Contents Management in First-Level Multibanked Data Caches[⋆]

E.F. Torres[1], P. Ibañez[1], V. Viñals[1], and J.M. Llabería[2]

[1] DIIS, Universidad de Zaragoza, Spain.
{enrique.torres, imarin, victor}@unizar.es
[2] DAC, Universidad Politécnica de Catalunya, Spain. llaberia@ac.upc.es

Abstract. High-performance processors will increasingly rely on multibanked first-level caches to meet frequency requirements. In this paper we introduce *replication degree* and *data distribution* as the main multibanking design axes. We sample this design space by selecting current data distribution policy proposals, measuring them on a detailed model of a deep pipelined processor and evaluating the trade-off introduced when the replication degree is taken into account. We find that the best design points use data address interleaving policies and several degrees of bank replication.

1 Introduction

Future superscalar processors will require a low-latency and high-bandwidth memory in order to attain high degrees of ILP. Among other things, a non-blocking, multiported, first-level cache able to feed data to ALUs in as few cycles as possible is needed[11]. This is specially true for integer codes without much data parallelism, where performance can be very sensitive to load-use delay.

To get a view of the involved trade-offs, Figure 1.a shows IPC (instructions committed per cycle) for an 8-issue out-of-order processor when varying three key parameters of a true-multiported first-level cache: size, latency (*lat*) and number of ports (*P*). Numbers come from executing SPEC2K integer codes in processors having three levels of on-chip cache (simulation details in Section 4). The uppermost line shows a 2-cycle latency, four-ported cache *2lat-4P*. The bottom line shows a 4-cycle latency, four-ported cache *4lat-4P*. Enclosed between the two lines there is a significant gap ranging from 1.98 to 2.37 IPC. As we can see size is an important parameter (roughly, a 2% IPC increase for each doubling), but it is not the most important one. The number of ports is increasingly important as we limit them: when moving from *2lat-4P* to *2lat-3P* and further to *2lat-2P* we see a 1.5% and 6% IPC decrease, respectively, no matter the size. On the other hand, increasing the cache latency by one cycle has a fixed 7% penalty, both from *2lat-4P* to *3lat-4P* and from *3lat-4P* to *4lat- 4P*.

⋆ This work is supported by the Ministry of Education of Spain under grant TIC 2001-0995-C02-02 and the Diputación General de Aragón (BOA 58,14/05/03). We gratefully thank Elena Castrillo Soto for her contributions in editing this paper.

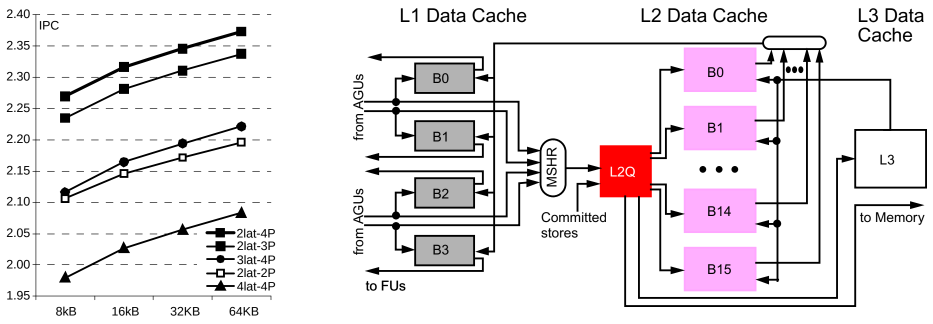

Fig. 1. a) Performance of a true multiported L1 cache. b) Data path of the memory data stream implemented in multibanking schemes. Load routing is made from IQ

Clearly, we will want to include the 64kB design with four ports and a 2-cycle latency as a first-level cache. Unfortunately such a cache cannot be built with such low latency if we plan to use current or future technologies targeted to deliver fast clocking [1].

Multibanking has lately been proposed as a feasible alternative to a monolithic multiported cache [3, 4, 7–9, 11–13]. Multibanking distributes data (cache lines or words) into several disjoint banks which are physically located close to address generation units or even close to functional units, minimizing then the wiring delay.

Multibanking adds two main additional freedom degrees to the design, namely number of ports per bank and bank content management. There are no works exploring big areas of this new and broad design space, since research up to now has focused on the proposal and evaluation of specific multibanking schemes.

Our contribution is twofold, on the one hand we describe what we believe are the two main design axes for a multibanked scheme: *replication degree* and *distribution policy*, placing current proposals within our taxonomy. On the other hand, we simulate a representative sample of design options in detail taking as a design unit, exclusively, a simple single-ported bank. We do not consider banks having multiported bit cells intentionally, because we seek a fair comparison among alternatives totalling the same raw storage capacity and keeping constant access time. Instead, we replicate data into as many banks as needed in order to achieve the desired multiport effect which in turn will reduce effective capacity (mirror caching [11]).

Simulations assume a high-frequency 8-issue superscalar processor with three levels of cache within the chip. We carefully model the collateral effects of deep pipelining, such as speculative instruction issue by making latency-prediction and using a mechanism to recover from latency misprediction. As we will see, simple static distribution policies coupled with partial replication of data may outperform some dynamic distribution schemes [3, 9].

Next section gives details about the processor and memory we model. Section 3 introduces a bank content management taxonomy, mapping existing proposals into it and exposing the main performance trade-offs. Section 4 shows the experimental framework and discusses the performance of the alternatives selected in order to cover a broad design space. Section 5 ends the paper with our conclusions.

2 Processor and memory

We model an 8-way processor with eight stages from Fetch to Issue Queue (IQ) and one stage between IQ and Execution. Other processor and memory parameters are listed in Figure 2. After issuing, instruction payload and operands are read and execution starts. Cache hit latency is assumed for a load instruction. Dependent instructions on the latency-predicted instructions can be issued speculatively while waiting for the resolution of all latency predictions. Because IQ is used as the Recovery stage, the predicted latency instruction itself and its dependent instructions remain in IQ until the last check arrives. We use a chained recovery mechanism which starts recovery as soon as a misprediction notification reaches IQ [12]. This mechanism is selective, meaning that only the instructions that depend on a mispredicted load will be re-executed.

fetch and decode width	8
branch predictor: hybrid (bimodal, gshare)	16 bits
reorder buffer entries	256
in-flight load/store	128
integer/FP IQ entries	64 / 32
integer/FP units	8 / 4

L1 I-cache	64KB, 4-way
L1 D-cache	2 cycles
banks	4
ports per bank	1 r/w
bank size	2-16 KB, 4-way
line size	32 B
L1 MHSR	16 entries

L2 Unified Cache 256 kB	16 banks, 8-way, 7 cycles
line size	128 B
L2 MHSR	8 entries
L3 Unified Cache	4MB, 16-way, 19 cycles
line size	128 B
main memory lat.	200 cycles

Fig. 2. Microarchitectural parameters

Figure 1.b shows the memory data path. The L1 cache is sliced into four logically independent banks. Each bank is tied only to one address computation unit (AGU) and placed as close to the functional units (FU) as possible. Each bank has only one read/write port shared among loads, committed stores and refills from the L2 cache. The banks are tied to the L2 cache through a single refill bus of 32 bytes.

The memory access takes one cycle to access the bank plus an extra cycle to reach the bypass network (line 1 in Figure 3). Load instructions that miss in L1 cache are reissued from IQ in time to catch the data when the refill is performed.

Requests to L2 cache are managed by the L2Q after accessing L2 tags (line 2 in Figure 3) in the same way Itanium II does [7]. The L2Q can send up to four independent and non-conflicting request to the sixteen address interleaved 2 cycles banks in each cycle. On an L2 cache miss a request is made to L3 cache

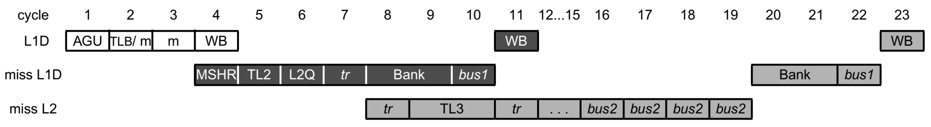

Fig. 3. Load pipeline . *tr* and *busX* mean transport cycles and bus use, respectively

and to memory (line 3 in Figure 3). A refill to L2 cache takes eight banks. The model can stand 16 primary L1 misses and 8 L2 misses.

We assume memory disambiguation based on the Alpha 21264 approach, where loads place addresses into a load buffer, and stores place address and data into a store buffer at once. We model a multiported store buffer (not shown in figure) with the same latency as the first-level cache and assume an oracle predictor for store-load independency.

Store instructions use a memory port and are routed to the store buffer when issued. L1 is write-through and write-not-allocate. Store instructions are committed to L2 and, when they hit L1 cache (filtered by the L2 cache directory), placed in one or more local buffers (depending on the replication degree). The 8-entry coalescing local write buffers update the L1 cache banks in unused cycles.

3 Design axes for bank content management

3.1 Data replication degree

The replication degree sets the number of copies a data can have spread across the cache banks. In order to achieve a given replication degree, the item required in a bank miss is refilled into a single bank (no replication at all), a bank subset (partial replication) or into all banks (full replication).

Full replication refills all banks with each missed line (Figure 4.a). Effective capacity is the smallest possible (the single bank size), but the cache access is conflict-free because any four simultaneous requests are satisfied at once without issue memory ports contention (no *bank conflicts).*

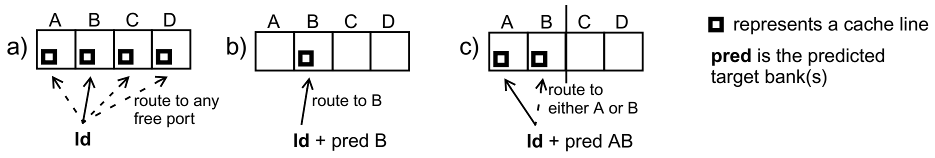

Fig. 4. Data replication options in a 4-bank (A,B,C,D) cache. a) Full Replication, b) No Replication line interleaving and c) Partial Replication line interleaving.

No replication allows only a single copy of the missed item (Figure 4.b). Here the effective capacity is the largest possible (the aggregated size of all banks) but cache access may frequently undergo issue port contention (*bank conflict*).

The middle ground between previous policies is *partial replication*: two or more copies of the missed item are placed in a bank subset (Figure 4.c). Partial replication trades off effective capacity against bank conflict reduction in an interesting way. A second-order effect of data replication is the accuracy of the bank predictor (if any) tied to the data distribution policy: the higher the replication the better the predictability, simply due to the fact that predicting one among n banks is usually harder than predicting one among $n/2$.

3.2 Data distribution policy

A distribution policy determines in which bank subset (one or more banks) a data has to be placed, trying to minimize bank conflicts and cache misses. A distribution policy has a mechanism that predicts or suggests the destination bank of every load. Static policies always place the same line in the same bank subset. Dynamic Policies allow lines to move or to replicate among subsets. Next, we briefly explain each policy.

Static policies

Distribution by data type. It consists in distributing different data structures to different bank subsets. In the first-level cache context this idea has only been applied by Cho et al. to separate the stack region from the remaining ones [4]. Both this work and our simulation use a 4 KB Access Region predictor to guess the right region.

Distribution by data address. This is the most conventional approach and consists in distributing data according to a hash function applied to some bits of their addresses [8, 11–13]. The data can either be a word (word interleaving) or a line (line interleaving). This policy requires a bank or bank subset predictor because the load routing is made from IQ, prior to computing addresses. Bank mispredictions are corrected by re-routing loads to the correct place from IQ. As a bank predictor we use an *enhanced skewed binary predictor*, originally proposed by Michaud et al. for branch prediction [6, 13]. We choose to predict each address bit separately: up to 2 bits for 4 banks. Every bit predictor is identically sized: 8K entries for the three required tables and a history length of 13, totalling 9KB per predictor. Since we are interested in recognizing the hard-to-predict loads, we further add a 2-bit confidence saturating counter to every entry in all tables. This allows us to use spare bandwidth by broadcasting those loads having a low-confidence prediction. Load broadcast can increase performance by lowering the number of recoveries [12].

Dynamic Policies They allow lines to move or replicate dynamically among bank subsets to avoid bank conflicts or maintain effective capacity. Two policies have been suggested up to now, both using load identities (program counter) to drive load routing and data placement.

Distributing the working set of individual loads (Instruction Working Set). This policy by Racunas and Patt tries to take the whole data working set referenced by each load and place it in a single bank subset [9]. Lines referenced from several loads can migrate among subsets according to the relative confidence of predictions. Such predictions come from two tables (called iPAT and dPAT, respectively) requiring a total of 16KB for four banks. Bank mispredictions are corrected by accessing L2, and so they have the same cost that a first-level cache miss. The authors evaluate a first level made up of 8 two-ported banks in 16-way processor. We scale this proposal to two bank subsets with two replicated banks each and also evaluate a no replication version, which performs poorly due to frequent bank conflicts caused by lack of ports.

Conflict-aware distribution. This policy by Limaye et al. replicates a line as soon as a bank conflict appears [3]. A table remembers the last bank (a *cachelet*) assigned to each load and will repeatedly route it to such a bank unless a bank conflict appears, in which case a free bank will be referenced and refilled from level two. This proposal may allocate a line to one, several or all the banks. Our four single-ported banks are similar to the configuration they evaluated, requiring an 8 KB table to remember the last bank.

3.3 Studied design points

The policies we have described above have been evaluated by their authors using very different processor and memory hierarchy models. So, in Section 4 we evaluate them within the framework described in Section 2. We will use IPC as the main metric, correlating it with cache misses and bank conflicts. Data replication introduces a new trade-off into the data distribution policies because data replication reduces bank conflicts but at the same time the effective capacity reduction increases cache misses.

As design points with no replication we choose the following: distribution by data address (*Word* interleaving and *Line* interleaving) and distribution by instruction working set (*IWS*). Partial replication is evaluated by arranging the four banks into two subsets with two replicated banks. These design points are word interleaving (*Word-2R*), line interleaving (*Line-2R*), instruction working set (*IWS-2R*) and distribution by access region (*AR-2R*). Conflict-aware distribution (*CA*) is another design point with partial replication. Finally, we also evaluate a full replication organization (*4R*).

4 Results

We use SimpleScalar3.0c [2] to carefully model the processor and memory stated in Section 2. As workload we use all SPECint 2K but *mcf* because it has a very low IPC (0.23) no matter the L1 cache we consider. We use Alpha binaries simulating a contiguous run of 100M instructions from the SimPoints [10] after a warming-up of 200M instructions. All figures show the IPC harmonic mean.

Figure 5 shows the IPC achieved by the different distribution policies and replication degrees with banks ranging from 2KB to 16KB. Figure 5.a and Figure 5.b show policies with no replication and partial replication, respectively, while Figure 5.c compares the best ones to full replication, assuming a 2-cycle latency for L1.

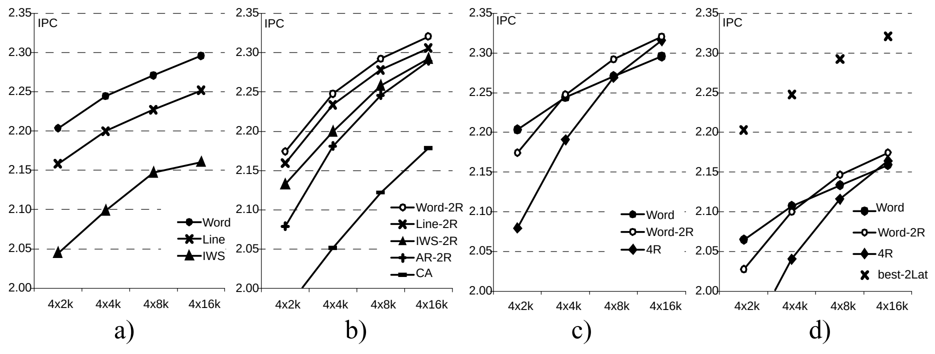

Fig. 5. IPC achieved with banks ranging from 2KB to 16KB with 2-cycle latency. a) no replication, b) partial replication, c) the best of the previous ones plus full replication, and d) merging the best points for latency 2 (x) with the best 3-cycle latency points.

In order to better understanding the IPC results we also show L1 cache miss ratio (Figure 6.a, grey), and bank misprediction ratio (Figure 6.a, black) on 8KB banks. The total length of each bar represents the load percentage undergoing latency misprediction. Besides, Figure 6.b exposes the average number of cycles a load, once ready, waits on IQ due to issue memory ports contention (*conflicts*)

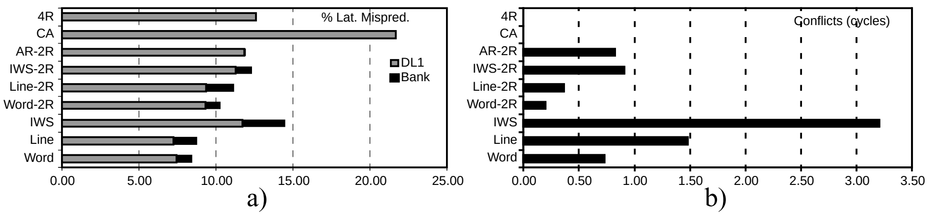

Fig. 6. a) Percentage of latency mispredictions, b) average number of lost cycles due to issue memory ports contention (*bank conflicts*).

Starting from Figure 5.a, notice that Instruction Working Set (*IWS*) behaves clearly worse than the interleaving policies (*Word* and *Line*) due to its higher number of bank conflicts (Figure 6.b) and of L1 cache misses (Figure 6.a). *Word* performs always better than *Line*. *Word* undergoes less bank conflicts, has a lower L1 cache miss rate and exhibits better bank predictability than *Line*.

In general, partial replication policies (Figure 5.b) reduce the number of bank conflicts but at the expense of increasing the L1 cache miss ratio. Conflict-Aware (*CA*) has an IPC lower than the other policies. *CA* has a very high L1 cache miss ratio (Figure 6.a) on account of the changes made in bank mapping to eliminate bank conflicts.

Access Region (*AR-2R*) and *IWS-2R* have a worse IPC than *Word-2R* and *Line-2R*. They experience a worse distribution across the bank subsets generating more L1 cache misses and bank conflicts (Figure 6). As in Figure 5.a, *Word-2R* is better than *Line-2R*.

Finally we analyze Figure 5.c, where we compare Full Replicated (*4R*) against *Word* and *Word-2R*, the best performing design points up to now. *Word-2R* provokes less bank conflicts and bank mispredictions than *Word*, but has more L1 cache misses because it is half the effective size than *Word* (Figure 6). As capacity is the main factor of performance losses (Figure 5), with 4 banks of only 2KB, *Word* is the best option. However, with banks greater than 2KB the best design point is *Word-2R*.

Increasing the bank size makes *4R* appealing. With 16KB the performance of *4R* is more or less the same as the best design point, but at much lower cost and complexity.

Summing up, the design point giving the best performance is *Word*. The optimal replication degree is subject to the bank size. For 2KB banks it is better not to replicate to keep effective capacity, whereas on 4KB and 8KB banks, partial replication removes enough bank conflicts to offset the effective capacity loss. Finally, full replication achieves the highest performance for 16KB banks with no predictor at all.

4.1 Effect of rising L2 latency

Doing over the same experiments with a 10 cycle latency to reach L2 shows up a lower IPC across all design points. The L1 cache miss penalty increase is more deeply suffered by those configurations with higher L1 miss rates. For example, with 2KB the IPC for *4R* decreases by 4.3% while *Word* does by 2.3%. The crossing between *Word-2R* and *Word* now goes to the right and *Word* becomes the best option for banks up to 4KB.

4.2 Effect of rising L1 Latency

Up to this point we have assumed, regardless of the bank size, a 2-cycle L1 cache access latency. Bank latency, in cycles, depends on the particular technology used and on the design of the rest of the processor (cycle time, routing,...). It is out of the scope of this work to determine this latency. At the same time it is evident that banks of different sizes will have unequal latencies. This is why we present results over a range of values.

Figure 5.d shows performance for a 3-cycle L1. Moreover, for each bank size the best design points achieved with a 2-cycle L1 (extracted from Figure 5.c) are market with a cross. As we can see, even the smallest 2-cycle banks surpass any 3-cycle design points. Moreover a 2-cycle *Word* with 4 banks of 2KB outperforms most 4-port true-multiported schemes of Figure 1 with latencies of 3 or 4 cycles. Increasing the bank size is never a good option if it involves latency add-on.

5 Conclusions

There are two major decisions regarding contents management when facing the design of a first-level multibanked cache: *distribution policy* and *replication degree*. Choosing a given design point sets how many copies of a line are allowed and the identity of the banks able to hold them. In general, distribution policies that spread data flow by data address lead to the smallest number of cache misses and bank conflicts. Our results show that the most suitable distribution policy is *Word* interleaving.

The optimum replication degree is subject to the bank size. For a 2KB bank the optimal is no replication. Partial replication over two banks is the most effective method for sizes between 4KB and 8KB. On greater banks, Full replication performance is equivalent to the best bank content management at a lower cost. On the explored design space (banks between 2KB and 16KB with latencies of 2 and 3 cycles), increasing bank size is never a good option if it involves latency add-on.

References

1. V. Agarwal et al.: Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. Proc. of 27th ISCA (2000) 248–259
2. D.C. Burger and T.M. Austin: The SimpleScalar Tool Set, Version 2.0. UW Madison Computer Science Technical Report #1342 (1997)
3. D. Limaye, R. Rakvic, and J.P. Shen: Parallel Cachelets. Proc. 19th ICCD 284–292. Sept. 2001.
4. S. Cho, P. Yew, and G. Lee: A High-Bandwidth Memory Pipeline for Wide Issue Processors. IEEE Trans. on Computers, vol. 50, no. 7 (2001) 709–723
5. R.E. Kessler, E.J. MacLellan, and D.A. Webb: The Alpha 21264 Microprocessor Architecture. Proc. of ICCD98 90–95. Oct. 1998.
6. P. Michaud, A. Seznec, and R. Uhlig: Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. Proc. of 24th ISCA (1997) 292–303
7. S. Naffziger et al.: The implementation of the Itanium 2 Microprocessor. IEEE J. Solid State Circuits, vol. 37, no. 11 (2002) 1448–1460
8. H. Neefs, H. Vandierendonck, and K. De Bosschere: A Technique for High Bandwidth and Deterministic Low Latency Load/Store Accesses to Multiple Cache Banks. Proc. of 6th HPCA (2000) 313–324
9. C. Racunas and Y.N. Patt: Partitioned First-Level Cache Design for Clustered Microarchitectures. Proc. of 17th ICS 22–31. June 2003.
10. T. Sherwood, E. Perelman, G. Hamerly, and B. Calder: Automatically Characterizing Large Scale Program Behaviour. Proc. of ASPLOS (2002)
11. G.S. Sohi and M. Franklin: High-Bandwidth Memory Systems for Superscalar Processors. Proc. 4th ASPLOS (1991) 53–62
12. E. Torres, P.E. Ibaez, V. Vials, and J.M. Llabera: Counteracting Bank Mispredictions in Sliced First-Level Caches. 9th EuroPar, LNCS 2790 586–596, Sept. 2003.
13. A. Yoaz, E. Mattan, R. Ronen, and S. Jourdan.: Speculation Techniques for Improving Load Related Instruction Scheduling. Proc. of 26th ISCA (1999) 42–53