Practical, transparent operating system support for superpages

삼성전자 DMC 공학과
곽승진
2010.07.19(월)
TLB coverage trend

- TLB coverage as percentage of main memory
How to increase TLB coverage

• Typical TLB coverage 1 MB 이하

• Benefit: Increase TLB coverage
  – without increasing in TLB size
  – no internal fragmentation
  – no increase in TLB size
Superpage란

- 큰 사이즈의 memory page
  - 현재 CPU 대부분에서 지원
- Normal page보다 큰 page 묶음
  - 2의 n승으로 증가
  - TLB entry에 하나만 존재
  - Superpage는 동일한 protection attribute를 가져야 함.
  - Contiguous하고 aligned되여야 함(physical and virtual memory)
  - 하나의 superpage에는 같은 protection attributes가 있어야 함.
  - 하나의 reference bit, 하나의 dirty bit이 존재
Superpage TLB

Virtual memory

- Base page entry (size=1)
- Superpage entry (size=4)

TLB

Physical memory

Virtual address

Physical address

Alpha: 8, 64, 512KB; 4MB

Itanium: 4, 8, 16, 64, 256KB; 1, 4, 16, 64, 256MB
Superpage Issues

• Inappropriate use of large superpage
  – Footprint를 크게 만들 수 있음
  – Physical memory requirement를 증가 시킴
  – Paging traffic 증가
  – 이런 I/O cost는 superpage 이익보다 크게 작용할 수 있음.
Superpage Issues

• OS는 전 장의 단점을 극복하고자
  – page size의 mixture 사용이 필요함.
    • Multiple page size의 문제점
      – 물리적 메모리 fragmentation의 문제
      – Superpage의 미래 기회의 상실
    • Sustained performance를 위해
      – OS가 fragment를 관리해야 함.
Superpage Issues

- Allocation

How / when / what size to allocate?
Superpage Issues

- OS later wish to create a superpage for the object, already allocated pages may require relocation
  - i.e. physical copying
- 극복 방법?
  - Reservation-based allocation
Superpage Issues

• Fragmentation control
  – Memory fragmented
    • 실제 multiple page sizes를 사용하기 때문에
    • scattered *wired* (non-pageable) pages
  – Contiguity: contended resource

– **OS 관리 필요**
  • 효과적인 fragmenting available 물리적 메모리의 combine
  • use contiguity restoration techniques
  • trade off impact of contiguity restoration against superpage benefits
Superpage Issues

- **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 a page to touch?
  - Forcibly populate pages? May incur I/O cost or increase internal fragmentation.
  - May to increase TLB
Superpage Issues

• Demotion: convert a superpage into 보다 작은 superpage or base page

• 언제?
  – Process가 더 이상 superpage의 모든 portion을 사용 하는 activate가 아닐 때
  – Memory pressure가 unused base page 제거를 요청 할 때

• 문제?
  – H/W는 single reference bit만을 관리
  – OS가 superpage가 사용되는 portiong을 알기 어려움
Superpage Issues

- Eviction
  - 언제?
    - Memory pressure가 unused base page 제거를 요청할 때
    - Evicted page가 나중에 fault in되면 superpage는 전과 같은 방법으로 생성됨.
  - 문제
    - Dirty page가 paged out되었을 때 복잡
      - H/W는 오직 하나의 dirty bit을 관리
      - 일부가 clean해도 Superpage는 전부가 flush out되어야 함
Design

• Combinatio of techniques
  – Buddy allocator
    • Available physical memory is classified into contiguous regions of different size and managed
  – Multi-list
    • Track partially used memory reservations
    • Help in choosing reservation for preemption
  – Population map
    • Keep track of memory allocation in each memory object
Design
(reservation-based allocation)

• Page is accessed by a program and no mapping exists in the page table → OS’s page fault handler is invoked

• Handler attempt to allocate to locate the associate page in main memory
  – If not resident
    • Available page frame is allocated
    • Contents are either zero-filled or fetched from the paging device
    • Appropriate mapping is entered into the page table
Design
(reservation-based allocation)

• Our system: Prefixed superpage size for the region encompassing the base page whose access caused the page fault
Design
(preferred superpage size policy)

• Issue
  – Policy used to choose the desired superpage size
    • This decision is usually made early in a process’s execution
      hard to predict its future behaviour
  – Superpage size is too large
    • Decision will be later overridden by preempting the initial reservation
  – Superpage size is too small
    • Decision cannot be reverted without relocating pages
      → choose the maximum superpage size that can be effectively used in an object
Design
(preferred superpage size policy)

• Issue
  – Memory objects are fixed
    • Such as code segments, memory-mapped file,
    • Desired reservation size is the largest, aligned superpage
    • Does no overlap with existing reservation or allocated pages
    • Does not reach beyond the end of the object
  – Dynamically sized memory objects
    • Stack, heap → can grow one page at a time
    • Avoid wastage of contiguity for small objects
      – Size of superpage is limited to the current size of the object
Design
(preempting reservations)

• Free physical memory $\rightarrow$ scarce or excessively fragmented
• No extent of frames with the desired size is available
  – Refusing the allocation $\rightarrow$ reserving a smaller extent than desired
  – Preempting an existing reservation that has enough unallocated frames
  – Out policy: reservation is preempted: most recent page allocated occurred least recently, among all candidate reservations
Desig
(Fragmentation control)

• 필요성: Allocating physical memory in contiguous extents of multiple sizes leads to fragmentation of main memory

• Buddy allocator → coalescing of available memory regions

• Page replacement daemon → under persistent memory pressure, modify to perform contiguity-aware page replacement.
Design
(Incremental promotions)

- Promotion 발생: the smallest superpage size as soon as the population count corresponds to that size → population count reached the next larger superpage size
- Issue → artificially inflate the memory footprint of app.
- We design
  - Promote only regions that are fully populated by the app
  - Why?
    • Observe that most applications populate their address space densely and relatively early in their execution
Design
(Incremental promotions)
Design
(Spectulative demotions)

- Demotion 발생: a side-effect of page replacement
- When?
  - Page daemon select a base page that is part of a superpage for eviction → Demotion is increase → superpage first demoted to next smaller size
  - Protection attribute changed on part of superpage

- system may periodically demote active superpages spectulatively
- Why?
  - Determine if the superpage is still being used actively in its entirely
  - To find unused base pages and evict them

- Page daemon reset the reference bit of superpage base page
  - If memory press, the page daemon demotes the superpage that contains that base page
Design
(Paging out dirty superpage)

• Why?
  – When dirty superpage need to be written to diskLarge, OS doesn’t know dirty bit information for individual base pages

• Issue: partially dirty superpage. The performance degradation due to this superfluous I/O

• To prevent this issue
  – We denote clean superpages whenever a process attempt to write into them and only repromote those base pages to superpage if all base pages are dirty
Design
(Paging out dirty superpage)

• Inferring dirty base page using hash digests
  – Clean memory page read from disk \(\rightarrow\) take crypto hash of that page is computed and recorded.
  – If partial dirty set of base pages is promoted to a superpage
  – If clean superpage becomes dirty, all its constituent base pages are considered dirty

• When page is flushed out
  – Hash of each base page is recomputed and compared to determine if it was actually modified and must be written to disk

• Can reduce these overhead
  – Don’t compute hashes still there is a partially dirty superpage \(\rightarrow\) fully-dirty superpage and unpromoted base pages do not need to be hashed
  – Remote hashing cost from critical path by performing it in the an idle loop only
Design
(Multi-list reservation scheme)

• Reservation list keep track of reserved page frame extents that are not fully populated
• One reservation list of each page size supported (Except the largest page size)
• Each reservation appears
  – The list corresponding to the size of the largest free extent that can be obtained if the reservation is preempted
  – i.e. the largest unallocated block of the reservation: if the reservation is 512KB and 65KB of it have not been allocated, that reservation will appear in the 64KB reservation list)
• Reservation within each reserving list are kept sorted in order of most recent page frame allocations (most recent at tail, least recent at head)
• When system has to preempt a reservation of a certain size, it chooses the reservation at the head of the list for that size
Design
(Multi-list reservation scheme)

- When a reservation is preempted, it is broken up into smaller extents. ➔ Unpopulated extents referred to buddy allocator

- Partially populated extents put on appropriate reservation list
  - i.e. if there is 512KB reservation at head of 64KB reservation list, then it’s broken up into 8 * 64KB extents
  - If any of those extents are partially unallocated they are individually put on the reservation list representing the largest unallocated portion
  - If any of the 64KB extents are fully populated, they are not put on any reservation list

- If system needs contiguous region of free memory
  - Buddy allocator
  - Reservation list
    - First ask buddy allocator for 64KB
    - No! then preempt first reservation in 64KB reservation list
    - If list is empty ➔ preempt first reservation in 512 KB list
    - If that is empty ➔ trying larger size then if still fails
    - Try buddy allocator for smallest page size
    - If this fails try to preempt first reservation on smallest superpage size reservation list
Design
(Population MAP)

• **Why?**
  – Keep track of allocated base pages within each memory object

• **Goals?**
  – Used by OS to discover if page frame already reserved for a faulting page
  – When allocating contiguous regions used to detect overlapping regions
  – Assist in making page promotion decisions
  – When preempting a reservation, help in identifying unallocated regions of a reservation

• **Needs**
  – Efficient lookups
  – Why? It is queried on every page fault

• **Use of radix tree in which level corresponds to a page size**
  – Root = maximum superpage size supported by the H/W
  – Each subset = next smaller superpage size
Design
(Population MAP)

• Reserved frame lookup
  – On page fault, virtual address of the fault page is rounded down to a multiple of the largest page size,
  – Converted to the corresponding memory object, page index tuple
  – Hashed to determine the root of the population map
  – From root tree is traversed to locate the reserved page frame (if there is one)

• Overlap avoidance
  – Above procedure yields no reserved frame
  – Attempt to make a reservation
  – Maximum size \( \rightarrow \) does not overlap with previous reservation or allocation given by the first node in the path from the root whose somepop counter is 0
Design
(Population MAP)

- Promotion decision
  - After page fault is serviced
  - Promotion is attempted at the first node on the path from the root to the faulting page
    - Fully populated
    - Has an associated reservation
  - Attempt succeeds only if the faulting process has the pages mapped with uniform protection attribute and dirty bit

- Preemption assistance
  - When Reservation is preempted it is broken into smaller chunks that need to be freed or reinserted in the reservation list
  - Allocation status = population counts in the superpage map node to which the reservation refers.
Evaluation

• Platform

• Alpha 21264 processor at 500 MHz;
• four page sizes: 8KB base pages, 64KB, 512KB and 4MB superpages;
• fully associative TLB with 128 entries for data and 128 for instructions;
• software page tables, with firmware-based TLB loader;
• 512MB RAM;
• 64KB data and 64KB instruction L1 caches, virtually indexed and 2-way associative;
• 4MB unified, direct-mapped external L2 cache.
Evaluation

- **Workloads**
  - **CINT2000**: SPEC CPU2000 integer benchmark suite [7].
  - **CFP2000**: SPEC CPU2000 floating-point benchmark suite [7].
  - **Web**: The httpd web server [15] servicing 50000 requests selected from an access log of the CS departmental web server at Rice University. The working set size of this trace is 238MB, while its data set is 3.6GB.
  - **Image**: 90-degree rotation of a 800x600-pixel image using the popular open-source ImageMagick tools [8].
  - **Povray**: Ray tracing of a simple image.
  - **Linker**: Link of the FreeBSD kernel with the GNU linker.
  - **C4**: An alpha-beta search solver for a 12-ply position of the connect-4 game, also known as the ffourstones benchmark.
  - **Tree**: A synthetic benchmark that captures the behaviour of processes that use dynamic allocation for a large number of small objects, leading to poor locality of reference. The benchmark consists of four perations performed randomly on a 50000-node red-black tree: 50% of the operations are lookups, 24% insertions, 24% deletions, and 2% traversals. Nodes on the tree contain a pointer to a 128-byte record. On insertions a new record is allocated and initialized; on lookups and traversals, half of the record is read.
  - **SP**: The sequential version of a scalar pentadiagonal uncoupled equation system solver, from the NAS Parallel Benchmark suite [1]. The input size corresponds to the “workstation class” in NAS’s nomenclature.
  - **FFTW**: The Fastest Fourier Transform in the West [5] with a 200x200x200 matrix as input.
  - **Matrix**: A non-blocked matrix transposition of a 1000x1000 matrix.
Evaluation

• Common desktop app’s improvement
  – Linker(gnuld), bzip2 significant performance improvements

• Web
  – Cannot create enough superpages in spite of its large 315MB footprint
  – Why?
    • Web accesses a large number of small files
    • System does not attempt to build superpages that span multiple memory objects.

• Some app. Create a significant number of large superpages
  – FFTW → 60 superpages of size 4MB
  – Next section FFTW → good use of large superpages
Evaluation

• Best case benefits due to superpage
  – When free memory is plentiful and non-fragmentation
  – TLB miss reduction usually above 95%

• Table 1
  – Speedup
    • Computed against the unmodified system using the mean elapsed runtime for three runs after an initial warm-up run
    • CINT2000 & CFP2000 improved in SPECint2000 and SPECfp2000
  – Superpage requirements of each of these apps.
  – Percentage data TLB miss reduction with superpage
    • Most data TLB misses are virtually eliminated by superpage
    • Almost 100%
  – Among 35 benchmark \( \rightarrow \) 18 shows improvement over 5%, 10 show over 25%
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Superpage usage</th>
<th></th>
<th></th>
<th></th>
<th>Miss reduc (%)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 KB</td>
<td>64 KB</td>
<td>512 KB</td>
<td>4 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CINT2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gzip</td>
<td>204</td>
<td>22</td>
<td>21</td>
<td>42</td>
<td>80.00</td>
<td>1.007</td>
</tr>
<tr>
<td>vpr</td>
<td>253</td>
<td>29</td>
<td>27</td>
<td>9</td>
<td>99.96</td>
<td>1.383</td>
</tr>
<tr>
<td>gcc</td>
<td>1209</td>
<td>1</td>
<td>17</td>
<td>35</td>
<td>70.79</td>
<td>1.013</td>
</tr>
<tr>
<td>mcf</td>
<td>206</td>
<td>7</td>
<td>10</td>
<td>46</td>
<td>99.97</td>
<td>1.676</td>
</tr>
<tr>
<td>crafty</td>
<td>147</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>99.33</td>
<td>1.036</td>
</tr>
<tr>
<td>parser</td>
<td>168</td>
<td>5</td>
<td>14</td>
<td>8</td>
<td>99.92</td>
<td>1.078</td>
</tr>
<tr>
<td>eon</td>
<td>297</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>1.000</td>
</tr>
<tr>
<td>perl</td>
<td>340</td>
<td>9</td>
<td>17</td>
<td>34</td>
<td>96.53</td>
<td>1.019</td>
</tr>
<tr>
<td>gap</td>
<td>267</td>
<td>8</td>
<td>7</td>
<td>47</td>
<td>99.49</td>
<td>1.017</td>
</tr>
<tr>
<td>vortex</td>
<td>280</td>
<td>4</td>
<td>15</td>
<td>17</td>
<td>99.75</td>
<td>1.112</td>
</tr>
<tr>
<td>bzip2</td>
<td>196</td>
<td>21</td>
<td>30</td>
<td>42</td>
<td>99.90</td>
<td>1.140</td>
</tr>
<tr>
<td>twolf</td>
<td>238</td>
<td>13</td>
<td>7</td>
<td>0</td>
<td>99.87</td>
<td>1.032</td>
</tr>
<tr>
<td>CFP2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wupw</td>
<td>219</td>
<td>14</td>
<td>6</td>
<td>43</td>
<td>96.77</td>
<td>1.009</td>
</tr>
<tr>
<td>swim</td>
<td>226</td>
<td>16</td>
<td>11</td>
<td>46</td>
<td>98.97</td>
<td>1.034</td>
</tr>
<tr>
<td>nggrid</td>
<td>282</td>
<td>15</td>
<td>5</td>
<td>13</td>
<td>98.39</td>
<td>1.000</td>
</tr>
<tr>
<td>appblr</td>
<td>1927</td>
<td>1647</td>
<td>90</td>
<td>5</td>
<td>93.53</td>
<td>1.020</td>
</tr>
<tr>
<td>mesa</td>
<td>246</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>99.14</td>
<td>0.985</td>
</tr>
<tr>
<td>galgel</td>
<td>957</td>
<td>172</td>
<td>68</td>
<td>2</td>
<td>99.80</td>
<td>1.289</td>
</tr>
<tr>
<td>art</td>
<td>163</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>99.55</td>
<td>1.122</td>
</tr>
<tr>
<td>equake</td>
<td>236</td>
<td>2</td>
<td>19</td>
<td>9</td>
<td>97.56</td>
<td>1.015</td>
</tr>
<tr>
<td>facerec</td>
<td>376</td>
<td>8</td>
<td>13</td>
<td>2</td>
<td>98.65</td>
<td>1.062</td>
</tr>
<tr>
<td>ammnp</td>
<td>237</td>
<td>7</td>
<td>21</td>
<td>7</td>
<td>98.53</td>
<td>1.080</td>
</tr>
<tr>
<td>lucas</td>
<td>314</td>
<td>4</td>
<td>36</td>
<td>31</td>
<td>99.90</td>
<td>1.280</td>
</tr>
<tr>
<td>fnm3d</td>
<td>500</td>
<td>17</td>
<td>27</td>
<td>22</td>
<td>96.77</td>
<td>1.000</td>
</tr>
<tr>
<td>sixtr</td>
<td>793</td>
<td>81</td>
<td>29</td>
<td>1</td>
<td>87.50</td>
<td>1.043</td>
</tr>
<tr>
<td>apsi</td>
<td>333</td>
<td>5</td>
<td>5</td>
<td>47</td>
<td>99.98</td>
<td>1.827</td>
</tr>
<tr>
<td>Web</td>
<td>30623</td>
<td>5</td>
<td>143</td>
<td>1</td>
<td>18.67</td>
<td>1.019</td>
</tr>
<tr>
<td>Image</td>
<td>163</td>
<td>1</td>
<td>17</td>
<td>7</td>
<td>75.00</td>
<td>1.228</td>
</tr>
<tr>
<td>povray</td>
<td>136</td>
<td>6</td>
<td>17</td>
<td>14</td>
<td>97.44</td>
<td>1.042</td>
</tr>
<tr>
<td>link</td>
<td>6317</td>
<td>12</td>
<td>29</td>
<td>7</td>
<td>85.71</td>
<td>1.326</td>
</tr>
<tr>
<td>c4</td>
<td>76</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>95.65</td>
<td>1.360</td>
</tr>
<tr>
<td>tree</td>
<td>207</td>
<td>6</td>
<td>14</td>
<td>1</td>
<td>97.14</td>
<td>1.503</td>
</tr>
<tr>
<td>sp</td>
<td>151</td>
<td>103</td>
<td>15</td>
<td>0</td>
<td>99.55</td>
<td>1.193</td>
</tr>
<tr>
<td>fftw</td>
<td>160</td>
<td>5</td>
<td>7</td>
<td>60</td>
<td>99.59</td>
<td>1.549</td>
</tr>
<tr>
<td>matrix</td>
<td>198</td>
<td>12</td>
<td>5</td>
<td>3</td>
<td>99.47</td>
<td>7.546</td>
</tr>
</tbody>
</table>

Table 1: Speedups and superpage requirements when plenty of memory is available.
Evaluation

• Benefits from multiple superpage sizes
  – Table 2,3 present
    • Result → best superpage size depends on the app.
    • i.e. 64KB for SP, 512KB for vpr, 4MB for FFTW
    • Why? While some app only benefit from large superpages, others are too small to fully populate large superpages.
    • To use large superpages with small applications, the population threshold for promotion could be lowered. (cf Sec 4.5)
    • OS would have to populate regions that are only partially mapped by the app. → enlarge the footprint, slightly change the OS semantics
    • Extreme case → mcf percentage speedup when the system gets to choose among several sizes more than doubles the speedup with any single size
    • Some apparent anomalies → different speedup with same TLB miss reduction (Linker) → why? Coarse granularity of the Alpha processor’s TLB miss counter(512K misses)
Evaluation

• Mesa
  – Small performance degradatin of 1.5%
  – Why? Our allocator does not differentiate zeroed-out page from other free pages
  – OS allocates a page that need to be subsequently zeroed out
  – Buddy allocator ignores this hint
  – Estimate the cost of this omission by comparing base sysetm performance with and without the zeroed-page feature → average penalty 0.9% ~ 1.7%

• Side effect of using superpages
  – Subsumes page coloring
  – FreeBSD and other OS Technique use to reduce cache conflicts in physical-addressed and especially in direct-mapped cache
  – Our speedup results factor out the effect of page-coloring
  – Why? Superpage virtually contiguous pages map to physically contiguous frames → automatically map to consecutive locations in a physically mapped cache.
## Table 2: Speedups with different superpage sizes.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>64KB</th>
<th>512KB</th>
<th>4MB</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT2000</td>
<td>1.05</td>
<td>1.09</td>
<td>1.05</td>
<td>1.11</td>
</tr>
<tr>
<td>vpr</td>
<td>1.28</td>
<td>1.38</td>
<td>1.13</td>
<td>1.38</td>
</tr>
<tr>
<td>mcf</td>
<td>1.24</td>
<td>1.31</td>
<td>1.22</td>
<td>1.68</td>
</tr>
<tr>
<td>vortex</td>
<td>1.01</td>
<td>1.07</td>
<td>1.08</td>
<td>1.11</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.14</td>
<td>1.12</td>
<td>1.08</td>
<td>1.14</td>
</tr>
<tr>
<td>CFP2000</td>
<td>1.02</td>
<td>1.08</td>
<td>1.06</td>
<td>1.12</td>
</tr>
<tr>
<td>galgel</td>
<td>1.28</td>
<td>1.28</td>
<td>1.01</td>
<td>1.29</td>
</tr>
<tr>
<td>lucas</td>
<td>1.04</td>
<td>1.28</td>
<td>1.24</td>
<td>1.28</td>
</tr>
<tr>
<td>apsi</td>
<td>1.04</td>
<td>1.79</td>
<td>1.83</td>
<td>1.83</td>
</tr>
<tr>
<td>Image</td>
<td>1.19</td>
<td>1.19</td>
<td>1.16</td>
<td>1.23</td>
</tr>
<tr>
<td>Linker</td>
<td>1.16</td>
<td>1.26</td>
<td>1.19</td>
<td>1.32</td>
</tr>
<tr>
<td>C4</td>
<td>1.30</td>
<td>1.34</td>
<td>0.98</td>
<td>1.36</td>
</tr>
<tr>
<td>SP</td>
<td>1.19</td>
<td>1.17</td>
<td>0.98</td>
<td>1.19</td>
</tr>
<tr>
<td>FFTW</td>
<td>1.01</td>
<td>1.00</td>
<td>1.55</td>
<td>1.55</td>
</tr>
<tr>
<td>Matrix</td>
<td>3.83</td>
<td>7.17</td>
<td>6.86</td>
<td>7.54</td>
</tr>
</tbody>
</table>

## Table 3: TLB miss reduction percentage with different superpage sizes.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>64KB</th>
<th>512KB</th>
<th>4MB</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpr</td>
<td>82.49</td>
<td>98.66</td>
<td>45.16</td>
<td>99.96</td>
</tr>
<tr>
<td>mcf</td>
<td>55.21</td>
<td>84.18</td>
<td>53.22</td>
<td>99.97</td>
</tr>
<tr>
<td>vortex</td>
<td>46.38</td>
<td>92.76</td>
<td>80.86</td>
<td>99.75</td>
</tr>
<tr>
<td>bzip2</td>
<td>99.80</td>
<td>99.09</td>
<td>49.54</td>
<td>99.90</td>
</tr>
<tr>
<td>galgel</td>
<td>98.51</td>
<td>98.71</td>
<td>0.00</td>
<td>99.80</td>
</tr>
<tr>
<td>lucas</td>
<td>12.79</td>
<td>96.98</td>
<td>87.61</td>
<td>99.90</td>
</tr>
<tr>
<td>apsi</td>
<td>9.69</td>
<td>98.70</td>
<td>99.98</td>
<td>99.98</td>
</tr>
<tr>
<td>Image</td>
<td>50.00</td>
<td>50.00</td>
<td>50.00</td>
<td>75.00</td>
</tr>
<tr>
<td>Linker</td>
<td>57.14</td>
<td>85.71</td>
<td>57.14</td>
<td>85.71</td>
</tr>
<tr>
<td>C4</td>
<td>95.65</td>
<td>95.65</td>
<td>0.00</td>
<td>95.65</td>
</tr>
<tr>
<td>SP</td>
<td>99.11</td>
<td>93.75</td>
<td>0.00</td>
<td>99.55</td>
</tr>
<tr>
<td>FFTW</td>
<td>7.41</td>
<td>7.41</td>
<td>99.59</td>
<td>99.59</td>
</tr>
<tr>
<td>Matrix</td>
<td>90.43</td>
<td>99.47</td>
<td>99.47</td>
<td>99.47</td>
</tr>
</tbody>
</table>
Evaluation

• Sustained benefits in the long term
  – Conventional system
    • Can be subject to memory fragmentation even under moderately complex workloads
    • i.e. run instance of grep," emacs, netscape and kernel compilation on a freshly booted system → 15min → observe severe fragmentation
  – Our system
    • Using technique Fragmentation control & Contiguity-aware page daemon
    • To evaluate these method → fragment the system memory by running a web server and feeding it with requests from same access log as before. → the file-backed memory pages accessed by the web server persist in memory → reduce available contiguity to a minimum.
Evaluation

• Sequential Execution
  – After the request from the trace have been serviced → run the FFTW four times in sequence
  – Why? To see how quickly the system recovers just enough contiguous memory to build superpages and perform efficiently
  – Fig 4. compares the performance of two contiguity restoration techniques.
  – Cache → treat all cache pages as available
  – Daemon → out implementation of contiguity-aware page replacement and wired page clustering.

![Bar chart showing speedup with FFTW runs for Cache and Daemon techniques.](image)

Figure 4: Two techniques for fragmentation control.
Conclusion

• Superpages user
  – 30% improvement
  – transparently realized
  – low overhead
• Contiguity restoration is necessary
  – sustains benefits
  – low impact
• Multiple page sizes are important
  – scales to very large superpages