Revisiting LP-NUCA Energy Consumption: Cache Access Policies and Adaptive Block Dropping

Darío Suárez Gracia, Alexandra Ferrerón, Luis Montesano del Campo, Teresa Monreal Arnal, & Vćtor Viñals Yúfera

May 29, 2017

Abstract

Cache working-set adaptation is key as embedded systems moves to multiprocessor and simultaneous multithreaded architectures (SMT) because inter-thread pollution harms system performance and battery life. Light-Power NUCA (LP-NUCA) is a working-set adaptive cache that depends on temporal-locality to save energy. This work identifies the sources of energy waste in LP-NUCAs: parallel access to the tag and data arrays of the tiles and low locality phases with useless block migration. In order to counteract both issues, we prove that switching to serial access reduces energy without harming performance and propose a machine learning Adaptive Drop Rate controller (ADR) that minimizes the amount of replacement and migration when locality is low.

This work demonstrates that these techniques efficiently adapt the cache drop and access policies to save energy. They reduce LP-NUCA consumption 22.7% for 1SMT. With inter-thread cache contention in 2SMT, the savings rise to 29%. Versus a conventional organization, energy–delay improves 20.8 and 25% for 1 and 2SMT benchmarks, and, in 65% of the 2SMT mixes, gains are larger than 20%.

1 Introduction

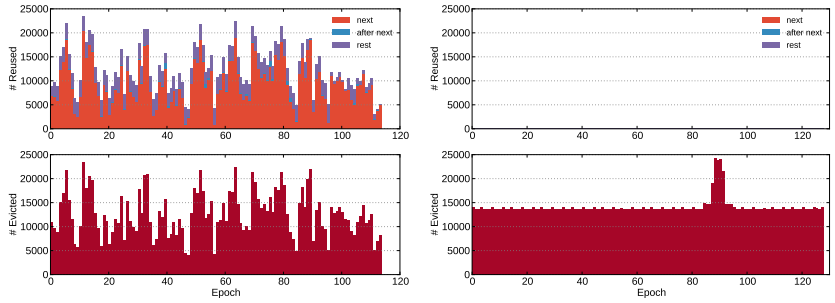
Due to locality of reference, cache hierarchies speed up applications and extend battery life [5]. Recent cache approaches, such as NUCA or LP-NUCA, advocates for heavily tiled designs interconnected with scalable networks [26, 43]. The key feature is that cache blocks are continuously migrating among tiles to keep track of locality at a fine grain level. Unfortunately, in programs or execution phases, with low locality, saving the data in these caches hampers both performance and energy consumption because migrations are useless. The penalty for storing low locality data is twofold: writing data in the cache tiles consumes energy, and these insertions may require the migration, or even the eviction, of blocks with higher locality.

Sharing the cache, either in Simultaneous Multithreading (SMT) or Chip Multiprocessor (CMPs) systems increases the migration harm because low locality threads can easily fill the cache with infrequent used data, trashing hot cache blocks used by other threads. This inter-thread pollution problem has been widely studied for large Last-Level Caches, LLCs, in CMPs [19, 4, 16, 40], but mostly overlooked in the first cache levels of SMT processors, where shared caches are preferable over private ones [46]. Many current embedded processors rely on thread rather than instruction level parallelism to improve their performance–energy ratio. For example, the Intel Xeon LC3528, the MIPS MIPS32-1004K, or the Netlogic XLP832 simultaneously execute between 2 and 4 threads [20, 31, 13].

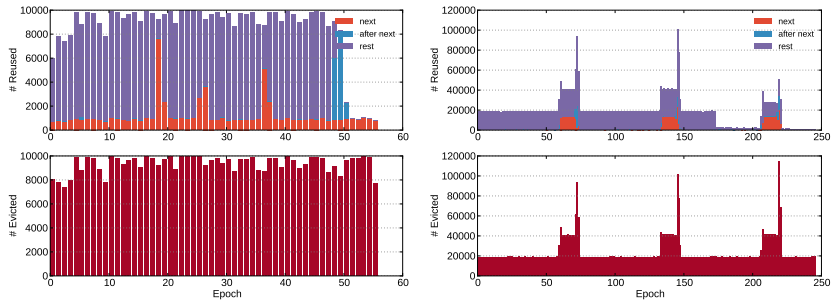
LP-NUCA merges the first two cache levels in a fabric of single processor cycle tiles. Three specialized Networks-in-Cache convey messages among tiles, and the access latency of blocks is proportional to their temporal locality. The LP-NUCA organization self-adapts to working set changes and reduces the memory access time, but at the cost of being very vulnerable to low locality phases and thread pollution. To avoid both issues, we propose two major changes targeting the following energy inefficiencies: (a) the parallel access policy to tag and data arrays in tiles, and, (b) the useless migration present in low locality phases. The switch to serial access and the use of an adaptive controller enable the new design to save energy without reducing performance for single and multi-threaded workloads.

To understand the variation of temporal locality among programs and execution phases, Figure 1 shows the number of block evictions and reuses in an LP-NUCA root tile of 32 KBytes from 4 representative benchmarks of SPEC CPU 2006. A reuse means that an evicted block is referenced again in the future. By construction, reuses are lower than evictions. The upper plots represent the number of reuses and the lower plots the numbers of evictions. Time is divided in 1 million cycle epoch, and each bar represents the amount of evicted or reused blocks during a single epoch. The reuse bars consist of three categories based on the reuse time: *next*, *after next*, and *rest*. *Next* means the evicted block is reused in less than 1 million cycles, *after next* means that the reuse occurs between 1 and 2 million cycles, and *rest* means the block is reused later. In the LP-NUCA, *next* and *after next* reuses are good candidates for being stored in the tiles and *rest* reuses in the LLC. In SMT mode, under high pressure, the dropping of *after next* blocks are preferable over *next* ones.

Some programs, such as 473.astar, Figure 1(a), reuse almost all the evicted blocks in the next epoch. On the contrary, some others, such as 470.lbm, Figure 1(b) behave in a way that no block is reused. Storing these blocks in the hierarchy after the root tile eviction wastes energy. Most programs, such as 456.hmmer or 183.equake, fall in between these two extremes. On the left side, Figure 1(c), 456.hmmer reuses almost every block until epoch 50, when reuse drops significantly, showing that reuses are scattered across time. So if 456.hmmer runs alone, keeping the blocks in the cache may increase performance, but otherwise it may be better to only store its evicted blocks during peak reuse epochs: 18, 19, 26, 36... On the right side, Figure 1(d), 183.equake starts evicting and reusing blocks; however, between epochs 170 to 210, the reuse



(a) 473.astar (b) 470.lbm



(c) 456.hmmer (d) 183.equake

Figure 1: Temporal evolution of the number of block evictions and reuses measured inside 1M cycle epochs

rate greatly decreases, and it is preferable to drop the evicted blocks rather than keep them in the cache. Dropping useless blocks is especially important in shared caches. For example, if 473.astar and 470.lbm run together, without an intelligent control, the latter can completely flush the useful content from the former harming both system performance and energy consumption.

This work extends LP-NUCA in several significant ways. First, we identify that most energy waste occurs during low locality phases and has two primary sources: (a) the continuous replacement of blocks among tiles and (b) the parallel access to the tag and data arrays when the likelihood of a hit is low in loads. Against intuition, the later source represents a larger fraction of the total energy inefficiency in some applications. Second, we analyze the cache access policy inside tiles and demonstrate that performing these accesses in serial, rather than in parallel, will reduce energy without a sacrifice in performance.

Third, in order to decrease consumption without lowering performance, we propose a learning-based adaptive controller relying on local search methods to dynamically select drop rate (percent of blocks that are discarded from the root tile cache). The controller sets a drop rate, recalls information on the accuracy of its choice during execution, and uses the feedback to continuously tune the drop rate.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 describes the LP-NUCA architecture. Section 4 details the methodology. Section 5 explores the impact of serial versus parallel access policy. Section 6 introduces the ADR adaptive drop controller, and Section 7 evaluates it. Section 8 shows how both techniques combine together. Section 9 comments on the system impact of the proposed optimization, and Section 10 concludes.

2 Background

Tullsen *et al.* compare the performance of private and shared L1 caches and observe that regardless the number of threads, shared data caches are always the best choice [46]. Hily and Seznec studied how secondary cache bandwidth limited SMT performance and conclude that: (a) the larger the number of executed threads, the larger the L1 cache size has to be; (b) when the number of threads increases, spatial locality decreases and conflict misses increase; (c) as the number of threads rise, a smaller block size (16-32 bytes), is more effective than increasing the associativity of the cache [17]. To improve SMT performance, Settle *et al.* define a cache partitioning scheme based on column caching. When a cache miss occurs, the replacement algorithm takes the thread id as input restricting the placement of blocks from a given thread to a set of ways [41]. Nemirovsky and Yamamoto analyzed the effect of varying cache capacity, associativity, and line size on miss rate for multistreamed architectures [34]. They observe that increasing both cache capacity and associativity reduces miss rate, specially for small caches, and that large blocks increase miss rate. The Multithreaded Virtual Processor (MVP) is a coarse-grain multi-threaded system with software support that explicitly forces context switching on long latency events such as

cache misses, I/O, or synchronization [27]. For SMT data caches, García *et al.* observed that large associativities reduce inter-thread misses and that XOR-based placement reduces inter-thread miss rate in some cases. Besides, they proposed several organizations combining the hash-rehash caches and static cache splitting [11]. Sarkar and Tullsen proposed two strategies to minimize inter-object data cache misses at compilation time [39]. López *et al.* studied control strategies for reconfigurable caches in SMT GALS processors. They conclude that the best control strategy to maximize performance is to minimize the harmonic mean of the per-thread weighted access time [29]. All this previous work deals with improving performance in workstation/server multi-threaded superscalar out-of-order processors (4 or more SMT) without considering energy. In contrast, our work focuses on energy consumption in embedded processors with limited multi-threading capability.

Architects have proposed a plethora of designs to save cache energy through reconfigurable caches that change their number of ways, sets, or both at run time [2, 3, 37, 47, 44]. Sundararajan *et al.* present a comprehensive view of the state-of-the-art for these techniques [44]. Özer *et al.* studied the fetch resources of SMT processors in soft-real time environments to provide an energy-efficient mechanism for speeding-up a single thread without starving the rest of threads [35]. These works adapt the cache at a finer granularity than our proposal requiring modifications in the cache organization. Increasing the cache complexity in LP-NUCA is not an option because LP-NUCA tiles operate at single processor cycles and the slack is minimal. Besides, reconfiguration techniques that do not increase the cache complexity are orthogonal to the ADR controller and could be applied to provide extra energy gains. Similarly, in large shared LLCs, intra- and inter-thread cache pollution and thrashing have been tackled by modifying either one or both of the insertion and replacement policies [40, 22].

Beckmann, Marty, and Wood proposed Adaptive Selective Replication (ASR) to replicate shared read-only blocks in private L2 caches of CMPs [4]. When the L1 cache evicts a shared clean block, the corresponding ASR module decides whether the block should be replicated into the local L2 bank. The replication level is dynamically adjusted to minimize the average L1 miss latency. The adjustment is done by simultaneously computing the score of the current, next lower, and next higher replication levels with the help of specialized structures. ASR is a clear precedent of our work because it probabilistically stores L1 evicted blocks into local L2 caches to improve performance. Our work uses a similar approach to save energy in a cache closely coupled to the processor. However, since the size of the LP-NUCA tiles is very small, adding extra complexity for simultaneously evaluate multiple states is not appealing. Instead, our proposal requires smaller and simpler structures, and evaluates drop ratios during execution.

Compared with the original LP-NUCA design [43], we introduce a reactive dynamic technique to save energy when the application is not profiting from temporal locality. Previous LP-NUCA energy saving techniques (Sectoring and Miss Wave Stopping) were completely static and application agnostic. This work

analyzes SMT workloads which have not been extensively studied for NUCA cache organizations.

Regarding the learning based approach, the Hill Climbing algorithm has been employed for distributing resources in SMT processors and controlling prefetch aggressiveness [7, 1], but not for dropping cache blocks in first level caches. Optimizing the memory behaviour based on temporal locality has been studied to improve performance of victim caches by only storing blocks with likely reuse [18]. Their policy is based on a fixed threshold, the victim cache only stores those blocks whose dead time is less than 1024 cycles. On the contrary, our proposed ADR controller does not require any threshold to operate properly and self-adjusts itself.

3 Light Power NUCA Operation and Energy Breakdown

Figure 2 shows a Light Power NUCA (LP-NUCA) cache. The LP-NUCA organization merges the L1 and L2 caches into a tiled fabric behaving as a very large distributed victim cache [23]. The root tile, RT, interfaces with the processor, and it is equivalent to an L1 cache, but including the required network components. The L2 cache is split in multiple small tiles that surround the RT and communicate over 3 networks-in-cache (one per activity): Search, Transport, and Replacement. The search network conveys the miss requests from the root tile to the rest of tiles through a broadcast tree (blue network). All tiles receiving a miss request at the same time form a level, and requests progress sequentially among them, until a hit is found or a *global miss* triggers a request to the next cache level. When a tile finds a request in its cache, the block returns directly to the root tile through the transport network (red network). If the corresponding set in the RT is full, a victim block is evicted through the replacement network (black network) to a neighbour tile with the minimum transport latency difference. The delay of all tiles but the RT is one processor cycle and, the tag and data arrays are accessed in parallel for reads. Write hits take two cycles to complete, one for the tag comparison and another to update the data array. A write buffer between the tag and data array enables the back-to-back single-cycle operations.

For example, in Figure 2, the RT evicts to a 3-cycle tile. If necessary, the destination tile will repeat the operation to a tile with transport latency of 4, and this domino operation continues until a tile has an empty way, or a block is evicted from the whole LP-NUCA. These sequences of operations ensure that blocks remain ordered by temporal locality, so the blocks recently evicted from the RT have a lower service latency than those evicted in the past. The performance advantage of LP-NUCA comes from servicing blocks recently evicted faster than conventional or S-NUCA caches. The drawback is that blocks without a nearer reuse waste energy as they traverse multiple tiles before leaving the fabric.

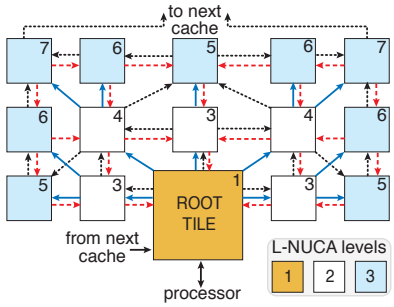


Figure 2: LP-NUCA basic organization with its three Networks-in-Cache: Search in blue, Transport in red, and Replacement in black. The number in the right upper corner of each tile represents its service latency assuming single-cycle tiles

Regarding search activity, LP-NUCA always accesses tag and data arrays in parallel to reduce latency (as most NUCA designs do except NuRAPID [6]). Since in LP-NUCA, a data array access roughly consumes more than 5× the energy of a tag array [43], this parallel policy may account for a major waste of energy for requests that are likely to cause a miss during low locality phases.

To quantify the magnitude of these energy inefficiencies, Figure 3 shows the dynamic energy breakdown of all activities in a 3-level baseline LP-NUCA (excluding the RT) for all programs under test. Section 4 details both the baseline and the workload. For each activity we include the involved network (routers, links, ...) and memory arrays. Transport includes cache hits, search includes cache misses, and replacement includes cache evictions and insertions.

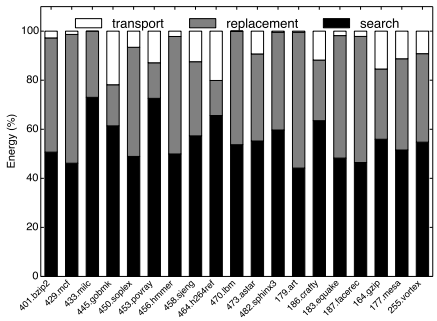


Figure 3: Dynamic energy breakdown of the LP-NUCA activities with parallel tile cache access

Starting top-down, transport, replacement and search have average percents of 7.5, 36.8, and 55.7%, respectively. Two reasons explain these results. First, because the low overhead of the Networks-in-Cache, most energy is spent in the memory arrays, up to 75% of the total energy required by a tile [43]. Transport

spends little energy as it only involves moving blocks through the corresponding network, while replacement and search add a substantial number of cache accesses. Second, replacement requires less energy than search. Notice that a global miss in a 3-level LP-NUCA involves 15 cache lookups, while a chain of replacements, starting in the RT and ending in a corner tile (worst case), performs up to 5 evictions and 5 insertions. Since the energy cost per tile is similar regardless the operation (lookup, eviction, or insertion), a chain of replacements can spend 67% of a global miss at most, but it often consumes less than 10% because RT replacements usually end up in a second-level tile, where previous hits have left empty ways to be filled.

In summary, more than 90% of the total consumption comes from search and replacement activities which are not always useful. With regard to the replacement activity, LP-NUCA incorrectly assumes that programs exhibit temporal locality across all their execution and triggers chain of domino replacements to keep the most recently evicted blocks nearby. So, during low locality phases, and specially in SMT mode, the RT can pollute the rest of tiles with useless blocks and move away useful blocks. Moreover, the parallel access policy may give a small performance advantage, but at the cost of higher energy consumption.

Since the energy waste occurs during low locality phases for search and replacement networks, we need to assert which access policy is the best for LP-NUCA (parallel, serial, or dynamic between both) and devise a mechanism able to drop blocks evicted from the RT when a thread enters in a low locality phase.

4 Methodology

The evaluation environment is based on SimpleScalar with a rewritten memory hierarchy, and the energy estimations for battery-powered devices in 32nm from previous LP-NUCA evaluations (both cross-validated with gate level simulations) [43]. The simulator has been extended to work with several independent threads. The energy consumption of the cache elements without VLSI implementation has been modeled with Cacti 6.5 [32], including the auxiliary tags presented in Section 6.

4.1 Baseline Configuration

The baseline processor resembles the IBM/LSI PowerPC 476FP [14, 30], but executing 1 or 2 threads in SMT mode. Instructions are fetched according to the ICOUNT 2.4 fetch policy [45]. Table 1 summarizes the main characteristics for the processor core and memory hierarchy, including the common L1 and L3 caches, and the tested L2 ones. Namely, we test 4 different second level cache organizations: a conventional L2, an S-NUCA, and the LP-NUCA with parallel and serial cache access, and LP-NUCA with serial access and the ADR controller.

Table 1: Simulator Micro-architectural parameters. BS, AM, lat, and init stand for block size, access mode, latency, and initiation rate, respectively

Clock Frequency	1 GHz	Fetch/Decode/ Commit width	2
Issue width	2(INT+MEM) +2FP	ROB / LSQ entries	32 / 16
INT/FP/MEM IW entries	8 / 8 / 8	branch predictor	bimodal + gshare, 16 bit
Miss. branch penalty	6	Instruction Cache	perfect
L1/L2/L3 MSHR entries	8 / 8 / 4	TLB miss latency	30
MSHR secon. misses	4	Store Buffer/ L2/ L3 WB size[a]	8 / 4 / 4

L1/RT[b]	32KB–4Way–32B BS, parallel AM, write-through, 2-cycle lat, 1-cycle init
L2	512KB–8Way–32B BS, serial AM, 4-cycle lat, 2-cycle init, copy-back
S-NUCA	2×2 128KB–2Way–32BS banks, parallel AM, 3-cycle lat, 3-cycle init, copy-back
LP-NUCA rest of tiles	32KB–2Way–32B BS, parallel AM, copy-back, levels: 3, total size: 448KB

L3	4MB eDRAM–16Way–128B BS, 14-cycle lat, 7-cycle init, copy-back
Main Memory	100 cycles/4 cycle inter chunk, 16 Byte bus

[a] L2, S-NUCA, LP-NUCA, and L3 Write Buffers coalesce entries
[b] In RT, copy-back and write-around

We have also implemented state-of-the-art scan- and trash-resistant replacement policies for the conventional L2: Thread-Aware Static Re-Reference Interval Prediction (TA-SRRIP), and its dynamic version (TA-DRRIP) [22]. As Jaleel *et al.* stated in their paper, these replacement techniques are thought for LLCs with bigger size and higher associativity, where the temporal locality has been filtered by lower levels. In lower levels (L1 and small L2), they do not offer significant performance advantages over the conventional LRU replacement. Thus, for the sake of clarity, we do not include these results in the following sections, and we assume a conventional L2 cache with LRU replacement.

4.2 Workload

Our workload extend the same embedded oriented applications that LP-NUCA previous work with some extra benchmarks to test the controller against more reuse patterns. In SMT mode, we focus on multiprogrammed workloads because they tend to stress the memory hierarchy more than parallel benchmarks as there is no shared data and instructions between the threads.

Table 2 shows all the benchmarks under test from the SPEC CPU2000 and CPU2006 suites. Instead of classifying the benchmarks according to the Misses per Kilo-instruction rate, MPKI, we employ Replacements per Kilo-instruction, RPKI, because the former does not imply a high degree of replacements when the cache does not allocate blocks for write misses. With our L1/RT cache

Table 2: Benchmark characterization. RT-RPKI represents the root tile replacements per kilo instructions, and a high value forecasts a risk of block pollution. RESTT-HPKI represents the number of hits from the rest of tiles, and a high value represents a good reuse. *I*, *F*, *0*, and *6* refer to Integer, Floating Point, SPEC CPU2000, and SPEC CPU2006, respectively

Low RPKI	RT-RPKI	RESTT-HPKI	**High RPKI**	RT-RPKI	RESTT-HPKI
186.crafty *I0*	2.8	2.9	164.gzip *I0*	10.0	17.7
255.vortex *I0*	4.9	4.9	179.art *F0*	130.0	7.4
177.mesa *F0*	2.0	6.6	183.equake *F0*	75.5	21.4
187.facerec *F0*	9.3	3.9	401.bzip2 *I6*	14.8	6.3
445.gobmk *I6*	4.3	12.6	429.mcf *I6*	56.2	14.4
456.hmmer *I6*	6.2	2.1	450.soplex *F6*	28.9	22.0
458.sjeng *I6*	1.2	2.0	453.povray *F6*	12.9	14.1
433.milc *F6*	4.0	0	464.h264ref *I6*	13.2	27.0
482.sphinx3 *F6*	0.21	0.1	470.lbm *F6*	18.1	0.0
			473.astar *I6*	21.7	16.7

[a] In cases where RESTT-HPKI is higher than RT-RPKI, we found a high number of write hits in the rest of tiles. Because the LP-NUCA root tile follows a write-around policy, a write hit in a tile does not trigger a block migration to the root tile.

allocation policies, fetch-on-miss for loads and no-fetch-on-miss for stores, RPKI is exactly the load misses per kilo-instructions (see Table 2 footnote). Note that RPKI offers another additional advantage over MPKI, because it does not account for secondary misses[1]. RPKI measures the flow of blocks from the RT to the rest of tiles, whether the blocks are reused or not. On the contrary, hits per kilo-instruction, HPKI, makes out those benchmarks profiting from the cache, so it is shown in Table 2 as well.

Benchmarks with higher RT-RPKI represent a higher pollution risk and can reduce the cache space for those benchmarks that take profit of the LP-NUCA, those with high RESTT-HPKI. Benchmarks with a high value in both metrics require a careful balance in the controller engine we are going to develop.

Since we have two groups of benchmarks, Low and High RPKI, results are broken down in these two groups for single-threaded, 1SMT, experiments, and in three groups for dual-threaded mixes, 2SMT, namely, Low, Medium and High. Low and High results refer to thread pairs belonging to the same Low and High RPKI group, respectively. Medium results refer to thread pairs not belonging to the same RPKI group.

For each benchmark, we simulate 100M representative instructions selected with the SimPoint methodology [15] with the inputs suggested by Phansalkar *et al.* for SPEC CPU2006 and Sherwood *et al.* for SPEC CPU2000 [42, 36]. In 1SMT, stateful structures, such as branch predictors or caches, are warmed-up for 200M instructions. In 2SMT, we use *last* simulation methodology where a simulation ends when the slowest thread commits 100M instructions. To resemble the load of real system, when a thread finishes, its statistics collection

[1] In processors with Miss Status Holding Registers, the first level data cache can have multiple outstanding misses to the first block, one primary and several secondaries depending on the MSHR organization.

stops, its cache content is invalidated without modifying the replacement stack, and it is re-executed to keep the system load. To guarantee that the initial state for each thread at re-execution do not change, there is no warm-up for 2SMT simulations. We have repeated the experiments without and with warm-up in 1 and 2SMT, and the results were the same because the traces are long enough so the effect of cold structures is negligible.

Regarding measurement methodology, we follow the same approach than Li *et al.* and account for all the energy consumed until the last thread commits 100M instructions [28]. Since our goal is to reduce energy, during its evaluation, our main metric is total energy consumption. We do not normalize the energy results because we have verified than the difference in total executed instructions between configurations is less than 0.8%. For completeness, we also evaluate other metrics such as energy per instruction [12], IPC throughput[2], and fairness [9].

$$IPC\ througput = \sum_{i=1}^{n} IPC_i \qquad fairness = \frac{\min\limits_{i}\left(\frac{IPC_i^{MT}}{IPC_i^{ST}}\right)}{\max\limits_{j}\left(\frac{IPC_j^{MT}}{IPC_j^{ST}}\right)}$$

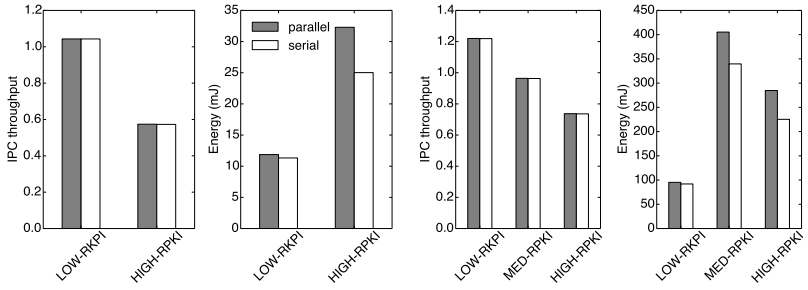
MT and ST stand for multi- and single-threaded, respectively. Fairness ranges between 0, complete starvation, to 1, perfectly fair. This strict definition ensures that different single thread performance across configurations does not affect the results.

5 Tile Cache Access Policy Evaluation

Contrary to most secondary level caches, the original NUCA and LP-NUCA designs access the tag and data arrays in parallel instead of serial. Parallel access reduces access time at the cost of extra energy consumption; however, previous works do not quantify the performance advantage of parallel access if any.

From the point of view of the LP-NUCA VLSI implementation, serial access does not require any main change in the tiles. In fact, writes are always performed serially, and the same circuitry with some additional control suffice to support read serial accesses as well. On the other hand, an ideal access policy should dynamically switch from serial in misses to parallel in hits. LP-NUCA could adopt such dynamic policies leveraging the Network-in-Cache congestion mechanism. Switching between access mode encompasses two operations: (a) choose between parallel or serial, and (b), mark the search request accordingly to the mode. For the first operation, we can use the controller proposed in Section 6 to select between parallel or serial. For marking, it suffices to add an extra access mode bit in the Search Network. In serial mode, the bit disables the accesses to the data arrays in the tiles, and misses propagate back-to-back over the fabric as in

[2]IPC throughput is advantageous because it allows an absolute comparison among configurations. We can use IPC throughput whenever mixes are made from independent threads, because no unpredictable instruction spinning can arise; e.g., before entering a critical section.



(a) 1 SMT (b) 2 SMT

Figure 4: IPC Throughput and Total Energy Consumption for parallel and serial tile access

the conventional LP-NUCA. Nevertheless, if a tile hits during a serial (tag-only) access, the data array has to be accessed. Since we can not stop the request propagation in the Search network because it lacks any flow control mechanism, we set the congestion bit in the request to notify that it must be re-injected again, but in parallel mode. This re-injection feature is already supported to cope with congestion of the Transport network. When a Search request hits in a tile and no output transport link is available, the tile sets the request congestion bit and forwards the request to its leaf tiles. Eventually, the request arrives to the global miss control logic that will reset the serial bit and re-inject the request with parallel access.

The potential of the dynamic approach is bounded by the performance drop from parallel to serial access. In serial mode, when a tile hits, the injection of the block in the transport network occurs one cycle later than in parallel access, but the latency to transport the block to the RT is the same (1 cycle per tile hop). Quantitatively, for a 3-level LP-NUCA, the overhead ranges between 14% and 33% for tiles with latencies 7 and 3, respectively. But since the load to use latency—time elapsed between a load instruction starts executing and the data is ready—is at least 8 cycles, the real overhead is smaller.

Figure 4 shows the IPC throughput and total energy consumption—energy required to execute all the benchmarks one after the other—assuming fixed parallel or serial access, either for one or two threads, 1SMT and 2SMT, Figures 4a and 4b. Independently of the number of threads and RPKI benchmark group, serial has almost identical performance at lower energy, with gains ranging between 13.2% and 31.7% for 1SMT-LOW-RPKI and 2SMT-HIGH-RPKI, respectively.

As regards the dynamic breakout of the activities, if we compare Figure 5, serial access case, with the previous Figure 3, parallel case, we can extract several conclusions. First, switching to serial improves efficiency because, on average, the transport component almost doubles from the parallel version, 14% vs. 7.5%. Second, replacements has overtake search as the largest consumer, 62.2% vs.

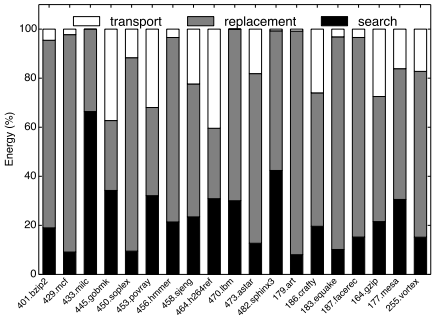


Figure 5: Dynamic energy breakdown of the LP-NUCA activities with serial tile cache access

23.8%, because replacement accesses the data array consumes $5x$ times more than the tag array [43].

Based on all these results, the best access mode for LP-NUCA caches is serial because neither a parallel nor a dynamic policy can provide any advantage for the tested workloads; serial performs almost equal than parallel, and there is no room for improving IPC by switching between serial and parallel. Besides, the consumption of any parallel or dynamic policy will always be higher than the consumption of the serial one.

6 Adaptive Drop Rate Controller Fundamentals

In order to detect low locality phases and drop useless blocks evicted from the RT, we propose to use an Adaptive Drop Ratio Controller, ADR, based on steepest Hill Climbing [38]. Hill Climbing is an iterative optimization algorithm for finding the maximum of a target function $f(d)$, where d represents the set of controllable parameters; i.e., the drop rates. At each iteration, the algorithm modifies a single element of the vector d and checks whether this change improved the value of the target function. When a (local) maximum is reached, hill climbing cannot find any change in d improving the current value of $f(d)$ and terminates. It is worth recalling that our function $f(d)$ is non-stationary and extremely difficult to model analytically. Hill climbing provides a simple yet efficient way to continuously search for the maximum of the target function $f_t(d)$ at time t. Due to non-stationarity, more sophisticated techniques that attempt to find the global maxima as simulated annealing are useless [38].

The set of controllable parameters $d = (dr_1, \ldots, dr_{n_{threads}})^T$ forming the search space comprises all possible combinations of per thread drop rates. Each dr_i represents the fraction of blocks of thread i that the RT drops instead of evict to the rest of tiles or main memory during an epoch. Its range, $dr_i \in [0, 1]$, varies between 0, *no* dropping, and 1, *all* dropping. For example, the default drop rate

of the RT is 0 because all block evictions go the second level tiles [3]. Figure 6 shows the search space for a dual thread execution. The $dr_0, dr_1 = [0.5, 0.5]$ point represents that both threads are dropping 50% of RT block evictions and inserting another 50% into the rest of tiles.

In our implementation, the possible drop rates are discretized with a resolution of Δ. The smaller the Δ, the larger the number of possible drop values, and, hence, the number of states in the search space. The resulting problem is therefore an instance of combinatorial optimization. In Figure 6 Δ equals 0.5, and the search space has nine different states. Selecting the correct Δ is crucial for the ADR because tiny Δ increases the accuracy of the solution but also increases the convergence time. The convergence time is the number of required epochs to reach the best dropping configuration. Also, large Δ may cause oscillations in the optimization process around the optimal drop rate.

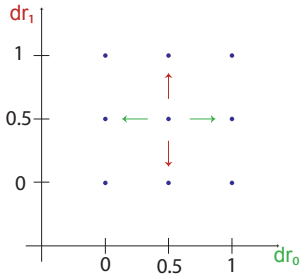


Figure 6: State map for a two thread execution with 3 states, $\Delta = 0.5$. Each dot represents an state, and the arrows represent the possible next drop rate for each thread, green and red for threads 0 and 1, respectively

Hill Climbing searches the space of possible solutions by modifying only one component of d at each iteration, that is, only one thread changes its drop ratio. Similarly to tabu search [8], we incorporate some memory and additional rules to avoid local minima and improve the exploration of the space. Given the discretized representation for d, each drop rate dr_i can be increased, adding Δ, or decreased, subtracting Δ. In the example of Figure 6, $\Delta = 0.5$ and $dr = [0.5, 0.5]$, the drop state in the next epoch can be: $[0, 0.5]$, $[1, 0.5]$, $[0.5, 0]$, and $[0.5, 1]$. The main difficulty for the ADR is that the value of the target function for a given drop state d is unknown and has to be empirically estimated by executing the program with the corresponding drop rates for each thread. As the number of threads increases, the number of neighboring drop states increases exponentially. Since the behavior of the program changes with time, it is possible that, by the time the set of all drop states have been tried, the behaviour of the program in terms of locality has changed. To constrain the number of drop states that have to be evaluated, we use a short memory to limit the exploration of each single drop rate dr_i to a single direction, upwards and downwards, based on whether

[3] In copy-back caches, dropped dirty blocks have to be sent to the next cache level even when the drop rate is 1.

the last drop rate change for thread i improved or not the target function. This information is kept in a direction variable dir_i that can take two values: -1 for downwards (the drop rate of thread i will be decreased by Δ) and 1 for upwards (the drop rate of thread i will be increased by Δ). Therefore, the ADR computes the next drop rate of thread i at age $j+1$ as:

$$dr_{i_{j+1}}^{trial} = dr_{i_j}^{ref} + dir_{i_j}\Delta \tag{1}$$

where $dr_{i_{j+1}}^{trial}$ and $dr_{i_j}^{ref}$ represent the next drop rate and the reference drop rate, respectively.

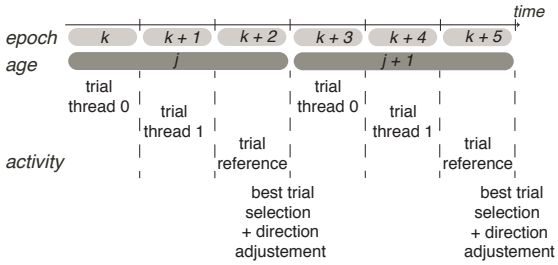


Figure 7: ADR operation phases for a 2SMT system

We are now ready to describe the ADR operation. Figure 7 depicts this process for two threads. The initial reference state is set to no dropping for all threads ($dr_i = 0$, $\forall i = 1..n_{threads}$). Program execution is divided in ages. Each age is formed by $n_{threads} + 1$ epochs. At each epoch, the ADR modifies the drop rate of one thread using Equation 1 in round robin fashion and evaluates the target function. After evaluating the new drop rates for each thread, one last epoch evaluates the current reference state since its previous score may be outdated if the program variability is large, epoch *k + 2* and *k + 5*. When an age ends, the reference state is updated with the combination with higher score among the age epochs, dir^{best}. The direction variables are also updated at the end of each age; dir_i reverts if the epoch score in the target function is lower than the reference one, otherwise it remains equal.

Additionally, to avoid local optimal and speed up the exploration of the space two empirical rules are applied to select the reference state at the end of each age. Both of them reduce the converge time and save energy:

$$dr_{i_{j+1}}^{ref} = \begin{cases} 1\ (all) & if\ \text{rest-tiles hits} = 0 \\ 0\ (no) & if\ \text{RT evictions} < \alpha \\ dr_{i_j}^{best} & \text{otherwise} \end{cases} \tag{2}$$

The rules improve the ADR behaviour specially with small Δs and large number of threads. In this case, it can take a large amount of time until a polluting thread reaches the *all* dropping state. The first rule shortens the convergence time by setting the drop rate of a thread to one when there is no temporal locality. The second rule avoids the problem of programs with low

RT block eviction rate, offering low energy savings potential. For them, it is preferable to inject all the evictions in the rest of tiles because the performance penalty of servicing blocks from the LLC, and the extra static energy consumption may offset the ADR savings. We tested α values between one eight and the full RT capacity in blocks, and the best value is $1/4$ closely followed by $1/8$ and $1/2$.

One key design aspect for the ADR is the epoch length. We can consider time and event based epochs. On the former, the epoch changes after a given number of cycles, and, on the latter, a new epoch triggers when an event, such as the number of RT evictions, reaches a given threshold. Both have advantages and disadvantages, time based is easier to implement, but may provide less accuracy than event based. Subsection 7.1 evaluates both approaches and shows that both performs similarly if the epoch length is selected correctly.

Up to now we have described a generic ADR that can optimize any function that depends on the controllable parameters of each thread. Since our goal is to reduce the cache energy, the first candidate for target function will be cache energy. Nevertheless, energy counters are not mainstream in current processors, and we cannot rely on them for the target function. As a simple yet efficient heuristic, we can think in the cache hits as the target function. First, the hardware overhead will be minimal because current processors already include cache counters. Second, in LP-NUCA, cache hits are a good measure of pollution. When the RT evicts low-locality blocks, they pollute the rest of tiles and most of cache services come from the last level cache, LLC. Since LLC accesses consume more than LP-NUCA ones, a large number of rest-tiles cache hits entails low energy consumption.

Unfortunately, cache hits may not follow a gradient function across program execution. It may be the case that the hit rate decreases because of the execution phase and not because of an incorrect drop rate. Therefore, we need a target function that combines the real hits with the potential ones. Potential hits refer to those hits that would have occurred with a lower drop rate. At first glance, a good target function would subtract the LP-NUCA hits minus LLC ones, but since LLC caches are larger than LP-NUCA, their temporal locality window (amount of time that a block can reside in a cache) is larger and the result will be misleading.

As a solution for getting a traceable function, we propose to use auxiliary tags. The auxiliary tags are an structure that approximates the number of hits that the LP-NUCA would have attained without dropping blocks. It is made of two components: a tag-only cache memory and a hit counter. With them, the ADR keeps track of all dropped blocks and enables the computation of the potential hits. Storing extra tag entries has been used in many applications, such as page allocation in the OS, cache replacement policies, or cache distribution between cores [10, 24].

The ADR performs two operations in the auxiliary tags: (a) *insertion* and (b) *address look-up*. The former, insertion, is triggered every time the ADR drops a block from the RT, and stores the address of the dropped block in the auxiliary tags. The latter, look-up, is triggered only when a request has missed in all LP-NUCA tiles, and it searches in the auxiliary tags array for the requested

address. If found, the potential hit counter is incremented by one, and the entry is removed because the block will be written back in the LP-NUCA. Since the LP-NUCA is small, we will see in Sections 6.1 and 7.3 that a small 1024-entry auxiliary tags structure provides very good performance with little overhead.

Using auxiliary tags, the target function maximizes low energy accesses and minimizes high energy accesses as follows[4]:

$$f(d) = \sum_{i=0}^{n_{threads}} hits_i - k \times aux\_tags\_hits_i \quad (3)$$

where $hits_i$ represents the LP-NUCA hits (low energy consumption), $aux\_tags\_hits_i$ the hits to dropped blocks (high energy consumption) from thread i, and k is a constant that represents the ratio of energy cost between both cases. As a rule of thumb, k can be computed dividing the energy cost of an LLC hit by the search energy of accessing all LP-NUCA tiles minus the root tile and the transport energy of servicing a block to the root tile. To keep the temporal locality window similar to that of the LP-NUCA, the auxiliary tags are flushed at the end of every epoch.

6.1 Hardware Cost

The ADR controller leverages existing processor and LP-NUCA mechanisms to minimize the overhead in terms of latency, area, and energy. Figure 8 shows the controller organization and the interface with the RT and processor. Now, we detail the cost of each component. Starting from the target function, current processors include hardware counters for cache hits, so the computation of the target function only requires a few wires from the cache to drive the cache hit values to the controller.

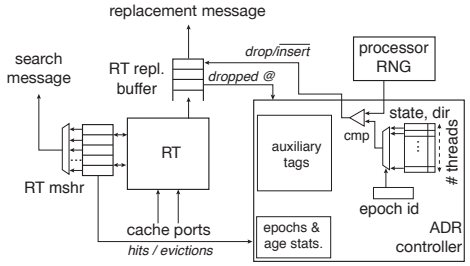


Figure 8: ADR organization. Size does not indicate the complexity

Following with the epoch and statistics, the storage requirements are negligible because the ADR controller stores the result of the target function in a 32-bit

[4]Equation 3 does not make explicit the time dependency of function $f(d)$. The target function is computed over an epoch and its value depends on the code actually executed during that epoch. Consequently, the target function will provide different values for the same d when computed in different epochs.

counter. For example, a 4SMT processor would require 160 bits, 5 epochs $\times$ 32 bits/epoch. Apart from the statistics, the ADR controller stores the reference state and the trial direction for each thread, and requires an epoch counter register. The reference states includes, per thread, one bit for the direction (up or down), and $\log_2(\textit{drop states})$ bits for the potential drop states values. For instance, a 4 SMT processor with 8 drop states would need 4 bits per thread and, thus, 16 bits in total. The epoch identifier register operates modulo the number of threads + 1 arithmetic. In our 4 SMT example, it would require 3 bits. Altogether, the epoch statistics, drop states, and epoch counter register for 4 threads would require 179 bits.

The final component of the ADR controller is the auxiliary tags. The most straightforward implementation is a CAM-based design with a comparator per entry storing the block address of each entry. However, we can take advantage of the LP-NUCA organization to simplify the implementation. The key observation is that if a block would be expelled from the LP-NUCA, the auxiliary tags should do the same thing. Since the global associativity of the LP-NUCA is the number of tiles times tile associativity, we can change the fully associative CAM design with a set associative SRAM design. For example, for a 1024-entry auxiliary tags, a 3-level LP-NUCA with 2-way 32KByte tiles with 32Byte blocks and 40 bit addresses, the size of each entry reduces from 35 to 26 bits and the number of comparisons reduces from 1024 to 28. To reduce even more the number of comparisons and energy, the ADR controller takes advantage of the fact that its look-up latency can be as slow as the next cache level, because this is the minimum round-trip delay (when a block is inserted in the RT, its address is removed from the auxiliary tags). Since look-up latency is the same as LLC latency, the ADR controller employs serial tags [25] and compares the 28 entries in fours. Regarding the area, a 1024-entry auxiliary tags stores 3328 bytes in total, 81% of the capacity of each tile tag array. Since the tag array takes an 8% of a single tile, in a 3 level LP-NUCA, the auxiliary tags would take a 0.5% extra area vs. an standard LP-NUCA.

To choose if a block is dropped or inserted, the ADR controller relies on the processor support as well; in this case, on the cryptographic Random Number Generator engine. RNGs are already included in machines such as Intel Bull Mountain or Sun Niagara T2 [33, 21] and will be commonplace in the near future. To save energy, the ADR controller only interacts with the RNG when a thread is neither in *zero* nor in *all* dropping states because during these epochs, the dropping decision is always the same. For the rest of states, RNG provides an stream of random bits—normally between 64 and 256 bits—that is divided in chunks of $\log_2(\textit{drop states})$. When the RT evicts a block, a chunk is compared against the drop state. When the drop state is smaller than the chunk, the block is dropped. This simple implementation ensures low area and energy costs.

Delay To select the best state, the controller has to evaluate and compare the results of all the epochs. To minimize this delay, most of the work can be done off-line during the evaluation of other threads. Hence, at the end of the last

epoch of each age, the controller already knows which of the previous epoch has the highest score and only computes the score for the last epoch, compares the two values, selects the best, and updates the reference state and the directions. We estimate that these operations can take in the order of 100 cycles. For the sake of completeness, Subsection 7.3 evaluates the impact of the controller delay in the performance.

7 ADR Controller Experimental Evaluation

The ADR controller has multiple parameters that can interfere themselves, so we evaluate them one after another leaving the other parameters as in the baseline configuration. The analysis begins discussing the epoch length and triggering mode, continues with the optimal number of drop states, and finishes assessing the sensitivity to the remaining controller parameters; namely, delay, weight in the target function of the number of hits in the auxiliary tags (k constant), and auxiliary tags size. Table 3 shows the parameters of the baseline ADR controller.

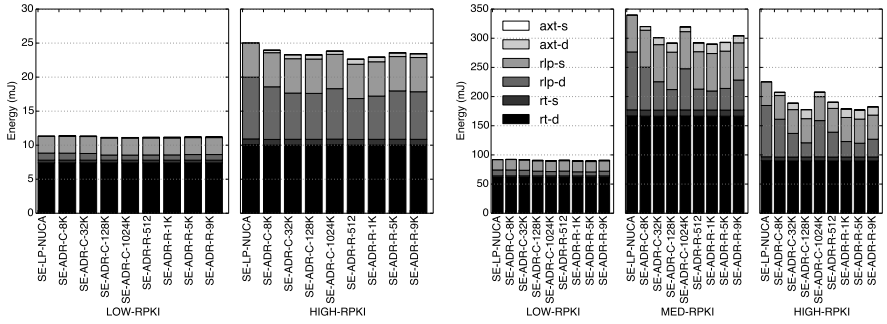
Table 3: Baseline ADR controller configuration

Δ / # States	0.5 / 3	Epoch	1 MCycles	Delay	50 Cycles
Auxiliary Tags	16 KEntries	k constant	10		

7.1 Time or Event based Epochs

In order to narrow the design space for the optimal epoch length, we performed some estimations based on the values from the replacement rates of Table 2. The RT roughly evicts 20 blocks per KInstruction. Assuming an IPC of 1, typical eviction rate would be around 20 blocks per KCycle. This means that around 50 KCycles are necessary to evict all the 1024 blocks of the considered RT (32 KBytes with 32 Byte blocks). Therefore, our cycle-based epoch length experiments explore epochs of 8, 32, 128 KCycles and 1 MCycles. On the other hand, for epochs based on replacement events, we analyze the following number of RT replacements: 512, 1024, 5120, and 9216 that correspond to half, all, 5×, and 9× the number of RT blocks. 5× and 9× are the number of blocks in the second and third level tiles, respectively.

Figure 9 shows the total energy consumption for the LP-NUCA with ADR controllers with the epoch length defined either by number of cycles or replacement events. Each bar stacks the static (s suffix) energy over the dynamic (d suffix) for the auxiliary tags (axt), the rest of tiles (rlp), and the root tile (rt). For all configurations, RT energy is constant because the ADR controller only reduce the dynamic activity of the rest of tiles (rlp-d). The prefix SE indicates that the access policy is serial. Names continue with ADR plus c or r, for time (cycles) and event (replacement) based configurations, and end with the number of cycles or replacements.



(a) 1 SMT (b) 2 SMT

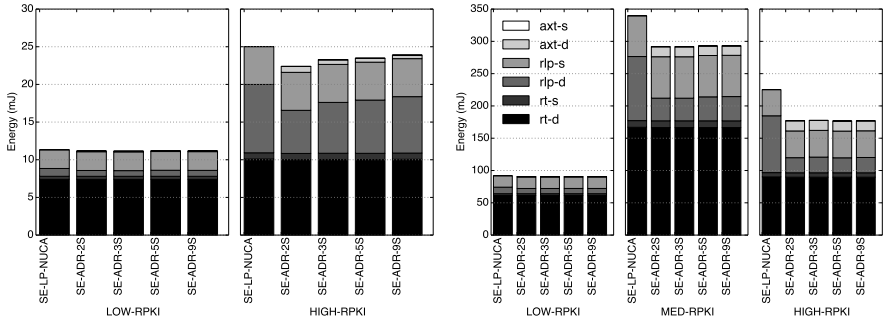
Figure 9: Total Energy Consumption for different epoch length and triggering modes for 1 and 2 threads. rt, rlp, axt refer to the RT, the rest of LP-NUCA tiles, and the auxiliary tags, and d and s refers to dynamic and static consumption

Beginning with 1 SMT, Figure 9a, low-RPKI benchmarks almost offer no potential for reducing energy by dropping useless blocks. For example, ADR-c-1024K and ADR-c-128K reduce LP-NUCA total energy 1.7 and 1.6%, respectively. Results improve for high-RPKI benchmarks, and, the best configuration of each group, ADR-c-128K and ADR-r-512, saves 7 and 9.4%, respectively. Most important, gains occur among all benchmarks; namely, half of them save at least 4.2% energy in any ADR configuration. 2SMT results, Figure 9b, follow the same trend with larger gains due to the reduction of inter-thread pollution. Focusing on med-RPKI and high-RPKI, ADR-c-128K and ADR-r-5K are the best configurations with similar gains, around 14% and 21% reduction, respectively.

Results are similar for both time and event based epochs. ADR-c-128K performs slightly better with larger improvements in high-RPKI benchmark, and it is the best for time based configurations in both 1 and 2SMT. However, in event-based configurations, the best configurations differ, ADR-r-512 and ADR-r-5K for 1 and 2SMT, respectively. Given these results, the simpler implementation tips the scale in favour of the ADR-c-128K time based approach.

7.2 Optimal Number of Drop States

The value of Δ, drop variation between states, controls the variations in the drop rate during the optimization of the controller. It represents a trade-off between resolution and speed and as such can be computed from a given number of states $\Delta = \frac{1}{states-1}$. Figure 10 depicts the total energy consumption for all the configurations under test. Plots show that having large Δ, or few controller states, provide the largest energy savings when the epoch length is 128 KCycles. SE-ADR-2s followed by SE-ADR-3s obtain the best results with little difference between them, 5.0% and 0.2% for total rest of tiles energy in 1 and 2SMT, respectively. Even in 1SMT, SE-ADR-3s has a lower consumption than SE-ADR-



(a) 1 SMT (b) 2 SMT

Figure 10: Total Energy Consumption for multiple Δs: 2, 3, 5, and 9. rt, rlp, axt refer to the RT, the rest of LP-NUCA tiles, and the auxiliary tags, and d and s refers to dynamic and static consumption

2s in more benchmarks than the other way around, but when SE-ADR-2s beats SE-ADR-3s it does by a higher percentage. SE-ADR-2s excels when RPKI is high while low RPKI programs—whose working sets are smaller and, therefore, the risk of dropping a useful block larger—prefer SE-ADR-3s. Lower number of states are preferred because when Δ is high, the effect on dropping is partially diluted with the system noise. For example, in 179.art, ADR-5s only reaches the all dropping state in 10 epochs of the 1220 total epochs, and ADR-9s never reaches that state. A final observation is that Δ size is tied to the epoch length. The same experiment with 1 MCycle epochs shows lower gains because no Δ correctly follows the target function. From now onwards, Δ is set to 1 (2 states).

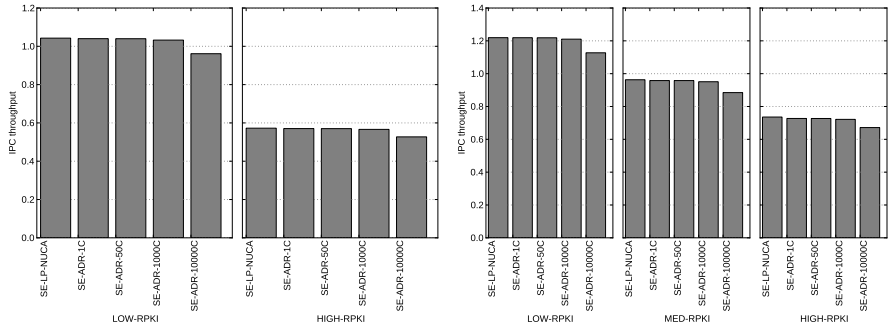
7.3 Sensitivity to Controller Delay, k Constant, and Auxiliary Tags Size

Next, we analyze those controller parameters with lower impact on the behaviour: delay, k constant, and auxiliary tags size.

Controller Delay and Energy At the end of every epoch, the ADR adjusts the drop rates stalling the processor. The critical aspect is the ratio between the controller delay and the epoch length; e.g., a 1 KCycle delay after every 128 KCycle epoch only increases execution time 0.8%. This little overhead impacts neither performance nor energy.

To support this claim, Figure 11 shows the reduction in IPC Throughput as delay increases from 1 to 10000 cycles. Throughput remains constant when the controller delay is lower than 1000 cycles irrespectively of RPKI and number of threads. In other words, the ADR controller drops mostly useless blocks, otherwise performance would be severely affected. Nevertheless, large delays

affect performance, and ADR-10000c reduces IPC between 7.6% and 8.6% for low-RPKI–1SMT and high-RPKI–2SMT, respectively because it adds 7.8% delay overhead to every epoch.



(a) 1 SMT (b) 2 SMT

Figure 11: Harmonic Mean of IPC Throughput for several controller delays and benchmark groups

Since the ADR easily operates in the range of 50–100 cycles (see Section 6.1) and the optimal epoch length is around 128 Kcycles, the controller delay does not penalize the system. Focusing on the controller energy consumption, it has two primary contributors: dynamic and static. The ADR dynamic consumption is small because it only updates some counters and, during the evaluations, performs some comparisons and register updates. On the other hand, there is no rise in the static energy because execution time does not grow[5], and the design assumes a low standby power technology [43].

The first rows of Table 4 summarizes aggregated energy consumption for the different delays. Results are not split into groups because the trends do not change from previous parameters. For each row, the table shows the energy of all LP-NUCA tiles but the RT including the auxiliary tags, and the total energy of the cache hierarchy, including the RT and the L3 cache. Then, each cell contains the values for 1 and 2 threads separated by an slash. All configurations reduce energy, and gains range between 6.1% (Total, ADR-10000c, 1SMT) and 30.5% (Rest-T + Auxiliary Tags, ADR-1c, 2 SMT).

k constant and Energy Regarding the value of the k constant, see Equation 3, we explore three values: 10, 1, and 100. 10 approximates the quotient between a read hit access to the L3 and five times the cost of a tile insertion plus a tile eviction [6]. The numbers 1 and 100 act as more and less aggressive policies, respectively. As the middle part of Table 4 shows, changing k has little variance

[5]Static consumption is directly proportional to execution time.

[6]Five is the number of tiles that a block visits before its eviction to the L3.

Table 4: Effect of controller delay, k constant, and auxiliary tags size in energy consumption

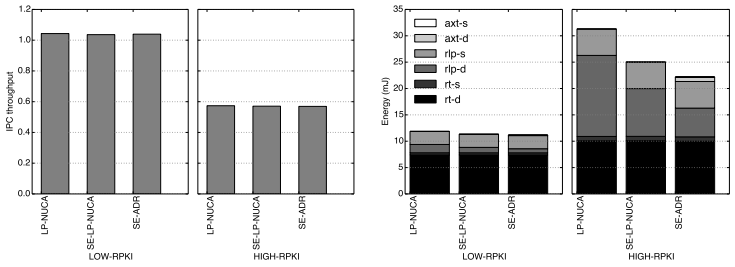
	Energy (mJ)	Rest-T + Auxiliary Tags 1 / 2 SMT	Total 1 / 2 SMT
	SE-LP-NUCA	17.6 / 318.8	36.3 / 656.9
controller delay	SE-ADR-1c	14.8 / 221.4	33.4 / 558.9
	SE-ADR-50c	14.9 / 222.4	33.5 / 559.9
	SE-ADR-1000c	14.9 / 222.7	33.5 / 560.4
	SE-ADR-10000c	15.3 / 231.0	34.1 / 570.2
k constant	SE-ADR-k1	14.8 / 221.6	33.5 / 559.1
	SE-ADR-k10	14.9 / 222.4	33.7 / 559.9
	SE-ADR-k100	15.2 / 222.8	33.8 / 560.2
auxiliary tag size	SE-ADR-1024	14.9 / 221.9	33.5 / 559.5
	SE-ADR-8192	14.9 / 222.4	33.5 / 559.9
	SE-ADR-16384	14.9 / 222.4	33.5 / 559.9
	SE-ADR-32768	14.9 / 222.4	33.5 / 559.9

on energy consumption, and there are only subtle differences in 1 SMT workloads where ADR-k1 performs better.

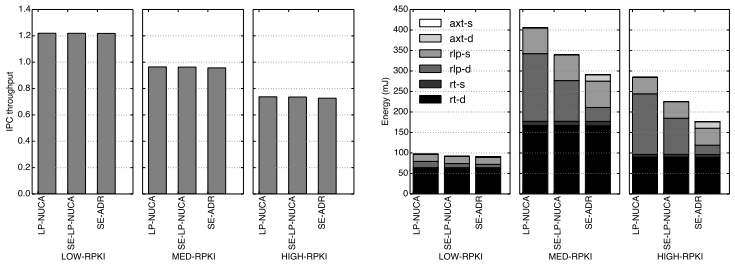
Auxiliary Tags size and Energy Lastly, the lower part of Table 4 shows the energy consumed as the number of entries in the auxiliary tags increases from 1024 to 32768. Results are almost the same irrespective of the size, and surprisingly ADR-1024, having only 1024-entry auxiliary tags, obtains the lowest consumption. Three reasons explain this result. First, due to the temporal locality, the percent of auxiliary tags hits, does not reduce linearly with the size. Second, 75% and 50% of all epochs evict less than 1000 blocks for 1 and 2 threads, respectively. In two threads, around 2200 entries are required to store the addresses of all the evicted blocks during 75% of epochs. Third, it occurs a very interesting effect. During some epochs, the ADR-1024 has a lower number of hits than the rest of configurations, so the controller decides to stay in a high dropping state. The rest of configurations have a higher number of hits in the auxiliary tags and choose to reduce the dropping rate. However, in the following epochs, the LP-NUCA reuse rate reduces and only ADR-1024 is dropping blocks.

7.4 Controller Effectiveness

To verify that the ADR correctly follows the drop gradient, we have performed an experiment that randomly drops blocks with a fixed drop rate of 0.25 and 0.50. These uniform schemes would reduce the energy consumption, if the ADR was unable to track locality. Nevertheless, the best uniform drop, 0.50, consumes 30% more energy only in the rest of tiles. Adding the L3 consumption increases the difference and reinforces the conclusion that random dropping is not effective at all.



(a) 1 SMT



(b) 2 SMT

Figure 12: Overall energy and IPC throughput evaluation of serial tile access and adaptive drop rate controller

8 Combined Analysis of Serial Tile Access and Adaptive Drop Rate Controller

Now, we compare the energy consumption and IPC throughput of the LP-NUCA, a serial LP-NUCA (SE-LP-NUCA), and the combination of serial access and the controller (SE-ADR). In all experiments we will assume the best parameters of the ADR controller 128 Kcycles epoch lenght, Δ and k equal 1, and a delay of 50 cycles.

Figure 12 shows energy and IPC results for 1 and 2SMT. In both cases, neither serial access nor the controller reduces performance, but both reduce energy specially for medium and high RPKI groups. Focussing on the target of the techniques, the energy of rest tiles (including the auxiliary tags) halves for all benchmarks combined, and for each individual benchmark the energy savings are larger than 20%.

9 System Impact

This section analyzes the SE-ADR system impact comparing a broad spectrum of cache hierarchies: conventional cache (CONV-L2), static NUCA (S-NUCA)[7], LP-NUCA, and LP-NUCA with serial tile access and an ADR controller, (SE-ADR). All configurations share the rest of components, including the L3, as described in Section 4, and energy results comprise the L3 cache as well.

Energy per Instruction Many embedded processors run with batteries and demand power efficient cache hierarchies to sustain high throughput for as long as possible. Energy per Instruction, EPI, is a suitable metric for this environment because it represents the required energy to execute the basic work unit of processors (instructions), so the lower the EPI, the larger the device uptime.

Table 5: Average Energy per Instruction (pJ/I)

		CONV-L2	S-NUCA	LP-NUCA	SE-ADR
1 SMT	low	27.2	28.7	20.7	19.9
	high	55.3	53.0	56.9	48.1
	all	42.0	41.5	39.7	34.8
2 SMT	low	28.5	30.9	20.8	19.8
	med	42.8	41.0	38.9	33.0
	high	56.4	53.3	57.3	46.0
	all	43.4	42.1	39.9	33.7

Table 5 shows the average EPI for both 1 and 2 SMT scenarios. In 1SMT, SE-ADR always performs better than the rest of configurations for all programs except 450.soplex, which has a high hit rate in the last level tiles. In this situation, the ADR cannot reduce the migration without reducing the hit rate as we can see in Figure 13. The plot zooms the drop rate and the target function between 30 and 60 millions of cycles. During this interval, the controller is unable to track the gradient because the function is completely non-stationary. In 2SMT, the ADR controller reduces both intra-thread, as in 1SMT, and inter-thread pollution. SE-ADR improves EPI 22.4%, 20%, and 15.5% on average for CONV-L2, S-NUCA, and LP-NUCA, respectively, for the 171 2SMT mixes. Besides the switching to serial access, the 15.5% savings with regards to LP-NUCA also are due to an 85.5% reduction in total block replacements with 1.25% increase in misses. The latter proving the ability of the ADR to detect low locality phases. Since inter-thread pollution increases with the number of threads, the SE-ADR has potential to rise these improvements with larger number of threads in future systems.

Energy–Delay Figures 14a and 14b show the Energy–Delay product relative to the CONV-L2 approach for the 4 cache hierarchies considered in 1SMT and

[7]The static NUCA does not migrate blocks reducing the energy cost of moving blocks among tiles.

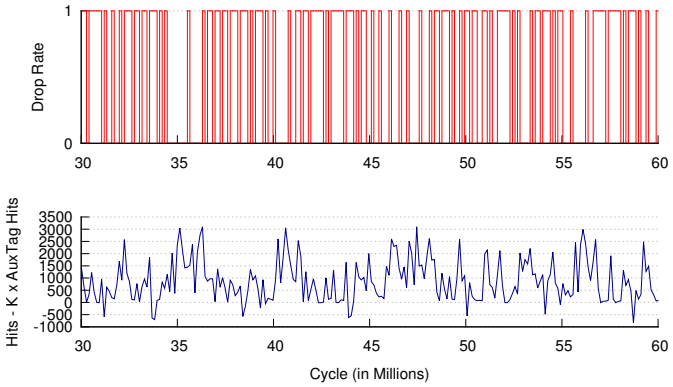
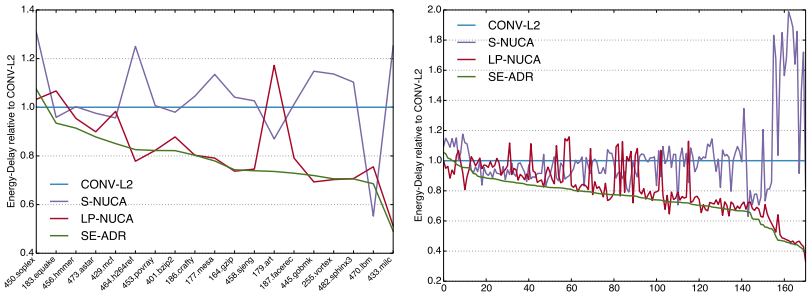


Figure 13: Drop rate and target function evolution in program 450.soplex

2SMT environments, respectively. Single benchmarks, in 1SMT, and mixes, in 2SMT, are sorted from SE-ADR lowest to highest improvement regarding CONV-L2.



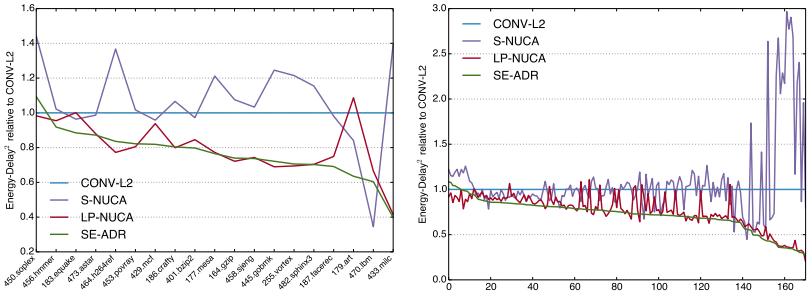
(a) 1SMT (b) 2SMT

Figure 14: Energy–Delay relative to CONV-L2 per benchmark for 1 and 2SMT. Results are sorted from SE-ADR lowest to highest gain relative to CONV-L2. Lower is better.

Starting with 1SMT, Figure 14a, S-NUCA performs irregularly; well in 470.lbm, but bad in 450.soplex, 464.h264ref, and 433.milc. Both LP-NUCA and SE-ADR perform better except for some benchmarks with high reuse in the last level tiles. Even in those, savings has improved, and the worst consumption relative to the CONV-L2 reduces from 17% to 7.5%. In terms of gains, SE-ADR is better than LP-NUCA in all but 4 benchmarks. In 2SMT, SE-ADR gain excels more as shown in Figure 14b. In all but 5 mixes, SE-ADR achieves a better ED than the conventional CONV-L2, and in 65% of them the improvement is larger than 20%. Also, the SE-ADR reduction of inter-thread pollution improves

the results of the combinations where the LP-NUCA performs worse than the conventional approach, CONV-L2. S-NUCA static placement suffers when memory pressure rises and performs up to 98% worse than the CONV-L2. On the contrary, the SE-ADR reduces the LP-NUCA worse energy-delay results from 16% to 4.8%. Not surprisingly, SE-ADR bigger gains relative to LP-NUCA occur in pairs where each benchmark belongs to one of the groups (high and how RPKI), such as 433.milc and 164.gzip, or where the two benchmarks belong to high-RPKI group, such as 464.h264ref and 179.art.

Energy–Delay^2 For systems where performance stands out over energy (set-top boxes, routers, ...) ED^2 is a metric because its bias towards delay. In our particular case, ED^2 proves that SE-ADR energy improvement does not carry along a loss of performance. Figure 15a shows the ED^2 relative to the baseline CONV-L2 for all single thread applications. In single thread applications, S-NUCA has the absolute maximum gain in 470.lbm with 65.5% improvement; however, on average, it performs 6.7% and 45.2% worse than CONV-L2 and SE-ADR, respectively. Finally, SE-ADR outperforms LP-NUCA by 3.6% with improvements up to 42.0% in 179.art.



(a) 1SMT (b) 2SMT

Figure 15: 1 and 2SMT Energy–Delay^2 per benchmark sorted from lowest to highest SE-ADR to CONV-L2 relative gains

In 2SMT experiments, Figure 15b, the trend remains the same. The behaviour of S-NUCA is the most erratic, and it shows the largest dispersion relative to CONV-L2, with a 56% maximum improvement and 196% decline in 470.lbm–473.astar and 433.milc–482.sphinx3, respectively. LP-NUCA and SE-ADR improve that gains, and SE-ADR also reduces variance. Namely, the worst LP-NUCA decline is 10%, 179.art–473.astar, while SE-ADR improves ED^2 in all combinations, 14.4% in this particular case. Most importantly, SE-ADR overpasses CONV-L2 by more than 25% in 51.4% of the combinations.

Fairness Previous metrics may provide distorted results in terms of resource distribution or fairness; e.g., IPC throughput reflects the amount of work per unit

of time regardless if some threads are starving. To prove SE-ADR improvements are not due to a prioritization of some threads, we have computed the fairness of all 2SMT mixes as described in Section 4.2. The SE-ADR does not affect the fairness of the LP-NUCA, and in 81% of the mixes, SE-ADR overpasses the CONV-L2 and S-NUCA. In a few thread mixes, the SE-ADR fairness is worse than the CONV-L2 fairness. These mixes are made up of benchmarks with the largest IPC improvements compared with CONV-L2 executing in single thread mode. In other words, when running in single thread mode, their IPC is almost ideal, so when they run shared their IPC reduces a lot.

10 Conclusions

LP-NUCA is a tiled cache organization that improves performance by keeping cache blocks ordered by temporal locality. Ordering requires continuous migrations of blocks among tiles and makes LP-NUCA vulnerable to waste energy during polluting phases for single and multithreaded workloads.

This paper analyzes the tile cache access mode and proposes and Adaptive Drop Ratio Controller to reduce dynamic energy in LP-NUCA caches. We demonstrate that serial access reduces energy without harming performance. Also, we propose a hill climbing based controller detecting low locality program phases, so that useless blocks are silently dropped during them. Dropping saves energy and avoids the eviction of more useful blocks. The controller implementation is straightforward and requires a negligible amount of area.

Through an extensive parameter evaluation, we prove that the ADR works well for all kinds of applications and that its effectiveness does not depend on the configuration parameters. In single thread mode, a LP-NUCA with serial access and ADR controller, SE-ADR, reduces LP-NUCA energy consumption 22.7%. When inter-thread cache contention appears in 2SMT, the energy savings rise to 29%, the controller reduces the total number of migrations 81% and only increases miss rate 1.7%. If the full cache hierarchy is considered, the SE-ADR improves energy–delay and energy–delay^2 versus conventional, static NUCA, and LP-NUCA cache organizations. Namely, the SE-ADR improves ED 20.8 and 14.1% versus the conventional and static NUCA in single threaded applications. In 2SMT, the improvement rises to 25% and 23%.

Finally, note that the use of the ADR controller is not restricted to LP-NUCA, and could be used for example to filter victims into the L3 cache of the IBM Power7 or to migrate important data between the exclusive L2 and L3 caches of the AMD Bulldozer. This extension is not straightforward, because these block streams present different locality properties.

ACKNOWLEDGEMENTS

The authors would like to thank Laura Neville, Jimi Xenidis, Jorge Albericio Latorre, and the anonymous referees for their helpful comments and suggestions.

References

[1] Jorge Albericio, Rubén Gran, Pablo Ibáñez, Víctor Viñals, and Jose María Llabería. Abs: A low-cost adaptive controller for prefetching in a banked shared last-level cache. *ACM Trans. Archit. Code Optim.*, 8(4):19:1–19:20, 2012.

[2] David H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO 32: Proc. of the 32nd annual ACM/IEEE int'l symp. on Microarch.*, pages 248–259, 1999.

[3] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor arch.s. In *MICRO 33: Proc. of the 33rd annual ACM/IEEE int'l symp. on Microarch.*, pages 245–257, 2000.

[4] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. Asr: Adaptive selective replication for cmp caches. In *Proc. of the 39th Annual Int'l Symp. on Microarchitecture*, pages 443–454, 2006.

[5] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54:67–77, May 2011.

[6] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache arch.s. In *Proc. of the 36th Annual IEEE/ACM Int'l Symp. on Microarch.*, page 55. IEEE Computer Society, 2003.

[7] Seungryul Choi and Donald Yeung. Hill-climbing smt processor resource distribution. *ACM Trans. Comput. Syst.*, 27(1):1–47, 2009.

[8] F.Glover. Tabu searchpart i. *ORSA Journal on Computing*, 1(3):190206, 1989.

[9] Ron Gabor, Shlomo Weiss, and Avi Mendel son. Fairness and throughput in switch on event multithreading. In *Proc. of the 39th Annual IEEE/ACM Int'l Symp. on Microarch.*, MICRO 39, pages 149–160, 2006.

[10] Hongliang Gao and Chris Wilkerson. Dueling segmented lru replacement algorithm with adaptive bypassing. In *Proc. of the 1st JILP Workshop on Computer Arch. Competitions: Cache Replacement Championship"*, pages 1–4, 2010.

[11] Montse García, José González, and Antonio González. Data caches for multithreaded processors. In *Proc. of the Workshop on Multithreaded Execution, Arch. and Compilation*, 2000.

[12] Ed Grochowski and Murali Annavaram. Energy per instruction trends in intel® microprocessors. *Technology@ Intel Magazine*, 4(3):1–8, 2006.

[13] Tom R. Halfhill. Netlogic broadens XLP family. *Microprocessor Report*, 24(7):1–11, 2010.

[14] Tom R. Halfhill. The rise of licensable SMP. *Microprocessor Report*, 24(2):11–18, 2010.

[15] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0: Faster and more flexible program analysis. In *Proc. of Workshop on Modeling, Benchmarking and Simulation*, 2005.

[16] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive nuca: near-optimal block placement and replication in distributed caches. In *ISCA '09: Proc. of the 36th annual int'l symp. on Computer arch.*, pages 184–195, 2009.

[17] Sébastien Hily and André Seznec. Contention on 2^{nd} level cache may limit the effectiveness of simultaneous multithreading. Technical Report 1086, IRISA, fébrier 1997.

[18] Zhigang Hu, Stefanos Kaxiras, and Margaret Martonosi. Timekeeping in the memory system: predicting and optimizing memory behavior. In *Proc. of the 29th annual int'l symp. on Computer arch.*, pages 209–220, 2002.

[19] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. A nuca substrate for flexible cmp cache sharing. In *ICS '05: Proc. of the 19th annual int'l conference on Supercomputing*, pages 31–40, 2005.

[20] Intel Embedded. Intel® Xeon® processor C5500/C3500 series. Datasheet–Volume 1, February 2010.

[21] Intel Software. *Bull Mountain: Software Implementation Guide*, June 2011.

[22] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). In *Proc. of the 37^{nd} Annual Int'l Symp. on Computer Arch.*, pages 60–71, 2010.

[23] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *ISCA '90: Proc. of the 17th annual int'l symp. on Computer Arch.*, pages 364–373, New York, NY, USA, 1990. ACM Press.

[24] K. Kedzierski, M. Moreto, F.J. Cazorla, and M. Valero. Adapting cache partitioning algorithms to pseudo-lru replacement policies. In *Proc. of Int'l Symp. on Parallel Distributed Processing*, april 2010.

[25] R. E. Kessler, R. Jooss, A. Lebeck, and M. D. Hill. Inexpensive implementations of set-associativity. In *Proc. of the 16^{th} annual int'l symp. on Computer arch.*, pages 131–139, 1989.

[26] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive cache structure for future high-performance systems. In *Proc. of the IBM Austin Center for Advanced Studies Workshop*, February 2002.

[27] Hantak Kwak, Ben Lee, Ali R. Hur-son, SukHan Yoon, and Woo-Jong Hahn. Effects of mul-ti-threading on ca-che per-for-man-ce. *IEEE Trans. on Com-pu-ters*, 48:176–184, 1999.

[28] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. Understanding the energy efficiency of simultaneous multithreading. In *Proc. of the 2004 int'l symp. on Low power electronics and design*, ISLPED '04, pages 44–49, New York, NY, USA, 2004. ACM.

[29] Sonia López, Steve Dropsho, David H. Albonesi, Oscar Garnica, and Juan Lanchares. Dynamic capacity-speed tradeoffs in smt processor caches. In *Proc. of the 2nd int'l conference on High performance embedded arch.s and compilers*, HiPEAC'07, pages 136–150, Berlin, Heidelberg, 2007. Springer-Verlag.

[30] LSI Corporation. PowerPC™ processor (476FP) embedded core product brief, `http://www.lsi.com/DistributionSystem/AssetDocument/PPC476FP-PB-v7.pdf`, January 2010.

[31] MIPS Technologies. MIPS32® 1004K™coherent processing system (CPS), 2010.

[32] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. CACTI 6.0: A tool to model large caches. Technical Report HPL-2009-85, HP Laboratories, April 2009.

[33] U. G. Nawathe, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *IEEE Journal of Solid-State circuits*, 43(1):6–20, 2008.

[34] Mario Nemirovsky and Wayne Yamamoto. Quantitative study of data caches on a multistreamed arch. In *In Workshop on Multithreaded Execution, Arch. and Compilation*, 1998.

[35] Emre Özer, Ronald G. Dreslinski, Trevor Mudge, Stuart Biles, and Krisztián Flautner. Energy-efficient simultaneous thread fetch from different cache levels in a soft real-time smt processor. In *Proc. of the 8th int'l workshop on Embedded Comp. Systems: Archs., Modeling, and Simulation*, pages 12–22, 2008.

[36] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. Analysis of redundancy and application balance in the spec cpu2006 benchmark suite. In *ISCA '07: Proc. of the 34th annual int'l symp. on Computer arch.*, pages 412–423, New York, NY, USA, 2007. ACM.

[37] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. Reconfigurable caches and their application to media processing. In *ISCA '00: Proc. of the 27th annual int'l symp. on Computer arch.*, pages 214–224, 2000.

[38] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

[39] Subhradyuti Sarkar and Dean M. Tullsen. Data layout for cache performance on a multithreaded arch. In *Trans. on high-performance embedded architectures and compilers III*, pages 43–68. 2011.

[40] Vivek Seshadri, Onur Mutlu, Michael A. Kozuch, and Todd C. Mowry. The evicted-address filter: a unified mechanism to address both cache pollution and thrashing. In *Proc. of the 21st Int'l Conf. on Parallel Architectures and Compilation Techniques*, pages 355–366, 2012.

[41] Alex Settle, Dan Connors, Enric Gibert, and Antonio González. A dynamically reconfigurable cache for multithreaded processors. *J. Embedded Comput.*, 2(2):221–233, 2006.

[42] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proc. of the 10th int'l conf. on Arch. Support for Prog. Lang. and Operating Systems*, pages 45–57, 2002.

[43] D. Suárez Gracia, G. Dimitrakopoulos, T. Monreal Arnal, M.G.H. Katevenis, and V Viñals Yúfera. Lp-nuca: Networks-in-Cache for high-performance low-power embedded processors. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 20(8):1510 –1523, aug. 2012.

[44] Karthik T. Sundararajan, Timothy M. Jones, and Nigel Topham. Smart cache: A self adaptive cache arch. for energy efficiency. In *Proc. of the Int'l Conference on Embedded Computer Systems: Arch.s, Modeling, and Simulation (SAMOS)*, 2011.

[45] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc.. 23nd Annual Int'l Symp. on Computer Arch.*, volume 24, pages 191–202. ACM, 1996.

[46] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc.. 22nd Annual Int'l Symp. on Computer Arch.*, pages 392–403, Jun 1995.

[47] Chuanjun Zhang, Frank Vahid, and Walid Najjar. A highly configurable cache arch. for embedded systems. In *ISCA '03: Proc. of the 30th annual int'l symp. on Computer arch.*, pages 136–146. ACM Press, 2003.