The Dynamic Block Remapping Cache

Felipe Thomaz Pedroni, Alberto F. De Souza, Claudine Badue
Departamento de Informática
Universidade Federal do Espírito Santo
{fpedroni, alberto, claudine}@lcad.inf.ufes.br

Abstract

In this paper we present a new architecture of Level 2 (L2) cache – the Dynamic Block Remapping Cache (DBRC). DBRC mimics important characteristics of virtual memory systems to reduce the impact of L2 in system performance.

Similar to virtual memory systems, the DBRC uses a hierarchy of tables to map blocks of L2 cache into blocks of physical memory. It also uses a Block-TLB to speedup accesses to previously performed block translations. We verified that the benefits of fully associativity and the consequent possibility of employment of global block replacement algorithms allow hit rates higher than those of equivalent standard caches.

We compare DBRC with standard caches in terms of miss rate, energy consumption and impact on the instruction-level parallelism (ILP) of a simulated superscalar processor. Our results show that DBRC outperforms standard caches in terms of miss rate, energy consumption and impact on ILP.

1. Introduction

Currently, in standard desktop and laptop computers, the time it takes to bring data from memory into the processor registers – the main memory latency – approaches 500 processor cycles and several levels of cache memory are used to diminish the impact of that latency on system performance. Cache memory is growing in size, particularly the level directly connected to the main memory (typically, the level two cache, or L2 for short). These, the size and the latency, are making the L2 cache-main memory interface, as seem by the L2 controller, more and more similar to the main memory-hard disk interface, as seem by virtual memory systems. In this paper, we present the *dynamic block remapping cache* (DBRC), which borrows some ideas from virtual memory systems to reduce the impact of the main memory high latency while reducing L2 energy consumption.

In virtual memory systems, any virtual page can be allocated into any physical page, which makes the main memory a fully associative cache of the disk. A fully associative organization allows sophisticate global page substitution algorithms, which contributes to higher hit rates; this is important because the cost of page faults (in terms of time) is large. Address translations, once performed, are typically saved in a small cache called Translation Look-aside Buffer (TLB, [9]). Later accesses are first directed to the TLB to check for translations, which allows avoiding the cost of examining the hierarchy of tables.

Similar to virtual memory systems, which use a hierarchy of tables to map pages of virtual memory into pages of physical memory, the DBRC uses a hierarchy of tables to map blocks of L2 cache into blocks of physical memory. Most of this hierarchy of tables is stored in L2 itself and, as in virtual memory systems, a Block-TLB, or B-TLB, is used to speedup accesses to previously performed block translations. Thanks to its hierarchy of tables, the DBRC is fully associative and, although fresh translations may take more processor cycles, they are infrequent. Our results show that the benefits of fully associativity and the consequent possibility of employment of global block replacement algorithms allow hit rates significantly higher than those of equivalent standard caches.

We have performed experiments with many configurations of DBRC acting as L2 cache of a Simplescalar based simulated system (www.simplescalar.com), and compared its performance with that of standard L2 caches. Our results show that the DBRC achieves an average miss rate reduction of 27.71% on the SPEC CPU2000 benchmark suite [14] when compared with an equivalent (in size) 8-way set associative L2 cache. This translates into an IPC improvement of 20.34%. In

addition, the DBRC has an energy consumption 80.94% inferior than the standard L2 cache.

This paper is organized as follows. After this introduction, Section 2 presents the DBRC architecture. Section 3 describes our experimental methodology and Section 4 analyzes our experimental results. Section 5 presents related work. Finally, our conclusions follow in Section 6.

2. The Dynamic Block Remapping Cache

Figure 1 presents the Dynamic Block Remapping Cache (DBRC) architecture. It is composed of five parts: (i) data block array (DBA), (ii) block table hierarchy (BTH), (iii) data utilization table (DUT), (iv) tag table (TT), and (v) block translation lookaside buffer (B-TLB).

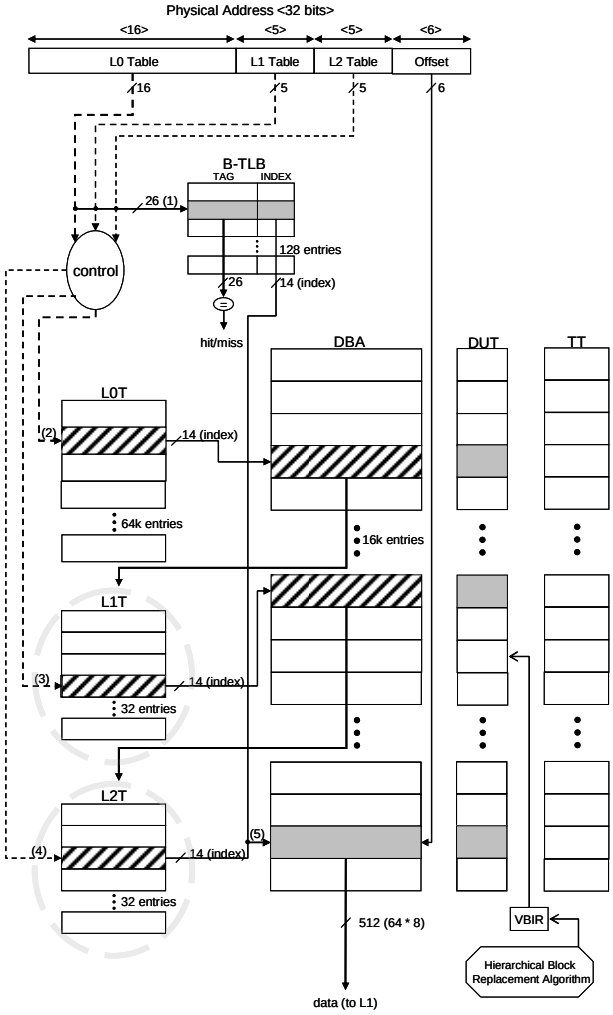

Figure 1. DBRC architecture

2.1. Data Block Array - DBA

Data or instructions (from now on we will refer to data or instructions stored in a DBRC simply as data) stored in a DBRC are organized into blocks of fixed size (in our experiments, 64 bytes), which correspond to the pages of a virtual memory system. These blocks form the data block array (DBA) of the DBRC (see Figure 1), which is basically static RAM memory.

To check whether a data block is present in the DBA of a DBRC, its block table, equivalent to the page table of a virtual memory system, is examined. In fact, as in current virtual memory systems, this block table is built as a hierarchy of tables (the block table hierarchy – BTH – see Figure 1) and can, depending on the configuration, have three, four or even five levels, as described below.

2.2. Block Table Hierarchy – BTH

Figure 2 presents a diagram of a five level BTH. As the figure shows, in order to check if a datum mapped to a given physical address is present in the DBRC, the physical address is divided in several parts, and each part addresses one level of BTH. The number of parts, or levels of BTH, and their sizes in bits depends on physical address size (in bits) and DBRC configuration. We evaluated experimentally DBRCs with BTHs with three, four and five levels.

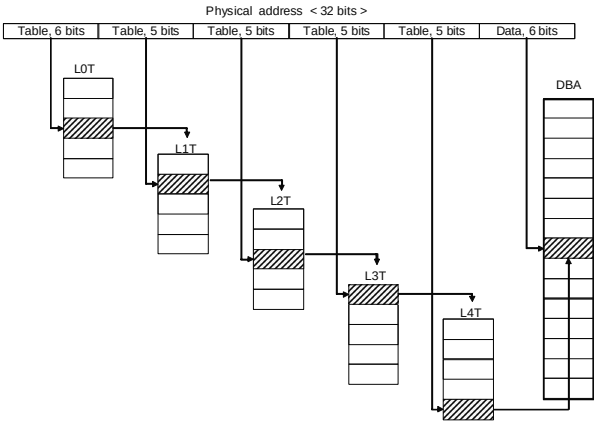

Figure 2. Five Level Block Table Hierarchy (BTH)

In the DBRC we modeled, the level 0 of BTH (L0T) is physical, i.e., it is implemented in hardware to allow low DBRC latency. The other levels (L1T, L2T, L3T and L4T of Figure 2) are logical and stored in DBA. The L0T is the first table accessed when BTH is examined in order to find a data block in DBRC. Each entry of L0T has a valid bit (*V*) and an index field (*I*) to child tables that may point to any block of DBA. This block hopefully will contain a L1T table (we describe

DBRC misses further on). The entries of a L1T or any other level of BTH have the same format as the entries of L0T. So, to find a data block in DBA, each level of BTH is examined in sequence until the data block is reached in DBA.

2.3. Data Utilization Table – DUT

The DUT stores information related to the type and frequency of reutilization of each DBA block.

For each DBA block there is a DUT entry and each of those has: a valid (*V*), dirty (*D*), and lock (*L*) bits; a field that indicates the level of the BTH table eventually residing in the block (*LF*); a parent BTH table valid bit (*PV*); and a saturated counter that registers the frequency of reutilization of the block (*R*). This information is required by the block replacement algorithm, which also requires a pointer, stored into the victim block index register (VBIR – see Figure 1), which points to the DUT entry (and associated DBA block) that is the current candidate for replacement. We describe the global block replacement algorithm further below.

2.4. Tag Table – TT

The TT holds the main memory physical address tag of each DBA block and information required by the global block replacement algorithm. Therefore, as in the case of DUT, there is a TT entry for each DBA block. Each TT entry has two fields: *TAG* field and parent table field (*PT*). The *TAG* field stores the high order bits of the physical address of the main memory block stored into the DBA block, and the *PT* field stores an index to the block in DBA that holds the BTH parent table that has the pointer to the DBA block.

The importance of the TT *PT* field and DUT will became apparent with the description DBRC accesses. But, before going into that, let's briefly present the B-TLB.

2.5. Block Translation Lookaside Buffer – B-TLB

Similar to the TLB of virtual memory systems, which stores translations from virtual page numbers to physical page numbers, the B-TLB stores translations from main memory physical addresses to DBA block indexes. It is a small fully associative cache (128 entries in our experiments) and each one of its entries has: a *TAG* field that stores the high order bits of the main memory physical address, an index (*I*) to a DBA block, and a valid bit (*V*).

B-TLB *I* fields point to DBA blocks holding data or BTH tables. As we will see in Section 4, it is more advantageous to make B-TLB *I* fields point to tables of a specific level of BTH instead of data blocks. The best performing DBRC we have simulated has a five level BTH and its B-TLB points to BTH level 4 (L3T).

2.6. DBRC Accesses

An access to a DBRC starts with a check if B-TLB already has a translation from the main memory physical address to a DBA block index. If it has, the DBA block indexed by B-TLB is retrieved and, depending on the level it points to: (i) an entry of a BTH table is retrieved; or (ii) a data block is delivered to the requester (typically the L1 cache controller) in case of a DBRC read, or it is modified in case of a write. Case (ii) is a DBRC hit, and case (i) may be a hit or a miss, depending on BTH, which has to be partially examined in this case.

If a translation is not available in B-TLB (a B-TLB miss), the level 0 of BTH (L0T) is examined; note that an access to L0T typically occurs in one processor cycle and can be done in parallel with the B-TLB access, which also take one cycle. If the addressed L0T entry is valid (if its valid bit, *V*, is set), its index *I* is used to access a block of DBA in search for the corresponding L1T. The L0T index, *I*, and corresponding main memory physical address L1T displacement (see Figure 2) point to a possible L1T entry in DBA, which is retrieved for examination – this takes several processor cycles.

The L0T index *I* is also used to retrieve an entry of DUT. The *R* field of this DUT entry is incremented and the L1T entry is used to access DBA in search for the next level of BTH; otherwise, it is a DBRC miss. The accesses to the other levels of BTH follow the same principles. If there is no miss, this also constitutes a DBRC hit, although a more expensive one. The information gathered in the process is used to update B-TLB, so that it may allow a B-TLB hit next time.

2.7. DBRC Misses

There is a miss in DBRC when it is accessed using a physical memory address for which there is no translation to a DBA block index in BTH. In a five-level BTH, a miss in the first level (L0T) requires allocation of five blocks of DBA: one for each remaining levels of BTH and one for the data block. Misses in lower levels of BTH requires allocation of a proportionally smaller number of blocks of DBA.

Misses in DBRC caused by misses in a given level *N* of BTH are served according to the hierarchical block replacement algorithm (HBRA), shown in Figure 3 and detailed below.

Select a DBA victim block. To select a DBA block to serve a miss (a victim block), the DBRC controller uses DUT's victim block index register (VBIR). This register is initialized with zero at system reset, is

incremented during the process of selecting DBA victim blocks, and points to the current victim block candidate.

```
1      b = Select a DBA victim block;
2      Make the BTH entry in level N point to b;
3      if (b's DUT entry bits V == true and PV == true)
3.1      Invalidate the entry of the BTH table that points to b;
3.2      Invalidate an eventual entry in B-TLB that points to b;
3.3      if (b's DUT entry LF field indicates that b holds a BTH
         table)
3.3.1      Invalidate DUT entries associated with b's children;
3.4      else if (b's DUT entry dirty bit D == true)
3.4.1      Save b contents into physical memory;
4      Install block level N+1;
5      N = N + 1; if (N < data block level) goto 1;
```

Figure 3. Hierarchical block replacement algorithm

So, DBA blocks are victim candidates in a round robin fashion. However, a victim block may be spared depending on its DUT *R* counter (see Section 2.3).

A victim block is selected if it is not locked (DUT *L* != 1) and: (i) its DUT valid bit (*V*) is equal to zero, in which case the block does not contain valid data or BTH table; or (ii) its DUT parent BTH table valid bit (*PV*) is equal zero, in which case the block is orphan because its parent BTH table was removed from DBA; or (iii) its DUT *R* counter is equal zero, which indicates that this block was not reused since the last time it was a victim candidate (VBIR pointed to it).

If none of these three conditions is true, the *R* counter of the DUT entry associated with the victim block is zeroed and VBIR incremented to point to the next DBA block. The whole process is repeated until one of the three conditions occurs or a Maximum Number of Attempts (MNA) is made, in which case the block examined with the smaller *R* counter value is selected.

Make the BTH entry in level *N* point to *b*. A block needs to be selected because an entry in a table of BTH was found invalid during a DBRC access (a BTH miss). Once a block *b* is selected, the *I* field of the invalid BTH table entry is made to point to the selected block *b* and its valid bit, *V*, is set (see Section 2.2).

Invalidate DUT entries associated with *b* and its children. If the *V* and *PV* bits of the DUT entry of the victim block *b* are both set, there is a valid entry of a BTH table that points to it. The DBRC controller invalidates it using the parent table field (*PT*) of TT. In addition, it invalidates an eventual entry in B-TLB that points to *b* using the *TAG* field of TT.

If *b*'s DUT entry *LF* field indicates that it holds a BTH table, all DUT entries associated with *b*'s children have to be invalidated. For that, the DBRC controller reads *b* from DBA and uses the *I* fields of each one of its valid entries to invalidate the *PV* bits of the corresponding DUT entries. This may take several processor cycles. Note that this does not invalidate *b*'s children, which still may be reached via B-TLB; but it makes then strong victim candidates.

Install block level *N*+1. If *b* was selected to receive a data block from physical memory (a read miss), the lock bit (*L*) of the DUT entry associated with it is set. To set the lock bit is necessary because it takes many cycles (100s) to bring data from main memory into *b*. To allow serving other DBRC misses in the mean time, a MSHR (*Miss Status Holding Register* [8, 5]) of the DBRC controller is filled with the information required to write the data coming from memory into *b* when it (the data) arrives.

If *b* was selected to receive a data block from L1 (a L1 writeback), the data is immediately written into *b* and the dirty bit (*L*) of the DUT entry associated with it is set. If *b* was selected to hold a new BTH table, the valid bit (*V*) of the DUT entry associated with it is zeroed; in the next loop of HBRA, an entry of the BTH table it now holds will be properly filled (step 2 of HBRA – see Figure 3).

This completes the description of DBRC. In the next sections we describe how we tuned its parameters and compared it against a standard set associative L2 cache.

3. Methods

To evaluate DBRC, we employed execution driven simulation based on the CACTI ("An Enhanced Cache Access and Cycle Time Model" – CACTI [15]) and Simplescalar (www.simplescalar.com) tools, both widely used by the research community.

We used CACTI to configure different DBRC and equivalent standard L2 caches, and to estimate their access time, and energy consumption. We used Simplescalar to configure single core systems running SPEC2000 benchmark programs [14] while employing DBRC and standard L2 caches, and to estimate the average number of instructions executed per cycle (IPC).

3.1. Experimental Setup

In our experiments we have used a version of the Simplescalar that emulates the Alpha21264 superscalar processor [4]. We have used this version because it has been validated against a real Alpha21264 experimentally [3], and because there are precompiled SPEC2000 benchmark programs and properly set SPEC2000 workload (the MinneSPEC workload [7]) for it which are widely used by the research community. With the MinneSPEC workload, each SPEC2000 benchmark program executes about 2

billion instructions in total (about 1 second in current machines).

The Alpha21264 processors were set up as shown in Table 1, and their memory hierarchy for both DBRC and standard L2 was set up as shown in Table 2 (Table 2 also presents the standard L2 configuration).

Table 1. Alpha21264 setup

Pipeline	7 stages – 4-wide Fetch, Slot, and Map; 6 wide Issue, RegRead, Execute, and Write-back; and 11-wide Retire.
Functional Units	4 integer and 2 floating-point.
Issue Queues size	20-instrution integer and 15-instruction floating point.
Number of Renaming Registers	41 integer, 41 floating-point and 32 memory (load-store queues).
Branch Predictor	Tournament branch predictor with a three predictors combination: two level local predictor (1024 10-bit local history), path-based global predictor (12-bit history register which points to a table of 4K 2-bit saturating counters) and a choice predictor with a table of 4K 2-bit saturating counters.
Processor Clock	3GHz
Technolofy	65ηm

Table 2. Memory hierarchy and standard L2 setup

L1 Instruction Cache	64KB 2-way set associative (LRU), with 64B blocks and 1-cycle latency.
L1 Data Cache	64KB 2-way set associative (LRU), with 64B blocks and 3-cycle latency.
Standard L2 Cache	1MB 16-way set associative (LRU), with 64B blocks, 9-cycle latency and 32 MSHRs.
Memory Bus	16B-wide, 750MHz.
Main Memory	Unlimited, 225-cycle latency, 120-cycle precharge and 120-cycle pipeline access.

3.2. DBRC Configuration

The DBRC main parameters are: DBA size, DBA block size, B-TLB size, number of BTH levels, B-TLB target BTH level, DUT *R* field size, and the maximum number of attempts (MNA) to select a victim block.

In our experiments, we used DBRC and standard L2 caches with 1MB for data storage and blocks of 64B because these sizes are currently used in many Intel and AMD systems. So, we used DBA size equal to 1MB and DBA block size equal to 64B. Following the same reasoning, we used 128-entry fully associative B-TLB (this is the current TLB configuration on most current processors).

The number of levels of BTH affects several aspects of DBRC. Perhaps, the most important one is the size of BTH's level 0, L0T, since it is implemented in hardware instead of residing in DBA, as the other levels of BTH do. As L0T entries point to DBA blocks (see Figure 1), to calculate the size of L0T we start calculating the DBA number of blocks. For a 1MB DBA with 64B blocks, we have 16K blocks (1M/64). So, each entry of L0T must have an *I* field of 14 bits plus one valid (*V*) bit (15 bits total), and the same happens with the other levels of BTH, which, with 64B blocks, can accommodate only 32 entries each (by rounding 15 bits to the next power of 2, i.e., 16 bits, which gives us 2B per entry, or 32 entries for 64B). Assuming a 32 bits physical address (1GB main memory), the Table 3 shows how it (the physical address) can be divided to address each BTH level for BTHs with hierarchies of 3 (H3L), 4 (H4L) and 5 (H5L) levels (Figure 3 depicts the last line of Table 3), while Table 4 shows the number of entries of the tables of each level of these BTHs.

As Table 4 shows, a BTH with a hierarchy of 3 levels (H3L) would require a 64K-entry L0T, which would require approximately 128KB of storage space. On the other hand, a H5L BTH would require only a 64-entry L0T (approximately 128 bytes); accesses to a L0T of this size would be very fast, and larger physical addresses (i.e., larger main memories) could be easily accommodated (in the foreseeable future) by increasing it accordingly without a strong effect on L0T access time or chip area.

Table 3. Physical address fields

BTH	Address Field					
	f	e	d	c	b	a
H3L	-	-	16	5	5	6
H4L	-	11	5	5	5	6
H5L	6	5	5	5	5	6

Table 4. Number of entries on each BTH level

BTH	BTH Table				
	L0T	L1T	L2T	L3T	L4T
H3L	64K	32	32	-	-
H4L	2K	32	32	32	-
H5L	64	32	32	32	32

But BTHs with deep hierarchies may occupy too many DBA blocks, hurting DBRC hit rates. To examine this and to find appropriate values for the other DBRC parameters (B-TLB target BTH level, DUT *R* counter size, and maximum number of attempts (MNA) to select a victim block) we had to run experiments.

4. Experiments

To examine the impact of the BTH hierarchy depth on BTRC miss rate, we run the SPEC2000 benchmarks in the described experimental framework (Section 3.1). Figure 4 shows the results (the twof and vpr benchmarks were omitted because they have very small miss rates).

In Figure 4, the y-axis is the DBRC miss rate, while x-axis lists the SPEC2000 benchmarks and the arithmetic mean (A.M.) of their miss rates for 3 BTH

depths: 3 (H3L), 4 (H4L) and 5 (H5L). As Figure 4 shows, the number of levels of BTH does not affect the DBRC miss rate significantly; so, for now on, we consider DBRCs with BTHs of 5 levels only.

Ideally, B-TLB entries would point directly to the block addressed by the processor; however, a 1MB DBA of 64B blocks has 16K blocks. So, as the number of entries of B-TLB is small, hit rates may be low if it targets DBA blocks. Figure 5 shows B-TLB hit rates for different B-TLB targets.

In Figure 5, the y-axis is the B-TLB hit rate, while x-axis lists the SPEC2000 benchmarks and the arithmetic mean of their hit rates for the following B-TLB targets: L1T, L2T, L3T, L4T and data blocks in DBA. As Figure 5 shows, the performance of a 128-entry B-TLB targeting data blocks is to low. This is to be expected, since a data block is small and can be mapped anywhere in the physical memory address space (remember that, in virtual memory systems, TLBs target pages of typically 4KB or higher).

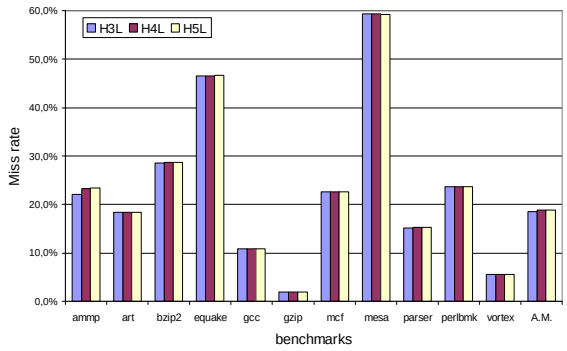

Figure 4. Impact of BTH depth on DBRC miss rates

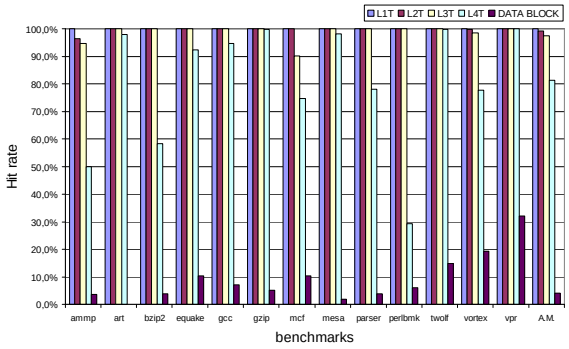

Figure 5. B-TLB hit rates for different B-TLB targets

From the results of Figure 5, we decided to use B-TLBs that target L3T. This means that, for each DBRC access, even with a hit in B-TLB, DBA must be accessed 3 times: one for accessing a L3T entry, one for accessing a L4T entry and one for accessing the data block. This increase DBRC hit time and energy consumption. We will address these concerns later on. Let's first examine the other DBRC parameters, DUT *R* counter size, and MNA.

The DUT entries *R* field holds a saturated counter that registers the frequency of reutilization of the associated block of DBA. Figure 6 shows the impact of DUT *R* counters size on DBRC miss rate, for 1, 5 and 10 bit sizes. As Figure 6 shows, on average, the larger the *R* field the better (lower) the average miss rate. So, we used *R* counters 10-bit sized.

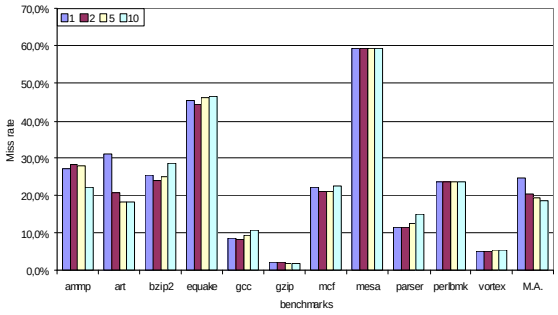

Figure 6. Impact of DUT *R* field size on DBRC

Ideally, the maximum number of attempts (MNA) to find a victim block in case of a DBRC miss would be 1 because it would be the fastest option. However, in this case, in case of a DBRC miss, we would remove valuable DBA blocks, such as high level tables of BTH or frequently used data blocks. To find a proper value of MNA, we run the experiments shown in Figure 7.

As Figure 7 shows, a MNA equal 5 or 10 already provide the protection that valuable blocks need; so, we used MNA equal 5.

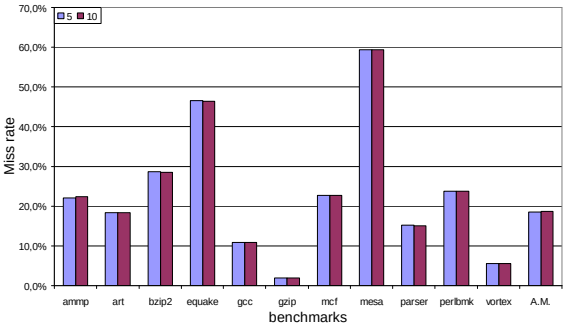

Figure 7. Impact of MNA on DBRC miss rate

After selecting the parameters of DBRC, we compared its performance with that of the standard L2 cache of Table 2 in terms of miss rate, impact on ILP and energy consumption. Figure 8 presents the comparison in terms of miss rate.

As shown in Figure 8, DBRC outperforms a standard 8-way set associative L2 of equivalent size in terms of miss rate for a large margin – an average miss rate reduction of 27.71%. This occurs thanks to DBRC

fully associativity and global block replacement algorithm.

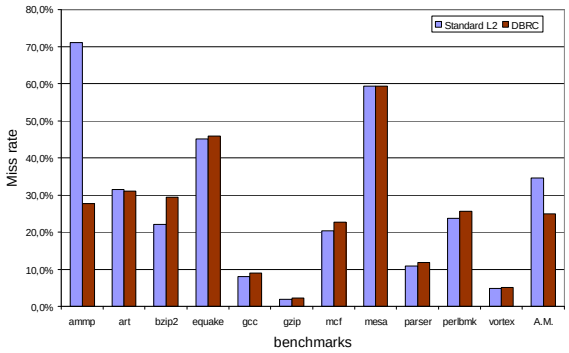

Figure 8. DBRC versus Standard L2: miss rate

Figure 9 presents the comparison DBRC versus standard L2 in terms of ILP, measured as instructions per cycle (IPC). In order to make this comparison, the hit and miss access times of DBRC had to be properly modeled.

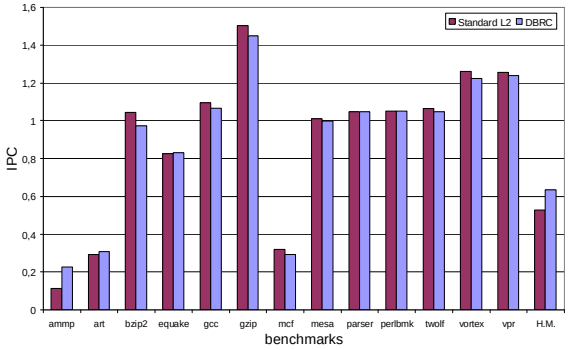

Figure 9. DBRC versus Standard L2: ILP

The hit and miss access times of DBRC are influenced by the access times of B-TLB, DBA, L0T, DUT and TT, and, depending on the type of hit (hits with hit in B-TLB, hits with miss in B-TLB) and miss (misses in different levels of BTH, which may require a single or several attempts to get DBA blocks) require different numbers of cycles (see algorithm of Figure 3). We have made a careful modeling of all aspects that contributes to DBRC hit and miss times. Unfortunately, due to space restrictions, we could not present it here; nevertheless, it is available in [10]. But an important aspect of this modeling have to be mentioned: we have partitioned DBA into 32 banks so that an access to a BTH table entry does not involve moving an entire 64B block of DBA to DBRC control. This saves power and reduces DBA access times, which is very important for the DBRC impact on ILP and energy consumption.

As Figure 9 shows, DBRC outperforms the standard L2 cache in terms of impact on ILP – an average (harmonic mean) IPC increase of 20.34%. This gain comes from the reduction of miss rates provided by the fully associativity and global block replacement algorithm of DBRC, and DBRC's DBA banking.

Finally, Figure 10 presents the comparison of DBRC with the standard L2 cache in terms of dynamic energy consumption. Again, in order to make this comparison, the energy consumption of DBRC and standard L2 had to be properly modeled. They depend on the many details of implementation of each, which we carefully modeled. Unfortunately, due to space restrictions, we could not present this modeling here; nevertheless, it is available in [10].

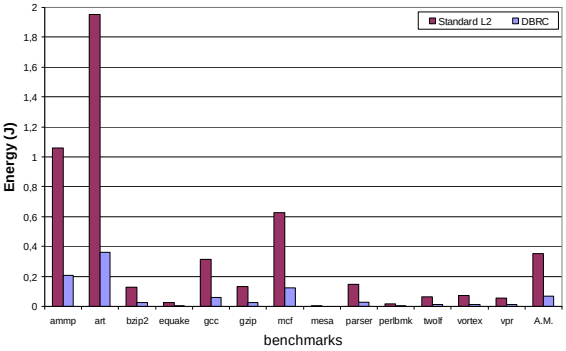

Figure 10. DBRC versus Standard L2: Energy

To obtain the results shown in Figure 10, we measured the total amount of Joules consumed by each cache architecture. For that, we added all tiny ηJ amounts of energy dynamically consumed by each peace of each cache architecture in accordance with the CACTI model. As Figure 10 shows, DBRC consumes much less energy dynamically than an equivalent L2 cache – 80.94% less. This, yes, is due in part to the DBRC lower miss rate and DBA banking organization, but also and mostly due to the energy cost of tag comparison in the large tag array of standard L2 caches, which DBRC does not have (TT is only accessed for block invalidation during DBRC misses).

5. Related Work

Puzak [11] proposed the inclusion of extra tags in a shadow directory to provide feedback to a local replacement engine in a set-associative cache. Batson and Vijaykumar [1] proposed the reactive-associative cache (r-a cache), which provides flexible associativity by placing most blocks in direct-mapped positions and reactively displacing only conflicting blocks to set-associative positions. Prime modulo hashing [6] and skewed associativity [13], on the other hand, attempt to distribute memory accesses uniformly across cache sets by targeting the indexing function. The "Non-uniform access with Replacement And Placement usIng

Distance associativity" cache, or NuRAPID [2], leverages sequential tag-data access to decouple data placement from tag placement. Qureshi et al. [12] proposed a technique to vary the associativity of a cache on a per-set basis in response to the demands of the program, while Zhang [16] proposed a cache design that allows the accesses to cache sets to be balanced by using a special block address decoder. All these approaches are variations of the standard cache design. DBRC departs from the standard design and tries to obtain performance mimicking the architecture of virtual memory systems.

6. Conclusions

In this paper, we proposed and evaluated a new architecture of Level 2 (L2) cache – the Dynamic Block Remapping Cache (DBRC). DBRC borrows some ideas from virtual memory systems to reduce the impact of L2 on system performance.

Analogous to virtual memory systems, which use a hierarchy of tables to map pages of virtual memory into pages of physical memory, the DBRC uses a hierarchy of tables to map blocks of L2 cache into blocks of physical memory. Also, as in virtual memory systems, a B-TLB is used to hold translations from main memory physical addresses to cache block indexes.

We compared the performance of DBRC with that of standard L2 caches using Simplescalar to model single core systems running SPEC2000 benchmarks. Our results showed that the DBRC achieves 27.71% reduction on average miss rate, 20.34% improvement in IPC, and 80.94% reduction on energy consumption when compared with an equivalent (in size) 8-way set associative L2 cache.

A direction for future work is to measure the static energy dissipation instead of using approximations based on the occupied area. Other direction for further research is to take into account the energy consumption and the B-TLB area.

As future work, we will examine the performance of DBRC in multi-core systems.

7. References

[1] B. Batson, T.N. Vijaykumar, "Reactive-associative caches", *Proceedings of IEEE International Conference on Parallel Architectures and Compilation Techniques*, 2001, pp. 49-60.

[2] Z. Chishti, M.D. Powell, and T.N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures", *Proceedings of the 36th Annual ACM/IEEE International Symposium on Microarchitecture*, 2003, pp. 55–66.

[3] R. Desikan, D. Burger, and S.W. Keckler, "Measuring Experimental Error in Microprocessor Simulation", *Proceedings of the 28th Annual International Symposium on Computer Architecture*, 2001, pp. 226-277.

[4] Digital Equipment Corporation, "Alpha Architecture Handbook", Digital Equipment Corporation, 1992.

[5] K.I. Farkas and N.P. Jouppi, "Complexity/Performance Tradeoffs with Non-Blocking Loads", *Proceedings of the 21st International Symposium on Computer Architecture*, 1994, PP. 211-222.

[6] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee, "Using prime numbers for cache indexing to eliminate conflict miss", *Proceedings of the 10th IEEE International Symposium on High Performance Computer Architecture*, 2004, pp. 288–299.

[7] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research", *Computer Architecture Letters, vol. 1*, 2002.

[8] D. Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization", *Proceedings of the 8th International Symposium on Computer Architecture*, 1981, pp. 195-201.

[9] D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Third Edition, Morgan Kaufmann Publishers, Inc., 2003.

[10] F.T. Pedroni, "A Dynamic Block Remapping Cache", M.Sc. thesis, Universidade Federal do Espírito Santo, Departamento de Informática, 2008 (in Portuguese).

[11] T.R. Puzak, "Analysis of cache replacement algorithms", Ph.D. thesis, University of Massachusetts, ECE Department, Amherst, MA., 1985.

[12] M.K. Qureshi, A. Jaleel, Y.N. Patt, S.C. Steely Jr., J. Emer, "Adaptive Insertion Policies for High Performance Caching", *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 381-391.

[13] A. Seznec, "A case for two-way skewed-associative caches", *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 169–178.

[14] Standard Performance Evaluation Corporation, "SPEC CPU2000 V1.2", http://www.spec.org/osg/cpu2000/, last access in March 2002.

[15] S.J.E. Wilton and N.P. Jouppi, "CACTI: an enhanced *cache* access and cycle time model", *IEEE Journal of Solid-State Circuits, vol. 31*, 1996, pp. 677–688.

[16] C. Zhang, "Balanced Cache: Reducing Conflict Misses of Direct-Mapped Caches through Programmable Decoders", *Proceedings of the 33rd International Symposium on Computer Architecture*, 2006, pp. 155-166.