The Effect of Page Allocation on Caches

William L. Lynch, Brian K. Bray, and M. J. Flynn

Computer Systems Laboratory
Stanford University

ABSTRACT

Medium to large physically-indexed low-associativity caches, where physical page number bits index the cache, present two problems. First, cache miss rate varies between runs, as data location in the cache depends on the placement of virtual pages in physical memory. Secondly, the virtual-to-physical address translation must precede cache indexing, increasing latency. This paper summarizes simulation results of instruction, data, and unified caches with conventional page allocation, and explores improving the mean miss rate by controlling (coloring) page allocation. A more strict page coloring algorithm reduces latency by allowing cache indexing to precede address translation.

Keywords: cache, virtual memory, page allocation, page coloring

1 Introduction

Modern computer systems use virtual memory [1] to provide large protected address spaces to processes, and use cache memory [2] to provide faster effective access times to physical memory. As technology progresses and appropriate organizations and sizes of caches change, interactions between the virtual memory and cache memory systems become important. Unless the physical memory access only requires non-translated bits the virtual memory system must translate virtual to physical addresses before physical memory can be accessed.

Caches comprise sets; a data item can reside anywhere in one set selected by the data's address. Lower cache associativity (i.e., smaller sets) can reduce latency because of smaller tag-check-and-mux of each set, while larger cache size can reduce the miss rate; both are desirable characteristics [3]. The cache size per degree of associativity determines the required index size. Low associativity caches larger than the page size can thus require page number bits for cache indexing. For example, the MIPS R4000 requires one page-number bit to index its 8 KB primary caches [4], and the IBM RS/6000 Model 520 requires two page-number bits to index its 64 KB four-way-set-associative primary data cache [5].

Frequently, caches are associative enough that the index is available just from the page offset (e.g. SuperSPARCs 16 KB 4-way-associative data cache and 20 KB 5-way-associative instruction cache [6]). These designs allow physical cache indexing to proceed as soon as the virtual page offset is available, as the virtual and physical page offsets are identical. However, when low associativity and large size require page number bits (the color bits) for cache indexing, designers face a difficult choice. Either the virtual color must be used to index the cache, or indexing must wait for the completion of the virtual-to-physical address translation. Virtually-indexed caches have aliasing problems which increase operating systems complexity [7]. But waiting for the translation increases the latency (as in the MIPS R2/3000 where the translation is afforded its own pipeline stage [8]). In addition, the cache location depends on the physical color, which is determined by the page allocation of the operating system. Conventional page allocation may locate different pages into different colors on different runs, resulting in different collisions and different cache miss rates for otherwise apparently identical runs.

We call conventional page allocation, where the page placement decision is based solely on non-cache factors, uncolored page allocation. Sites and Agarwal [9] note the existence of a difference in miss rate between virtually and physically indexed caches because of uncolored allocation.

We simulated colored page allocation, where the operating system allocates pages conscious of cache performance. Kessler and Hill [10, 11] simulated uncolored and colored allocation, but only for very large (1 to 10 MB) unified second-level caches. They found their colorings typically reduced second-level miss rates in the 1% to 30% range, with a 55% reduction in one instance. Our first-level cache simulations showed reductions of...
20% to 25% for a wide range of data caches sizes, and 50% to 90% for many instruction cache sizes.

There are two kinds of colorings: static and dynamic. Static coloring uses the virtual address to determine an appropriate allocation color. Static coloring relies on spatial locality, where nearby pages are likely to be referenced concurrently, and thus should not be allocated to the same color. Dynamic coloring uses the recent allocation history to determine an appropriate color for the next allocated page. Dynamic coloring relies on temporal locality, where pages faulting close to each other are likely to be referenced concurrently. This paper presents simulation results from identity static coloring, where the virtual and physical colors are identical, and round robin dynamic coloring, where colors are allocated strictly in rotation.

Colorings further subdivide into perfect and imperfect colorings. Perfect colorings require that the allocation paradigm hold without fail. Perfect coloring may require excess pages of specific colors; thus page faults may increase under perfect coloring due to contention on a single color. This effect on page fault rate was observed to be small for many main memory sizes [12]. Even so, imperfect coloring could mitigate these paging effects: if a free page of the desired color is not available, another color can be selected. MIPS uses imperfect coloring [13] for this reason. Kessler and Hill simulated imperfect coloring, allocating only to pages which happen to be available; we simulated perfect coloring.

There are advantages to perfect static coloring beyond reduction in miss rates. The function relating the virtual and physical color could be implemented in hardware and a physically-indexed cache could be indexed with the (transformed) virtual color, without the need for full address translation (Figure 1). (The MIPS R6000 TLB-slice [13] approaches this functionality with imperfect coloring.) The low latency of virtually-indexed caches is available in a physically-indexed cache; the cache is both virtually and physically indexed.

2 Methodology

Trace-driven simulations of gas, gcc1, fft, lloops, spice3, and tex generated the results in this paper. The Ultrix 3.1 C compiler with the -O2 option compiled traces for the MIPS R2000 architecture. The unprocessing traces included references from application code, string routines, and printf routine references, but not system calls, scanf routines, or math libraries.

The cache performance simulations assume an infinite memory system; specifically, once the system allocated a virtual page to a physical color it was never deallocated or reallocated. Thus, there were no cache flushes, and pages never changed colors. The initial order of the free list was (color) random, as it could be after a length of time of page allocation and freeing. The cache simulation results presented in this paper are all for direct-mapped caches with 16-byte lines, and a page size of 4 KB. The results are presented in misses per instruction (MPI), so that the effect of cache misses on cycles per instruction (CPI) can be easily calculated by multiplying by the miss penalty.

3 Uncolored Allocation

Figure 2 presents misses-per-instruction (MPI) results for a variety of direct-mapped cache sizes. The mean MPI of thirteen different uncolored runs are given, along with the MPI for identity perfect static coloring and round robin dynamic coloring.

For 4 KB caches, all three colorings have identical MPIs, as the cache only has one color. For caches large relative to the program working set, the uncolored allocations approach the minimum miss rate because the probability pages colliding decreases as available empty space increases. For cache sizes near each benchmark's working set size uncolored allocation performs significantly worse than the colored allocations because any coloring flaw results in unnecessary collisions.

For the data caches, the mean MPI of the uncolored allocations tends to be about 20% worse than the colored-allocation MPIs. The instruction cache, however, demonstrates far more dramatic behavior (except for fft and lloops which essentially fit in one I-page). The mean MPI of the uncolored allocations sometimes exceeds the colored MPIs by one to two orders of magnitude. Instruction pages have very regular and dense access patterns, and any coloring collision can cause a large number of misses.
Figure 2: Uncolored, Identity Colored, and Round Robin Colored MPIs
4 Static Coloring

The static coloring algorithm presented here is perfect identity coloring. Perfect static coloring assures absolutely no inter-run variation, as the virtual, and therefore physical, addresses used for cache indexing stay constant between runs.

For instruction caches, perfect static coloring is generally better than round robin in terms of MPI, although it performs poorly at a couple of cache sizes for two of the benchmarks. Page-based spatial locality is higher for the I-stream; the compiler packs instructions into sequential addresses, while data exists in several disjoint segments. The performance of perfect static coloring is obviously a function of how the compiler allocates data and instructions; if the compiler allocated objects to random virtual addresses then there would be no advantage to static coloring over uncolored allocation. The simulations of the unified caches demonstrate the results of virtual-address collisions even more dramatically, as instructions and data collide unnecessarily in large caches (e.g., gcc, gci, and tex)\(^1\). In these cases, perfect static coloring can perform as badly as, or even worse than, the mean uncolored allocation. However, perfect static coloring performs as well as round-robin for small unified caches, better than uncolored allocation for instruction caches, and well for data caches.

5 Dynamic Coloring

The dynamic coloring algorithm presented here is round robin coloring. Round robin coloring does not vary between runs; once an initial color is arbitrarily selected, a trace is completely deterministic. Round robin coloring has the virtue of placing all pages in the cache in different colors if the cache can contain them.

The simulation results for instruction and data caches demonstrate this behavior, as the round robin colorings tend to slightly precede perfect static colorings in reaching the minimum MPI (e.g., spice 128 KB instruction cache). Round robin provides significant performance improvement over uncolored allocation over a large range of cache sizes.

Round robin coloring tends to out-perform static coloring on the data cache because the data space comprises disjoint segments which may overlap in the virtual address space (modulo the cache size), and thus collide in a static-colored cache. This effect is particularly noticeable on the larger unified caches, where the perfect static colored cache performs poorly, while the round robin colored cache performs well.

6 Conclusion

Virtual memory and cache memory interact when the index size (the cache size per degree of associativity) is larger than the page size. This paper shows the mean MPI for a variety of instruction, data, and unified first-level direct-mapped cache sizes. For very small and very large first-level caches, uncolored allocation has little effect. However, for midrange caches (16 KB to 512 KB) colored allocations often reduced the MPI by 20% for data caches, from 50% to 99% for instruction caches, and from 20% to 90% for unified caches.

Round robin dynamic coloring gives good colorings, slightly better than perfect static coloring for data caches, and guarantees an optimal coloring when the cache is larger than the number of pages touched. Round robin performs particularly well on large unified caches.

Identity perfect static coloring performed slightly better on instruction caches than round robin coloring. This coloring performed similar to round robin on data caches and small unified caches, but performed worse than round robin on large unified caches as instructions and data collided unnecessarily. However, identity perfect static coloring allows physically-indexed caches to be virtually indexed, decreasing latency.

REFERENCES


\(^1\)1loops and fft do not collide as significantly because of their very small instruction working set.