A Comparison of Online Superpage Promotion Mechanisms

Zhen Fang and Lixin Zhang

{zfang | lizhang}@cs.utah.edu
http://www.cs.utah.edu/impulse [1]

UUCS-99-021

Department of Computer Science
3190 Merrill Engineering Building
University of Utah
Salt Lake City, UT 84112

December 2, 1999

Abstract

The amount of data that a typical translation lookaside buffer (TLB) can map has not kept pace with the growth in cache sizes and application footprints. As a result, the cost of handling TLB misses limits the performance of an increasing number of applications. The use of *superpages*, multiple adjacent virtual memory pages that can be mapped with a single TLB entry, extends a TLB's reach without significantly increasing its size or cost. The difficulty of identifying what sets of pages should be promoted to superpages combined with the overhead of performing these promotions restricts superpage use almost exclusively to wired system data structures. Previous studies have shown that simple online policies that decide to create superpages dynamically can be effective in reducing TLB penalties.

In this paper we analyze the performance of online superpage promotion for nine benchmarks on a simulated HP PA-RISC system running a BSD Unix kernel. We extend previous work in two ways. First, we study the impact of creating superpages dynamically by *remapping* pages at the memory controller instead of copying pages to make them contiguous. The use of such a hardware mechanism affects the choice between two previously described superpage promotion policies. Previous work has shown that an online approximation to a competitive policy is the best choice. Our results show that having hardware support makes a greedy policy perform equally well. Second, we use execution-driven simulation, whereas previous studies have used trace-driven simulation. Our results show that the differences in accuracy are significant, especially when studying complex interactions between operating systems and modern architectures.

Keywords: memory architecture, TLB performance, competitive algorithms, simulation
Technical Areas: architecture, memory systems, operating systems

---

[1]This effort was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number F30602-98-1-0101 and DARPA Order Numbers F393/00-01 and F376/00. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official polices or endorsements, either express or implied, of DARPA, AFRL, or the US Government.

1 Introduction

The translation lookaside buffers (TLBs) on most modern processors support *superpages*: groups of contiguous virtual memory pages that can be mapped with a single TLB entry [8, 16, 27]. Using superpages makes more efficient use of the TLB, but the physical pages that back a superpage must be contiguous and properly aligned. Dynamically coalescing smaller pages into a superpage thus requires that all the pages be reserved *a priori*, be coincidentally adjacent and aligned, or be copied so that they become contiguous. The overhead costs of promoting superpages by copying include the direct costs of copying the pages and changing the mappings. Other indirect costs are also important, such as the increased number of instructions executed on each TLB miss (due to the new decision-making code in the miss handler) and the increased contention in the cache hierarchy (due to the code and data used in the promotion process). When deciding whether to create superpages, these costs must be balanced against the improvement in TLB performance.

Romer *et al.* [22] study several different policies for dynamically creating superpages. Their trace-driven simulations and analysis show how a policy that balances potential performance benefits and promotion overheads can improve performance in some TLB-bound applications by about 50%. Our work extends theirs by measuring the added performance benefit, as well as the effect on the choice of policy, of using hardware support at the memory system to make creating superpages cheaper.

The hardware that we model is the Impulse Memory Controller [28], which helps create superpages *without copying* by adding another level in the memory hierarchy at the memory controller. In Impulse, superpages are built through reampping.

Our research shows that combining the work of Romer *et al.* and the Impulse technology changes the tradeoffs in designing an online superpage promotion policy. Romer *et al.* find that a competitive promotion policy that tries to balance the overheads of creating superpages with their benefits achieves the best average performance. Our experiments confirm this result when promotion is accomplished via copying, but we find that the use of a more aggressive promotion policy that promotes superpages as soon all of their constituent sub-pages have been touched performs best when coupled with a remapping-based promotion mechanism. We further find that the performance of Romer's competitive promotion policy can be improved by tuning it to create superpages more aggressively, even when copying is employed to promote pages. In addition, by using a detailed execution-driven simulator, we identify the impact of several performance factors not covered by Romer *et al.*'s trace-based study, such as the detrimental effects of the cache pollution induced by copying. Finally, we find that online superpage promotion achieves performance comparable to the hand-coded superpage promotion mechanism employed by Swanson *et al.* [28]

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 explains the two methods used to create superpages, along with the two policies investigated for promoting superpages at run time. Section 4 describes our simulation environment and benchmark suite, and Section 5 gives the experimental methodology and the results of our study. Section 6 summarizes our conclusions and discusses future work.

2 Related Work

Competitive algorithms perform cost/benefit analyses dynamically to make online decisions that guarantee performance within a constant factor of an optimal offline algorithm. Romer *et al.* [22] adapt this approach to TLB management, employing a competitive strategy to decide when to perform dynamic superpage promotion. They also investigate online software policies for dynamically remapping pages to improve cache performance [3, 21]. Competitive algorithms have been used to help increase the efficiency of other operating system functions and resources, including paging [24], synchronization [14], and file cache management [5].

Chen *et al.* [7] report on the performance effects of various TLB organizations and sizes. Their results indicate that the most important factor for minimizing the overhead induced by TLB misses is *reach*, the amount of address space that the TLB can map at any instant in time. Even though the SPEC benchmarks they study have relatively small memory requirements, they find that TLB misses increase the effective CPI (cycles per instruction) by up to a factor of five. Jacob and Mudge [13] compare five virtual memory designs, including combinations of hierarchical and inverted page tables for both hardware-managed and software-managed TLBs. They find that large TLBs are necessary for good performance, and that TLB miss handling overhead accounts for much of the memory-management overhead. They also project that the individual cost of TLB miss traps will increase in future microprocessors.

Proposed solutions to this growing TLB performance bottleneck range from changing the TLB structure to retain more of the working set (e.g., multi-level TLB hierarchies [1, 9]), to implementing better management policies (in software [12] or hardware [11]), to masking TLB miss latency by prefetching entries (in software [2] or hardware [23]).

All of these approaches can be improved by exploiting superpages. Most TLBs now support superpages, and have for several years [16, 27], but more research is needed into how best to make general use of this capability. Chen *et al.* [7] suggest the possibility of using variable page sizes to improve TLB reach, but do not explore the implications of their use. Khalidi *et al* [15] and Mogul [17] discuss benefits of systems that support superpages, advocating static allocation via compiler or programmer hints. Talluri *et al.* [18] report many of the difficulties attendant upon general utilization of superpages, most of which result from the requirement that superpages map physical memory regions that are contiguous and aligned.

On a system with four-kilobyte base pages, Talluri *et al.* [19] find that judicious use of 32-kilobyte superpages can reduce the impact of TLB misses on CPI by as much as a factor of eight. Exclusive use of the larger pages increases application working sets by as much as 60%, which can lead to inefficient use of main memory. However, mixing both page sizes limits this bloat to around 10%, and allowing the TLB to map superpages without requiring that all the underlying base pages be present (partial superpages) eliminates the problem altogether.

Swanson *et al.* [28] build superpages through a remapping approach with the Impulse hardware support. They statically modify applications to create superpages via system calls. Their simulation results demonstrate a two-fold increase in TLB reach and a 5%-20% improvement in the performance of some SPECint95 and Splash2 applications with medium to high TLB miss rates. Our work is based on the same hardware as theirs, while with different kernel modifications.

3 Research Background

We measure the impact of combining no-copy superpage promotion with the two online promotion algorithms proposed by Romer *et al.* [22]. The methodological differences between this study and Romer *et al.*'s are described in Section 4. In this section we describe the promotion policies we study, and then we briefly discuss the hardware used by Swanson *et al.* to support no-copy superpage promotion.

3.1 Promotion Algorithms

We evaluate two of the online superpage promotion policies developed by Romer *et al.* [22], **asap** and **approx-online**. **asap** is a greedy policy that promotes a superpage as soon as all of its component pages have been referenced. The algorithm does not consider reference frequency for the potential superpages, which minimizes bookkeeping overhead. The price for this simplicity is that the **asap** policy may build superpages that are rarely referenced later, in which case the benefits of these superpages would not offset the costs of building them.

approx-online uses a competitive strategy to determine when superpages should be coalesced. If a superpage P accrues many misses, we expect that it will be referenced again in the future, and that promoting it will prevent many future TLB misses. Such promotions effectively *prefetch* the translations for the non-resident base pages in the new superpage. To track this reference information, the **approx-online** algorithm maintains a counter *P.prefetch* for each potential superpage P. On a TLB miss, the policy increments the counters for all potential superpages that would have prevented the miss. In other words, on a miss to base page p, *P.prefetch* is incremented for each potential superpage P that contains the referenced page p and at least one current TLB entry. When the miss charges for a superpage P_0 reach a pre-set threshold for superpages of size $P_0.size$, the pages that constitute P_0 are promoted into a superpage.

The miss charges of a potential superpage should reflect the number of misses that earlier promotion would have eliminated. So, when page P_0 is created, the prefetch counters of all larger superpages containing it must be adjusted to reflect the now-diminished benefits of their promotion. For all superpages P that contain P_0, *P.prefetch* is decremented by $P_0.prefetch$, since whenever $P_0.prefetch$ was incremented, *P.prefetch* was, too.

Consider a system for which the base page size is 4096 bytes, superpages are built using powers of two base pages, and the largest superpage contains 64 base pages. **approx-online** behaves as follows. Let (va, n) denote a superpage starting at virtual page number va and composed of 2^n base pages. Assume that the application incurs a TLB miss at virtual address $0x60005023$ and the TLB contains a translation for virtual base page $0x60006$ but has no translation for $0x60004$. The prefetch counters for potential superpages containing the virtual page $0x60005$ and the TLB entry $0x60006$ are incremented by one. These superpages are $(0x60004, 2)$, $(0x60000, 3)$, $(0x60000, 4)$, $(0x60000, 5)$, and $(0x60000, 6)$. **approx-online** then finds the largest potential superpage that has reached its promotion threshold and creates it. For example, if $(0x60004, 2).prefetch$ has reached the threshold for superpages of size four, the operating system promotes the superpage and decrements the prefetch counters for the containing superpages (i.e., $(0x60000, 3)$ through $(0x60000, 6)$) by the value of $(0x60004, 2).prefetch$.

A simple but inefficient way to compute prefetch charges is to scan the TLB on a miss to page p, and check whether some potential superpage contains both p and at least one current TLB entry. To avoid the overhead of scanning the contents of the TLB on each miss, Romer *et al.* propose tracking an additional value, $P.tlbcount$, for each superpage P. This counter indicates how many of the superpage's component subpages (one power of two smaller in size) are currently in the TLB or contain TLB entries. $P.tlbcount$ takes on one of four values: -1, 0, 1, or 2. If P is a superpage or part of a larger superpage that has been promoted, then $P.tlbcount = -1$. Otherwise, let P_1 and P_2 be the two component subpages of P. $P.tlbcount$ is 0, 1, or 2, depending on how many of its component subpages are in the TLB. This strategy allows the prefetch charges to be updated efficiently on a TLB miss. [2]

Note that **approx-online** is a simplification of the more complex **online** policy [22], which not only charges a TLB miss to the potential superpages containing p, but also blames the eviction of p on the fact that unrelated pages were not coalesced into superpages. **online** thus tries to coalesce other superpages (those that do not contain p. Romer [21] shows that **approx-online** is as effective as **online**, but has much lower bookkeeping overhead.

[2] The kernel on HP PA-RISC only decides which pagetable entry to insert to TLB. It has no control over which TLB entry is going to be evicted upon a TLB insert instruction, nor does it have knowledge of which TLB entry has become the victim. To maintain tlbcount, however, the TLB miss handler needs information about the evicted virtual address and the size of the victim page. In our implementation, we massaged HP PA-RISC instruction *idtlba* so that it returns the victim TLB entry information in a particular control register, which is accessible from the kernel. Though this causes a modification of instruction *idtlba*, we believe it is appropriate in a research circumstance.

The choice of threshold value used to decide when to promote a set of pages to a superpage is critical to the effectiveness of **approx-online**. The ideal threshold is small enough for useful superpages to be promoted early, thereby eliminating future TLB misses, but large enough so that the cost of promotion does not dominate TLB overhead. We quantify this tradeoff in Section 5.1.

Romer *et al.* choose an appropriate threshold value by using a competitive strategy — a collection of pages is promoted to a superpage as soon as it has suffered enough TLB misses to pay for the cost of the promotion. Theoretically, the promotion threshold should be the promotion cost divided by the TLB miss penalty. For example, if the average TLB miss penalty is 40 cycles and copying two base pages to a contiguous two-page superpage costs 16,000 cycles, the threshold for superpage promotion would be 400 (16,000 divided by 40). Romer [21] proves that a system employing **approx-online** can suffer no more than twice the combined TLB miss and superpage promotion overheads that would be incurred by a system employing an optimal offline promotion algorithm. Although the theoretical threshold bounds worst-case behavior to an acceptable level, smaller thresholds tend to work better in practice. In our experiments, we therefore run **approx-online** with a range of different threshold values.

3.2 Promotion via Remapping

No-copy superpage creation relies on hardware support provided by the Impulse memory controller [28]. Such hardware provides an extra level of address remapping at the memory: unused physical addresses are remapped into "real" physical addresses. In keeping with Impulse terminology, we refer to these remapped addresses as *shadow addresses*, or the *shadow address space*. From the point of view of the processor and OS memory management system, shadow addresses are used in place of real physical addresses. Shadow addresses will be inserted into the TLB as mappings for virtual addresses, they will appear as physical tags on cache lines, and they will appear on the memory bus when cache misses occur. The existence of shadow memory is completely transparent to user programs and the processor. It is the Impulse memory controller that identifies a shadow address and translates it to physical address through the shadow-to-physical memory controller pagetable. The operating system is responsible for managing this new level of address translation, but the memory controller maintains its own page tables for shadow memory mappings. Building superpages

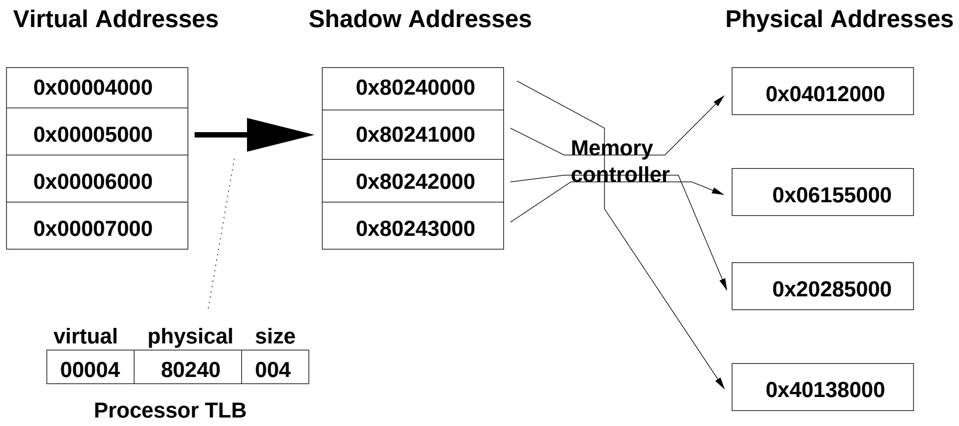


Figure 1: Detailed Example of Using Shadow Physical Regions

from base pages that are not physically contiguous can be accomplished by simply remapping the virtual pages to contiguous, aligned shadow pages. The memory controller then maps the shadow pages to the original physical pages. There is a much larger TLB in the memory controller which we call MTLB. MTLB works in the same way as the processor TLB except that it translates shadow addresses to real physical addresses. MTLB translation is not on the virtual address resolution critical path, which means it can be built to a fairly large size.

Our HPCA99 paper [6] describes the hardware design of Impulse memory controller. The shadow address space is divided into seven contiguous regions. A hardware shadow descriptor register is introduced for each shadow region to speed up backend memory access request. The shadow descriptor contains the start addresses of the shadow region and of the shadow-to-physical memory controller pagetable, and the size of the pagetable.

Figure 1 illustrates how superpage mapping works on Impulse. For simplicity, internal details of Impulse memory controller are not shown. Interested readers can refer to Carter *et al.* [6] Suppose 2G bytes are reserved for real physical memory space. Physical address starting from 0x8000000, therefore, is unused and are interpreted as shadow address space. In this example, the OS has mapped a contiguous 16KB virtual address range to a single shadow superpage at "physical" page frame $0x80240$. Upon a reference to a virtual address within this superpage range, virtual to physical address translation is executed in the usual way at the processor TLB. The processor places the obtained "physical" address onto the system memory bus to get the data from memory controller. This "physical" address is in fact in the shadow address space. The memory controller detects that this "physical" address needs to be retranslated using its local shadow-to-physical translation MTLB/pagetable. The seven shadow descriptors are searched for a match, and the selected descriptor provides the needed information. In the example in Figure 1, the processor translates an access to virtual address $0x00004080$ to shadow physical address $0x80240080$, which the controller, in turn, translates to real physical address $0x40138080$. In this case, the memory controller can determine if an address on the system memory bus is a real physical address or a shadow address by simply checking the highest bit, and proceed accordingly.

Impulse has shown its effectiveness in improving TLB performance with superpages built through system calls. For this part of work the readers can refer to our ISCA paper [28]. That's kind of static approach, which in fact requires compiler support to be applied to existant applications, or the programmer has to insert hard coded system calls. The work described in this paper does not recompile or massage the programmes with system calls. Instead, superpages are built dynamically as the programs go no.

Currently we do not implement superpage tear-down for the online superpage promotion research. Consequently there is one potential limit to Impulse based superpage promotion, *i.e.*, the limit on the number of shadow descriptors. It is possible that at some time the kernel wants to promote more superpages but there is not new shadow descriptor available. Since the size of shadow address range that a shadow descriptor can map is decided by the size of the corresponding shadow-to-physical pagetable, we could make this pagetable arbitrarily large, which in fact makes this potential restriction moot.

4 Simulation Environment

Our studies use the execution-driven simulator Paint [26], which models a variation of a 240 MHz, single-issue, HP PA-RISC 1.1 processor running a BSD-based microkernel, and a 120 MHz HP Runway bus. The 64-kilobyte L1 data cache is non-blocking, single-cycle, write-back, write-around, virtually indexed, physically tagged, and direct-mapped with 32-byte lines. The 512-kilobyte L2 data cache is non-blocking, write-allocate, write-back, physically indexed and tagged, and 2-way set-associative with 128-byte lines. Instruction caching is assumed to be perfect. The split-transaction bus multiplexes addresses and data. The memory system has a total memory latency of 60 cycles. The simulated remapping memory controller is based on the HP controller [10] used in servers and high-end workstations. The MTLB is configured at 1024 entries.

The TLB holds both instruction and data translations. It is fully associative, employs a not-recently-used replacement policy, and returns a translation in one cycle. In addition to the main TLB, a single-entry micro-ITLB holds the most recent instruction translation. The base page size is 4096 bytes. Superpages are built in power-of-two multiples of the base page size, and the biggest superpage that the TLB can map contains 1024 base pages. Kernel code and data structures are mapped using a single block-TLB entry that is not subject to replacement. Our results include measurements for two TLBs, a small one with only 32 entries, and a larger one with 128 entries, which lets us examine how scaling the TLB affects the applications that we study. The smaller TLB size also is close to the TLB size that Romer *et al.* used in their study. They generate their traces using ATOM [25] on a DEC Alpha 3000/700 running DEC OSF/1 2.1, a system that contains a 225 MHz Alpha 21064 processor with a 32-entry DTLB and an 8 entry ITLB, a 2-megabyte offchip cache, and 160 megabytes of main memory.

4.1 Microbenchmark

When comparing online superpage promotion schemes, an important performance factor is the number of TLB misses that must be eliminated per promotion to amortize the cost of implementing the promotion algorithm. This cost includes the extra time spent in the TLB miss handler determining when to coalesce pages, plus the time spent performing the actual promotions (via either copying or remapping). To explore the cost/performance tradeoffs for each approach, we run a synthetic microbenchark consisting of a loop that touches 4096 different base pages for a configurable number of iterations:

```
char A[4096][4096];

for (j = 0; j < test_iterations; j++)
  for (i = 0; i < 4096; i++)
    sum += A[i][j];
```

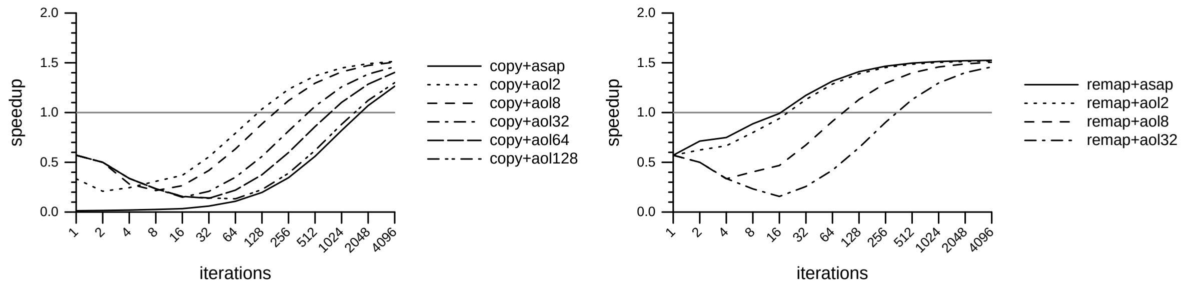
Without superpages, each memory access in the synthetic microbenchmark suffers a TLB miss. However, since every page is touched repeatedly, superpages can be used to reduce the aggregate cost of these TLB misses. This experiment determines the break-even point for each approach, i.e., the number of iterations at which the benefit of creating superpages exceeds the cost of doing so.

4.2 Benchmark Suite

To evaluate the different superpage promotion approaches on real-world problems, we use nine programs from a mix of sources. Our benchmark suite includes three SPEC95 benchmarks (`compress`, `gcc`, and `vortex`), three image processing benchmarks (`raytrace`, `rotate`, and `filter`), two scientific benchmarks (`cga` and `matmul`), and one SPLASH-2 benchmark (`radix`) [29]. All benchmarks were compiled with gcc 2.7.2 and optimization level "-O2".

`Compress` is the SPEC95 data compression program run on an input of one million characters. Note that the default SPEC95 implementation of `compress` executes the compression algorithm 25 times, whereas the version of `compress` employed by Romer *et al.* appears to have executed the algorithm only once. Running a different number of iterations does not affect the relative performance of the various superpage promotion algorithms, but it does make the raw numbers (e.g., the number of TLB misses) incomparable. `Gcc` is the `cc1` pass of the version 2.5.3 gcc compiler (for SPARC architectures) used to compile the 306-kilobyte file "1cp-dec1.c". `Vortex` is an object-oriented database program measured with the SPEC95 "test" input. `Radix` is an integer radix sort program (based on the method of Blelloch *et al.* [4]) run with the SPLASH-2 default arguments. `Cga` is the NPB2.3 benchmark suite's class A conjugate gradient benchmark, which performs a sparse matrix-vector product. `Matmul` is the conventional, tiled version of dense matrix-matrix multiplication run on 1024×1024 matrices with 32×32 tiles. `Raytrace` is an interactive isosurfacing volume renderer whose input is a 1024×1024×1024 volume; its implementation is based on work done by Parker et al. [20] `Filter` performs an order-129 binomial filter on a 32×1024 color image. `Rotate` turns a 1024×1024 color image clockwise through one radian.

Two of these benchmarks, `gcc` and `compress`, are also included in Romer *et al.*'s benchmark suite, although we use SPEC95 versions, whereas they use SPEC92 versions. We do not use the other SPEC92 applications from that study, due to the benchmarks' obsolescence. Several of Romer *et al.*'s remaining



(a) *copying* (b) *remapping*

Figure 2: Microbenchmark performance for 4096 base pages. aolxx: approx-online with threshold xx

benchmarks are based on tools used in the research environment at the University of Washington, and are not readily available to us.

5 Results

The performance results presented here are obtained through complete simulation of the benchmarks, including both kernel and application time, the direct overhead of implementing the superpage promotion algorithms, and the resulting effects on memory system. We first present the results of our microbenchmark experiments exploring the break-even points for each of the superpage promotion policies and mechanisms, then we present comparative performance results for our application benchmark suite.

5.1 Microbenchmark Results

Figure 2(a) and Figure 2(b) illustrate our microbenchmark results for online superpage promotion via copying and remapping, respectively. The microbenchmark's working set is sufficiently large that performance is the same for both a 32-entry and a 128-entry TLB. The x axes indicate the number of times the microbenchmark's main loop is repeated, i.e., the number of times each page is referenced. The execution times include kernel startup times. These graphs emphasize the performance differences among the **asap** and **approx-online** policies.

For instance, copying-based **asap** only becomes profitable after each page is touched more than one thousand times, whereas the same policy breaks even after only sixteen references per page when remapping is used. Copying performs much worse when pages are seldom referenced: execution time is 94 times slower than the baseline when the **asap** policy is employed but each page is touched only once. This causes all of the pages to be promoted (copied), even though they are never accessed again. In contrast, the remapping promotion mechanism delivers more robust performance, and both remap+asap and remap+aol2 result in a slowdown of less than a factor of two when the microbenchmark touches each page between one and eight times. The cost of a TLB miss increases from around 30 cycles in the baseline to 700 cycles for remapping **asap**, and to 88,000 cycles for copying **asap**. In this example, **asap** suffers only one TLB miss per subpage before promoting the set of pages to a superpage, so this average TLB miss time includes the cost promotion.

Performances for all the **approx-online** configurations suffer when the threshold is larger than the number of references to each page. The additional overheads in the TLB miss handler dominate the microbenchmark execution time. The number of references required for this policy to be profitable increases with the threshold. For copying-based promotion and thresholds of two and eight, the number of references per page must be 64 and 16 times the threshold, respectively. For remapping and for copying with thresholds of 32 or more, **approx-online** improves performance when the number of references per page is at least eight times the threshold. The TLB miss penalty goes from about 30 cycles in the baseline to 800 cycles for remapping **approx-online** and 5800 cycles for copying **approx-online**.

In general, the remapping-based policies deliver performance benefits at much lower thresholds, and all policies and mechanisms perform well when pages are referenced at least 2048 times. **asap** exhibits the largest variation in performance, delivering the best speedups when superpages are built via remapping, and the worst slowdowns when superpages are built via copying.

5.2 Full-Application Results

Table 1 lists the characteristics of the baseline run of each benchmark, where no superpage promotion occurs. These benchmarks demonstrate varying sensitivity to TLB performance: on the system with the smaller TLB, between 14% and 77% of their execution time is spent in the data TLB miss handler. The percentage of time spent handling TLB misses falls to between less than 1% and 58% on the system with a 128-entry TLB.

Benchmark	Total cycles (millions)	Loads (thousands)	Stores (thousands)	Cache misses (thousands)	DTLB misses (thousands)	DTLB miss time
32-entry TLB						
compress	7369	928492	434950	7117	63878	60.06%
gcc	1196	248426	126507	545	3305	15.66%
vortex	1372	274983	186154	740	5699	23.01%
radix	544	35541	10722	544	1960	20.00%
cga	3059	432698	7514	18572	7384	14.97%
matmul	9609	824743	144160	13136	138585	77.01%
raytrace	1311	86628	9894	2290	8459	33.98%
filter	603	131340	67961	537	4227	37.46%
rotate	483	32358	26149	3016	3617	53.64%
128-entry TLB						
compress	3983	672041	370687	6846	22	0.06%
gcc	883	220213	120795	508	170	1.20%
vortex	912	236734	177643	684	616	4.05%
radix	520	33706	10261	543	1502	16.28%
cga	2704	431972	7332	18525	786	3.72%
matmul	2386	269850	4839	12147	136	0.33%
raytrace	1303	86042	9776	2283	8399	33.89%
filter	578	129410	67476	538	3745	34.70%
rotate	483	32354	26148	3016	3617	53.65%

Table 1: Characteristics of each baseline run.

Figures 3 and 4 show the normalized speedups of the different combinations of promotion policies (**asap** and **approx-online**) and mechanisms (*remapping* and *copying*) compared to the baseline instance of each benchmark. We can make two orthogonal comparisons from these figures: *remapping* versus *copying*, and **asap** versus **approx-online**. The two dark bars on the left of the figure for each benchmark illustrate results for remapping-based **asap** (*remap+asap*) and copying-based **asap** (*copy+asap*). The two light bars on the right represent the best results from remapping-based **approx-online** (*remap+aol*) and copying-based **approx-online** (*copy+asap*). The numbers appended to the **approx-online** labels indicate the optimal thresholds for initiating the promotion of two 4-kilobyte base pages to one 8-kilobyte superpage. When the base threshold is eight (e.g., for *copy+aol8*), the corresponding threshold for promoting two 8-kilobyte superpages to a 16-kilobyte superpage is $2 \times 8 = 16$, and so forth. Online superpage promotion can improve performance by up to a factor of 4.5 (on `matmul`, our tiled matrix multiply routine). However, it also can decrease performance by up to 35% (when using the copying version of **asap** on `rotate`).

We first compare the two promotion algorithms, **asap** and **approx-online**, using the results from Figures 3 and 4. The relative performance of the two algorithms is strongly influenced by the choice of promotion mechanism, *remapping* or *copying*. Using a remapping promotion mechanism, **asap** slightly outperforms **approx-online** in the average case. It exceeds the performance of **approx-online** in thirteen of the eighteen

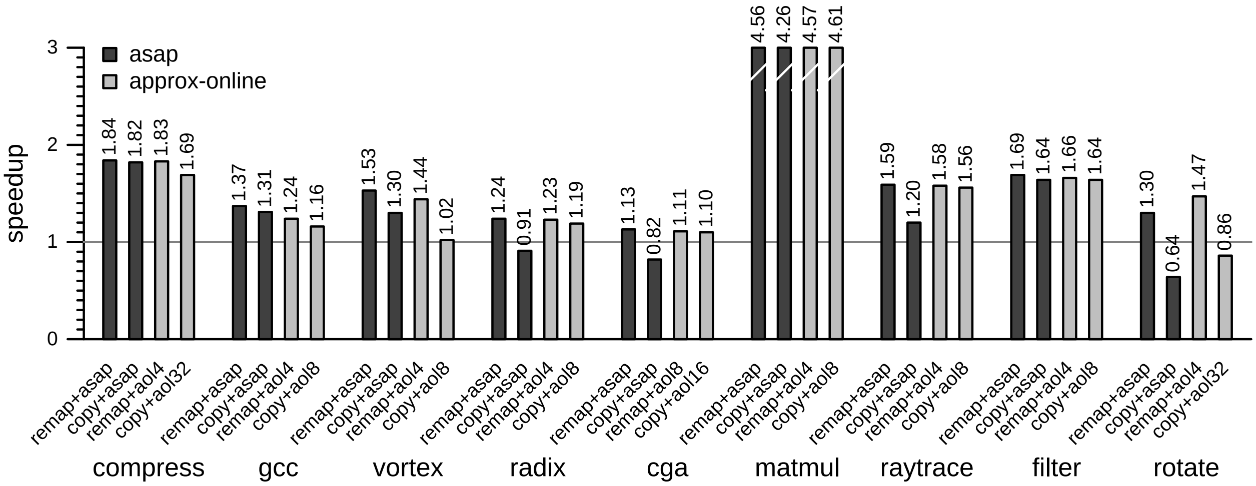


Figure 3: Normalized speedups for our two superpage promotion mechanisms for each of two promotion policies on a system with a 32-entry TLB. This graph shows the best performance for any policy configuration in each category.

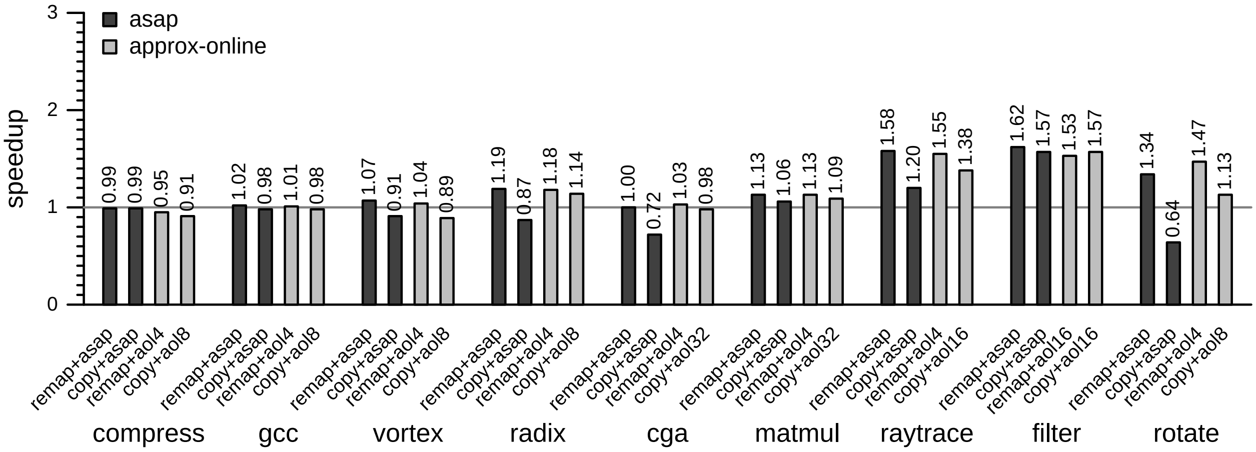


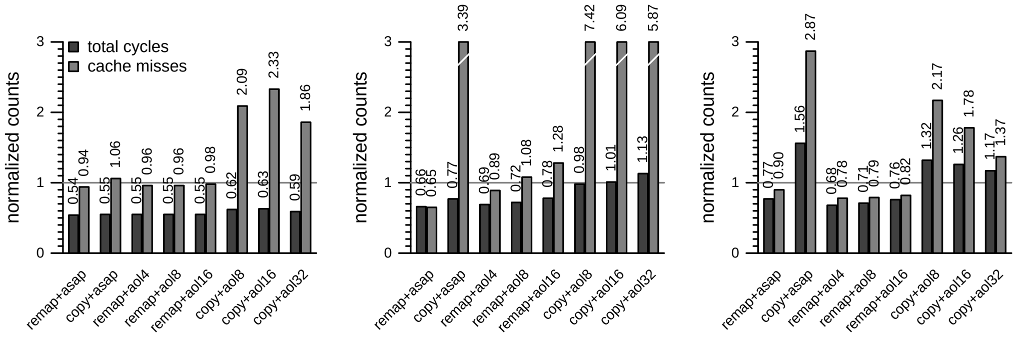
Figure 4: Normalized speedups for our two superpage promotion mechanisms for each of two promotion policies on a system with a 128-entry TLB. These results give the best performance for any policy configuration in each category.

experiments, and trails the performance of **approx-online** in only four cases. The differences in performance range from *asap+remap* outperforming *aol+remap* by 13% for `gcc` with a 32-entry TLB, to *aol+remap* outperforming *asap+remap* by 17% for `rotate` with a 32-entry TLB. In general, however, the performance differences between the two policies are small. Considering that the results we present for **approx-online** are for the optimal threshold in each case (rather than some fixed system-wide threshold) and that **asap** is a much simpler policy to implement, we believe that the **asap** policy is the best choice when remapping is an option.

The results change noticeably when we employ a *copying* promotion mechanism. In this case, **approx-online** outperforms **asap** in ten of the eighteen experiments, while the policies performs identically in three of the other eight cases. The magnitude of the disparity between **approx-online** and **asap** results is also dramatically larger. The differences in performance range from **asap** outperforming **approx-online** by 28% for `vortex` with a 32-entry TLB, to **approx-online** outperforming **asap** by 36% for `raytrace` with a 32-entry TLB. Overall, our results confirm those of Romer *et al.*: the best promotion policy to use when creating superpages via copying is **approx-online**.

The relative performance of the **asap** and **approx-online** promotion policies changes when we employ different remapping mechanisms because **asap** tends to create superpages more aggressively than **approx-online**. The design assumption underlying the **approx-online** algorithm (and the reason that it performs better than **asap** when copying is used) is that superpages should not be created until the opportunity cost of TLB misses equals the cost of creating the superpages. Given that remapping has a much lower cost for creating superpages than copying, it is not surprising that the more aggressive **asap** policy performs relatively better than **approx-online** when combined with the remapping mechanism (and vice versa).

Note that we present numbers for an *optimal* **approx-online** policy for each benchmark — one that uses the promotion threshold that we observe to deliver the best performance for that benchmark. This optimal threshold ranges from four (for *remap+aol* on most applications) to 32 (for *copy+aol* on two of the applications). The wrong choice of threshold can hurt performance, as demonstrated by Figures 5 and 6. For a 32-entry TLB, the difference between the performances of *copy+aol* with the best and worst threshold



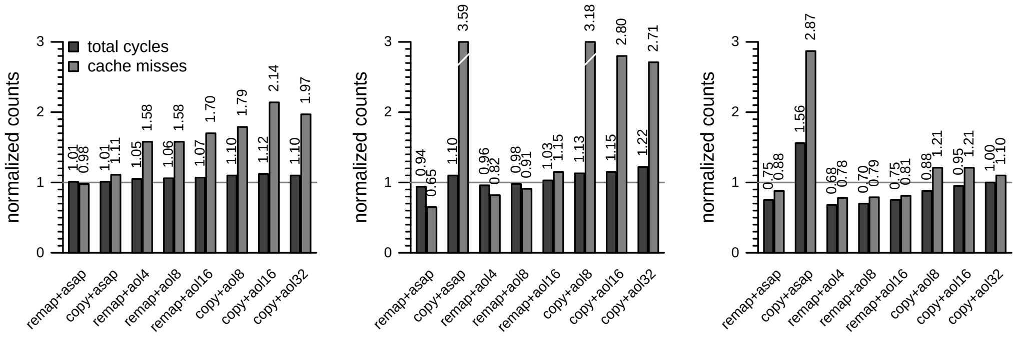
(a) *compress* (b) *vortex* (c) *rotate*

Figure 5: Performance details for selected benchmarks on the system with a 32-entry TLB.

choices between eight and 32 is as large as 23% for `cga` (1.14 versus 0.91). The performance spread with a 128-entry TLB is as large as 45% for `matmul`. The magnitude of the impact of proper threshold selection is atypically large for these two programs, however. The choice of a fixed compromise threshold, e.g., 16, reduces the average performance of **approx-online** by roughly 4%. In their earlier study, Romer *et al.* employ a fixed threshold of 100, which we find to be far too large. This issue will be discussed in more detail in Section 5.3.

When we compare the two superpage creation mechanisms, *remapping* and *copying*, *remapping* is the clear winner, but by highly varying margins. The differences in performance between the best overall remapping-based algorithm (*asap+remap*) and the best copying-based algorithm (*aonline+copying*) is as large as 51% in the case of `vortex` on a 32-entry TLB. Overall, *asap+remap* outperforms *aonline+copying* by more than 15% in seven of the eighteen experiments, although the margin is less than 5% in all but one of the other tests.

Figure 5 and Figure 6 illustrate one of the secondary reasons that the remapping mechanism outperforms the copying mechanism — cache pollution. These figures show the relative numbers of cache misses suffered by the benchmarks when superpage promotion is enabled (versus the baseline execution) for three of our benchmarks. The dark gray bars indicate the relative execution time of the benchmarks with superpage promotion enabled, while the medium gray bars indicate the relative number of cache misses. A shorter bar thus indicates improved performance or a reduction in the number of cache misses. For both configurations



(a) *compress* (b) *vortex* (c) *rotate*

Figure 6: Performance details for selected benchmarks on the system with a 128-entry TLB.

of TLBs, the number of cache misses is big for copying based promotion for each of the three benchmarks. Take `compress` for an example. For 32-entry TLBs, superpage promotion improves the performance of all policies and both approaches. However, the number of cache misses grows substantially for the copying variant of **approx-online** on `compress`. A closer examination of `compress` reveals that the high DTLB miss rate (60%) and resulting large number of superpage promotions leads to significant cache pollution as the superpages are created. Nevertheless, due to the dramatic reduction in TLB misses, the performance of `compress` improves by roughly 40%, despite the large increase in L1 cache misses. For `vortex` and `rotate`, the cost of page copying and cache pollution is not amortized enough so that almost all copying based promotions fail to benefit at 32-entry TLBs.

5.3 Discussion

Romer *et al.* show that **approx-online** is generally superior to **asap** when copying is used. When remapping is used to build superpages, though, we find that the reverse is true. Using Impulse-style remapping results in larger speedups and consumes much less physical memory. Since superpage promotion is cheaper with a remapping mechanism, policies are much less likely to promote pages too aggressively.

Romer *et al.*'s traced-based simulation models no cache interference between the application and the TLB miss handler; instead, that study assumes that each superpage promotion costs a total of 3000 cycles per kilobyte copied [22]. Table 2 shows our measured lower bounds of the per-kilobyte cost (in CPU cycles)

benchmark	cycles per 1KByte promoted	average cache hit ratio	baseline cache hit ratio
rotation	16,316	82.12%	87.47%
compress	5,969	98.88%	99.41%
raytrace	5,186	96.18%	97.61%
radix	5,503	97.67%	98.81%

Table 2: Comparison of cache performances with average costs in cycles for approx-online superpage promotion via copying.

to promote pages by copying for four representative benchmarks. We measure this bound by subtracting the execution time of *aol+remap* from that of *aol+copy* and dividing by the number of kilobytes copied. For our simulation platform and benchmark suite, superpage promotion costs vary with an application's cache performance. For `compress`, `raytrace`, and `radix`, all of which have cache hit ratios in excess of 96%, superpage promotion is about twice as expensive as Romer *et al.* assumed. For `rotate`, which has a cache hit ratio of only 82%, superpage promotion costs more than five times the cost charged in the trace-driven study.

We also find that even when copying is used to promote pages, **approx-online** performs better with a more aggressive (lower) threshold than is used by Romer *et al.* Specifically, the optimal threshold in our experiments varies from 8 to 32, while their study uses a fixed threshold of 100. This difference in thresholds has a significant impact on performance. For example, when we run the `gcc` benchmark using a threshold of 128, **approx-online** with copying *slowed* performance by 4.3% with a 32-entry TLB, which is close to the 0.9% slowdown reported in Romer *et al.*'s study – the difference is likely to be caused by the higher per-kilobyte promotion costs we measured. In contrast, when we run **approx-online** with copying using the optimal threshold of 8, performance is *improved* by 16%. Given that our measured cost of promoting pages is much higher than the 3000 cycles estimated in their study, we expected our optimal thresholds to be higher, not lower than theirs. In general, we find that to achieve their maximum potential, even the copying-based promotion algorithms need to be much more aggressive about creating superpages than was suggested by the earlier study.

Finally, we can compare the results of our experiments using online superpage promotion against those reported by Swanson *et al.* using a static promotion policy and a remapping mechanism. For their study [28], the authors hand annotate their benchmarks to insert system calls at the start of execution or during

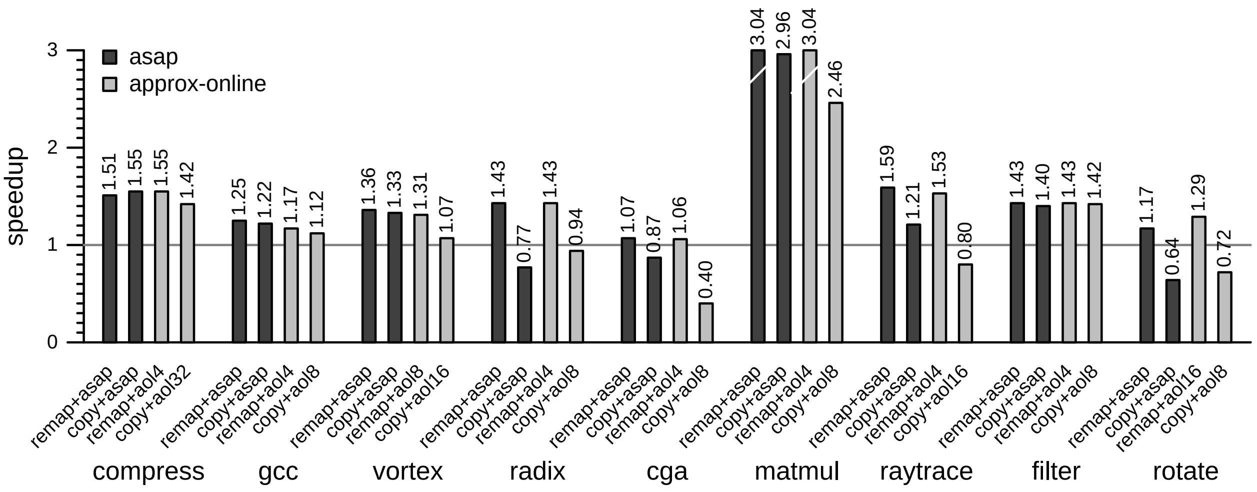


Figure 7: Normalized speedups as *tlbcount* checking is taken off. This graph shows the best performance for any policy configuration in each category with a 32-entry TLB.

`malloc()` operations to request that a particular region of virtual memory be made into a superpage. They find that static superpage promotion coupled with remapping improves the performance of `compress` by approximately 5%, `gcc` by approximately 2%, `radix` by approximately 20%, and `vortex` by approximately 10% for a 128-entry processor TLB. Using a dynamic superpage promotion algorithm that automatically selects pages for promotion without user input (*asap+remap*), the performance of `compress` drops by 1%, while the performances of `gcc`, `radix`, and `vortex` improve by 2%, 19%, and 7%, respectively. Thus, we find that in most cases, the algorithmic overhead of running an online superpage promotion does not mask the potential benefits of promotion even when coupled with a low-overhead promotion mechanism. This result confirms the basic premise of Romer *et al.*'s study, that online promotion algorithms are a potentially valuable operating system technique for improving memory system performance on a wide variety of platforms.

Remember that in **approx-online** algorithm, *P.prefetch* is not incremented unless potential superpage *P* contains a current TLB entry as one of its component pages. To evaluate the importance of *tlbcount* checking, we take it off to see what happens. That is, *P.prefetch* is incremented as long as any of its component pages causes a TLB miss. Figure 7 shows the normalized speedups obtained on a 32-entry TLB without *tlbcount* checking. Compare Figure 7 with Figure 3 and we can see that performance drops noticeably for almost all the configurations. This experiment proves the effectiveness of *tlbcount* checking, which makes **approx-online** superpage promotion less speculative.

6 Conclusions and Future Work

To summarize our results, we find that when creating superpages dynamically:

- Remapping-based promotion outperforms copying-based promotion by up to 30%.
- Remapping-based superpage promotion has better cache performance than copying-based promotion. Depending on the application, the difference in cache performance can significantly affect the speedup of superpage promotion.
- Remapping-based **asap** superpage promotion is the most promising approach (because the cost of promotion is relatively low).

Although our results for copying-based promotion are qualitatively similar to Romer *et al.*'s, they differ quantitatively. Romer *et al.* use trace-driven simulation, thus their cost model for promotion is quite simple. Based on our measurements, the costs for copying-based promotion are significantly higher in a real system, largely due to cache effects. In addition, we find that the promotion thresholds used in Romer *et al.*'s **approx-online** simulations tend to be too high.

As applications continue to consume larger amounts of memory, the necessity of using superpages will grow. Our most significant result is that, given relatively simple hardware at the memory controller, a straightforward greedy policy for constructing superpages works well.

Further work in this area should look at how the different promotion mechanisms and policies interact with multiprogramming. When multiple programs compete for TLB space, it is possible that the choice of which mechanism and policy is best will change. In particular, the penalty for being too aggressive in creating superpages increases when the memory subsystem might be forced to tear down superpages to support demand paging. Our intuition is that remapping-based **asap** will likely remain the best choice, because it combines the lowest overhead promotion policy with the lowest overhead promotion mechanism.

References

[1] Advanced Micro Devices. AMD Athlon processor technical brief. `http://www.amd.com/products/cpg/athlon/-techdocs/pdf/22054.pdf`, 1999.

[2] K. Bala, F. Kaashoek, and W. Weihl. Software prefetching and caching for translation buffers. In *Proc. of the First OSDI*, pp. 243–254, Nov. 1994.

[3] B. Bershad, D. Lee, T. Romer, and J. Chen. Avoiding conflict misses dynamically in large direct-mapped caches. In *Proc. of the 6th ASPLOS*, pp. 158–170, Oct. 1994.

[4] G. Blelloch, C. Leiserson, B. Maggs, C. Plaxton, S. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine cm-2. In *Proc. of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 3–16, July 1991.

[5] P. Cao, E. Felten, and K. Li. Implementation and performance of application-controlled file caching. In *Proc. of the First OSDI*, pp. 165–177, Nov. 1994.

[6] J. Carter, W. Hsieh, L. Stoller, M. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: Building a smarter memory controller. In *Proc. of the Fifth HPCA*, pp. 70–79, Jan. 1999.

[7] J. B. Chen, A. Borg, and N. P. Jouppi. A simulation based study of TLB performance. In *Proc. of the 19th ISCA*, pp. 114–123, May 1992.

[8] Compaq Computer Corporation. *Alpha 21164 Microprocessor Hardware Reference Manual*, July 1999.

[9] HAL Computer Systems Inc. SPARC64-GP processor. `http://mpd.hal.com/products/SPARC64-GP.html`, 1999.

[10] T. Hotchkiss, N. Marschke, and R. McClosky. A new memory system design for commercial and technical computing products. *Hewlett-Packard Journal*, 47(1):44–51, Feb. 1996.

[11] Intel Corporation. *Pentium Pro Family Developer's Manual*, Jan. 1996.

[12] B. Jacob and T. Mudge. Software-managed address translation. In *Proc. of the Third HPCA*, pp. 156–167, Feb. 1997.

[13] B. Jacob and T. Mudge. A look at several memory management units, tlb-refill mechanisms, and page table organizations. In *Proc. of the 8th ASPLOS*, pp. 295–306, Oct. 1998.

[14] A. Karlin, K. Li, M. Manasse, and S. Owicki. Empirical studies of competitive spinning for shared memory multiprocessors. In *Proc. of the 13th SOSP*, pp. 41–55, Oct. 1991.

[15] Y. Khalidi, M. Talluri, M. Nelson, and D. Williams. Virtual memory support for multiple page sizes. In *Proc. of the 4th WWOS*, pp. 104–109, Oct. 1993.

[16] MIPS Technologies, Inc. *MIPS R10000 Microprocessor User's Manual, Version 2.0*, Dec. 1996.

[17] J. Mogul. Big memories on the desktop. In *Proc. 4th WWOS*, pp. 110–115, Oct. 1993.

[18] M.Talluri and M. Hill. Surpassing the TLB performance of superpages with less operating system support. In *Proc. of the 6th ASPLOS*, pp. 171–182, Oct. 1994.

[19] M.Talluri, S. Kong, M. Hill, and D. Patterson. Tradeoffs in supporting two page sizes. In *Proc. of the 19th ISCA*, pp. 415–424, May 1992.

[20] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proc. of the Visualization '98 Conference*, Oct. 1998.

[21] T. Romer. *Using Virtual Memory to Improve Cache and TLB Performance*. PhD thesis, University of Washington, May 1998.

[22] T. Romer, W. Ohlrich, A. Karlin, and B. Bershad. Reducing TLB and memory overhead using online superpage promotion. In *Proc. of the 22nd ISCA*, pp. 176–187, June 1995.

[23] A. Saulsbury, F. Dahlgren, and P. Stenstrom. Recency-based TLB preloading. `http://www.ce.chalmers.se/-ash/recency-preloading.pdf`, 1999.

[24] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28:202–208, 1985.

[25] A. Srivastava and A. Eustace. ATOM: A system for building customized program analysis tools. In *Proc. of the 1994 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 196–205, June 1994.

[26] L. Stoller, R. Kuramkote, and M. Swanson. PAINT: PA instruction set interpreter. TR UUCS-96-009, University of Utah Department of Computer Science, Sept. 1996.

[27] SUN Microsystems, Inc. *UltraSPARC User's Manual*, July 1997.

[28] M. Swanson, L. Stoller, and J. Carter. Increasing TLB reach using superpages backed by shadow memory. In *Proc. of the 25th ISCA*, pp. 204–213, June 1998.

[29] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22nd ISCA*, pp. 24–36, June 1995.