Improving Power Efficiency of D-NUCA Caches

A. Bardine, P. Foglia,
G. Gabrielli, C.A. Prete
Dip. di Ingegneria dell'Informazione
Facoltà di Ingegneria, Università di Pisa
Via Diotisalvi, 2 – 56126 PISA (Italy)
{alessandro.bardine, foglia,
giacomo.gabrielli, prete}@iet.unipi.it

P. Stenström
Dep. of Computer Science and Engineering
Chalmers University of Technology
S-412 96 Gothenburg (Sweden)
pers@ce.chalmers.se

Members of the HIPEAC EU Network of Excellence

Abstract

D-NUCA caches are cache memories that, thanks to banked organization, broadcast search and promotion/demotion mechanism, are able to tolerate the increasing wire delay effects introduced by technology scaling. As a consequence, they will outperform conventional caches (UCA, Uniform Cache Architectures) in future generation cores.

Due to the promotion/demotion mechanism, we have found that, in a D-NUCA cache, the distribution of hits on the ways varies across applications as well as across different execution phases within a single application. In this paper, we show how such a behavior can be utilized to improve D-NUCA power efficiency as well as to decrease its access latencies. In particular, we propose a new D-NUCA structure, called Way Adaptable D-NUCA cache, in which the number of active (i.e. powered-on) ways is dynamically adapted to the need of the running application. Our initial evaluation shows that a consistent reduction of both the average number of active ways (42% in average) and the number of bank access requests (29% in average) is achieved, without significantly affecting the IPC.

1. Introduction

CMOS trends and bandwidth demands of cores are leading to the use of large, on-chip, level-two (L2) and level-three (L3) cache memories. For high clock frequency designs, the latencies of such caches are dominated by wire delays [1]. In order to reduce the effects of such latencies, NUCA Caches (Non-Uniform Cache Architectures) [2], [3] have been proposed as a new paradigm for on-chip L2 cache memories.

NUCA caches. In a NUCA architecture, the cache is partitioned into many independent banks while the communication among banks and with the controller is supported by a switched network. Such organizations allow banks to be closer to the processor, hence allowing shorter access latency to the local banks compared to banks that are farther away. Mapping between cache lines and physical banks can be either Static or Dynamic (namely S-NUCA and D-NUCA) [2]. In the former, each line can exclusively reside in a single predetermined bank and the whole cache is used like a direct mapped cache; in the latter (Figure 1), each cache line is mapped to different banks as in a traditional set associative cache, while the cache lines can dynamically migrate from one bank to another. As shown in Figure 1, in a D-NUCA cache the banks are logically grouped in rows and columns, each bank containing a fixed number of lines. The entire addresses space is mapped on the banks belonging to each row. Banks belonging to the same columns behave like different ways of a set associative memory, i.e. a cache line may reside in each bank of the column. When a search is performed, the controller first determines the column that can contain the searched data using a direct mapping law, then it broadcasts the request to all the banks belonging to such column. As soon as a hit happens in one of the column banks, the search is satisfied without the need of waiting for the replies from the other, farther, column banks. To reduce latencies, the "promotion/demotion" mechanism is adopted: if a hit happens in a row that is other than the first, the data line is promoted by swapping it with the line that holds the same column position in the next row closer to the controller. If a miss happens, the new line is inserted in the farthest bank (row 7 in Figure 1) evicting the corresponding data line. As a consequence, the most recently accessed lines are “promoted” in the banks closer to the processor improving the performance. With such a policy, D-NUCA caches succeed in achieving high hit rates while keeping the access latencies low, in spite of the wire-delay effects introduced by high clock rates and technology scaling. These characteristics make D-NUCA an attractive cache architecture for next generation high performance CPUs, where large storage capabilities, high clock rates and low access latencies will be required.

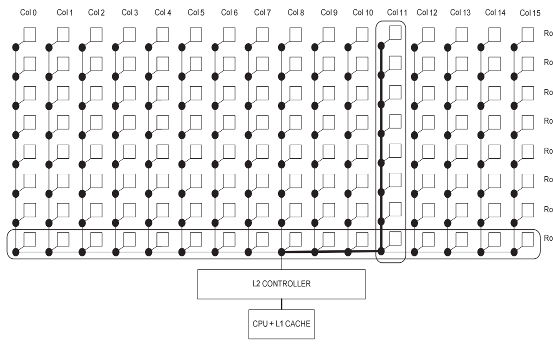

Figure 1: The reference D-NUCA cache used in our work. Such an 8-Mbyte D-NUCA cache uses 128 banks (the squares in the picture) organized in rows and columns; each row represents a way so that the cache behaves like an 8-way set associative cache. The rounded corner contours highlight a single row and a single column of banks. The bold line shows the path followed by requests for a search: from the address, the controller determines the column to be searched (the 11th in the figure), a request is sent along the horizontal link till the column, then the request moves along the column. At each switch (the black circles), the request is replicated to perform the search in the bank.

Problem. As a consequence of the promotion/demotion mechanism, we observed that in a D-NUCA cache, the hits are not uniformly distributed along the ways. Instead, their distribution shows strong variations among different applications as well as during different execution phases of the same application. Such a behavior suggests that not all the cache ways are needed during the whole execution of an application: due to their poor usage, the unused or less used ways may be powered-off without significantly affecting performance. Accordingly, it can be achieved a reduction of the cache static power consumption that, according to technology projections [5], will account for a greater fraction of the total chip power consumption. Moreover, powering off some ways may also be useful to reduce both the miss detection time and the network traffic, as there are fewer banks that must be searched in order to detect a miss. This may have beneficial effects on performance as well as on dynamic power consumption.

Contribution. In this work, we contribute with Way Adaptable D-NUCA Caches that leverage our observation on hit distribution across the ways. In such a cache, a mechanism is introduced to dynamically turn on or off ways, according to the needs of the running application. In particular, our contribution can be stated as follows: 1) we show that it is possible to dynamically control the ways in a D-NUCA cache, and it is promising, as it implies a reduction of the power consumption, cache access time, and network traffic with a negligible performance loss (42% in average of reduction of active ways, 29% of reduction in bank access requests, 3% of reduction in cache access latency, with an average degradation of 0,38% for the IPC); 2) we develop a scheme to control the number of active ways in such a cache, that differs from proposed techniques [12], as: a) it is the first to be applied to a L2 D-NUCA cache, b) its metric is based on the intrinsic D-NUCA data line ordering thus not requiring the complex additional logic that previously utilized LRU/MRU based metric needs, c) it uses such ordering also to choose the way to be powered-off, offering more guarantees of being performance preserving than already experienced random choices.

2. Analysis of the Hits Distribution in a D-NUCA Cache

As a consequence of the promotion mechanism, during the execution of an application, in a D-NUCA cache the hits are not uniformly distributed along the ways. In fact, as also observed in [2], MRU data migrate towards the controller, while LRU data migrate to the opposite side. Particularly, we have found that the distribution of hits along the ways varies among different applications and also among different execution phases of the same application.

We have conducted an analysis of the distribution of cache hits in the set of SPEC2K applications listed in Table 2. We now show illustrative results that clearly demonstrate that applications have different associativity needs in different execution phases.

Assuming the D-NUCA Cache of Figure 1, Figure 2 shows the distribution of cache hits to the different ways for the SPEC2K applications Mcf and Twolf. For both the applications, the number of hits decreases when moving from Way0 to Way7. As expected, Mcf hits are distributed across the cache. The distribution of hits in Twolf is however quite surprising: hits are fully contained in the first two ways. Figure 3 shows the hit distributions for Parser when executing 10 Millions of instructions starting from 3.709B and from 3.859B. In the first case the hits are concentrated to 4 ways while in the second case they are spread quite uniformly across all the ways.

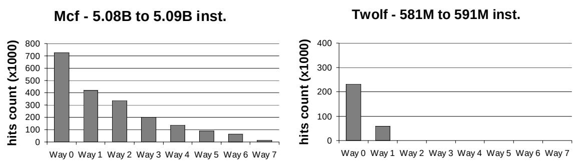

Figure 2: Distribution of cache hits across the reference D-NUCA ways for the Mcf application, in the running phase included between 5.08 and 5.09 billion of instructions and for the Twolf application in the running phase included between 581 and 591 million of instructions.

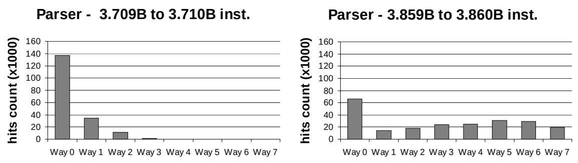

Figure 3: Hits distribution across the reference D-NUCA ways when the application is Parser, in the running phase included between 3.709 and 3.710 billion of instructions and between 3.858 and 3.860 billion of instructions.

These results suggest that, although a highly associative L2 D-NUCA cache can always contain the working set of an application, there are many cases in which the use of such a large cache is unnecessary and it wastes space and power. On the other hand, the use of caches with a limited associativity could be unsuitable (i.e. generates too high miss rates and too low IPC) for those applications whose working set requires a high associativity (compare the two distribution in Figure 2) while tuning the associativity to one application would mean losing the flexibility required by general-purpose cpus. Furthermore, both the solutions don't fully solve the problem because of the different locality exhibited in different portions of the execution of the same application as shown if we compare the results in Figure 3.

Based on these considerations, we propose to adopt a highly associative D-NUCA cache as a basic architecture, and to introduce a mechanism that allows to dynamically switch on and off the ways as a function of the number of ways needed by the current execution phase of the running application. We call this structure "Way Adaptable D-NUCA Cache".

3. Way Adaptable D-NUCA Cache

In a Way Adaptable D-NUCA Cache, each way can be dynamically turned off or on during the execution of an application, depending on the locality exhibited by the current execution phase of the running application. To decide when to turn on new ways and when to shut down those that are unnecessary, a prediction mechanism for the working set size is needed. The one we utilize is described in Figure 4.

The hardware implementation of our prediction mechanism is simple. Only two counters and the combinatorial logic for the two steps of the algorithm in Figure 4 are needed. As stated in [15], the impact of similar logic on both the dynamic and the static power consumption is negligible, when compared to a moderately sized cache.

All the control logic is assumed to be embedded in the cache controller, while the on/off switching of ways is realized via the Gated-V_{dd} transistors technology [9]. Such a technique has shown to be particularly effective in reducing static power consumption, especially for L2 caches [19].

```
Every K L2-cache hits do:
{
        D = Number of hits on the farthest powered-on way / Number of hits on the first way;
        If (D < $T_1$) then "shut down the farthest powered-on way";
        else if (D > $T_2$) then "turn on the closest powered-off way";
        else "stay in current configuration";
}
```

Figure 4: Description of the algorithm we used to decide when to switch on and off ways in a Way Adaptable D-NUCA cache. Note that "farthest" and "closest" are related to the position of the ways with respect to the controller.

Table 1: Parameters of the simulated architecture and of the prediction algorithm

Manufacturing Technology	70nm
Cpu Architecture	Alpha 21264
L1 d-cache	64K, 2-way, 64 bytes blocks, 3 cycle hit latency, 1 port
L1 i-cache	64K, 2-way, 64 bytes blocks, 1 cycle hit latency, 1 port
L2 unified cache	8Mbyte, 8 ways
L2 NUCA organization	16 x 8 banks
L2 banks	64Kbytes, 64 bytes blocks, 3 cycle data access latency, 2 cycle tag access latency
L2 inter banks links	Switched Network, 1 cycle latency per hop
Main memory latency	132 cycles
K	100000
T_1	0.005
T_2	0.02

Table 2: The benchmarks from the Spec2K suite used to evaluate the Way Adaptable D-Nuca Cache

	Phase			Phase	
SPECINT2000	FFWD	RUN	SPECFP2000	FFWD	RUN
176.gcc	2.367B	300M	172.mgrid	550M	1.06B
181.mcf	5.0B	200M	177.mesa	570M	200M
197.parser	3.709B	200M	173.applu	267M	650M
253.perlbmk	5.0B	200M	179.art	267M	200M
256.bzip2	744M	1.0B	178.galgel	4.0B	200M
300.twolf	511M	200M	183.equake	4.459B	200M

4. Experimental Methodology

The evaluation of the proposed solution is performed via execution driven simulation by modifying the SimAlpha simulator [8]. We built an extended version of the simulator that is able to make a cycle accurate simulation (and provide related statistics) of the memory and of the communication network taking into account the packets propagation over the links and throughout the switches, the memory bank accesses and the conflicts in the use of such elements.

We compare the performance of a Way Adaptable D-NUCA Cache with the baseline D-NUCA cache by measuring the IPC, the average number of active ways (i.e. the average number of ways that are powered on during the execution time of an application), the average access latencies and the number of accesses to the cache banks. For both the designs, the system parameters and the numerical values for the parameters K, T1 and T2 of the prediction algorithm used are given in Table 1.

Table 2 lists the benchmarks from the Spec2K suite we used in the simulations. The system parameters and the benchmarks together with their running conditions are the same ones used in [2].

5. Results

Figure 5 shows the achieved IPC (Instructions Per Cycle) and Figure 6 shows the average number of active ways when using a plain D-NUCA and a Way Adaptable D-NUCA to run the Spec2K applications listed in Table 2.

The IPC of the Way Adaptable NUCA is close to that of the reference Plain D-NUCA with a tiny 0,38% performance loss in average (last bars in Figure 5). The average number of active ways is considerably lower in the Way Adaptable D-NUCA Cache and, with respect to the baseline D-NUCA, it is reduced by 42% (last bars in Figure 6). We don't report an analytic evaluation of the static power consumption reduction, but the average number of turned off ways gives a good metric for such reduction, as the static power consumption strictly depends on leakage currents that are directly comparable with the number of powered on transistors and thus, in our case, with the number of powered on ways.

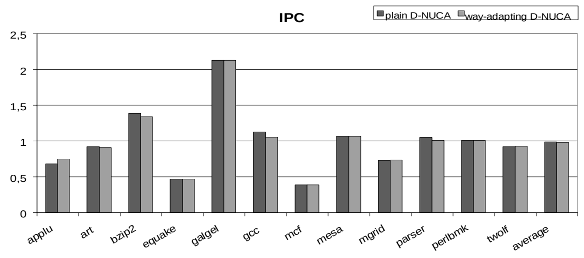

Figure 5: The IPC (Instructions per Cycle) achieved by the D-NUCA structures considered in our evaluation for the various benchmarks and as average.

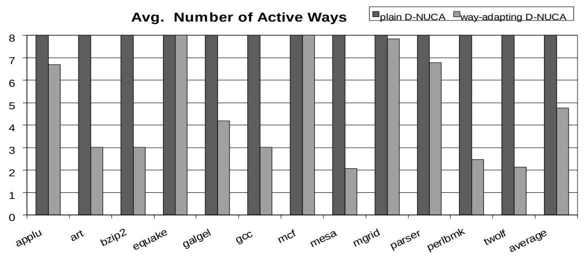

Figure 6: The average number of active ways for the D-NUCA structures considered in our evaluation for the various benchmarks and as average.

Table 3: Average Latencies and Bank Request Count for Plain D-NUCA and for Way Adaptable D-Nuca

	Plain D-NUCA	Way Adaptable D-NUCA
Hit latency	13.34 cycles	13.13 cycles
Miss latency	23.11 cycles	15.99 cycles
Avg. lat.	14.14 cycles	13.68 cycles
Avg. Bank Requests	143.46 Millions	101.23 Millions

As one would expect, turning off one or more ways also helps reducing the average cache latencies: one can turn off the remotely located ways, so new data enter the cache closer to the controller and they are promoted to the faster ways after a lower number of hits; in the same time a cache miss detection requires the access to a fewer number of banks. Table 3 lists the average latency values we measured for the Plain D-NUCA in comparison with the Way Adaptable D-NUCA cache. With respect to the Plain D-NUCA, we obtain a 30% reduction in miss detection time and a 3.25% reduction in overall cache access time. For the same reasons, the cache network traffic and related power consumption are reduced: on each cache reference a fewer number of physical banks must be accessed. Table 3 also reports the average number of requests that are performed on the cache banks, showing a 29% reduction of requests in the Way Adaptable case.

6. Related Works

The D-NUCA cache architecture was proposed in [2], [3], showing that a dynamic NUCA structure achieves 1.5 times higher IPC than a traditional Uniform Cache Architecture (UCA) when maintaining the same size and manufacturing technology. Extensions of the original idea are Nurapid [10] and Triangular D-NUCA cache [11]. They improve performance by decoupling tags from data or by changing mapping and size of each way.

Many techniques have been proposed to dynamically adapt the caches size to the working set size [4]. Most techniques are focused on performances and dynamic power saving and are not directly applicable for reducing the static power consumptions. As an example, in [13], [14], [15] the Selective Cache Ways and similar techniques have been proposed. They require an always powered on cache to reduce miss rate and dynamic power consumption.

The techniques focusing on the reduction of static power consumption rely on putting memory cells in a low leakage mode either loosing or maintaining their data contents. Examples of such techniques and related circuital solutions are MTCMOS Caches [17], Drowsy Caches [18], Decay Lines Caches [16], Slumberous cache [20], and Way Adaptable Caches [12]. The last work, in particular, introduces a technique to control active ways in a conventional, associative L1 cache. Such a technique adopts a LRU/MRU metric, so it could be applied to D-NUCA only with more complex hardware extensions than ours. Besides, it utilizes a random policy to power-off ways, that we verified being more performance affecting than powering off the last ways. Adaptable D-NUCA Caches seem to be very promising as, in addition to the power consumption reduction, they reduce both network traffic and the average cache latency, as showed in section 5.

Finally, our work is related to the more general framework of detecting program phases [6] to adapt at runtime the number of active resources to the need of the running program. A comparison of the performances of both ideal and practical implementations of such techniques may be found in [21].

7. Conclusions

D-NUCA cache power efficiency can be enhanced through the adoption of a control mechanism that dynamically turns on and off the ways in dependency of the need of the running application. Experimental results have shown that, in a Way Adaptable D-NUCA cache, the average number of active ways can be reduced by 42%, on average, with respect to a conventional D-NUCA without affecting the overall IPC. Reducing the number of active ways contributes also to reduce hit and miss cache latencies and the network traffic. Future evolutions of this work will be in multiple directions. An accurate tuning of the K, T_1 and T_2 parameters will be performed. Different working set prediction algorithms will be investigated together with the possibility of turning on/off more ways in the same step. An accurate evaluation of the actual power saving achieved with our proposed technique will be conduced.

Acknowledgements

We wish to thank Massimo Macucci who furnished us with the Alpha platform that we used to compile the SPEC 2K benchmarks.

This works is partially supported by the SARC project founded by European Union under the contract no. 27648.

References

[1] V. Agarwal, M.S. Hrishikesh, S. Keckler. D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. Proc. 27th Int. Symp. on Computer Architecture, pp.248-259, Vancouver, Canada, June 2000

[2] C. Kim, D. Burger, S.W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. Proc. 10th Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 211-222, San Jose, CA, Oct. 2002.

[3] C. Kim, D. Burger, S.W. Keckler. Nonuniform cache architectures for wire-delay dominated on-chip caches. IEEE Micro, vol. 23(6), pp. 99-107, Nov./Dec., 2003

[4] V. Venkatachalam, M. Franz. Power Reduction Techniques For Microprocessor Systems. ACM Computing Surveys, vol. 37(3), pp. 195-237, Sept. 2005

[5] The International Technology roadmap for Semiconductors. Semiconductor Industrial Association, 2005.

[6] T. Sherwood, B. Calder. Time varying behavior of programs. UC San Diego Technical Report UCSD-CS99-630, 1999

[7] Standard Performance Evaluation Corporation http://www.spec.org/

[8] R. Desikan et al., Sim-Alpha: A Validated Execution-Driven Alpha2164 Simulator, tech report TR-01-23, Dept. of Computer Sciences, Univ. Texas at Austin, 2001.

[9] M.Powell, S.Yangh, B.Falsafi, K.Roy, T.N. Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. Proc. 2000 Int. Symp. Low Power Electronics and Design, pp. 90-95, Rapallo, Italy, July 2000.

[10] Z. Chisti, M.D. Powell, T.N. Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. Proc..36th Int. Symp. on Microarchitecture, pp.55-66, San Diego, CA, Dec. 2003

[11] P. Foglia, D. Mangano, C.A. Prete. A Cache Design for High Performance Embedded Systems. Journal of Embedded Computing, vol. 1(4), pp. 587-598, 2005

[12] H. Kobayashi, I.Kotera, H. Takizawa. Locality Analysis to Control Dynamically Way-Adaptable Caches. ACM SIGARCH Computer Architecture News, vol. 33(3), pp. 25-32, June 2005

[13] D. H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. Proc. 32nd Int. Symp on Microarchitecture, pp.248-259, Israel, November 1999

[14] R. Balasubramonian, et al: Memory hierarchy reconfiguration for energy and performance in general purpose processor architectures. Proc. 33rd Int. Symp. On Microarchitecture, pp.245-257, Monterey, CA, Dec. 2000.

[15] D. Dropsho, et al. Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power. Proc. 2002 Int. Conf. on Parallel Architectures and Compilation Techniques, pp.141-152, Charlottesville, CA, Sep.2002.

[16] Z. Hu, S. Kaxiras, M. Martonosi. Let caches decay: reducing leakage energy via exploitation of cache generational behavior. ACM Trans. on Computer Systems, vol. 20(2), pp. 161-190, May 2002.

[17] H. Hanson, et al. Static energy reduction techniques for microprocessor caches. IEEE Trans. on VLSI, vol. 11(3), pp. 303-313, June 2003

[18] K. Flautner, N.S.Kim, S.Martin D.Blaauw, T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. Proc. 29th Int. Symp. on Computer Architecture, pp.148-157, Anchorage, AK, May 2002

[19] Y. Meng, T. Sherwood, Kastner. Exploring the limits of leakage power reduction in caches. ACM Trans. on Architectures and Code Optimization, vol. 2(3), pp.221-246, Sep. 2005

[20] N. Mohyuddin, R. Bhatti, M. Dubois. Controlling Leakage Power with the Replacement Policy in Slumberous Cache. Proc. 2nd Conf. on Computing Frontiers, pp 161-170, Ischia, Italy, May 2005.

[21] A. S. Dhodapkar, J. E. Smith. Comparing Program Phase Detection Techniques. Proc. of the 36th Int. Symp. on Microarchitecture, pp. 217-227, San Diego, CA, Dec. 2003.