Design and Implementation of Power-Aware Virtual Memory

DMC 공학과
최영범
Contents

- Motivation
- Goals
- Basic Idea
- Implementation
- Evaluation
- Results
- Conclusion
- Future works
Motivation

- Improvement in fabrication technology
- Demand for higher performance in highly-integrated system
- Battery technology falls further behind
Motivation

- Previous Architecture / OS Energy studies
  - Managing power level
  - Processor voltage and clock scaling
  - Mems–based storage
Memory System Power Consumption

- Laptop power budget
- Handheld power budget

- Laptop: memory is small percentage of total power budget
- Handheld: low power processor, memory is more important
Goals

- Use power management feature in current memory technology
- Put individual memory devices into low power modes dynamically

- Using a novel power-aware virtual memory
  - Reduce from 4.1W to 0.5–2.7W based on Rambus
- But, a hardware bug in chipset

- Advanced technique
  - Reduce to 0.2–1.7W
BackGround

- Dram architecture

- Resynchronization cost
**Background**

- RDDram architecture

![RDDram Architecture Diagram](image)

<table>
<thead>
<tr>
<th>Power Level</th>
<th>Power</th>
<th>Active Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attention</td>
<td>313</td>
<td>Refresh, clock, row, col decoder</td>
</tr>
<tr>
<td>Standby</td>
<td>225</td>
<td>Refresh, clock, row decoder</td>
</tr>
<tr>
<td>Nap</td>
<td>11</td>
<td>Refresh, clock</td>
</tr>
<tr>
<td>Powerdown</td>
<td>7</td>
<td>Refresh</td>
</tr>
</tbody>
</table>
Basic Idea

- Power state transitions under SW control (not HW controller)
- Treated explicitly as memory hierarchy
  - A process’s active set of nodes set higher power state
  - Active nodes set Standby mode, all other nodes to Nap mode
- NUMA (Non-Uniform Memory Access) Layer
  - Share memory in multi processor system
Basic Idea

- Size of active node set in kept small by grouping process’s pages in nodes together – “energy footprint”
  - Page mapping – viewed as NUMA layer
  - Active nodes set $\alpha$, put on preferred nodes $\rho$

<table>
<thead>
<tr>
<th>count</th>
<th>$n_0$</th>
<th>$n_1$</th>
<th>...</th>
<th>$n_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_0$</td>
<td>108</td>
<td>2</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_n$</td>
<td>193</td>
<td>240</td>
<td>4322</td>
<td></td>
</tr>
</tbody>
</table>
Implementation

- Each node’s power level can be separately controlled
  => Total memory power reduced
- Selecting nodes may incur resynchronization cost
  => Masked by the context switch
Implementation

- Problem
  - Dlls and files shared by multiple processes become scattered all over memory to process’s active nodes
  - large energy footprints

<table>
<thead>
<tr>
<th>Process</th>
<th>$\rho$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>syslog</td>
<td>14</td>
<td>0(3) 8(5) 9(51) 10(1) 11(1) 13(3) 14(76)</td>
</tr>
<tr>
<td>login</td>
<td>11</td>
<td>0(12) 8(7) 9(112) 11(102) 12(5) 14(20) 15(1)</td>
</tr>
<tr>
<td>startx</td>
<td>13</td>
<td>0(21) 7(12) 8(3) 9(7) 10(12) 11(25) 13(131) 14(43)</td>
</tr>
<tr>
<td>X</td>
<td>12</td>
<td>0(125) 7(23) 8(47) 9(76) 10(223) 11(19) 12(1928) 13(82)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14(77) 15(182)</td>
</tr>
<tr>
<td>sawfish</td>
<td>10</td>
<td>0(180) 7(5) 8(12) 9(1) 10(278) 13(25) 14(5) 15(233)</td>
</tr>
<tr>
<td>vim</td>
<td>10,15</td>
<td>0(12) 9(218) 10(5322) 14(22) 15(4322)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Implementation

- Solutions
  - DLL Aggregation
    - Special case DLLs by allocating Sequential first touch in low numbered nodes

<table>
<thead>
<tr>
<th>Process</th>
<th>$\rho$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>syslog</td>
<td>14</td>
<td>0(108) 1(2) 11(13) 14(17)</td>
</tr>
<tr>
<td>login</td>
<td>11</td>
<td>0(148) 1(4) 11(98) 15(9)</td>
</tr>
<tr>
<td>startx</td>
<td>13</td>
<td>0(217) 1(12) 13(25)</td>
</tr>
<tr>
<td>X</td>
<td>12</td>
<td>0(125) 1(417) 9(76) 11(793) 12(928) 13(169) 14(15)</td>
</tr>
<tr>
<td>sawfish</td>
<td>10</td>
<td>0(193) 1(281) 10(179) 13(25) 14(11) 15(50)</td>
</tr>
<tr>
<td>vim</td>
<td>10,15</td>
<td>0(12) 1(240) 10(5322) 15(4322) ... ... ...</td>
</tr>
</tbody>
</table>
Implementation

- Solutions
  - Page Migration
    - Kernel thread – kmigrated – running in background when system is idle (every 3s)
    - Scans pages used by each process, migrating if conditions met
      - Private page not on
      - Shared page outside

<table>
<thead>
<tr>
<th>Process</th>
<th>$\rho$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>syslog</td>
<td>14</td>
<td>0(15) 14(125)</td>
</tr>
<tr>
<td>login</td>
<td>11</td>
<td>0(76) 11(183)</td>
</tr>
<tr>
<td>startx</td>
<td>13</td>
<td>0(172) 13(82)</td>
</tr>
<tr>
<td>X</td>
<td>12</td>
<td>0(225) 1(2) 12(2220)</td>
</tr>
<tr>
<td>sawfish</td>
<td>10</td>
<td>0(207) 1(56) 10(436)</td>
</tr>
<tr>
<td>vim</td>
<td>10,15</td>
<td>0(12) 1(240) 10(5322) 15(4322)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Evaluation

- Linux implementation
- Workload
  - 3 Mixes
    - light (editing, browsing, MP3)
    - Poweruser (light + kernel compile)
    - Multimedia (playing mpeg movie)
- Platform – 16 nodes, 512MB of RDRAM
Results

(a) Light workload
(b) Power user workload
(c) Multimedia workload
Results

- Base stand by when not accessing
- On/Off nap when system idle
- PAVM
- PAVM1 – Dll aggregation
- PAVM2 – both DLL aggregation & migration

<table>
<thead>
<tr>
<th></th>
<th>Light</th>
<th>Poweruser</th>
<th>Multimedia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>4100 mW</td>
<td>4118 mW</td>
<td>4230 mW</td>
</tr>
<tr>
<td>On/Off</td>
<td>892 mW</td>
<td>2324 mW</td>
<td>3991 mW</td>
</tr>
<tr>
<td>PAVM</td>
<td>465 mW</td>
<td>986 mW</td>
<td>2687 mW</td>
</tr>
<tr>
<td>PAVM1</td>
<td>397 mW</td>
<td>791 mW</td>
<td>2442 mW</td>
</tr>
<tr>
<td>PAVM2</td>
<td>237 mW</td>
<td>646 mW</td>
<td>1725 mW</td>
</tr>
</tbody>
</table>
Conclusion

- Rudimentary version
  - save 34–89% energy of 16 device RDRAM
- Advanced technique
  - additional 20–50%
- Works with other kinds of power-aware memory devices
Limitation

- DMA
- Background Kernel threads
- Implement on modern operating system
  - Virtual Memory
Future works

- Explore other NUMA techniques.
- Investigate the interaction
  - OS-controlled and hardware-implement
Power Aware Buffer Cache

- PAVM
  - Physical memory
- PABC (Power Aware Buffer Cache)
  - Physical Memory + Buffer Cache

<저전력 버퍼 캐시, 이민, 한국과학기술원>

- PAVM 은 Kernel 모드시 메모리 참조는 고려안함
  - 시스템이 사용하는 메모리가 증가할수록 끌수 없는 시스템 랭크의 수가 증가
Power Aware Buffer Cache

- PAVM is not considered when accessing memory in Kernel mode. This is due to the increase in memory consumption which leads to an increase in the number of system ranks.
Power Aware Buffer Cache

- 기본 디자인
  - \( \alpha (p) \): 프로세스 \( p \)의 랭크 셋
  - \( \beta (i) \): inode에 대한 버퍼 캐시의 랭크 셋
  - \( S \): 시스템 랭크 셋

- Compaction
  - PABC: 랭크 셋의 확장을 항상 허용
  - PABC – mempol1: oom killer(out of memory killer)를 호출하기 직전에 랭크 셋의 확장을 허용
  - PABC – mempol2: 디스크 I/O를 일으키기 직전이 시점에 랭크 셋의 확장을 허용
Power Aware Buffer Cache

- Results
  - 기존 커널에 비해 67%
  - PAVM에 비해 61% 메모리 시스템에서 에너지 절약 가능
Thank You