A Case for NUMA-aware Contention Management on Multicore Systems

Sergey Blagodurov  sergey_blagodurov@sfu.ca
Sergey Zhuravlev  sergey_zhuravlev@sfu.ca
Mohammad Dashti  mohammad_dashti@sfu.ca
Alexandra Fedorova  alexandra_fedorova@sfu.ca

USENIX ATC’11 / Scheduling session
15th of June
An AMD Opteron 8356 Barcelona domain
An AMD Opteron system with 4 domains
Contestion for the shared last-level cache (CA)
Contention for the memory controller (MC)
Contention for the inter-domain interconnect (IC)
Remote access latency (RL)
Isolating Memory controller contention (MC)
Memory Controller (MC) and InterConnect (IC) contention are key factors hurting performance.
Characterization method

- Given two threads, decide if they will hurt each other’s performance if co-scheduled

Scheduling algorithm

- Separate threads that are expected to interfere
Limited observability

- We do not know for sure if threads compete and how severely
- Hardware does not tell us

Trial and error infeasible on large systems

- Can’t try all possible combinations
- Even sampling becomes difficult

A good trade-off: measure LLC Miss rate!

- Assumes that threads interfere if they have high miss rates
- No account for cache contention impact
- Works well because cache contention is not dominant
Sort threads by LLC missrate:  

A  B  X  Y

Goal: isolate threads that compete for shared resources

High contention:

A  B  MC  HT

Memory node 1

Domain 1

Y  X  MC  HT

Memory node 2

Domain 2

Low contention?

A  X  MC  HT

Memory node 1

Domain 1

Y  B  MC  HT

Memory node 2

Domain 2

Migrate competing threads to different domains

Our previous work: an algorithm for UMA systems
Distributed Intensity (DI-Plain)
Failing to migrate memory leaves MC and introduces RL
DI-Plain hurts performance on NUMA systems because it does not migrate memory!
Goal: isolate threads that compete for shared resources and pull the memory to the local node upon migration

Migrate competing threads along with memory to different domains

Solution #1: Distributed Intensity with memory migration (DI-Migrate)
DI-Migrate performs too many migrations for MPI. Migrations are expensive on NUMA systems.
Migrating too frequently causes IC
DI-Migrate:  
- threads sorted by miss rate  
- if array positions change, we migrate thread and memory

DINO:  
- threads sorted by class  
- only migrate if we jump from one class to another

Solution #2: Distributed Intensity NUMA Online (DINO)
Loose correlation between miss rate and degradation, so most migrations will not payoff
Average number of memory migrations per hour of execution (DI-Migrate and DINO)

<table>
<thead>
<tr>
<th></th>
<th>SPEC CPU 2006</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>soplex</td>
<td>milc</td>
<td>lbm</td>
<td>gamess</td>
<td>namd</td>
</tr>
<tr>
<td>DI-Migrate</td>
<td>36</td>
<td>22</td>
<td>11</td>
<td>47</td>
<td>41</td>
</tr>
<tr>
<td>DINO</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SPEC MPI 2007</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>leslie</td>
<td>lamps</td>
<td>GAPgeofem</td>
<td>socorro</td>
<td>lu</td>
</tr>
<tr>
<td>DI-Migrate</td>
<td>381</td>
<td>135</td>
<td>237</td>
<td>340</td>
<td>256</td>
</tr>
<tr>
<td>DINO</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

DINO significantly reduces the number of migrations
DINO results

USENIX ATC’11 / Scheduling session

SPEC CPU 2006

SPEC MPI 2007

LAMP

% improvement over DEFAULT

ibm  namd  omnetpp  milc  ibm2  namd2  omnetpp2  milc2

milc  leslie  GemsFDTD  f1ds  pop  tachyon  lammps  GAPgeofem  socorro  zesmp  lu

Apache #0  Apache #1  Apache #2  Mysql #0  Mysql #1  Mysql #2  Mysql #3
On NUMA systems we need to schedule threads and memory

- Memory Controller contention when memory is not migrated
- Interconnect Contention when memory is migrated too frequently

DINO is the contention-aware scheduling algorithm for NUMA systems that

- migrates the memory along with the application
- eliminates excessive migrations by trying to keep the workload on their old nodes, if possible
- utilizes Instruction Based Sampling to perform partial memory migration of “hot” pages

Summary
Read our Linux Symposium 2011 paper:

“User-level scheduling on NUMA multicore systems under Linux”

Source code is available at:

http://clavis.sourceforge.net

For further information
Any [time for] questions?

A Case for NUMA-aware Contention Management on Multicore Systems

USENIX ATC’11 / Scheduling session