Scheduling on current multi-core clusters

Emílio Francesquini
emilio@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

University of São Paulo, Brazil
Agenda

• Current scenario overview
• Some existing approaches
• Weaknesses of the current approaches
• Some of our ideas to improve the current solutions
Current scenario overview

• Multi-core processors are becoming cheaper and more common every day

• 9 out of the Top 10 (www.top500.org) computers use multi-core processors*
  – 8 of them have more than 2 cores

• Most of the Top 500 (410/500) computers are already clusters

* Source: www.top500.org
Current scenario (cont.)
Current scenario (cont.)

• Memory Bottleneck

Source: www.cs.virginia.edu/stream/
Some existing approaches

How to efficiently use all that computational power?

• Message passing
  • SLURM
    • Heavy use of process pinning

• NUMA/DSM
  • Memory pinning
  • Process pinning
SLURM
Simple Linux Utility for Resource Management

- Open source
- Manages resources and controls queues for exclusively reserved resources
- Allows users to dispatch, to execute and to monitor jobs
- BlueGene/L at LLNL with 106,496 dual-core processors
SLURM (Cont.)

- Has three levels of hierarchy for each processing unit in the system

https://computing.llnl.gov/linux/slurm/
SLURM

- Works with the concept of process pinning
- Has low and high level flags to assert process scheduling to specific cores
- Low-level flags gives more control whereas high-level flags are much more user-friendly
SLURM
Low-level flags

• Allows process pinning to the cores
• User must be aware of the numeration scheme for their system

Block numeration

<table>
<thead>
<tr>
<th></th>
<th>c0</th>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>p1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>p2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>p3</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Cyclic numeration

<table>
<thead>
<tr>
<th></th>
<th>c0</th>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>p1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>p2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>p3</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
SLURM
Low-level flags

• --cpu-bind=
  – mask_cpu
  – map_cpu

• Examples
  – Block numbering
    • srun -n 8 -N 4 -cpu_bind=mask_cpu:0x1,0x4 prog
    • srun -n 8 -N 4 -cpu_bind=map_cpu:0,2 prog
SLURM
High-level flags

• Created to simplify the usage

• Automatically generates the task masks
  – --sockets-per-node=S
  – --cores-per-socket=C
  – --threads-per-socket=T
  – Shortcut: -B S[:C[:T]]

• Example:
  – srun -n 8 -N 4 -B 2:1:1 prog
  – srun -n 8 -N 4 -B 2-2:1-1:1-1 prog
SLURM
Multi-core performance results
[Balle and Palermo, JSSP'07]

• Linpack on 16 cores
  – 4 nodes X 2 sockets X 2 cores

<table>
<thead>
<tr>
<th>Configuration</th>
<th>CPUs</th>
<th>Time (sec)</th>
<th>% Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>No affinity control used</td>
<td>16</td>
<td>467.16</td>
<td></td>
</tr>
<tr>
<td>taskset 0xf</td>
<td>16</td>
<td>481.83</td>
<td>-3.04%</td>
</tr>
<tr>
<td>taskset 0x1; 0x2; 0x4; 0x8</td>
<td>16</td>
<td>430.44</td>
<td>8.53%</td>
</tr>
<tr>
<td>--cpu_bind=map_cpu:0,1,2,3</td>
<td>16</td>
<td>430.36</td>
<td>8.55%</td>
</tr>
<tr>
<td>-B 1:1</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multi-core performance results
[Balle and Palermo, JSSP'07]

- LSDyna – Simulates the nonlinear dynamic response of three-dimensional inelastic structures
- Simulation of three cars collision
- Executed on a 16 core machine
  - 4 nodes X 2 sockets X 2 cores
## SLURM – LSDyna performance results

<table>
<thead>
<tr>
<th>Cores</th>
<th>Nodes</th>
<th>Binding</th>
<th>CPU binding option</th>
<th>Time (sec)</th>
<th>Time (D:H:M:S)</th>
<th>Speedup</th>
<th>% Speedup vs. no binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>No</td>
<td></td>
<td>194,809</td>
<td>2:06:06:49</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>-cpu_bind=map_cpu:0</td>
<td>194,857</td>
<td>2:06:07:37</td>
<td>1.00</td>
<td>-0.02%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>No</td>
<td></td>
<td>104,994</td>
<td>1:05:09:54</td>
<td>1.86</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Yes</td>
<td>-cpu_bind=map_cpu:0,1</td>
<td>110,702</td>
<td>1:06:45:02</td>
<td>1.76</td>
<td>-5.16%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Sockets</td>
<td>-cpu_bind=map_cpu:0,2</td>
<td>104,620</td>
<td>1:05:03:40</td>
<td>1.86</td>
<td>0.36%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>No</td>
<td></td>
<td>102,336</td>
<td>1:04:25:36</td>
<td>1.90</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Yes</td>
<td>-cpu_bind=map_cpu:0</td>
<td>100,266</td>
<td>1:03:51:06</td>
<td>1.94</td>
<td>4.72%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>No</td>
<td></td>
<td>33,616</td>
<td>0:09:20:16</td>
<td>5.80</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>Yes</td>
<td>-cpu_bind=map_cpu:0,1,2,3</td>
<td>31,996</td>
<td>0:08:53:16</td>
<td>6.09</td>
<td>5.06%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>No</td>
<td></td>
<td>28,815</td>
<td>0:08:00:15</td>
<td>6.76</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Yes</td>
<td>-cpu_bind=map_cpu:0,1</td>
<td>28,532</td>
<td>0:07:55:32</td>
<td>6.83</td>
<td>0.99%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Sockets</td>
<td>-cpu_bind=map_cpu:0,2</td>
<td>26,081</td>
<td>0:07:14:41</td>
<td>7.47</td>
<td>10.48%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-B 1:1-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NUMA
Non-Uniform Memory Access

- OpenMP and pthreads
- NUMA support (Linux kernel >= 2.6)
  - Memory pinning
  - Process pinning
- Manual control over memory and process pinning
NUMA

• Techniques
  – First touch initialization/Parallel initialization → no guarantees
  – Memory/Process pinning
    • sched_setaffinity
    • Mbind
      – bind/interleave/preferred
NUMA performance test architecture
[Pousa, Méhaut et al. , WSCAD'08]

- NUMA factor: 2 → 2.5
- 16 Itanium2 at 1.6 GHz
- 64 GBytes of RAM

- NUMA factor: 1.2 → 1.5
- 8 dual-core Opteron at 2.2 GHz
- 32 Gbytes of RAM
NUMA Performance comparison

Ondes 3D

• Application for seismic wave propagation simulation

• Regular data access pattern

Itanium Cluster

Opteron cluster
NUMA Performance comparison

Benchmark NAS

- Simulation of fluid dynamics
- CG Kernel
  - Large memory footprint
  - Irregular data access pattern

Itanium

Opteron
Weaknesses of the current approaches

• Lack of portability
• Not suitable (or even usable) for heterogeneous clusters
• Demands expertise from the developer and the executor (not always the same person)
• Scheduling relies too much on the users
Some of our ideas...

- Profiling
- Dynamic Scheduling using online profiling and profiles obtained from previous runs
- Let the user specify the architecture/topology of his network. But also try to discover what is possible without user intervention
Why?

- To allow the developer to focus on the problem, and not on architectural details
- Portability
- Deal with node idiosyncrasies seamlessly
- We believe the simplicity pays off the eventual losses in performance in most cases
Why? (cont.)

• Application behavioral patterns may change
  – During execution
  – From inputs
  – During its lifecycle
Conclusion

• We've presented a current problem

• Future steps
  – To propose a theoretical model
    • Cache proximity
  – To evaluate it
Thank you!
So why should I be so happy about the future that hardware vendors promise? They think a magic bullet will come along to make multicores speed up my kind of work; I think it’s a pipe dream. No!—that’s the wrong metaphor! "Pipelines" actually work for me, but threads don’t. Maybe the word I want is "bubble."

Donald Knuth