A Micro-Architectural Power-Saving Technique for D-NUCA Caches

Alessandro Bardine[1*], Pierfrancesco Foglia[1*], Giacomo Gabrielli[1*],
Cosimo Antonio Prete[1*], Per Stenström[2*]

***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 observed that the distribution of hits across the ways of a D-NUCA cache varies across applications as well as across different execution phases within a single application. In this work, we show how such a behavior can be leveraged to improve the D-NUCA power efficiency as well as to decrease its access latency.

In particular, we propose: 1) A new micro architectural technique to reduce the static power consumption of a D-NUCA cache by dynamically adapting the number of active (i.e. powered-on) ways to the need of the running application; our initial evaluation shows that a strong reduction of the average number of active ways (36.9%) is achievable, without significantly affecting the IPC (-2.97%), leading to a resultant reduction of the Energy Delay Product (EDP) of 30.9%. 2) A strategy to tune the characteristic parameters of the proposed technique. 3) A variant of the technique which leads to a more aggressive power reduction strategy

***Index Terms*—Cache memories, NUCA, Dynamic-NUCA, Adaptable Computing, Wire Delay, Power Consumption**

I. Introduction

CMOS technology 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 access latency of such caches are dominated by the wire-delay [1]. In order to reduce this effect, 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 communications among the banks and the cache controller are supported by a switched network. This organization allows some banks to be closer to the processor, hence allowing shorter access latencies to these banks with respect to banks that are located farther away. The mapping between cache lines and physical banks can either be Static or Dynamic (namely S-NUCA and D-NUCA) [2]. In the former, each line can exclusively reside in a single predetermined bank and, in the latter (Fig. 1), each line can be mapped to one of a set of different banks, similarly to a set associative cache, and it can dynamically migrate from one bank to another. As shown in Fig. 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 address space is spanned on the banks belonging to each row. Each bank belonging to a column behaves like a single way of a set associative cache, so a line is allowed to reside only in a single bank belonging to that column. When a cache line search is performed, the controller first determines the column that could contain the requested data using the lowest-order bits of the index field from the address, then it broadcasts the request to all the banks belonging to that column. As soon as a hit happens in one of these banks, the request is satisfied without the need of waiting for the replies from the other, farther, banks. To further reduce access latencies, the "promotion/demotion" mechanism is adopted: if a hit happens in a row other than the first, the cache line is promoted by swapping it with the line that holds the same 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 Fig. 1), possibly evicting any corresponding line. As a consequence, the most frequently accessed lines are likely to be located into the banks closer to the processor, thus improving the access latency. With such a policy, D-NUCA caches succeed in achieving high hit rates while keeping the access latency 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 chips, where large storage capabilities, high clock rates and low memory access latencies will be required.

Problem. Big SRAM structures, like the ones employed in a modern L2 cache, exhibit high energy requirements, especially due to the static component caused by leakage currents typical of deep submicron CMOS technologies [24] [25]. A previous work [22] highlights that D-NUCA caches are better performing and more energy efficient than UCA and S-NUCA caches. The D-NUCA scheme exhibits an increase

* Members of the HiPEAC Network of Excellence on High-Performance Embedded Architecture and Compilation

[1] are with the Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Via Diotisalvi 2, 56126 Pisa, Italy ({alessandro.bardine, foglia, giacomo.gabrielli, prete}@iet.unipi.it)

[2] is with the Department of Computer Science and Engineering, Chalmers University of Technology, S-412 96 Gothenburg, Sweden (pers@ce.chalmers.se)

of the dynamic power components due to the higher number of bank accesses and network transmissions, but this is overwhelmed by the static energy saving due to the shorter execution time achieved. Anyway, like in the UCA and S-NUCA schemes, the D-NUCA energy budget is dominated by leakage, which consequently should be the most effective objective for power-saving techniques.

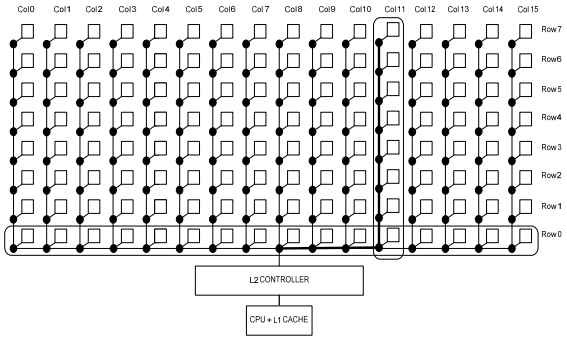

Fig. 1 – The reference D-NUCA cache employed in our study. This 8MB D-NUCA cache is made up of 128 banks (the squares in the picture) organized in rows and columns; each row represents a way, so that the cache behaves like an 8-way set associative cache. The rounded corner contours highlight a single row and a single column of banks. The bold line shows the path followed by a request for a specific cache line: from the address, the column to be searched is determined (the 11th in the figure); a request is sent along the horizontal links till the column is reached; then the request is forwarded along the column. At each switch (the black circles) belonging to that column the request is forwarded to the cache banks to perform the accesses.

Contribution. As a consequence of the promotion/demotion mechanism, we observed that in a D-NUCA cache the hits are not uniformly distributed across the different ways. Instead, their distribution shows strong variations across different applications as well as across different execution phases within a single application. Such a behavior suggests that not all the cache ways are needed during the whole execution of a program: due to their poor usage, the least frequently accessed ways could be powered-off without significantly affecting performance. Accordingly, a reduction of the cache's static power consumption can be achieved. 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 accessed in order to detect a cache miss. This feature could have beneficial effects on performance as well as on dynamic power consumption.

In this work we introduce a power-saving technique targeted at D-NUCA caches, called "Way Adaptable", which leverages our observations on the distribution of hits across the different ways. This technique is based on a mechanism to dynamically turn on/off the ways, according to the need of the running application. In particular, our contribution can be stated as follows: 1) We develop a specific algorithm to adapt the number of active ways of a D-NUCA cache to the needs of the running application. The metric of the algorithm and the selection of the way to be powered off are based on the intrinsic D-NUCA data line ordering. The use of the proposed algorithm implies a reduction of the power consumption, cache access time, and network traffic with a narrow performance loss (average 36.9% reduction of number of active ways, 29% reduction of bank access requests, 3.25% reduction of cache access latency, 2.97% degradation of IPC, 34.6% reduction of energy consumption and 30.9% reduction of EDP – Energy Delay Product). 2) We propose a methodology to tune, on an application basis, the parameters of the proposed algorithm and we show the sensitiveness of such parameters to the change of the running application. 3) We propose a variant of the original Way Adaptable scheme, called Differential Way Adaptable, which is even more power preserving.

II. Analysis of the Distribution of Hits 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 across the different ways. In fact, as also observed in [2], most frequently accessed data tend to migrate toward the controller, while least frequently accessed data migrate to the opposite side. Particularly, we have found that the distribution of hits across the ways varies between different applications as well as between different execution phases of the same application.

We have conducted an analysis of the distribution of cache hits for the SPEC CPU2000 applications listed in Table 2. We now report illustrative results that show that applications have different associativity needs for different execution phases.

Assuming the D-NUCA cache represented in Fig. 1, Fig. 2 shows the distribution of cache hits across the different ways for the SPEC CPU2000 applications *mcf* and *twolf*. For both the applications, the number of hits decreases when moving from Way0 to Way7. While the hits for *mcf* span the entire cache, *twolf* exhibit a different behavior: the hits involve only the first two ways, which are able to fully contain the application's working set. Fig. 3 shows the distribution of hits for *parser* when executing 10 Millions of instructions, starting at 3.709B and at 3.859B respectively. In the first case the hits are concentrated into the first 4 ways, while in the second case they span quite uniformly the entire cache.

These results suggest that, although a large highly associative L2 D-NUCA cache most of the time is able to 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, while tuning the associativity to a single 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 by different execution phases of the same application, as shown if we compare the results in Fig. 3.

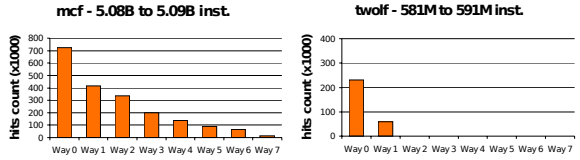

Fig. 2 – Distribution of cache hits across the different ways of the reference D-NUCA cache for the *mcf* application, in the running phase included between 5.08 and 5.09 billion of committed instructions, and for the *twolf* application, in the running phase included between 581 and 591 million of committed instructions.

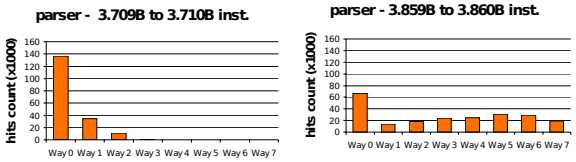

Fig. 3 – Distribution of cache hits across the different ways of the reference D-NUCA cache for the parser application, in the running phase included between 3.709 and 3.710 billion of committed instructions, and in the phase between 3.859 and 3.860 billion of committed instructions.

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 level of associativity needed by the current execution phase of the running application. We call this structure "Way Adaptable D-NUCA cache".

III. Way Adaptable D-NUCA Cache

In a Way Adaptable D-NUCA cache, each way can be dynamically turned on/off during the execution of an application, depending on the locality exhibited by the current execution phase. 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. Fig. 4 shows the one we developed. Two counters accumulate the number of hits in the first and the farthest powered on ways during the program execution. Every K cache hits the ratio D between the counters is evaluated and compared against two thresholds T_1 and T_2 in order to decide to shut down a way (if $D < T_1$) or to turn on a new way (if $D > T_2$) or to stay in the current configuration (if $T_2 > D > T_1$).

The hardware complexity of the proposed prediction mechanism is very limited. Only three counters and the combinatorial logic for the two steps of the algorithm in Fig. 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}* transistor 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 = farthest_way_hits_counter  /first_way_hits_counter;
   if (D < T1) then
      "shut down the farthest powered-on way";
   else if (D > T2) then
      "turn on the closest powered-off way";
   else "keep current configuration";

   last_way_hits_counter=0;
   first_way_hits_counter=0;
}
```

Fig. 4 – Description of the proposed algorithm to decide when to switch on/off ways in a Way Adaptable D-NUCA cache. Note that "farthest" and "closest" refer to the position of the ways with respect to the cache controller.

TABLE 1
PARAMETERS FOR THE SIMULATED SYSTEM AND FOR THE PREDICTION ALGORITHM

Manufacturing Technology	70nm
Cpu Architecture	Alpha 21264
L1 d-cache	64 KB, 2-way, 64 bytes block, 3 cycle hit latency
L1 i-cache	64 KB, 2-way, 64 bytes block, 1 cycle hit latency
L2 unified cache	8 MB, 8-way set associative
L2 NUCA organization	16 x 8 banks
L2 banks	64KB, 64 bytes block, 3 cycle data access latency, 2 cycle tag access latency
L2 interconnection network	switched network, 1 cycle latency per hop
Main memory latency	300 cycles
K	100000
T_1	0.005
T_2	0.02

TABLE 2
THE BENCHMARKS FROM THE SPEC CPU2000 SUITE USED TO EVALUATE THE WAY ADAPTABLE TECHNIQUE

	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

IV. Experimental Methodology

The evaluation of the proposed solution has been performed via execution driven simulation, employing a modified version of the *sim-alpha* 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 NUCA cache memory and of the communication network taking into account the propagation of packets over the links and throughout the switches, the bank accesses and the conflicts in the use of such resources.

We compare the performance of a Way Adaptable D-NUCA cache with the baseline D-NUCA cache by measuring the IPC (Instructions Per Cycle), 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, T_1 and T_2 of the prediction algorithm are given in Table 1. Table 2 lists the benchmarks from the SPEC CPU2000 suite we used in the simulations. The system parameters and the benchmarks together with their running conditions are the same as described in [2].

Fig. 5 and Fig. 6 respectively show the achieved IPC and the average number of active ways for plain D-NUCA and Way Adaptable D-NUCA caches, under the SPEC CPU2000 workload described in Table 2. The IPC achieved by the Way Adaptable scheme is close to the one achieved by the reference plain D-NUCA, with a narrow average 2.97% performance loss (last bars in Fig. 5). The average number of active ways is considerably lower for the Way Adaptable D-NUCA cache and, with respect to the baseline D-NUCA, it is reduced by 36.9% (last bars in Fig. 6).

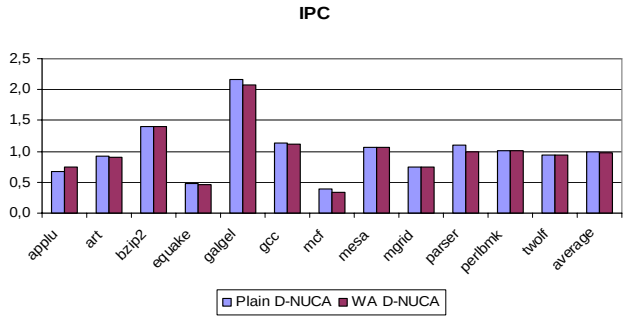

Fig. 5 – The IPC (Instructions Per Cycle) achieved by the D-NUCA structures considered in our evaluation for the different benchmarks and on average.

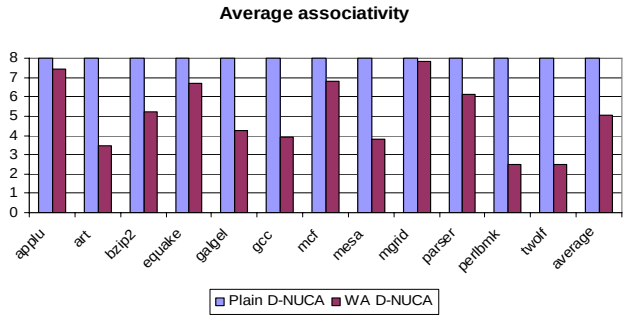

Fig. 6 – The average number of active ways for the D-NUCA structures considered in our evaluation for the different benchmarks and on average.

A quantitative evaluation of the energy consumption of the L2 cache and main memory subsystem has been performed assuming an underlying energy model proposed in [22]. Since static energy largely depends on temperature, here we assume a fixed operating temperature of 80°C, which is quite representative of typical working conditions of on-chip L2 caches [26]. Fig. 7 shows the normalized energy consumption for Way-Adaptable and plain D-NUCA caches under the given workload. 10 benchmarks out of 11 benefit from the application of the proposed technique, with an average reduction of energy consumption by 34.6%.

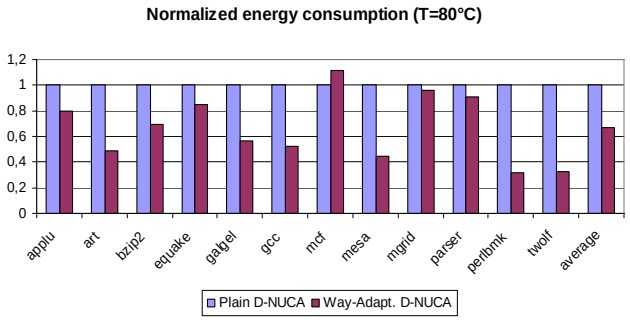

Fig. 7 – Comparison of energy consumption (normalized with respect to plain D-NUCA) for plain and Way-Adaptable D-NUCA caches at a fixed temperature of 80°C. The trend shows a significant reduction of energy consumption achieved by the proposed technique. Only for the *mcf* benchmark the energy consumption is larger for the Way-Adaptable scheme since the longer execution time (due to performance degradation) introduces extra static energy consumption, which is not balanced by the reduction of active ways.

A representative synthetic metric to evaluate the goodness of micro-architectural solutions is the EDP (Energy Delay Product) [23]. Fig. 8 shows the EDP achieved by the Way-Adaptable D-NUCA scheme normalized with respect to plain D-NUCA. The average improvement is 30.9%. The exhibited trend is similar to the one exhibited by normalized energy since this technique is able to limit the performance degradation in almost all cases.

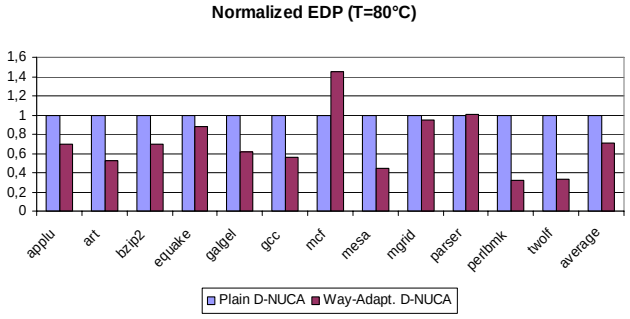

Fig. 8 – Comparison of achieved EDP (normalized with respect to plain D-NUCA) for plain and Way-Adaptable D-NUCA caches at a fixed temperature of 80°C.

TABLE 3
AVERAGE LATENCIES AND NUMBER OF BANK REQUESTS FOR PLAIN AND 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. latency	14.14 cycles	13.68 cycles
Avg. n. of 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; at the same time a cache miss detection requires the access to a lower number of banks. Table 3 lists the average latency values we measured for the Plain D-NUCA, compared 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 reduced 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 the number of requests for the Way Adaptable case.

V. Algorithm Parameters Tuning

The values for T_1 and T_2 thresholds reported in Table 1 were heuristically determined. The chosen values lead to a good average behaviour of the Way Adaptable D-NUCA cache. However, the accuracy of the prediction algorithm and thus the performance/power trade-off can be improved by acting on such parameters. In particular, in this section we introduce a strategy that is suitable to tailor the values of thresholds T_1 and T_2 to a single specific application.

In the algorithm (Fig. 4), a reconfiguration event is performed once every a fixed number K of L2 cache hits (i.e once every 100000 cache hits). Each reconfiguration event is one of: “turn on a way” (+1 way), “turn off a way” (-1 way), or “keep current configuration”. The first step of our strategy is to identify the sequence of reconfiguration events for the given application that, among all the possible sequences, is optimal for a chosen goodness metric. As a metric, we have selected the miss rate but other alternatives are possible such as IPC, EDP, etc. To determine the optimal reconfiguration sequence, we would need to explore all the possible combinations and calculate the metric for each one, resulting in an extremely high number of tests to perform (i.e. a simulation has to be run for each possible sequence to be evaluated). So an incremental approach can be adopted: after a first warm-up phase, the program’s execution is halted every K cache hits, and the execution state is saved. Then the execution is restarted in three different runs assuming a different reconfiguration event being applied at the start of each run; at the end of the three runs, the results are collected and the most suitable reconfiguration event is chosen and assumed to be applied; in particular (since higher associativities and larger capacities naturally lead to lower miss-rates) the reconfiguration event that is chosen is the most conservative one which leads to a miss-rate within 1% of the lowest achievable miss-rate, meaning that we choose the event which allows the highest energy consumption reduction and which limits the miss-rate degradation to 1% with respect to the optimal case. Starting from it, and focusing on the next interval, three new simulations are conducted in the same way. This procedure is repeated up to the end of the selected application. Fig. 9 depicts the described strategy.

At each step, the selected reconfiguration events place restrictions on the values of T_1 and T_2, so, at the end, we obtain a set of inequalities for the two thresholds; solving them we obtain values for T_1 and T_2. If the set is not fully solvable, we can restrict the number of inequalities reducing the accuracy of the proposed strategy. The main drawback of the procedure described above is that the optimal reconfiguration events are determined according to a metric that is evaluated locally to each stable phase, so the reconfiguration sequence results to be a pseudo-optimal one.

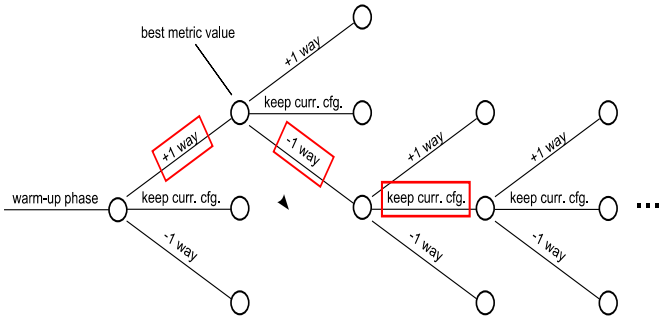

Fig. 9 – Application of the procedure to identify the pseudo-optimum reconfiguration sequence for a simple case. In the example the selected sequence is: {+1 way; -1 way; keep current configuration}.

VI. Differential Way Adaptable Technique

The availability of the pseudo-optimal reconfiguration sequence introduced in the previous section, gives us the chance to explore some variants of the Way Adaptable technique. In this section we illustrate the “Differential Way Adaptable” technique, a different version of the technique, which is more aggressive in terms of number of powered off ways, without further affecting the IPC.

Starting from the pseudo-optimal reconfiguration sequence, we have found that, if we calculate at the end of each interval the following value:

$$Q = \frac{D_i - D_{i-1}}{D_i},$$

where D_i is the ratio between the number of hits on the first way and the number of hits on the last way at the step i (as calculated in the original Way Adaptable algorithm), we obtain a value Q that reflects the actual trend of cache memory usage of the running program more accurately than the previously used D. This value, similarly to what happens in the original technique, is compared against the two thresholds T_1 and T_2 to trigger a reconfiguration event. In order to determine the value of these thresholds, we applied the same strategy described in the previous section, thus tailoring them to a specific application.

Fig. 10 and Fig. 11 show the IPC and the average associativity for two benchmarks, *equake* and *perlbmk*, comparing the plain D-NUCA scheme, the Way Adaptable scheme and the Differential Way-Adaptable scheme. For each application, the Differential Way Adaptable scheme uses the two thresholds that has been obtained by the application of the strategy described so far. As the figures suggest, using thresholds that are tuned on a specific application results in a bigger static power reduction, without affecting performances.

In order to evaluate the sensitivity of performance and power reduction to different values of the two thresholds, we

measured the IPC (Fig. 12) and the average associativity (Fig. 13) for the entire workload, first applying a Differential Way Adaptable D-NUCA scheme with thresholds tuned on *equake* (Diff. WA D-NUCA 1), then applying a Differential Way Adaptable D-NUCA scheme with thresholds tuned on *perlbmk* (Diff. WA D-NUCA 2). For each benchmark, the two schemes exhibit similar behaviors, as suggested by the values of IPC and average associativity. Thus, even if the thresholds are tuned on a specific application, the algorithm still exhibits a good effectiveness when applied to other applications.

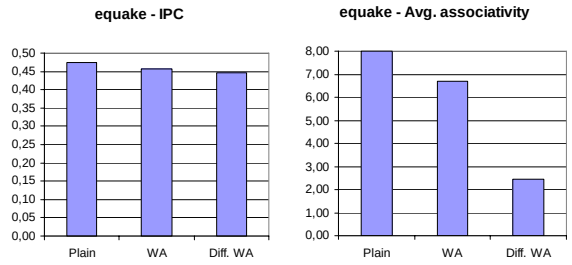

Fig. 10 – IPC and average associativity for the *equake* application, comparing plain, Way Adaptable (WA) and Differential Way Adaptable (Diff. WA) schemes, under the same experimental conditions. For the Differential version, the employed thresholds are the following: T_1 = -1.2, T_2 = 0.6; for the original Way Adaptable version, the values of the thresholds are shown in Table 1.

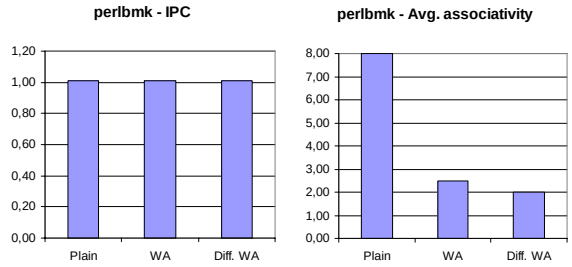

Fig. 11 – IPC and average associativity for the *perlbmk* application, comparing plain, Way Adaptable (WA) and Differential Way Adaptable (Diff. WA) schemes, under the same experimental conditions. For the Differential version, the employed thresholds are the following: T_1 = -0.6, T_2 = 0.33; for the original Way Adaptable version, the values of the thresholds are shown in Table 1.

Such results indicate that, if the application to be run is known in advance, the parameters of the Diff. Way Adaptable technique can be customized in order to obtain the highest power reduction without affecting performances. The thresholds values could be determined with a profile-driven approach at compile time, adopting the methodology described above. However, if the application is not known in advance and thus it is not possible to accurately calculate the thresholds, the use of standard medium values is not disadvantageous and still allows to obtain a relevant static power reduction and limited performance loss.

Fig. 12 and Fig. 13 also show a comparison of the two Diff. Way Adaptable D-NUCAs with Plain D-NUCA and with the traditional Way Adaptable D-NUCA. The Diff. Way Adaptable technique results to be more aggressive than the traditional Way Adaptable in turning off ways (45% as average), at the cost of a limited IPC loss (7% as average).

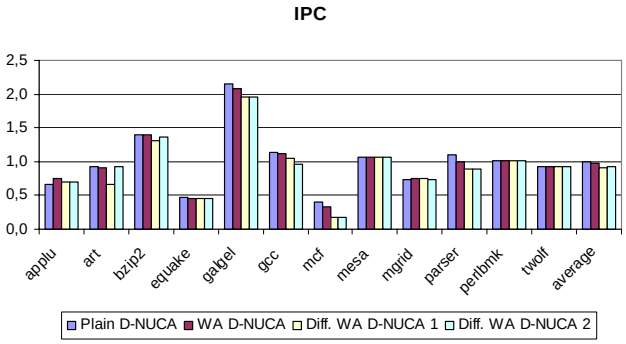

Fig. 12 – IPC for the entire workload, comparing plain, Way Adaptable (WA), Differential Way Adaptable (Diff. WA D-NUCA 1) with T_1 and T_2 tuned for *equake*, Differential Way Adaptable (Diff. WA D-NUCA 2) with T_1 and T_2 tuned for *perlbmk*.

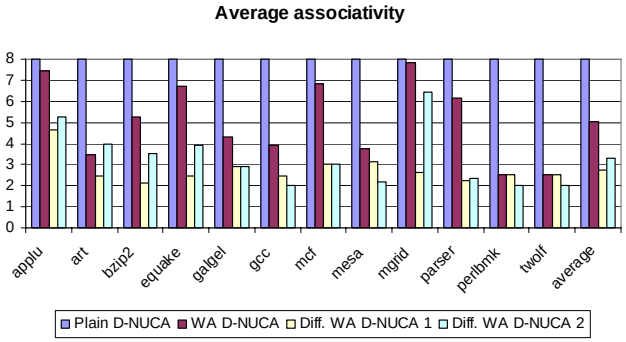

Fig. 13 – Average associativity for the entire workload, comparing plain, Way Adaptable (WA), Differential Way Adaptable (Diff. WA D-NUCA 1) with T_1 and T_2 tuned for *equake*, Differential Way Adaptable (Diff. WA D-NUCA 2) with T_1 and T_2 tuned for *perlbmk*.

VII. 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. An energy/performance trade off evaluation for NUCA and its comparisons with UCA is given in [22].

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. MTCMOS Caches [17] and Drowsy Caches [18] are data preserving techniques while Decay Lines Caches [16] is a data loosing technique. In order to utilize them in a D-NUCA cache, the characteristic concepts and parameters on which they rely need to be redefined: in fact, because of the promotion/demotion mechanism, D-NUCA caches exhibit a different access pattern to data lines with respect to a UCA cache. However, it is worth noting that the Way Adaptable technique can be used in conjunction with such other techniques: in fact while the former acts increasing or decreasing the cache size/associativity, the later work at block granularity without further modifying the cache behavior. Thus, it is possible to conjunctly adopt them in a D-NUCA cache obtaining the benefits of both.

Slumberous cache [20] is another data preserving technique which uses the LRU/MRU state of each cache line to adapt the level of activity of the memory cells and consequently their leakage power consumption. The direct application of such techniques to a D-NUCA would imply a relevant logic complexity to calculate the LRU/MRU state on each reference. The Way Adaptable Caches [12] is a data loosing technique that predicts cache needs adopting a metric based on the LRU/MRU state of each way and turn on and off entire ways based on a random choice. Similarly to previous one, the direct application of such technique to a D-NUCA would imply the complexity of calculating the LRU/MRU state; furthermore the random choose of a way to turn off wouldn't take into account the actual data usefulness. Our proposed technique is as an extension of such technique to the D-NUCA cache.

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].

VIII. Conclusions

D-NUCA caches are a promising technology as they are able to hide to the CPU the wire-delay effects introduced by high clock rates and technology scaling. Anyway, similarly to the other L2 cache structures, they are affected by high power requirements. In this work we have shown that D-NUCA cache power efficiency can be enhanced by dynamically adapting the size of the cache to the actual needs of the running applications. The size adaption is obtained by using a simple prediction algorithm that decides to turn on and off the cache ways and has limited drawback on performance.

Experimental results have shown that, in a Way Adaptable D-NUCA cache, the average number of active ways can be reduced by 36.9%, on average, with respect to a conventional D-NUCA with a slight 2.97% average IPC reduction. Consequently the average energy consumption is reduced by 34.6% and the average Energy Delay Product is improved by 30.9%. Differently from what happens in previously proposed power reduction techniques, reducing the number of active ways contributes also to reduce hit and miss cache latencies and the network traffic.

A methodology has been defined to tune, on an application basis, the parameters of the prediction algorithm via a pseudo-optimal reconfiguration sequence that can be determined throughout simulation. Finally, an alternative prediction algorithm has been presented

Future works will be focused on further investigation of prediction techniques that are able to better reflects the memory needs of the running application. In particular techniques that will be able to turn on or off two or more ways on each step will be evaluated. Besides the study of the proposed techniques with multithreaded workloads in a multicore environment will be performed.

Acknowledgments

We wish to thank Massimo Macucci who furnished us with the Alpha platform that we used to compile the SPEC CPU2000 benchmarks.

This work is partially supported by the SARC project funded by the 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.
[22] A. Bardine, P. Foglia, G. Gabrielli, C. A. Prete. Analysis of Static and Dynamic Energy Consumption in NUCA Caches: Initial Results. Proc. of the MEDEA 2007 Workshop, pp. 105-112, Brasov, Romania, Sept. 2007.
[23] M. Horowitz, T. Indermaur, R. Gonzales. Low-Power Digital Design. Proc. Of the IEEE Symposium on Low Power Electronics, 1994, pp 8-11.
[24] Nam Sung Kim et al. Leakage current: Moore's law meets static power. IEEE Computer, Vol. 36(12). Pages68–75, 2003
[25] John Wuu et al. The Asynchronous 24 MB On-Chip Level 3 Cache for a Dual-Core Itanium Architecture Processor. In Proc. of the Int'l Solid tate Circuits Conf.(ISSCC'05), 2005.
[26] D. Greenhill, J.Alabado. Power Savings in the UltraSPARC T1 Processor. Sun Microsystem Whitepaper. Dec. 2005