Lecture 14: Practical, transparent operating system support for superpages

Juan Navarro · Sitaram Iyer
Peter Druschel · Alan Cox
Rice University
(slides adapted from OSDI 2002 presentation)

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

How to increase TLB coverage

• Typical TLB coverage = 1 MB
• Use superpages!
  • Both large and small pages - power-of-2 size
  • 1 TLB entry per superpage
  • Contiguous, and virtually and physically aligned
  • Uniform attributes (protection, valid, ref, dirty)
• Benefit: Increase TLB coverage
  • no increase in TLB size
  • no internal fragmentation

If only large pages: larger working sets, more I/O.
A superpage TLB

- Alpha: 8, 64, 512KB; 4MB
- Itanium: 4, 8, 16, 64, 256KB; 1, 4, 16, 64, 256MB

virtual memory

virtual address

base page entry (size=1)

physical memory

physical address

Why multiple superpage sizes

<table>
<thead>
<tr>
<th>App</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>

- Different apps have different “best” size
  - Different data structures in a single app have different “best” size

Previous research approaches

- Reservations
  - Talluri & Hill “Surpassing the TLB performance of superpages with less operating system support”
  - one superpage size only, designed to work with proposed partial sub-block TLBs

- Relocation
  - move pages at promotion time
  - must recover copying costs
  - E.g. Romer, et al. “Reducing TLB and memory overhead using online superpage promotion”.

- Not known to be implemented in non-research OS

Prior commercial OS approaches

- Eager superpage creation (IRIX, HP-UX)
  - Superpage is allocated at page fault time
  - Size specified by user: non-transparent
    - IRIX
      - can select different page size for any suitably-aligned range of the virtual address space
      - OS maintains list of free pages of each size, coalescing daemon periodically tries to refresh
      - Large pages can be demoted under memory pressure
    - HP-UX
      - Can select different sizes for text and data segment only
      - Hint is associated with binary, not selectable at run-time
The Superpage Problem

- Main Issues
  - Allocation
  - Promotion
  - Demotion
  - Fragmentation

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?
  - Forcibly populate pages? May cause internal fragmentation.
  - Create small superpage? May waste overhead.
  - Wait for app to touch pages? May lose opportunity to increase TLB coverage.

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
  - persistence of file cache pages
  - scattered wired (non-pageable) pages
- Contiguity: contended resource
- OS must:
  - use contiguity restoration techniques
  - trade off impact of contiguity restoration against superpage benefits

Design

- Now look in detail at Navarro et al.'s design decisions for:
  - Allocation
  - Promotion
  - Demotion
  - Fragmentation control

Superpage allocation

Use preemptible reservations

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

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
- Q: What is a memory object to the OS?
**Allocation: reservation size**

Opportunistic policy

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

---

**Allocation: managing reservations**

- largest unused (and aligned) chunk
- best candidate for preemption at front:
  - reservation whose most recently populated frame was populated the least recently

---

**Incremental promotions**

Promotion policy: opportunistic

- Superpage is created whenever any superpage-sized and aligned extent within a reservation is fully populated.

---

**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
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
  - \[ \text{write} \rightarrow \text{dirty} \rightarrow \text{clean} \]
- Re-promote (incrementally) as other pages are dirtied

Fragmentation control

- Low contiguity: modified page daemon
  - restore contiguity
    - move clean, inactive pages to the free list
    - minimize impact
      - prefer pages that contribute the most to contiguity
      - keep contents for as long as possible (even when part of a reservation: if reactivated, break reservation)
- Cluster wired pages

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 \((200^3\ \text{matrix})\): 55%
  - 1000x1000 matrix transpose: 655%
- 30%+ in 8 out of 35 benchmarks
**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

- Source code and more info at:
  - www.cs.rice.edu/~jnavarro/superpages