

# Online Superpage Promotion Revisited

Lixin Zhang, Sally Mckee, John Carter, Wilson Hsieh and Zhen Fang

<http://www.cs.utah.edu/impulse>  
Department of Computer Science  
University of Utah

## 1 Introduction

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.

## 2 Research Background

### 2.1 Promotion Algorithms

We evaluate two of the online superpage promotion policies developed by Romer *et al.*, **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 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. Then 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.

The choice of threshold value 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 quantified this tradeoff.

### 2.2 Promotion via Remapping

No-copy superpage creation relies on hardware support provided by the Impulse memory controller. 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 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 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.

## 3 Simulation

### 3.1 Simulation Environment

Our studies use the execution-driven simulator Paint, which models a variation of a 240 MHz, HP PA-RISC 1.1 processor running a BSD-based microkernel. The 64-kilobyte L1 data cache is non-blocking, virtually indexed, and direct-mapped with 32-byte lines. The 512-kilobyte L2 data cache is non-blocking, physically indexed and 2-way set-associative with 128-byte lines. The memory system has a total memory latency of 60 cycles. The simulated remapping memory controller is based on the HP controller used in high-end workstations. The MTLB is configured at 1024 entries.

The TLB holds both instruction and data translations. 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 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.

### 3.2 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`). All benchmarks were compiled with gcc 2.7.2 and optimization level “-O2”.

### 3.3 Results and Conclusions

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.