CS/ECE 752 - Advanced Computer Architecture
Prof. Mikko Lipasti
Fall 2003

Project-2 Report

Non Uniform Cache Architectures For Wire Delay Dominated Caches

THE UNIVERSITY of WISCONSIN MADISON

By
Abhishek Desai
Bhavesh Mehta
Devang Sachdev
Gilles Muller

Department of Electrical and Computer Engineering
University of Wisconsin, Madison

TABLE OF CONTENTS

Fall 2003 1
Project-2 Report 1
Non Uniform Cache Architectures 1
For Wire Delay Dominated Caches 1
Department of Electrical and Computer Engineering 1
1. Introduction 4
1.1 Motivation 4
1.2 UCA and ML-UCA 4
2. Static Non Uniform Cache Architecture 5
2.1 Static-NUCA 1 5
2.2 Static-NUCA 2 6
3.1 Mapping 8
3.2 Searching 10
3.3 Dynamic movement 10
4. Results and Analysis 12
4.1 CACTI Results 13
4.2 Simulated Cache Configuration 14
4.3 Performance Evaluation 14
4.3.1 IPC comparison 14
4.3.2 Miss Rate comparison 17
4.3.3 Hit Rate Distribution 18
5. Related Work Survey 20
6. Summary and Conclusions 21
7. Division of Work Effort 21
8. References 22

LIST OF FIGURES

Figure 1: UCA and ML-UCA 4
Figure 2: D-NUCA 8
Figure 3: Simple Mapping policy 8
Figure 4: Fair Mapping policy 9
Figure 5: Shared Mapping policy 9

LIST OF TABLES

Table 1: Results obtained from CACTI for UCA 13
Table 2: Results obtained from CACTI for NUCA 13
Table 3: IPC Results SPEC INT 2000: UCA vs D-NUCA (16x8) 15
Table 4: IPC Results SPEC INT 2000: UCA vs D-NUCA (16x4) 15
Table 5: IPC Results SPEC FP 2000: UCA vs D-NUCA (16x8) 16
Table 6: IPC Results SPEC FP 2000: UCA vs D-NUCA (16x4) 16
Table 7: Miss Rate Results SPEC INT 2000: UCA vs D-NUCA (16x4) 17
Table 8: Miss Rate Results SPEC FP 2000: UCA vs D-NUCA (16x4) 18

1. Introduction

We studied and verified the paper "An Adaptive , Non - Uniformed Cache Structure for Wire - Delay Dominated On-Chip Caches" by Changkyu Kim, Doug Burger and Stephen W. Keckler [1].

Wire delays continue to grow as the dominant component of latency for large 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. This report mainly discusses the issues involved with Dynamic - NUCA. We basically implemented the D-NUCA into the Sim – alpha simulator and ran various integer and floating point benchmarks on it to compare the D-NUCA IPC with the IPC of the default cache (UCA) in the Sim-alpha simulator. We found that IPC does increase with D-NUCA although the IPC of the default UCA was rather optimistic since wire delays and latch delays were not accounted for in the default design.

1.1 Motivation

The future trend for L2 and L3 caches is to get bigger because of reasons like:

1. Bigger Program sizes.
2. Multithreading requires large caches for spatial locality.
3. Bandwidth demands have increased on the package.
4. Smaller technologies permit more bits per mm^2.

These large caches will have very large latencies due to increasing global wire delays across the chip. Bulk of the access time will involve routing to and from the banks, not the bank accesses themselves. 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 differences in latencies among subarrays. Hence there is a need for non-uniform cache architectures where the cache is broken into banks that can be accessed at different latencies.

1.2 UCA and ML-UCA

Figure 1: UCA and ML-UCA

Large modern Uniform Caches are subdivided into multiple banks to reduce 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.

Even multilevel caches are posing problems now-a-days due to large access latencies. Also care has to be taken to ensure cache coherence in multilevel caches. Since each level obeys inclusion (for cache coherency) there is redundancy of data and thus space is not efficiently utilized.

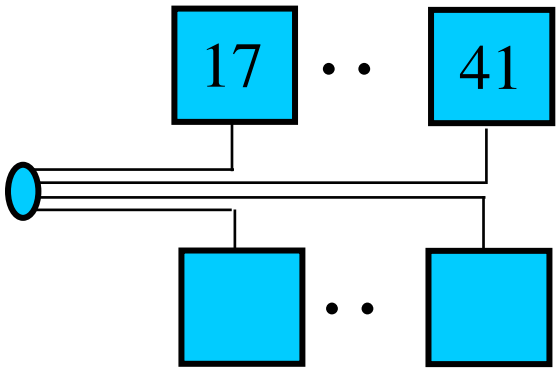
2. Static Non Uniform Cache Architecture

Much performance is lost by requiring worst-case uniform access in a wire-delay dominated cache. If each bank of the cache can be accessed at different speeds, proportional to the distance of the bank from the cache controller, performance would improve. Indeed, on the figure below, the closest banks can be accessed in 17 clock cycles, compared to the worst-case 41 cycles delay.

Data are statically mapped into banks, with the low-order bits of the index determining the bank. 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.
These caches are called Static-NUCA since the mappings of data to banks are static, and banks have non-uniform access times.

2.1 Static-NUCA 1

As shown in the figure below, each addressable bank in the S-NUCA-1 organization has private address and data channels. 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. With this architecture, port and channel conflicts are considerably reduced. But bank conflicts can still occur.



S-NUCA-1

Avg. access time: 34 cycles
Banks: 32
Size: 16MB
Technology: 50nm
Area: Wire overhead 20.9%

Figure 2:S-NUCA-1

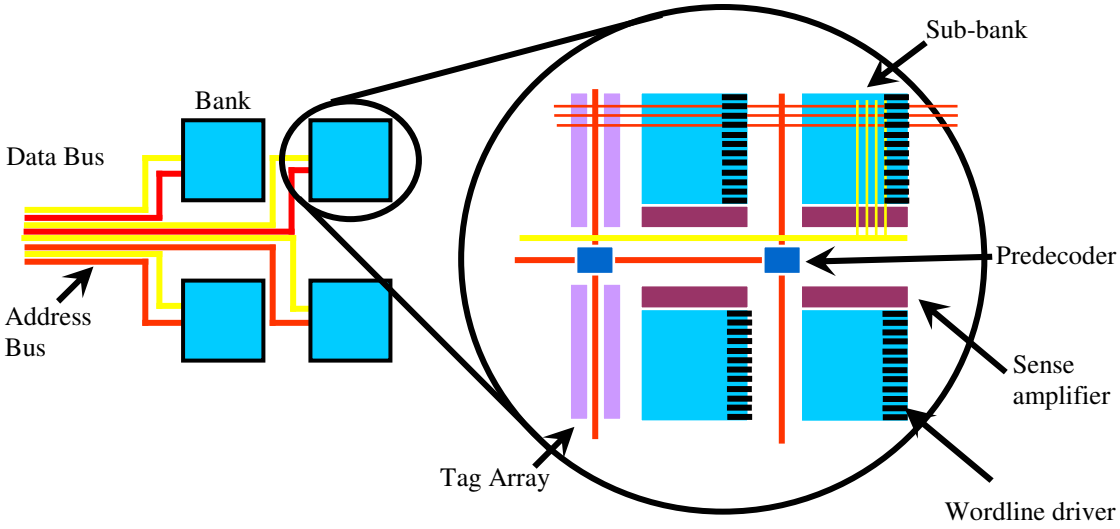
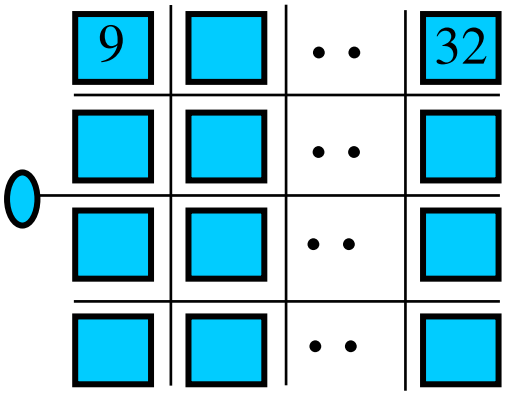


Figure 3: S-NUCA-1 cache design

Assuming that the cache controller resides in the middle of one side of the bank array, the farthest distance that must be traversed is half of one dimension and the entire other dimension. The major bottleneck of these SNUCA-1 caches is the increased routing delays and wire area overhead due to the private channels. For example, with a 50nm technology, the wire area overhead is 20.9% for a 16MB cache. S-NUCA 2 is a solution to this problem.

2.2 Static-NUCA 2

This organization removes most of the large number of wires resulting from per-bank channels. It embeds a 2-D mesh with point-to-point links in the cache, placing simple switches at each bank. Each link has two separate channels.



S-NUCA-2

Avg. access time: 24 cycles
Banks: 32
Size: 16MB
Technology: 50nm
Area: Channel overhead 5.9%

Figure 4: S-NUCA-2

But we can't have free lunch. This organization increases contention probability. To reduce it to a low level, we need to introduce buffers.
Each bank contains a larger buffer able 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, and delaying other requests requiring those switches. Of course, other banks along different network paths could still be accessed in parallel.

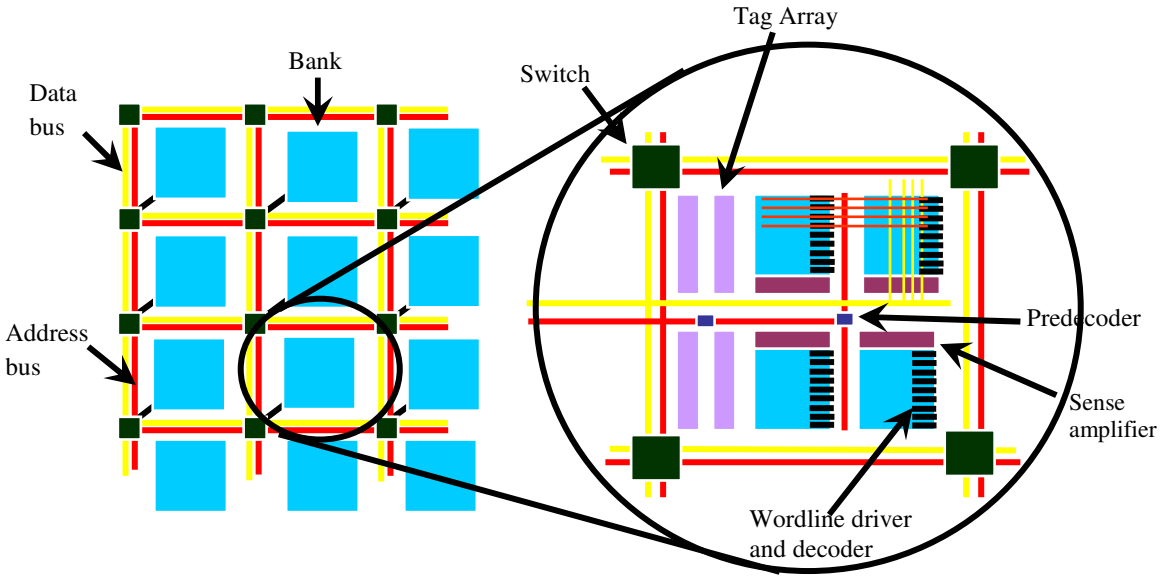
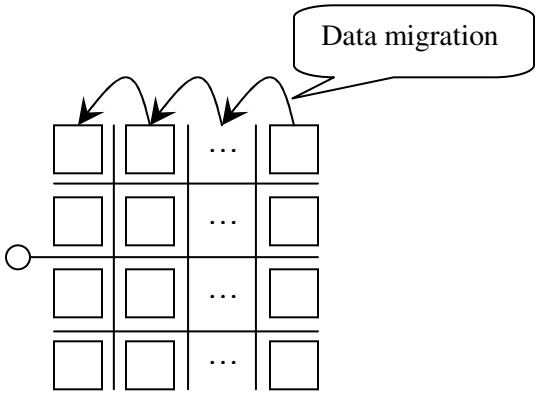


Figure 5: S-NUCA-2 cache design

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, 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% of the total area of the banks. Thus, the S-NUCA-2 cache is faster at every technology than SNUCA-1, and furthermore, the average loaded latency found in the paper is 24.2 cycles, as opposed to 34.2 cycles for S-NUCA-1.

But we can improve the latency using a dynamic mapping.

3. Dynamic Non Uniform Cache Architecture



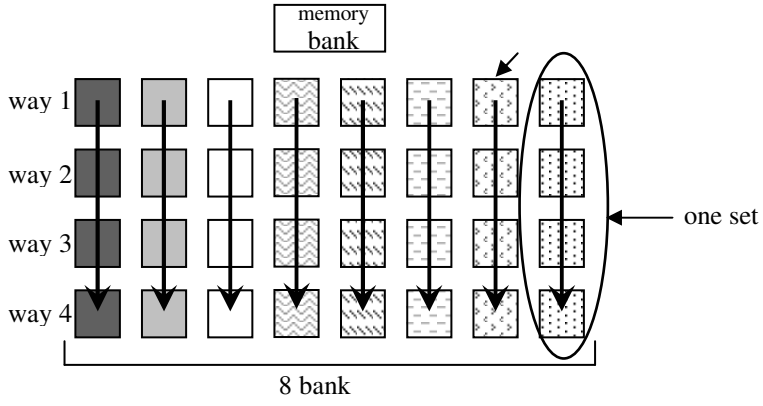
D-NUCA
Figure 6: D-NUCA

Dynamic-NUCA gets its name because data is dynamically mapped to the cache. Frequently accessed data migrates to closer (faster) banks and less important – yet still cached – data is moved to farther banks. Three issues have to be addressed when exploring the design space for D-NUCA.

1. Mapping:
   How the data are mapped to the banks and in which banks a datum can reside?
2. Searching:
   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?

3.1 Mapping

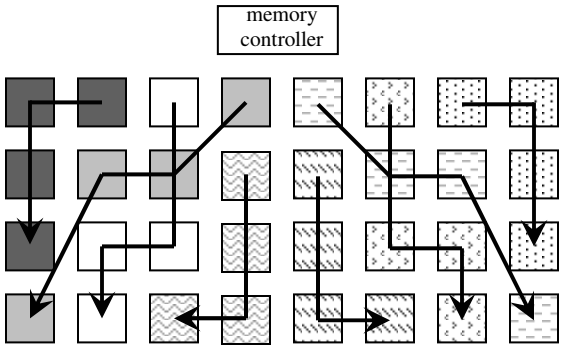
In ***Simple Mapping***, each column of banks in the cache becomes a bank set, and all banks within that column comprise the set-associative ways. The drawback of this scheme are that the number of rows may not correspond to the number of desired ways in each bank set, and that latencies to access all bank sets are not the same.



Simple Mapping

Figure 7: Simple Mapping policy

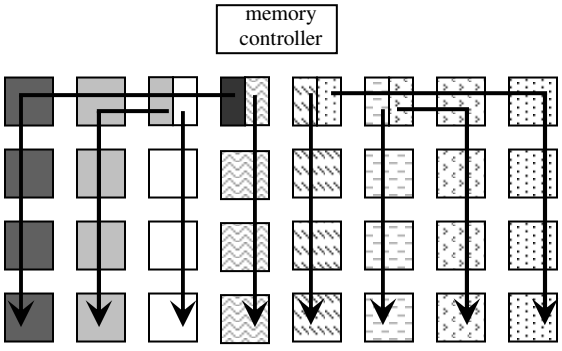
The ***Fair Mapping*** policy addresses the problems of the simple mapping at the cost of additional complexity. With this model, banks are allocated to bank sets so that the average access time across all bank sets are equalized. However the disadvantage is a more complex routing path from bank to bank within a set, causing potentially longer routing latencies and more contention in the network.



Fair mapping

Figure 8: Fair Mapping policy

The ***Shared mapping*** policy 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. This 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.



Shared mapping

Figure 9: Shared Mapping policy

3.2 Searching

Different search techniques can be applied to find the particular line in the cache.

The simplest of all is the *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. Since fewer banks are accessed while checking for a hit, the energy consumption is less at the cost of reduced performance.

The second method is called *multicast search*, in which the requested address is multicast to some or all of the banks in the requested bank set. This scheme offers higher performance since hits to banks far from the processor will be serviced faster. However extra address bandwidth will be needed as the address is routed to each bank and this might slow other accesses. This scheme has been implemented in the code.

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. 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.

There are some smart search techniques that can be applied to decrease the miss resolution time. Here, we store the partial tag bits into a smart search array located in the cache controller. In the *ss-performance* technique, 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. Care must be taken to ensure that the smart search array contains enough of the tag bits per line so that probability of a false hit is reduced.

In the *ss-energy* technique, the closest bank and the partial tags are searched in parallel. If the closest bank misses, we go to the location(s) specified by the partial tag match. This reduces the number of banks to be searched and hence conserves energy.

3.3 Dynamic movement

Policy on hit: Every time a hit occurs the lines of the cache can be arranged so that the closest bank holds the MRU line and the farthest bank holds the LRU line. However this would result in heavy movement of lines among banks and might interfere with normal data access.

To reduce heavy movement of lines we follow the policy of one-bank promotion on each hit. When a hit occurs to a cached line, it is swapped with the line in the bank that is the next closest to the cache controller. This policy has been implemented in the code.

Policy on miss: The Line in the furthest (slowest) bank is always chosen for replacement. This policy is implemented in the code. The new line that is brought into the cache is placed in the furthest (slowest) bank. It is promoted to nearer banks depending on the

subsequent number of hits to it. The victim line chosen for replacement is evicted to the main memory. This is called the zero copy policy and has been implemented in the code.

One can implement the one copy policy also where in the victim line is not removed from the cache but placed in a lower priority bank. This policy works well when the new line brought into the cache is placed in the closest (fastest) bank.

4. Results and Analysis

Alpha 21264 processor was used with the same configuration as used in the paper to verify the results presented. We simulated the processor and the L2 cache using alpha-sim 1.0 to do comparison analysis of D-NUCA with UCA.

We modified the L2 cache and the controller in sim-alpha to carry out simulations for D-NUCA (modified code attached as appendix).

Here we present some of the critical data structures and methods added to support base D-NUCA. Several other files related to cache structure, cache access and cache timing were also modified. We have commented our code quite well and attached as **Appendix-I.** We have also attached configuration file for the CPU and memory in the appendix.

Data Structures:

- int read_req_map [16][16]; // request map
- int read_rpl_map [16][16]; // replacement map
- static tick_t prev_cycle;
- counter_t read_req_conflicts; //number if read request conflicts
- counter_t write_req_conflicts; //number if write request conflicts
- counter_t read_rpl_conflicts; //number if replacement conflicts
- counter_t total_hitlat; // total hit latency
- counter_t total_hitcnt; //total number of hits
- counter_t total_misslat; //total miss latency
- counter_t total_misscnt; //total number of misses

// The following structures stores entire D-NUCA in a grid.

- struct cachebanks_type *cachebanks[MAX_ROW][MAX_COL];
- int data_ACC;
- int tag_ACC;
- int cachebanks_row;
- int cachebanks_col;
- counter_t sector_hits[16]; //stores sector-wise hits

Methods:

- void all_map_clear(int x, int y); // clears all the mappings (req, rpl)
- void routing_func (loc_type curr_loc, loc_type dest_loc, loc_type *next_loc, enum bank_packet_type packet_type); //returns 1 when packet arrives at destination.
- int is_arrived (loc_type curr_loc, loc_type dest_loc);//checks whether packet has arrived or not

// The following method creates bank access packets.

- bank_acc_packet_type * create_bank_acc_packet (cache_access_packet *c_packet, loc_type curr_loc, loc_type req_dest_loc, loc_type target_dest_loc, loc_type origin_dest_loc, enum bank_packet_type packet_type);

- void bank_network_traverse(tick_t now, bank_acc_packet_type *bank_acc_packet); //traverses network of bank according the determined policy.
- void response_packet_insert (int req_dest_x, int req_dest_y, struct cache *cp, tick_t now); //inserts the response packet
- void miss_packet_insert(int req_dest_x, int req_dest_y, cache_access_packet *new_c_packet, tick_t tlb_lat, tick_t now, int membus_lat, struct cache *cp, enum resource_type type); //inserts the miss packet
- void hit_packet_insert (int req_dest_x, int req_dest_y, cache_access_packet *c_packet, tick_t tlb_lat, tick_t now); //inserts hit packet
- void wb_packet_insert (int req_dest_x, int req_dest_y, struct cache *cp, tick_t now); //inserts write back packet
- void find_sector_hits (struct cache *cp, md_addr_t set, md_addr_t tag, int csize, int is_write); //finds sector hits and increments corresponding index.
- int is_way_full (struct cache *cp, md_addr_t set); //checks whether particular way of the set is full.
- struct cache_blk * find_next_empty_blk (struct cache_blk *head_blk); //finds next empty block.
- int find_blk_pos (struct cache_blk *head_blk, struct cache_blk *curr_blk) ; //finds block position in integer.
- void create_all_cachebanks (int row, int col); //creates all cachebanks

4.1 CACTI Results

CACTI 3.0 was used to simulate the UCA and the NUCA in order to determine the access time delays for different rows of the cache implemented in different technologies.

Table 1: Results obtained from CACTI for UCA

Technology	L2 Size	Unloaded Latency
130 nm	2 MB	8
100 nm	4 MB	12
70 nm	8 MB	28
50 nm	16 MB	47

Table 2: Results obtained from CACTI for NUCA

Technology	L2 Size	Rows x Columns	Unloaded Latency		
			Min	Max	Avg.
130 nm	2 MB	4 x 4	4	8	6
100 nm	4 MB	8 x 4	4	12	9
70 nm	8 MB	16 x 8	4	28	17
50 nm	16 MB	16 x 8	3	47	25

4.2 Simulated Cache Configuration

The cache configuration for the UCA L2 cache was:

- Size – 16 MB
- Technology – 50 nm
- Replacement policy – LRU
- No of banks – 32

The cache configuration for the D-NUCA L2 cache was:

- Size – 16 MB
- Technology – 50 nm
- No. of Rows x Columns – 16 x 8 / 16 x 4
- Replacement policy – replace block from the tail of the cache in the last row
- Mapping – Simple mapping
- Search policy – Multicast search
- Promotion policy – migration to the next closest row towards the processor on hit

4.3 Performance Evaluation

4.3.1 IPC comparison

Programs from SPEC INT 2000 and SPEC FP 2000 were simulated and the following IPC results were obtained for UCA vs. D-NUCA (16x8). Only that phase of the program where the accesses to L2 cache were maximized and repeated accesses to the same location in L2 cache was simulated by fast forwarding on an average 2 billion instructions for each benchmark. On average 200 – 500 million instructions were run for each benchmark. We observed that the IPC of the D-NUCA cache was higher in almost all the benchmarks tested. The obtained results showed similar performance improvements as the paper claimed. The results obtained for UCA were comparatively optimistic. The reason behind this was that the UCA cache that we used had not been modeled using parameters obtained from CACTI. Hence the wire delays; dominant at 50 nm scale were not considered accurately. Moreover, the pipeline latch delay for the UCA was also not taken into account.

Table 3: IPC Results SPEC INT 2000: UCA vs D-NUCA (16x8)

specint2000	176.gcc	181.mcf	197.parser	253.perlbmk	256.bzip2	300.twolf
uca	0.893	0.5568	0.8587	0.836	1.3131	0.8431
nuca	0.9606	0.8284	0.9062	0.9611	1.3187	0.8635

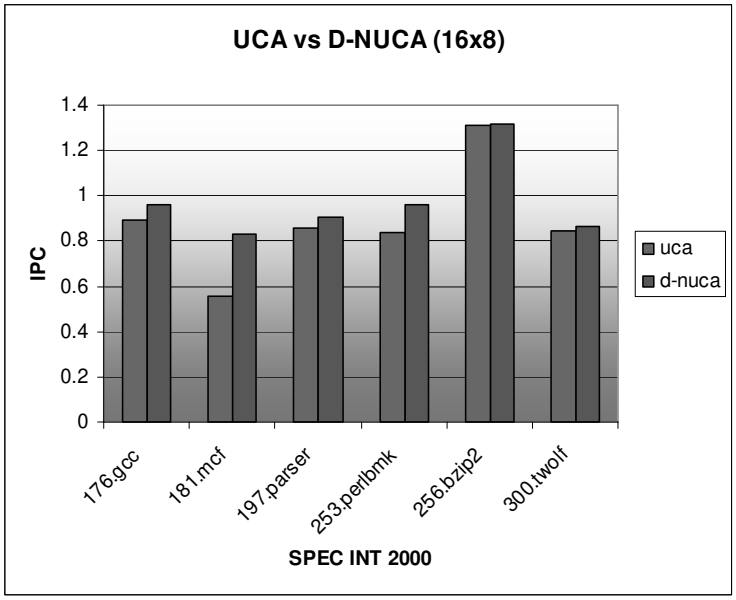


Table 4: IPC Results SPEC INT 2000: UCA vs D-NUCA (16x4)

specint2000	176.gcc	197.parser	256.bzip2	300.twolf
uca	0.9061	0.8531	1.3027	0.8461
nuca	0.9966	1.0101	1.4694	0.9621

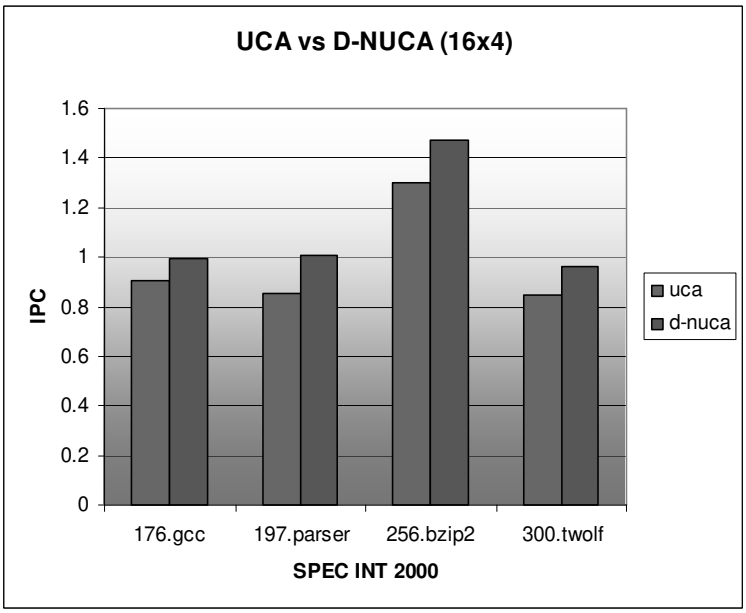


Table 5: IPC Results SPEC FP 2000: UCA vs D-NUCA (16x8)

specfp2000	172.mgrid	173.applu	177.mesa	178.galgel	179.art	183.equake
uca	0.7417	0.4526	1.1102	0.6785	0.581	0.9666
d-nuca	0.6432	0.7015	1.2973	1.6423	0.6541	1.2653

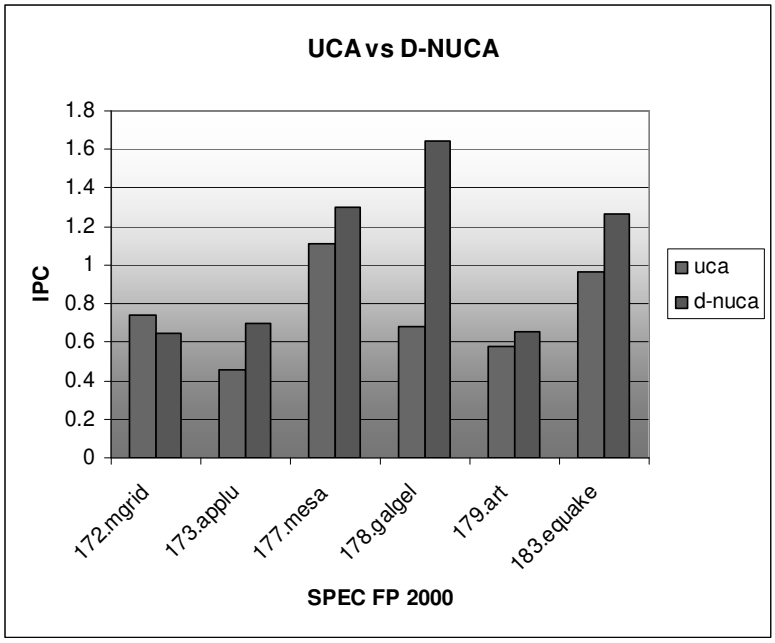
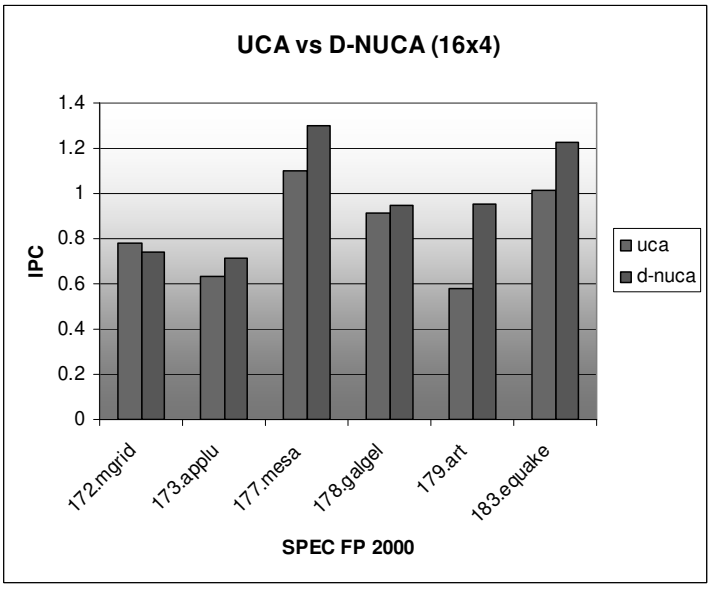


Table 6: IPC Results SPEC FP 2000: UCA vs D-NUCA (16x4)

specfp2000	172.mgrid	173.applu	177.mesa	178.galgel	179.art	183.equake
uca	0.7832	0.6337	1.1028	0.9105	0.581	1.0123
d-nuca	0.7368	0.7145	1.299	0.9435	0.9561	1.2248



Similar pattern of higher IPC for D-NUCA was observed in the FP benchmarks.

4.3.2 Miss Rate comparison

As expected the miss rate for the NUCA is much higher than that of the UCA. The replacement policy implemented in NUCA is replacement at the tail. This policy is an approximate LRU policy. As compared to the UCA which is implemented using LRU policy, the NUCA is bound to give a higher miss rate. This could lead to a higher traffic on the mem bus and hence would need a higher bandwidth. We have identified this as one of the problems associated with the NUCA. This however can be resolved by using a small 2 or 4 entry cache (similar to victim cache) which would store the evicted line. However this remains a topic of further research.

Table 7: Miss Rate Results SPEC INT 2000: UCA vs D-NUCA (16x4)

specint2000	176.gcc	197.parser	256.bzip2	300.twolf
miss rate uca	0.0065	0.0578	0.0425	0
miss rate d-nuca	0.022	0.1083	0.03	0.025

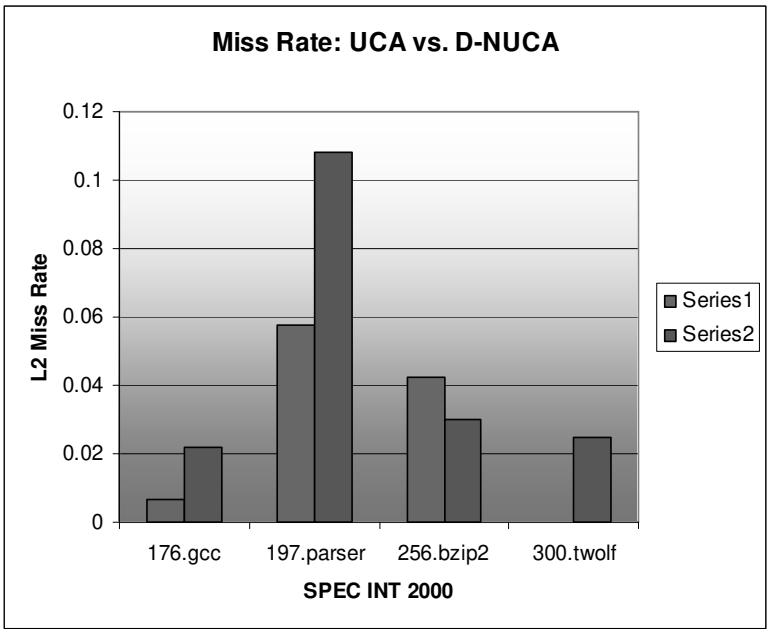
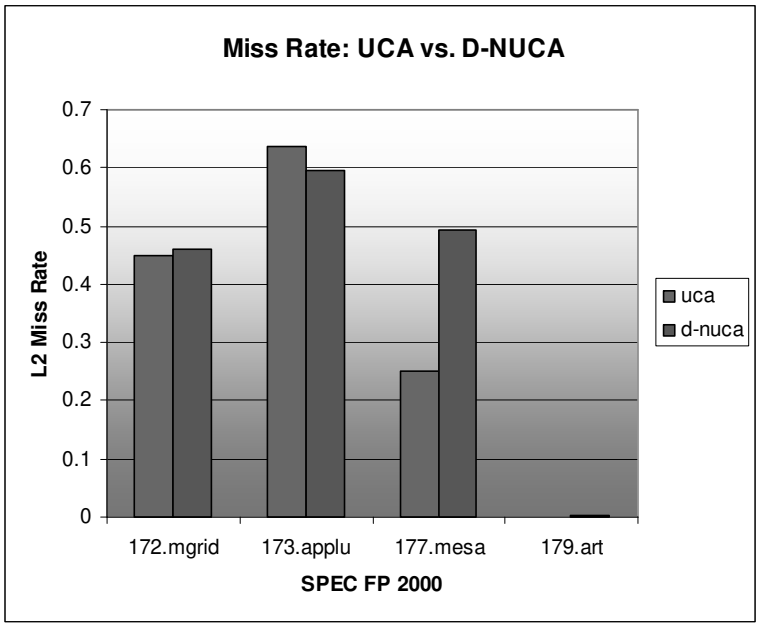


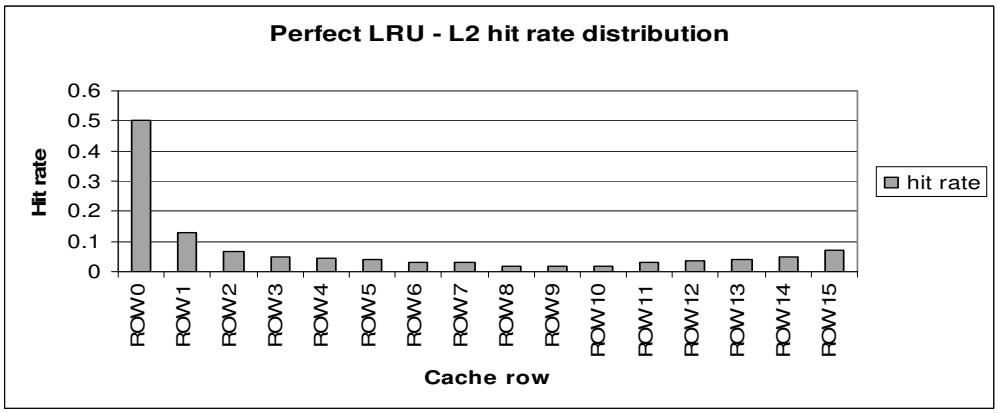
Table 8: Miss Rate Results SPEC FP 2000: UCA vs D-NUCA (16x4)

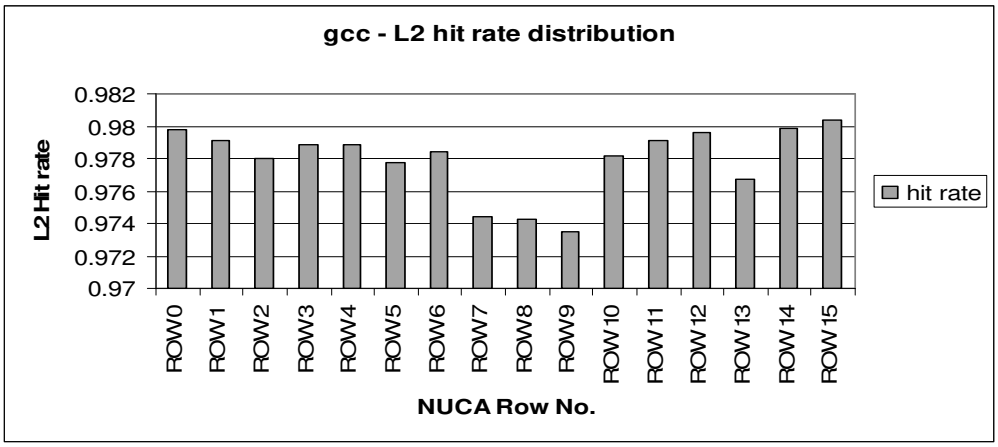
specfp2000	172.mgrid	173.applu	177.mesa	179.art
miss rate uca	0.4491	0.6374	0.2497	0.0011
miss rate d-nuca	0.4615	0.5958	0.4928	0.0018

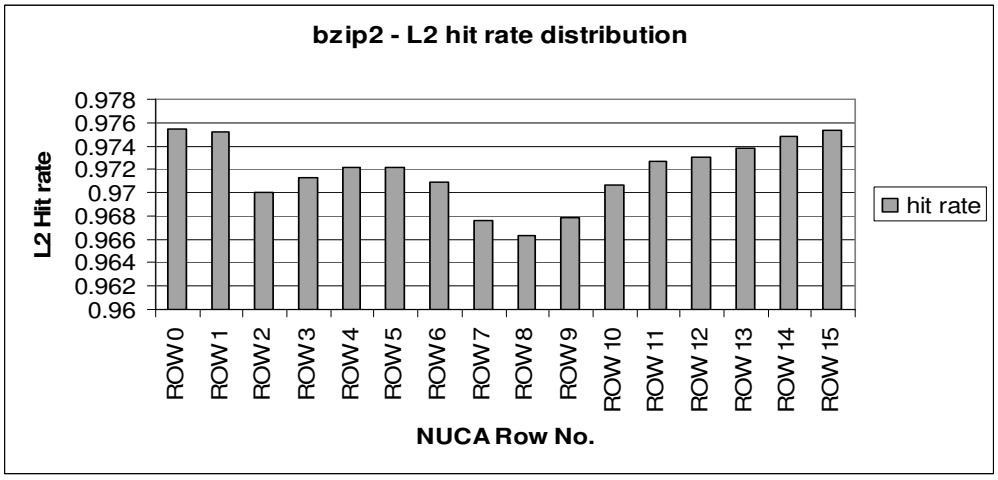


4.3.3 Hit Rate Distribution

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 a 16-way bank set. The distribution of hits to the various sets for each cache, for the gcc and bzip2 benchmark. For both caches, most hits are concentrated in the first two ways of each set.




gcc - L2 hit rate distribution
L2 Hit rate
0.982
0.98
0.978
0.976
0.974
0.972
0.97
ROW0
ROW1
ROW2
ROW3
ROW4
ROW5
ROW6
ROW7
ROW8
ROW9
ROW10
ROW11
ROW12
ROW13
ROW14
ROW15
NUCA Row No.
hit rate


bzip2 - L2 hit rate distribution
L2 Hit rate
0.978
0.976
0.974
0.972
0.97
0.968
0.966
0.964
0.962
0.96
ROW0
ROW1
ROW2
ROW3
ROW4
ROW5
ROW6
ROW7
ROW8
ROW9
ROW10
ROW11
ROW12
ROW13
ROW14
ROW15
NUCA Row No.
hit rate

5. Related Work Survey

There has been earlier work on large caches designs but which have uniform access latency. For example fully associative software managed caches have been researched by Hallnor and Reinhardt [11]. Other researchers have examined using multiple banks for high bandwidth to reduce contention. Sohi and Franklin [9] proposed interleaving banks to create ports, and also examined the need for L2 cache ports on less powerful processors than today's. However, D-NUCA aims to flatten deepening hierarchies

There has also been research on reducing power consumption in large caches. Albonesi [10] found that ways of the set can be turned off when the demand on the cache is reduced.

Many researchers have examined adaptive cache policies, a concept which is inherent in the D-NUCA organization. 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. Gonzfilez, Aliagas, and Valero [8] examined adaptive caches that avoid caching low-locality blocks, or adapt block size.

Recently T. N. Vijaykumar et al [2] have proposed the NuRAPID (Non-uniform access with Replacement And Placement usIng Distance associativity) cache, which leverages sequential tag-data access to decouple data placement from tag placement.

The Itanium II uses a large, low-bandwidth L3 cache that is optimized for size and layout efficiency. Both Itanium II L3 and Alpha 21164 L2 use sequential tag data access.

6. Summary and Conclusions

Non-uniform accesses have started to appear in high performance cache designs. In this project, we implemented several designs that treat the cache as a network of banks and facilitates non uniform accesses to different physical regions. By doing a comparison analysis we saw 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 25 cycles at an 8 FO4 clock, which is a lower absolute latency than conventional L2 caches (UCA) with average access latency of 47 cycles.
- ***Technology scalability***: 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 6 cycles for a 2MB, 180nm cache to 25 cycles for a 16MB, 50nm cache rather than the UCA scaling from 8 to 47.
- ***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.

We also identified certain problems associated with the NUCA. The first was the no. of ports on the cache would need to be high in order to reduce bank contention. This would make the cache access slower and the cache size bigger. This problem could be resolved using switched channels but would lead to complex control for the cache. Moreover on promotion the swapping of the data from one to another would potentially lead to higher movement internal to the cache. But as the accesses to the L2 cache are very limited this swapping function could be carried out when the cache is not being accessed by the processor. The placement of the tag array would also dominate the access time of the cache hence some of the recent papers have proposed to place the centralized tag array closer to the processor for faster determination of a hit/miss

7. Division of Work Effort

Name	Contribution	Signature
Abhishek Desai	25%	_____
Bhavesh Mehta	25%	_____
Devang Sachdev	25%	_____
Gilles Muller	25%	_____

8. References

1. Changkyu Kim, Doug Burger and Stephen W. Keckler. *An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches*.
2. Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar. *Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures*.
3. John Paul Shen and Mikko H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, Beta Edition.
4. Mark Hill, Norm Jouppi, and Guri Sohi. *Readings in Computer Architecture.* Morgan Kauffman,1999.
5. The national technology roadmap for semiconductors. Semiconductor Industry Association, 1999.
6. P. Shivakumar and N. P. Jouppi. CACTI 3.0: An integrated cache timing, power and area model. Technical report, Compaq Computer Corporation. August 2001.
7. R. Desikan, D. Burger, S. W. Keckler, and T.M. Austin. Sim-alpha: A validated execution-driven alpha 21264 simulator. Technical report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.
8. A. Gonzfilez, 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.
9. 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.
10. 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.
11. 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.