Application Performance Monitoring on Hopper: Integrated Performance Monitoring

David Skinner
deskinner@lbl.gov
• One of many: There are lots of good vendor supplied tools, we encourage their use
• Adaptable: If you can’t get what you need from those we can adapt IPM based on your feedback
• Performance Portability: IPM provides long-term continuity to performance data between machines, applications, ERCAP etc.
Performance is Relative

- **To your goals**
  - Time to solution, $T_{\text{queue}} + T_{\text{run}}$
  - Efficient use of allocation
  - Do FLOPs even matter?

- **To the**
  - application code
  - input deck
  - machine type/state

In general the first bottleneck wins.
IPM can help find first order bottlenecks
What can IPM do?

- **Provide high level performance numbers with tiny overhead**
  - To get an initial read on application runtimes
  - For allocation/reporting, ERCAP perf data
  - To check the performance weather (not an issue on XE knock wood)

- **What’s going on overall in my code?**
  - How much comp, comm, I/O?
  - Where to start with optimization?

- **How is my load balance?**
  - Domain decomposition vs. concurrency (M work on N tasks)
1) Do “module load ipm”, link with $IPM, then run normally
2) Upon completion you get

```bash
##IPM2v0.xx##################################################
#####
# command : ./fish -n 10000
# start   : Tue Feb  8 11:05:21 2011   host      : nid06027
# stop    : Tue Feb  8 11:08:19 2011   wallclock  : 177.71
# mpi_tasks : 25 on 2 nodes            %comm    : 1.62
# mem [GB]  : 0.24                       gflop/sec : 5.06
...```

Maybe that’s enough. If so you’re done.
Have a nice day 😊
Generalities in Scalability and Performance
Scaling: definitions

- Scaling studies involve changing the degree of parallelism. Will we be change the problem also?
- **Strong scaling**
  - Fixed problem size
- **Weak scaling**
  - Problem size grows with additional resources
- **Speed up** = $T_s / T_p(n)$
- **Efficiency** = $T_s / (n * T_p(n))$

Be aware there are multiple definitions for these terms.
The scalability landscape

Why does efficiency drop?

- Serial code sections → Amdahl’s law
- Surface to Volume → Communication bound
- Algorithm complexity or switching
- Communication protocol switching

3D complex-complex FFTW (N=n\times n\times n)

<table>
<thead>
<tr>
<th>MPI Tasks</th>
<th>MLOP/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

\[ n = 100 \rightarrow 1000 \]
Load Balance: cartoon

Unbalanced:

Task 1
Task 2
Task 3
Task 4

Balanced:

Task 1
Task 2
Task 3
Task 4

Universal App

Sync
Flop
I/O

Time saved by load balance
Load (Im)balance

Communication Time: 64 tasks show 200s, 960 tasks show 230s

MPI ranks sorted by total communication time
while(1) {
    do_flops(N_i);
    MPI_Alltoall();
    MPI_Allreduce();
}
Simple Stuff: What’s wrong here?

Communication

% of MPI Time

Communication Event Statistics (100.00% detail)

<table>
<thead>
<tr>
<th>Event</th>
<th>Buffer Size</th>
<th>Ncalls</th>
<th>Total Time</th>
<th>Min Time</th>
<th>Max Time</th>
<th>%MPI</th>
<th>%Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Allreduce</td>
<td>8</td>
<td>3278848</td>
<td>124132.547</td>
<td>0.000</td>
<td>114.920</td>
<td>59.35</td>
<td>16.88</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>0</td>
<td>35173439489</td>
<td>43439.102</td>
<td>0.000</td>
<td>41.961</td>
<td>20.77</td>
<td>5.91</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>98304</td>
<td>13221888</td>
<td>15710.953</td>
<td>0.000</td>
<td>3.586</td>
<td>7.51</td>
<td>2.14</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>196608</td>
<td>13221888</td>
<td>5331.236</td>
<td>0.000</td>
<td>5.716</td>
<td>2.55</td>
<td>0.72</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>589824</td>
<td>206848</td>
<td>5166.272</td>
<td>0.000</td>
<td>7.265</td>
<td>2.47</td>
<td>0.70</td>
</tr>
</tbody>
</table>
Some more specific examples
Example

- We can use your own code in the hands-on session
- In prep for that here is a worked example with the Sharks and Fish code
  - A Newtonian particle pushing code w/ predator-prey dynamics between sharks and fish. Used in UCB CS267
  - See a glimpse here:
    http://www.leinweb.com/snackbar/wator/
A scaling study w/ 100 Fish

Make sure you’re profiling the actual code
A real scaling study

Sharks and Fish (MPI)

Straight lines good
Discontinuities bad
Off the rails

Sharks and Fish (MPI)

2 sec @ 24 cores

0.8 sec @ 500+ cores
Too much communication
Summary
1) Do “module load ipm”, link with $IPM, then run normally

2) Upon completion you get

```bash
#IPM2v0.xx###################################################################
#
# command   : ./fish -n 10000
# start     : Tue Feb 08 11:05:21 2011   host      : nid06027
# stop      : Tue Feb 08 11:08:19 2011   wallclock : 177.71
# mpi_tasks : 25 on 2 nodes            %comm     : 1.62
# mem [GB]   : 0.24                     gflop/sec : 5.06
...
```

We value your feedback on how to extend or improve IPM

help@nersc.gov
The transition to many-core has brought complexity to the once orderly space of hardware performance counters. NERSC, UCB, and UTK are all working on improving things.

IPM on XE, currently just the banner is in place. We think PAPI is working (recently worked with Cray on bug fixes).
Thanks!

Questions about IPM?

deskinner@lbl.gov