

Page-Level Behavior of Cache Contention

Siddhartha Tambat
Dept. of Computer Science and Automation
Indian Institute of Science
svtambat@csa.iisc.ernet.in

Sriram Vajapeyam
Independent Consultant
Surathkal, India
sriram_vajapeyam@yahoo.com

Abstract—Cache misses in small, limited-associativity primary caches very often replace live cache blocks, given the dominance of capacity and conflict misses. Towards motivating novel cache organizations, we study the comparative characteristics of the virtual memory address pairs involved in typical primary-cache contention (block replacements) for the SPEC2000 integer benchmarks. We focus on the cache tag bits, and results show that (i) often just a few tag bits differ between contending addresses, and (ii) accesses to certain segments or page groups of the virtual address space (i.e. certain tag-bit groups) contend frequently. Cache-conscious virtual address space allocation can further reduce the number of conflicting tag bits. We mention two directions for exploiting such page-level contention patterns to improve cache cost and performance.

Index Terms—Data Cache, Memory Access Characterization, Cache Tags, Cache Contention

I. INTRODUCTION

THE importance of effective, efficient uni-processor caches is increasing given not only the processor-memory speed disparity but also newer constraints such as power, wire-lengths, etc. Primary (L1) caches today are typically small in size (8KB to 64KB), of limited associativity (1-way, 2-way, or 4-way), and employ multi-word cache blocks¹ of 32 to 64 bytes (8-16 words). The resulting dominant capacity and conflict misses in L1 caches imply that a major fraction of the L1 cache misses replace *live* cache blocks, i.e., blocks that will be accessed again.

While capacity misses are a function of cache size, three fundamental factors can be manipulated to reduce conflict misses: cache placement (cache indexing

and/or data layout in memory), associativity, and (address) block size. Several cache organizations that emulate higher associativity to alleviate conflicts have been proposed, ranging from Column-Associative Cache [1] to Victim Caches [4]. An OS-based approach [2] dynamically re-maps frequently conflicting *physical* pages to pages whose cache indices are mutually less conflicting. Non-traditional cache indexing schemes have been explored, for example [6].

We approach the issue of live-block replacement from a different angle. We study the characteristics of conflicting virtual-address pairs (Sections II & III), towards triggering novel cache features and cache-conscious data layouts. The value of our study is in focusing on the higher order virtual address bits (the cache tag bits), as opposed to the lower order (cache index) address bits. The study reveals that often just a few tag bits, and a relatively small number of page groups and segments of the virtual-address space, are involved in cache contention. We surmise that this behavior stems from simultaneous access to different data structures.

The observed patterns in contending virtual addresses can be exploited to improve caches in at least two ways. First, better cache indexing could potentially be achieved by using the more frequently differing tag bits as part of the cache index. Second, the tag overheads of set-associative and decoupled sector caches can be reduced by sharing a common main tag, consisting of the typically non-conflicting tag bits, across all the blocks of a set or sector. Per-block individual sub-tags then consist of only the remaining frequently conflicting tag bits. The individual sub-tag and the shared main tag together make up the tag for a cache block. This organization reduces tag overheads, an important issue in scenarios such as low power, small cache blocks, 64-bit addresses, short wire-lengths, etc.

In this paper, we identify patterns in contending virtual addresses of the SPECInt2000 benchmarks for

¹ Fixed block sizes increase conflict misses for programs with smaller optimal blocks. The ideal block size has been shown to vary widely across programs [3].

Manuscript submitted: 4 June 2002. Manuscript accepted: 19 June 2002. Final manuscript received: 1 July 2002.

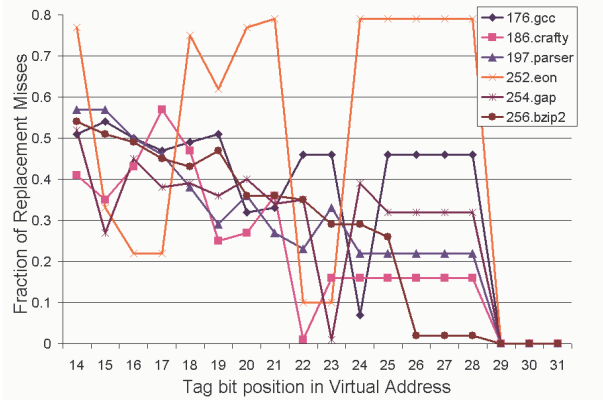
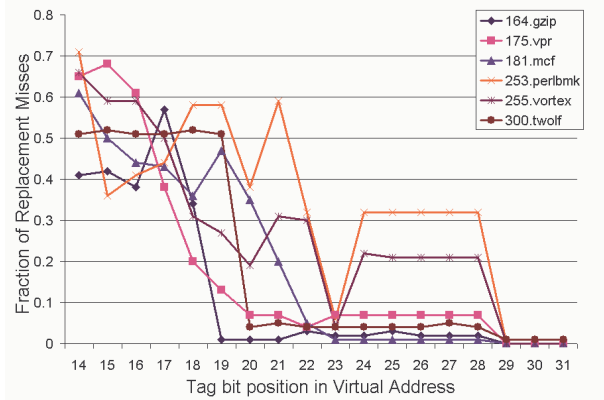


Fig. 1. Participation of tag bits in cache replacement misses: SPECint2000 benchmarks, 16KB direct-mapped cache.

L1 caches (Sections II and III). We summarize our observations and mention *caveats* in Section IV. Caches that exploit the observed address patterns are beyond the scope of this short paper.

II. STUDY FRAMEWORK

We study the behavior of the twelve SPEC2000 integer benchmarks compiled on an IBM RS/6000 AIX 4.3 workstation using the native C and C++ compilers with the standard SPEC -O3 optimization flags. Virtual memory addresses are 32 bits wide on this platform. We used the training inputs for all benchmarks, except *gap* for which we used the test input. All traces are for user-level activity only, including any activity in the statically linked libraries. We studied each benchmark for memory references generated by 200M instructions after discarding the appropriate number of instructions in the start-up phase of the program. We observed that such a 200M instruction simulation window, chosen from later in the program's execution, reliably gives representative results for the SPEC2000 integer. (We also studied traces corresponding to 2 billion instructions for each benchmark, and found cache miss ratio and memory traffic for the two trace inputs to be within 1% of each other for each benchmark.)

We consider virtual address L1 (primary) data caches of sizes 8KB, 16KB, 32KB, and 64KB, but report results only for select cases due to space constraints. The block size is 32 bytes for 8KB and 16KB caches, and 64 bytes for 32KB and 64KB caches. We choose the cache and block sizes based on the cache configurations of current microprocessors. Though several current commercial caches are real address caches, we consider only virtual addresses in order to study the inherent behavior of individual programs and

to exclude the effects of physical memory allocation policies and multiprogramming.

III. PAGE-LEVEL CONTENTION BEHAVIOR

A. Tag-bit Conflicts

We first study the *participation* of individual tag bits in cache block replacements: how many and which tag bits differ between the address of an in-coming cache block and its victim. This is aimed at getting an idea of how contending accesses walk the virtual address space – whether there are any exploitable patterns in contending addresses. A contending access causes a *replacement miss*, defined to be a cache miss that replaces a live cache block. This definition covers all cache misses except those to cache lines that are either invalid or hold valid but dead blocks. Using the notion of a replacement miss enables us to target L1 cache behavior more broadly than possible by considering only conflict misses. In our study, we let our focus on highly frequent contending accesses automatically sideline the less frequent non-replacement L1 misses, instead of explicitly identifying and discarding the latter.

Figure 1 plots the fractional participation in contention of individual tag bits for the SPEC2000 integer benchmarks, for a 16KB direct-mapped data cache with 32-byte blocks (512 cache blocks). Since we have 32-bit virtual addresses, bits 0-13 form the cache index and offset. Thus the x-axis shows only address bits 14 through 31 (the 18 cache tag bits). Reading the figure, bit 14 in *perlbnk* participates in more than 70% of the replacement misses. Figure 2 shows the cumulative fraction of replacement misses in which exactly 1, 2, 3, etc. tag bits differ. From the figure, 60% of replacement misses in *gzip* have only the two low-

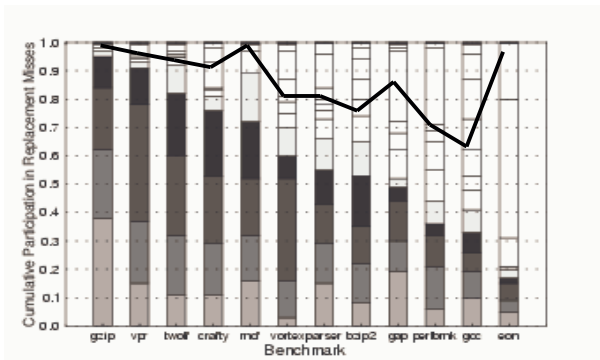


Fig. 2. Cumulative participation of tag bits in replacement misses, ordered from LSB (bottom bar) to MSB (top bar). The bold line shows the cumulative participation of the 6 most-conflicting tag bits (i.e. irrespective of bit position) for each benchmark.

order tag bits 14 and 15 (one or both of them) participating.

The first observation from the figures is that the different tag bits do not participate equally in replacements – frequently a large fraction of the replacements have only a few tag bits differing between the contending addresses. For example, in *vpr*, bits 14 through-17 are the primary participants in replacements. In *gzip*, only the low-order 5 tag bits (or a subset) differ between the contending addresses in almost all replacements (Figure 2). Overall, for five benchmarks, 80% or more of the replacements differ in just the 5 low-order tag bits (or a subset). Considering the tag bits in order of contention participation rather than bit position (the bold line in Figure 2), just 6 tag bits account for more than 90% of the replacements for six of the twelve benchmarks, and for more than 80% replacements in another three benchmarks.

The second observation is the dichotomy in conflict behavior of the tag’s MSBs and LSBs. The frequently conflicting bits are typically the low-order tag bits, the exception being the uniformly participating high-order bits 22-through-28 (for example, bits 24-28 participate the most for *eon*). This dichotomy is not just a result of capacity misses – the observation holds (results not shown here) even when the L1 cache size increases beyond 16KB, which is a knee in the capacity miss ratio curve for SPECint2000.

B. Tag-bit Group Conflicts

An interesting observation is that often a group of tag bits participate uniformly in replacements (Figure 1). This is especially striking for the higher order bits 22-through-28 across most of the benchmarks. In addition, tag bits 22-23 and 20-21 for *gcc*, 16-17 and 22-23 for *eon*, 18-19(-21) for *perlbnk*, and other less-

striking *tag-bit groups* have very similar contributions to tag conflicts.

The reason for the uniform contribution of the higher order bits 22-through-28 is that stack and heap addresses on our platform differ exactly in these bits, due to the particular stack and heap region start addresses used by the IBM *xlc* compiler. Stack and heap accesses obviously contend frequently in these benchmarks. If one wishes to reduce to a single bit the number of MSB tag bits that participate in such contentions, a cache-conscious compiler could easily pick appropriate base addresses for the stack and heap. For example, relocating the stack’s base address from 0x2ff2ffff to 0x2000ffff on our platform will result in tag bit 28 alone (among MSBs) differing in stack-heap contention.

Extrapolating the above observations to the other tag-bit groups that participate uniformly in replacements, we infer that specific groups of virtual pages, each group perhaps holding a particular data structure, are accessed simultaneously and conflict in the data cache. For example, in the benchmark *parser*, two virtual *page groups* that have address bits 14 and 15 set to “00” and “11” respectively and have identical MSBs otherwise (e.g. page groups 00..000XX..X and 00..011XX..X) are simultaneously accessed and conflict in the cache. This results in the *tag-bit group* 14-15 contributing uniformly to replacements. Just as the stack can be relocated, a compiler with fore knowledge of such conflicts could relocate these page groups (i.e. the data structures contained in them) so as to reduce the number of conflicting tag bits (from 2 to 1 in this particular example). Such fore knowledge could be obtained via program profiling, for example.

Apart from the strikingly obvious contending page-groups discussed above, it is possible that frequent contention between particular page groups lay hidden behind other non-uniformly participating tag bits. To unearth such hidden patterns, we identified for each benchmark the contribution of all possible tag-bit groups to replacements (Table 1 shows the top-10 groups per benchmark) for 16KB (32B block size) and 64KB (64B block size) direct-mapped primary caches. These two cache sizes capture two different behavioral points: a realistic L1 that includes both capacity and conflict misses, and an upper-bound L1 that emphasizes conflict misses.

Table I shows that once capacity misses are significantly reduced (i.e. moving from 16KB to 64KB cache size), typically only a few unique tag-bit groups

TABLE I
CUMULATIVE CONFLICTS OF TOP-10 CONFLICTING
TAG-BIT GROUPS

Benchmark	16KB Cache	64KB Cache
Eon	83.68 %	99.47 %
Gzip	58.63 %	98.20 %
Perlbnk	59.69 %	94.29 %
Vpr	60.84 %	93.77 %
Gap	77.01 %	91.87 %
Crafty	49.24 %	78.82 %
Twolf	23.18 %	74.61 %
Parser	39.23 %	56.61 %
Mcf	31.93 %	55.79 %
Gcc	30.40 %	54.20 %
Bzip2	22.63 %	45.70 %
Vortex	52.09 %	33.89 %

determine a majority of the replacements. For the 64KB cache, the top-10 conflicting tag-bit groups participate in more than 90% of the replacements for five of the benchmarks, more than 70% for two others, and more than 50% for three other benchmarks. This is in marked contrast to the 16KB case where capacity misses are significant and the top-10's participation is significantly less. (The 32KB cache behaves more like the 64KB cache.) Examples of top conflicting tag-bit groups include just bit 17 for *gzip* and the group 18,21-28 for *eon*. It is not possible to unearth these dominant tag-bit groups from just figures 1 and 2.

Tag bits from such predominant tag-bit groups could be used in the cache index in lieu of traditional index bits, to improve performance. For example, for a 64KB cache, address bit 17 of *gzip* seems a much more prominent cache index candidate than the traditional index bits 14 and 15. Further, hardware schemes that dynamically select *some* of the index bits on a per-program basis, without slowing down access latency, seem possible.

IV. SUMMARY AND FUTURE DIRECTIONS

We have reported a characterization of contending virtual address pairs for small primary caches and the SPECInt2000 benchmarks. Typically just a few of the high-order bits (tag bits) differ between contending addresses – especially for larger L1 caches. Further, certain virtual page groups (tag-bit groups) tend to conflict frequently. A contention-aware compiler (e.g. one that is profile-guided) could suitably allocate

frequently conflicting virtual page groups so as to further reduce the number of conflicting tag bits. A *caveat* for our observations is that programs that exercise the virtual address space differently may not exhibit these patterns, especially dynamically linked applications; this study needs to be extended to a wider spectrum of applications.

The observed tag-bit conflict patterns could be exploited to design novel cache organizations. Directions we are pursuing include non-traditional, possibly dynamic, choice of cache index bits, and sharing portions of the tag across a set (of an associative cache) or a sector (of a decoupled-sector cache). While we have focused on virtual addresses so as to understand inherent program behavior, the tag-bit conflict patterns reported could be useful for physical-address caches as well, e.g. in guiding cache-conscious page allocation. Overall, we hope this work triggers explorations of novel caches.

ACKNOWLEDGMENT

S. Muthulaxmi [7] carried out an early study of the ideas mentioned in this paper for her Masters' thesis in 1996, under the guidance of the second author. We thank Jim Goodman, Matthew Thazhuthaveetil, Tulika Mitra, and the anonymous reviewers for their comments and feedback. An equipment grant from Intel Microprocessor Research Labs enabled the simulations reported in this paper.

REFERENCES

- [1] A. Agarwal and S. D. Pudar, "Column-associative caches: a technique for reducing the miss rate of direct-mapped caches," In *Proceedings of the 20th Annual Intl. Symposium on Computer Architecture*, May 1993.
- [2] B. N. Bershad, *et al.*, "Avoiding conflict misses dynamically in large direct-mapped caches," In *Proceedings of the 6th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, October 1994.
- [3] D. Burger, J. R. Goodman, and A. Kagi, "Memory bandwidth limitations of future microprocessors," In *Proceedings of the 23rd Annual Intl. Symposium on Computer Architecture*, May 1996.
- [4] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers," In *Proceedings of the 17th Annual Intl. Symposium on Computer Architecture*, May 1990.
- [5] A. Seznec, "Decoupled sectored caches: conciliating low tag implementation cost and low miss ratio," In *Proceedings of the 21st Annual Intl. Symposium on Computer Architecture*, April 1994.
- [6] Hans Vandierendonck and Koen De Bosschere, "Evaluation of the performance of polynomial set index functions," In *Proceedings of the First Workshop on Duplicating, Deconstructing and Debunking*, held in conjunction with ISCA 2002, May 2002.
- [7] S. Muthulaxmi, "A study of variable block size caches," M.S. Thesis, Supercomputer Education and Research Centre, Indian Institute of Science, October 1997.