Is it me, or is it the machine?

Judit Gimenez (judit@bsc.es)
Parallel efficiency model

- Parallel efficiency = LB eff * Comm eff

- Parallel efficiency = LB eff * Serialization eff * Transfer eff
My favorite default app for trainings: Lulesh 2.0

- Easy to install and does not require large input files
- Iterative behaviour, well balanced except one region due to instructions unbalance

- Requires a cube number of MPI ranks
  - my target = 27 ranks; no nodes/sockets sized 27 :)
  - No OpenMP
- Expected problem: some extra unbalance due to the unbalanced mapping
Same code, different behavior

<table>
<thead>
<tr>
<th>Code</th>
<th>Parallel efficiency</th>
<th>Communication efficiency</th>
<th>Load Balance efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>lulesh@machine1</td>
<td>90.55</td>
<td>99.22</td>
<td>91.26</td>
</tr>
<tr>
<td>lulesh@machine2</td>
<td><strong>69.15</strong></td>
<td>99.12</td>
<td><strong>69.76</strong></td>
</tr>
<tr>
<td>lulesh@machine3</td>
<td>70.55</td>
<td>96.56</td>
<td>73.06</td>
</tr>
<tr>
<td>lulesh@machine4</td>
<td>83.68</td>
<td>95.48</td>
<td>87.64</td>
</tr>
<tr>
<td>lulesh@machine5</td>
<td><strong>90.92</strong></td>
<td>98.59</td>
<td><strong>92.20</strong></td>
</tr>
<tr>
<td>lulesh@machine6</td>
<td>73.96</td>
<td>97.56</td>
<td>75.81</td>
</tr>
<tr>
<td>lulesh@machine7</td>
<td>75.48</td>
<td><strong>88.84</strong></td>
<td>84.06</td>
</tr>
<tr>
<td>lulesh@machine8</td>
<td>77.28</td>
<td>92.33</td>
<td>83.70</td>
</tr>
<tr>
<td>lulesh@machine9</td>
<td>88.20</td>
<td>98.45</td>
<td>89.57</td>
</tr>
<tr>
<td>lulesh@machine10</td>
<td>81.26</td>
<td>91.58</td>
<td>88.73</td>
</tr>
</tbody>
</table>

Warning::: Higher parallel efficiency does not mean faster!

Huge variability and worse than expected. Can I explain why?
## Same code, same machine... still different behaviors

<table>
<thead>
<tr>
<th>MPI</th>
<th>Playing with MPI @ A</th>
<th>Parallel efficiency</th>
<th>Communication efficiency</th>
<th>Load Balance efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPI</td>
<td></td>
<td>85.65</td>
<td>95.09</td>
<td>90.07</td>
</tr>
<tr>
<td>MPT</td>
<td></td>
<td>70.55</td>
<td>96.56</td>
<td>73.06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Playing with binding @ B configuration</th>
<th>Parallel efficiency</th>
<th>Communication efficiency</th>
<th>Load Balance efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>81.26</td>
<td>91.58</td>
<td>88.73</td>
</tr>
<tr>
<td>binding</td>
<td>75.10</td>
<td>97.44</td>
<td>77.07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Playing with both @ C configuration</th>
<th>Parallel efficiency</th>
<th>Communication efficiency</th>
<th>Load Balance efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>BullMPI / default</td>
<td>84.00</td>
<td>93.41</td>
<td>89.35</td>
</tr>
<tr>
<td>OpenMPI / default</td>
<td>79.45</td>
<td>98.35</td>
<td>80.73</td>
</tr>
<tr>
<td>OpenMPI / binding</td>
<td>82.10</td>
<td>95.08</td>
<td>86.35</td>
</tr>
<tr>
<td>BullMPI / binding</td>
<td>85.15</td>
<td>96.59</td>
<td>88.18</td>
</tr>
</tbody>
</table>
The expected

- Balance between nodes and across sockets

Using 2 nodes x 2 sockets
- 3 sockets with 7 ranks, 1 socket with 6 ranks — small time unbalance

Less frequent than expected!

Parallel eff. 90.55%
Comm 99.22%
LB 91.26%

6 guys with more resources
The good

- 27 fits in a node
- Alternate between sockets
- Small IPC variability in the 2 main regions

Parallel eff. 90.92%
Comm 98.59%
LB 92.20%
The ugly

• Same code! but now two trends (one per node)

Parallel eff. 69.15%
Comm 99.12%
LB 69.76%

Slow node □ significant lower IPC for almost all the regions

But all nodes are equal!
The ugly

Clock frequency sanity check

Insight checking hardware counters differences

<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
<th>Cluster 4</th>
<th>Cluster 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Duration</td>
<td>347038882</td>
<td>218366724</td>
<td>191517484</td>
<td>76087437</td>
<td>57184253</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>4253914</td>
<td>6369746</td>
<td>6360328</td>
<td>2195915</td>
<td>1540034</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>1647293</td>
<td>1984225</td>
<td>1864857</td>
<td>448603</td>
<td>258565</td>
</tr>
<tr>
<td>PAPI_L3_TCM</td>
<td>945265</td>
<td>1399329</td>
<td>1390967</td>
<td>160690</td>
<td>111273</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>881368852</td>
<td>880939731</td>
<td>881303626</td>
<td>275104547</td>
<td>274379604</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>900031546</td>
<td>566347103</td>
<td>496687512</td>
<td>197363737</td>
<td>148307597</td>
</tr>
<tr>
<td>RESOURCESTALLS:SB</td>
<td>257407726</td>
<td>154854806</td>
<td>106684134</td>
<td>74990020</td>
<td>38463538</td>
</tr>
<tr>
<td>RESOURCESTALLS:ROB</td>
<td>3191720</td>
<td>3170735</td>
<td>2722761</td>
<td>2684869</td>
<td>562054</td>
</tr>
<tr>
<td>RESOURCESTALLS:RS</td>
<td>43484959</td>
<td>63717139</td>
<td>63017342</td>
<td>38768779</td>
<td>32699838</td>
</tr>
</tbody>
</table>

Guess: Memory problem – confirmed by sysadmin tests!
The unbalanced

- Balance between nodes not between sockets

What lulesh does

What the machine does running lulesh

Balance between nodes 9 per node
Fill first a socket 6 + 3

Parallel eff. 79.45%
Comm 98.35%
LB 80.73%
The unbalanced

Two main regions suffer the penalty of the different socket occupancy

Most frequent behaviour!
The bipolar

• MPI modify the computing phases behavior!

Parallel eff. 70.55%
Comm 96.56%
LB 73.06%

Parallel eff. 85.65%
Comm 95.09%
LB 90.07%
The bipolar

... because it can select a different mapping / binding
The undecided

- Same number of instructions per iteration showing a noisy behavior w.r.t time

... use to correspond to a system that does unnecessary process migrations
The braking

- But not always noise is caused by migrations

Parallel eff. 75.48%
Comm 88.84%
LB 84.06%

node id
The braking

• Clue: the noise affects to all processes within a node

Histogram of cycles/us

The OS of each node is reducing for a while the clock frequency asynchronously!
The mingling

• Even 64 may have problems

Two trends with a perfectly balanced number of MPI ranks on each node
The mingling

Histogram of useful duration

... without specifications, a request of 64 cores always allocates two nodes BUT one node of each type.
Conclusions

• As code developer, better not to assume machines will do a good job running your code because you did a good job programming your application

• As performance analyst, do not assume where are the bottlenecks, be open minded and equipped with flexible tools (like Paraver ;)

As Bruce Lee said “Be water my friend!”