PAGE ALLOCATION TO REDUCE ACCESS TIME OF PHYSICAL CACHES

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

Technical Report No. CSL-TR-90-454

November 1990

Supported by NASA under NAG2-248 using facilities supplied under NAGW 419.

PAGE ALLOCATION TO REDUCE ACCESS TIME OF PHYSICAL CACHES

by

Brian K. Bray

William L. Lynch

M. J. Flynn

Technical Report No. CSL-TR-90-454

November 1990

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305

Abstract

A simple modification to an operating system's page allocation algorithm can give physically addressed caches the speed of virtually addressed caches. Colored page allocation reduces the number of bits that need to be translated before cache access, allowing large low-associativity caches to be indexed before address translation, which reduces the latency to the processor. The colored allocation also has other benefits: caches miss less (in general) and more uniformly, and the inclusion principle holds for second level caches with less associativity. However, the colored allocation requires main memory partitioning, and more common bits for shared virtual addresses. Simulation results show high non-uniformity of cache miss rates for normal allocation. Analysis demonstrates the extent of second-level cache inclusion, and the reduction in effective main-memory due to partitioning.

Key Words and Phrases: cache, page allocation, page coloring, physically addressed, virtually addressed, set-large, set-small, virtual memory.

Contents

List of Figures

1 Introduction

Most modern operating systems require paged virtual memory to efficiently provide a large address space. In a virtual memory system, designers face a difficult decision as to where in the memory hierarchy to perform the virtual to physical address mapping. In general, memory accessed virtually (such as a virtually indexed cache) requires significant operating system complexity to prevent data inconsistency problems when two or more virtual addresses map to the same physical address or there is a change in the virtual to physical mapping [Che87]. However, memory accessed physically usually requires virtual to physical translation.

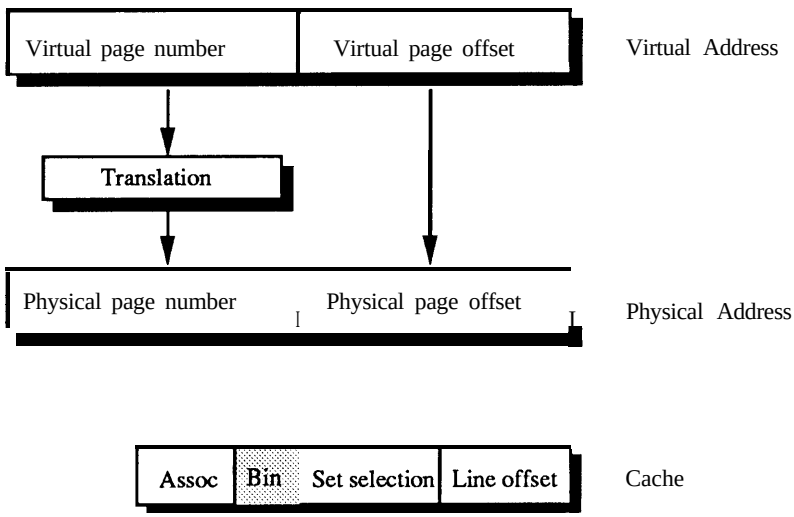

Figure 1: Virtual to Physical Address Translation

Figure 1 shows conventional address translation and cache addressing. ***The set-selection*** bits are required to begin a cache access (index a line in ***the*** cache). ***The associativity*** bits do not actually address the cache, but represent the increase in cache size due to associativity; thus increasing the associativity increases cache size without increasing the number of set-selection bits; i.e., it increases the number of sets but not ***the*** set size. ***The bin*** [KH90] bits (the bits of set selection which are not page-offset bits) indicate in which page-sized section (bin) of a cache a page resides. The number of bins equals the cache size per degree of associativity divided by the page size.

In a ***set-large*** cache, the cache size per degree of associativity (number of sets times the line size) exceeds the page size; conversely, in a ***set-small*** cache, the cache size per degree of associativity is less than the page size. Specifically, a set-large direct-mapped cache is larger than a page, and a set-small direct-mapped cache is smaller than a page. Thus, a set-large cache requires a greater number of bits than are available in the page offset for set selection (the bin bits); a set-small cache does not.

First level caches should have low latency, which implies low associativity, yet low associativity caches should be large enough to maintain a low miss rate [Hil88]. For a given page size, both of these factors drive caches into the set-large domain. Physically indexed caches significantly simplify the operating systems handling of aliases and synonyms [Che87, KW88]. However, physically-indexed set-large caches

conventionally require translation to provide physical bin bits ***before*** cache access (Figure 2). This organization may require cycle extension or an extra pipeline stage to allow the address translation to be performed, offsetting some of the latency advantages of the low-associativity cache. The overall lower-latency organization of translation during the first-level cache access (Figure 3) usually require set-large caches to be indexed with the virtual address. However, this superior organization can be realized with significantly less operating system complexity than virtual-indexing when the page selection algorithm forces the bin bits to correspond in the virtual and physical address space, known as ***page coloring.***

2 Page Coloring

With ***page coloring*** [TDF90] the operating system allocates pages such that some low-order bits of the physical page number are only a function of low-order bits of the virtual page number (Figure 4). Conventionally, the physical bin number is a function of the entire virtual page number; page coloring makes the bin number a function of fewer bits of the virtual address. For a set-large cache, each page resides in a bin of the cache based on the bin bits, and if the physical bin bits are a known function of the virtual bin bits, then physically indexing a cache can begin before full address translation. If the coloring function is the identity function, then the bin bits need no translation.

Table 2 shows (in hex) colored and uncolored translation of two 32-bit virtual addresses into 24-bit physical addresses. The example system consists of a 64KB cache with 16B lines, and 16MB of memory in 4KB pages. Note that with uncolored allocation, the two virtual addresses from consecutive pages collide in the same cache bin; with identity-colored allocation, they cannot.

Memory Address	Page Number		Page Offset		
Cache Address		Bin	Page Offset		
		Set Selection		Line Offset	
	ffeb	a	bc	d	Virtual Address
	34	8	bc	d	Uncolored Physical Address
	e3	a	bc	d	Colored Physical Address
	ffeb	b	bc	d	Virtual Address
	28	8	bc	d	Uncolored Physical Address
	9c	b	bc	d	Colored Physical Address

Table 1: Example of Colored Allocation

This colored allocation merely requires that the operating system keep one free list per bin. Freed pages are added to, and allocated pages are removed from, the correctly colored list.

Synonyms require that low-order bits of the physical and virtual address are identical to allow different virtual pages to map to one physical page. Page coloring forces extra bits to be non-translatable, and thus synonyms must have appropriate colored bits as well as page offset bits. However, operating systems for virtually-indexed-cached machines commonly have their sharing limit defined larger than their page size (128KB for AT&T System V [Che87] and 32KB for Apollo Domain [FR88]), so coloring up to the sharing limit for physically-indexed caches would not require any code changes. Without sharing-code modifications, the sharing limit constrains the size of each (e.g., instruction ***and*** data) direct-mapped cache

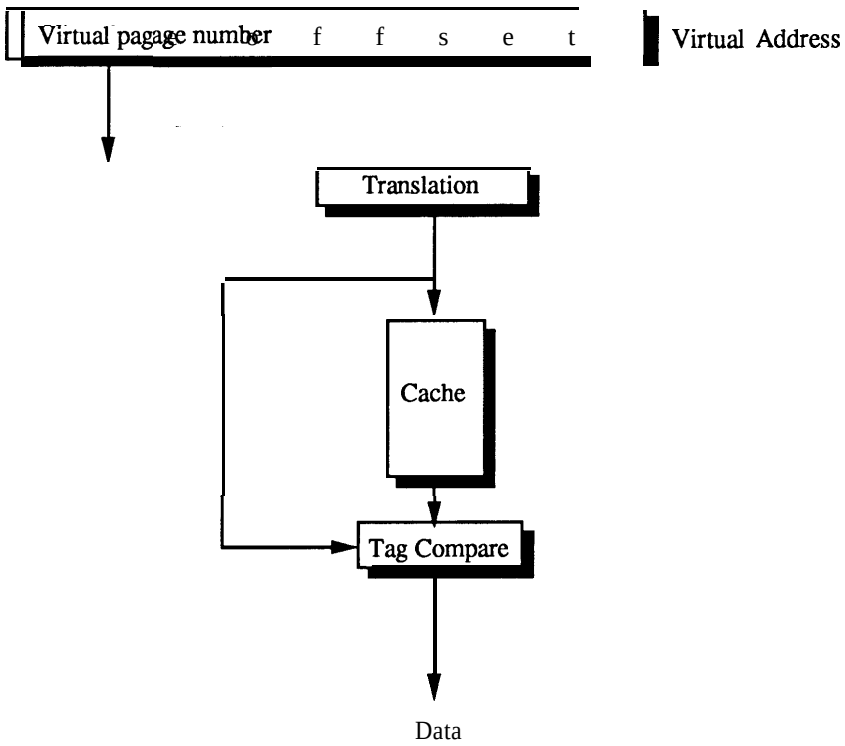

Figure 2: Sequential Translation and Access

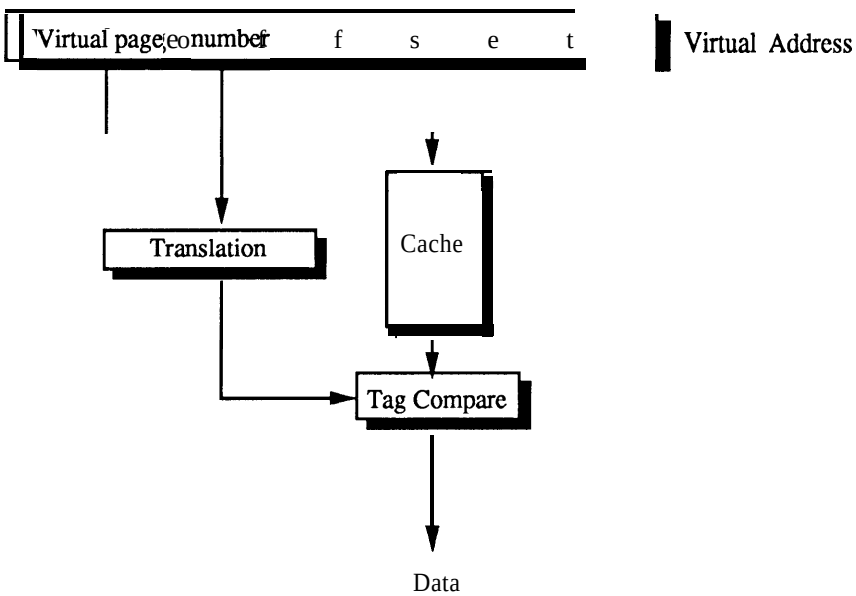

Figure 3: Parallel Translation and Access

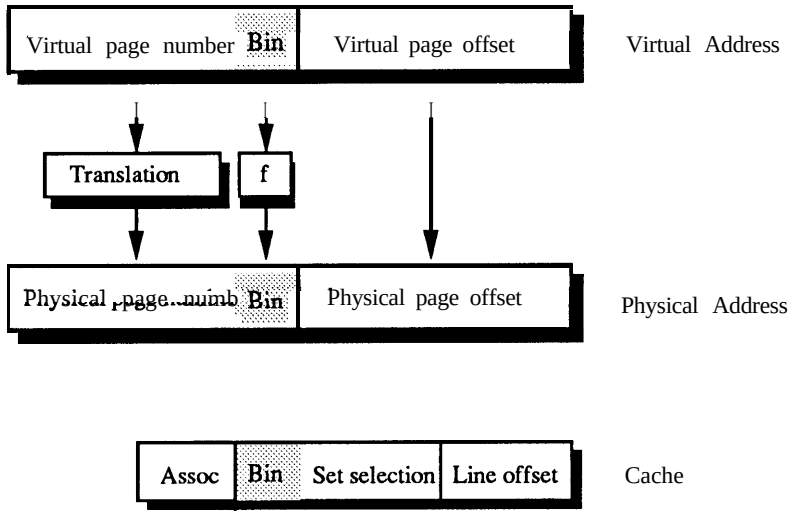

Figure 4: Translation with Coloring

because addresses are colored up to that size. These sharing limits already exceed common first-level cache sizes, which are likely to be in the 32KB to 128KB range. Total cache sizes above 128KB may not improve performance if there is any cycle time penalty [Prz88].

3 Effects of Page Coloring

3.1 Larger TLB

Translation in parallel with cache access allows more time to access the translation lookaside buffer (*TLB*) than if combined into a cycle with the address generation or cache access. With more time, a larger TLB can be accessed, thus decreasing the miss rate, which is desirable as TLBs' can have large miss penalties [CE85]. (Incomplete page coloring on the MIPS R6000 [TDF90] reduces the miss rate of its 8-entry TLB slice.)

3.2 Cache Performance

Apart from [KH90], the effects of page allocation on set-large caches has been largely ignored. Worst-case page allocation (i.e., all pages mapped to one bin) results in an effective cache size equal to the page size! Page allocation can also improve cache performance.

A trace-driven simulator produced data comparing colored and uncolored (conventional) page mapping. An initial random free list is assigned for each uncolored run; free lists are constant over each curve. The coloring function f is the identity function. Figures 5 through 10 present data, instruction, and unified cache miss-rate performance relative to page coloring for our gas, gccl, `spice3`, and `TeX` benchmarks. Each of the four curves for each benchmark are for a different random initial free list.

Virtual memory affects cache performance in a very benchmark dependent way, but some generalizations

can be made. Conventional memory allocation has a significant effect on cache miss rate. Instruction caches have much higher variability than data or unified caches. This variability arises from two main characteristics of instruction reference streams: 1) they have a much greater density than data streams, thus any collisions are exacerbated and are relatively larger, and 2) within a page, instructions are much more likely to cover the entire page than data. Thus, overlapping two or more instruction pages is more likely to cause collisions than overlapping data pages. For large caches, coloring frequently allocates most of the active pages to otherwise empty bins, thus uncolored allocation can only create extra collisions. Identity coloring is not optimal [KH90], and performance can sometimes improve with uncolored allocation, but high variance between runs is undesirable.

Another benefit of page coloring is that the cache mapping is known at compile time, so program layout optimization techniques (such as [McF89]) are not limited in applicability to the page size bins (in physical caches), or virtual caches.

3.3 Inclusion Benefit

The inclusion property of a second level cache is important to reduce the cache coherence complexity of two level cache organizations, and to screen unnecessary cache coherency traffic from the first level cache. [BW88, WBL89] have formulated the necessary associativity of a second level cache to insure inclusion, given by:

$$A_2 \geq (\frac{Size_1}{PageSize}) \times (\frac{B_2}{B_1}),$$

where $S_2 > S_1$, $B_2 \geq B_1$, $Size_2 > Size_1$, and $B_1 S_1 \geq PageSize$;
A_n = associativity in the level n cache, B_n = block size in the level n cache, S_n = number of sets in the level n cache.

A more general definition would be

$$A_2 \geq (\frac{Size_1}{X}) \times (\frac{B_2}{B_1}),$$

$X = minimum(\frac{Size_1}{A_1}$, size of unit where virtual and physical addresses are equal).

Normally:

$$X = minimum(\frac{Size_1}{A_1}, PageSize).$$

With page coloring:

$$X = minimum(\frac{Size_1}{A_1}, 2^b \times PageSize) \text{ for } b \geq 0,$$

where b = number of low order page number bits forced to be the same by page coloring.

Page coloring can significantly reduce the necessary associativity in the second level cache. For example, with PageSize = 4KB, $Size_1$ = 32KB, B_1 = 16B, A_1 = 1, Size2 = 1MB, B_2 = 32B, A_2 would normally have to be at least 16-way associative, but with page coloring and b = 3, A_2 has to be only 2-way associative.

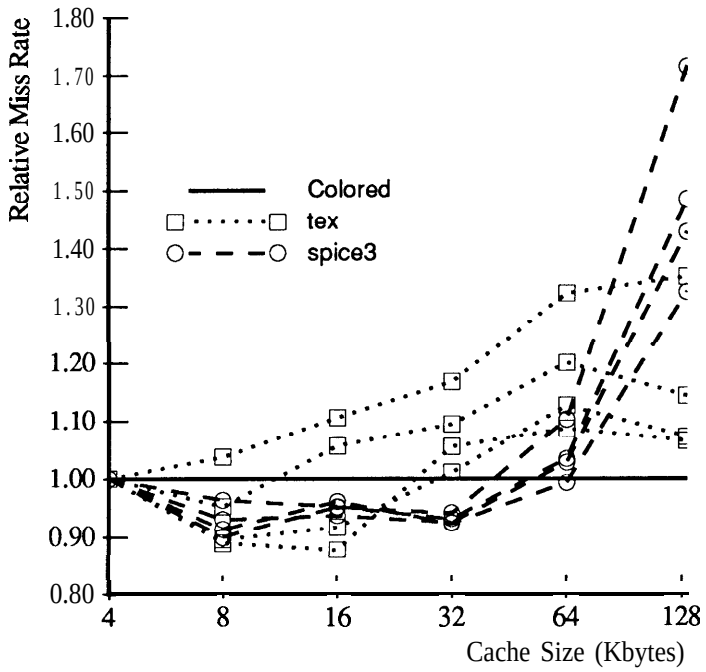

Figure 5: Data Cache - Random Allocation and Coloring – `TeX` and `spice3`

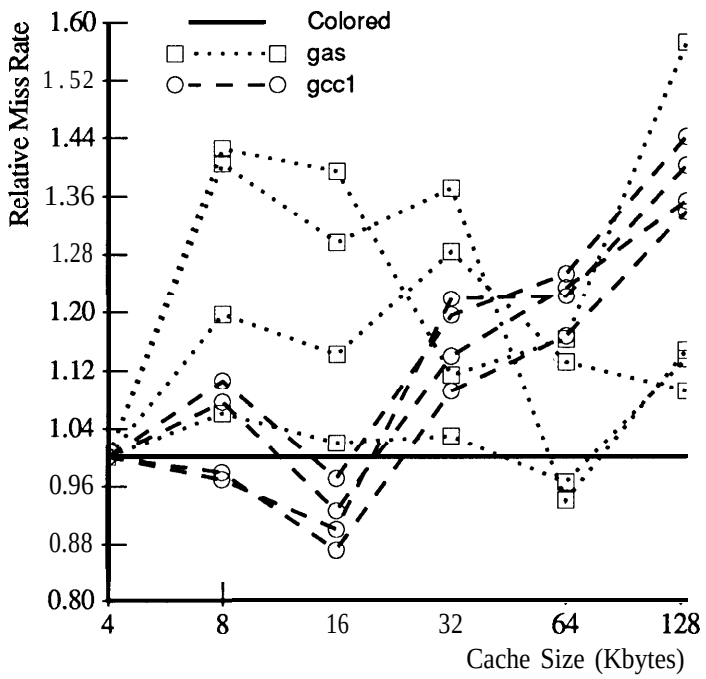

Figure 6: Data Cache - Random Allocation and Coloring – `gas` and `gcc 1`

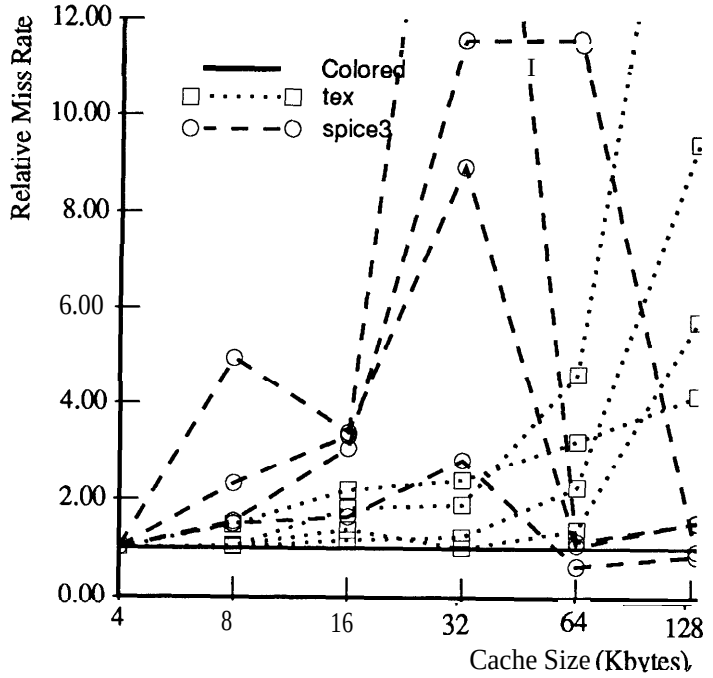

Figure 7: Instruction Cache - Random Allocation and Coloring – `TeX` and `spice3`

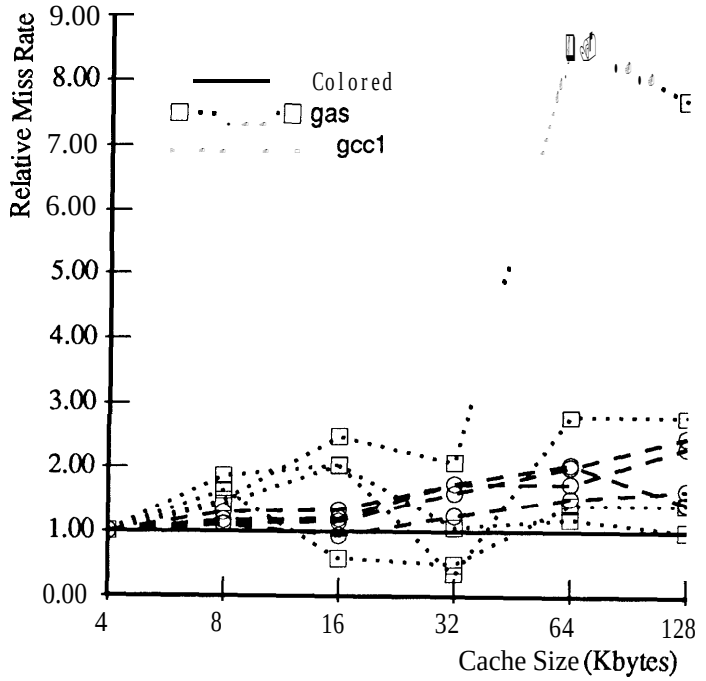

Figure 8: Instruction Cache - Random Allocation and Coloring – `gas` and `gcc1`

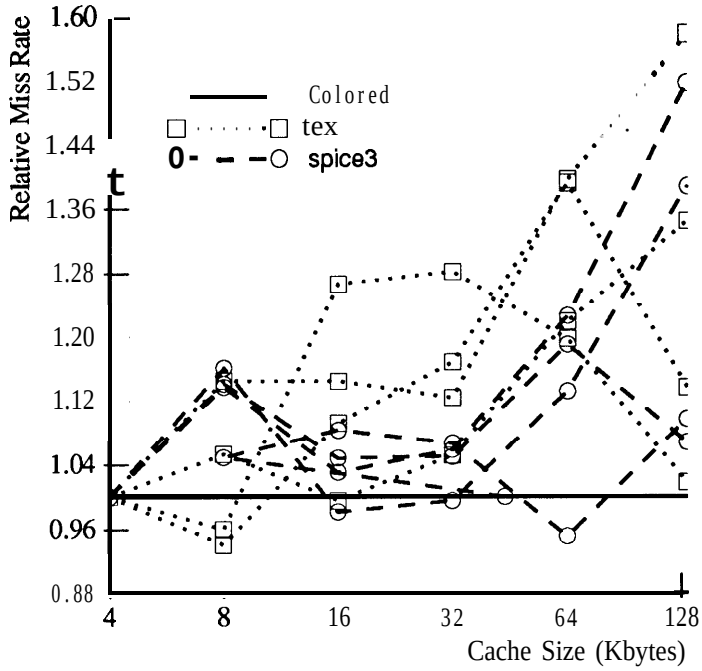

Figure 9: Unified Cache - Random Allocation and Coloring – `TeX` and `spice3`

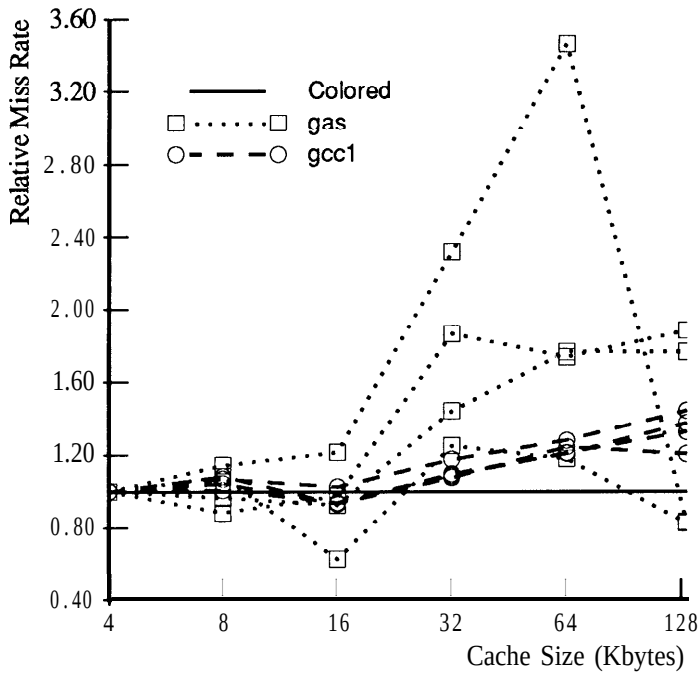

Figure 10: Unified Cache - Random Allocation and Coloring – `gas` and `gcc1`

3.4 Memory Partitioning

Unfortunately, page coloring does not come without a price: main memory is partioned into 2^b sets (where b = number of low order page number bits colored). Page allocation no longer consists of selection of the first element on a free list (fully associative), rather a colored page must be selected from the correctly-colored free list (set associative). A set associative policy frequently performs worse than fully associative, thus causing expensive extra paging. However, it has been shown that with a sufficiently large set size [Smi78], the performance of a page replacement policy can very closely approximate a full associative policy.

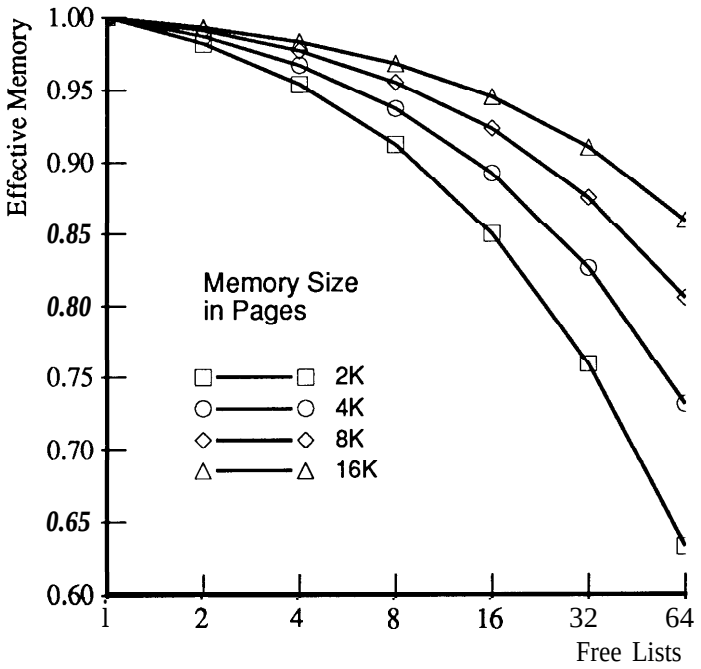

Figure 11: Effective Memory

For example, an entry level machine with 4KB pages, 8MB (2K pages) of main memory, and 64KB instruction and 64KB data cache would have $64/4$ = 16 bins, or b = 4 bits for page coloring. With b = 4, main memory is partitioned into 16 free lists (sets) with 128 pages per list. Therefore, main memory is now 128-way set associative instead of full associative.

Unfortunately, the relatively small number of pages touched by our current benchmarks is too small for realistic paging studies. However, the effect of memory partitioning can be calculated, assuming that the virtual page distribution is random. The problem then becomes a variant of the classic birthday occupancy problem [Par67]. The question is: what is the expected number of pages allocated before one of 2^b lists is full? This number is then the expected equivalent number of pages in a single free list. The solution is given by

$$E(2^b, k) = \int_0^\infty [S_k(t/2^b)]^{2^b} e^{-t} dt,$$

where

$$S_k(x) = \sum_{j=0}^{k} x^j / j!,$$

2^b is the number of free lists, and k is the length of each free list. This analysis is independent of page size.

The approximations given by Parker for $b \rightarrow$ oo are not applicable for this problem, so the above equations were numerically integrated. The results are shown in Figure 11 in terms of the single-free-list equivalent fraction of total partitioned memory, for main memory sizes of 2K, 4K, 8K, and 16K pages, and up to 64 free lists (6 colored bits). For example, a 16K-page main memory partitioned into 16 free lists is as effective as an unpartioned 15.2K-page ($16K$ x 0.95) main memory.

Clearly having many free lists for small memory systems is a bad idea, but the amount of effective memory loss due to coloring is small over a significant design range.

4 Conclusion

Page coloring removes the need for virtual-to-physical address translation to precede cache access for set-large, physically indexed caches. Proper use of page coloring allows set-large low-associativity cache organizations to retain overall low latency and gain repeatable performance without significant operating system complexity. Additionally, the TLB is accessed in parallel with the cache and hence a larger TLB, with a lower miss rate, can be used with no latency increase. Page coloring also reduces the associativity needed by a second level cache for inclusion. The drawbacks are that that main memory is partitioned, effectively decreasing its size, and sharing is limited such that virtual and physical addresses must correspond on larger than page sized objects. Both of these drawbacks can be small over common design spaces.

References

[BW88] Jean-Loup Baer and Wen-Hang Wang. On the Inclusion Properties for Multi-Level Cache Hierarchies. In *Conference* ***Proceedings, The 15th Annual Symposium on Computer Architecture,*** pages 73-80, May 1988.

[CE85] Douglas Clark and Joel Emer. Performance of the VAX-l 1/780 Translation Buffer: Simulation and Measurement. In ***ACM Transactions on Computer Systems,*** pages 31-62, February 1985.

[Che87] Ray Cheng. Virtual Address Cache in UNIX. ***In Usenix Conference Proceedings,*** pages 217-224, June 1987.

[FR88] Craig Frink and Paul Roy. A Virtual Cache-Based Workstation Architecture. In ***2nd IEEE Conference on Computer Workstations,*** pages 80-87, 1988.

[Hil88] Mark Hill. The Case for Direct-Mapped Caches. ***IEEE Computer,*** 21(12):25–40, December 1988.

[KH90] R. Kessler and Mark Hill. Miss Reduction in Large Real-Indexed Caches. Technical Report No. 940, Department of Computer Science, University of Wisconsin-Madison, June 1990.

[KW88] S. Kleiman and D. Williams. SunOS on Sparc. In *CompCon* ***Spring 88,*** pages 289-293, February 1988.

[McF89] Scott McFarling. Program Optimization for Instruction Caches. In ***Conference Proceedings, ASPLOS-III,*** pages 183-191, April 1989.

[Par67] E. T. Parker. A Result in Balanced Incomplete Block Designs. ***Journal*** *of* ***Combinatorial Theory,*** 3:283–285, ***1967.***

[Prz88] Steven Przybylski. Performance-Directed Memory Hierarchy Design. Technical Report No. CSL-TR-88-366, Computer Systems Laboratory, Stanford University, September 1988.

[Smi78] Alan J. Smith. A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory. ***IEEE Transactions on Software Engineering,*** SE-4(2):121–130, March 1978.

[TDF90] George Taylor, Peter Davies, and Michael Fannwald. The TLB Slice – A Low-Cost High Speed Address Translation Mechanism. In ***Conference Proceedings, The 17th Annual Symposium on Computer Architecture,*** pages 355-363, May 1990.

[WBL89] Wen-Hang Wang, Jean-Loup Baer, and Henry Levy. Organization and Performance of a Two-Level Virtual-Real Cache Hierachy. In ***Conference Proceedings, The 16th Annual Symposium on Computer Architecture,*** pages 140-148, May 1989.

A Appendix:

A.1 Benchmarks

Trace driven simulation produced the presented results. The architecture simulated was essentially the MIPS R2000. The four C benchmarks were optimized with the Ultrix 3.1 C compiler with optimization level 02. The code simulated was the application code, string routines, and printf routines, while the code executed in system calls, scanf routines, and math libraries was not simulated.

Name	Description	Instr. (10^6)	% loads	% stores	4KB pages touched
`gas`	gnu assembler - assembling a 1800 line file	6.9	25.8	13.4	104
`gcc1`	gnu C compiler version 1.36 - compiling (and optimizing) to assembly code a 1500 line C program	44.8	23.7	13.0	492
`spice3`	circuit simulator - simulation of a Schottky TTL edge-triggered register	231	38.2	8.8	238
`TeX`	document preparation system - formatting of a 14 page technical report	83.2	24.9	15.6	239

Table 2: Benchmarks