Leveraging Data Promotion for Low Power D-NUCA Caches

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

[1] *Dip. di Ingegneria dell'Informazione, Università di Pisa*
Via Diotisalvi 2, 56126 Pisa, Italy
{alessandro.bardine, manuel.comparetti, foglia, giacomo.gabrielli, prete}@iet.unipi.it

[2] *Dept. of Computer Science and Engineering, Chalmers University of Technology, S-412 96 Gothenburg, Sweden*
pers@ce.chalmers.se

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 evaluation shows that a strong reduction of the average number of active ways (37.1%) is achievable, without significantly affecting the IPC (-2.25%), leading to a resultant reduction of the Energy Delay Product (EDP) of 30.9%. 2) A strategy to estimate the characteristic parameters of the proposed technique. 3) An evaluation of the effectiveness of the proposed technique in the multicore environment.

This work is partially supported by the SARC project funded by the European Union under the contract no. 27648.

* Members of HiPEAC – The European Network of Excellence on High-Performance Embedded Architecture and Compilation.

1. 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, while in the latter (Figure 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 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 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 Figure 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 [5] [25] [26]. A previous work [22] highlights that D-NUCA caches are better performing and more energy efficient than UCA and S-NUCA caches. With respect to UCA and S-NUCA, the D-NUCA scheme exhibits an increase 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. However, 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.

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 limited 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.

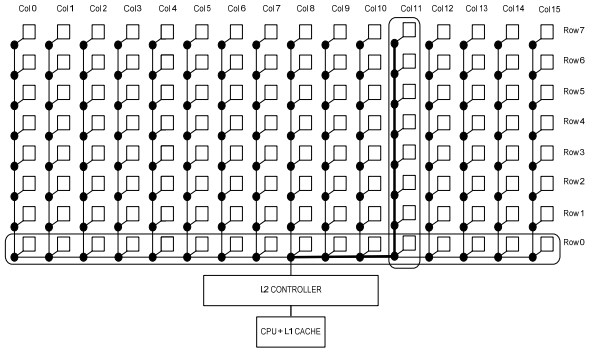

Figure 1: The reference D-NUCA cache employed in our study. This 8MB D-NUCA cache is made up of 128 banks (the squares in the picture) organized in rows and columns; each row represents a way, so that the cache behaves like an 8-way set associative cache. The rounded corner contours highlight a single 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 corresponding column to 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.

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 37.1% reduction of number of active ways, 29% reduction of bank access requests, 3.25% reduction of cache access latency, 2.25% degradation of IPC, 34.6% reduction of energy consumption and 30.9% reduction of EDP – Energy Delay Product). 2) We propose a methodology to estimate the parameters of the proposed algorithm and we show the sensitiveness of such parameters as

the running application changes. 3) We show the effectiveness of the proposed technique in the multicore environment.

2. 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.

Assuming the 8 MB D-NUCA cache represented in Figure 1, we have conducted an analysis of the distribution of cache hits for the SPEC CPU2000 [7] applications listed in Table 2. We now report illustrative results that show that applications have different associativity needs for different execution phases.

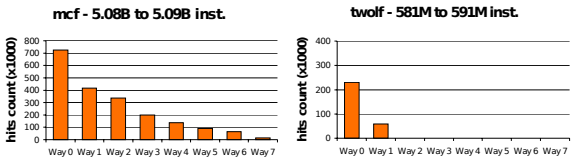

Figure 2: Distribution of cache hits across the different ways of the reference D-NUCA cache for the *mcf* application, in the running phase included between 5.08 and 5.09 billion of committed instructions, and for the *twolf* application, in the running phase included between 581 and 591 million of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

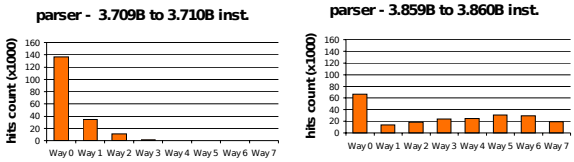

Figure 3: Distribution of cache hits across the different ways of the reference D-NUCA cache for the *parser* application, in the running phase included between 3.709 and 3.710 billion of committed instructions, and in the phase between 3.859 and 3.860 billion of committed instructions.

Figure 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. Figure 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, like the reference one, is able most of the time to contain the working set of an application, there are many cases in which the use of such a large cache is unnecessary, thus wasting space and power. On the other hand, the use of caches with a limited associativity could be unsuitable (i.e. it 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 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 level of associativity 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 on or 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. Figure 4 shows the one we developed. During the program execution, two counters accumulate the number of hits in the first way and in the farthest powered on way. 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 $T_1 < D < T_2$).

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 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.

```
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;
}
```

Figure 4: Description of the proposed algorithm to decide when to switch on or 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.

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}* technology [9]. Such a technique has shown to be particularly effective in reducing static power consumption, especially for L2 caches [19].

4. 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 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.

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	64 KB, 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

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. The characteristics of the simulated systems and the selected values for the parameters K, T_1 and T_2 of the prediction algorithm are given in Table 1. Table 2 reports the benchmarks from the SPEC CPU2000 suite [7] we used in the simulations. The system parameters and the benchmarks together with their running conditions are the same as described in [2].

Table 2: The benchmarks from the SPEC CPU2000 suite used to evaluate the Way Adaptable technique

	Phase			Phase	
SPECINT2000	FFWD	RUN	SPECFP2000	FFWD	RUN
gcc	2.367B	300M	mgrid	550M	1.06B
mcf	5.0B	200M	mesa	570M	200M
parser	3.709B	200M	applu	267M	650M
perlbmk	5.0B	200M	art	267M	200M
bzip2	744M	1.0B	galgel	4.0B	200M
twolf	511M	200M	equake	4.459B	200M

Figure 5 and Figure 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 workloads 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 2.25% performance loss (last bars in Figure 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 37.1% (last bars in Figure 6).

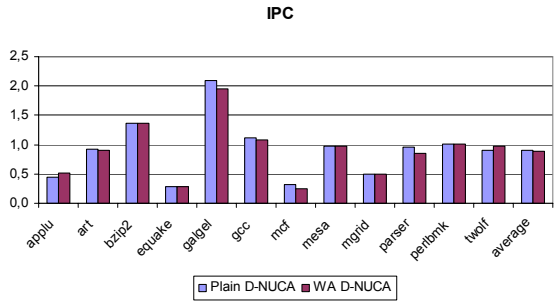

Figure 5: The IPC (Instructions Per Cycle) achieved by the D-NUCA structures considered in our evaluation for the different benchmarks and on average.

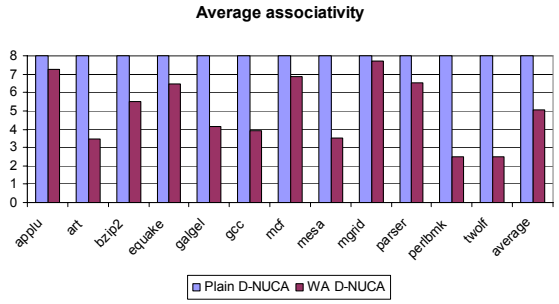

Figure 6: The average number of active ways for the D-NUCA structures considered in our evaluation for the different benchmarks and on average.

A quantitative evaluation of the energy consumption of the L2 cache and of the main memory subsystem has been performed assuming the 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 [24]. Figure 7 shows the normalized energy consumption for Way-Adaptable and plain D-NUCA caches under the given workload. 11 benchmarks out of 12 benefit from the application of the proposed technique, with an average reduction of energy consumption by 34.6%.

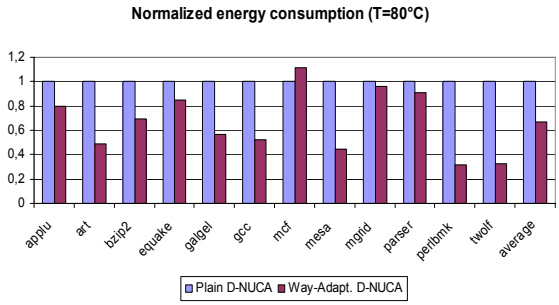

Figure 7: Energy consumption (normalized with respect to plain D-NUCA) for plain and Way-Adaptable D-NUCA caches at a fixed temperature of 80°C. A significant reduction of energy consumption achieved by the proposed technique. Only *mcf* exhibits an increase of the energy consumption since the longer execution time (due to performance degradation) introduces extra static energy consumption, which is not balanced by the reduction of active ways.

The EDP (Energy Delay Product) [23] is a representative metric to compare the energy-performance trade-off achieved by different micro-architectural solutions. Figure 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.

As one would expect, turning off one or more ways also helps to reduce 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 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 the number of requests for the Way Adaptable case.

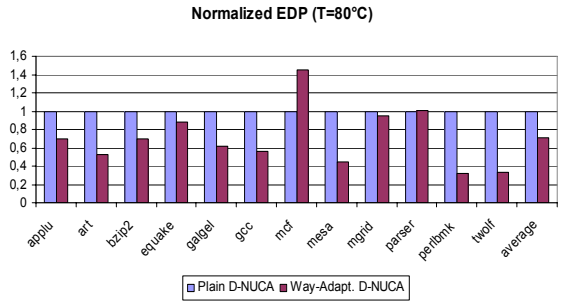

Figure 8: Comparison of achieved EDP (normalized with respect to plain D-NUCA) for plain and Way-Adaptable D-NUCA caches at a fixed temperature of 80°C.

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 detection 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

5. Algorithm Parameters Estimation

The values for T_1 and T_2 thresholds reported in Table 2 were heuristically determined. The chosen values lead to a good average behavior of the Way Adaptable D-NUCA cache. However, the accuracy of the prediction algorithm and the performance/power trade-off depend on such parameters. In particular, in this section we introduce a strategy that is suitable to estimate the values of thresholds T_1 and T_2.

In the algorithm (Figure 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 workload that, among all the possible sequences, is optimal for a chosen 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 evaluate). So an incremental approach can be adopted: the program's execution is halted every K cache hits and the current execution state is saved; then, from this state, 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 that leads to a miss-rate within 1% of the lowest achievable miss-rate, meaning that we choose the event that allows the highest energy consumption reduction and that 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 workload. Figure 9 depicts some steps of the described strategy.

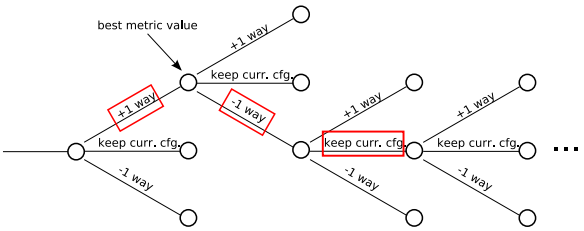

Figure 9: Application of the procedure to identify the pseudo-optimum reconfiguration sequence for a simple case. In the example, the selected sequence is: {+1 way; -1 way; keep current configuration}.

At each step, the selected reconfiguration event places restrictions on the values of T_1 and T_2, so in the end, we obtain a set of inequalities for the two thresholds; by solving them, we obtain values for T_1 and T_2. If they are 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 step, so the reconfiguration sequence results to be a pseudo-optimal one.

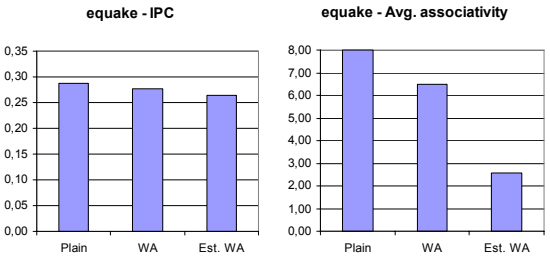

Figure 10: IPC and average associativity for the *equake* application, comparing plain D-NUCA, Way Adaptable D-NUCA with heuristic thresholds (WA) and Way Adaptable D-NUCA with thresholds estimated with the described strategy (Est. WA).

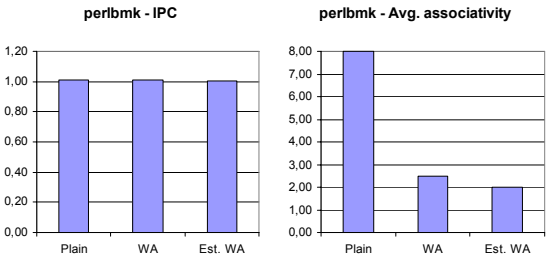

Figure 11: IPC and average associativity for the *perlbmk* application, comparing plain D-NUCA, Way Adaptable D-NUCA with heuristic thresholds (WA) and Way Adaptable D-NUCA with thresholds estimated with the described strategy (Est. WA).

Figure 10 and Figure 11 show the IPC and the average associativity for two benchmarks, *equake* and *perlbmk*, comparing the plain D-NUCA scheme, the original Way Adaptable scheme and the Way-Adaptable scheme with the T_1 and T_2 thresholds calculated according to the described strategy (that we will refer as *Est. Way Adaptable* in the following). Using such thresholds 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 (Figure 12) and the average associativity (Figure 13) for the entire benchmark set, first using a Way Adaptable D-NUCA scheme with thresholds estimated using the *equake* application as workload (Est. WA D-NUCA 1), then

applying a Way Adaptable D-NUCA scheme with thresholds estimated with the *perlbmk* as workload (Est. 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 have been estimated with a different workload, the algorithm exhibits a low sensitiveness and a good effectiveness when applied to the actual one. The use of the proposed heuristic values is not disadvantageous and still allows to obtain a relevant static power reduction and limited performance loss.

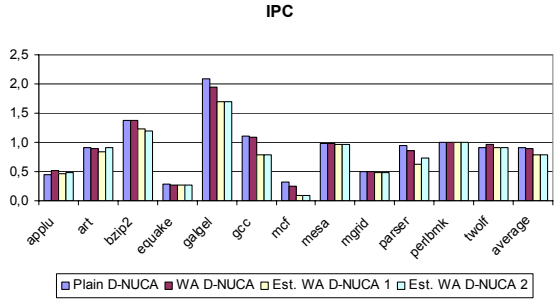

Figure 12: IPC for the entire benchmark set comparing plain D-NUCA, Way Adaptable (WA), Est. Way Adaptable (Est. WA D-NUCA 1) with T1 and T2 calculated for *equake*, Est. Way Adaptable (Est. WA D-NUCA 2) with T1 and T2 calculated for *perlbmk*.

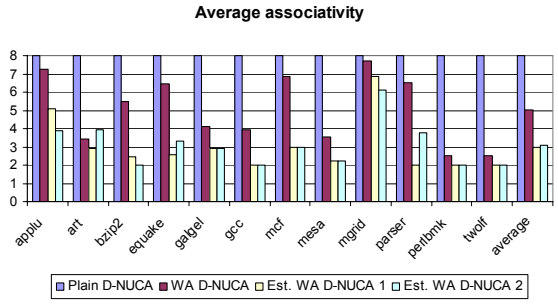

Figure 13: Average associativity for the entire benchmark set comparing plain D-NUCA, Way Adaptable (WA), Est. Way Adaptable (Est. WA D-NUCA 1) with T1 and T2 calculated for *equake*, Est. Way Adaptable (Est. WA D-NUCA 2) with T1 and T2 calculated for *perlbmk*.

6. Way Adapting in the Multicore Environment

NUCA caches are expected to be employed as shared cache memories, especially in the multicore environment where memory requirements of the running workloads are higher than in the single core case. In this section, we show the effectiveness of the Way Adapting technique in such an environment. In particular, we have examined a CMP architecture composed by two Alpha 21264 cores employing a shared L2 D-NUCA cache identical to the reference one adopted for the single core environment. We evaluated the system by further modifying the Sim-Alpha simulator in order to support multiprogrammed execution and by modeling the architecture shown in Figure 14. The two cores are connected to the L2 cache controller through a shared bus. The requests coming from the cores are injected into the D-NUCA from the same side as in the single core architecture. Thus, the same proposed prediction algorithm can be applied also in the multicore environment. The simulation parameters of the multicore system are identical to the one used in the single core (Table 1). For the sake of comparison, we adopted the same heuristic values for T_1 and T_2 used for the single core.

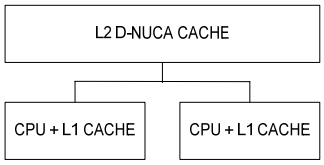

Figure 14: The architecture employed for the evaluation of the Way Adaptable technique in the multicore environment

As for the benchmark selection, we simulated the concurrent execution of couples of applications from the SPEC CPU2000 suite that were representative of different memory load conditions. In order to identify such couples, we considered the average associativity exhibited in the single core Way Adaptable case as a synthetic measure of the cache resource needs of an application. We have identified the three groups of couples of applications listed in Table 4 (the FFWD and RUN phases of each application are the same as listed in Table 2). The first group is formed by couples of applications whose sum of average associativities in the single core Way Adaptable D-NUCA is lower than 8 (this number represents the associativity of the baseline D-NUCA cache), the second group is formed by couples of applications that have a sum of average associativities that is close to 8 and a third group is formed by applications showing a sum of average associativities greater than 8. Intuitively, if WA1 and WA2 are the number of ways (estimated by the Way Adaptable technique) needed by two applications running in a single core environment, they will need WA1+WA2 ways when running together in the multicore environment. Thus, it is reasonable that the Way Adapting techniques still works well when WA1+WA2 is lower than the number of available cache ways (group1). When the sum WA1+WA2 approaches (group2) or exceeds (group3) the number of available cache ways, it is expected that the chances for shutting down ways are lower or null, and that the

competition for the cache lines exhibited by the two running applications will degrades performances.

Table 4: The couples of benchmarks used to evaluate the Way Adaptable D-NUCA cache in the multicore environment.

Group 1		Group 2		Group 3	
Core 1	Core 2	Core 1	Core 2	Core 1	Core 2
art	art	galgel	galgel	mgrid	mgrid
perlbmk	perlbmk	gcc	gcc	mgrid	perlbmk
art	perlbmk	galgel	gcc	mgrid	galgel

As in the single core environment, we compared the Way Adaptable D-NUCA cache with the plain D-NUCA cache in terms of average associativity and we used the Weighted Speed-Up [30] as a performance metric.

The *Weighted Speed-Up* can be assumed as a metric for quantifying the performance of parallel processing, in which multiple applications execute in parallel on different cores. Let *SingleIPC* be the IPC of an application when it is executed by the reference single core system, and *MultiIPC* be the IPC of the same application when running on a core of the CMP system together with another application on the other core (and in particular sharing the L2 cache). The Weighted Speedup for two applications is the sum of the ratios between SingleIPC and MultiIPC of each application and is given by:

$$Weighted\ SpeedUp = MultiIPC_{App1}/SingleIPC_{App1} + MultiIPC_{App2}/SingleIPC_{App2}$$

Figure 15 reports the Weighted Speed-Up obtained for plain D-NUCA and Way Adaptable D-NUCA (assuming in both cases the plain D-NUCA single core system as baseline) for the chosen benchmarks couples. The comparison of the values of Speed-Up obtained in the two considered architectures allows to evaluate the performance loss in the multicore Way Adaptable case with respect to the multicore plain D-NUCA case. Such loss is limited to 2.24% on average; it is comparable to the one achieved in the single core environment, and doesn't show relevant variations among different benchmarks.

The performance reduction is compensated by the reduction of the average associativity shown in Figure 16. All the considered couples of benchmarks show a reduction varying between 2% and 68%. The reduction is 38% on average, meaning that also in the multicore environment it is possible to effectively adopt the Way Adaptable technique to reduce the static power consumption at the cost of a slight performance reduction. Better results are obtained with couples of applications of the first and the second group; however, as Figure 16 shows, also for couples of the third group the Way Adaptable technique allows a reduction of the average associativity with negligible performance loss.

The average associativity achieved in the multicore execution is lower than the sum of the average associativities achieved in the single core execution (indicated by crosses in Figure 16); this counterintuitive result is a consequence of the migration mechanism of the D-NUCA. In fact, even if the traffic pressure on the L2 cache is increased when moving from the single core to the multicore environment, the promotion mechanism concentrates the most frequently accessed data in the first ways. Such a behavior is shown in Figure 17, which compares the distribution of hits for the *gcc* benchmark over a phase of 10 millions of executed instructions when moving from the single core to the multicore. The increase of the total number of hits translates in an increase of hits in the first ways, that are now more efficiently used, and affects only marginally the number of hits in the remote ways that still can be turned off as in the single core case.

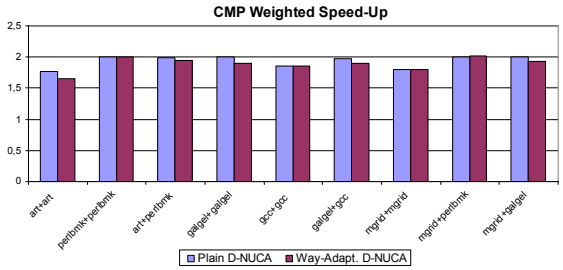

Figure 15: Performance comparisons between D-NUCA and Way Adaptable D-NUCA in the multicore environment. The performance loss is limited to 2.24% on average.

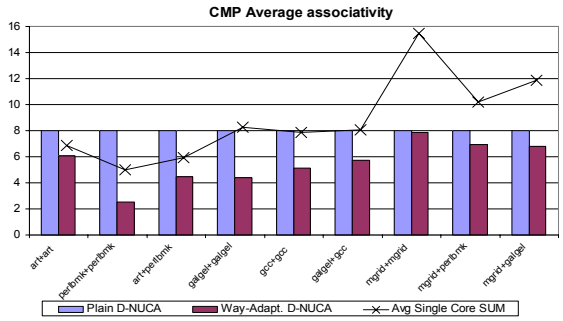

Figure 16: Average associativity comparison between D-NUCA and Way Adaptable D-NUCA. The reduction of associativity is 38% on average. The average associativity obtained in the multicore environment is always lower than the sum of the average associativities obtained by the couple of applications in the single core environment (indicated by the crosses in the figure).

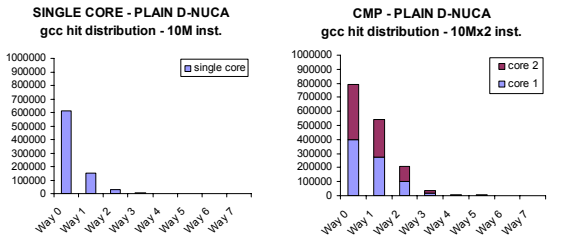

Figure 17: Comparison of hit distributions for the *gcc* benchmark in the single core architecture over a phase of 10 million instructions and for two instances of the *gcc* benchmark in the multicore environment over a phase of 10 million instructions for each core. As a consequence of the increased memory traffic in the multicore environment, the total number of hits is duplicated. However, thanks to the promotion mechanism, the hits are still concentrated in the first ways.

7. Related Work

The D-NUCA cache architecture was proposed in [2] [3], showing that a dynamic NUCA structure achieves an IPC 1.5 times higher than a traditional Uniform Cache Architecture (UCA) when maintaining the same size and manufacturing technology. Extensions to 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]. In the context of on-chip multiprocessors, the behavior of D-NUCA as shared L2 caches has been analyzed in [27][28][29]; these works evaluate the performance of different data mapping and migration policies but none of them explicitly focuses on power and energy aspects.

Many techniques have been proposed to dynamically adapt the cache 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], Drowsy Caches [18] and Slumberous cache [20] 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 latter 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.

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. The direct application of this technique to a D-NUCA would imply the complexity of calculating the LRU/MRU state; furthermore, the random choice of a way to turn off wouldn't take into account the actual data usefulness.

Finally, our work is related to the more general context of detecting program phases [6] to adapt the number of active resources to the need of the running program. In [21] a comparison of ideal and actual implementations of such techniques is proposed.

8. 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 show 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 adaptation 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 37.1%, on average, with respect to a conventional D-NUCA, with a slight 2.25% 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 evaluate the parameters of the prediction algorithm via a pseudo-optimal reconfiguration sequence that can be

determined through simulation. The Way Adaptable technique may effectively be used also in the multicore environment, allowing again a reduction of the average associativity with limited performance loss.

Future works will be focused on further investigation of prediction techniques that are able to better reflect the memory needs of the running application, especially in the multicore environment. In particular, techniques that will be able to turn on or off two or more ways at each step will be evaluated.

9. Acknowledgments

We wish to thank Professor Stephen W. Keckler who furnished us with the initial version of the D-NUCA Sim-Alpha simulator.

10. 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 Comp. Arch., 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. SIA Industrial Association, 2005.

[6] T. Sherwood, B. Calder. Time varying behavior of programs. UC San Diego Tech. Rep. UCSDCS99630, 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, Nov. 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, R. 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, Sep. 2007.

[23] M. Horowitz, T. Indermaur, R. Gonzales. Low-Power Digital Design. Proc. IEEE Symposium on Low Power Electronics, 1994, pp 8-11.

[24] D. Greenhill, J.Alabado. Power Savings in the UltraSPARC T1 Processor. Sun Microsystem Whitepaper. Dec. 2005

[25] N. S. Kim et al. Leakage current: Moore's law meets static power. IEEE Computer, Vol. 36(12). Pp. 68–75, 2003

[26] 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. Solid State Circuits Conf., 2005

[27] J. Huh, C. Kim, H. Shafi, L.Zhang, D. Bourger, S. W. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. Proc. of the 19th Int. Conf. on Supercomputing. Cambridge, MA, June 20 - 22, 2005

[28] B.M Beckmann and D.A. Wood. Managing wire delay in large chip-multiprocessors caches. Proc. of 37th Int. Symp. on Microarchitecture, pp.55-66, San Diego, CA, Dec. 2003

[29] Z. Chisti, M.D. Powell, T.N. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. Proc. of the 32nd Int. Symp. on Computer Architecture , Madison, WI, USA June, 2005

[30] A. Snavely, D.M. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor. Proc. of the 9th Int. Conf. Architectural Support for Programming Languages and Operating Systems, pp.234-244, Cambridge, MA, Nov. 2000.