Practical, transparent operating system support for superpages

Juan Navarro ● Sitaram Iyer
Peter Druschel ● Alan Cox

Rice University

https://dl.acm.org/citation.cfm?id=844138
Overview

- Increasing cost in TLB miss overhead
  - growing working sets
  - TLB size does not grow at same pace

- Processors now provide superpages
  - one TLB entry can map a large region

- OSs have been slow to harness them
  - no transparent superpage support for apps

- This talk: a practical and transparent solution to support superpages
Translation look-aside buffer

- TLB caches virtual-to-physical address translations

- TLB coverage
  - amount of memory mapped by TLB
  - amount of memory that can be accessed without TLB misses
How to increase TLB coverage

- Typical TLB coverage ≈ 1 MB

- Use superpages!
  - large and small pages
  - Increase TLB coverage
  - no increase in TLB size
What are these superpages anyway?

- Memory pages of larger sizes
  - supported by most modern CPUs
- Otherwise, same as normal pages
  - power of 2 size
  - use only one TLB entry
  - contiguous
  - aligned (physically and virtually)
  - uniform protection attributes
  - one reference bit, one dirty bit
A superpage TLB
A superpage TLB

virtual memory

virtual address

base page entry (size=1)

superpage entry (size=4)

TLB

physical memory

Alpha:
8,64,512KB; 4MB

Itanium:
4,8,16,64,256KB; 1,4,16,64,256MB
II
The superpage problem
Issue 1: superpage allocation

How / when / what size to allocate?
Issue 1: superpage allocation

How / when / what size to allocate?
Issue 1: superpage allocation

How / when / what size to allocate?
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page

- When to promote?

Wait for app to touch pages? May lose opportunity to increase TLB coverage.
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?

Create small superpage? May incur overhead.
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?

Forcibly populate pages? May incur I/O cost or increase internal fragmentation.
Issue 3: demotion

Demotion: convert a superpage into smaller pages

- when page attributes of base pages of a superpage become non-uniform
- during partial pageouts
Issue 4: fragmentation

• Memory becomes externally fragmented due to
  • use of multiple page sizes
  • Scattered wired pages
    • Wired pages = pages that can’t be paged out to swap device
    • break contiguity of free base pages since they cannot be relocated.

• External fragmentation occurs at superpage sizes.
  • No external fragmentation at base page granularity

• Contiguity of free pages is a contended resource
  • Contiguous pages = pages that are next to each other
  • Allocating a superpage requires that sufficient number of contiguous base pages must be free.

• OS must
  • use contiguity restoration techniques
  • trade off impact of contiguity restoration against superpage benefits
Previous approaches

- **Reservations**
  - one superpage size only

- **Relocation**
  - move pages at promotion time
  - must recover copying costs

- **Eager superpage creation (IRIX, HP-UX)**
  - size specified by user: non-transparent

- **Hardware support**
  - Contiguous virtual superpage mapped to discontiguous physical base pages

- **Demotion issues not addressed**
  - large pages partially dirty/referenced
III
Design
Key observation

Once an application touches the first page of a memory object then it is likely that it will quickly touch every page of that object.

- Example: array initialization
- Opportunistic policies
  - superpages as large and as soon as possible
  - as long as no penalty if wrong decision
Superpage allocation

Preemptible reservations

Virtual memory

Superpage boundaries

Physical memory

Reserved frames
Superpage allocation

Preemptible reservations

virtual memory

superpage boundaries

physical memory
Superpage allocation

Preemptible reservations

virtual memory

superpage boundaries

physical memory
Opportunistic policy

- Go for biggest size that is no larger than the memory object (e.g., file)
- If required size not available, try preemption before resigning to a smaller size
  - preempted reservation had its chance
Allocation: managing reservations

best candidate for preemption at front:

- reservation whose most recently populated frame was populated the least recently
Incremental promotions

Promotion policy: opportunistic

- 2
- 4
- 4+2
- 8
Speculative demotions

- One reference bit per superpage
  - How do we detect portions of a superpage not referenced anymore?

- On memory pressure, demote superpages when resetting ref bit

- Re-promote (incrementally) as pages are referenced

- Demote also when the page daemon selects a base page as a victim page.
Demotions: dirty superpages

- One dirty bit per superpage
  - what’s dirty and what’s not?
  - page out entire superpage
- Demote on first write to clean superpage

- Re-promote (incrementally) as other pages are dirtied
Fragmentation control

- Low contiguity: modified page daemon for victim selection
  - restore contiguity
    - move clean, inactive pages to the free list
  - minimize impact
    - prefer victim pages that contribute the most to contiguity

- Cluster wired pages
  - Assign a dedicated region of physical memory for wired pages
  - So that they break contiguity for superpage allocations from rest of the memory.
IV
Experimental evaluation
Experimental setup

- FreeBSD 4.3
- Alpha 21264, 500 MHz, 512 MB RAM
- 8 KB, 64 KB, 512 KB, 4 MB pages
- 128-entry DTLB, 128-entry ITLB
- Unmodified applications
Best-case benefits

- TLB miss reduction usually above 95%
- SPEC CPU2000 integer
  - 11.2% improvement (0 to 38%)
- SPEC CPU2000 floating point
  - 11.0% improvement (-1.5% to 83%)
- Other benchmarks
  - FFT (200³ matrix): 55%
  - 1000x1000 matrix transpose: 655%
- 30%+ in 8 out of 35 benchmarks
### Why multiple superpage sizes

<table>
<thead>
<tr>
<th></th>
<th>64KB</th>
<th>512KB</th>
<th>4MB</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>1%</td>
<td>0%</td>
<td>55%</td>
<td>55%</td>
</tr>
<tr>
<td>galgel</td>
<td>28%</td>
<td>28%</td>
<td>1%</td>
<td>29%</td>
</tr>
<tr>
<td>mcf</td>
<td>24%</td>
<td>31%</td>
<td>22%</td>
<td>68%</td>
</tr>
</tbody>
</table>

**Improvements with only one superpage size vs. all sizes**
Conclusions

• **Superpages**
  • OS can provide transparent support for a mix of superpages by applications.

• **Contiguity restoration is necessary**
  • sustains benefits; low impact

• **Multiple page sizes are important**
  • scales to very large superpages
More references:

- Multiple page sizes in different processors
  - [https://en.wikipedia.org/wiki/Page_(computer_memory)#Multiple_page_sizes](https://en.wikipedia.org/wiki/Page_(computer_memory)#Multiple_page_sizes)

- Linux Transparent Hugepages
  - [https://lwn.net/Articles/423584/](https://lwn.net/Articles/423584/)