FLSCHED: A Lockless and Lightweight Approach to OS Scheduler for Xeon Phi

Heeseung Jo | Chonbuk National University
Woonhak Kang | Georgia Institute of Technology
Changwoo Min | Virginia Tech
Taesoo Kim | Georgia Institute of Technology
Motivation

Growth of Manycore Processors

• Processor manufacturers have increased the number of cores

• Manycore processors are now prevalent
  • in all types of computing devices
  • include mobile devices, servers and h/w accelerators

• Intel Xeon Phi has up to 76 cores, 304 threads
Motivation

Intel Xeon Processors vs. Xeon Phi Processors

<table>
<thead>
<tr>
<th></th>
<th>Xeon Processors</th>
<th>Xeon Phi Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>Up to 24 cores</td>
<td>Up to 76 cores</td>
</tr>
<tr>
<td>Threads</td>
<td>Up to 48 threads</td>
<td>Up to 304 threads</td>
</tr>
<tr>
<td>Vector Registers</td>
<td>16 * 512-bit registers</td>
<td>32 * 512-bit registers</td>
</tr>
</tbody>
</table>

- 3.17x more cores
- 6.33x more threads
- 2x more registers
Motivation

Inefficiency of Existing Schedulers

• When CFS scheduler was introduced, 4-core servers were dominant in datacenters

• Now, 32-core servers are standard in datacenters

• Moreover, more than 100 cores are becoming popular
Motivation

Inefficiency of Existing Schedulers

• The revolution of OS schedulers is slow to follow up emerging manycore processors
  • They have various lock primitives
  • Frequent context switches
  • But, these are less important in manycore processors like Xeon Phi

• Due to these issues, we propose the new OS scheduler, FLSCHED
  • Lockless design
  • Less context switches
Motivation

Inefficiency of Existing Schedulers

- Hackbench on a Xeon Phi
- Frequent context switches $\rightarrow$ slower
Motivation

Inefficiency of Existing Schedulers

- Comparison on NAS Parallel Benchmark
- Locks in the schedulers degrade the performance
Design

FLSCHED

• Feather-Like Scheduler

• Designed for manycore processors
  • like Intel Xeon Phi

• Lockless design

• Minimizing the number of context switches
Locklessness

<table>
<thead>
<tr>
<th>Lock types</th>
<th>CORE</th>
<th>CFS</th>
<th>FIFO/RR</th>
<th>FL</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw_spin_lock</td>
<td>16</td>
<td>1</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>raw_spin_lock_irq/irqsave</td>
<td>13</td>
<td>5</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>rcu_read_lock</td>
<td>14</td>
<td>5</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>spin_lock</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>spin_lock_irq/irqsave</td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>read_lock</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>read_lock_irq/irqsave</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mutex_lock</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>65</td>
<td>11</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

- Core scheduler code includes highest number of locks

- FLSCHED is implemented **without locks in itself**
  - by restructuring and optimizing the mechanisms
Design

Locklessness: Comparing to RR

- 2 locks are for the runtime statistics
  - It is NOT critical to make scheduling decisions on Xeon Phi
- 5 locks are to balance the load of cores
  - FLSCHED doesn’t use periodic load balance
- 8 locks are used for bandwidth control mechanism
  - It is not important features for Xeon Phi
- Now, We removed 15 locks
  - Since Xeon Phi processors are mostly used for HPC
Design

Less Context Switches

• FLSCHED delays all settings of the reschedule flag to avoid context switches as many as possible

• Computation throughput is MORE important than responsiveness, and fairness
  • Since Xeon Phi processors are mostly used for HPC
Design

Less Context Switches

• Most of preemption is incurred by priority
  • Priority preemption is NOT crucial for Xeon Phi

• FLSCHED does not immediately perform preemption
  • Instead, FLSCHED moves the location of tasks in runqueues and performs normal task switches in later term
  • Since Xeon Phi processors are mostly used for HPC
Design

Faster and efficient scheduling decision

• Scheduling information updates are minimized
  • To make scheduler faster and more efficient

• Remove “update_curr_fair” function
  • It takes very short time
  • But it is called huge number of times with a spinlock
  • It can be non-negligible overhead in manycore processors

• Instead, FLSCHEDED works based on a given time slice with RR
Design

Faster and efficient scheduling decision

• FLSCHED does not provide 3 scheduling features:
  • Control groups
  • Group scheduling
  • Autogroup scheduling

• These are considered NOT important features for manycore systems like Xeon Phi
  • To get the great performance improvement, sometimes we have to yield small things
Evaluation

Evaluation Environments

• Intel Xeon E5-2699
  • 18 cores
  • 36 threads

• 64 GB main memory

• Intel Xeon Phi 31S1P
  • 57 cores
  • 228 threads
  • 8 GB internal memory
Evaluation

Performance comparison of NAS Parallel Benchmark

- It shows better performance with FLSCHED
Evaluation

Performance comparison of NAS Parallel Benchmark

- Execution time of spinlock while executing NPB

<table>
<thead>
<tr>
<th>NPB program</th>
<th>CFS (%)</th>
<th>FIFO (%)</th>
<th>RR (%)</th>
<th>FL_SCHED (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt</td>
<td>7.29</td>
<td>8.53</td>
<td>8.60</td>
<td>3.05</td>
</tr>
<tr>
<td>cg</td>
<td>10.73</td>
<td>13.61</td>
<td>12.93</td>
<td>4.11</td>
</tr>
<tr>
<td>ep</td>
<td>0.97</td>
<td>0.89</td>
<td>0.91</td>
<td>1.10</td>
</tr>
<tr>
<td>ft</td>
<td>5.34</td>
<td>5.25</td>
<td>5.57</td>
<td>4.04</td>
</tr>
<tr>
<td>is</td>
<td>0.21</td>
<td>0.17</td>
<td>0.18</td>
<td>0.12</td>
</tr>
<tr>
<td>mg</td>
<td>6.84</td>
<td>7.30</td>
<td>7.15</td>
<td>2.85</td>
</tr>
<tr>
<td>sp</td>
<td>8.23</td>
<td>9.95</td>
<td>9.98</td>
<td>3.58</td>
</tr>
<tr>
<td>ua</td>
<td>14.63</td>
<td>15.79</td>
<td>15.46</td>
<td>5.96</td>
</tr>
</tbody>
</table>
Evaluation

Performance comparison of hackbench

• Execution time and number of context switches

One group uses 40 tasks

In X axis,
‘p’ with the number denotes pipe
The other denotes socket
### Evaluation

Performance comparison of hackbench

- **Execution count and time of scheduler functions**

<table>
<thead>
<tr>
<th>Scheduler functions</th>
<th>CFS</th>
<th>FLSCHED</th>
<th>Normalized ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Count</td>
<td>Average time (ns)</td>
<td>Count</td>
</tr>
<tr>
<td>check_preempt</td>
<td>42,184,784</td>
<td>5,058</td>
<td>3,202</td>
</tr>
<tr>
<td>dequeue_task</td>
<td>42,476,857</td>
<td>19,008</td>
<td>10,646</td>
</tr>
<tr>
<td>enqueue_task</td>
<td>42,479,016</td>
<td>17,314</td>
<td>10,792</td>
</tr>
<tr>
<td>pick_next_task</td>
<td>66,951,729</td>
<td>5,261</td>
<td>5,532,392</td>
</tr>
<tr>
<td>pre_schedule</td>
<td>-</td>
<td>-</td>
<td>10,646</td>
</tr>
<tr>
<td>put_prev_task</td>
<td>66,503,232</td>
<td>6,185</td>
<td>10,647</td>
</tr>
<tr>
<td>select_task_rq</td>
<td>42,426,871</td>
<td>10,837</td>
<td>8,031</td>
</tr>
<tr>
<td>set_cpus_allowed</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>task_tick</td>
<td>906,640</td>
<td>13,131</td>
<td>112</td>
</tr>
<tr>
<td>task_waking</td>
<td>42,418,867</td>
<td>2,290</td>
<td>10,792</td>
</tr>
<tr>
<td>update_curr</td>
<td>342,354,453</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

**Total Execution Time:**
- **CFS:** 28.037s
- **FLSCHED:** 11.102s
FLSCHED

• Feather-Like Scheduler
  • Designed for manycore processors like Intel Xeon Phi
  • Lockless design
  • Minimizing the number of context switches

• FLSCHED shows better performance than CFS up to
  • 1.73x for HPC applications
  • 3.12x for micro-benchmarks
Thank you

If you have any questions,
Please contact the first author via email:

Prof. Heeseung Jo
heeseung@jbnu.ac.kr