Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures

Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar
School of Electrical and Computer Engineering, Purdue University
{zchishti, mdpowell, vijay}@purdue.edu

Abstract

Wire delays continue to grow as the dominant component of latency for large caches. A recent work proposed an adaptive, non-uniform cache architecture (NUCA) to manage large, on-chip caches. By exploiting the variation in access time across widely-spaced subarrays, NUCA allows fast access to close subarrays while retaining slow access to far subarrays. While the idea of NUCA is attractive, NUCA does not employ design choices commonly used in large caches, such as sequential tag-data access for low power. Moreover, NUCA couples data placement with tag placement foregoing the flexibility of data placement and replacement that is possible in a non-uniform access cache. Consequently, NUCA can place only a few blocks within a given cache set in the fastest subarrays, and must employ a high-bandwidth switched network to swap blocks within the cache for high performance. In this paper, we propose the Non-uniform access with Replacement And Placement usIng Distance associativity" cache, or NuRAPID, which leverages sequential tag-data access to decouple data placement from tag placement. Distance associativity, the placement of data at a certain distance (and latency), is separated from set associativity, the placement of tags within a set. This decoupling enables NuRAPID to place flexibly the vast majority of frequently-accessed data in the fastest subarrays, with fewer swaps than NUCA. Distance associativity fundamentally changes the trade-offs made by NUCA's best-performing design, resulting in higher performance and substantially lower cache energy. A one-ported, non-banked NuRAPID cache improves performance by 3% on average and up to 15% compared to a multi-banked NUCA with an infinite-bandwidth switched network, while reducing L2 cache energy by 77%.

1 Introduction

CMOS scaling trends are leading to greater numbers of smaller transistors in a single chip but a relative increase in wire delays. The availability of transistors leads to large, on-chip caches. While small, fast, L1 caches remain close to the processor, L2 or L3 (i.e., lower level) caches use many SRAM subarrays spread out throughout the chip and connected through long wires. Increasing wire delays will continue to grow as the dominant latency component for these caches. The access time of conventional lower-level caches has been the longest access time of all subarrays, but such uniform access fails to exploit the difference in latencies among subarrays.

A recent work [7] proposed an adaptive, non-uniform cache architecture (NUCA) to manage large on-chip caches. By exploiting the variation in access time across subarrays, NUCA allows fast access to close subarrays while retaining slow access to far subarrays. NUCA pioneered the concept of placement based on the access time of the selected block. Frequently-accessed data is placed in subarrays closer to the processor while infrequently-accessed data is placed farther.

While the idea of non-uniform access is attractive, NUCA's design choices have the following problems. To understand the problems, we make the key observation that large caches are implemented significantly differently than small caches.

(1) **Tag search:** While small caches probe the tag and data arrays in parallel, large, lower-level caches often probe the tag array first, and then access only the matching data way [3, 14]. Because the tag array latency is much smaller than the data array latency in large caches and because parallel access results in considerably high energy [3, 9], the small increase in overall access time due to sequential tag-data access is more than offset by the large savings in energy. Although intended for large lower-level caches, NUCA does not use sequential tag-data access; instead it either does a parallel (multicast) search of the ways (albeit sometimes a subset of the ways), or searches the ways sequentially, accessing *both* tag and data, from the closest to the farthest. Because the entire tag array is smaller than even one data way, sequential tag-data access is more energy-efficient than sequential way search if the matching data is not found in the first way (e.g., if the data is found in the second way, sequential way accesses two tag ways and two data ways, while sequential tag-data accesses the entire tag array once and one data way).

NUCA's tag layout adds to the energy inefficiencies of searching for the matching block. NUCA's tag array is distributed throughout the cache along with the data array. Consequently, searching for the matching block requires traversing an switched network, which consumes substantial energy and internal bandwidth.

(2) **Placement:** Even the most flexible placement policy proposed by NUCA is restrictive. NUCA (and conventional caches) artificially *couples* data placement with tag placement; the position in the tag array *implies* the position in the data array. This coupling means that NUCA can place only a small number of ways of each set in the fastest *distance-group* (d-group), which we define as a collection of data subarrays all of which are at the same latency from the processor. For example,

in a 16-way set-associative NUCA cache, this number may be two — i.e., two specific ways of each set may be in the fastest d-group. As long as the frequently-accessed ways within a set are fewer than this number, their access is fast. However, if a "hot" set has more frequently-accessed ways, the accesses are not all fast, even though the fastest d-group is large enough to hold all the ways of the set. To mitigate this problem, NUCA uses a policy of promoting frequently-accessed blocks from slower to faster d-groups, by swapping the ways within a set. These swaps are energy-hungry and also consume substantial internal bandwidth.

(3) **Data Array Layout:** While [7] considered several d-group sizes, the best design choice for NUCA was to divide the cache into many, small d-groups (e.g., many 64-KB d-groups) to provide a fine granularity of access times. In conventional large caches, the bits of individual cache blocks are spread over many subarrays for area efficiency, and hard- and soft-error tolerance. While [7] does not consider error tolerance, doing so would require a NUCA cache block to be spread over only one d-group to achieve the same latency for all the bits in the block. Unfortunately, NUCA's design choice of small d-groups would constrain the spread to only a few small subarrays while conventional caches spread the bits over a much larger space (e.g., the 135 subarrays making up the 3-MB L3 in Itanium II [14]).

(4) **Bandwidth:** To support parallel tag searches and fast swaps, NUCA's best design assumes a multi-banked cache and a complicated, high-bandwidth switched network among the subarrays of the cache ([7] considered a non-banked design without the switched network, but found that design inferior). While the bandwidth demand due to NUCA's tag searches and swaps is artificial, the real bandwidth demand from the CPU is filtered by L1 caches and MSHRs. As such, the real demand for lower-level cache bandwidth is usually low and does not justify the complexity of multibanking and a switched network.

To address problem (1) we use a sequential tag-data access with a centralized tag array which is placed close to the processor. To address problem (2), we make the key observation that sequential tag-data access creates a new opportunity to *decouple* data placement from tag placement. Because sequential tag-data access probes the tag array first, the exact location in the data array may be determined even if there is no implicit coupling between tag and data locations. This decoupling enables *distance associativity*, which allows a completely flexible choice of d-groups for data placement, as opposed to NUCA's set-associativity-restricted placement. Hence, unlike NUCA, all the ways of a hot set may be placed in the fastest d-group. To allow the working set to migrate to the fastest d-group, we swap data out of d-groups based on *distance replacement* within the d-group, although eviction from the cache is still based on *data replacement* within the set.

To address problem (3), we use a few, large d-groups instead of NUCA's many, small d-groups. Our large (e.g. 2-MB) d-groups retain the area-efficiency and fault-tolerance advantages of spreading cache blocks over many subarrays. Because of the higher capacity of our large d-groups and our flexible placement, the pressure on our fastest d-group is significantly lower. Therefore, we do not need to swap blocks in and out of d-groups as often as NUCA. Though our larger d-groups may be slower than NUCA's smaller d-groups, the reduction in swaps more than offsets the longer latency. This reduction in data movement combined with elimination of parallel tag searches reduces bandwidth demand, obviating NUCA's multiple banks and switched network mentioned in problem (4).

In solving these problems, this paper shows that distance associativity *fundamentally* changes the trade-offs made by NUCA's best-performing design, resulting in higher performance and substantially lower energy.

While sequential tag-data access and non-uniform access are not new, the novel contributions of this paper are:

- Our leverage of sequential tag-data access to introduce the concept of distance associativity.
- Our strategy of using a few, large d-groups.
- Our distance-placement and distance-replacement policies. Because our policies require fewer swaps, our cache has 61% fewer d-group accesses than NUCA.
- Over 15 SPEC2K applications, our one-ported, non-banked cache improves performance by 3% on average and up to 15% over a multi-banked NUCA with an infinite-bandwidth switched network, while consuming 77% less dynamic cache energy. Our cache reduces processor energy-delay by 7% compared to both a conventional cache and NUCA.

In Section 2 we discuss NuRAPID caches. In Section 3, we discuss layout of large caches into subarrays for area efficiency and fault tolerance and the floorplan concepts behind d-groups. Section 4 describes our experimental methodology and Section 5 our results. In Section 6 we discuss related work. We conclude in Section 7.

2 Distance Associativity

We propose the "Non-uniform access with Replacement And Placement usIng Distance associativity" cache, or *NuRAPID*. As shown in Figure 1, NuRAPID divides the data arrays into several d-groups, with different access latencies. Upon a cache access, the tag array is accessed before the data array (sequential tag-data access as discussed in Section 1.) The decoupling of placement in the two arrays allows blocks within a set to be placed within the same d-group, such as blocks A and B in the figure, or different d-groups, such as blocks C and D. Distance associativity is a data placement flexibility and should not be confused with tag placement and the index-to-set mapping of set associativity. We maintain conventional set associativity in the tag array and manage distance associativity separately.

We first discuss distance-associative (d-a) placement and implementation. Then we explain replacement in NuRAPID. In our discussion of placement and replacement, we contrast our design to the dynamic-NUCA (D-NUCA, not to be confused with d-groups) scheme, the best-performing policy proposed in

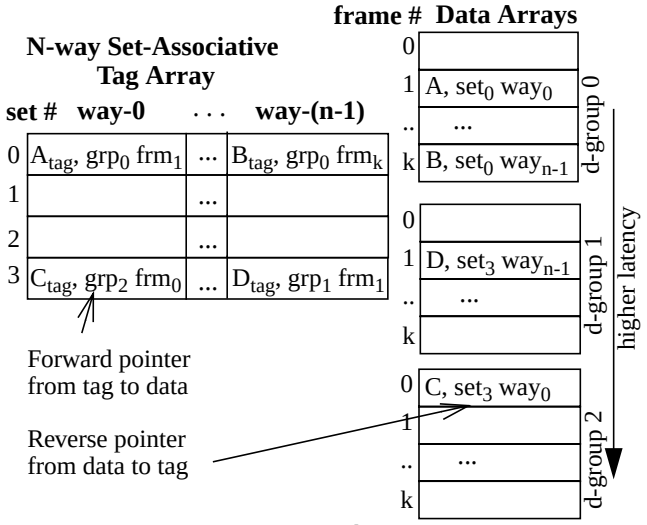

FIGURE 1: NuRAPID cache.

[7]. We then discuss bandwidth and timing issues. Finally, we discuss optimizations and implementation simplifications for NuRAPID.

2.1 Distance-Associative Placement

The key to a successful memory system, whether a conventional cache hierarchy, D-NUCA, or NuRAPID, is to place frequently-accessed data where it may be accessed quickly. A conventional cache hierarchy attempts to meet this goal by placing recently accessed data in the fastest level (or all levels when inclusion is maintained).

To meet this goal, D-NUCA "screens" cache blocks to determine if they merit placement in the fastest d-group by initially placing a block in the slowest d-group and then moving it to faster d-groups after additional accesses. (in D-NUCA terms, a d-group consists of the ways within a bank-set having the same latency.) This conservative screening process is necessary because space in the fastest d-group is quite limited. Unfortunately, this conservative initial placement policy is costly in both performance and energy. D-NUCA's placement policy requires that initial accesses are slow, and the repeated promotion operations for a block cause high-energy swaps.

It would seem that initially placing a block in the fastest d-group (instead of the slowest) might alleviate both the performance and energy problems for D-NUCA. However, this initial placement policy would require demoting a frequently-accessed block within the same set to a slower d-group upon every cache miss. The tendency of individual sets to be "hot" with many accesses to many ways over a short period makes such a policy undesirable. In fact, [7] evaluates such an initial placement policy and concludes that it is less effective. The set-associativity-coupled placement in D-NUCA makes space in the fastest d-group too valuable to "risk" on a block that may be accessed only a few times.

NuRAPID, in contrast to D-NUCA, can place blocks from any number of ways within a set in any d-group. Because of this flexibility, NuRAPID can place *all* new cache blocks in the fastest d-group without demoting a member of the same set to a slower d-group. This policy makes the initial accesses to a block fast, and allows many blocks in a hot set (up to the set associativity of the tag array), to be placed in the fastest d-group. This change in the initial placement is one of the fundamental changes in design choices afforded by distance associativity, as mentioned in Section 1.

Distance associativity is implemented by introducing a *forward pointer*, completely decoupling distance placement from tag array placement. A forward pointer, which allows an entry in the tag array to point to an arbitrary position within the data subarrays, is added for each entry in the tag array as shown in Figure 1. The figure shows a tag array with forward pointers for blocks (e.g., block A has a forward pointer to d-group$_0$ frame$_1$). A tag match occurs in the same way as in a conventional set-associative cache with sequential tag-data access. However, the successful tag match now returns the hit signal and a forward pointer which is used to look up the data array. Because the tag array outputs the forward pointer in parallel with the tag, the only minimal impact on access speed is that the tag array is a little wider than usual. While the tag match is still set associative (i.e., in an n-way cache, only n blocks within a set may be present in the tag array), the data array is fully distance associative in that any number of blocks within a set may be placed in a d-group.

2.2 Distance-Associative Replacement

We establish a novel replacement policy for NuRAPID that complements the use of d-groups. Conventional caches are concerned only with data replacement, defined as the choice of which block to evict from the cache. Distance associativity adds a new dimension, distance replacement, which concerns the replacement of a far block with a closer one. The key difference between data replacement and distance replacement is that data replacement involves evicting a block from the cache. Distance replacement *does not evict any block from the cache*, but instead swaps blocks within the cache. Conventional caches couple replacement in the tag array to replacement in the data array; it does not make sense in a conventional cache to replace data in one way with data from another, because all ways are equivalent.

Because D-NUCA's tag and data placement are coupled, its data replacement and distance replacement policies are also coupled. Both forms of replacement occur within a single set, and both are accomplished by swapping blocks in a cache set that are in d-groups of adjacent-latencies. We call this policy bubble replacement because it is like a bubble sort; frequently-accessed blocks bubble toward the fastest set. For data replacement, bubble replacement means that D-NUCA evicts the block in the slowest way of the set. The evicted block may not be the set's LRU block. For distance replacement, bubble replacement means that blocks must bubble all the way from the slowest to fastest set before having fast access time, and vice-versa before eviction.

In contrast, NuRAPID uses a completely flexible distance

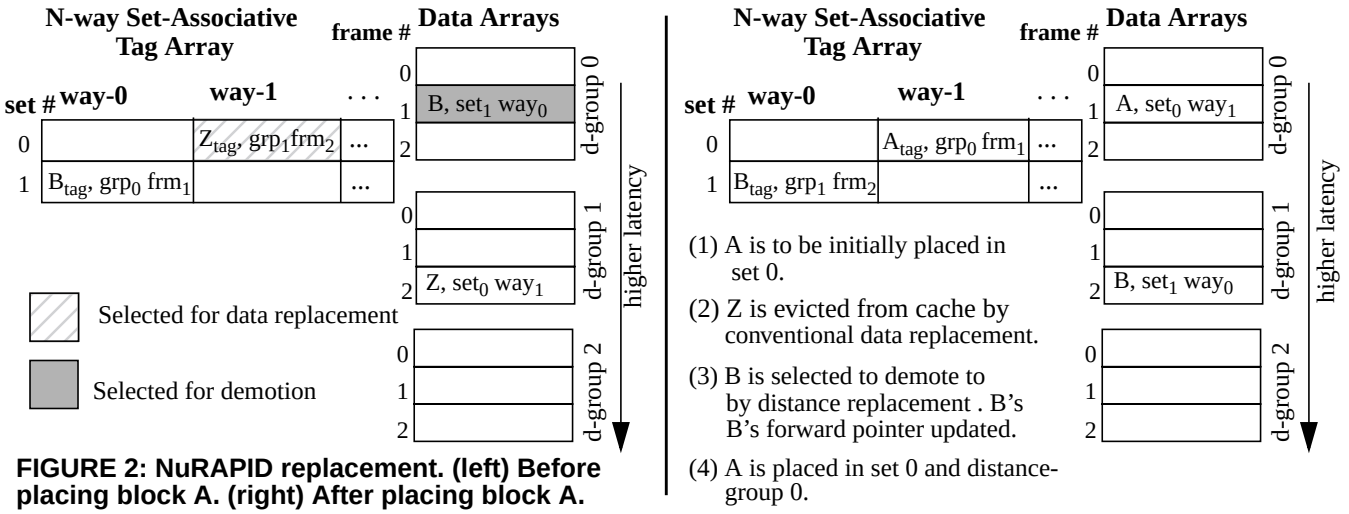

FIGURE 2: NuRAPID replacement. (left) Before placing block A. (right) After placing block A.

replacement policy that is independent of data replacement. In distance replacement, when a block needs to be placed in or migrated to a d-group, another block currently in that d-group is moved to a slower d-group, *even if the concerned blocks are not in the same set*. Distance replacement has the advantage of selecting the block to demote from among a large d-group (e.g., a d-group may be 2-MB) while D-NUCA's choice is limited to selecting from among the much smaller set (e.g., an 16-way or 32-way set). Of course, data replacement is still necessary in NuRAPID (i.e., evicting data from the cache, not just a d-group). However NuRAPID completely decouples distance replacement from data replacement. Consequently, the specific data replacement policy (e.g., LRU, random) may be *different* from the specific distance-replacement policy, as described in Section 2.4.2.

We illustrate our distance replacement by means of an example in Figure 2. The cache has 3 d-groups, and d-group 0 is the fastest. A cache miss has occurred, and block-A is to be placed in the cache. First, space must be created in the tag array within the set for block A; therefore a block from A's set, say block Z, is evicted from the cache using conventional data replacement. This data replacement creates an empty frame somewhere within one of the d-groups; in our example it happens to be d-group 1.

According to our placement policy (Section 2.1), the new block A is to be placed in d-group 0. Therefore, unless there is an empty frame within d-group 0, an existing block in d-group 0 must be replaced. Our distance replacement chooses a block within d-group 0, which may or may not be in the same set as block A. In the example, block B, a member of another set, is chosen for demotion to d-group 1.

Our policy chooses block B for distance replacement, but to demote B to d-group 1, we must update the forward pointer for block B to point to its new location. To do so, we must find the tag entry for block B. In a conventional cache, the data array does not provide an explicit pointer to the tag array because the two are coupled and the mapping is implied.

To provide the location of the tag entry in the decoupled NuRAPID, we introduce a *reverse pointer* within the frame that locates the tag entry for each block. Figure 1 and Figure 2 show the reverse pointer for a block X as "set_i way_j". Each data frame contains a reverse pointer that can point to any location within the tag array, just as the forward pointer can point to any location within the data array. In Figure 2, we use B's reverse pointer to locate the tag entry for block B and update B's forward pointer. Much like the forward pointer, the data array outputs the reverse pointer in parallel with the data, the only minimal impact on access speed is that the data array is a little wider than usual.

The demotion is complete when block B is moved into the empty frame (vacated by block Z) in d-group 1. If there had not been an empty frame in d-group 1, then distance replacement would have created space for block B within d-group 1 by demoting another block to d-group 2. However, *at no time does distance replacement evict a block from the cache*. Upon a cache miss, conventional data replacement evicts a block from the cache, creating an empty frame within some d-group. If the empty frame is in d-group 0, which will occur if all members of a set are in d-group 0, the new block simply occupies that frame with no demotions necessary. If the evicted block was in a slower d-group, demotions occur d-group by d-group until an empty frame is filled by a demoted block. Once an empty frame is filled, no additional demotions are necessary. In the worst case, where the block is evicted from the slowest of n d-groups, n-1 demotions will be required.

2.3 Distance-Associative Bandwidth and Timing

The reduction in swaps for NuRAPID compared to D-NUCA significantly reduces bandwidth requirements. To provide high bandwidth and allow many tag searches and swaps to occur simultaneously, D-NUCA needs a multi-banked cache with full-fledged switched network among d-groups ([7] considers designs without multibanking and network, and found them to be inferior). In contrast, NuRAPID uses only one port and is not banked, permitting only one operation to occur at a time. For example, any outstanding swaps must complete before a new access is initiated. In Section 5.4 we show that

because of greatly-reduced swaps and elimination of parallel tag searches, NuRAPID's performance is not hindered by the reduced bandwidth. Thus, distance associativity fundamentally changes the trade-offs by removing the artificial high-bandwidth needs of D-NUCA, as mentioned in Section 1.

2.4 Optimizations and Simplifications

In this section, we discuss optimizations and simplifications for NuRAPID. These techniques reduce energy and hardware complexity and improve NuRAPID performance.

2.4.1 Distance Placement and Distance Replacement

The distance placement and distance replacement policies discussed in Section 2.1 and Section 2.2 assume that blocks are initially placed into the fastest d-group and then demoted by distance replacement. Blocks were not *promoted* from a slow to fast d-group. We call this a *demotion-only* policy.

Disallowing promotions may be undesirable because a frequently-accessed block may become stuck in a slow d-group. To prevent this problem, we propose two optimizations that add promotions to the basic distance replacement policy described in Section 2.2. In a *next-fastest* promotion-policy, when a cache block in any d-group other than the fastest is accessed, we promote it to the next-fastest d-group (correspondingly demoting the LRU-block in the faster d-group). While the next-fastest policy may seem similar to bubble replacement within cache sets in D-NUCA, it is key to note that our promotions and demotions occur within large d-groups, not cache sets. An alternative is the *fastest* promotion policy. In this policy, when a cache block in any d-group other than the fastest is accessed, we promote it to the fastest d-group (correspondingly applying distance replacement if demoting other blocks). In addition to the demotion-only policy, we evaluate both promotion policies in Section 5.2.

2.4.2 Data- and Distance-Replacement Policies

For data replacement, we use conventional LRU to select an individual block from a set for eviction from the cache. Distance replacement must be handled differently because d-groups are large and contain many cache blocks (e.g., 16384 cache blocks in a 2-MB d-group). Tracking true-LRU among thousands of cache blocks is not comparable to tracking LRU among a handful of ways in a cache as in data replacement. While using LRU as the selection policy for distance replacement is desirable for performance, its implementation may be too complex. The size of LRU hardware is of $O(n^2)$ in the number of elements being tracked [12].

Approximate-LRU can reduce the complexity but still may be undesirable for large d-groups. Random replacement provides a simpler alternative but risks accidental demotion of frequently-accessed blocks. Promotion policies such as next-fastest and fastest, discussed in the previous subsection, compensate for these errors by re-promoting those blocks. In Section 5.3.1 we show that using random replacement over true LRU for distance replacement has minimal impact on the performance of NuRAPID.

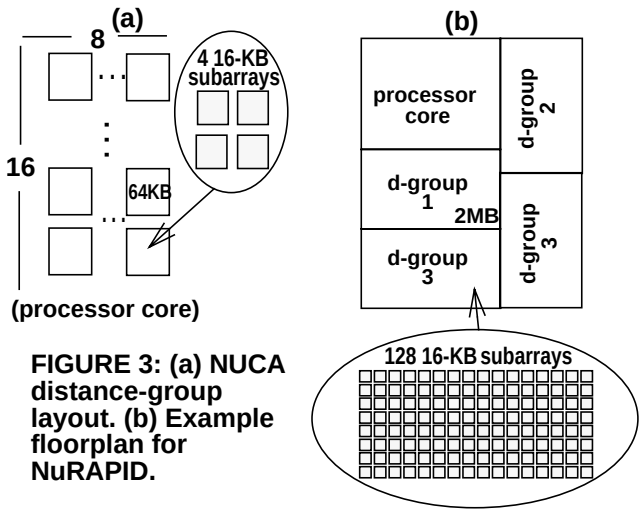

FIGURE 3: (a) NUCA distance-group layout. (b) Example floorplan for NuRAPID.

2.4.3 Restricting Distance Associativity

Section 2.1 and Section 2.2 assume that the forward pointer in the tag array may point to any arbitrary cache block within any d-group, and that the reverse pointer may point to any arbitrary tag-array-entry. This complete flexibility may result in undesirably large forward and reverse pointers.

For example, in an 8-MB cache with 128B blocks, 16-bit forward and reverse pointers would be required for complete flexibility. This amounts to 256-KB of pointers. While only a 3% overhead compared to the total cache size, such overhead may be undesirable in some situations. (For reference, the 51-bit tag entries for this 64-bit-address cache are a 5% overhead.)

There are two considerations for the size of the forward and reverse pointers. The first is that as block sizes increase, the size of the pointers (in addition to the size of the tag array) will decrease. Large caches are trending toward larger block sizes. The second is that the pointer overhead can be substantially reduced by placing small restrictions on data placement within d-groups. If our example cache has 4 d-groups, and we restrict placement of each block to 256 frames within each d-group, the pointer size is reduced to 10 bits. Because of the advantages of NuRaPID, we believe the pointer overhead is acceptable.

3 Layout

In this section, we discuss the data-array layout of large caches, whether conventional, NUCA, or distance associative, into many subarrays, and explain why using a small number of large d-groups for data is also desirable from a layout standpoint. (The tag array is small and is not relevant to this discussion.)

3.1 Conventional Data Array Floorplan and Circuitry

Large caches are made up of many SRAM subarrays to optimize for access time, area, and energy consumption. For example, the 3-MB L3 cache in the Itanium II consists of 135 subarrays that are laid out approximately in an L shape [14]. The block address is divided into a row address, determining

which rows of which subarrays contain the block, and a column address, determining which columns of that row contain the block. When the cache is accessed, portions of the block are read from many subarrays. Because the columns of each block are spread among several subarrays, column muxes are required to reassemble the output into a complete block.

There are several reasons for spreading individual blocks among several subarrays. First, if each subarray row were to contain bits from only one block, then the address row decoders would be required to pinpoint the exact location of the block within a specific row of a specific subarray before choosing a subarray, and the output of the subarrays would be wires that contain no selection logic. From a circuit standpoint, such large decoders or muxes are undesirable. For example, it is preferable to have a 5-to-1 decoder and 2, 2-to-1 muxes over a single 10-to-1 decoder. Placing the contents of several blocks in each subarray row distributes the task of identifying a block's location between the address row decoders and the output column muxes.

Second, distributing blocks among subarrays facilitates use of spare subarrays to compensate for defects caused by hard errors. Spare subarrays are often included in a design to protect against hard errors. The L3 cache in the Itanium II contains 2 spare subarrays out of a total of 135 [14]. During chip testing, defective subarrays are identified and permanently unmapped from the cache using on-die fuses. If the selection logic is spread between the address row decoders and the output column muxes as discussed above, row decoders may point to both defective and non-defective subarrays. The correct data is statically selected (with permanent fuses) using *only* the column muxes, avoiding interference with the more complex address row decoders. This configuration also allows many blocks (that share common row addresses) to share a small number of spare subarrays.

Third, distributing blocks among subarrays helps reduce the chance of data corruption due to soft errors caused by alpha particle strikes. Large caches often contain extra subarrays for error-correcting-code (ECC). If error-corrected data is physically spread among several subarrays, it is less likely that an alpha particle strike will corrupt more bits than are protected by ECC.

Table 1: System parameters.

Issue width	8
RUU	64 entries
LSQ Size	32 entries
L1 i-cache	64K, 2-way, 32 byte blocks, 3 cycle hit, 1 port, pipelined
L1 d-cache	64K 2-way, 32 byte blocks, 3 cycle hit, 1 port, pipelined, 8 MSHRs
Memory latency	130 cycles + 4 cycles per 8 bytes
Branch predictor	2-level, hybrid, 8K entries
Mispredict penalty	9 cycles

3.2 NUCA Data Arrays

Unfortunately the best-performing NUCA is not amenable to these large cache design considerations. While [7] considers several d-group sizes, the best-performing NUCA uses a large number of small d-groups. For example, the 8-MB, 16-way NUCA has 128, 64-KB d-groups, as shown in Figure 3(a). Each of the d-groups is limited to a small number of block addresses, has its own tag, and must be accessed independently, meaning blocks cannot be distributed in subarrays across the small d-groups. This restriction violates the considerations discussed in Section 3.1. For example, a spare subarray cannot be shared across blocks in different d-groups in NUCA, because 1) the d-groups do not share common row addresses, and 2) the d-groups may have different access latencies.

3.3 NuRAPID Cache Data Arrays

To exploit the variation in access time between close and far subarrays while retaining the advantages of distributing data among several subarrays, NuRAPID uses a few large d-groups (in contrast to NUCA's numerous small d-groups). An example floorplan for an 8-MB, 8-way, NuRAPID with 4 d-groups is shown in Figure 3(b). Each d-group in NuRAPID contains many more subarrays than those in NUCA. We use a typical L-shaped floorplan for NuRAPID with the processor core in the unoccupied corner. It should be noted that distance associativity is not tied to one specific floorplan; many floorplans are possible but all will have many subarrays and substantial wire delays to reach distant subarrays.

While the advantages from Section 3.1 are retained, the large d-groups will have longer latencies than the smaller d-groups in NUCA. However, the access latency of distant subarrays is dominated by the long wires between the subarrays and the core. Therefore, many similarly-distant subarrays may be combined into one d-group which shares a large number of block addresses, column muxes, and data blocks, without significantly compromising access time.

Because d-groups in NuRAPID are larger than those of NUCA, latency of the fastest d-group in NuRAPID will of course be longer than that of the fastest d-group in NUCA. However, we show in Section 5.4 that the impact of the longer latency is more than offset by the reduction in swaps.

4 Methodology

Table 1 shows the base configuration for the simulated systems. We perform all our simulations for 70 nm technology, with a clock frequency of 5 GHz. Our base configuration has a 1-MB, 8-way L2 cache with 11-cycle latency, and an 8-MB, 8-way L3 cache, with 43-cycle latency. Both have 128-B blocks. We use the same configuration for L2 and L3 in our base case as used by [7] in their comparison with multi-level cache.

For NUCA evaluation, we assume an 8-MB, 16-way D-NUCA L2 cache with 8 d-groups per set. This configuration is the same as the one mentioned as the optimal configuration in

Table 2: Example cache energies in nJ.

Operation	Energy
Tag + access: closest of 4, 2-MB d-groups	0.42
Tag + access: farthest of 4, 2-MB d-groups (includes routing)	3.3
Tag + access: closest of 8, 1-MB d-groups	0.40
Tag + access: farthest of 8, 1-MB d-groups (includes routing)	4.6
Tag + access: closest 64-KB NUCA d-group	0.18
Tag + access: other 64-KB NUCA d-groups (includes routing)	0.18-4.0
Access 7-bit-per-entry, 16-way NUCA smart-search array	0.19
Tag + access: 2 ports of low-latency 64-KB 2-way L1 cache	0.57

[7]. We model contention for d-group access (i.e., bank contention in [7] terminology) and allow an infinite-bandwidth channel (switched network) for D-NUCA in our simulations.

We allow swaps and cache accesses to proceed simultaneously regardless of channel contention, giving a bandwidth advantage to D-NUCA. D-NUCA uses a smart-search array, which caches partial tag bits. We allow infinite bandwidth for the smart-search array. We model the smart-search array in D-NUCA by caching 7 bits from each tag, as recommended by [7]. We use the least significant tag bits to decrease the probability of false hits in the smart-search array. We obtain latencies for each D-NUCA d-group and smart-search array from [7].

We assume an 8-MB, 8-way NuRAPID L2 cache with 4 d-groups. We assume an L-shaped floorplan for NuRAPID, as shown in Figure 3(b). We assume a one-ported NuRAPID which allows only one access to occur at a time, so that any outstanding swaps amongst d-groups must complete before a new access is initiated. We modify Cacti [11] to derive the access times and wire delays for each d-group in NuRAPID. Because Cacti is not generally used for monolithic large caches (e.g., greater than 4 MB), we make the following modifications: 1) Treat each of our d-groups (one to four MB) as independent (although tagless) caches and optimize for size and access time; 2) Account for the wire delay to reach each d-group based on the distance to route around any closer d-groups using the RC wire-delay models in Cacti; and 3) Optimize our unified tag array for access time.

Table 3: SPEC2K applications and L2 accesses per thousand instructions.

Benchmark/Type	IPC	Accesses	Benchmark/Type	IPC	Accesses
High Load					
applu/FP	0.9	42	lucas/FP	0.5	37
apsi/FP	1.3	18	mcf/Int	0.2	188
art/FP	0.4	107	mgrid/FP	1.1	23
equake/FP	0.7	39	parser/Int	1.1	15
galgel/FP	0.9	28	perl/Int	1.0	28
gcc/Int	1.3	28	twolf/Int	1.0	25
Low Load					
bzip2	1.7	9	wupwise/FP	2.0	10
mesa	1.9	3			

We extend Wattch [1] and SimpleScalar [2] running the Alpha ISA to simulate an out-of-order processor with different cache organizations and obtain performance and energy statistics. Because Wattch assumes a conventional uniform-access cache, we cannot use the Wattch energy model for caches. We modify Cacti as described above to derive the energy consumption for D-NUCA, NuRAPID, and conventional caches, taking into account the wire energy to route cache blocks from distant locations. For D-NUCA, we assume that the switched network switches consume zero energy. For NuRAPID, we consider both access time *and* energy overhead of forward and reverse pointers, and include this overhead in our calculations. We replace the Wattch energy model for all caches with our Cacti-derived model. We show representative numbers in Table 2. For all other processor components, we use the Wattch energy model.

Table 3 summarizes the SPEC2K applications used in our simulations and shows their base IPC. Because this paper focuses on lower-level caches, we are primarily interested in applications with substantial lower-level cache activity. We categorize the applications into two classes as shown in Table 3- *high load*, and *low load* - based on the frequency of L2 accesses. We show results for a subset of the applications that focuses on high load. Out of the remaining SPEC2K applications, 3 are high-load and 8 are low-load. Their behavior is similar to that of the applications shown.

For each application, we use ref inputs, fast-forward 5 billion instructions, and run for 5 billion instructions. During the fast-forward phase, we warm-up both L1 and L2 caches.

5 Results

In Section 5.1, we show that the latencies for larger d-groups in NuRAPID are higher than the average latencies for smaller d-groups in D-NUCA. Section 5.2 demonstrates that NuRAPID outperforms set-associative placement or an L2/L3 hierarchy and that distance-replacement optimizations result in a considerable improvement over NuRAPID with demotion-only policy. In Section 5.3, we show that random distance replacement performs almost as well as true-LRU and that the variation in number of d-groups in NuRAPID has both perfor-

Table 4: Cache latencies in cycles.

Capacity	2 d-groups NuRAPID	4 d-groups NuRAPID	8 d-groups NuRAPID	D-NUCA (average)*
1st MB (fastest)	19	14	12	4-9 (7)
2nd MB	19	14	19	9-12 (11)
3rd MB	19	18	20	12-15 (14)
4th MB	19	18	31	15-18 (17)
5th MB	43	36	32	18-21 (20)
6th MB	43	36	32	21-24 (23)
7th MB	43	44	48	24-27 (26)
8th MB (slowest)	43	44	49	27-31 (29)

*Because D-NUCA's 64-KB d-groups are smaller than 1 MB, we report the latency range and average latency for each MB.

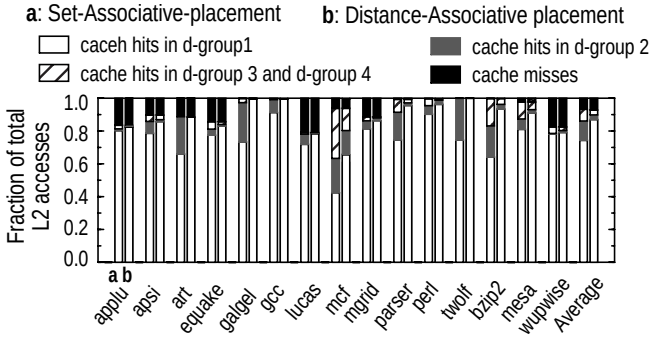

FIGURE 4: Distribution of group accesses for set-associative and distance-associative placement.

mance and energy implications. Section 5.4 shows that NuRAPID outperforms D-NUCA in most applications and consumes less energy than both D-NUCA and the L2/L3 hierarchy.

5.1 Cache Latencies

In this section, we compare latencies of NuRAPID and D-NUCA. For an 8-MB NuRAPID, we show configurations of 2, 4, and 8 d-groups. We expect configurations with a few large d-groups to have longer latencies than those with many, small d-groups.

Table 4 shows cache latencies for the various configurations. The nth row shows the access latency of the nth fastest (and closest) megabyte. The first three columns show d-group configurations for NuRAPID. For example, the third and fourth row of the 4-d-group column have the same latency because the third and fourth row are in the same 2-MB d-group. The latencies for NuRAPID are derived from Cacti [11] cache latencies for various capacities and also include 8 cycles for the 8-way tag latency. (Recall from Section 1 that NuRAPID uses sequential tag-data access.)

As expected, the larger d-groups have longer access latencies than smaller ones. For example the fastest 1-MB d-group in the 8-d-group configuration has a latency of only 12 cycles, compared to a latency of 19 cycles for the fastest d-group in the 2-d-group configuration.

Two other behaviors are worth noting. First, as the number of d-groups increases, the latency of the slowest megabyte increases even as the latency of faster megabytes decreases. This behavior occurs because small, far d-groups are placed in remote locations on the floorplan. The second behavior is that when there are many d-groups, some have similar latency because they fit into the floorplan at about the same distance.

The fourth column in Table 4 shows latencies for the 8-MB, 16-way, D-NUCA which is divided into 128, 64-KB d-groups. Because each megabyte in D-NUCA contains many d-groups, we report both the range of latencies and average latency for each megabyte to facilitate comparison to our d-groups. When reporting latencies for D-NUCA, we report the latencies for each megabyte regardless of set mapping (i.e., the first megabyte contains different numbers of ways in different sets).

D-NUCA's latencies for fast and slow d-groups are lower than our d-group latencies for three reasons. First, D-NUCA uses parallel tag-data access, not sequential tag-data access. Second, D-NUCA has small d-groups that allow fast access to the closest d-groups. Third, D-NUCA assumes a more aggressive, rectangular floorplan than the L-shaped floorplan we assume for our d-groups. However, in the next sections we will show that in spite of longer latencies for large d-groups, the placement and distance replacement policies in NuRAPID allow it to outperform D-NUCA.

5.2 Placement and Replacement Policy Exploration

5.2.1 Set-Associative vs. Distance-Associative Placement

In this section, we compare the performance of set-associative cache placement to decoupled, distance-associative placement in a non-uniform cache. We expect distance-associative placement to have a greater fraction of accesses to faster d-groups than set-associative placement.

Our comparison uses an 8-MB 8-way cache with 4 2-MB d-groups. (Although 8-way with 8 d-groups is simpler to understand, we use 4 d-groups because it is the primary configuration used throughout the results.) Because this design is an 8-way cache with 4 d-groups, in the set-associative-placement cache, each cache block can map to either of 2 frames within each d-group. For distance-associative placement, the location of a cache block can be anywhere within any d-group. While the set-associative cache uses LRU replacement, NuRAPID uses LRU for data replacement and random for distance replacement. To isolate the effects of set-associative and d-a placement, *both* caches initially place blocks in the fastest d-group, demote replaced blocks to the next slower d-group, and use the next-fastest promotion policy. (In the next subsection, we show the next-fastest promotion policy is best.) This set-associative cache is similar to D-NUCA's best design with bubble replacement but with initial placement in the fastest d-group.

Figure 4 shows the fraction of cache accesses to each d-group and the fraction of cache misses. The x-axis shows our applications with the average on the right. The black portion of each stacked bar represents the misses in the L2 cache. For the set-associative cache, an average of 74% of accesses hit in the first d-group. In contrast, the flexible placement policy of NuRAPID results in 86% of accesses hitting in the first d-group. The set-associative cache also makes 8% of accesses to the last 2 d-groups, substantially more than NuRAPID's 2%. This experiment illustrates the benefits of decoupled distance-associative placement.

5.2.2 Distance Replacement Policy Exploration

In this section, we evaluate the performance of the distance replacement policies explained in Section 2.4.1. We use 4 d-groups of 2-MB each for NuRAPID. We use random distance replacement for each 2-MB d-group. We expect the distance replacement optimizations to improve performance over the demotion-only policy.

Figure 5 shows the distribution of accesses to different d-groups as a percentage of total L2 accesses for demotion-only,

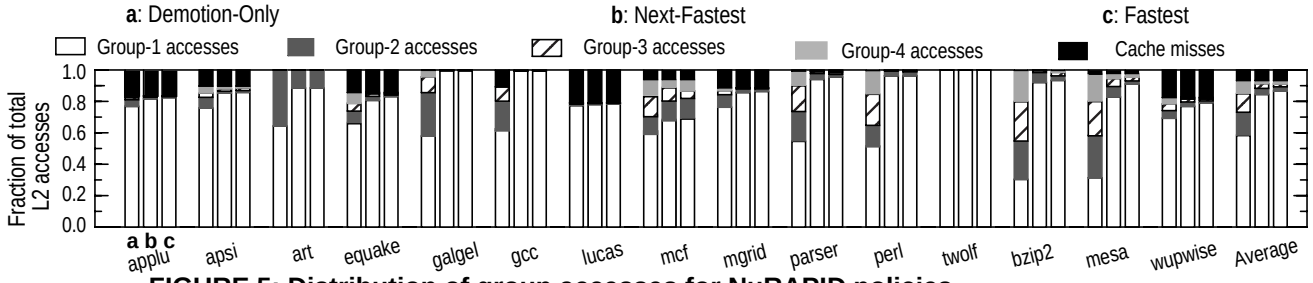

FIGURE 5: Distribution of group accesses for NuRAPID policies.

next-fastest and fastest. The miss rates for NuRAPID remain the same for the three policies because, as mentioned in Section 2.2, distance replacement does not cause evictions.

Next-fastest and fastest result in more L2 hits to the nearest d-groups compared to demotion-only. On average, demotion-only, next-fastest and fastest result in 50%, 84% and 86% accesses to the first d-group respectively. Because demotion-only does not allow the demoted blocks to be promoted upon future accesses, frequently-accessed blocks often become stuck in slower d-groups, thus decreasing the percentage of accesses to faster d-groups. Next-fastest and fastest solve this problem.

Figure 6 compares the performance of different policies with the baseline L2/L3 hierarchy. We also show the performance for an ideal case where every hit in NuRAPID hits in the first d-group, resulting in a constant 14-cycle hit latency. The y-axis shows the performance for demotion-only, next-fastest, fastest, and the ideal case relative to the base case performance.

The demotion-only policy performs slightly worse than the base case, while the next-fastest and fastest policies outperform the base case and perform almost as well as the ideal case. As shown in Figure 5, the percentage of accesses hitting in slower d-groups for both next-fastest and fastest is small, so performance near the ideal case is expected. On average, demotion-only performs 0.3% worse as compared to the base case, and the performance improvements for the next-fastest, fastest, and ideal relative to the base case are 5.9%, 5.6%, and 7.9% respectively. On average, next-fastest and fastest perform within 98% and 97% of the ideal case performance respectively.

The next-fastest and fastest policies show more performance improvements for high-load applications than for low-load applications. Both next-fastest and fastest show maximum improvements for the high-load application *art*. The distance replacement optimizations distribute the large working set in *art* in faster d-groups, resulting in 43% improvement for next-fastest and 42% improvement for fastest. On average, next-fastest shows performance improvements of 6.9% and 1.7%, while fastest shows performance improvements of 6.6% and 1.3% for the high-load and low-load applications respectively. Low-load applications provide less opportunity for a high-performance L2 cache, lessening the effect of distance associativity on overall performace.

Because the next-fastest policy performs better than the fastest policy, we choose next-fastest as our optimal distance replacement policy and do not show fastest any more.

5.3 Design Sensitivity

In this section, we discuss the impact of distance replacement policies and the number of d-groups on performance of NuRAPID.

5.3.1 LRU approximation

We compare the performance of random distance replacement with true LRU. We do not show any graph here. The performance gap between LRU and random distance replacement is significant only for demotion-only. A 4-d-group NuRAPID, using demotion-only and perfect-LRU has 64% first-group accesses on average. Random distance replacement has 54% first d-group accesses on average. A random policy increases the chances of a frequently-accessed block being demoted. Because demotion-only does not promote the blocks from slower to faster d-groups, demoted blocks become stuck. In contrast, random performs nearly as well as true-LRU for next-fastest. On average, for next-fastest, random distance replacement has 84% accesses in the first d-group. and LRU distance replacement has 87% accesses in the first d-group. By giving demoted blocks a chance to be promoted, the next-fastest policy compensates for errors in random distance replacement. For the remainder of the paper, all NuRAPID results use random distance replacement and next-fastest promotion policy.

5.3.2 Number of distance-groups

In this section, we compare results for NuRAPID using 2, 4, and 8 d-groups. We expect to see a performance trade-off between the larger capacity of large, slow d-groups and the lower latency of small, fast d-groups.

Figure 7 shows the distribution of accesses to different d-

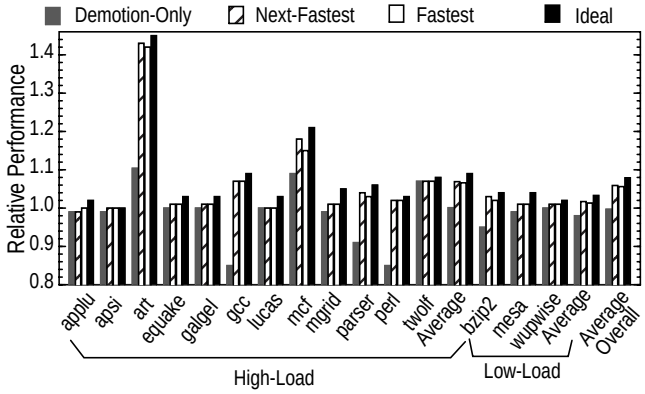

FIGURE 6: Performance of NuRAPID policies.

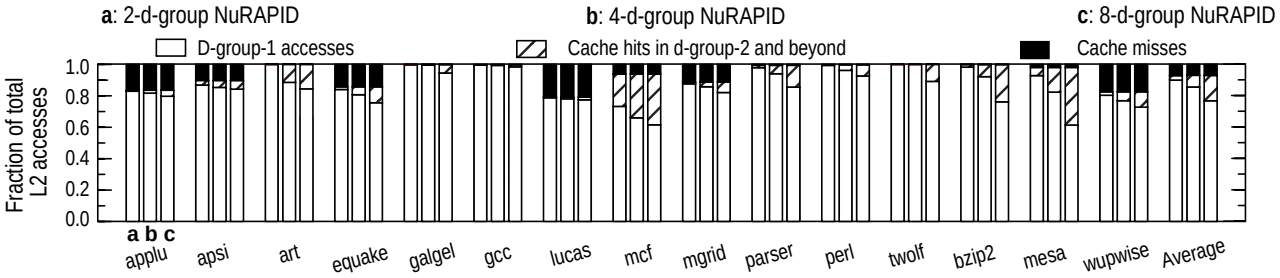

FIGURE 7: Distribution of d-group accesses for NuRAPIDs with 2,4 and 8 d-groups.

groups for 2-d-group, 4-d-group, and 8-d-group NuRAPIDs. The white portion of the stacked bars represent first-group accesses, the striped portion represents accesses to remaining d-groups and black-portion represents misses. The bar for 4-d-groups is the same as for next-fastest in Figure 5.

On average, the 2, 4, and 8-d-group NuRAPIDs have 90%, 85% and 77% accesses to the first d-group respectively. As the capacity of individual d-groups decreases, the percentage of accesses to the fastest d-group also decreases. (Note that miss rates are the same in all three cases because total cache capacity is the same.) A substantial decrease in accesses to the fastest d-group occurs between 4 and 8 d-groups because many of our applications' working sets do not fit in the 1-MB d-groups of the 8-d-group NuRAPID. In contrast, the decrease in accesses to the fastest d-group is smaller between 4 and 2 d-groups.

Figure 8 shows the performance of 2, 4, and 8-d-group NuRAPIDs relative to the base case. The 4-d-group performance numbers are the same as the ones shown in Figure 6 for the next-fastest policy. The 2-d-group NuRAPID shows only marginal improvement over the base case, whereas the 4- and 8-d-group NuRAPIDs significantly outperform the base case. On average, the 2-d-group, 4-d-group and 8-d-group NuRAPIDs perform 0.5%, 5.9% and 6.1% better than the base case respectively. The small increase in fastest-d-group accesses for the 2-d-group NuRAPID over the 4-d-group NuRAPID does not offset the increased latency of the large, 4-MB d-groups.

The 8-d-group results show that the higher latency of the small d-groups does not offset their reduced capacity. Because the 8-d-group NuRAPID has small, 1-MB d-groups, it incurs 2.2 times more swaps due to promotion compared to the 4-d-group NuRAPID, while performing only 0.2% better than the 4-d-group NuRAPID. As we will see in the next section, the additional swaps substantially increase the energy of the 8-d-group NuRAPID.

5.4 Comparison with D-NUCA

In this section, we compare the performance and energy of NuRAPID and D-NUCA. For D-NUCA energy and performance, we use the ss-energy and ss-performance policies, respectively, which were identified by [7] as *separately* optimal while we have a *single* design for NuRAPID. Both of these policies use the smart search array mentioned in Section 4. Recall that by default, D-NUCA searches for a cache block in *every* d-group. Ss-energy accesses the smart search array while only searching closest relevant d-group; if additional d-group accesses are needed, the partial tag matches narrow the search. Ss-performance uses the smart-search array to identify misses early but still searches all d-groups. If no partial-tag-match occurs in the smart-search array, a miss can be initiated before accesses to the d-group tag arrays return. Recall also the infinite switched-network and smart-search bandwidth advantage we give to D-NUCA (discussed in Section 4) and that D-NUCA is multibanked. In contrast, NuRAPID is one-ported and non-banked (Section 2.3).

5.4.1 Performance

Figure 9 shows the performance comparison of 4-d-group and 8-d-group NuRAPIDs with D-NUCA's ss-performance policy. The black bars represent the performance of D-NUCA, 4-d-group NuRAPID, and 8-d-group NuRAPID relative to the base case. The 4-d-group NuRAPID and 8-d-group NuRAPID numbers are the same as those in Figure 8.

The 4-d-group and 8-d-group NuRAPIDs outperform D-NUCA for most applications because the flexible placement policy, reduced swaps, and elimination of ss-performance's parallel tag searches offset the longer latencies of the NuRAPID d-groups. The large bandwidth provided by the multiple banks and switched network of D-NUCA is consumed by both frequent swaps and parallel tag searches on each access. (We discuss swaps, which also relate to energy, in more detail in the next subsection.) The 4-d-group and 8-d-group NuRAPIDs outperform D-NUCA by 2.9% and 3.0% respectively on average and up to 15%. The performance improvement over the base case for D-NUCA, 4-d-group NuRAPID, and 8-d-group NuRAPID are 2.9%, 5.9% and 6.0% respectively.

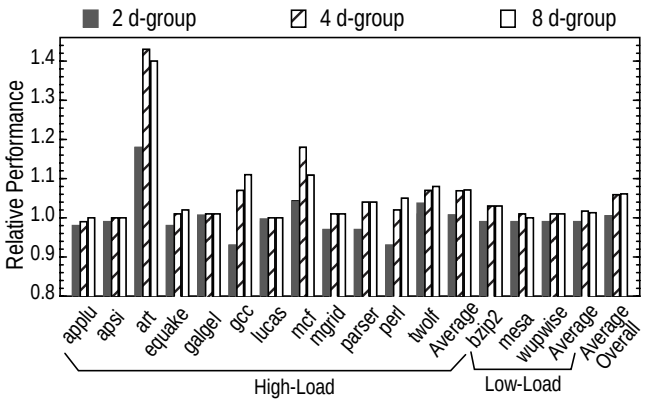

FIGURE 8: Performance of 2, 4, and 8-d-group NuRAPIDs.

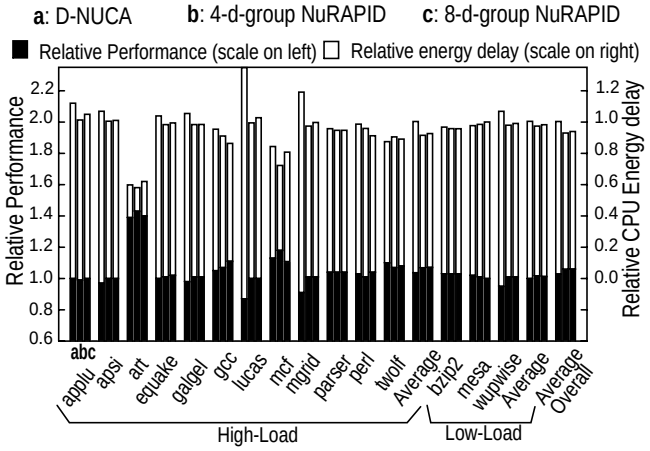

FIGURE 9: Performance and energy-delay comparison of NuRAPID and D-NUCA.

For high-load applications, NuRAPID's advantage is greater. The 4-d-group and 8-d-group NuRAPIDs outperform the base case by 6.9% and 7.1%, and D-NUCA by 3.1% and 3.3%. Low-load applications, with fewer L2 accesses, do not benefit as much. An exception is *wupwise*, which has little data reuse, preventing promotions (and fast accesses) in D-NUCA. For *wupwise* the d-a placement policy allows the 4-d-group NuRAPID to outperform D-NUCA by 6.1%.

D-NUCA outperforms NuRAPID in four applications, *applu, twolf, perl,* and *bzip2* by 1.0%,2.9%, 2.7% and 2.9% respectively*. Applu, twolf* and *perl*, despite being high-load applications, have small working sets, resulting in more frequent fast accesses for D-NUCA.

In [7] D-NUCA outperforms the base case by 9% for 70nm technology compared to our value of 2.9%. However, for most applications those results were based on simulations running only 200 million instructions which do not fill the 8-MB cache. We run 5 billion instructions for all applications to simulate cache activity more accurately. (We also ran simulations for the same number of instructions as [7] and obtained results similar to theirs.)

5.4.2 Energy

Figure 10 compares cache energy for the ss-energy D-NUCA and the 4- and 8-d-group NuRAPIDs. Below the bars, we show the average number of d-groups touched per cache access. The bars represent cache energy (including the overheads discussed in Section 4) relative to the combined energy of the base L2 and L3 caches. We show both d-group accesses and energy as their relationship is not straightforward. (For example, close and far d-groups have different access energies due to wires as shown in Table 2, and in D-NUCA some d-group accesses do not return a block because a smart-search incorrectly targeted a d-group.) However, d-group accesses provide a guide to discussing energy comparisons. We expect NuRAPID to consume substantially less energy than D-NUCA because of reduced d-group accesses and because D-NUCA's placement policy causes many accesses to far, high-energy d-groups.

D-NUCA, 4-d-group, and 8-d-group NuRAPIDs have an average of 3.1, 1.2, and 1.5 d-group accesses per cache access. The large difference between D-NUCA and NuRAPID is due to reduced swaps required under the d-a placement policy. The large reduction in d-group accesses for NuRAPID dramatically reduces bandwidth requirements, obviating the need for D-NUCA's switched network. The smaller reduction in accesses between the 4-d-group and 8-d-group NuRAPIDs occurs because of reduced swaps. This reduction is due not to placement policy but rather to the 4-d-group NuRAPID's better capture of the working-set, as described in Section 5.3.2. Relative to D-NUCA, the 4-d-group NuRAPID performs particularly well for applications shown in Section 5.2 to have the vast majority of accesses to the fastest d-group and high L2 miss rates, such as *lucas, mgrid,* and *wupwise.* For these applications, D-NUCA's initial placement causes many swaps and accesses to slow d-groups. In contrast, the 4-d-group NuRAPID initially places blocks in the fastest d-group, incurring few swaps and accesses to slow d-groups.

When our energy models are applied, D-NUCA, the 4-d-group NuRAPID, and the 8-d-group NuRAPID have relative cache energies of 4.4, 0.96, and 1.81 compared to the base. The 4-d-group NuRAPID is clearly best. The many swaps required to move blocks to the closest d-group greatly penalize D-NUCA. To a lesser extent, swaps discussed earlier penalize the 8-d-group NuRAPID, preventing net energy savings.

Finally, considering both energy and performance we show overall processor *energy-delay* in the full-height bars of

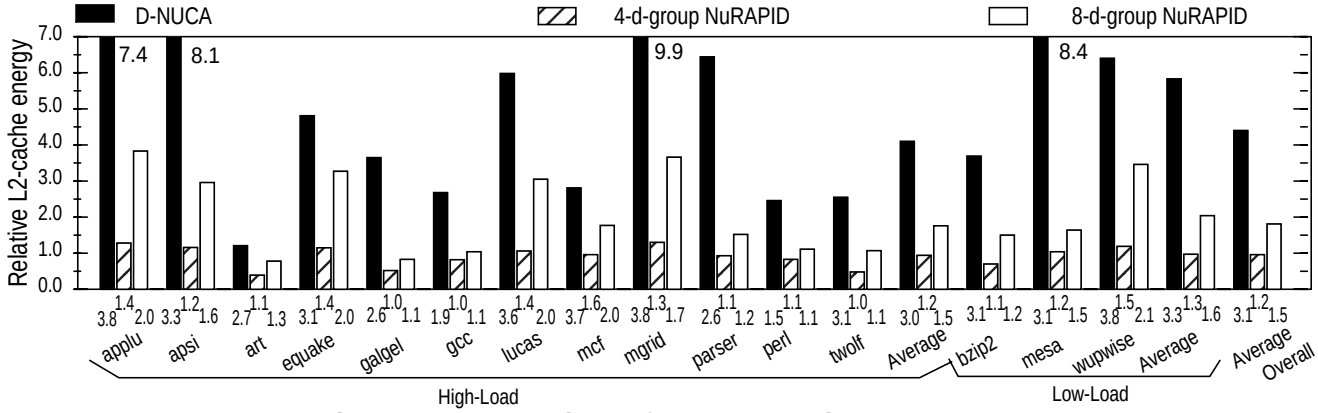

FIGURE 10: Cache energy comparison of NuRAPID and D-NUCA.

Figure 9. L2 energy is generally a small component of overall processor energy (5%-10% or less), so we expect small changes in processor energy. Compared to the base case processor, D-NUCA, the 4-d-group NuRAPID, and the 8-d-group NuRAPID have overall processor energy-delays of 1.0, 0.93, and 0.94. The small performance advantage of D-NUCA is offset by its high energy. The 4-d-group NuRAPID, taking advantage of d-a placement, large d-group capacity, and swap reduction, lowers processor energy delay by 7% compared to both the base case and D-NUCA.

6 Related Work

Other research has focused on uniform-access large cache design, such as the software-managed cache in [5] or analyzed depth of cache hierarchies to optimize performance [10]. Several papers have examined large caches in production microprocessors. The Itanium II uses a large, low-bandwidth L3 cache that is optimized for size and layout efficiency [14, 8]. Both the Itanium II L3 and Alpha 21164 L2 use sequential tag-data access [14, 3]. NuRAPID adaptively places infrequently-accessed data in slower d-groups. In addition adaptive placement in [7], others have examined adaptive caches that avoid caching low-locality blocks [4, 13] or adapt block size [6].

7 Conclusions

Large caches have design considerations different from small caches. Unlike in small caches, subarrays in large caches have variable access latency due to wire-delays. NUCA is the first proposal to exploit the variation in access time across subarrays, allowing fast access to close subarrays while retaining slow access to far subarrays. We propose "Non-uniform access with Replacement And Placement usIng Distance associativity" caches (NuRAPID) which, like NUCA, divide the cache into distance-groups (d-groups), each with its own latency. Unlike the best-performing NUCA, NuRAPID (1) uses sequential tag-data access, a common large-cache technique for low power; and (2) uses a few large d-groups to facilitate common large-cache techniques for fault-tolerance and area efficiency.

The key novelty of NuRAPID is leveraging sequential tag-data access to decouple tag and data placement. The decoupling enables flexible data placement within the large d-groups. D-a placement initially places all cache blocks in a fast d-group, while our distance-replacement policy moves only rarely-accessed blocks to slow d-groups. These policies allow the vast majority of accesses to occur to the fastest d-group with infrequent data swaps between d-groups. Thus, distance associativity fundamentally changes the trade-offs made by NUCA's best-performing design, resulting in higher performance and substantially lower energy.

Our simulations show that NuRAPID reduces the number of d-group accesses compared to NUCA by 61%, obviating NUCA's multibanking and high-bandwidth switched network. A one-ported, non-banked NuRAPID outperforms a multi-banked NUCA with an infinite-bandwidth switched network by 3% on average and up to 15% while reducing L2 cache energy by 77%. NuRAPID reduces processor energy-delay by 7% compared to NUCA. The growth of wire delays in future technologies makes NuRAPID important for future processors.

Acknowledgements

We would like to thank Doug Burger and the anonymous reviewers for their comments on an earlier draft of this paper. This research is supported in part by NSF under CAREER award 9875960-CCR, NSF Instrumentation grant CCR-9986020, DARPA contract F33615-02-1-4003, and an NSF Graduate Research Fellowship.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[2] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[3] J. H. Edmondson et al. Internal organization of the Alpha 21164, a 300-MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal*, 7(1), 1995.

[4] 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.

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

[6] T. L. Johnson and W. W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *24th International Symposium on Computer Architecture*, pages 315–326, July 1997.

[7] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, pages 211–222, Oct. 2002.

[8]S. D. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski. The implementation of the Itanium 2 microprocessor. *IEEE Journal of Solid-State Circuits*, 37(11):1448–1460, Nov. 2002.

[9] M. D. Powell, A. Agarwal, T. N. Vijaykumar, and B. Falsafi. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO 34)*, pages 54–65, Dec. 2001.

[10] S. A. Przybylski. Performance-directed memory hierarchy design. Technical Report 366, Stanford University, Sept. 1988.

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

[12] A. J. Smith. Cache memories. *ACM Computing Surveys (CSUR)*, 14(3):473–530, 1982.

[13] 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.

[14] D. Weiss, J. J. Wuu, and V. Chin. The on-chip 3-mb subarray-based third-level cache on an Itanium microprocessor. *IEEE Journal of Solid-State Circuits*, 37(11):1523–1529, Nov. 2002.