An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches

Changkyu Kim Doug Burger Stephen W. Keckler

Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin

cartel@cs.utexas.edu - www.cs.utexas.edu/users/cart

ABSTRACT

Growing wire delays will force substantive changes in the designs of large caches. Traditional cache architectures assume that each level in the cache hierarchy has a single, uniform access time. Increases in on-chip communication delays will make the hit time of large on-chip caches a function of a line's physical location within the cache. Consequently, cache access times will become a continuum of latencies rather than a single discrete latency. This non-uniformity can be exploited to provide faster access to cache lines in the portions of the cache that reside closer to the processor. In this paper, we evaluate a series of cache designs that provides fast hits to multi-megabyte cache memories. We first propose physical designs for these Non-Uniform Cache Architectures (NUCAs). We extend these physical designs with logical policies that allow important data to migrate toward the processor within *the same level of the cache. We show that, for multi-megabyte level-two caches, an adaptive, dynamic NUCA design achieves 1.5 times the IPC of a Uniform Cache Architecture of any size, outperforms the best static NUCA scheme by 11%, outperforms the best three-level hierarchy–while using less silicon area–by 13%, and comes within 13% of an ideal, minimal hit latency solution.*

1. INTRODUCTION

Today's high performance processors incorporate large level-two (L2) caches on the processor die. The Alpha 21364 [8] will contain a 1.75MB L2 cache, the HP PA-8700 will contain 2.25MB of unified on-chip cache [10], and the Intel Itanium2 will contain 3MB of on-chip L3 cache. The sizes of on-chip L2 and L3 cache memories are expected to continue increasing as the bandwidth demands on the package grow, and as smaller technologies permit more bits per mm^2 [13].

Current multi-level cache hierarchies are organized into a few discrete levels. Typically, each level obeys inclusion, replicating the contents of the smaller level above it, and reducing accesses to the lower levels of the cache hierarchy. When choosing the size of each level, designers must balance access time and capacity, while staying within area and cost budgets.

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 [1, 22]. 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, the closest bank in a 16-megabyte, on-chip L2 cache built in a 50-nanometer process technology could be accessed in 4 cycles, while an access to the farthest bank might take 47 cycles. The bulk of the access time will involve routing to and from the banks, not the bank accesses themselves.

In this paper, we explore the design space for large, wire-delay-dominated caches. We first show that traditional cache designs, in which a centralized decoder drives physically partitioned sub-banks, will be ineffective in future technologies, as data in those designs can be accessed only as fast as the slowest sub-bank. We propose designs in which the cache is broken into banks that can be accessed at different latencies. Our exploration of that design space addresses the three following questions:

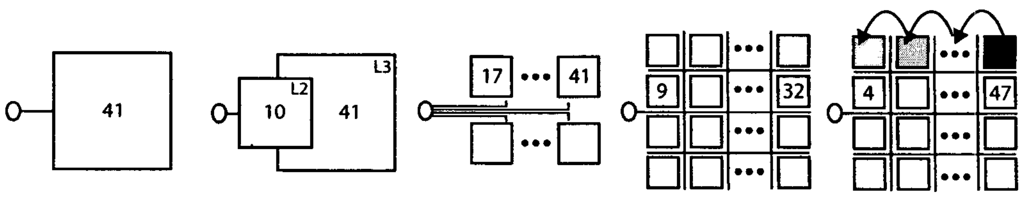
- *Mapping*: How many addressable banks should future caches contain, and how should lines be mapped into those banks?
- *Search*: What is the right strategy for searching the set of possible locations for a line?
- *Movement*: Should a line always be placed in the same bank? If not, how is a line moved, either while resident in the cache or across different lifetimes in the cache?

Figure 1 shows the types of organizations that we explore in this paper, listing the number of banks and the average access times, assuming 16MB caches modeled with a 50nm technology. The numbers superimposed on the cache banks show the latency of a single contentionless request, derived from a modified version of the Cacti cache modeling tool. The average loaded access times shown below are derived from performance simulations that use the unloaded latency as the access time but which include port and channel contention.

We call a traditional cache a Uniform Cache Architecture (UCA), shown in Figure 1a. Although we support aggressive sub-banking, our models indicate that this cache would perform poorly due to internal wire delays and restricted numbers of ports.

Figure 1b shows a traditional multi-level cache (L2 and L3), called ML-UCA. Both levels are aggressively banked for supporting multiple parallel accesses, although the banks are not shown in the figure. Inclusion is enforced, so a line in the smaller level implies two copies in the cache, consuming extra space.

ASPLOS X, 10/02, San Jose, CA, USA.
Copyright 2002 ACM ISBN 1-58113-574-2-02/0010 ...$5.00



	(a) UCA	(b) ML-UCA	(c) S-NUCA-1	(d) S-NUCA-2	(e) D-NUCA
Number of banks:	1 bank	8/32 banks	32 banks	32 banks	256 banks
Avg. loaded access time:	*255 cycles*	*11/41 cycles*	*34 cycles*	*24 cycles*	*18 cycles*

Figure 1: Level-2 Cache Architectures.

Figure 1c shows an aggressively banked cache, which supports non-uniform access to the different banks without the inclusion overhead of ML-UCA. The mapping of data into banks is predetermined, based on the block index, and thus can reside in only one bank of the cache. Each bank uses a private, two-way, pipelined transmission channel to service requests. We call this statically mapped, non-uniform cache S-NUCA-1.

When the delay to route a signal across a cache is significant, increasing the number of banks can improve performance. A large bank can be subdivided into smaller banks, some of which will be closer to the cache controller, and hence faster than those farther from the cache controller. The original, larger bank was necessarily accessed at the speed of the farthest, and hence slowest, sub-bank. Increasing the number of banks, however, can increase wire and decoder area overhead. Private per-bank channels, used in S-NUCA-1, heavily restrict the number of banks that can be implemented, since the per-bank channel wires adds significant area overhead to the cache if the number of banks is large. To circumvent that limitation, we propose a static NUCA design that uses a two-dimensional switched network instead of private per-bank channels, permitting a larger number of smaller, faster banks. This organization, called S-NUCA-2, is shown in Figure 1d.

Even with an aggressive multi-banked design, performance may still be improved by exploiting the fact that accessing closer banks is faster than accessing farther banks. By permitting data to be mapped to many banks within the cache, and to migrate among them, a cache can be automatically managed in such a way that most requests are serviced by the fastest banks. Using the switched network, data can be gradually promoted to faster banks as they are frequently used. This promotion is enabled by spreading sets across multiple banks, where each bank forms one way of a set. Thus, cache lines in closer ways can be accessed faster than lines in farther ways.

In this paper, we compare this dynamic non-uniform scheme (D-NUCA), depicted in Figure 1e, to the S-NUCA schemes and ML-UCA. A D-NUCA organization incurs fewer misses than an inclusive ML-UCA design using the same area, since D-NUCA does not enforce inclusion, preventing redundant copies of the same line. We show that a D-NUCA cache achieves highest IPC across diverse applications, by adapting to the working set of each application and moving it into the banks closest to the processor. In an ML-UCA organization, conversely, the faster level may not match the working set size of an application, either being too large (and thus too slow), or being too small, incurring unnecessary misses to the slower backing level. For large caches in wire-delay-dominated technologies (16MB at 50nm), the best D-NUCA organization we explore outperforms the best S-NUCA organization by 18% and the best ML-UCA organization by 20%.

The remainder of this paper is organized as follows. Section 2 describes the limitations of single-banked caches, and describes our per-bank evaluation methodology. Section 3 describes and evaluates the S-NUCA-1 and S-NUCA-2 designs, as well as details our modeling of contention and wire pipelining. Section 4 presents a D-NUCA, in which flexible mapping policies dynamically migrate important data to nearer, faster banks along a switched network. In this section, we also compare the D-NUCA design with a number of ML-UCA hierarchies. We briefly discuss the effect of changing technology projections in Section 6. We discuss related work in Section 7, and conclude in Section 8.

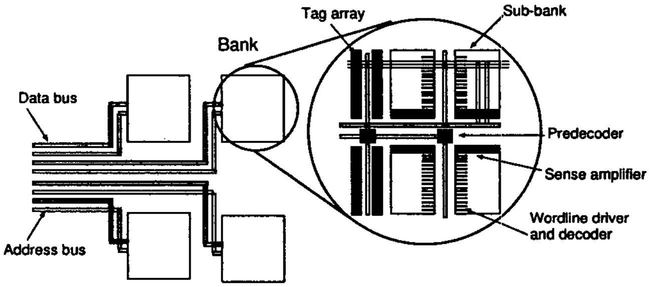


Figure 2: UCA and S-NUCA-1 cache design

2. UNIFORM ACCESS CACHES

Large modern caches are subdivided into multiple sub-banks to minimize access time. Cache modeling tools, such as Cacti [16, 33], enable fast exploration of the cache design space by automatically choosing the optimal sub-bank count, size, and orientation. To estimate the cache bank delay, we used Cacti 3.0, which accounts for capacity, sub-bank organization, area, and process technology [27].

Figure 2 contains an example of a Cacti-style bank, shown in the circular expanded section of one bank. The cache is modeled assuming a central pre-decoder, which drives signals to the local decoders in the sub-banks. Data are accessed at each sub-bank and returned to the output drivers after passing through muxes, where the requested line is assembled and driven to the cache controller. Cacti uses an exhaustive search to choose the number and shape of sub-banks to minimize access time. Despite the use of an optimal sub-banking organization, large caches of this type perform poorly in a wire-delay-dominated process, since the delay to receive the portion of a line from the slowest of the sub-banks is large.

2.1 Experimental Methodology

To evaluate the effects of different cache organizations on system performance, we used Cacti to derive the access times for

	Phase		L2 load accesses/		Phase		L2 load accesses/
SPECINT2000	FFWD	RUN	Million instr	SPECFP2000	FFWD	RUN	Million instr
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.perlbmk	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.equake	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

caches, and extended the `sim-alpha` simulator [6] to simulate different cache organizations with parameters derived from Cacti. `sim-alpha` models an Alpha 21264 core in detail [18]. We assumed that all microarchitectural parameters other than the L2 organization match those of the 21264, including issue width, fetch bandwidth, and clustering. The L1 caches we simulated are similar to those of the 21264: 3-cycle access to the 64KB, 2-way set associative L1 data cache, and single-cycle access to the similarly configured L1 I-cache. All line sizes in this study were fixed at 64 bytes. In all cache experiments, we assumed that the off-chip memory controller resides near the L2 memory controller. Thus, writebacks need to be pulled out of the cache, and demand misses, when the pertinent line arrives, are injected into the cache by the L2 controller, with all contention modeled as necessary. However, we do not model any routing latency from the off-chip memory controller to the L2 cache controller.

Since Cacti produces timing estimates in nanoseconds, we converted cache delays to processor cycles by assuming an aggressive clock of 8 FO4 inverter delays[1] per cycle for each technology. Recent work has shown that 8 FO4 delays is close to the optimal clock for superscalar microprocessors [12]. We assume an unloaded 132-cycle access to main memory, obtained by scaling the memory latency of an actual Alpha 21264 system–a DS-10L workstation at 66 cycles–by a factor equal to the ratio of the assumed and actual clock rates. Since the 21264 has approximately twice as many gate delays per cycle (16–20 FO4 versus 8 FO4), we multiplied the DS-10L 66 cycle memory latency by two.

Table 1 shows the benchmarks used in our experiments, chosen for their high L1 miss rates. The 16 applications include six SPEC2000 floating-point benchmarks [30], six SPEC2000 integer benchmarks, three scientific applications from the NAS suite [3], and Sphinx, a speech recognition application [21]. For each benchmark we simulated the sequence of instructions which capture the core repetitive phase of the program, 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. Table 1 lists the number of instructions skipped to reach the phase start (FFWD) and the number of instructions simulated (RUN). Table 1 also shows the anticipated L2 load, listing the number of L2 accesses per 1 million instructions assuming 64KB level-1 instruction and data caches (this metric was proposed by Kessler *et al.* [19]).

2.2 UCA Evaluation

Tech (nm)	L2 Capacity	Num. Sub-banks	Unloaded Latency	Loaded Latency	IPC	Miss Rate
130	2MB	16	13	67.7	0.41	0.23
100	4MB	16	18	91.1	0.39	0.20
70	8MB	32	26	144.2	0.34	0.17
50	16MB	32	41	255.1	0.26	0.13

Table 2: Performance of UCA organizations

Table 2 shows the parameters and achieved instructions per cycle (IPC) of the UCA organization. For the rest of this paper, we assume a constant L2 cache area and vary the technology generation to scale cache capacity within that area, using the SIA Roadmap [26] predictions, from 2MB of on-chip L2 at 130 nm devices to 16MB at 50 nm devices. In Table 2, the unloaded latency is the average access time (in cycles) assuming uniform bank access distribution and no contention. The loaded latency is obtained by averaging the actual L2 cache access time–including contention–across all of the benchmarks. Contention can include both *bank contention*, when a request must stall because the needed bank is busy servicing a different request, and *channel contention*, when the bank is free but the routing path to the bank is busy, delaying a request.

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; we varied the number and aspect ratio of sub-banks exhaustively, as well as the number of banks.

In the UCA cache, the unloaded access latencies are sufficiently high that contention could be a serious problem. Multiported cells are a poor solution for overlapping accesses in large caches, as increases in area will expand loaded access times significantly: for a 2-ported, 16MB cache at 50nm, Cacti reports a significant increase in the unloaded latency, which makes a 2-ported solution perform worse than a single-ported L2 cache. Instead of multiple physical ports per cell, we assume perfect pipelining: that all routing and logic have latches, and that a new request could be initiated at an interval determined by the maximal sub-bank delay, which is shown in column 4 of Table 2. We did not model the area or delay consumed by the pipeline latches, resulting in optimistic performance projections for an UCA organization.

Table 2 shows that, despite the aggressive cache pipelining, the loaded latency grows significantly as the cache size increases, from 68 cycles at 2MB to 255 cycles at 16MB. The best overall cache size is 2MB, at which the increases in L2 latency are subsumed by the improvement in miss rates. For larger caches, the latency increases overwhelm the continued reduction in L2 misses. While the UCA organization is inappropriate for large, wire-dominated caches, it serves as a baseline for measuring the performance improvement of more sophisticated cache organizations, described in the following section.

[1] One FO4 is the delay of one inverter driving four copies of itself. Delays measured in FO4 are independent of technology; we model one FO4 as 360 picoseconds times the transistor's effective gate length in microns [11].

Technology	L2	Num.	Unloaded latency				Conservative		Aggressive	
(nm)	size	banks	bank	min	max	avg.	Loaded	IPC	Loaded	IPC
130	2MB	16	3	7	13	10	11.3	0.54	10.0	0.55
100	4MB	32	3	9	21	15	17.3	0.56	15.3	0.57
70	8MB	32	5	12	26	19	21.9	0.61	19.3	0.63
50	16MB	32	8	17	41	29	34.2	0.59	30.2	0.62

Table 3: S-NUCA-1 evaluation

3. STATIC NUCA IMPLEMENTATIONS

Much performance is lost by requiring worst-case uniform access in a wire-delay dominated cache. Multiple banks can mitigate those losses, if each bank can be accessed at different speeds, proportional to the distance of the bank from the cache controller. In our banked cache models, each bank is independently addressable, and is sized and partitioned into a locally optimal physical sub-bank organization. As before, the number and physical organization of banks and sub-banks were chosen to maximize overall IPC, after an exhaustive exploration of the design space.

Data are statically mapped into banks, with the low-order bits of the index determining the bank. Each bank we simulate is four-way set associative. These static, non-uniform cache architectures (S-NUCA) have two advantages over the UCA organization previously described. First, accesses to banks closer to the cache controller incur lower latency. Second, accesses to different banks may proceed in parallel, reducing contention. We call these caches S-NUCA caches, since the mappings of data to banks are static, and the banks have non-uniform access times.

3.1 Private Channels (S-NUCA-1)

As shown in Figure 2, each addressable bank in the S-NUCA-1 organization has two private, per-bank 128-bit channels, one going in each direction. Cacti 3.0 is not suited for modeling these long transmission channels, since it uses the Rubenstein RC wire delay model [14] and assumes bit-line capacitative loading on each wire. We replaced that model with the more aggressive repeater and scaled wire model of Agarwal *et al.* for the long address and data busses to and from the banks [1].

Since banks have private channels, each bank can be accessed independently at its maximum speed. While smaller banks would provide more concurrency and a greater fidelity of non-uniform access, the numerous per-bank channels add area overhead to the array that constrains the number of banks.

When a bank conflict occurs, we model contention in two ways. A *conservative* policy assumes a simple scheduler that does not place a request on a bank channel until the previous request to that bank has completed. Bank requests may thus be initiated every $b + 2d + 3$ cycles, where b is the actual bank access time, d is the one-way transmission time on a bank's channel, and the additional 3 cycles are needed to drain the additional data packets on the channel in the case of a read request following a writeback. Since each channel is 16 bytes, and the L2 cache line size is 64 bytes, it takes 4 cycles to remove a cache line from the channel.

An *aggressive* pipelining policy assumes that a request to a bank may be initiated every $b + 3$ cycles, where b is the access latency of the bank itself. This channel model is optimistic, as we do not model the delay or area overhead of the latches necessary to have multiple requests in flight on a channel at once, although we do model the delay of the wire repeaters.

Table 3 shows a breakdown of the access delays for the various cache sizes and technology points: the number of banks to which independent requests can be sent simultaneously, the raw bank access delay, the minimum, average, and maximum access latency of a single request to various banks, 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. Unlike UCA, the average IPC increases as the cache sizes increases, until 8 MB. At 16MB, the large area taken by the cache causes the hit latencies to overwhelm the reduced misses, even though the access latencies grow more slowly than with an UCA organization.

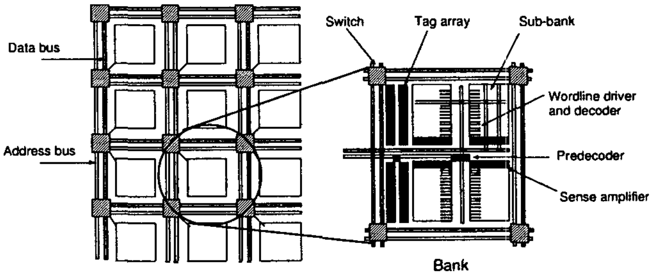


Figure 3: Switched NUCA design

As technology advances, both the access time of individual banks and the routing delay to the farthest banks increase. The bank access times for S-NUCA-1 increase from 3 cycles at 100nm to 8 cycles at 50 nm because the best organization at smaller technologies uses larger banks. The overhead of the larger, slower banks is less than the delays that would be caused by the extra wires required for more numerous, smaller banks.

The greater wire delays at small technologies cause increased routing delays to the farther banks. At 130nm, the worst-case routing delay is 10 cycles. It increases steadily to reach 33 cycles at 50nm. While raw routing delays in the cache are significant, contention is less of a problem. Contention for banks and channels can be measured by subtracting the average loaded latency from the average unloaded latency in Table 3. The aggressive pipelining of the request transmission on the channels eliminates from 1.3 to 4.0 cycles from the conservative pipelining average loaded bank access latency, resulting in a 5% improvement in IPC at 16MB.

The number of banks increases from 16 at 2MB to 32 at 4MB. At 8MB and 16MB, the optimal number of banks does not increase further, due to the area overhead of the per-bank channels, so each bank grows larger and slower as the cache size increases. That constraint prevents the S-NUCA-1 organization from exploiting the potential access fidelity of small, fast banks. In the next subsection, we describe a inter-bank network that mitigates the per-bank channel area constraint.

3.2 Switched Channels (S-NUCA-2)

Figure 3 shows an organization that removes most of the large number of wires resulting from per-bank channels. This organization, called S-NUCA-2, embeds a lightweight, wormhole-routed 2-D mesh with point-to-point links in the cache, placing simple switches at each bank. Each link has two separate 128-bit channels for bidirectional routing. We modeled the switch logic in HSPICE

Technology	L2	Num.	Unloaded Latency				Loaded		Bank
(nm)	Size	Banks	bank	min	max	avg.	Latency	IPC	Requests
130	2MB	16	3	4	11	8	9.7	0.55	17M
100	4MB	32	3	4	15	10	11.9	0.58	16M
70	8MB	32	5	6	29	18	20.6	0.62	15M
50	16MB	32	8	9	32	21	24.2	0.65	15M

Table 4: S-NUCA-2 performance

to obtain the delay for each switch and incorporate that delay into our performance simulations. We again used the Agarwal *et al.* model for measuring wire delay between switches. As in the previous configurations, we assume 4-way set associative banks.

We modeled contention by implementing wormhole-routed flow control, and by simulating the mesh itself and the individual switch occupancy in detail as a part of our performance simulations. In our simulations, each switch buffers 16-byte packets, and each bank contains a larger buffer to hold an entire pending request. Thus, exactly one request can be queued at a specific bank while another is being serviced. A third arrival would block the network links, buffering the third request in the network switches and delaying other requests requiring those switches. Other banks along different network paths could still be accessed in parallel, of course.

In the highest-performing bank organization presented, each bank was sized so that the routing delay along one bank was just under one cycle. We simulated switches that had buffer slots for four flits per channel, since our sensitivity analysis showed that more than four slots per switch gained little additional IPC. In our 16MB S-NUCA-2 simulations, the cache incurred an average of 0.8 cycles of bank contention and 0.7 cycles of link contention in the network.

Table 4 shows the IPC of the S-NUCA-2 design. For 4MB and larger caches, the minimum, average, and maximum bank latencies are significantly smaller than those for S-NUCA-1. The switched network speeds up cache accesses because it consumes less area than the private, per-bank channels, resulting in a smaller array and faster access to all banks. At 50nm with 32 banks, our models indicate that the S-NUCA-1 organization's wires consume 20.9% of the bank area, whereas the S-NUCA-2 channel overhead is just 5.9% the total area of the banks.

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 24.2 cycles, as opposed to 34.2 cycles for S-NUCA-1. At 16MB, that reduction in latency results in a 10% average improvement in IPC across the benchmark suite. An additional benefit from the reduced per-bank wire overhead is that larger numbers of banks are possible and desirable, as we show in the following section.

4. DYNAMIC NUCA IMPLEMENTATIONS

In this section, we show how to exploit future cache access nonuniformity by placing frequently accessed data in closer (faster) banks and less important–yet still cached–data in farther banks. We evaluate a number of hardware policies that migrate data among the banks to reduce average L2 cache access time and improve overall performance. For these policies, we answer three important questions about the management of data in the cache: (1) *mapping*: how the data are mapped to the banks, and in which banks a datum can reside, (2) *search*: how the set of possible locations are searched to find a line, (3) *movement*: under what conditions the data should be migrated from one bank to another. We explore these questions in each of the following three subsections.

4.1 Logical to Physical Cache Mapping

A large number of banks provides substantial flexibility for mapping lines to banks. 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 the latter approach maximizes 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 an intermediate solution called *spread sets* in which the multibanked 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 corresponds to the associativity.

A cache can be comprised of multiple bank sets. For example, as shown in Figure 4a, 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 *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, 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 rows 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; some bank sets will be faster than others, since some rows are closer to the cache controller than others.

Figure 4b shows the *fair mapping* policy, which addresses both problems of the simple mapping policy at the cost of 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 sets are equalized. We do not present results for this policy, but describe it for completeness. The advantage of the fair mapping policy is an approximately equal average access time for each bank set. The disadvantage is a more complex routing path from bank to bank within a set, causing potentially longer routing latencies and more contention in the network.

The *shared mapping* policy, shown in Figure 4c, attempts to provide fastest-bank access to all bank sets by sharing the closest banks among multiple bank sets. This policy requires that if n bank sets share a single bank, then all banks in the cache are n-way set associative. Otherwise, a swap from a solely owned bank into a shared bank could result in a line that cannot be placed into the solely owned bank, since the shared bank has fewer sets than the non-shared bank. In this study, we allow a maximum of two bank sets to share a bank. Each of the $n/2$ farthest bank sets shares half of the closest bank for one of the closest $n/2$ bank sets. This policy

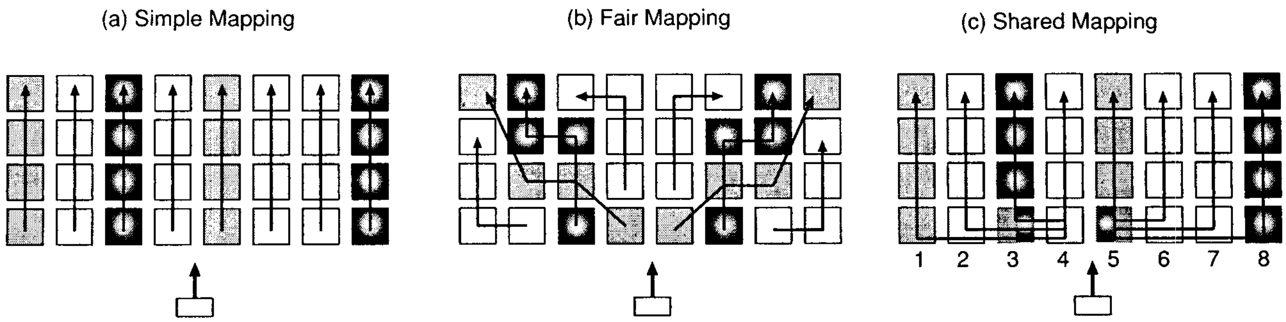


Figure 4: Mapping bank sets to banks.

results in some bank sets having a slightly higher bank associativity than the others, which can offset the slightly increased average access latency to that bank set. That strategy is illustrated in Figure 4c, in which the bottom bank of column 3 caches lines from columns 1 and 3, the bottom bank of column 4 caches lines from columns 2 and 4, and so on. In this example the farthest four (1, 2, 7, and 8) of the eight bank sets share the closest banks of the closest four (3, 4, 5, and 6).

4.2 Locating Cached Lines

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 requested line is found or a miss occurs in the last bank. 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, at the cost of reduced performance.

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 actual 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. One potential performance drawback to multicast search is that the extra address bandwidth consumed as the address is routed to each bank may slow other accesses.

Hybrid intermediate policies are possible, such as *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. Another hybrid policy is *partitioned multicast,* in which the bank set is broken down into subsets of banks. Each subset is searched iteratively, but the members of each subset are searched in parallel, similar to a multi-level, set-associative cache.

4.3 Smart Search

A distributed cache array, in which the tags are distributed with the banks, creates two new challenges. First, many banks may need to be searched to find a line on a cache hit. Second, if the line is not in the cache, the slowest bank determines the time necessary to resolve that the request is a miss. The miss resolution time thus grows as the number of banks in the bank set increases. While the incremental search policy can reduce the number of bank lookups, the serialized tag lookup time increases both the hit latency and the miss resolution time.

We applied the idea of the *partial tag comparison* proposed by Kessler *et al.*[20] to reduce both the number of bank lookups and the miss resolution time. The D-NUCA policy using partial tag comparisons, which we call *smart search,* stores the partial tag bits into a smart search array located in the cache controller.

We evaluated two smart search policies: *ss-performance* and *ss-energy*. In the *ss-performance* policy, the cache array is searched as in previous policies. However, in parallel, the stored partial tag bits are compared with the corresponding bits of the requested tag, and if no matches occur, the miss processing is commenced early. In this policy, the smart search array must contain enough of the tag bits per line to make the possibility of *false hits* low, so that upon a miss, accidental partial matches of cached tags to the requested tag are infrequent. We typically cached 6 bits from each tag, balancing the probability of incurring a false hit with the access latency to the smart search array.

In the *ss-energy* policy, the partial tag comparison is used to reduce the number of banks that are searched upon a miss. Since the smart search array takes multiple cycles (typically 4 to 6) to access, serializing the smart search array access before any cache access would significantly reduce performance. As an optimization, we allowed the access of the closest bank to proceed in parallel with the smart search array access. After that access, if a hit in the closest bank does not occur, all other banks for which the partial tag comparison was successful are searched in parallel.

4.4 Dynamic Movement of Lines

Since the goal of the dynamic NUCA approach is to maximize the number of hits in the closest banks, a desirable policy would be to use LRU ordering to order the lines in the bank sets, with the closest bank holding the MRU line, second closest holding second most-recently used, etc. The problem with that approach is that most accesses would result in heavy movement of lines among banks. In a traditional cache, the LRU state bits are adjusted to reflect the access history of the lines, but the tags and data of the lines are not moved. In an n-way spread set, however, an access to the LRU line would result in n copy operations. Practical policies must balance the increased contention and power consumption of copying with the benefits expected from bank set ordering.

We use *generational promotion* to reduce the amount of copying required by a pure LRU mapping, while still approximating an LRU list mapped onto the physical topology of a bank set. Generational replacement was recently proposed by Hallnor *et al.* for making replacement decisions in a software-managed UCA called the Indirect Index Cache [9]. 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 cache controller. Heavily used lines will thus migrate toward close, fast banks, whereas infrequently used lines will be demoted into farther, slower banks.

A D-NUCA policy must determine the placement of an incoming block resulting from a cache miss. A replacement may be loaded

Technology		Bank org.	Unloaded Latency				Loaded		Miss	Bank
(nm)	L2 Size	(rows x sets)	Bank	min	max	avg.	avg.	IPC	Rate	Accesses/Set
130	2MB	4x4	3	4	11	8	8.4	0.57	0.23	73M
100	4MB	8x4	3	4	15	10	10.0	0.63	0.19	72M
70	8MB	16x8	3	4	31	18	15.2	0.67	0.15	138M
50	16MB	16x16	3	3	47	25	18.3	0.71	0.11	266M

Table 5: D-NUCA base performance

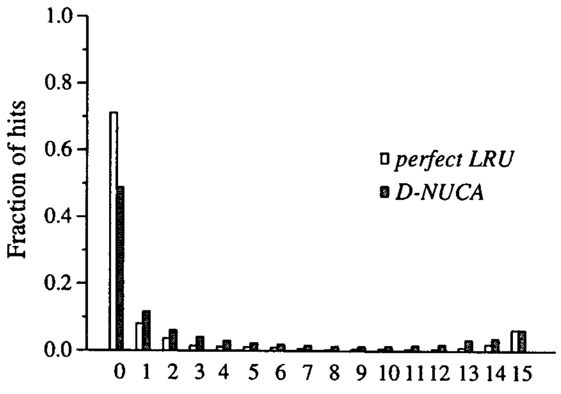


Figure 5: Way distribution of cache hits

close to the processor, displacing an important block. The replacement may be loaded in a distant bank, in which case an important block would require several accesses before it is eventually migrated to the fastest banks. Another policy decision involves what to do with a victim upon a replacement; the two polices we evaluated were one in which the victim is evicted from the cache (a *zero-copy* policy), and one in which the victim is moved to a lower-priority bank, replacing a less important line farther from the controller (*one-copy* policy).

4.5 D-NUCA Policies

The policies we explore for D-NUCA consist of four major components: (1) **Mapping:** simple or shared. (2) **Search:** multicast, incremental, or combination. We restrict the combined policies such that a block set is partitioned into just two groups, which may then each vary in size (number of blocks) and the method of access (incremental or multicast). (3) **Promotion:** described by *promotion distance*, measured in cache banks, and *promotion trigger*, measured in number of hits to a bank before a promotion occurs. (4) **Insertion:** identifies the location to place an incoming block and what to do with the block it replaces (*zero copy* or *one copy* policies).

Our simple, baseline configuration uses simple mapping, multicast search, one-bank promotion on each hit, and a replacement policy that chooses the block in the slowest bank as the victim upon a miss. To examine how effectively this replacement policy compares to pure LRU, we measured the distribution of accesses across the sets for a traditional 16-way set associative cache and a corresponding 16MB, D-NUCA cache with an 16-way bank set. Figure 5 shows the distribution of hits to the various sets for each cache, averaged across the benchmark suite. For both caches, most hits are concentrated in the first two ways of each set. These results are consistent with the results originally shown by So and Rechtschaffen [28], which showed that more than 90% of cache hits were to the most recently used ways in a four-way set associative cache. So and Rechtschaffen noted that a transient increase in non-MRU accesses could be used to mark phase transitions, in which a new working set was being loaded.

The D-NUCA accesses are still concentrated in the banks corresponding to the most recently used bank. However, the experiments demonstrate a larger number of accesses to the non-MRU ways, since each line must gradually traverse the spread set to reach the fastest bank, instead of being instantly loaded into the MRU position, as in a conventional cache.

5. PERFORMANCE EVALUATION

Table 5 shows the performance of the baseline D-NUCA configuration, which uses the simple mapping, multicast search, tail insertion, and single-bank promotion upon each hit. As with all other experiments, for each capacity, we chose the bank and network organization that maximized overall performance. Since our shared mapping policy requires 2-way associative banks, we modeled all banks in each experiment as being 2-way set associative.

As the capacities increase with the smaller technologies, from 2MB to 16MB, the average D-NUCA access latency increases by 10 cycles, from 8.4 to 18.3. The ML-UCA and S-NUCA designs incur higher average latencies at 16MB, which are 22.3 and 30.4 cycles, respectively. Data migration enables the low average latency at 16MB, which, despite the cache's larger capacity and smaller device sizes, is *less* than the average hit latency for the 130nm, 2MB UCA organization.

At smaller capacities such as 2MB, the base D-NUCA policy shows small (~4%) IPC gains over the best of the S-NUCA and UCA organizations. The disparity grows as the cache size increases, with the base 16MB D-NUCA organization showing an average 9% IPC boost over the best-performing S-NUCA organization.

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 rate decreases and fewer cache fills and evictions are required. However, at 8MB and 16MB the number of bank accesses increase significantly because the multicast policy generates substantially more cache bank accesses when the number of banks in each bank set doubles from 4 to 8 at 8MB, and again from 8 to 16 at 16MB. Incremental search policies reduce the number of bank accesses at the cost of occasional added hit latency and slightly reduced IPC.

5.1 Policy Exploration

Table 6 shows the IPC effects of using the baseline configuration and adjusting each policy independently. Changing the mapping function from simple to fair reduces IPC due to contention in the switched network, even though unloaded latencies are lower. Shifting from the baseline multicast to a purely incremental search policy substantially reduces the number of bank accesses (from 266 million to 89 million). However, even though most data are found in one of the first two banks, the incremental policy increases the average access latency from 18.3 cycles to 24.9 cycles and reduces IPC by 10%. The hybrid policies (such as multicast-2/multicast-6) gain back most of the loss in access latency (19.1 cycles) and nearly

Policy	Av. Lat.	IPC	Miss Rate	Bank Access	Policy	Av. lat.	IPC	Miss Rate	Bank Access
Search					*Promotion*				
Incremental	24.9	0.65	0.114	89M	1-bank/2-hit	18.5	0.71	0.115	259M
2 mcast + 6 inc	23.8	0.65	0.113	96M	2-bank/1-hit	17.7	0.71	0.114	266M
2 inc + 6 mcast	20.1	0.70	0.114	127M	2-bank/2-hit	18.3	0.71	0.115	259M
2 mcast + 6 mcast	19.1	0.71	0.113	134M	*Eviction*				
Mapping					insert head, evict random, 1 copy	15.5	0.70	0.117	267M
Fast shared	16.6	0.73	0.119	266M	insert middle, evict random, 1 copy	16.6	0.70	0.114	267M
Baseline: simple map, multicast, 1-bank/1-hit, insert at tail						18.3	0.71	0.114	266M

Table 6: D-NUCA policy space evaluation

Configuration	Loaded Latency	Average IPC	Miss Rate	Bank Accesses	Tag Bits	Search Array
Base D-NUCA	18.3	0.71	0.113	266M	-	—
SS-performance	18.3	0.76	0.113	253M	7	224KB
SS-energy	20.8	0.74	0.113	40M	7	224KB
SS-performance + shared bank	16.6	0.77	0.119	2661	6	216KB
SS-energy + shared bank	19.2	0.75	0.119	47M	6	216KB
Upper bound	3.0	0.83	0.114	—	-	—
Upper bound + SS-performance	3.0	0.89	0.114	—	7	224KB

Table 7: Performance of D-NUCA with smart search

all of the IPC, while still eliminating a great many of the extra bank accesses.

The data promotion policy, in which blocks may be promoted only after multiple hits, or blocks may be promoted multiple banks on a hit, has little effect on overall IPC, as seen by the three experiments in Table 6. The best eviction policy is as shown in the base case, replacing the block at the tail. By replacing the head, and copying it into a random, lower-priority set, the average hit time is reduced, but the increase in misses (11.4% to 11.7%) offsets the gains from the lower access latencies.

While the baseline policy is among the best-performing, using the 2 multicast/6-multicast hybrid look-up reduces the number of bank accesses to 134 million (a 50% reduction) with a mere 1% drop in IPC. However, the number of bank accesses is still significantly higher than any of the static cache organizations. Table 7, shows the efficacy of the smart search policy at improving IPC *and* reducing bank accesses. We computed the size and access width of the different possible smart search configurations, and model their access latencies accurately using Cacti.

By initiating misses early, the SS-performance policy results in a 8% IPC gain, at the cost of an additional 1-2% area (a 224KB smart search tag array). In the SS-energy policy, a reduction of 85% of the bank lookups can be achieved by caching 7 bits of tag per line, with a 6% IPC gain over the base D-NUCA configuration. Coupling the SS-energy policy with the shared mapping policy results in a slightly larger tag array due to the increased associativity, so we had to reduce the smart search tag width to 6 bits to keep the array access time at 5 cycles. However, that policy results in what we believe our best policy to be: 47M bank accesses on average, and a mean IPC of 0.75. The last two rows of Table 7 shows two upper bounds on IPC. The first upper bound row shows the mean IPC that would result if all accesses hit in the closest bank with no contention, costing 3 cycles. The second row shows the same metric, but with early initiation of misses provided by the smart search array. The highest IPC achievable was 0.89, which is 16% better than the highest-performing D-NUCA configuration. We call the policy of SS-energy with the shared mapping the "best" D-NUCA policy DN-best, since it balances high performance with a relatively small number of bank accesses. The upper bound is 19% than the DN-best policy.

5.2 Comparison to ML-UCA

Multi-level hierarchies permit a subset of frequently used data to migrate to a smaller, closer structure, just as does a D-NUCA, but at a coarser grain than individual banks. We compared the NUCA schemes with a two-level hierarchy (L2 and L3), called ML-UCA. We modeled the L2/L3 hierarchy as follows: we assumed that both levels were aggressively pipelined and banked UCA structures. We also assumed that the L3 had the same size as the comparable NUCA cache, and chose the L2 size and L3 organization that maximized overall IPC. The ML-UCA organization thus consumes more area than the single-level L2 caches, and has a greater total capacity of bits. In addition, we assumed no additional routing penalty to get from the L2 to the L3 upon an L2 miss, essentially assuming that the L2 and the L3 reside in the same space, making the multi-level model optimistic.

Table 8 compares the IPC of the ideal two-level ML-UCA with a D-NUCA. In addition to the optimistic ML-UCA assumptions listed above, we assumed that the two levels were searched in parallel upon every access[2]. The IPC of the two schemes is roughly comparable at 2MB, but diverges as the caches grow larger. At 16MB, overall IPC is 17% higher with DN-best than with the ML-UCA, since many of the applications have working sets greater than 2MB, incurring unnecessary misses, and some have working sets smaller than 2MB, rendering the ML-UCA L2 too slow.

The two designs compared in this subsection are not the only points in the design space. For example, one could view a simply-mapped D-NUCA as an n-level cache (where n is the bank associativity) that does not force inclusion, and in which a line is migrated to the next highest level upon a hit, rather than the highest. A D-NUCA could be designed that permitted limited inclusion, supporting multiple copies within a spread set. Alternatively, a ML-UCA in which the two (or more) levels were each organized as S-NUCA-2 designs, and in which inclusion was not enforced, would start to resemble a D-NUCA organization in which lines could only be mapped to two places. However, our experiments with many D-NUCA policies indicate that the ability to effectively adjust the size of the active working set, clustered near the processor, provides

[2] IPC of ML-UCA was 4% to 5% worse when the L2 and L3 were searched serially instead of in parallel.

Techology (nm)	L2/L3 Size	Num. Banks	Unloaded Latency	Loaded Latency	ML-UCA IPC	DN-best IPC
130	512KB/2MB	4/16	6/13	7.1/13.2	0.55	0.58
100	512KB/4MB	4/32	7/21	8.0/21.1	0.57	0.63
70	1MB/8MB	8/32	9/26	9.9/26.1	0.64	0.70
50	1MB/16MB	8/32	10/41	10.9/41.3	0.64	0.75

Table 8: Performance of an L2/L3 Hierarchy

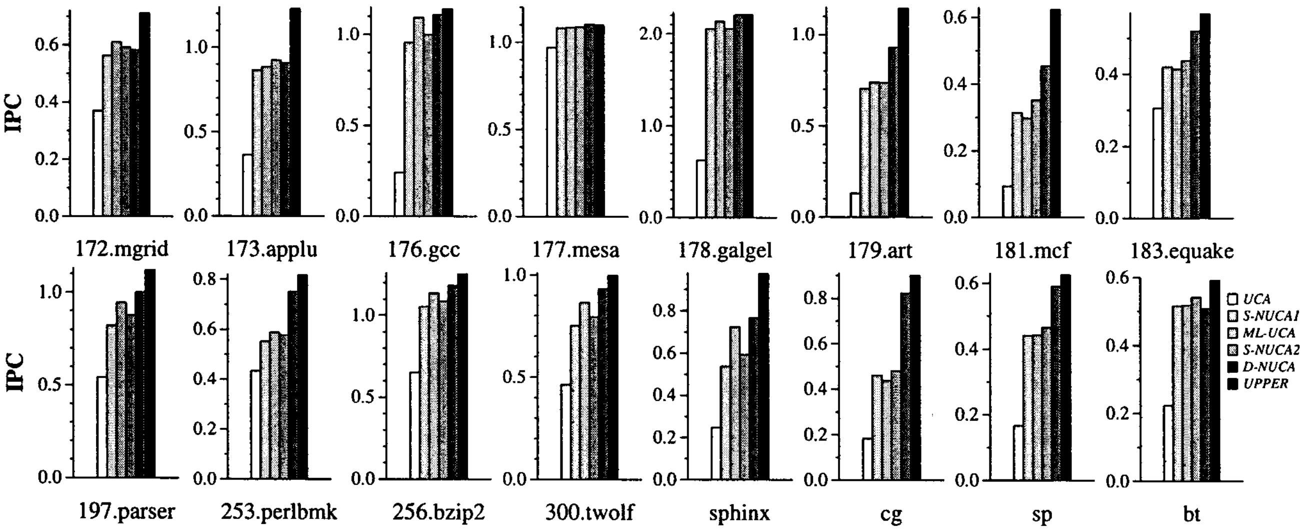


Figure 6: 16MB Cache Performance

better performance and performance stability than competing alternatives.

5.3 Cache Design Comparison

Figure 6 compares the 16MB/50nm IPC obtained by the best of each major scheme that we evaluated: (1) UCA, (2) aggressively pipelined S-NUCA-1, (3) S-NUCA-2, (4) aggressively pipelined, optimally sized, parallel lookup ML-UCA, (5) DN-best, and (6) an ideal D-NUCA upper bound. This ideal bound is a cache in which references always hit in the closest bank, never incurring any contention, resulting in a constant 3-cycle hit latency, and which includes the smart search capability for faster miss resolution.

The results show that DN-best is the best cache for all but three of the benchmarks (*mgrid*, *gcc*, and and *bt*). In those three, DN-best IPC was only slightly worse than the best organization. The second-best policy varies widely across the benchmarks; it is ML-UCA for some, S-NUCA-1 for others, and S-NUCA-2 for yet others. The DN-best organization thus offers not only the best but the most stable performance. The ideal bound (labeled *Upper* on the graphs) shows the per-benchmark IPC assuming a loaded L2 access latency of 3 cycles, and produces an average ideal IPC across all benchmarks of 0.89. Surprisingly, the DN-best IPC is only 16% worse than *Upper* on average, with most of that difference concentrated in four benchmarks (*applu*, *art*, *mcf*, and *sphinx*). This result indicates that DN-best is close to ideal and unlikely to benefit from better migration policies or compiler support to place data in the right banks.

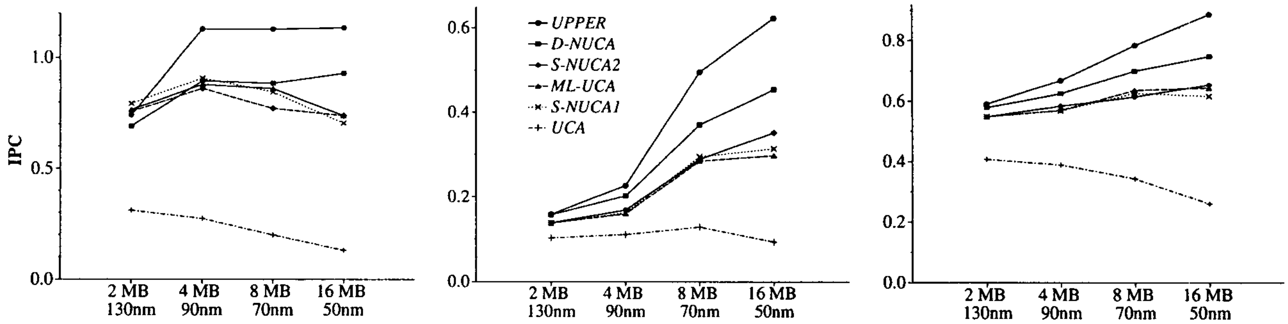
Figure 7 shows how the various schemes perform across technology generations and thus cache sizes. The IPC of *art*, with its small working set size, is shown in Figure 7a. Figure 7b shows the same information for a benchmark (*mcf*) that has a larger-than-average working set size. Figure 7c shows the harmonic mean IPC across all benchmarks.

First, the IPC improvements of D-NUCA over the other organizations grows as the cache grows larger. The adaptive nature of the D-NUCA architecture permits consistently increased IPC 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 twelve applications, within 1% of the best for two applications, within 5% for one application, and within 10% for one application. Figure 7a shows this disparity most clearly in that D-NUCA is the only organization for which *art* showed improved IPC for caches larger than 4MB.

6. TECHNOLOGY PROJECTIONS

We initially performed this study using the technology projections from the 1999 SIA roadmap. Subsequently, the 2001 SIA projections were released, in which wire latencies are projected to grow more severe in future technologies. All results presented in this paper thus far assume the 2001 projections. However, it is instructive to compare the ideal NUCA designs using both the 1999 and 2001 projections, since the technology projections have a major effect on the ideal cache organization. Table 9 shows the effect of technology projections on different NUCA configurations. For example, at 50nm technology, the 2001 projections reduce the wire aspect ratio from 2.8 to 2.5. They also increase the conductor resistivity and interlevel dielectric constant from 1.8 to 2.2 and from 1.5 to 2.1, respectively.

These changes in the projections of material properties results in increased wire resistance and capacitance, and therefore increased delay. The increased projected wire delays causes an increase in the optimal number of D-NUCA banks, from 64 to 256, to reduce the channel delay between banks. However, the area overhead of private channels in the S-NUCA-1 organizations restricts the number of banks. The performance of S-NUCA-1 drops by 9% when the



(a) 179.art (b) 181.mcf (c) All Benchmarks

Figure 7: Performance summary of major cache organizations

Tech. model	Global wire Aspect Ratio	Conductor Resistivity	Insulator Dielectric Constant	Num. banks	Configuration	Loaded latency	Average IPC	Miss rate	Bank accesses
SIA 1999	2.8	1.8	1.5	32	S-NUCA1	21.9	0.68	0.13	15M
				64	Shared bank D-NUCA	12.5	0.78	0.12	144M
					SS-energy + shared bank	15.6	0.78	0.12	36M
SIA 2001	2.5	2.2	2.1	32	S-NUCA1	30.2	0.62	0.13	15M
				256	Shared bank D-NUCA	16.6	0.73	0.12	266M
					SS-energy + shared bank	19.2	0.75	0.12	47M

Table 9: Effect of technology models on results

2001 technology projections replace those from 1999, whereas the DN-BEST performance is reduced by a smaller 4%. Because of the data migration capability, which permits more numerous, smaller banks as the wires slow, D-NUCA organizations are more robust, when faced with slowing wires, than are S-NUCA-1 designs. Finally, we note that the smart search capabilities become more important for technologies with slower wires, as the increased number of banks would otherwise result in a corresponding increase in bank accesses.

7. RELATED WORK

This work is the first to propose novel designs for large, wire-dominated on-chip caches, but significant prior work has evaluated large cache designs. Kessler examined designs for multi-megabyte caches built with discrete components [17]. Hallnor and Reinhardt [9] studied a fully associative software-managed design for large on-chip L2 caches, but not did not consider non-uniform access times.

While our concept of bank-mapped spread sets is new, other work has examined using associativity to balance power and performance. Albonesi examines turning off "ways" of each set to save power when cache demand is low [2]. Powell *et al.* evaluate the balance between incremental searches of the sets to balance power and performance [24], as we do with our multicast versus incremental policies, and as Kessler *et al.* did to optimize for speed [20]. Bank sets do not lend themselves to less regular set mappings that reduce conflicts, such as skewed associativity [4].

Other researchers have examined using multiple banks for high bandwidth, as we do to reduce contention. Sohi and Franklin [29] 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 [32] performed an exhaustive study of the trade-offs involved with port and bank replication and line buffers for level-one caches. Our work aims to flatten deepening hierarchies; a goal that should be compared with Przybylski's dissertation, in which he exhaustively searched the space of multi-level caches to find the performance-optimal point [25].

Finally, many researchers have examined adaptive cache policies, a concept which is inherent in the D-NUCA organization. Dahlgren *et al.* 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 [5]. They also studied, as did Johnson and Hwu [15], adapting the block size to different workload needs. While the D-NUCA scheme leaves data with low locality in banks far from the processor, an alternative approach is not to cache low-locality lines at all. Gonzâlez, Aliagas, and Valero examined both 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 [7]. Tyson *et al.* also proposed a scheme to permit low-locality data to bypass the cache [31].

8. SUMMARY AND CONCLUSIONS

Non-uniform accesses have started to appear in high performance cache designs [23]. In this paper, we proposed several new designs that 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 17 cycles at an 8 FO4 clock, which is a lower absolute latency than conventional L2 caches.

- *Technology scalability*: Increasing wire delays will increase access times for traditional, uniform access caches. The D-NUCA design scales much better with technology than conventional caches, since most accesses are serviced by close banks, which can be kept numerous and small due to the switched network. Keeping cache area constant, the average loaded D-NUCA access times increase only slowly, from 8.4 cycles for a 2MB, 180nm cache to 18.3 cycles for a 16MB, 50nm cache.

- *Performance stability*: The ability of a 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.

- *Flattening the memory hierarchy*: The D-NUCA design outperforms multi-level caches built in an equivalent area, since the multi-level cache has fixed partitions that are slower than an individual bank. This D-NUCA result augurs a reversal of the trend of deepening memory hierarchies. We foresee future memory hierarchies having two or at most three levels: a fast L1 tightly coupled to the processor, a large on-chip NUCA L2, and perhaps an off-chip L3 that uses a memory device technology other than SRAM. Future work will examine a further flattening of the cache hierarchy into a single NUCA structure.

While the emergence of non-uniform cache latencies creates difficulties for some traditional optimization techniques, such as load-use speculation or compiler scheduling, we view the emergence of non-uniform accesses as inevitable. Those optimization techniques must be augmented to handle the non-uniformity where possible, or simply discarded where not.

Maintaining coherence among multiple NUCA caches presents new challenges. A variant of the partial tag compare scheme of Kessler *et al.* [20] may make snooping feasible. The undesirable alternatives to a smart search coherence array are (1) maintaining a huge centralized tag bank for snooping, or (2) broadcasting snoops into the NUCA array upon every bus read and write request.

Finally, 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. Since the banks are individually addressable, the memory system may be reconfigured to support different programming models–such as streaming or vector workloads–by sending configuration commands to individual banks.

ACKNOWLEDGMENTS

We gratefully acknowledge Chuck Moore and Seongmoo Heo for their helpful comments on the final version of this paper. We also thank Mark Hill for his suggestion to use the partial tag matching scheme, which resulted in our highly effective smart search policy. This research is supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-1892, NSF CAREER grants CCR-9985109 and CCR-9984336, two IBM University Partnership awards, and a grant from the Intel Research Council.

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] D.H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd International Symposium on Microarchitecture*, pages 248–259, December 1999.

[3] D. Bailey, J. Barton, T. Lasinski, and H. Simon. The NAS parallel benchmarks. Technical Report RNR-91-002 Revision 2, NASA Ames Research Laboratory, Mountain View, CA, August 1991.

[4] 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.

[5] F. Dahlgren and P. Stenström. On reconfigurable on-chip data caches. In *Proceedings of the 24th International Symposium on Microarchitecture*, pages 189–198, November 1991.

[6] 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.

[7] A. González, 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.

[8] L. Gwennap. Alpha 21364 to ease memory bottleneck. *Microprocessor Report*, 12(14), October 1998.

[9] E.G. Hallnor and S.K. Reinhardt. A fully associative software-managed cache design. In *Proceedings of the 27th International Symposium on Computer Architecture*, pages 107–116, June 2000.

[10] 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.

[11] M. Horowitz, R. Ho, and K. Mai. The future of wires. In *Seminconductor Research Corporation Workshop on Interconnects for Systems on a Chip*, May 1999.

[12] M.S. Hrishikesh, Norman P. Jouppi, Keith I. Farkas, Doug Burger, Stephen W. Keckler, and Premkishore Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, May 2002.

[13] J. Huh, D. Burger, and S.W. Keckler. Exploring the design space of future CMPs. In *Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, September 2001.

[14] 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.

[15] T.L. Johnson and W.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.

[16] 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.

[17] R.E. Kessler. *Analysis of Multi-Megabyte Secondary CPU Cache Memories*. PhD thesis, University of Wisconsin-Madison, December 1989.

[18] R.E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[19] R.E. Kessler, M.D. Hill, and D.A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, June 1994.

[20] R.E. Kessler, R. Jooss, A. Lebeck, and M.D. Hill. Inexpensive implementations of set-associativity. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 131–139, May 1989.

[21] 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(1):35–44, 1990.

[22] D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, 30(9):37–39, September 1997.

[23] 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.

[24] M.D. Powell, A. Agarwal, T.N. 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*, pages 54–65, December 2001.

[25] S.A. Przybylski. *Performance-Directed Memory Hierarchy Design*. PhD thesis, Stanford University, September 1988. Technical report CSL-TR-88-366.

[26] The national technology roadmap for semiconductors. Semiconductor Industry Association, 1999.

[27] P. Shivakumar and N.P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical report, Compaq Computer Corporation, August 2001.

[28] K. So and R.N. Rechtshaffen. Cache operations by MRU change. *IEEE Transactions on Computers*, 37(6):700–109, July 1988.

[29] 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, April 1991.

[30] Standard Performance Evaluation Corporation. *SPEC Newsletter*, Fairfax, VA, September 2000.

[31] 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, December 1995.

[32] K.M. Wilson and K. Olukotun. Designing high bandwidth on-chip caches. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 121–132, June 1997.

[33] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.