Practical, transparent operating system support for superpages

Juan Navarro • Sitaram Iyer
Peter Druschel • Alan Cox

Rice University

OSDI 2002
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 $\approx 1$ 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 than standard pages
  - supported by most modern CPUs

- Superpage size = power of 2 x the base page size

- Only one TLB entry per superpage
  - But multiple (identical) page-table entries, one per base page

- Constraints:
  - contiguous (physically and virtually)
  - aligned (physically and virtually)
  - uniform protection attributes
  - one reference bit, one dirty bit
A superpage TLB

Virtual address

Base page entry (size=1)

Superpage entry (size=4)

TLB

Physical address

Virtual memory

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 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.

Wait for app to touch pages? May lose opportunity to increase TLB coverage.

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 fragmented due to
  - use of multiple page sizes
  - scattered *wired* (non-pageable) pages

- Contiguity: contended resource

- 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
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

How much do we reserve? Goal: good TLB coverage, without internal fragmentation.
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

largest unused (and aligned) chunk

4

2

1

best candidate for preemption at front:

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

Promotion policy: opportunistic
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 pages that contribute the most to contiguity

- Cluster wired pages
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: 30%+ improvement
  ▪ transparently realized; low overhead
◆ Contiguity restoration is necessary
  ▪ sustains benefits; low impact
◆ Multiple page sizes are important
  ▪ scales to very large superpages
More info at
www.cs.rice.edu/~jnavarro/superpages