MESA: Reducing Cache Conflicts by Integrating Static and Run-Time Methods

Xiaoning Ding[1], Dimitrios S. Nikolopoulos[2], Song Jiang[3], and Xiaodong Zhang[1]

[1]CSE Department
The Ohio State University
Columbus, OH 43210
{dingxn,zhang}@cse.ohio-state.edu

[2]CS Department
College of William and Mary
Williamsburg,VA 23187
dsn@cs.wm.edu

[3]CCS-3 Division
Los Alamos National Laboratory
Los Alamos,NM 87544
sjiang@lanl.gov

Abstract

The paper proposes MESA (Multicoloring with Embedded Skewed Associativity), a novel cache indexing scheme that integrates dynamic page coloring with static skewed associativity to reduce conflicts in L2/L3 caches with a small degree of associativity. MESA associates multiple cache pages (colors) with each virtual memory page and uses two-level skewed associativity, first to map a page to a different color in each bank of the cache, and then to disperse the lines of a page across the banks and within the colors of the page. MESA is a multi-grained cache indexing scheme that combines the best of two worlds, page coloring and skewed associativity. We also propose a novel cache management scheme based on page remapping, which uses cache miss imbalance between colors in each bank as the metric to track conflicts and trigger remapping. We evaluate MESA using 24 benchmarks from multiple application domains and with various degrees of sensitivity to conflict misses, on both an in-order issue processor (using complete system simulation) and an out-of-order issue processor (using SimpleScalar). MESA outperforms skewed associativity, prime modulo hashing, and dynamic page coloring schemes proposed earlier. Compared to a 4-way associative cache, MESA can provide as much as 76% improvement in IPC.

1. Introduction

Conflict misses are a perennial problem of caches with limited associativity. Caches have gone through a sequence of optimizations to reduce conflicts misses, including victim buffers [14], pseudo-associativity [2] and eventually higher associativity. Architects have considered intelligent cache indexing functions[6, 16, 23] to disperse data more evenly across the cache. Software techniques such as page coloring and bin hopping [7, 15] have also been proposed to reduce conflict misses via careful mapping of pages in the cache. Another approach to the problem is to use run-time information and run-time cache management mechanisms that alleviate hot spots. Techniques such as cache bypassing [13], cache miss classification and isolation [10], and various forms of cache reconfiguration, including reconfigurable associativity and dynamic cache partitioning [4, 9, 21] fall into this category. Page remapping [5, 24] with hardware support is yet another technique that attempts to reduce conflicts by sampling misses and changing page colors at run-time.

Concerned with the high penalties of conflict misses in secondary caches, we make three contributions in this paper. The first is a novel cache indexing scheme called *MESA (Multicoloring with Embedded Skewed Associativity)*. *MESA* combines a skewed-associative page coloring scheme with a skewed-associative set indexing scheme to disperse both pages and cache lines within a page between the banks of set-associative secondary caches. *MESA* associates each page with multiple colors, one per way of associativity in the cache. The memory lines within each page are dispersed using skewed associative mapping, under the constraint that all memory lines in the page are contained within the colors of the page. *MESA* combines the advantages of page coloring in resolving inter-page conflicts with the advantages of skewed associativity in resolving inter-bank conflicts within sets. It is in essence a multi-grained indexing scheme that attempts to eliminate conflict misses within and across pages.

The second contribution of this paper is a dynamic page remapping algorithm for a multicolored secondary cache. The algorithm departs from page remapping algorithms proposed earlier in that it tracks the hottest cache color at run-time and uses *cache miss imbalance* between the hottest and the coldest colors in each bank to make better remapping decisions. The remapping criterion used in our mechanism is more aggressive than criteria based on miss sampling [24]. Results from simulations show that this aggressiveness pays off due to the high cost of conflict misses to memory.

The third contribution of this paper is a comprehensive comparison between static cache indexing and dynamic coloring and remapping schemes for secondary caches. This comparison shows that static indexing schemes perform remarkably well. Most often, they are better than standalone

run-time remapping schemes, if the latter are used without any provision for dispersing cache lines instead of just pages. However, carefully designed run-time algorithms can enhance sophisticated static indexing schemes, by conveying cache miss information which is statically undetectable. This argument is supported by the superior performance of *MESA* when combined with our remapping algorithm. Our integrated scheme reduces significantly the number of secondary cache misses compared to both sophisticated static indexing schemes and standalone dynamic remapping schemes.

We have experimented with a wide set of benchmarks (24 in total) from multiple application domains. Our simulations show that *MESA* outperforms two of the best static cache indexing schemes that disperse memory lines between sets —skewed associativity and prime number based hashing— and one previously proposed dynamic coloring algorithm based on miss sampling, yielding IPC improvements of up to 76% compared to conventional 4-way associative caches, and at least 1%–5% compared to the next best cache indexing scheme.

The rest of this paper is organized as follows. Section 2 discusses static and dynamic cache indexing schemes for avoiding conflicts and motivates multicoloring with skewed associativity. Section 3 presents the implementation and design choices for *MESA* and our dynamic remapping algorithm. Section 4 discusses the experimental setting and Section 5 presents our results. Section 6 overviews related work and Section 7 concludes the paper.

2. Static and Run-Time Schemes for Eliminating Conflicts in Secondary Caches

Previous work adopts both *static* and *run-time* approaches to distribute data in the cache and eliminate hot spots,so that a cache with a small degree of associativity behaves like a cache with a larger degree of associativity. Static approaches use fixed indexing functions for data placement. Run-time approaches utilize on-line dynamic statistics on accesses and misses to adaptively change the mapping of data located in *hot spots* and scatter these data throughout the cache.

Different mechanisms for eliminating conflict misses use different data granularities. *Page-based* mechanisms attempt to eliminate conflicts by intelligently mapping virtual pages to page-sized slots in the cache and decoupling the indexing of physical memory from the indexing of the cache. *Set-based* mechanisms attempt to eliminate conflicts at the granularity of individual cache sets. In the following we review some representative static and run-time schemes, to motivate the work proposed in this paper.

2.1. Page-Based Schemes: Page Coloring and Dynamic Remapping

Page coloring [15] is a well-known technique that reduces conflict misses by mapping consecutive pages in the virtual address space into consecutive cache frames. An alternative to page coloring is bin hopping, in which pages accessed sequentially are mapped to consecutive cache *bins* (page-sized slots). Although intuitive and simple to implement, page coloring variants are to some extent limited in sophistication, since they do not take into account run-time information on cache misses.

A natural extension to static page coloring is dynamic page coloring, in which the hardware tracks information on cache misses and take actions to remap pages to different colors, if these pages are located in hot spots [5, 24]. Dynamic remapping can correct poor operating system decisions and adapt cache mapping to the access patterns of programs. The benefit of dynamic coloring depends on the capability of the hardware to detect cache conflicts and to react to them timely. This issue has not been discussed extensively in the literature. An earlier paper from Sherwood et. al [24] appears to be the only one to investigate this issue in detail. In their scheme, if misses in hot sets are constantly above a threshold, a remapping interrupt is fired and the operating system changes the color of a page that maps to the congested color in the cache. This mechanism will henceforth be referred to as *DynColoring*.

The main drawback of *DynColoring* is that it tracks only colors with number of misses above a fixed threshold in a sampling period. To avoid mis-remapping pages from 'normal-temperature' colors in memory intensive process or execution phases, the threshold is usually set high. Thus, the scheme tends to be more conservative than needed. Moreover, this scheme may detect multiple colors as remapping candidates, without making an informed choice between them to identify which one is the hottest. In other words, the mechanism may detect hot pages but not necessarily the ones that cause most of the conflicts. It may be infeasible to collect the entire information to track the hottest page in hardware. However, it is possible to locate this page if the hardware keeps track of the hottest (not just a hot) color in the cache at any given time. We exploit this idea to derive an improved remapping algorithm based on continuous balancing of misses between the hottest and the coldest color in each cache bank.

2.2. Static Set Indexing Schemes: Skewed Associativity and Related Techniques

Although page-based schemes can reduce conflicts by mapping and moving data at a coarse granularity, static hardware schemes exist that address the same problem at the granularity of cache sets. The seminal work on skewed associativity [23] has introduced a design to scatter data in the banks of associative caches, by allowing as many indexing functions per cache line as the degree of associativity. With skewed associativity, each memory line is mapped to a different offset in each cache bank. Using carefully chosen XOR-mapping schemes [11, 23] or other indexing schemes, such

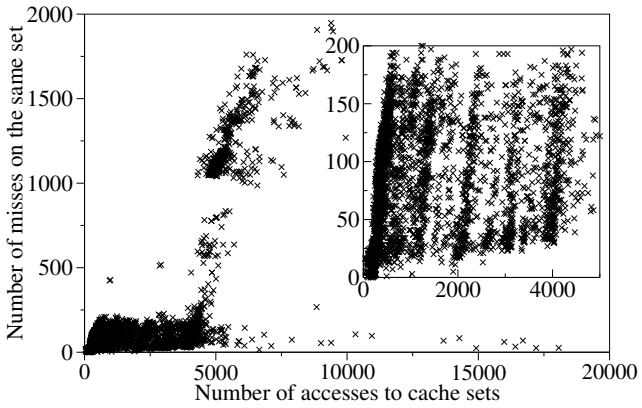

Figure 1. Accesses and Misses per set in NAS LU.

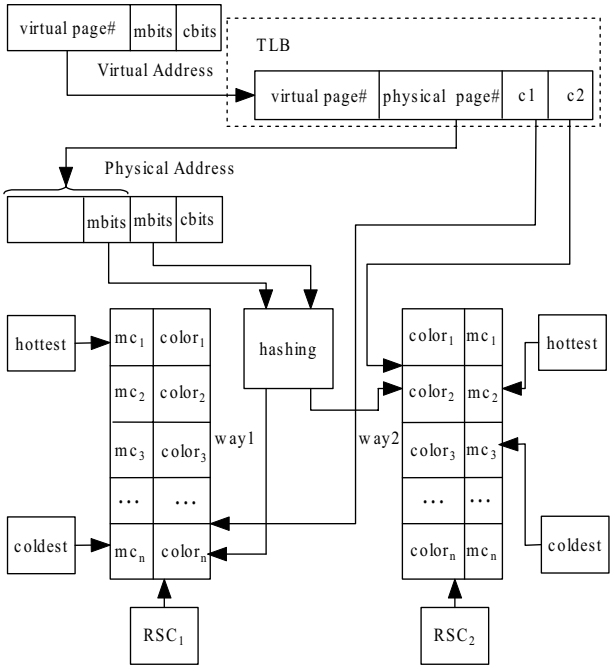

Figure 2. Multicoloring and run-time remapping implementation in *MESA*.

as prime modulo hashing [16], memory lines mapped to a given offset in one way are highly unlikely to map to the same offset in another way. At the same time, consecutive memory lines are unlikely to be mapped to the same cache line in the same bank.

Static schemes try to achieve a highly randomized yet uniform distribution of memory accesses between the cache sets. A balanced distribution of accesses though is not necessarily a reflection of the distribution of cache misses. Figure 1 illustrates the problem. We recorded the number of accesses and the number of misses per cache set while running NAS LU, a program which suffers from conflicts in the L2 cache. The (X,Y) scatter plot has a point per cache set. For each set, the X coordinate is the number of accesses to the set in a typical execution phase of the program and the Y coordinate is the number of misses to the same set. The embedded plot is a zoom into the lower left corner of the chart. The larger the X coordinate the higher the access frequency to the cache set. The chart shows several dark regions (large concentrations of cache sets) with high access frequencies but low miss frequencies. Such regions are shown more clearly in the zoom chart, for sets with 1K to 4K accesses.

2.3. MESA Outline

Interestingly enough, the literature has given little emphasis on comparing and combining sophisticated static set-indexing schemes against dynamic, page-based schemes in terms of effectiveness in reducing conflicts. A cost-effective scheme should make a trade-off between fully static and fully dynamic approaches. One important contribution of our work is that we provide such a framework and quantitative evidence on which of three approaches, namely static set indexing, hardware page coloring or an integrated scheme is the best for a wide set of programs.

We propose a new cost-effective scheme called *MESA (Multicoloring with Embedded Skewed Associativity)*, which works in a two-level hierarchical fashion. Its high level component is responsible for static and dynamic page (color) mapping, while its low level component is responsible for static cache line skewing within the colors of each page.

MESA treats a set-associative L2 cache as multiple banks. Rather than using only one color per page and having the color span all cache banks for a page, as in conventional coloring, *MESA* associates each page with multiple colors, as many as the degree of associativity. The colors of each page are identified with a skewed associative mapping scheme, applied at the page level. Furthermore, a new dynamic recoloring algorithm based on miss balancing is implemented on the multi-colored pages. In this way, each bank is treated as a large direct-mapped cache [5, 15], in which MESA utilizes access information to resolve intra-bank conflicts.

The fixed placement of memory lines within a page can still cause inter-bank conflicts, despite remapping. To resolve these conflicts, MESA disperses the memory lines within a page across the ways of the cache exactly as in a conventional skewed associative cache. The difference from a conventional skewed associative cache is that each line is confined within one of the colors assigned to the page by *MESA*.

3. Implementation Details

MESA divides the cache into pages (colors) and uses the TLB to identify the color of each virtual page, as in conven-

tional page coloring [24]. The main difference with previous coloring schemes is that each page has multiple colors, represented by multiple fields in the TLB entry. The TLB entry includes as many colors as the number of banks (ways) in the cache. The color entries of each page are modifiable via a dynamic remapping algorithm. In addition, hardware similar to that used in a skewed associative cache is used to implement skewing functions to select the color of a page in each bank. The following discussion describes an implementation of *MESA* for large secondary caches.

3.1. Static Multicoloring

Figure 2 depicts the hardware implementation of *MESA* for a cache with two cache banks. The 2-way associative cache is used merely as an example for illustration purposes. Each TLB entry includes two color fields c_1 and c_2. The color fields enable remapping of a page without memory copying. Each field stores the color of the page in one way (bank) of the cache. The cache color bits decouple cache indexing from physical memory indexing. *MESA* performs an N-dimensional division of the cache into pages (N represents the degree of associativity), whereas standard coloring performs a one-dimensional division. We use a skewed associative mapping similar to the one proposed in [6], of a page number from the TLB to set the initial colors of the page.

3.2. Miss Balancing Algorithm

Our dynamic remapping scheme is based on an algorithm called MBA (Miss Balancing Algorithm), which uses cache miss imbalance between colors *independently* in each bank, as the remapping trigger. The basic idea is to remap pages from the hottest (most overloaded) color in a bank to the coldest (most underloaded) color in the same bank, with the intuition that the hottest color tends to be the host of pages that are heavily responsible for conflicts and that reshuffling these pages in the bank is likely to ameliorate the problem. Note that a cache color is a page-sized block of cache sets *inside* a bank. Our remapping algorithm attempts to avoid inter-bank interference and retain the benefit of skewed associativity by not moving pages across banks.

The remapping algorithm uses one miss counter per color per bank. These counters are denoted with $mc_1, mc_2, \ldots, mc_n$ in Figure 2. The counters are incremented upon cache misses. They are aged periodically (every 16K misses in our implementation) so that they do not accumulate stale historical information. Aging is done progressively by rightward bit shifting of the counters.

For each cache bank, two additional counters are used to point to the hottest and the coldest color in the bank. Upon a miss to a color in the cache, the color miss counter is compared to the hottest color's miss counter and the hottest color pointer is updated, if needed. The overhead of this check is overlapped with the memory access. The coldest color is maintained upon cache accesses as follows. On each access to the cache to a specific color, the miss counter of the color is compared against the miss counter of the coldest color, which is initialized to 0. If the currently accessed color has less misses, the coldest color pointer is updated. The overhead of this check is overlapped with the cache access.

Remapping is effected by moving pages from the hottest color to the coldest color in the same bank. It is necessary to throttle remapping activity for several reasons. First, remapping does have a non-negligible overhead. Second, too aggressive remapping may thrash the cache by frequently moving pages back and forth between two colors. To throttle remapping we use two thresholds. The first threshold is used for controlling the speed of remapping (we call it the RSC threshold, where RSC stands for *R*emapping *S*peed *C*ontrol). An RSC counter per bank is used for this purpose. The counter is initialized to 0, and incremented whenever a cache miss occurs in the hottest color. While the cache miss is being served, the miss counters of the coldest and the hottest color are compared, and remapping is triggered if the RSC of the hottest color exceeds a hardwired threshold, set to 32 in the current implementation. The RSC counter is reset to 0 after one page is remapped from the hottest color.

The second threshold corresponds to the minimum cache miss imbalance (CMI_{min}) between the hottest and the coldest color that needs to be observed by the hardware to trigger remapping. CMI_{min} is set to be equal to the number of misses in the coldest color. In other words, the hottest color should see twice the number of cache misses of the coldest color to activate remapping. The specific threshold allows for a fast calculation of CMI in hardware.

3.3. Embedded Skewing

When a memory line is accessed, *MESA* uses a skewed associative scheme that provides as many candidate set indices for the line, as the degree of associativity. The c_i field in the TLB determines the cache color (page) in the ith cache way for the memory line, as well as the starting index of that color. We use hashing hardware and a XOR-based indexing scheme in the address of the memory line to generate as many offsets as cache ways, which are in turn used to locate the target cache set inside each color. In our prototype implementation we use the same skewing functions as in [6], although alternative indexing schemes can be considered to resolve pathological cases [16, 26].

4. Experimental Setting

We used Simics [19], a complete system simulator, to evaluate *MESA* and other static and run-time schemes for reducing cache conflicts. We used the out-of-the-box UltraSPARC-II processor of Simics, executing the SPARC V9 instruction set and running SPARC Linux 7.2. While Simics is capable of conducting a complete system simu-

lation and evaluating the performance impact of *MESA* in the system as a whole, it models in-order issue processors. To show the effectiveness of *MESA* for modern out-of-order issue processors, we used SimpleScalar [8], and modeled a 4-issue, out-of-order processor. We set the load/store queue(LSQ) size to be 8, and used the Alpha instruction set for the SPEC 2000 benchmarks, and the PISA instruction set for the rest of the benchmarks.

We simulated a two-level cache hierarchy. The parameters are given in Table 1. The size of the secondary cache is set to 1MB, to realistically support the data sets used in our benchmarks.

Parameter	Value
L1 cache	32KB, 32-byte lines, 2-way set associative, write-back,write-allocate, 1 cycle latency,LRU
L2 cache	1MB, 64-byte lines, 4-way set associative, 10 cycles latency, LRU replacement
TLB	64-entry iTLB, 64-entry dTLB, fully associative
Memory	4K page size,120 cycles latency in Simics, 80 cycles for the first chunk and 10 cycles for each additional chunk in SimpleScalar

Table 1. Simulation parameters.

Parameter	Value
Length of interval	16K misses
Miss counter aging	right shift 4 bits each interval
RSC threshold	32
Remapping overhead	400 cycles

Table 2. Remapping algorithm parameters.

For the simulation of *MESA*, we chose the parameters listed in Table 2. Besides *MESA*, we implemented four other schemes for comparison, namely, *4-way* (a standard 4-way set-associative L2 cache), *DynColoring* (a conventional coloring scheme with dynamic remapping on a 4-way cache [24]), *prime* (a 4-way associative cache with prime modulo hashing [16]), and *skewed* (a 4-way skewed associative cache [23]). *4-way* is used as a baseline for comparisons. *DynColoring* is used to assess how much additional benefit *MESA* can obtain from its multicoloring design, as well as to investigate if the remapping algorithm used in conjunction with *MESA* achieves further improvements. We compare *prime* and *skewed* against *MESA* to investigate whether run-time remapping reduces conflict misses further than plain skewed associativity or prime modulo hashing.

In the experiments, we used 4093 as the prime number used in the hashing function of *prime*, which is very close to 4096, namely the number of cache sets of the L2 cache used in our experiments. We used low thresholds for both the sampling miss counters and the interrupt miss counters in the implementation of *DynColoring* (64 and 4 as opposed to 128 and 8 used in the original implementation [24]), both because we are using a smaller page size than in the original work and because we found that the original thresholds were overly conservative for the workloads we tested.

We evaluated the aforementioned schemes with a total of 24 applications, including 19 applications from SPEC CPU2000, 4 applications from the NAS Benchmarks [3], and one application (FFT) from SciMark 2.0 (C version) [1]. We used the *reference* input size for SPEC benchmarks, the Class A input size for the NAS benchmarks except from LU, in which we used the Class C input size, and the *large* input size for SciMark. All benchmarks were run on a simulated uniprocessor system. The results include operating system activity as recorded in Simics. For each application except FFT, we simulated 2 billion instructions starting with a cold cache. FFT completes in about 937 million instructions in Simics and 755 million instructions in SimpleScalar.

5. Simulation Results

We divide the results from the 24 programs into three groups: Group 1 contains the 10 integer programs from SPEC 2000, group 2 contains the 9 floating point programs from SPEC 2000, and group 3 consists of the rest programs from the NAS suite as well as FFT from the SciMark suite.

5.1. Overall Results

5.1.1. Results with an In-Order Issue Processor

The normalized numbers of L2 misses for the programs in each group are shown in Figures 3, 5, and 7, respectively. All results are normalized to those of the baseline 4-way associative cache. The run times of the programs in each group are shown in Figures 4, 6, and 8, respectively. We break run times into busy CPU cycles and memory stall cycles. We observe that across a wide and diverse range of programs, *MESA* provides consistently better performance than the baseline 4-way set-associative cache, though the performance gains vary between programs. Programs like $ammp$, FFT, gcc, LU, $twolf$ and $sixtrack$ obtain significant performance improvements while programs like $applu$, gap, $gzip$, IS, and $wupwise$ obtain little or no benefit from any of the conflict avoidance schemes. The other programs obtain some benefit by suffering less L2 cache misses, but this benefit is not translated into substantial performance improvements, as these programs access the L2 cache rarely. The reaction of the programs to different cache mapping algorithms can be explained by characterizing the imbalance of their miss distributions across cache sets. To characterize the instantaneous miss distributions, we calculate the coefficient of variation (CoV) of misses for every interval of 16384 misses, and average the CoVs. The larger the ratio, the more serious the imbalance. Figure 9 shows the ratios for all 24 programs.

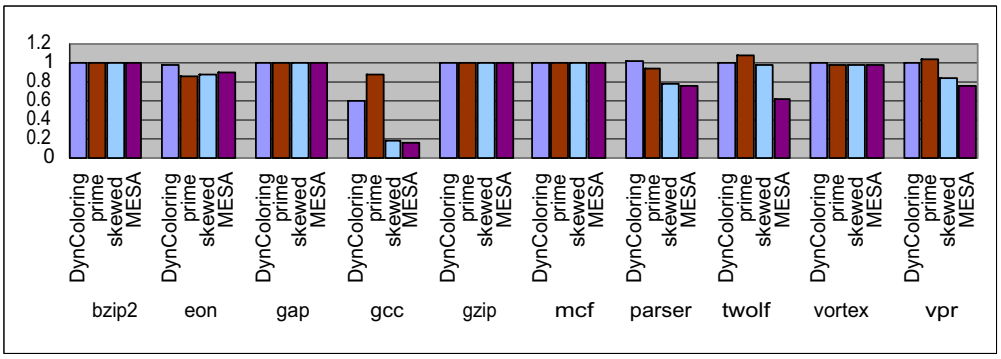

Figure 3. Normalized number of L2 cache misses of integer benchmarks in SPEC 2000.

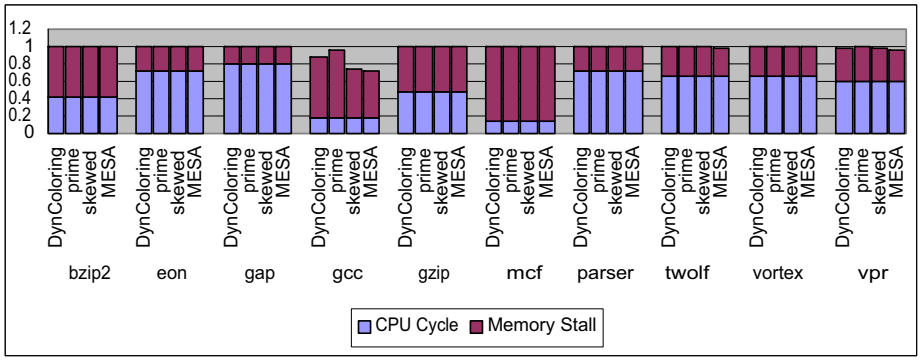

Figure 4. Normalized run time of integer benchmarks in SPEC 2000.

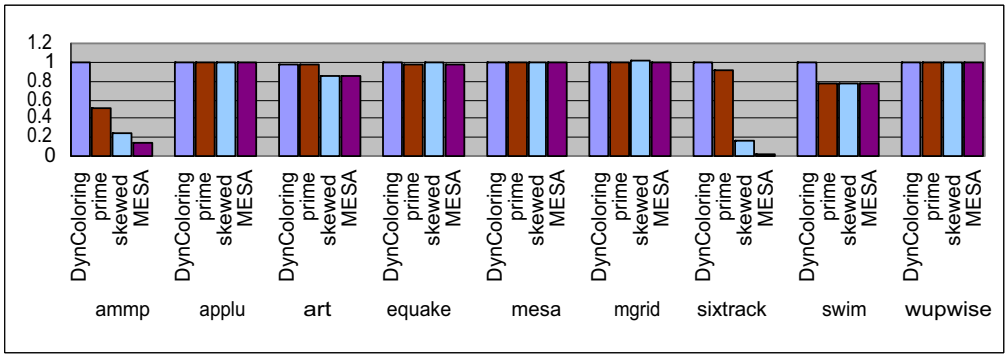

Figure 5. Normalized number of L2 cache misses of floating point benchmarks in SPEC 2000.

For programs with high average CoVs, which indicate serious load imbalance among cache sets, we observe significant improvements from *MESA* in terms of number of misses and run times. For example, FFT has an average CoV of 1.9. Accordingly, *MESA* can reduce its misses by 52.1%, and reduce its run time by 41.4%. This happens because the imbalance causes a large number of conflict misses for overloaded cache sets, while leaving other sets underutilized. On the other hand, for $gzip$, a well-balanced program with a very low average CoV of 0.08, *MESA* reduces its misses by a mere 0.2% and its run time by 0.1%.

For the programs with serious miss imbalance, *DynColoring*, *prime* and *skewed* also achieve better performance over the baseline scheme by reducing conflict misses. However, their improvements are consistently lower than those from *MESA*. The performance advantages of *MESA* over *DynColoring*, *prime* and *skewed* are shown in Table 3.

In general, *prime* and *skewed* are more effective than *Dyn-*

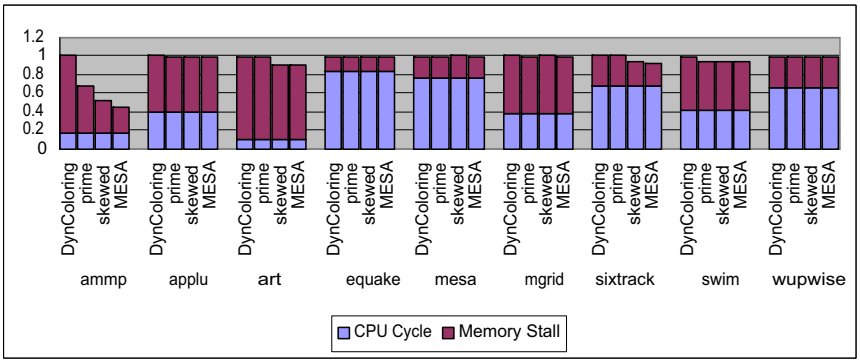

Figure 6. Normalized run time of floating point benchmarks in SPEC 2000.

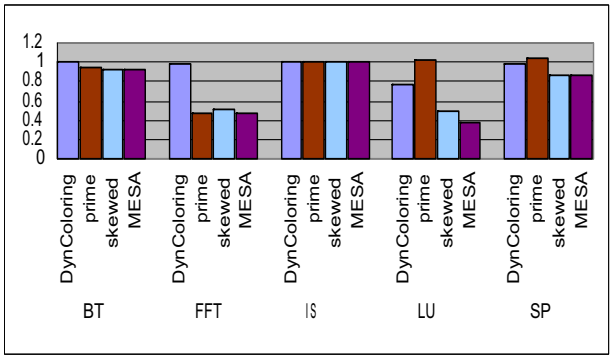

Figure 7. Normalized number of L2 cache misses of NAS benchmarks and FFT benchmark.

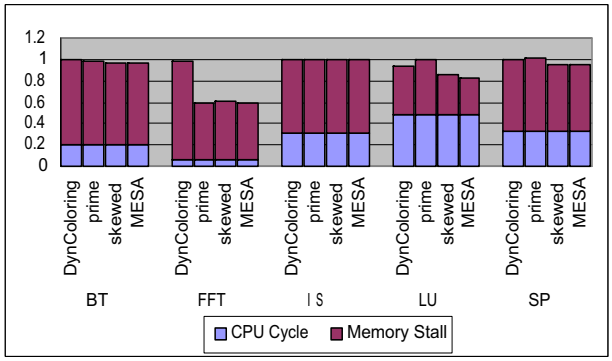

Figure 8. Normalized run time of NAS benchmarks and FFT benchmark.

Coloring. Some imbalance that is detected at the fine granularity of cache lines remains undetected at the coarse granularity of pages. Both *prime* and *skewed* resolve this imbalance implicitly, by dispersing hot spots at the granularity of cache lines. The run-time *DynColoring* scheme cannot afford such an option because of its excessive implementation overhead. In some programs such as eon, *prime* performs better than *skewed*. In other programs such as $ammp$, LU, and $sixtrack$, *skewed* performs better than *prime*. The average reductions shown in Table 3 indicate that *skewed* is better than *prime* in general.

5.1.2. Results from an Out-of-Order Processor

We observe that there is a strong correlation between reductions of secondary cache misses and the improvements in runtime in all benchmarks. An out-of-order execution processor may weaken this correlation due to memory access overlapping, thus lessening the improvements. To investigate whether *MESA* can achieve significant performance improvements in an out-of-order processor, we ran experiments with the SimpleScalar simulator using five representative benchmarks, namely *vpr* and *twolf* from group 1, *ammp* from group 2, *FFT* from group 3, and a new SPEC 2000 benchmark *galgel*, which we could not compile to run in Simics. We compare the performance of *MESA* with *skewed*, which is the cache management scheme which has exhibited the best performance among the schemes used to compare against *MESA* in our simulations with Simics. The run times and miss reductions achieved by the *skewed* cache and *MESA* running on SimpleScalar are shown in Table 4. We can see that the relative performance trends observed in our simulations with Simics remain the same in the simulations with an out-of-order processor. For both *ammp* and *FFT*, skewed associativity can reduce misses and run times by a large percentage, but *MESA* can still outperform skewed associativity due to its ability to dynamically resolve more conflicts. This trend is consistent with the results on the in-order issue processor. However, both the skewed cache and *MESA* cannot reduce run times as much as in the in-order issue processor in *FFT*. The reason is that the out-of-order processor overlaps successfully a significant fraction of L2 cache

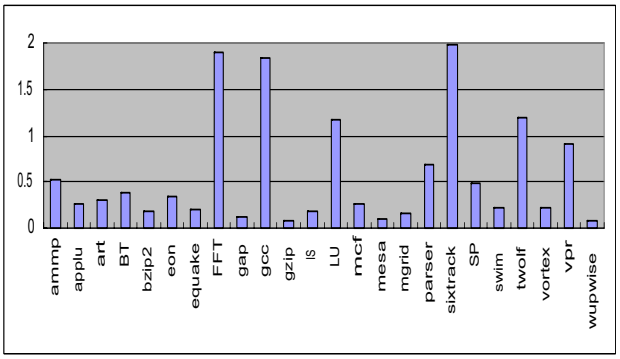

Figure 9. Average CoVs showing the balance of miss distribution.

Reduction in Cache Misses			
	group 1	group 2	group 3
DynColoring	4.31%	3.83%	0.18%
prime	8.41%	2.16%	9.65%
skewed	20.16%	13.67%	21.82%
MESA	22.70%	18.18%	24.56%
Run Time Improvements			
	group 1	group 2	group 3
DynColoring	1.02	1.01	1.00
prime	1.14	1.00	1.05
skewed	1.18	1.04	1.11
MESA	1.20	1.05	1.15

Table 3. Reductions in cache misses and run time improvements of various confict avoidance schemes.

misses. For *vpr* and *twolf*, *skewed* can hardly reduce misses and run times. In these two programs, *MESA* can reduce L2 misses by about 14% and 23%. It therefore provides measurable runtime improvements, despite that these programs have very low L2 miss rates. For *galgel*, *skewed* can reduce a large percentage of L2 cache misses. However, the reduction of cache misses can hardly be reflected on runtime. *MESA* eliminates 20% more cache misses, most of which can not be overlapped, and achieves a measurable IPC improvement. In all five programs, *MESA* outperforms *skewed*. Finally, *MESA* achieves up to 76% improvement in IPC compared to a conventional 4-way associative cache, which implies that it can serve as a good substitute for caches with large associativity and number of banks.

5.2. Analysis of *MESA* Performance Gains

To investigate how much each component of *MESA* contributes to the total performance improvement and to understand why each component is necessary to obtain an integrated cache scheme with consistently improved performance, we take the following steps: We first evaluate the remapping algorithm of *MESA* in isolation (i.e. without any support for skewed-associative multicoloring), in order to assess the effectiveness of using miss imbalance for run-time elimination of conflict misses. For this purpose, we replaced the remapping algorithm of *DynColoring* with our *MBA* remapping algorithm, and obtained a scheme, which we also call *MBA*. We then evaluate a stripped-down implementa-

Reduction in Cache Misses					
	twolf	vpr	galgel	ammp	FFT
skewed	1.24%	3.37%	40.99%	88.13%	44.49%
MESA	14.28%	23.94%	60.85%	95.38%	49.13%
Run Time Improvements					
	twolf	vpr	galgel	ammp	FFT
skewed	0.13%	0.61%	2.47%	71.26%	29.94%
MESA	1.23%	3.90%	6.47%	75.62%	32.83%

Table 4. Performance of *MESA* and *skewed* in out-of-order processors.

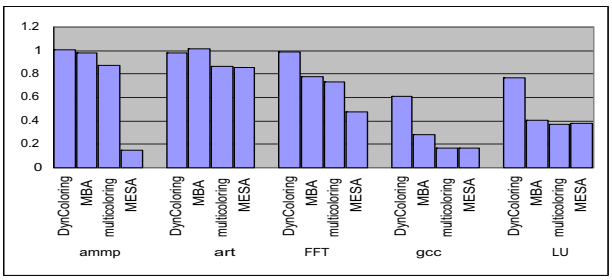

Figure 10. Normalized number of misses for full and stripped-down implementations of *MESA* and *DynColoring*.

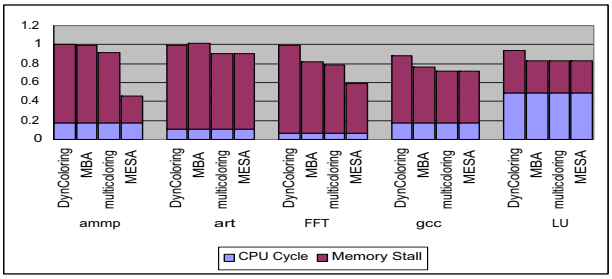

Figure 11. Normalized run times for full and stripped-down implementations of *MESA* and *DynColoring*.

tion of *MESA* called *multicoloring*, in which a skewed associative mapping is enforced for multi-colored pages across banks, but not for memory lines within a page. Comparison of the simplified *MESA* against *MESA* coupled with the *MBA* scheme and line-level skewed associativity gives an idea of the effectiveness of fine-grain skewed associativity indexing on cache performance.

We selected five programs, namely $ammp$, art, FFT, gcc, and LU, where *MESA* achieves striking performance improvements over the two intermediate schemes (*MBA* and *multicoloring*), as well as over *DynColoring*. Figure 10 plots the normalized number of misses of *DynColoring*, *MBA*, *multicoloring*, and *MESA*. Figure 11 plots the normalized run times achieved with the same schemes. All results are normalized to those of the 4-way set-associative cache.

From the results we make the following interesting observation. Though for the five programs *MESA* always achieves considerable performance gains, the amount of contributions made by each of its components vary significantly among the programs. For example, in LU, *MBA* obtains the most performance improvement over its counterpart, namely *DynColoring*. This means that our remapping algorithm is more effective in identifying and resolving hot spots. For the same program, *multicoloring* and *MESA* achieve much lesser improvements over their coarse grain dynamic counterpart, namely *MBA*. This shows that skewed indexing on either a page-sized granularity or a cache-line granularity is not by itself effective for the access patterns exhibited in the specific program. For $ammp$, we observe different trends. $Multicoloring$ and *MESA* generate the most performance gains, while *MBA* contributes little to performance. The experiments show that the effectiveness of each individual technique depends heavily on the specific program and its secondary cache access patterns. While none of the static and run-time techniques prevails across all programs or access patterns, a comprehensive scheme like *MESA*, which integrates both static and dynamic mechanisms, yields a consistent performance improvement.

6. Related Work

This section overviews related work which shares similar objectives with MESA and our remapping algorithm.

Column caching allows software to specify which specific data is mapped to specific ways (columns) of a set-associative cache [9]. Under a column caching scheme, pages are mapped to tints (the equivalent of colors), so that data within a tint cannot evict data from other tints. Our scheme bares partial similarity to column caching in that it restricts the remapping of pages within columns. The column caching scheme proposed in [9] allows for dynamic remapping of data, however no specific information on remapping mechanisms and their performance was provided in this work.

Min and Hu [20] have proposed to decouple cache accesses from memory accesses to allow for more flexible mapping of cache accesses and to avoid conflict misses. Similar decoupling is encountered in other dynamic page remapping schemes, including ours. Contrary to Min and Hu's work which uses reference information to tune cache mapping, we use cache miss information for the same purpose.

Page coloring is a well-known technique to reduce cache conflicts [5, 7, 15]. Software page coloring is simple to implement, however it incurs significant overhead in the operating system. It also interferes with page replacement algorithms, since the operating system needs to keep per-color free lists. And it lacks the information for making informed recoloring decisions at run-time. Hardware page coloring schemes have been proposed for large secondary caches and instruction caches [24, 18]. Our work differs from these schemes in the mechanism used to activate remapping and in that it combines run-time remapping with static schemes to reduce cache conflicts. Compiler transformations for reducing conflict misses have also been proposed [22]. However, such transformations are only applicable in the limited context of loop-intensive scientific kernels.

7. Conclusions and Future Work

This paper makes several contributions in the direction of designing more effective secondary caches with a reasonable degree of associativity. We introduced MESA, a two-level skewed associative cache indexing scheme that merges effectively skewed-associative page coloring with skewed-associative cache indexing. We have extended MESA with a new run-time page remapping algorithm based on cache miss imbalance within the banks of a set-associative cache. Our integrated scheme combines the advantages of skewed associativity, page coloring and page remapping to yield a cost-effective, high-performance design. This paper has also shown that fine-grained static indexing schemes are often more effective than coarse-grain dynamic remapping schemes, but at the same time, static and dynamic schemes can complement each other to maximize the benefits of both approaches.

Although this paper provides a proof of concept by showing that MESA combined with our remapping algorithm can reduce significantly conflict misses, we plan to perform an accurate estimation of the impact of multicoloring on cache area, cache access time and power consumption using a cache model described in [25]. We also plan to investigate MESA in the context of more aggressive processor architectures, by looking at the impact of multicoloring on the caches of multithreaded processors and chip multiprocessors, which suffer from destructive interference between threads, as well as in caches with non-uniform access latencies [17] and software-managed caches [12].

8. Acknowledgments

This research is supported by the National Science Foundation (Grants CCF-0346867, ACI-0312980, CNS-0521381, CNS-0098055, CNS-0405909, and CNS-0509054/0509061) the U.S. Department of Energy (Grant DE- FG02-05ER2568), and an equipment grant from the College of William and Mary. The work with the third author is supported by Los Alamos National Laboratory under grant LDRD ER 20040480ER.

References

[1] J. S. 2.0. http://math.nist.gov/scimark2.

[2] A. Agarwal and S. Pudar. Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches. In *Proc. of the 20th Annual International Symposium on Computer Architecture (ISCA-20)*, pages 179–190, San Diego, California, May 1993.

[3] D. Bailey, T. Harris, W. Saphir, R. V. der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Dec. 1995.

[4] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory Hierarchy Reconfiguration for Energy and Performance in General Purpose Processor Architectures. In *Proc. of the 33rd Annual International Symposium on Microarchitecture (MICRO-33)*, pages 245–256, Monterey, California, Nov. 2000.

[5] B. Bershad, D. Lee, T. Romer, and J. Chen. Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches. *ACM SIGPLAN Notices*, 29(11):158–170, Nov. 1994.

[6] F. Bodin and A. Seznec. Skewed Associativity Enhances Performance Predictability. In *Proc. of the 22nd Annual International Symposium on Computer Architecture (ISCA-95)*, pages 265–274, St. Margherita Ligure, Italy, June 1995.

[7] E. Bugnion, J. Anderson, T. Mowry, M. Rosenblum, and M. Lam. Compiler-Directed Page Coloring for Multiprocessors. *ACM SIGPLAN Notices*, 31(9):244–255, Sept. 1996.

[8] D. Burger and T. Austin. The SimpleScalar toolset, version 2.0. Technical report, University of Wisconsin-Madison, June 1997.

[9] D. Chiou, L. Rudolph, S. Devadas, and B. Ang. Dynamic Cache Partitioning via Columnization. Technical report, MIT LCS, 2000. CSG Memo 430.

[10] J. Collins and D. Tullsen. Runtime Identification of Cache Conflict Misses: The Adaptive Miss Buffer. *ACM Transactions on Computer Systems*, 19(4):413–439, Nov. 2001.

[11] A. Gonzàlez, M. Valero, N. Topham, and J. Parcerisa. Eliminating Cache Conflict Misses through XOR-based Placement Functions. In *Proc. of the 1997 ACM International Conference on Supercomputing (ICS'97)*, pages 76–83, Vienna, Austria, July 1997.

[12] E. Hallnor and S. Reinhardt. A Fully Associative Software-Managed Cache Design. In *Proc. of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 107–116, Vancouver, Canada, June 2000.

[13] T. Johnson, D. Connors, M. Merten, and W. Hwu. Run-Time Cache Bypassing. *IEEE Transactions on Computers*, 48(12):1338–1354, Dec. 1999.

[14] N. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In *Proc. of the 17th Annual International Symposium on Computer Architecture (ISCA-17)*, pages 364–373, Seattle, Washington, May 1990.

[15] R. Kessler and M. Hill. Page Placement Algorithms for Large Real-Indexed Caches. *ACM Transactions on Computer Systems*, 10(4):338–359, Nov. 1992.

[16] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee. Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses. In *Proc. of the 2004 International Symposium on High Performance Computer Architecture (HPCA-10)*, pages 288–299, Madrid, Spain, Feb. 2004.

[17] C. Kim, D. Burger, and S. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. In *Proc. of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, pages 211–222, San Jose, California, Nov. 2002.

[18] S. Kim, N. Vijaykrishnan, M. Kandemir, and M. Irwin. Energy-Efficient Instruction Cache using Page-Based Placement. In *Proc. of the 2001 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'2001)*, pages 229–237, Atlanta, GA, Nov. 2001.

[19] P. Magnusson, F. Dahlgren, H. G. amd M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, and P. Stenström. SimICS/sun4m: A Virtual Workstation. In *Proc. of the 1998 USENIX Annual Technical Conference*, pages 119–130, New Orleans, LA, June 1998.

[20] R. Min and Y. Hu. Performance of Large Physically Indexed Caches by Decoupling Memory Addresses from Cache Addresses. *IEEE Transactions on Computers*, 50(11):1191–1201, Nov. 2001.

[21] P. Ranganathan, S. Adve, and N. Jouppi. Reconfigurable Caches and their Applications to Media Processing. In *Proc. of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 214–224, Vancouver, Canada, June 2000.

[22] G. Rivera and C. Tseng. Data Transformations for Eliminating Conflict Misses. In *Proc. of ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation (PLDI'1998)*, pages 85–96, Montreal, Canada, June 1998.

[23] A. Seznec. A Case for Two-Way Skewed-Associative Caches. In *Proc. of the 20th Annual International Symposium on Computer Architecture (ISCA-20)*, pages 169–178, San Diego, CA, May 1993.

[24] T. Sherwood, B. Calder, and J. Emer. Reducing Cache Misses using Hardware and Software Page Placement. In *Proc. of the 1999 ACM International Conference on Supercomputing (ICS'99)*, pages 155–164, Rhodes, Greece, June 1999.

[25] P. Shivakumar and N. Jouppi. CACT 3.0: An Integrated Cache Timing, Power and Area Model. Technical report, WRL Research Report, Aug. 2001.

[26] N. Topham, A. González, and J. González. The Design and Performance of a Conflict-Avoiding Cache. In *Proc. of the 30th Annual International Symposium on Microarchitecture (MICRO-30)*, pages 71–80, Research Triangle Park, NC, Nov. 1997.