

# An Adaptive Cache Structure for Future High-Performance Systems

Changkyu Kim   Doug Burger   Stephen W. Keckler

Computer Architecture and Technology Laboratory  
Department of Computer Sciences  
The University of Texas at Austin  
cart@cs.utexas.edu — www.cs.utexas.edu/users/cart

*IBM Technical Sponsors - John Keaty and Rob Bell*

## Abstract

*On-chip cache sizes are likely to continue to grow over the next decade as working sets, available chip capacity, and memory latencies all increase. Traditional cache architectures, with fixed sizes and discrete latencies, lock one organization down at design time, which will provide inferior performance across a range of workloads. In addition, expected increases in on-chip communication delays will make the time to retrieve data in a cache a function of the data's physical location. Consequently, cache access times will become a continuum of latencies rather than a single one. This non-uniformity will make static organizations particularly limited for single-chip servers, in which multiple processors will be different distances from the cache controller. In this paper, we propose a set of adaptive, high-performance cache design, called Non-Uniform Cache Architectures (NUCAs). We extend these physical designs with logical policies that allow important data to migrate closer to the processor within the same cache. We show that these adaptive level-two NUCA designs provide 1.6 times the performance of a Uniform Cache Architecture of any size, and that the adaptive NUCA scheme outperforms static NUCA schemes by 9% for multi-megabyte, on-chip server caches with large numbers of banks.*

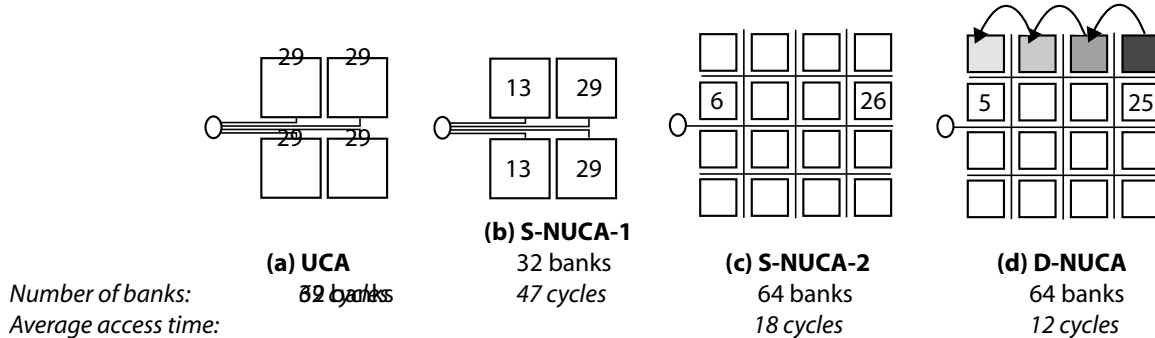
## 1 Introduction

Long memory latencies and limited off-chip bandwidth have driven steady, consistent increases in the sizes of on-chip caches. Processors in late 1980s only included a small level-1 cache (such as the 8KB cache on the first Intel 80486), and these structures grew to 64-128KB in the mid 1990's [19]. Today's high performance processors have continued to increase cache capacities, such as the Alpha 21364 [10] with a 1.5MB L2 cache, and the HP PA-8700

with 2.25MB of combined on-chip cache capacity [12]. The size of cache memories integrated on the processor dies are expected to continue to increase as the bandwidth demands on the package grow [14], as smaller technologies permit more bits per  $mm^2$ , and as larger workloads produce correspondingly larger working sets. Demonstrating the likely trend toward even larger on-chip memory systems is the development of large off-chip level-3 caches in today's computers, such as those found in IBM's POWER4 systems [8].

Current multi-level cache hierarchies are organized into a few discrete levels. Typically, each level replicates the contents of the smaller level above it (if inclusion is obeyed), and accesses to the levels in the cache hierarchy are serialized. An access to main memory requires misses in all levels of the hierarchy, assuming that parallel lookups are not used. Cache designers have typically sized the caches so that each successively larger level of cache has an order of magnitude greater capacity and access time. For many applications, the large increase in cache capacity at each level greatly reduces number of misses there and compensates for the added overheads of having the additional level.

In future technologies, large on-chip caches with a single, discrete hit latency will be undesirable, due to increasing global wire delays across the chip [23]. Data residing in the part of a large cache close to the processor could be accessed much faster than data that reside physically farther from the processor. For example, according to our projections the closest bank in a 16-megabyte, on-chip level-two cache built in a 50-nanometer process technology, could be accessed in 5 cycles, while an access to the farthest bank might take 25 cycles. A secondary problem with a monolithic cache architecture is that it can force accesses to be sequentialized, resulting in higher latencies when accesses are pending. While this problem can be mitigated by adding



**Figure 1. Level-2 Cache Architectures.**

additional ports to the cache, the cost of these ports for a large cache is prohibitive.

Allowing the banks within a cache to be accessed at their best individual speeds has the potential to improve performance. In this paper, we evaluate physical and logical designs for Non-Uniform Cache Access (NUCA) organizations and compare them to traditional Uniform Cache Access (UCA) architectures. Figure 1 shows the range of designs that we evaluate. We first examine an aggressive UCA organization (a) with multiple independently accessible banks but equal access times to all banks. We then examine three NUCA architectures: (b) Static-NUCA-1: multiple independent banks with different latencies and each bank possessing a private channel to the processor; (c) Static-NUCA-2: the same organization as S-NUCA-1 but with a switched network to reduce the interconnect costs and increase the scalability; and (d) Dynamic NUCA: which employs a dynamic policy to migrate frequently used data to the banks closer to the processor. To implement the migratory policy requires a consistent addressing scheme so that the desired data can always be located within the cache. For this, we propose “spread sets”, in which multiple banks each hold one way of a set. The goal of our simple migratory policies of D-NUCA is to place the most-recently used line in each set into the bank closest to the processor, while limiting the number of times that line must be copied from bank to bank. Our policies gradually migrates frequently used lines into banks progressively closer to the processor. If a line is accessed infrequently, the migration policies move it to the farthest bank before finally evicting it to main memory.

Our results show that both latency and concurrent access are critical to high performance level-2 caches. Increasing the number of cache banks helps both the UCA and NUCA architectures, but the switched networks of S-NUCA-2 and D-NUCA provide sufficiently low link contention to different banks without imposing the more substantial area (and therefore speed) penalties from the dedicated bus schemes. The migration of data in D-NUCA enables this scheme to

outperform the static NUCA architectures because key data is automatically moved into closer banks, a strategy that permits stable performance across applications. In the UCA and static NUCA schemes, larger caches can actually perform worse than smaller caches if the reduction in miss rate does not offset the increase in access time to critical data that may be mapped to a distant region of the cache. The D-NUCA scheme permits a single large cache to outperform all smaller cache sizes, across all applications, resulting in greatly increased performance stability for a given cache capacity.

The remainder of this paper is organized as followed. Section 2 presents the basic methodology for evaluating cache architectures, including our method of estimating cache latencies and a description of the microprocessor timing simulator. Section 3 describes the baseline conventional cache organization and quantifies how they are likely to perform in emerging fast clock rate and wire-dominated technologies. Section 4 details the architecture of a scalable non-uniform access cache including both cache banks and network alternatives. Section 5 presents several data mapping policies to migrate important data to nearer cache banks to exploit the non-uniformity of the cache architecture. Our results show that the best policy achieves an average hit time of 11.5 cycles compared to 69 cycles for a uniform access cache 16MB cache, and 18 cycles for a statically mapped NUCA cache. This improvement in access time results in a 9% performance (IPC) improvement over a static NUCA cache. We discuss related work in Section 6, and conclude in Section 7.

## 2 Experimental Methodology

To evaluate a spectrum of alternatives, we developed simulators for the level-2 cache architecture and wedded them to an existing microarchitecture simulator. To estimate the cache bank delay, we used Cacti 3.0 which accounts for capacity, organization, and fabrication technology [30]. Since Cacti produces timing estimates in nanoseconds, we

SPECINT2000	Phase		L2 load acc/Minstr	SPECFP2000	Phase		L2 load acc/Minstr
	FFWD	RUN			FFWD	RUN	
176.gcc	2.367B	300M	25,900	172.mgrid	550M	1.06B	21,000
181.mcf	5.0B	200M	260,620	177.mesa	570M	200M	2,500
197.parser	3.709B	200M	14,400	173.applu	267M	650M	43,300
253.perlbnk	5.0B	200M	26,500	179.art	2.2B	200M	136,500
256.bzip2	744M	1.0B	9,300	178.galgel	4.0B	200M	44,600
300.twolf	511M	200M	22,500	183.quake	4.459B	200M	41,100
Speech				NAS			
sphinx	6.0B	200M	54,200	cg	600M	200M	113,900
				bt	800M	650M	34,500
				sp	2.5B	200M	67,200

**Table 1. Benchmarks used for performance experiments**

converted these cache delays to processor cycles by assuming an aggressive clock of 8 FO4 inverter delays per cycle at each technology<sup>1</sup>. To model the interconnect between the cache banks, we used existing wire delay models [1] and for the switched networks, we incorporated simple delay estimates for the switches obtained from circuit designs and HSPICE simulations. For all L2 cache configurations we simulate 64-byte lines and, unless otherwise stated, assume a 4-way set-associative L2.

To model the effects of different cache organizations on performance, we used the `sim-alpha` simulator [7], which models an Alpha 21264 core in detail [19]. All microarchitectural parameters in the processing core match those of the 21264, including issue width, fetch bandwidth, and clustering. The L1 data cache of `sim-alpha` is similar to those of the 21264: 3-cycle access to the 64KB L1 data cache (32-byte lines, 2-way associative), and single-cycle access to the similarly configured L1 I-cache. As with the cache memory system, we assumed (somewhat optimistically) that the processor clock period is equal to 8 FO4 inverter delays. Finally, we assume an unloaded 132-cycle access to main memory, obtained by scaling the memory latency of an actual Alpha 21264 system by the more aggressive clock rate.

Although our cache delay estimates in cycles are scaled for future technologies, the 4-wide issue processor that we simulate is likely to be significantly less powerful than the processors that will be available in the studied timeframe. However, we note that more powerful processors will place a higher load on the L2 cache, particularly if L1 caches continue their recent decline in size. The results presented in this paper are thus conservative, as we expect even better relative performance for the NUCA strategies with a higher-performing core and imposing more cache contention.

<sup>1</sup>One FO4 is delay of one inverter driving four copies of itself. Delays measured in FO4 are independent of technology, and one FO4 roughly corresponds to 360 pico-seconds times the transistor's drawn gate length in microns [13].

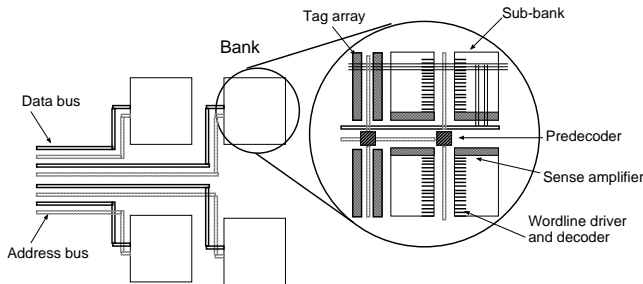
Table 1 shows the benchmarks used in the experiments of this paper. The suite of 16 applications includes six floating-point benchmarks from the SPEC2000 suite [32], six SPEC2000 integer benchmarks, three scientific applications from the NAS suite [4], and one speech recognition benchmark called Sphinx [22]. For each benchmark we simulated the sequence of instructions which capture the core repetitive phase of the program. The phases were determined empirically by plotting the L2 miss rates over one execution of each benchmark, and choosing the smallest subsequence that captured the recurrent behavior of the benchmark. For each benchmark, Table 1 lists the number of instructions skipped to reach the phase (FFWD) and the number of instructions simulated (RUN). A more rigorous method of choosing simulation phases will be used in future work [29]. Finally, Table 1 shows the anticipated load on the L2 cache by listing the number of L2 accesses per 1 million instructions given 64KB level-1 instruction and data caches (this metric was proposed by Kessler *et al.* [20]).

### 3 Uniform Access Caches

Modern level-two caches no longer employ a single monolithic data array, and instead are subdivided into multiple smaller sub-banks to minimize the access time. In addition, they are typically single ported as adding additional physical ports to the SRAM cells incurs a tremendous area penalty. Popular cache modeling tools, such as Cacti, enable fast exploration of the cache design spaces by automatically optimizing for sub-bank count, size, and orientation [17, 35]. However, as cache capacities grow, existing cache architectures and the tools used to model them break down. First, large caches are much more sensitive to wire delays and great care must be taken to optimize and accurately model the communication paths. Second, our experiments show that sequentialized access to large L2 caches increases the average access time by a factor of 8-10 over the access time of a single access to an idle cache, indicating

Technology (nm)	L2 capacity	Num. banks	Num. sub-banks	Unloaded latency	Loaded latency	IPC	Miss rate
180	1MB	8	4	8	18.8	0.46	0.31
130	2MB	8	4	11	27.6	0.47	0.23
100	4MB	8	4	14	36.7	0.47	0.21
70	8MB	16	8	20	46.3	0.49	0.18
50	16MB	32	4	29	68.8	0.43	0.14

**Table 2. Performance of UCA organizations**



**Figure 2. Banked cache design**

a greater demand for cache bandwidth.

An alternate and emerging cache architecture is shown in Figure 2. A large cache is first partitioned into banks, each with its own dedicated address and data bus. Then each bank is divided into sub-banks to minimize for local bank access time. This architecture has two advantages over the traditional monolithic design. First, contention can be reduced as multiple cache references can proceed simultaneously, as long as a bank conflict does not occur. Second, the coarser grain partitioning enables the bus wires to each bank to be engineered separate from a bank design, and thus be made faster. We use as our baseline this more aggressive cache architecture so as to not overemphasize the benefits of the non-uniform architectures presented in Sections 4 and 5. To model this uniform cache access (UCA) architecture, we start with an updated version of Cacti 3.0 which enables the user to optimize for both banking and sub-banking. However, to more realistically model large caches, we replace the Rubenstein RC wire delay model [15] in Cacti 3.0 with a more aggressive repeater and scaled wire model of Agarwal *et al.* [2] for the long address and data busses to the banks. Our complete cache delay model includes the access time of the banks, the transmission time on the wires, and the contention for the banks and the wires.

Table 2, shows the results for the UCA organization using the delay modeling described above, the *sim-alpha* simulator, and our custom L2 cache simulator. We vary the technology generation, and assume the largest cache size predicted by the SIA Roadmap [28], from 1MB of on-chip L2 at 180 nanometer devices to 16MB at 50 nanometer devices. The unloaded latency is the average access time

assuming a uniform bank access distribution and no contention. The loaded latency is obtained by averaging the L2 cache access time, including contention, across all of the benchmarks. The reported IPCs are the harmonic mean of all IPC values across our benchmarks, and the cache configuration displayed for each capacity is the one that produced the best IPC.

As the cache capacities increase, the average access latencies increase by a factor of 2-3 to a maximum of nearly 69 cycles for a 16MB cache when contention is considered. By contrast, modeling a more conventional cache with only sub-banking and sequential access, as produced by Cacti 2.0, has a average loaded latency of nearly 600 cycles for the same configuration. With the banked UCA cache, the miss rate steadily drops from 0.31 to 0.14 as capacity is increased. However, note that the best average performance is achieved at a cache capacity of only 8MB because the decrease in miss rate is counteracted by the increase in access time. Some of the applications in our benchmark suite have working sets smaller than 8MB; for these applications increasing the cache capacity only increases latency without helping the miss rate because the working set just gets spread out over a larger and slower cache structure. Better performance could be obtained for these benchmarks if closer banks can be accessed as faster speeds than more distant banks. The next section presents and evaluates two designs that permit non-uniformity in the bank access latencies.

## 4 Static Non-Uniform Access Caches (S-NUCA)

When the entire chip was reachable in one cycle, only small fractions of a cycle would differentiate bank access latencies within a cache. However, as global wire delays grow, the opportunity for accessing close banks in a cache faster than far banks will grow correspondingly. To access banks with *non-uniform* latencies in a single cache, the banks must be independently accessible, and the routing topology to and from the banks must support non-uniform accesses—one data bus shared among all the banks would not permit non-uniform access.

In this section, we examine two NUCA designs that have

Technology (nm)	L2 size	Num. banks	Num. sub-banks	Unloaded latency			Loaded latency	IPC	
				bank	min	max			avg.
180	1MB	4	4	6	6	7	7	17.2	0.47
130	2MB	16	8	5	7	11	10	24.4	0.48
100	4MB	32	4	5	8	15	13	31.8	0.49
70	8MB	32	4	6	10	21	16	36.3	0.53
50	16MB	32	4	7	13	29	21	46.9	0.52

**Table 3. S-NUCA-1 performance**

similar bank organizations but different interconnects: one using per-bank private channels (S-NUCA-1), and one using a lightweight switched 2-D mesh network that connects all the banks and the controller (S-NUCA-2). The S in S-NUCA denotes *static*; cache lines are statically mapped to a bank and cannot migrate to other banks. We explore that migratory capability with dynamic NUCA (D-NUCA) schemes in Section 5.

#### 4.1 Channel implementations (S-NUCA-1)

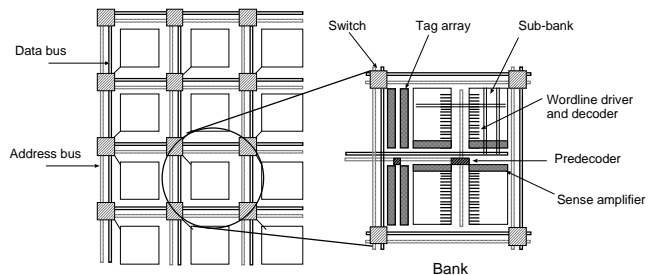
We first evaluate a NUCA implementation that closely resembles the UCA organization of the previous section (depicted in Figure 2), except that the controller can treat each bank as an independently addressable unit with its own distinct latency. As with all implementations measured thus far, each bank is 4-way set associative, and the organizations at each size (number of banks and sub-banks) are those that maximize overall performance.

In a NUCA implementation, there are three contributors to latency of each bank: the wire delay required to route addresses and data to and from the bank, the bank access time, and any contention incurred at the bank or channel. As the number of banks for a given size increases, the routing delay becomes a larger component of the access time and the bank access delay becomes smaller. In our model of the S-NUCA-1 organization, we assume that the wire delay is included in bank contention; that the controller cannot send a new request to a bank on its private channel until the previous request has returned.

In Table 3, we show a breakdown of the access delays as a function of technology: the raw bank access delay, the minimum, average, and maximum access latency, and the average latency seen at run-time (including channel contention). We assume that the cache controller resides in the middle of one side of the bank array, so the farthest distance that must be traversed is half of one dimension and the entire other dimension.

As technology advances, the access times of individual banks remain roughly the same, but the worst-case routing delay grows by 4 to 8 cycles each generation. At 180nm, the routing delay is negligible (one cycle), but at 50nm, the routing delay ranges from a best case of 6 cycles to a worst case of 22 cycles. The S-NUCA-1 cache shows lower la-

tency than the UCA organization at all caches greater than 1MB. At 16MB, the S-NUCA-1 average latency is 47 cycles instead of 69 for UCA. At 8MB, the optimal S-NUCA-1 organization has more banks than the optimal UCA organization, since more banks permit lower latency in a non-uniform cache. However, at 16MB, the area overhead added by the private channels limits both organizations to 32 banks. The best cache size for the S-NUCA-1 cache is 8MB, since the static data mapping causes overly slow access for some key blocks in the 16MB cache. Performance for the simulated 16MB cache system is 12% worse, on average, than for the 8MB cache system.



**Figure 3. Switched NUCA design**

#### 4.2 Switched implementations (S-NUCA-2)

While the S-NUCA-1 organization shows improved performance over the UCA cache, that organization doesn't scale to small technologies, simply due to the large number of wires that will be required to have private channels to the increasing number of banks. Figure 3 shows an alternative static NUCA design, called S-NUCA-2, that uses a lightweight 2-D mesh, with point-to-point links, and simple routers at each bank intersection to route addresses, read data, and writes. Each link has separate wires for bidirectional routing. We modeled the switch logic in HSPICE to obtain the FO4 delay for use in determining routing delay, which is the sum of the wire delay between two banks and switch delay, multiplied by the number of hops.

Since we assume that the network is synchronous, each switch-to-switch delay must take an integer number of cycles (the switches effectively function as latches). Thus, if

Technology (nm)	L2 size	Num. banks	Num. sub-banks	Unloaded latency				Loaded latency	IPC
				bank	min	max	avg.		
180	1MB	8	4	5	5	9	7	10.7	0.48
130	2MB	8	8	6	7	11	9	12.4	0.53
100	4MB	16	4	6	6	16	11	14.9	0.57
70	8MB	32	4	6	7	19	13	15.8	0.65
50	16MB	64	4	6	6	26	16	17.7	0.70

**Table 4. S-NUCA-2 performance**

the wire delay along one bank was 2.3 cycles, the penalty between switches would be 3 cycles. We make one optimistic assumption, that requests can be buffered at banks in the case of a bank conflict. This assumption is comparable to assuming that an intelligent scheduler places a request on the network so that it arrives immediately after the bank has just become free.

Table 4 shows the performance of the S-NUCA-2 design, with the same set of experiments shown in Table 3. For 4MB caches and up, the minimum, average, and maximum bank latencies are significantly reduced, for two reasons. First, the switched network consumes much less area than the private, per-bank channels, resulting in a smaller array and faster access to all banks. Second, since the wire overhead per bank is reduced, larger numbers of banks are possible: the optimal performance point of S-NUCA-2 is at 64 banks, not 32 as in S-NUCA-1. Since the latencies are lower, the queuing delay is less severe, resulting in significantly lower average access penalties. Finally, allowing a scheduler to hide the latency of the routing delay by putting requests on the switched network early prevents the communication delay from causing bank contention. The S-NUCA-2 cache is faster at every technology than S-NUCA-1, and furthermore at 50nm with a 16MB cache, the average loaded latency is a mere 18 cycles, as opposed to 47 cycles for S-NUCA-1. That reduction causes a 35% average improvement in IPC across the benchmark suite.

## 5 Dynamic NUCA Implementations

To enhance the effectiveness of NUCA architectures, more frequently accessed data should be placed in closer banks while less important (yet still cached) data should be placed in farther banks. In this section, we evaluate a number of policies that automatically migrate data among the banks to reduce average L2 cache access time and improve overall performance. With migration policies, we must answer three fundamental questions about the management of the data: (1) how the data are mapped to the banks—in which banks and which cache bank lines can the data reside, (2) how are the possible locations searched to find the data, and (3) under what conditions should the data be migrated from bank to bank.

### 5.1 Logical to Physical Mapping

At one extreme are the S-NUCA strategies, in which a line of data can only be mapped to a single statically determined bank. At the other extreme, a line could be mapped into any cache bank. While this maximizes the placement flexibility, the overhead of locating the line may be too large as each bank must be searched, either through a centralized tag store or by broadcasting the tags to all of the banks. We explore a solution called *spread sets* in which a the multi-banked cache is treated as a set-associative structure, each set is spread across multiple banks, and each bank holds one “way” of the set. The collection of banks used to implement this associativity is called a *bank set* and the number of banks in the set would correspond to the associativity. A cache can be comprised of multiple bank sets. For example, a cache array with 32 banks could be organized as a 4-way set-associative cache, with eight bank sets, each consisting of 4 cache banks. To check for a hit in a spread-set cache, the pertinent tag in each of the 4 banks of the bank set must be checked. Note that the primary distinction between this organization and a traditional set-associative cache is that the different associative ways have different access latencies.

We propose three methods of allocating banks to bank sets and ways: *simple mapping*, *fair mapping*, and *fast shared mapping*. With the simple mapping, shown in Figure 4a, each column of banks in the cache becomes a bank set, and all banks within that column comprise the set-associative ways. Thus, the cache may be searched for a line by first selecting the bank column, the selecting the set within the column, and finally performing a tag match on banks within that column of the cache. The two drawbacks of this scheme are that the number of columns may not correspond to the number of desired ways in each bank set, and that latencies to access all bank sets are not the same; columns (bank sets) farther from the L2 cache controller will be slower to access than columns closer to it.

Figure 4b shows the *fair mapping* policy, which addresses both problems with the simple mapping policy at the cost of some additional complexity. The mapping of sets to the physical banks is indicated with the arrows and shading in the diagram. With this model, banks are allocated to bank sets so that the average access time across all bank

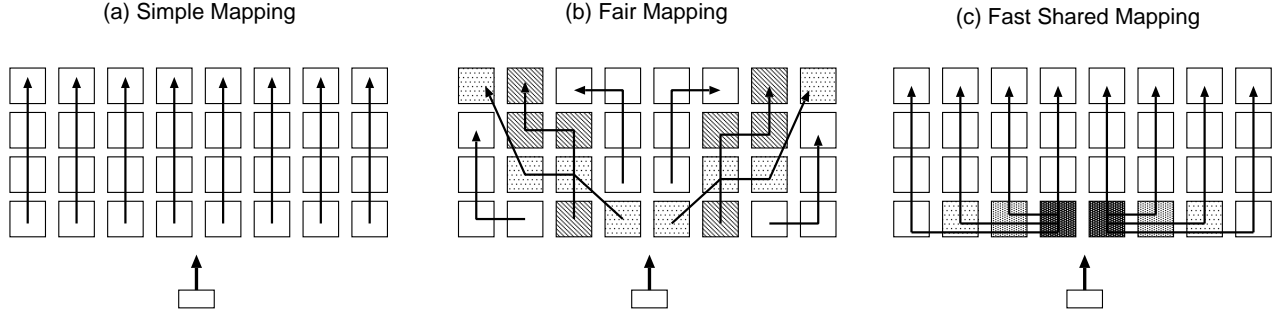


Figure 4. Mapping bank sets to banks.

sets are equalized. Since bank sets no longer correspond directly to a fixed physical topology (such as columns), the number of bank sets can be chosen to provide the desired number of ways per bank set. However, *fair mapping* does not completely solve the problem of uneven latencies across bank sets, as not all bank sets can have their most frequently accessed bank located next to the L2 cache controller.

The *fast shared* mapping policy, shown in Figure 4c, attempts to equalize all bank sets by allowing them to share the banks closest to the cache controller. This policy requires more complex per-bank and line migration control, but has the potential to have the most important lines from every bank set in the closest banks. The shading in the diagram indicates the intensity of contention for the banks in the row nearest to the cache controller. We do not evaluate this policy in this paper, leaving it for future work.

## 5.2 Locating Data

Searching for a line among a bank set can be done with two distinct policies. The first is *incremental search*, in which the banks are searched in order starting from the closest bank until the data is found or a miss occurs in the last bank. Since the closest banks ideally hold the needed data the majority of the time, this policy minimizes the number of messages in the cache network, and keeps energy consumption low, since fewer banks are accessed while checking for a hit.

The second policy is called *multicast search*, in which the requested address is multicast to some or all of the banks in the requested bank set. Lookups proceed roughly in parallel, but at different times due to routing delays through the network. This scheme offers higher performance at the cost of increased energy consumption and network contention, since hits to banks far from the processor will be serviced faster than in the incremental search policy. Miss signals must still be propagated across the banks since off-chip memory must be accessed only if the data is not found in any bank.

These two policies are actually the two endpoints of a

continuum. Intermediate policies are *limited multicast*, in which the first  $M$  of the  $N$  banks in a bank set are searched in parallel, followed by an incremental search of the rest. Most of the hits will thus be serviced by a fast lookup, but the energy and network bandwidth consumed by accessing all of the ways at once will be avoided. A second intermediate policy is *limited search*, in which the first few close banks are searched incrementally, minimizing energy consumed on hits, and the remaining banks are searched with a multicast, bounding the worst-case hit delay. The last intermediate policy we discuss is *partitioned multicast*, in which the bank set is broken down into subsets of banks, each of which is searched iteratively, but the members of the subset are searched in parallel.

## 5.3 Migration of Data

Since the goal of the dynamic NUCA approach is to maximize the number hits in the closest and fastest banks, a natural policy is to use the replacement statistics of the cache, such as the least-recently-used (LRU) bits, to dictate the location of the lines within the spread set. Thus the most-recently used line is moved to the closest bank, the second most-recently used line moved to the second-closest bank, and so on. However, this approach results in a tremendous amount of traffic and data movement among the banks. On every miss, an  $N$ -way bank set associative cache would require  $N$  lines to be moved, as each line in the set would be migrated to the next lower position in the LRU list and the new line brought into the head of the line. On hits, the number of copies would also be substantial when the hit line was located in less-recently-used block position and was then promoted to the most-recently-used bank. Feasible policies must consider the primary cost of contention in the network when moving among the banks and the secondary cost of power consumption that accompanies the bank accesses for data movement.

We propose using *generational promotion* to reduce the number of copies, while still approximating an LRU list mapped onto the physical topology of a bank set. Gener-

ational replacement was recently proposed by Hallnor *et al.* for making replacement decisions in a software-managed UCA called the Indirect Index Cache [11]. In our scheme, when a hit occurs to a cache line, it is swapped with the line in the bank that is the next-closest to the controller from the bank holding the accessed line. With that policy, heavily used lines will migrate toward close banks, and infrequently used lines will be swapped into the banks further from the controller. We evaluate several heuristics for performing the swap, varying the number of hits required to cause a line to be promoted and the distance (measured in banks) a line moves toward the controller on each promotion. These parameters affect the speed at which line migrate as well as the traffic generated to perform the migration.

A second key question for data migration is the placement of incoming block resulting from a cache miss. If the incoming block is loaded into the set’s closest bank, it may displace an important and frequently accessed block. However, if it is placed at the most distant bank, subsequent accesses to the new block will be slow until it has been accessed enough to be promoted. We evaluate two policies: a zero-copy policy, in which the incoming line replaces the victim in the chosen bank, and a one-copy policy, in which the incoming line forces out a line, which then evicts an even lower-priority line. The one-copy policy is useful for moving misses quickly into the close banks without evicting important lines from the cache altogether.

#### 5.4 D-NUCA Policies

The policies we explore for D-NUCA consist of four components:

- **Mapping:** simple or fair.
- **Search:** multicast, incremental, or combination. We restrict the combined policies such that a block set is partitioned into just two groups. Each group can vary in size (number of blocks) and the method of access (incremental or multicast).
- **Promotion:** described by *promotion distance*, measured in cache banks, and *promotion trigger*, measured in number of hits to a bank before a promotion occurs.
- **Eviction:** identifies the location to place an incoming block and where to put the block it replaces.

Our baseline configuration is the simplest policy in which uses simple mapping, multicast search, one-bank promotion on each hit, and a replacement policy that evicts the least-recently-used block and inserts the incoming block in its place. To examine how this policy compares to pure-LRU, we measured the distribution of accesses across the

sets for an 8-way set associative cache and a corresponding 16MB D-NUCA cache with the aforementioned parameters. Figure 5 shows the distribution of hits to the various sets for each cache, averaged across the benchmark suite. The graph shows that most hits are concentrated in the first two ways of the set for both configurations. The concentration is less pronounced for the D-NUCA runs, simply because the generational promotion policy does not maintain perfect LRU ordering.

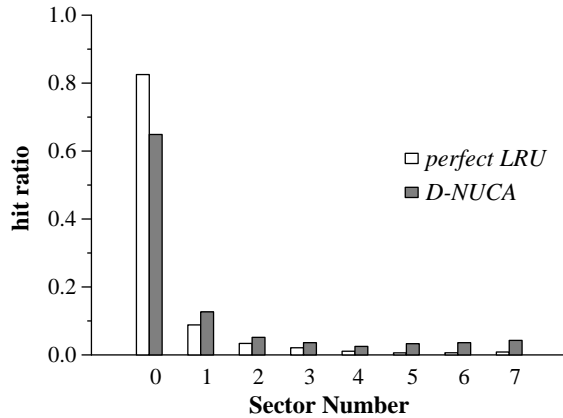


Figure 5. Way distribution of cache hits

#### 5.5 Baseline Performance

Table 5 shows the performance of the baseline D-NUCA configuration across the span of cache sizes and technologies. For each technology (and hence capacity) we chose the bank organization that maximizes performance (IPC). As the capacities and corresponding clock rates increase with the smaller technologies, the average access latency increases by only 20%. Unlike the UCA and bus-based S-NUCA-1 architectures, this D-NUCA strategy achieves larger capacities with only a small increase in average access time, resulting in a monotonic increase in performance across the technologies. For the smaller caches, the D-NUCA organization provides a small (~4%) performance gain over the best of the S-NUCA or UCA organizations. The disparity grows as the cache size increases, with the base 16MB D-NUCA organization showing an average 10% performance boost over the best-performing S-NUCA organization. For the 50nm, 16MB cache, the average loaded access latency is 12.4 cycles. The data migration enables such a low average latency, which, despite the cache’s larger capacity and smaller device sizes, is *less* than the average hit latency for the 180nm, 1MB UCA and S-NUCA-1 cache organizations.

Table 5 also lists miss rates and the total number of accesses to individual cache banks. The number of bank accesses decreases as the cache size grows because the miss

Technology (nm)	L2 size	Bank org. (rows x sets)	Unloaded latency				Loaded avg.	IPC	Miss rate	Bank accesses/set
			bank	min	max	avg.				
180	1MB	4x4	3	4	12	8	10.0	0.50	0.30	124M
130	2MB	8x4	3	4	16	10	10.8	0.55	0.23	110M
100	4MB	4x4	5	5	13	9	11.1	0.62	0.20	95M
70	8MB	8x4	5	5	17	11	11.3	0.69	0.15	85M
50	16MB	8x8	5	5	25	15	12.4	0.77	0.11	147M

**Table 5. D-NUCA base performance**

Policy	Av. lat.	IPC	Miss rate	Bank access	Policy	Av. lat.	IPC	Miss rate	Bank access
Mapping					Promotion				
Fair	11.7	0.78	0.11	147M	1-bank/2-hit	12.5	0.76	0.12	145M
Search					2-bank/1-hit	12.0	0.77	0.11	147M
Incremental	17.9	0.70	0.11	77M	2-bank/2-hit	12.4	0.77	0.12	144M
2 mcast + 6 inc	15.2	0.73	0.11	84M	Eviction				
2 inc + 6 mcast	15.1	0.74	0.11	84M	insert head, evict random	11.5	0.76	0.12	154M
2 mcast + 6 mcast	13.6	0.76	0.11	92M	insert bank 3, evict random	11.6	0.76	0.12	154M
<b>Baseline:</b> simple map, multicast, 1-bank/1-hit, insert at tail						12.4	0.77	0.11	147M
<b>Best:</b> fair map, 2 mcast + 6 mcast, 2-bank/1-hit, insert at tail, evict random						11.5	0.76	0.11	90M

**Table 6. D-NUCA policy space evaluation**

rate decreases and fewer cache fills and evictions are required. However, at 16MB the number of accesses increases suddenly because the multicast policy generates substantially more cache bank accesses when the number of banks in each bank set doubles from 4 to 8.

## 5.6 Policy Exploration

Table 6 shows the effect of adjusting the policies independently on the performance of the baseline configuration. Changing the mapping function from simple to fair results in a 6% reduction in average access time, but only a small (1.5%) improvement in overall performance. Shifting from the baseline multicast to a purely incremental search policy substantially reduces the number of bank accesses (from 147 million to 77 million). However, even though most data is found in one of the first two banks, the incremental policy increases the average access latency from 12.4 cycles to 17.9 cycles and reduces performance by 10%. However, the hybrid policies gain back half of the loss in access latency (15.1 cycles) and nearly all of the performance, while still eliminating most of the extra bank accesses.

The data promotion policy (where to move data on hits) minimizes average latency with fast (2-bank) promotions on each hit. However, the number of bank accesses is lower by 2% if two hits are required before a datum is required, with no noticeable drop in performance.

Finally, the best eviction policy is as shown in the base case, replacing the block in the tail. By replacing the head or 3rd blocks, and copying those into a random, lower-priority

set, the average hit time is reduced from 11.5 to 10.8 and 11.2 cycles, respectively. However, the miss rate increases from 11.2% to 12.0%, offsetting the gains from the reduced hit latency. In addition, the number of bank accesses is increased by 5.6% due to the extra copy required. If no copy is performed, and one of the higher-priority blocks is evicted directly upon a replacement, the miss rate increases by several percent, causing performance to drop significantly.

The experiment labeled *best policy* at the bottom of the table represents a policy using the best-performing options from each of the four policy spaces, with the exception of using the bank access-intensive (but high-performance) pure multicast for lookups. This policy achieves close to the lowest latency access (11.5 cycles) while showing the lowest miss rate of all the policies. While this scheme has 2.5% lower performance than the baseline, the carefully tuned policies permit a 39% reduction in bank accesses over the base policy, from 147 million to 90 million. The “best” policy thus strikes the best compromise between high performance and low energy consumed.

## 5.7 Summary results

Figure 6 summarizes the performance of the best configuration for each of the cache architectures described in this paper: UCA, S-NUCA-1, S-NUCA-2, and D-NUCA, for each cache size. The D-NUCA results correspond to the “best” policy shown previously in this section. Figure 6a shows how these cache organizations perform for a benchmark (*art*) that has a small working set. Figure 6b shows the

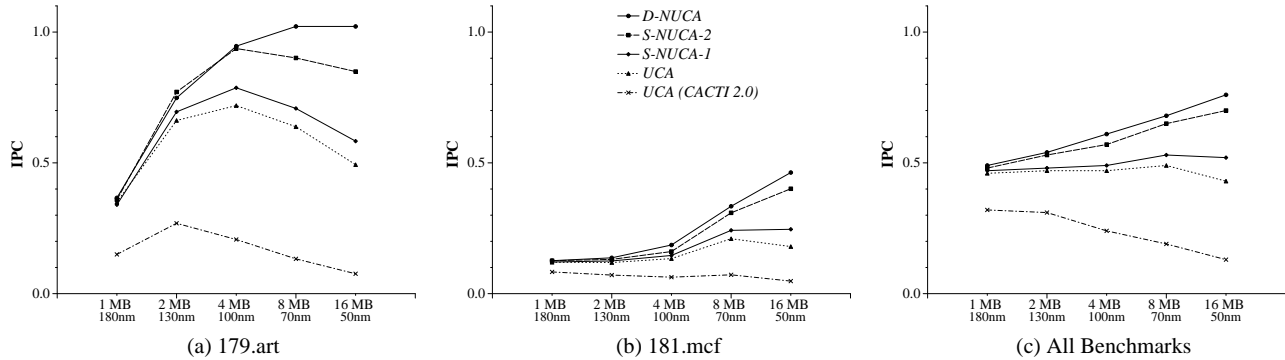


Figure 6. Performance summary of major cache organizations

same information for a benchmark (*mcf*) with a large working set. Figure 6c shows the harmonic mean performance across all benchmarks.

The results clearly show two major points. First, the D-NUCA organization is by far the highest-performing of the five for large (4MB and larger) caches. Its organization is amenable to large caches and the architecture is scalable across technologies and clock rates. The adaptive nature of the D-NUCA architecture permits increased performance with increased capacity, even in the face of longer wire and on-chip communication delays. Second, the D-NUCA organization is *stable*, in that it makes the largest cache size the best performer for 9 applications, within 1% of the best for 5 applications, and within 5% for 2 applications. Figure 6a for *art* shows this disparity most clearly in that D-NUCA is the only organization that continues to increase performance as the cache grows beyond 4MB.

## 6 Related Work

Although this work is the first to evaluate novel designs of large, on-chip caches in future wire-dominated technologies, there has been an enormous volume of literature published on many related caching topics. Researchers have previously studied issues associated with large caches. Kessler examined designs for multi-megabyte caches built with discrete components, and not in a wire-dominated technology [18]. Hallnor and Reinhardt [11] studied a fully associative software-managed design for large, on-chip L2 caches, but not did not consider non-uniform access times in future technologies.

The concept of bank sets permits flexible management of the cache by varying associativity. While our mechanism is new, other work has examined dynamically varying associativity to balance power and performance. Albonesi examines turning off “ways” of each set to save power when cache demand is low [3]. Powell *et al.* evaluate the balance between incremental searches of the sets to balance power and performance, as we do with our multicast versus incre-

mental policies [25], and as Kessler *et al.* did to optimize for speed [21]. Our concept of bank sets does not lend itself well to more creative set mappings to reduce conflicts, such as skewed associativity [5].

Other researchers have examined using multiple banks for high bandwidth, as we do to reduce contention. Sohi and Franklin [31] proposed interleaving banks to create ports, and also examined the need for L2 cache ports on less powerful processors than today’s. Wilson and Olukotun [34] performed an exhaustive study of the trade-offs involved with port and bank replication and line buffers for level-one caches.

Our work may permit a flattening of the cache hierarchy through the adaptive migration of data. Przybylski performed an exhaustive study on tuning multiple levels of the memory hierarchy to maximize performance in his dissertation [26]. With his colleagues [27], he also defined *global miss rates* at a lower-level cache, which was a concept we used in our studies.

Finally, many researchers have examined adaptive cache policies, a concept which is inherent in the D-NUCA organization. Dahlgren *et al.* [6] studied creative ways to avoid conflicts in direct-mapped on-chip caches by virtually binding regions of the address space to portions of the cache, as well as adapting the block size to different workload needs (as did Johnson and Hwu [16]). While the D-NUCA scheme leaves data with low locality in banks far from the processor, an alternative approach is not to cache those lines at all. González, Aliagas, and Valero [9] examined a cache organization that could adaptively avoid caching data with low locality and a *locality detection* scheme to split the cache into temporal and spatial caches. Tyson *et al.* [33] also proposed a scheme to bypass the cache with low-locality data.

## 7 Summary and Conclusions

Non-uniform accesses are just starting to appear in high-performance cache designs [24]. In this paper, we pro-

posed several new organizations for large, on-chip caches, that exploit non-uniform bank accesses, and that are well-matched to future technology constraints. These designs treat the cache as a network of banks and facilitates non-uniform accesses to different physical regions. We have shown that these non-uniform cache access (NUCA) architectures achieve the following three goals:

- *Low latency access*: the best 16MB D-NUCA configuration, simulated with projected 50nm technology parameters, demonstrated an average access time of 12 cycles at a fast 8 FO4 clock. Conventional cache designs produce much higher access latencies, even for smaller caches. With the combination of low hit latency and low miss rate, this cache organization provides significantly higher performance on memory-intensive benchmarks than conventional cache architectures.
- *Technology scalability*: As CMOS devices sizes shrink and global wire delays increase, the access times for traditional cache designs will increase correspondingly. The D-NUCA design scales much better with technology than conventional caches, since most accesses will be to close banks, which can be kept numerous and small due to the switched network. When holding the area of the cache roughly constant and scaling technology, the D-NUCA access times increase only slowly, from 10 cycles at a 1MB, 180nm cache to 12 cycles at a 16MB, 50nm cache.
- *Performance stability*: The ability of D-NUCA to migrate data eliminates the trade-off between larger, slower caches for applications with large working sets and smaller, faster caches for applications that are less memory intensive. Our results show that the 16MB D-NUCA organization achieves the best performance on 9 of the benchmarks, and is within an average of 1.7% on the remainder.

Further research challenges remain for the base NUCA architecture. While our experiments have assumed sufficient buffer space at each cache bank to hold pending requests, we expect that the L2 cache controller can subsume some of this role by efficiently scheduling bank accesses to reduce contention in the cache bank network. A second challenge is the design and analysis of the interface between the cache banks and the off-chip DRAM. It is possible that the inherent adaptivity and scalability of this class of designs will serve to flatten the memory hierarchy by eliminating cache levels beyond the L2. Future designs will have a tiny, fast L1 cache and as much NUCA capacity as will fit on a die. Communication delays will likely make off-chip caches much less attractive and increase the likelihood

of dedicated DRAM channels distributed around the chip, perhaps as logical terminal nodes in the cache network.

Emerging chip multiprocessors (CMP) architectures will likely benefit from the flexibility and scalability of NUCA memory systems. A natural organization places multiple processors and an array of cache banks on a single die. As the workload changes, NUCA cache banks can be dynamically partitioned and reallocated to different processors. Maintaining coherence among different logical caches presents a different set of challenges for NUCA architectures. A variant of the partial tag compare scheme of Kessler *et al.* [21] may permit fast discovery of shared blocks without necessitating slow, huge centralized tag banks.

## References

- [1] 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.
- [2] V. Agarwal, S. W. Keckler, and D. Burger. Scaling of microarchitectural structures in future process technologies. Technical Report TR2000-02, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, February 2000.
- [3] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd International Symposium on Microarchitecture*, pages 248–259, Dec. 1999.
- [4] D. Bailey, J. Barton, T. Lasinski, and H. Simon. The nas parallel benchmarks. Technical Report RNR-91-002 Revision 2, NASA Ames Research Laboratory, Ames, CA, Aug. 1991.
- [5] F. Bodin and A. Seznec. Skewed associativity enhances performance predictability. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 265–274, June 1995.
- [6] F. Dahlgren and P. Stenström. On reconfigurable on-chip data caches. In *Proceedings of the 24th International Symposium on Microarchitecture*, pages 189–198, Nov. 1991.
- [7] R. Desikan, D. Burger, S. W. Keckler, and T. M. Austin. Sim-alpha: a validated execution driven alpha 21264 simulator. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.
- [8] K. Diefendorff. Power4 focuses on memory bandwidth. *Microprocessor Report*, 13(13), Oct. 1999.
- [9] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *Proceedings of the 1995 International Conference on Supercomputing*, pages 338–347, July 1995.
- [10] L. Gwennap. Alpha 21364 to ease memory bottleneck. *Microprocessor Report*, 12(14), Oct. 1998.
- [11] E. Hallnor and S. Reinhardt. A fully associative software-managed cache design. In *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

- [12] J. M. Hill and J. Lachman. A 900MHz 2.25 MB cache with on-chip CPU now in Cu SOI. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 171–177, February 2001.
- [13] M. Horowitz, R. Ho, and K. Mai. The future of wires. In *Semiconductor Research Corporation Workshop on Interconnects for Systems on a Chip*, May 1999.
- [14] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *The 10th International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, September 2001.
- [15] J. Rubinstein, P. Penfield, and M.A. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design*, CAD-2(3):202–211, 1983.
- [16] T. Johnson and W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 315–326, June 1997.
- [17] N. Jouppi and S. Wilton. An enhanced access and cycle time model for on-chip caches. Technical Report TR-93-5, Compaq WRL, July 1994.
- [18] R. Kessler. *Analysis of Multi-Megabyte Secondary CPU Cache Memories*. PhD thesis, University of Wisconsin-Madison, December 1989.
- [19] R. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.
- [20] R. Kessler, M. Hill, and D. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, June 1994.
- [21] R. Kessler, R. Jooss, A. Lebeck, and M. Hill. Inexpensive implementations of set-associativity. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 131–139, May 1989.
- [22] K.-F. Lee, H.-W. Hon, and R. Reddy. An overview of the SPHINX speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38:35–44, 1990.
- [23] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.
- [24] H. Pilo, A. Allen, J. Covino, P. Hansen, S. Lamphier, C. Murphy, T. Traver, and P. Yee. An 833MHz 1.5w 18Mb CMOS SRAM with 1.67Gb/s/pin. In *Proceedings of the 2000 IEEE International Solid-State Circuits Conference*, pages 266–267, February 2000.
- [25] M. D. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the 34th International Symposium on Microarchitecture*, December 2001.
- [26] S. Przybylski. *Performance-Directed Memory Hierarchy Design*. PhD thesis, Stanford University, September 1988. Technical report CSL-TR-88-366.
- [27] S. Przybylski, M. Horowitz, and J. Hennessy. Characteristics of performance-optimal multi-level cache hierarchies. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 114–121, 1989.
- [28] The national technology roadmap for semiconductors. Semiconductor Industry Association, 1999.
- [29] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 International Symposium on Parallel Architectures and Compilation Techniques*, September 2001.
- [30] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical report, Compaq Computer Corporation, August 2001.
- [31] G. S. Sohi and M. Franklin. High-performance data memory systems for superscalar processors. In *Proceedings of the Fourth Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 53–62, Apr. 1991.
- [32] Standard Performance Evaluation Corporation. *SPEC Newsletter*, Fairfax, VA, Sept. 2000.
- [33] G. Tyson, M. Farrens, J. Matthews, and A. Pleszkun. A modified approach to data cache management. In *Proceedings of the 28th International Symposium on Microarchitecture*, pages 93–103, Dec. 1995.
- [34] K. M. Wilson and K. Olukotun. Designing high bandwidth on-chip caches. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.
- [35] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. In *IEEE Journal of Solid-State Circuits*, May 1996.