, Avinoam Kolodny, Uri C. Weiser

***Abstract*--This paper addresses cache organization in Chip Multiprocessor (CMPs). We introduce Nahalal, a novel *non-uniform cache* (NUCA) topology that enables fast access to shared data for all processors, while preserving the vicinity of private data to each processor. Our characterization of memory accesses patterns in typical parallel programs shows that such a topology is appropriate for common multi-processor applications. Detailed simulations in Simics demonstrate that Nahalal decreases the shared cache access latency by up to 54% compared to traditional CMP designs, yielding performance gains of up to 16.3% in run time.**

***Index Terms*—cache memories, chip-multiprocessors, shared memory systems**

I. INTRODUCTION

Emerging and future computer architectures are designed as Chip Multi-Processors (CMPs), which leverage the parallelism of multi-threaded applications to achieve higher performance within a given power envelope. Data access is typically a bottleneck in such systems, as multiple threads compete for limited on-die memory resources. Hence, the organization and management of on-chip cache memory become critical to system performance.

Two major factors impact the latency of on-chip memory access: wire delays and contention over shared memory. Global wire delays are becoming a dominant factor in VLSI design [1][2][3], and on-chip cache access time increasingly depends on the distance between the processor and the data. Concurrent access by multiple processors to a shared cache further increases the access time, as additional delays are incurred for resolving contention on the cache. Some communication fabric such as a Network-on-Chip [4] is used for interconnecting the

Zvika Guz, Idit Keidar and Avinoam Kolodny are with the Electrical Engineering Department, Technion, Israel Institute of Technology, Haifa, Israel. (emails: zguz@tx.technion.ac.il, idish@ee.technion.ac.il, kolodny@ee.technion.ac.il).

Uri C. Weiser is with Intel Corporation, Petach Tikva, Israel. (email: uri.weiser@intel.com).

processors to the on-chip cache.

Assuming the worst-case distance and latency for every memory access is wasteful. Hence, CMPs are shifting towards a *Non Uniform Cache Architecture* (NUCA) [5], where the cache is divided into multiple banks, and accesses to closer banks result in shorter access times. In NUCA, performance depends on the average (rather than worst-case) latency. The division of the cache into multiple banks also allows multiple processors to access different banks simultaneously, thus reducing contention.

Using NUCA, the vicinity of reference becomes of critical importance, and hence data should ideally reside close to processors that access it. Unfortunately, there is no straightforward way to do so, since, as we show in Section 3, a substantial fraction of the memory accesses involves blocks that are shared by many processors. Furthermore, due to coherence mechanisms, writing to shared blocks usually requires communication with all sharers, which increases the importance of expediting such accesses. Thus, a major challenge to address in the design and management of CMP caches is locating shared blocks in the vicinity of all processors that share them, while eliminating contention on banks containing blocks that are not shared.

Luckily, as we further show in Section 3, although shared blocks consume a substantial fraction of the total accesses to memory, these blocks comprise only a small fraction of the total working set. Moreover, most of the shared data is shared by all processors. We dub this phenomenon the *shared hot-blocks effect*.

We leverage the above observations in order to design a CMP topology that tackles both the growing wire delay and the shared data problems. In Section 4, we present our novel cache topology, Nahalal. This topology was inspired by the layout of the cooperative settlement Nahalal, shown in Figure 1(a), which is based on the *concentric circles* urban design from the

18th century [6]. Public buildings (school, administrative offices, warehouses, etc.) are located in the core, and are easily accessible to all. They are enclosed by a circle of homesteads. Beyond them, reside private tracts of land, each in proximity to its owner's house.

(a) Aerial view of Nahalal [7]. **(b) Schema of Nahalal CMP design.**

Figure 1. Nahalal design. The inner-most memory circle is designated for shared memory.

We project the same conceptual layout to our CMP, as schematically illustrated in Figure 1(b). (A detailed layout is given in Section 4). Generally speaking, in Nahalal, a small fraction of the memory is located in the center of the chip, enclosed by all processors, while the rest of the memory is placed on the outer ring. The inner memory is populated by the hottest shared data, and allows for fast access by all processors. The outer rings create a "private yard" for each processor in the periphery of the chip, improving vicinity of reference and reducing contention.

The Nahalal topology provides a platform for implementing the *Dynamic NUCA* (DNUCA) cache management approach [8], whereby data blocks can *migrate* among cache locations. Whereas DNUCA has been shown to achieve good performance in single-processor architectures [8] it does not improve performance in traditional CMP designs, because of the hot-blocks effect [5][9]. By allowing hot-blocks to reside in proximity to all processors, Nahalal provides a

platform where DNUCA can realize its potential, and achieve good performance in a CMP. In Section 5, we demonstrate the advantages of our approach via a full system simulation in Simics [10]. We compare the Nahalal topology to the classical CMP NUCA floorplan [5] of a shared L2 cache located in the center of the chip, surrounded by several processors. We show that Nahalal outperforms the classical architecture over a range of applications, including some of the most common parallel workloads. Nahalal achieves the most significant improvements, up to 16.3% in runtime, in commercial benchmarks like apache.

II. Related Work

The concept of Non Uniform Cache Architecture (NUCA), was introduced by Kim *et al.* [8] in the context of a uniprocessor system. To further reduce the average access time, the authors of [8] have suggested the use of dynamic block migration, called Dynamic NUCA (DNUCA). In DNUCA, every access to a block moves the block one step closer to the processor, thus gradually reducing distances and access times to popular data.

Beckmann *et al.* [5] applied NUCA to CMP systems, devising an eight node CMP structure where L2 banks are located in the center of the chip and are shared by eight symmetric processors that surround it (see Figure 2). We henceforth refer to this layout as *cache-in-the-middle* (CIM).

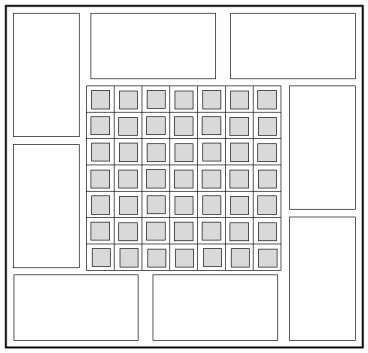


Figure 2. Traditional cache-in-the-middle CMP NUCA layout. A banked L2 is located in the center of the chip, surrounded by eight symmetric processors.

Previous studies of CIM-CMP [5][9] have compared DNUCA to a *Static NUCA* (SNUCA) design, where block placement is static, determined by address. Their findings show that the remoteness of shared data is a major drawback of the DNUCA approach: Whereas private data blocks, which migrate towards a single processor, enjoy fast access times, shared data blocks gradually migrate to the center of the chip, where they are essentially far from all processors. (Figure 3 illustrates this effect, presenting shared data blocks in dark colors). Due to coherence mechanisms, access to theses shared blocks usually requires communication with all sharers, further reducing the latency. Both studies have found that, as a result, DNUCA can achieve at most a modest performance improvement over the SNUCA scheme, and may even exhibit performance degradation when the percentage of accesses to shared blocks is high. In fact, both works have concluded that, in the CIM CMP design, DNUCA does not provide a sufficient performance improvement to justify its hardware overhead as compared to SNUCA. Our alternative layout resolves the major problem of remoteness of shared data, thus allowing one to achieve the performance advantages of DNUCA in CMPs.

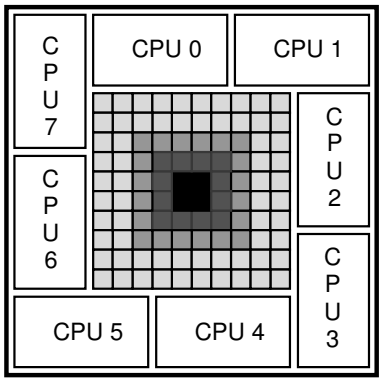


Figure 3. Block locations in CMP-NUCA. Blocks shared by many processors (depicted in dark colors) reside in the middle, while private blocks (depicted in light colors) reside close to their owners.

Several works have tried to ease the hot blocks problem using block replication, whereby every copy migrates towards one of the block sharers [11][12][13]. Such replication, however, reduces

the effective cache capacity, further increasing the on-chip capacity pressure. Moreover, block replication is only cost-effective when the shared blocks are read-only, since writing entails invalidation of all copies, which only intensifies the problem. Our work eliminates the need for replication by allowing a single copy to reside close to all the processors that share it.

Beckman *et al.* [5] [12] have studied memory access patterns in CMP, identifying the imbalance between the number of accesses to shared blocks and the actual number of shared blocks in the working set, and pointed out the importance of shared blocks to overall memory performance. The results we give in Section 3 continue this line of work, identifying the same trends and making additional observations.

III. Memory Accesses Characterization

In this section, we analyze the sharing patterns that occur in various standard multi-threaded programs, including SPEComp benchmarks [14], Splash2 benchmarks [15], an apache web server [16] and zeus web server [17]. More details about the benchmarks are given in Section 5.1.2 below. We use the Pin program analysis tool [18] to profile accesses to each cache block throughout the program execution, emulating a scenario where the program is run on an eight-way processor (a similar approach was used for studying L2 accesses in [12]). The results are summarized in Table 1.

We first examine, in the third column of Table 1, which percentage of the memory accesses are made to shared blocks, namely blocks accessed by multiple processors. The remaining accesses are to private blocks, i.e., blocks accessed by only one processor. We find: **(Observation 1) access to shared blocks comprises a substantial fraction of the total memory access**, ranging from 6.99% in ocean, through 32.05% in equake, and up to a 58.25% in apache. We conclude

that the performance of such accesses can have a significant impact on CMP performance.

Benchmarks		Shared blocks		Average Number of sharers		Read-write sharing	
		% accesses	% blocks	% accesses	% blocks	% accesses (out of total accesses)	% blocks
SPEComp	equake	32.05	0.73	7.72	2.02	2.78	0.40
	fma3d	8.93	0.16	7.99	2.03	0.37	0.14
Splash2	barnes	15.36	7.07	6.53	3.21	3.14	0.61
	ccean	6.99	2.48	5.68	2.06	3.01	2.30
	water	24.90	11.96	5.90	4.98	17.55	10.85
	lu	32.72	15.32	3.43	2.58	10.00	5.19
	radix	14.47	4.96	7.09	2.12	1.50	0.83
commercial	apache	58.25	34.33	3.69	2.95	47.91	25.26
	zeus	56.85	37.76	5.68	3.59	41.64	28.16

Table 1. Block sharing characteristics.

In the fourth column of Table 1, we examine the percentage of the data blocks that account for these accesses. We observe: **(Observation 2) a relatively small number of shared blocks make for a substantial fraction of the total accesses to memory**. For example, in equake, only 0.73% of the blocks in the working set are shared, but these blocks consume 32.05% of the total accesses to memory. In the apache benchmark, shared blocks comprise only 34.33% of the working set, but they account for 58.25% of the accesses to memory.

The fifth and sixth columns in Table 1 present the average number of processors that share a block per shared blocks and per access respectively. These two columns show that: **(Observation 3) shared blocks are typically shared by many processors.** In the SPEComp benchmarks, a vast majority of the accesses to shared blocks are to blocks shared among all processors. In zeus, the average number of sharers per access to a shared block is 5.68 (out of 8). These three observations characterize a phenomenon that we call the *shared hot blocks effect*, where a small subset of the working set is accessed numerous times by all or many processors.

Finally, the last two columns of Table 1 reveal: **(Observation 4) in commercial workloads,**

accesses to read-write shared blocks are a significant fraction of the total accesses to shared data.

While Table 1 presents the averages over the entire program execution, in Figure 4, we study short-term sharing. To this end, we consider a block to be shared only if it is accessed by several processors within a window of a given number of instructions. Figure 4 plots the percentage of access to shared blocks as the size of the windows increases. As can be seen from the figure, for most of the benchmarks, the percentage of accesses to shared block is noteworthy even when considering a narrow window of few million instructions. This implies that processors access the shared blocks concurrently. We have also explicitly checked which blocks are shared in each slice, and found that blocks rarely change their attribution to private or shared at different time slices. We conclude: **(Observation 5) shared blocks remain shared throughout the program execution.**

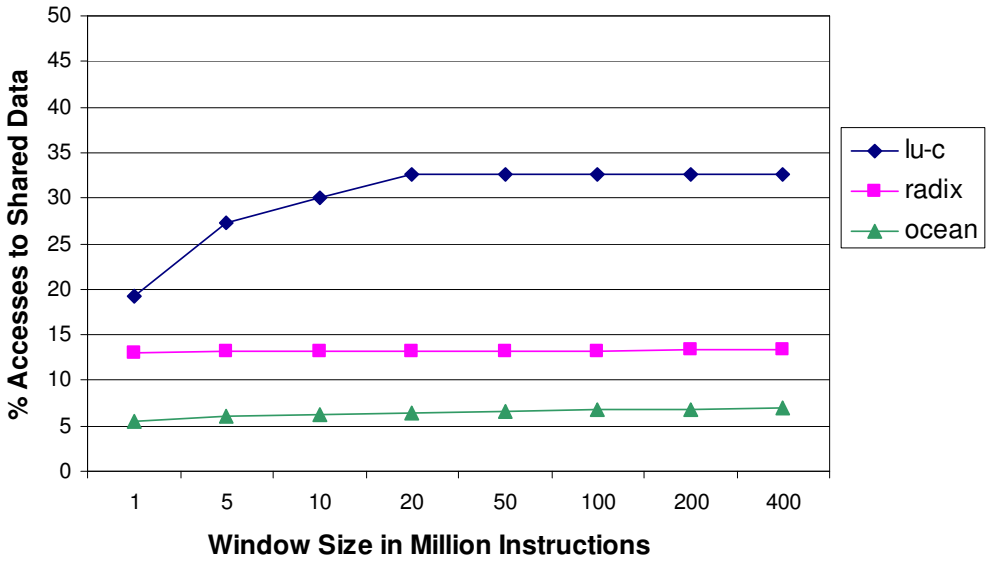


Figure 4. Percentage of accesses to shared data blocks, where a block is considered shared when several processors access it within a window of a given number of instructions. The x axis is the size of the windows in million instructions. The program exhibits stationary behavior over time.

To better understand the hot-block effect we focus on shared data in Figure 5, plotting the

distribution of the memory accesses to shared blocks. Figure 6 shows the distribution of memory accesses to read-write shared blocks in commercial workloads, where the portion of such blocks is significant. The results exhibit locality even within the set of hot blocks, i.e., a small fraction of the shared blocks - some very hot blocks – account for most of the accesses to shared data. We conclude: **(Observation 6) some hot blocks are more popular than others.** This observation implies that one can improve performance by allowing fast access to the "hottest" hot blocks, even if fast access to all hot blocks is not possible.

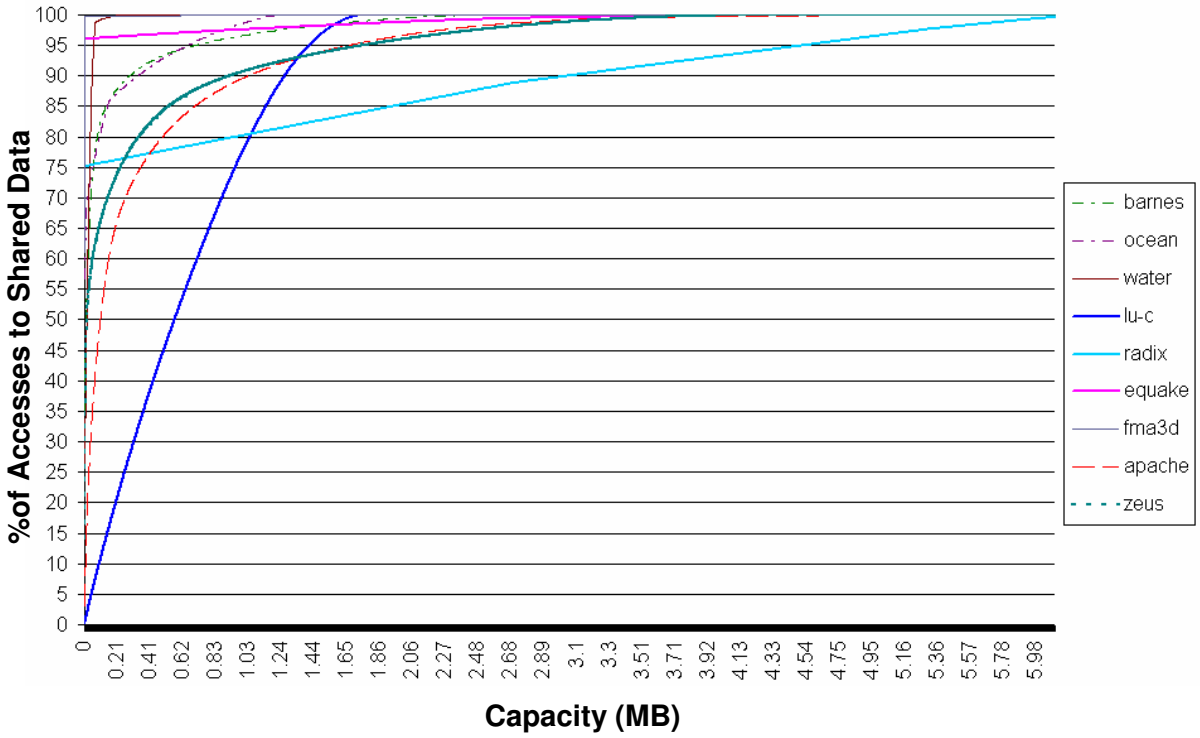


Figure 5. Accesses distribution of shared blocks. A small subset of the memory capacity suffices for holding blocks to which the majority of accesses are made.

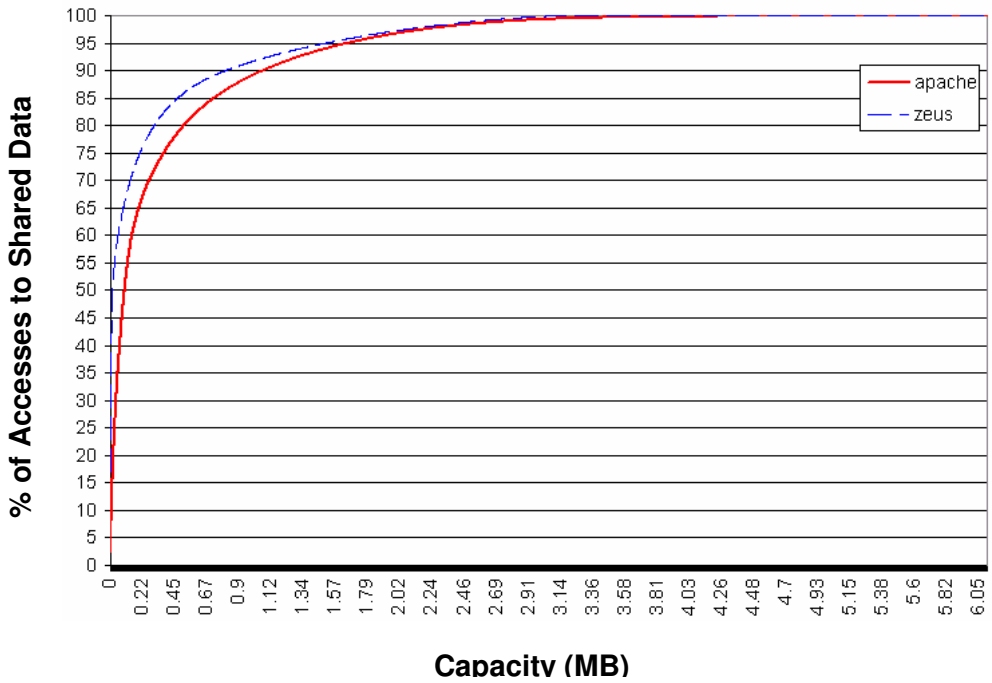


Figure 6. Access distribution of shared read-write blocks for the commercial workloads. A small subset of the memory capacity suffices for holding read-write blocks to which the majority of accesses are made.

IV. Cache Organization And Management

As mentioned in Section 2, NUCA-based CMPs had difficulties harnessing block migration to improve performance. The shared hot block phenomenon observed in Section 3 proved to be the Achilles' heel of performance in all such designs, since it causes shared hot blocks to migrate to the center of the cache array, far from all processors. While remoteness of shared data could have been acceptable if only a handful of all memory accesses would have involved such cache blocks, the many accesses to shared data (Observation 1) penalize the average memory access time and hinder memory performance. The fact that a substantial fraction of the accesses to shared data are to read-write shared blocks (Observation 4) further emphasizes the problematic character of such remoteness, since these accesses are the worst-case in terms of cache access time (due to coherence messages sent to all sharers).

Moreover, the fact that shared blocks are shared by many processors (Observation 3) implies that, in a traditional CIM-NUCA organization, regardless of the chosen migration policy, these blocks will inevitably reside far from at least some of the sharers. Hence, this problem is an inherent weakness of the classical CIM layout.

At the same time, Observations 2 and 5 suggest that the problem is restricted to a small number of shared blocks, which do not change over the program's time. Moreover, Observation 6 suggests that the hottest shared blocks, which account for most accesses, are even fewer. All of these observations motivate our novel architecture, Nahalal, which allows the few shared blocks (or the hottest thereof) to reside in proximity to all processors.

We present Nahalal's organization in Section 4.1. In Section 4.2 we discuss cache management in Nahalal, concentrating on block placement and migration.

4.1. The NAHALAL Layout

Our suggested CMP topology tackles the remoteness of shared data blocks by breaking the traditional cache-in-the-middle floorplan. As explained above, the main concept in Nahalal is placing shared data in a small area in the middle of the chip, surrounded by processors, and locating private data in the periphery. We now present one example of a detailed embodiment of the Nahalal concept in a setting similar to that of [5]. We consider a collection of eight processors, each of which has a private L1 cache, and roughly half the chip consists of L2 memory banks (this is typical for today's processors, where the relative area occupied by on-chip memory is increasing as VLSI generations progress).

The processors and banks are interconnected by a Network-on-Chip (NoC) [19][20]. NoCs are already a reality in MPSoCs, and are expected to dominate future CMP designs [4]. NoCs scale better than traditional bus and wiring solutions [21]; they increase spatial reuse while reducing

contention on the interconnect, and enabling concurrent memory transactions.

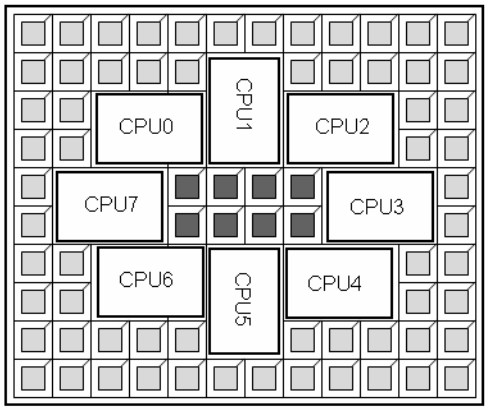


Figure 7. Example Nahalal layout. An 8-way CMP with a shared, banked L2 cache interconnected via NoC. The eight inner-most banks are designated for shared blocks while the rest of the banks (depicted in light gray) are used to store private data for nearby processors.

Whereas Figure 1(b) presented Nahalal's schematic layout, Figure 7 portrays a realistic layout for an 8-way CMP design. A small fraction of the total memory, designated for shared data is located in the center, enclosed by the processors. The rest of the cache is located in the peripheral rings, and is used mainly for private data. Unlike in the traditional CMP-DNUCA layout, here private data blocks do not obstruct or delay the path to the shared data blocks. Thus, the Nahalal layout enables fast access by all processors to shared data while still preserving the proximity of private data blocks.

This solution is feasible thanks to the shared hot blocks phenomenon (discussed in Section 3), which suggests that the shared area can be accommodated in a relatively small area in the center of the chip. This small shared cache area can be further reinforced to better handle multiple accesses and increased pressure without paying a severe area and power overhead (as would have been required for allowing concurrent accesses to all data banks). That is, this design allows one to improve performance by selectively investing additional resources in a small subset of the total cache – the same subset that is expected to experience the higher load.

4.2. NAHALAL Cache Management

The management of CMP-DNUCA based caches has been studied in several pervious papers [5][9][22]. Such mechanisms are, by-and-large, agnostic to the cache layout topology, and can be used in Nahalal. We therefore employ previously-suggested cache management mechanisms [5][9] in our system. We now briefly discuss the relevant details for the sake of completeness, while elaborating on issues that are unique to the Nahalal design.

We use the MESI coherence protocol [23] and enforce inclusion between the shared L2 and all L1 caches (as in [5][9]). Each L2 line's tag includes a *sharing status vector* comprised of a bitmask indicating which L1 has a valid copy and a *dirty bit* that indicates if one of the higher level caches has a modified version. When a processor modifies an L1 copy, an update message is sent to the L2 cache, which in turn sends invalidation messages to all the other L1 caches that hold the block (a similar approach was used in [9][24]).

Several block migration schemes for CMPs have been suggested [5][9]. The basic idea in all schemes is to partition the physical memory banks into groups named *bankset*s. Each block is mapped to one bankset via its address, and can reside in any bank that belongs to that bankset. The banks in each bankset are distributed in the chip such that they reside in varying distances from each processor. When a processor accesses a block, the block can migrate to another bank that belongs to the same bankset and resides closer to the processor. Migration is done gradually, so that blocks switch places in a bubble-sort like fashion.

We employ the same basic idea, except in the decision where to migrate blocks to. In Nahalal, only shared blocks migrate to banks located in the center of the chip, whereas private blocks migrate towards the processor within the processors' designated private area. We leverage the *sharing status vector* to identify shared blocks: During a block access, if more than one bit is set

in the sharing status vector bitmask, then the block is identified as shared and is migrated to the center.

Finally, a mechanism for locating blocks is needed. Searching all banks in parallel is prohibitively costly, since it overloads the interconnect, as well as the banks. Instead, some mechanism providing *hints* as to where to look for a given block is desirable. Recently, Ricci *et al.* [22] have leveraged bloom filters to devise such a complexity-effective search mechanism for CMPs. Since this mechanism is orthogonal to the concepts introduced in this paper, for simplicity, we use an oracle predictor that knows the location of all blocks in our simulations (this was also done in previous studies of CMP DNUCA performance [5]). Due to the effectiveness of the predictor, this is not expected to significantly offset performance measurements.

V. Performance Evaluation

In this section we evaluate the performance of the Nahalal architecture using a detailed system simulation in Simics [10]. Section 5.1 describes the simulation methodology, environment, and benchmarks. Simulation results are presented in Section 5.2

5.1. Methodology

5.1.1. Simulation Setup

We compare three solutions: Nahalal, a classical cache-in-the-middle DNUCA (CIM-DNUCA), and a cache-in-the-middle SNUCA (CIM-SNUCA). Our evaluation uses a full-system simulation in Simics [10], using the x86 in-order processor model as a building block. The architectural parameters used in most simulations are summarized in Table 2. Our parameter choices and cache-in-the-middle model closely follow those employed in [5]. We implement the

8-processor CMP design of Figure 7, assuming 45nm technology [25]. Each processor has a private 64KB L1 data cache. Instruction caching is not implemented, but rather instructions are assumed to be available with no delay. The processors share a 16MB NUCA L2 cache comprised of 256 banks. All system components are interconnected via a Network-On-Chip (NoC) [4][19][20]. Each link's bandwidth is assumed to be infinite, and no contention is modeled at the routers. Thus, the delay incurred by the network is proportional to the number of hops in the network between the source and the destination. Each hop is assumed to be a single link corresponding to the distance between adjacent cache banks.

Parameter	Value
Processor clock frequency	10GHz
Number of processors	8
L1/L2 cache block size	64B
Private L1 caches	16KB, 2-way, 3 cycle
Shared L2 cache	16MB, 256x64KB banks, 16-way
Cache bank latency (T_b)	6 cycles
Network per-hop delay (δ)	2 cycles
Main memory	4GB, 260 cycles

Table 2. System Parameters.

A processor's *cache access time* is comprised of the target bank's *cache latency*, the two-way network delay, and possibly a queuing delay resulting from contention on busy banks, since at most one access per bank is allowed at a given time. In accesses to read-write shared data, the MESI coherence protocol incurs additional delays: A read access to a read-write shared block may require fetching the block from another processor that changed it. A write access to a read-write shared block has to wait until the copies of all other shares are invalidated. Since we use an *oracle predictor*, we neglect the lookup time, and assume the accessed block's location is known.

In our simulation of Nahalal, 2MB (32 banks) of the 16MB cache are allocated in the center and are used for shared data. When these are full, less popular shared data can reside elsewhere.

Empirically, we found that 2MB suffice to hold most of the shared data in standard benchmarks. The remaining cache banks are placed in the outer ring.

Several block migration schemes were proposed in the literature [5][9]. Since block locations in the Nahalal and CIM topologies differ, the same migration scheme cannot be used in both. Therefore, in order to allow for a fair comparison of the two topologies, we run our simulations using an *oracle* placement approach, which approximates the final location of blocks in virtually all previously suggested migration schemes. The oracle places the most accessed shared blocks in the center banks, and the most accessed private blocks in the closest banks to each processor. Thus, the oracle eliminates the warm-up period, during which blocks migrates in bubble-sort fashion until reaching their equilibrium placement with respect to the pulling forces set by its sharers.

5.1.2. Benchmarks

In our simulated system, we run Mandrake 10.1 Linux with SMP kernel version 2.6, custom-compiled to interface with Simics, on top of which we run Linux programs as benchmarks. We have studied various commercial and scientific benchmarks. Our benchmarks include five SPLASH-2 benchmarks [15]: barnes, ocean water, lu and radix, two SPEComp benchmarks [14]: equake, and fma3d and two static web content serving: apache HTTP server and zeus web server. For both web benchmarks, we use the SURGE [16] toolkit to generate a workload of web requests from a 30,000-file, 700MB repository with a zero backoff time. This toolkit generates a Zipf distribution of file accesses, which was shown to be typical of real-world web workloads.

Typical parallel benchmarks begin with some setup code, which is run serially, and only then embark on parallel processing. In order to simulate the parallel part of each benchmark, which is the most interesting for our purposes, we fast-forward through the serial part, and then perform

measurements in a parallel part of the code. In all benchmarks other than those of Splash, the parallel part itself consists of loops that are iterated-through hundreds or thousands of times (in the SPEComp benchmarks), or even millions of times (in case of apache and zeus). All iterations have very similar characteristics. We therefore run only some of the iterations (hundreds in apache and zeus).

5.2. Results

We now present performance results. We begin, in Section 5.2.1, by examining the primary phenomenon impacted by Nahalal's layout, namely the distance between processors and their data. Having shown that Nahalal improves proximity of data to its owners, we continue to examine the impact this has on the average cache access time in Section 5.2.2. Finally, in Section 5.2.3, we study the benchmarks' overall running time, thus showing how the improved cache access time impacts the bottom line performance. We also study the trend exhibited as wire delays become more dominant.

5.2.1. Hop Count

Figure 8 depicts the average distance from processors to the data they access, measured in number of network hops. The average is computed over all cache accesses in each benchmark.

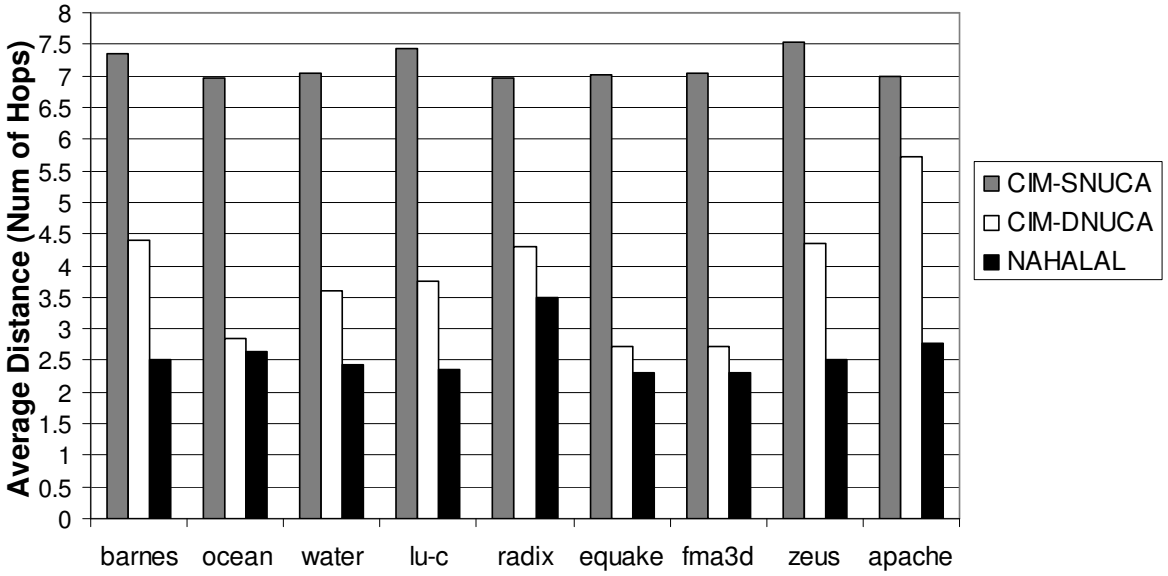


Figure 8. Average distance in number of hops between the processor and the cache block they access. CIM-DNUCA reduces the average distance over CIM-SNUCA, and Nahalal further reduces the distance, especially in benchmarks with many accesses to shared blocks.

We see that Nahalal achieves the best results, and exhibits significant improvements over CIM-SNUCA in all benchmarks. CIM-DNUCA is close to Nahalal in some benchmarks, but closer to CIM-SNUCA in others. This is explained by the discrepancy between to the distances to shared versus private blocks in CIM-DNUCA, as shown in Figure 9. Consequently, while in benchmarks like ocean (only 6.99% shared accesses) CIM-DNUCA does well, in benchmarks like apache, which have a high percentage of accesses to shared data, CIM-DNUCA presents only a modest improvement over CIM-SNUCA. This is a result of the long distance to shared blocks, which always end up in the center of the chip. The Nahalal design, on the other hand, reduces the average distance over the entire application spectrum, since it places both shared and private blocks close to all processors that access them.

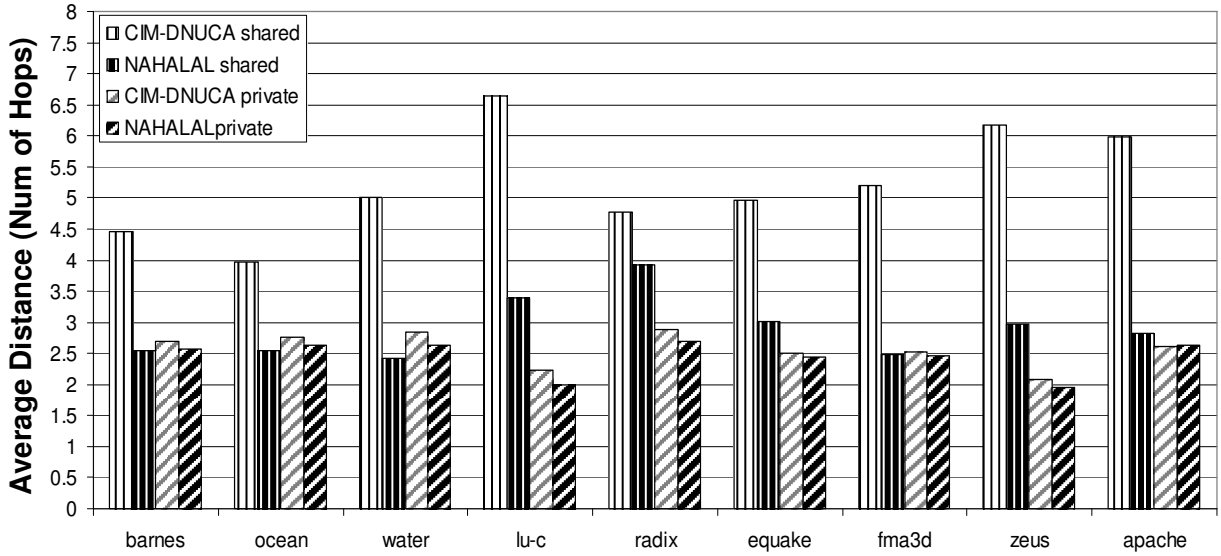


Figure 9. Average distance to shared versus private blocks for CIM-DNUCA and Nahalal. Nahalal reduces the distance to shared blocks, while the distance to private blocks is the same in both architectures.

5.2.2. L2 Cache Access Time

We have seen that Nahalal reduces the distance between data and its owners. We now examine how this impacts the cache access time. Figure 10 presents the average L2 cache access time (in cycles) for the three architectures. As can be seen from the figure, Nahalal reduces the average cache delay by 20.12% on average over DNUCA, to the extent of 37.2% in the apache benchmark.

To explain the results of Figure 10, the cache access time can be estimated using a simple Formula. The following parameters impact the access time:

$P_p =$	*Percentage of accesses to private data out of total L2 cache accesses*
$P_s =$	*Percentage of accesses to shared data out of total L2 cache accesses*
$P_{ws} =$	*Percentage of writes to shared data out of total L2 cache accesses (included in the above)*
$\bar{N}_p =$	*Average distance(in hops) of accesses to private banks*
$\bar{N}_s =$	*Average distance (in hops) of accesses to shared banks*

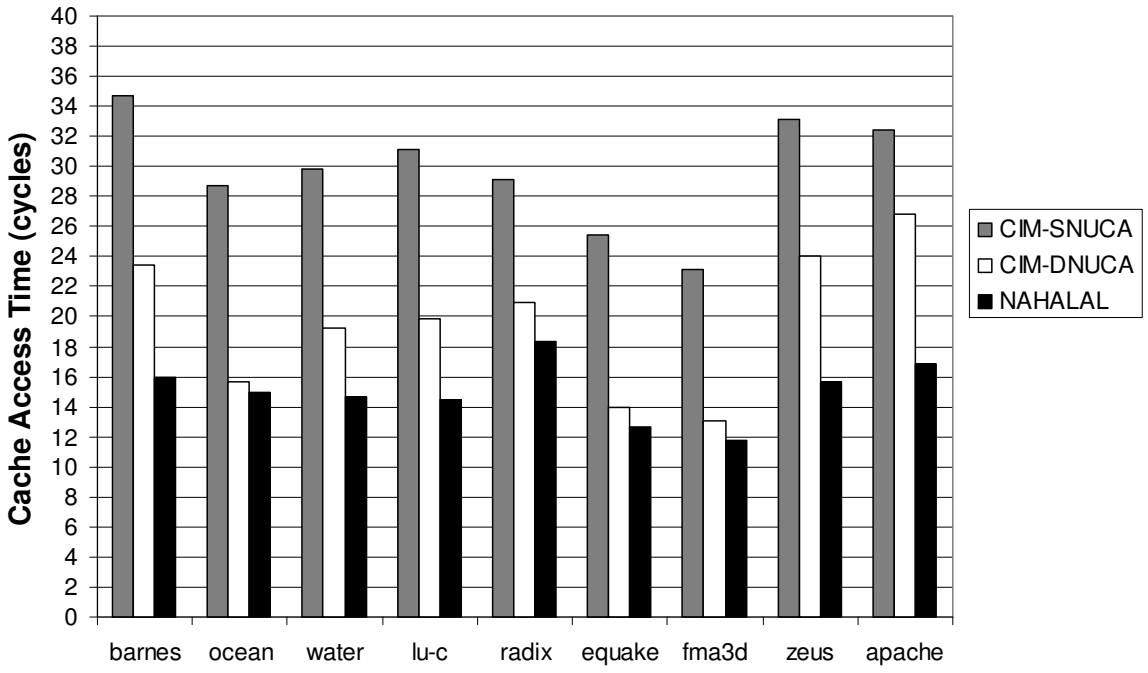


Figure 10. Average cache delay in clock cycles for CIM-SNUCA, CIM-DNUCA and Nahalal. The delay closely follows the average distance measured above.

In addition, the access time is affected by the per-hop latency in the NoC, namely δ cycles, and the cache bank latency, T_b cycles, defined in Table 2 above. Based on these parameters, we compute the predicted average cache access time (in cycles) using the following formula:

$$T_{L2} = T_b + P_p \cdot \left(2 \cdot \bar{N}_p \cdot \delta\right) + P_s \cdot \left(2 \cdot \bar{N}_s \cdot \delta\right) + P_{ws}\left(2 \cdot \bar{N}_s \cdot \delta\right) \quad (1)$$

We now explain the above formula. First, every memory access requires accessing the appropriate memory bank, which takes T_b cycles, if we neglect queuing time. Second, the access requires a two-way traversal of the distance between the processor and the data, which takes $2\bar{N}_p\delta$ cycles on average for private blocks, and $2\bar{N}_s\delta$ cycles for shared ones. In addition, every write operation to a read-write shared variable entails an additional penalty for invalidating copies held by other sharers. For simplicity, we assume that all invalidations occur in parallel, and the time to invalidate all relevant copies is the same as the average round-trip time. This gives a slight under-estimate of the invalidation time when there are many sharers.

We now examine how closely the above formula predicts the cache access times measured in the simulations. We extract from the simulations the average distance values (as presented in the previous section) as well as the percentages of private and shared accesses, and the percentage of accesses that are writes to shared data. We then use the formula above to predict the average cache access time.

Figure 11 shows the predicted cache access time versus the one measured in the simulations, for CIM-DNUCA and Nahalal. Although we have neglected some factors (such as contention, and multiple concurrent invalidations) in the formula, we see that their impact is small, and the formula provides a good explanation of the measured results. In particular, Nahalal's improved cache access time can be fully attributed to the fact that it places data closer to processors that access it.

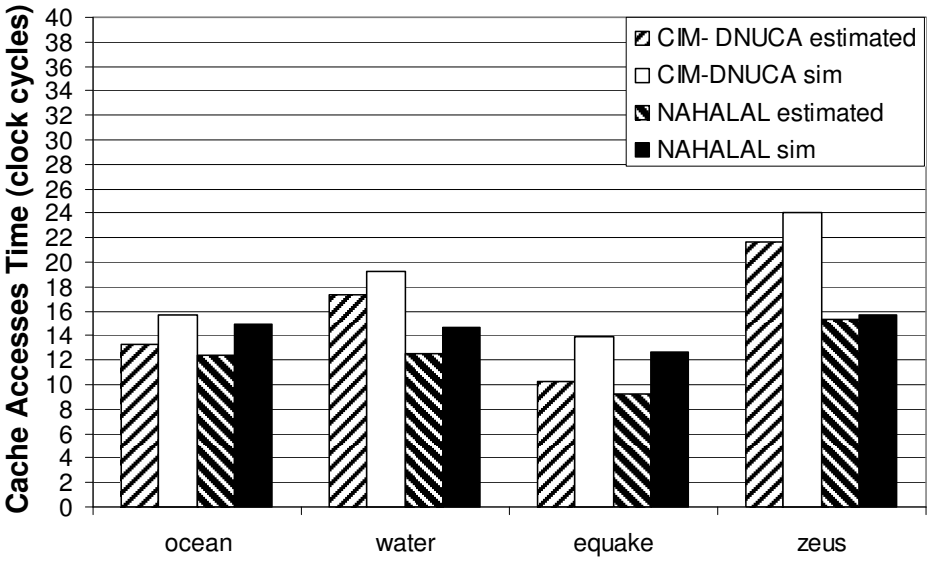


Figure 11. Predicted versus simulated cache access time for CIM-DNUCA and Nahalal. The prediction is a slight underestimate of the detailed simulation.

5.2.3. Overall Performance

Finally, we examine how Nahalal impacts the overall system performance. Figure 12 presents

benchmark running times, normalized to the running time of CIM-SNUCA. We see that Nahalal improves overall system performance in all benchmarks, but in some more significantly than in others. The average reduction in running time is 8.58%, and the best improvement is in apache, which improves by 16.3%. The differences in the improvement stem from variations in the role L2 caches play in the different benchmarks. Recall that Nahalal only modifies the organization of L2 caches, and does not impact performance of access to L1 or to off-chip memory.

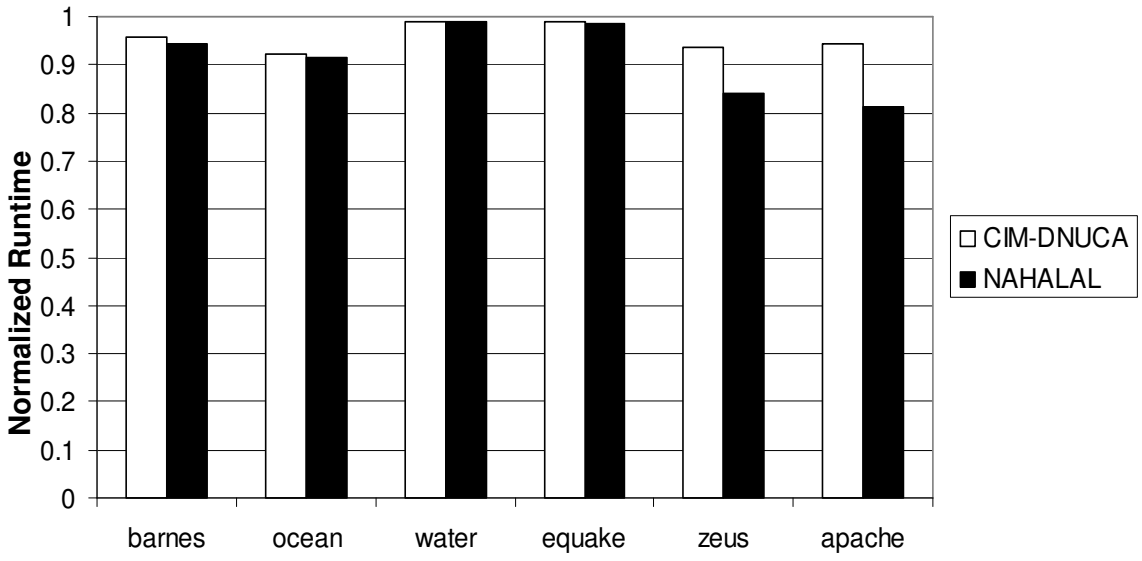


Figure 12. Runtime for CIM-DNUCA and Nahalal, normalized to the runtime of CIM-SNUCA. Nahalal achieves significant performance gains in benchmarks with many accesses to shared blocks in L2 caches, such as the typical commercial workloads (zeus and apache). In benchmarks with little sharing, like ocean and barnes, Nahalal's performance improvement is less significant. In benchmarks such as water, where most shared blocks are read-only and fit in L1 caches, and in benchmarks with a high L2 miss rate like equake, the organization of the L2 cache is immaterial to performance.

In commercial benchmarks like apache and zeus, there are many accesses to shared blocks that reside in L2 caches, both read-write shared blocks, and read-only blocks that do not fit in L1 caches. Therefore, Nahalal significantly improves the performance of these benchmarks. Other benchmarks, like barnes and ocean, rely heavily on L2, but have few accesses to shared data (see Table 1), and therefore benefit less from Nahalal. Nahalal's improvement is close to that of CIM-DNUCA in such benchmarks. In benchmarks like water, with a high L1 hit-rate, L2 cache

organization is immaterial to performance. Similarly, when the L2 hit-rate is low, e.g., as in equake, the performance bottleneck is access to external memory, and again L2 organization has virtually no impact

As global wire delays become more and more dominant, the importance of placing data in proximity to the clients is expected to become more evident. Figure 13 demonstrates this effect for the apache benchmark. It depicts the speedup in runtime over the CIM-SNUCA system for both CIM-DNUCA and Nahalal as the per-hop link delay, δ, increases. Although both systems exhibit more speedup as wire delay increases, the relative gain of Nahalal keeps growing as technology scales. This is because distance-related delay becomes dominant and Nahalal is more effective in reducing the average access distances. As expected, the performance improvement is linear, since the runtime is the sum of processing time and cache delay, and from Equation 1, the cache delay is proportional to δ.

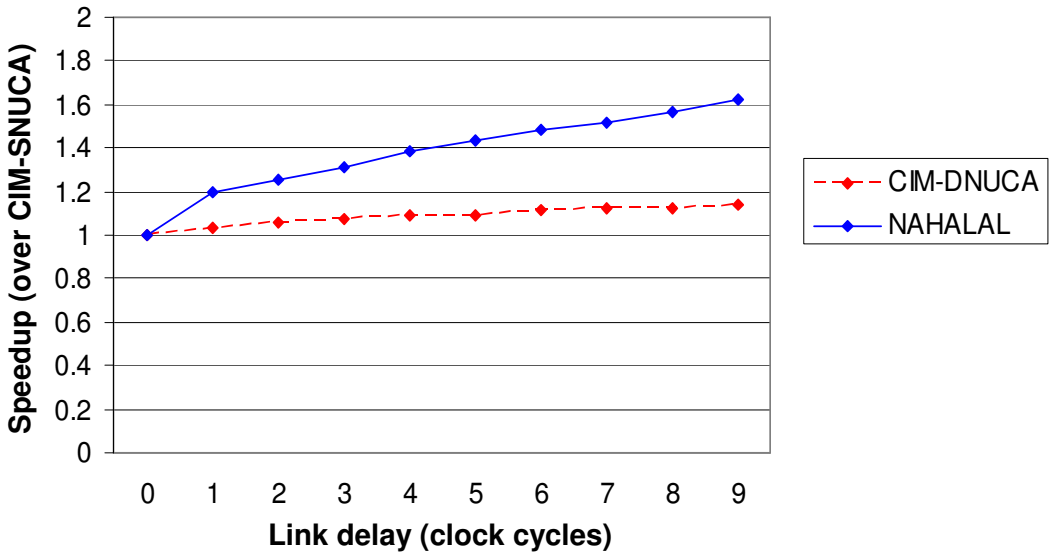


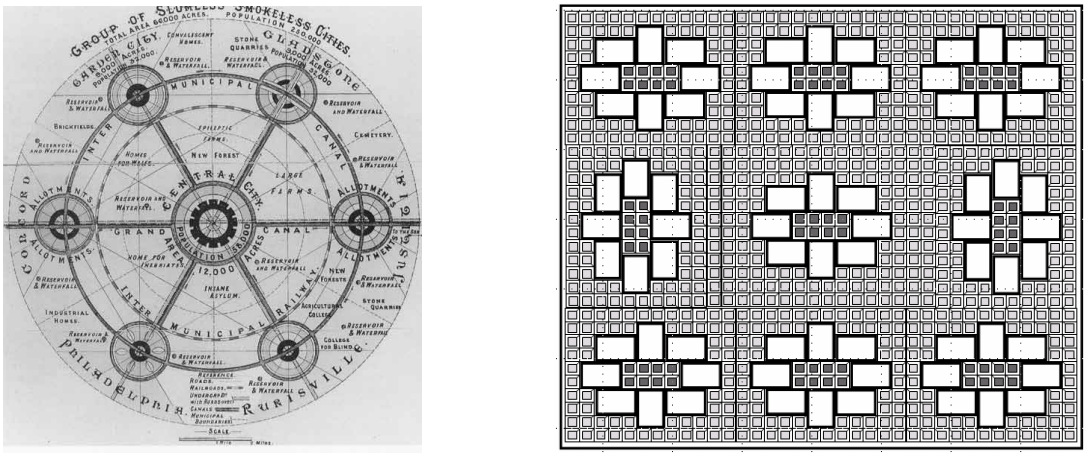
Figure 13. Runtime speedup over SNUCA for both DNUCA and Nahalal for different link delays (in clock cycles). The results are given for the apache benchmark. Nahalal's performance gain increases as the wire delay becomes more dominant.

VI. Summary and Discussion

The shift to CMPs makes the on-chip memory system a primary performance bottleneck. The use of non uniform cache techniques encourages vicinity of reference as means to mitigate the growing effect of global wire delays. Prior CMP designs suffer from the remoteness of shared data blocks, which end up residing far from all sharers. In this work, we have characterized the accesses to shared data and found that, in many standard parallel applications, accesses to shared data consume a significant fraction out of the total memory accesses. Therefore, the remoteness of shared blocks hinders cache performance and calls for an architectural solution.

We have leveraged this phenomenon (referred to as the shared hot-blocks effect) to devise Nahalal - a novel CMP topology that locates shared data close to all sharers and still preserves vicinity of private data for all processors. We have demonstrated the potential of our new design via a full-system simulation and comparison to the classic cache-in-the-middle design. We have shown that on standard benchmarks, Nahalal achieves performance improvement of up to 16.3% percent.

Despite its merits, it is important to note that, like the traditional cache-in-the-middle layout, Nahalal also has limited scalability. This is since the middle area grows as the square of the number of processors around the circumference. While Nahalal's design is feasible for a moderate number of processors, massively scalable CMPs employing hundreds of processors clearly require a different topology. In such systems, we believe that the most appropriate solution is to employ a *clustered* design, whereby the processors and memory banks are organized as a collection of closely knit clusters.



(a) A cluster of Garden-Cities [26]. **(b) Clustered Nahalal CMP design.**

Figure 14. A clustered design for CMP, where each cluster is organized in the form of Nahalal.

Such a design supports the execution of multiple multi-threaded applications, one on each cluster. In this context, one can organize each cluster like Nahalal, for maximum performance of the application running therein. Designing and evaluating such scalable architectures is an interesting direction for future work.

VII. REFERENCES

[1] R. Ho, K. Mai, and M. Horowitz, The future of wires. Proceedings of IEEE,89(4), April 2001.

[2] V. Agarwal, M. S. Hrishikesh, S.W. Keckler, and D. Burger. Clock rate vs. IPC: The end of the road for conventional microprocessors. In Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 248–259, June 2000

[3] WJ Dally and S. Lacy. VLSI Architecture: Past, Present, and Future, In Proceedings of the Advanced Research in VLSI conference, Jan. 1999, pp. 232--241.

[4] G. D. Micheli and L. Benini. Networks on Chips: Technology and Tools. Morgan Kaufmann, San Francisco 2006, ISBN-13:978-0-12-370521-1

[5] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip multiprocessor caches. In MICRO 37, pages 319-330, Dec. 2004

[6] R. Kauffmann. Planning of Jewish settlements in Palestine. Town Planning Review 12(2):93-116, 1926

[7] Frank Hurley, Nahalal Jewish Settlement N. Palestine (1), nla.pic-an23565289, National Library of Australia.

[8] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In ASPLOS X, pages 211–222, Oct. 2002

[9] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger and S.W. Keckler. A NUCA substrate for Flexible CMP Cache Sharing. International Conference on Supercomputing (ICS 05) , June, 2005

[10] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, and G. Hallberg. Simics: A full system simulation platform. IEEE Computer, 35(2):50–58, Feb. 2002.

[11] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in cmps. In Proceedings of the 32nd annual International Symposium on Computer Architecture (ISCA-32), 2005.

[12] Bradford M. Beckmann, Michael R. Marty, and David A. Wood, Balancing Capacity and Latency in CMP Caches, Univ. of Wisconsin Computer Sciences Technical Report CS-TR-2006-1554, February 2006

[13] M. Zhang and K. Asanovic, Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors, ISCA-32, June 2005.

[14] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. Jones, and B. Parady. SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. In Workshop on OpenMP Applications and Tools, pages 1–10, July 2001.
[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 24–37, June 1995.
[16] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In Measurement and Modeling of Computer Systems, pages 151–160, June 1998.
[17] http://www.zeus.com/products/zws/
[18] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. Programming Language Design and Implementation (PLDI), Chicago, IL, June 2005.
[19] P. Guerrier and A. Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. Proc. DATE 2000
[20] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. Proc. DAC 2001
[21] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "Cost Considerations in Network on Chip", Integration-The VLSI Journal, special issue on Network on Chip, Volume 38, Issue 1, October 2004, pp. 19-42.
[22] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian, Leveraging Bloom Filters for Smart Search Within NUCA Caches, 7th Workshop on Complexity-Effective Design (WCED), June 2006
[23] M. S. Papamarcos and J. H. Patel. A Low-overhead Coherence Solution for multiprocessors with Cache memories. In Proceedings of the 11th Annual International Symposium on Computer Architecture, pages 348--354, June 1984.
[24] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese., Piranha: A scalable architecture based on single-chip multiprocessing. In The 27th Annual International Symposium on Computer Architecture, pages 282–293, June 2000.
[25] I.T.R. for Semiconductors. ITRS 2003 Edition. Semiconductor Industry Association, 2003. http://public.itrs.net/Files/2003ITRS/Home2003.htm.
[26] Ebenezer Howard, Garden Cities of To-Morrow. 1902. London: Swan Sonnenschein & Co., Ltd.