# Reevaluating Online Superpage Promotion with Hardware Support

Zhen Fang, Lixin Zhang, John B. Carter, Wilson C. Hsieh, Sally A. McKee
*School of Computing*
*University of Utah*
{*zfang, lizhang, retrac, wilson, sam*}*@cs.utah.edu*
*http://www.cs.utah.edu/impulse/*

## Abstract

*Typical translation lookaside buffers (TLBs) can map a far smaller region of memory than application footprints demand, and the cost of handling TLB misses therefore limits the performance of an increasing number of applications. This bottleneck can be mitigated by the use of* superpages*, multiple adjacent virtual memory pages that can be mapped with a single TLB entry, that extend TLB reach without significantly increasing size or cost. We analyze hardware/software tradeoffs for dynamically creating superpages. This study extends previous work by using execution-driven simulation to compare creating superpages via copying with remapping pages within the memory controller, and by examining how the tradeoffs change when moving from a single-issue to a superscalar processor model. We find that remapping-based promotion outperforms copying-based promotion, often significantly. Copying-based promotion is slightly more effective on superscalar processors than on single-issue processors, and the relative performance of remapping-based promotion on the two platforms is application-dependent.*

## 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 [7, 16, 28]. Using superpages makes more efficient use of a 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 of promoting superpages by copying includes both direct and indirect costs. The direct costs come from copying the pages and changing the mappings. Indirect costs include 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, all costs must be balanced against the improvements in TLB performance.

Romer *et al.* [24] 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%. However, at least two significant architectural trends have emerged since Romer *et al*. performed their study. First, superscalar, out-of-order processor pipelines have replaced single-issue, in-order designs. Second, in response to the growing "memory wall" problem, architects have proposed a number of smart memory system designs [5, 14, 20]. Our work extends that of Romer *et al*. by considering the impact of these new architectural features when designing a dynamic superpage promotion mechanism. For example, Swanson *et al*. demonstrate that applications can create superpages *without copying* using the Impulse memory controller's physical-to-physical address remapping capabilities [29]. The resulting system yields a two-fold increase in TLB reach and a 5%-20% improvement in the performance of a mix of SPECint95 and Splash2 applications. We also extend previous work by employing an execution-driven simulation environment that more accurately models both the direct and indirect costs of a given promotion algorithm. Our results show that the differences in accuracy between the two

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.

simulation approaches are significant, especially when studying complex interactions between operating systems and modern architectures.

We draw several conclusions from this research. Combining the work of Romer *et al.* and Swanson *et al.* 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 performed via copying, but we find that for a remapping-based mechanism, a more aggressive policy that promotes superpages as soon as all of their constituent sub-pages have been touched performs best. In addition, we 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. When thus tuned, online superpage promotion via remapping achieves performance comparable to the hand-coded superpage promotion mechanism employed by Swanson *et al.*

We use an augmented version of the RSIM [21] simulation environment for all of our experiments. The simulator models a MIPS R1000-based workstation and a BSD-like micro kernel, including software TLB miss handlers. We model both a single-issue and four-way superscalar version of the processor pipeline, vary the TLB size from 64 to 128 entries, and consider two memory systems: a conventional memory controller and the Impulse memory controller [5], with its support for dynamic address remapping.

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. In particular, the direct and indirect costs of copying can lead to poor performance (as much as a 55% slow-down), whereas promotion via remapping performs well on all applications (ranging from a 230% speedup to a 5% slowdown). Copying-based promotion delivers fairly consistent benefits on both processor architectures, but remapping-based promotion yields especially high benefits on superscalar processors for applications with high degrees of instruction-level parallelism. Finally, we find that lost potential instruction-issue slots are a significant hidden source of TLB overhead in superscalar machines.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 describes Impulse, compares our simulation environment to Romer *et al.*'s, and outlines the two superpage promotion policies. Section 4 describes our benchmark suite and experimental methodology, and gives the results of our study. Section 5 summarizes our conclusions and discusses future work.

## 2 Related Work

Competitive algorithms perform online cost/benefit analyses to make decisions that guarantee performance within a constant factor of an optimal offline algorithm. Romer *et al.* [24] adapt this approach to TLB management, and employ 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, 23]. Competitive algorithms have been used to help increase the efficiency of other operating system functions and resources, including paging [26], synchronization [12], and file cache management [4].

Chen *et al.* [6] 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 [11] 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 accounts for much of the memory-management overhead. They also project that individual costs 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, 8]), to implementing better management policies (in software [10] or hardware [9]), to masking TLB miss latency by prefetching entries (again, in software [2] or hardware [25]).

All of these approaches can be improved by exploiting superpages. Most commercial TLBs support superpages, and have for several years [16, 28], but more research is needed into how best to make general use of them. Chen *et al.* [6] suggest the possibility of using variable page sizes to improve TLB reach, but do not explore the implications of their use. Khalidi [13] and Mogul [17] discuss the benefits of systems that support superpages, and advocate static allocation via compiler or programmer hints. Talluri *et al.* [18] report on 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%, though. Mixing both page size limits this bloat to around 10%, and allowing the TLB to map superpages without requiring that all the underlying base pages be present eliminates the bloat.

# 3 Experimental Parameters

We measure the performance impact of no-copy superpage promotion and instruction issue width for the two online promotion algorithms proposed by Romer *et al*. [24]. In this section we describe the Impulse memory controller, compare our processor models to Romer's, and review Romer's promotion policies.

## 3.1 Impulse Superpage Promotion

The Impulse memory system [29] supports an extra level of address remapping at the MMC (main memory controller): unused physical addresses are remapped into "real" physical addresses. We refer to the remapped addresses as *shadow addresses*. From the point of view of the processor, shadow addresses are used in place of real physical addresses when needed. 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 memory controller maintains its own page tables for shadow memory mappings. Building superpages from base pages that are not physically contiguous entails simply remapping the virtual pages to properly aligned shadow pages. The memory controller then maps the shadow pages to the original physical pages. The processor's TLB is not affected by the extra level of translation that takes place at the controller.

Figure 1 illustrates how superpage mapping works on Impulse. In this example, the OS has mapped a contiguous 16KB virtual address range to a single shadow superpage at "physical" page frame `0x80240`. When an address in the shadow physical range is placed on the system memory bus, the memory controller detects that this "physical" address needs to be retranslated using its local shadow-to-physical translation tables. 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`.

## 3.2 Processor Models

Our studies use the URSIM [30] execution-driven simulator, derived from RSIM [21]. URSIM models a microarchitecture close to MIPS R10000 microprocessor [16] with a 32-entry instruction window. It can be configured to issue either one or four instructions per cycle. We model a 64-kilobyte L1 data cache that is non-blocking, write-back, virtually indexed, physically tagged, direct-mapped, and has 32-byte lines. The 512-kilobyte L2 data cache is non-blocking, write-back, physically indexed, physically tagged, two-way associative, and has 128-byte lines. L1 cache hits take one cycle, and L2 cache hits take eight cycles.

URSIM models a split-transaction MIPS R10000 cluster bus with a snoopy coherence protocol. The bus multiplexes addresses and data, is eight bytes wide, has a three-cycle arbitration delay and a one-cycle turnaround time. We model two memory controllers: a conventional high-performance MMC based on the one in the SGI O200 server and the Impulse MMC. The system bus, memory controller, and DRAMs have the same clock rate, which is one third of the CPU clock's. The memory system supports critical word first, i.e., a stalled memory instruction resumes execution after the first quad-word returns. The load latency of the first quad-word is 16 memory cycles.

The unified TLB is single-cycle, fully associative, software-managed, and combined instruction and data. It employs a least-recently-used replacement policy. 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 2048 base pages. We model two TLB sizes: 64 and 128 entries.

Our system model differs from that modeled by Romer *et al*. in several significant ways. They employ a form of trace-driven simulation with ATOM [27], a binary rewriting tool. That is, they rewrite their applications using ATOM to monitor memory references, and the modified applications are used to do on-the-fly "simulation" of TLB behavior. Their simulated system has two 32-entry, fully-associative TLBs (one for instructions and one for data), uses LRU replacement on TLB entries, and has a base page size of 4096 bytes.

Romer *et al*. combine the results of their trace-driven simulation with measured baseline performance results to calculate effective speedup on their benchmarks. They execute their benchmarks on a DEC Alpha 3000/700 running DEC OSF/1 2.1. The processor in that system is a dual-issue, in-order, 225 MHz Alpha 21064. The system has two megabytes of off-chip cache and 160 megabytes of main memory.

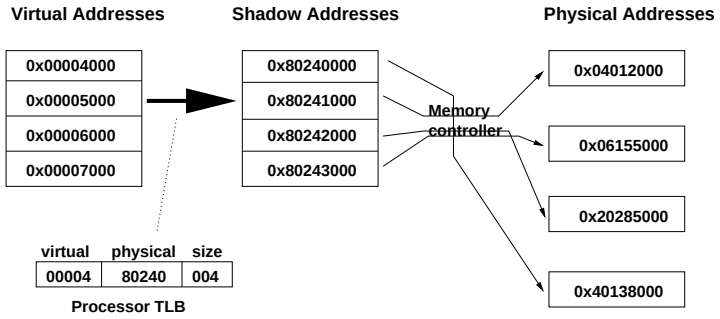



**Figure 1.** An Example of Creating Superpages Using Shadow Space

For their simulations, they assume the following fixed costs, which do not take cache effects into account:

- each 1Kbyte copied is assigned a 3000-cycle cost;
- the **asap** policy is charged 30 cycles for each TLB miss;
- and the **approx-online** policy is charged 130 cycles for each TLB miss.

### 3.3 Superpage Promotion Policies

We evaluate two of the online superpage promotion policies developed by Romer *et al*. [24], **asap** and **approx-online**. Note that **approx-online** is a simplification of a more complex **online** policy [24]. Romer [23] shows that **approx-online** is as effective as **online**, but has much lower bookkeeping overhead.

**asap** is a greedy policy that promotes a set of pages to a superpage as soon as each page has been referenced. The algorithm minimizes bookkeeping overhead by ignoring reference frequency for the potential superpages. The price for this simplicity is that the **asap** policy may build superpages that will rarely be referenced later, in which case the benefits of having the superpages would not offset the costs of building them.

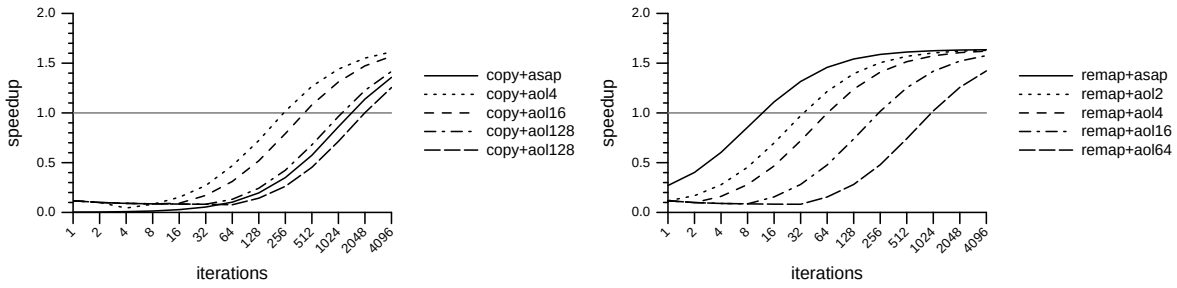
**approx-online** uses a competitive strategy to determine when superpages should be coalesced. If a superpage $P$ accrues many misses, probably it will be referenced again in the future, and promoting it should prevent future TLB misses. Such promotions effectively *prefetch* the translations for any non-resident base pages that the superpage contains. To track reference information, the **approx-online** algorithm maintains a *prefetch charge* counter *P.prefetch* for each potential superpage $P$. 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. Each superpage size is given a miss threshold; when the prefetch charge for a superpage reaches its miss threshold, that superpage is created.

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 must be small enough for useful superpages to be promoted early enough to eliminate future TLB misses; on the other hand, it must be large enough that the cost of promotion does not dominate TLB overhead.

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 [23] proves that a system that employs **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, we find that smaller thresholds tend to work better in practice.

## 4 Results

The performance results presented here are obtained through complete simulation of the benchmarks. We measure both kernel and application time, the direct overhead of implementing the superpage promotion algorithms and the resulting effects on the system, including the expanded TLB miss handlers, cache effects due to accessing the page tables and maintaining prefetch counters, and the overhead associated with promoting and using superpages with Impulse. We first present the results of our microbenchmark experiments to explore the break-even points for each of the superpage promotion policies and mechanisms, and then we present com-




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

**Figure 2.** Microbenchmark performance. aol*n*: approx-online with threshold *n*.

parative performance results for our application benchmark suite.

## 4.1 Microbenchmark Results

The number of TLB misses that must be eliminated per promotion to amortize the cost of implementing the promotion algorithm is an important performance factor when comparing online superpage promotion schemes. This cost includes the extra time spent in the TLB miss handler determining when to coalesce pages and 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 microbenchmark that consists of a loop that touches 4096 different base pages for a configurable number of iterations:

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

for (j = 0; j < 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 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.

Figure 2(a) and Figure 2(b) illustrate the 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 64-entry and a 128-entry TLB. The $x$ axes indicate the number of loop iterations (and thus the number of times each page is referenced). These graphs emphasize the performance differences among the **copying** and **remapping** policies.

Copying-based **asap** performs much worse than remapping-based **asap**, especially when pages are seldom referenced. Execution time with copying is 75 times greater than execution time for remapping when each page is touched only once. Copying-based **asap** only becomes profitable after each page is touched about 2000 times, but remapping-based **asap** breaks even after only 16 references per page. The mean cost of a TLB miss increases from around 37 cycles in the baseline to 412 cycles for remapping **asap**, and to 8100 cycles for copying **asap**.

Performance for all **approx-online** configurations suffers when the threshold is larger than the number of references to each page. The additional overheads in the TLB miss handler dominate the microbenchmark's execution time. 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 4096 times. The number of references required for **approx-online** to be profitable increases with the threshold, and for a given threshold, the number of references per page required to make copying-based promotion profitable is at least twice the number for the remapping-based approach. The TLB miss penalty goes from about 37 cycles in the baseline to 1100 cycles for remapping **approx-online** and 2300 cycles for copying **approx-online**.

For the microbenchmark, **asap** outperforms **approx-online** when remapping is employed, but **approx-online** beats **asap** when copying is used. **approx-online** carefully chooses the superpages to promote, but the history mechanism used to make the decisions is expensive. Since promoting superpages by copying is expensive, the cost of this promotion policy is justified. In contrast, the lower overhead of

the remapping mechanism allows for more aggressive promotion of pages to superpages.

## 4.2 Application Results

To evaluate the different superpage promotion approaches on larger problems, we use eight 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`), one scientific benchmark (`adi`), and one benchmark from the DIS benchmark suite (`dm`) [15]. All benchmarks were compiled with Sun cc Workshop Compiler 4.2 and optimization level "-xO4".

`Compress` is the SPEC95 data compression program run on an input of ten million characters. To avoid overestimating the efficacy of superpages, the compression algorithm was run only once, instead of the default 25 times. `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. `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. [22] `filter` performs an order-129 binomial filter on a 32×1024 color image. `rotate` turns a 1024×1024 color image clockwise through one radian. `adi` implements algorithm *alternative direction integration*. `dm` is a data management program using input file "dm07.in".

Two of these benchmarks, `gcc` and `compress`, are also included in Romer *et al.*'s benchmark suite, although we use SPEC95 versions, whereas they used 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 benchmarks were based on tools used in the research environment at the University of Washington, and were not readily available to us.

Table 1 lists the characteristics of the baseline run of each benchmark with a four-way issue superscalar processor, where no superpage promotion occurs. *TLB miss time* is the total time spent in the data TLB miss handler. These benchmarks demonstrate varying sensitivity to TLB performance: on the system with the smaller TLB, between 9.2% and 35.1% of their execution time is lost due to TLB miss costs. The percentage of time spent handling TLB misses falls to between less than 1% and 33.4% on the system with a 128-entry TLB.

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. In our experiments we found that the best **approx-online** threshold for a two-page superpage is 16 on a conventional system and is 4 on an Impulse system. These are also the thresholds used in our full-application tests. Figure 3 gives results with a 64-entry TLB; Figure 4 gives results with a 128-entry TLB. Online superpage promotion can improve performance by up to a factor of two (on `adi` with remapping **asap**), but it also can decrease performance by a similar factor (when using the copying version of **asap** on `raytrace`). We can make two orthogonal comparisons from these figures: *remapping* versus *copying*, and **asap** versus **approx-online**.

| Benchmark | Total cycles (M) | Cache misses (K) | TLB misses (K) | TLB miss time |
|---|---|---|---|---|
| 64-entry TLB | | | | |
| compress | 632 | 3455 | 4845 | 27.9% |
| gcc | 628 | 1555 | 2103 | 10.3% |
| vortex | 605 | 1090 | 4062 | 21.4% |
| raytrace | 94 | 989 | 563 | 18.3% |
| adi | 669 | 5796 | 6673 | 33.8% |
| filter | 425 | 241 | 4798 | 35.1% |
| rotate | 547 | 3570 | 3807 | 17.9% |
| dm | 233 | 129 | 771 | 9.2% |
| 128-entry TLB | | | | |
| compress | 426 | 3619 | 36 | 0.6% |
| gcc | 533 | 1526 | 332 | 2.0% |
| vortex | 423 | 763 | 1047 | 8.1% |
| raytrace | 93 | 989 | 548 | 17.4% |
| adi | 662 | 5795 | 6482 | 32.1% |
| filter | 417 | 240 | 4544 | 33.4% |
| rotate | 545 | 3569 | 3702 | 16.9% |
| dm | 211 | 143 | 250 | 3.3% |

**Table 1.** Characteristics of each baseline run

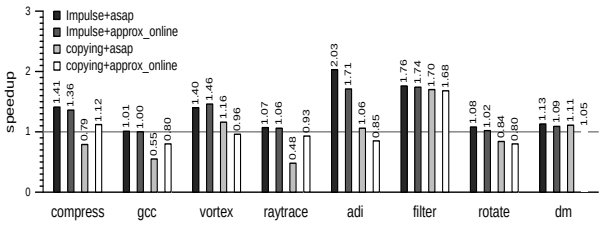



**Figure 3.** Normalized speedups for each of two promotion policies on a four-issue system with a 64-entry TLB.

### 4.2.1 Asap vs. Approx-online

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

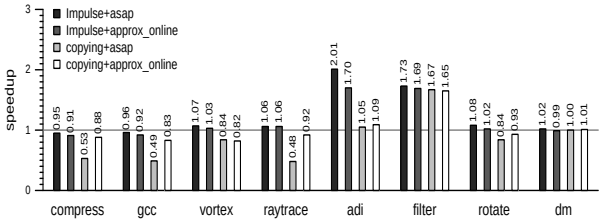



**Figure 4.** Normalized speedups for each of two promotion policies on a 4-issue system with a 128-entry TLB.

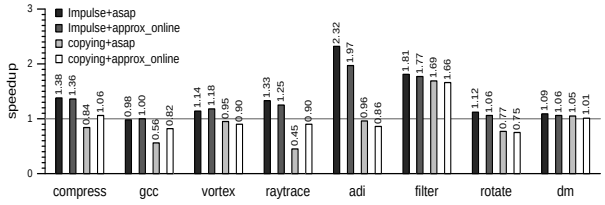



**Figure 5.** Normalized speedup on a single-issue processor with a 64-entry TLB

mechanism, *remapping* or *copying*. Using remapping , **asap** slightly outperforms **approx-online** in the average case. It exceeds the performance of **approx-online** in 14 of the 16 experiments, and trails the performance of **approx-online** in only one case (on `vortex` with a 64-entry tlb). The differences in performance range from *asap+remap* outperforming *aol+remap* by 32% for `adi` with a 64-entry TLB, to *aol+remap* outperforming *asap+remap* by 6% for `vortex` with a 64-entry TLB. In general, however, performance differences between the two policies are small: **asap** is on average 7% better with a 64-entry TLB, and 6% better with a 128-entry TLB.

The results change noticeably when we employ a *copying* promotion mechanism. With copying, **approx-online** outperforms **asap** in nine of the 16 experiments, while the policies perform almost identically in three of the other seven 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 20% for `vortex` with a 64-entry TLB, to **approx-online** outperforming **asap** by 45% for `raytrace` with a 64-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**. Taking the arithmetic mean of the performance differences reveals that **asap** is, on average, 6% better with a 64-entry TLB, and 4% better with a 128-entry TLB.

The relative performance of the **asap** and **approx-online** promotion policies changes when we employ different promotion 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 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 with it than **approx-online** does.

#### 4.2.2 Remapping vs. Copying

When we compare the two superpage creation mechanisms, *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 97% in the case of `adi` on both a 64-entry and 128-entry TLB. Overall, *asap+remap* outperforms *aonline+copying* by more than 10% in eleven of the sixteen experiments, averaging 33% better with a 64-entry TLB, and 22% better with a 128-entry TLB.

#### 4.2.3 Single-issue vs. Four-issue

The TLB miss time shown in Table 1 includes only the cycles spent in the TLB miss handler. This time is the primary cause of TLB miss overhead on a single-issue machine. On a superscalar machine, however, TLB misses also lead to unusable issue slots between when a miss is detected by the TLB and when the faulting instruction reaches the head of the instruction queue. At this point, the processor flushes the instruction queue and the program traps to an appropriate TLB miss handler. These lost issue slots represent a potentially serious performance problem.

To determine the importance of these lost potential issue slots, we compare the performance of the various promotion algorithms on both a single-issue machine (Figure 5) and a superscalar machine (Figure 3), both with 64 TLB entries. We find that the impact of copying-based promotion is fairly constant across platforms. However, five of the eight benchmarks (`compress`, `gcc`, `vortex`, `filter` and `dm`) benefit more from remapping-based superpage promotion on a superscalar processor than on a single-issue processor. For the other three, remapping-based superpage promotion is more effective on a single-issue processor.

| | Single-issue | | | | Four-way | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark | $G_{ipc}$ | $T_{ipc}$ | Handler time | Lost cycles | $G_{ipc}$ | $T_{ipc}$ | Handler time | Lost cycles |
| compress | 0.75 | 0.62 | 24.5% | 1.0% | 1.22 | 0.89 | 27.9% | 3.9% |
| gcc | 0.90 | 0.77 | 8.0% | 0.4% | 1.55 | 1.02 | 10.3% | 1.9% |
| vortex | 0.90 | 0.78 | 16.1% | 0.9% | 1.54 | 1.01 | 21.4% | 2.4% |
| raytrace | 0.45 | 0.53 | 28.8% | 3.1% | 0.57 | 1.05 | 18.3% | 43.0% |
| adi | 0.41 | 0.59 | 44.5% | 18.7% | 0.51 | 0.96 | 33.8% | 38.5% |
| filter | 0.83 | 0.77 | 36.1% | 1.4% | 1.07 | 1.03 | 35.1% | 8.7% |
| rotate | 0.56 | 0.74 | 23.2% | 25.7% | 0.64 | 1.09 | 17.9% | 50.1% |
| dm | 0.91 | 0.80 | 7.2% | 0.3% | 1.67 | 1.14 | 9.2% | 1.9% |

**Table 2.** IPCs and cycles lost due to TLB misses, with a 64-entry TLB

To understand why remapping-based promotion sometimes improves performance more on a superscalar machine than on a single-issue machine, while copying-based promotion is always most effective on a superscalar machine, one must consider a number of specific application characteristics. Table 2 presents the instructions per cycle (IPC) for the non-TLB handler code ($G_{ipc}$, for "global IPC"), the IPC for the TLB miss handler ($T_{ipc}$), the percentage of application cycles spent in the data TLB miss handler, and the percentage of potential issue slots that are lost due to TLB misses for each application.

The TLB lost time can be significant for some of the applications. For example, on a 4-issue processor, `rotate`, `raytrace` and `adi` waste 50%, 43%, and 39% of their potential issue slots while TLB misses are pending, respectively. With superpages we found that these lost cycles drop below 1% of total execution time for all the benchmarks.

Two factors influence the effectiveness of superpage promotion: the percentage of application cycles spent handling TLB misses and the cost of superpage promotion. For Impulse, superpage promotion overhead is negligible, and so the relative impact of remapping-based superpage promotion on the two platforms is controlled by the ratio $G_{ipc}/T_{ipc}$. When this ratio is greater than 1.0 (as is the case for `compress`, `gcc`, `vortex`, `filter` and `dm`), remapping-based promotion has a greater impact on a superscalar machine than on a single-issue machine. This is because $G_{ipc}/T_{ipc}$ above one means that the non-TLB miss handling code has greater instruction-level parallelism than the TLB miss handling code. Therefore, avoiding TLB misses is particularly beneficial on a superscalar machine. The converse is true for the three applications with smaller $G_{ipc}/T_{ipc}$ values. The relative performance of copying-based promotion, on the other hand, tends to be dominated by the cost of the copying itself, which is smaller on the superscalar processor than on the single-issue processor. However, for `adi`, `raytrace`, and `rotate`, copying doubles the total number of instructions executed. This overwhelms the potential performance benefits of avoiding TLB misses, even when the cost of copying is reduced by running on a superscalar processor.

| | cycles per 1K bytes promoted | average cache hit ratio | baseline cache hit ratio |
|---|---|---|---|
| gcc | 10,798 | 98.81% | 99.33% |
| filter | 5,966 | 99.80% | 99.80% |
| raytrace | 10,352 | 96.50% | 87.20% |
| dm | 6,534 | 99.80% | 99.86% |

**Table 3.** Average copy costs (in cycles) for **approx-online** policy.

## 4.3 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 Impulse, we can also afford to promote pages more aggressively.

Romer *et al.*'s trace-based simulation does not model any 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 [24]. Table 3 shows our measured per-kilobyte cost (in CPU cycles) to promote pages by copying for four representative benchmarks. (Note that we also assume a relatively faster processor.) 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, copying is at least twice as expensive as Romer *et al.* assumed. For `gcc` and `raytrace`, superpage promotion costs more than three times the cost charged in the trace-driven study. Part of these differences are due to the cache effects that copying incurs.

We find that when copying is used to promote pages, **approx-online** performs better with a lower (more ag-

gressive) threshold than used by Romer *et al.* Specifically, the best thresholds that our experiments revealed varied from four to 16, while their study used a fixed threshold of 100. This difference in thresholds has a significant impact on performance. For example, when we run the `adi` benchmark using a threshold of 32, **approx-online** with copying *slows* performance by 10% with a 128-entry TLB. In contrast, when we run **approx-online** with copying using the best threshold of 16, performance *improves* by 9%. In general, we find that even the copying-based promotion algorithms need to be more aggressive about creating superpages than was suggested by Romer *et al*. Given that our cost of promoting pages is much higher than the 3000 cycles estimated in their study, one might expect that the best thresholds would be higher than Romer's. However, the cost of a TLB miss far outweighs the greater copying costs; our TLB miss costs are about an order of magnitude greater than those assumed in his study.

## 5 Conclusions and Future Work

To summarize, we find that when creating superpages dynamically:

- Impulse delivers greater (often much greater) speedups than copying, due to lower direct and indirect overheads in setting up the superpages.
- To be effective, the different online superpage promotion algorithms need to be more aggressive than suggested in Romer *et al*.'s paper. We find that using a more modern simulation models make a big difference (quantitatively and qualitatively) in the results. In our experiments, **asap** works best for Impulse-style promotion, but **approx-online** still delivers better performance for copying promotion mechanisms.
- The effectiveness of promotion using Impulse is affected by both the percentage of time that the application spends handling TLB misses, and how full the pipeline tends to be when a TLB miss occurs. For applications with low IPCs (where the pipeline tends to advance slowly), Impulse-based promotion tends to have a larger impact on a single-issue machine. For applications with high IPCs, Impulse tends to have a bigger impact on a superscalar machine. The benefit of promotion via copying is largely determined by the percentage of time an application spends taking TLB misses – copying is only beneficial when TLB miss time is high.
- The time between when a miss is flagged by the TLB and when the instruction reaches the front of the instruction pipeline (and the trap actually happens), can give rise to the loss of many potential instruction-issue slots. These are a significant, hidden source of TLB overhead in a superscalar machine.

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 results are: given relatively simple hardware at the memory controller, a straightforward greedy policy for constructing superpages works well; and lost potential instruction-issue slots are a significant hidden source of TLB overhead in superscalar machines.

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 cheaper promotion policy with the cheaper 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 Symposium on Operating System Design and Implementation*, 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] P. Cao, E. Felten, and K. Li. Implementation and performance of application-controlled file caching. In *Proc. of the First Symposium on Operating System Design and Implementation*, pp. 165–177, Nov. 1994.

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] J. Kuskin and D. O. et al. The Stanford FLASH multiprocessor. In *Proc. of the 21st ISCA*, pp. 302–313, May 1994.

[15] J. W. Manke and J. Wu. *Data-Intensive System Benchmark Suite Analysis and Specification*. Atlantic Aerospace Electronics Corp., June 1999.

[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] M. Oskin, F. Chong, and T. Sherwood. Active pages: A model of computation for intelligent memory. In *Proc. of the 25th ISCA*, pp. 192–203, June 1998.

[21] V. Pai, P. Ranganathan, and S. Adve. Rsim reference manual, version 1.0. *IEEE Technical Committee on Computer Architecture Newsletter*, Fall 1997.

[22] 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.

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

[24] 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.

[25] A. Saulsbury, F. Dahlgren, and P. Stenstrom. Recency-based TLB preloading. In *Proc. of the 27th ISCA*, pp. 117–127, June 2000.

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

[27] 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.

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

[29] 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.

[30] L. Zhang. URSIM reference manual. TR UUCS-00-015, University of Utah, August 2000.