Software-based Performance Counters in Open MPI

David Eberius
9th JLESC Workshop
Knoxville, TN
4/16/2019
Tool Support in Open MPI

High-Level Information

The PMPI Interface
- MPI standard compliant
- Full support in Open MPI

Low-Level Information

The MPI_T Interface
- MPI standard compliant
- Little to no MPI_T pvars

The Peruse Interface
- Not MPI standard compliant
- No widespread adoption
Proposed Solution

Software-based Performance Counters (SPCs)

- Instrumentation of Open MPI with integer counters
  - Low-level information (ex. number of unexpected messages)
  - High-level information (ex. number of times MPI functions are called)

- Fine-grained performance metrics
  - Updated as soon as the information is available

- Registered as MPI_T pvars
Types of SPCs

• Regular Counters
  • Increment/decrement each time a certain event happens

• Timer Counters
  • Increment with the number of cycles spent in some event
  • Must be converted to time before reporting

• High Watermark Counters
  • Keep track of the highest a given value has been
  • Resets to current value level upon reading

• Bin Counters
  • Stores multiple categories of the same counter in bins
  • i.e. number of messages sent eagerly vs. otherwise
Reducing PMPI Burden

• Overlapping capabilities handled by SPCs
  • Counts of function calls

• Reduces overhead if PMPI is not used
  • No additional function call overhead

• Frees PMPI interface for other uses
Majority of counter overhead comes from timer counters.
Using cycles significantly reduces overhead.
Accessing SPCs

• MPI_T pvars
  • All SPCs can be accessed through MPI_T

• Use mmap interface to access SPCs
  • One SPC pvar gives path to XML file used for mmap interface
  • One data file per rank stored locally
  • Optionally record periodic snapshots
  • Accessible through RMA operations

• Print to stdout
  • Not advisable for large runs
  • Post-run analysis report (in-progress)
SPC Use Examples

- Identify performance bottlenecks
  - More MPI usage on certain ranks
  - Higher unexpected messages or queue lengths

- Identify program phases
  - Record snapshots and analyze differences in counters in fixed intervals
  - Possibly use this to increase snapshot frequency

- Help Open MPI developers diagnose issues in Open MPI
  - Out of sequence message issue
Identifying Open MPI OOS Problem

- Used SPCs to identify problematic number of out of sequence messages in vader and openib BTLs.

Table 1: Results of the MADNESS moldft tests using five water molecules. The counter values are the average of 10 runs with 18 threads per run of the simulation.

<table>
<thead>
<tr>
<th>Counter</th>
<th>openib</th>
<th>vader</th>
<th>tcp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Time (sec)</td>
<td>626.41</td>
<td>440.95</td>
<td>518.54</td>
</tr>
<tr>
<td>OMPI UNEXPECTED</td>
<td>126,339.4</td>
<td>21,654.5</td>
<td>14,868.5</td>
</tr>
<tr>
<td>OMPI OUT OF SEQUENCE</td>
<td>1,222,397.6</td>
<td>134,631.0</td>
<td>0.0</td>
</tr>
<tr>
<td>OMPI_MATCH_TIME</td>
<td>282,910.2</td>
<td>369,343.7</td>
<td>251,844.7</td>
</tr>
<tr>
<td>OMPI_OOS_MATCH_TIME</td>
<td>317,157.8</td>
<td>32,742.9</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 2: Results of the pairwise benchmark with a window size of 256 messages, message size of 64 bytes, and an iteration count of 100 using the openib BTL.

<table>
<thead>
<tr>
<th>Threads</th>
<th>Message Rate (msg/sec)</th>
<th>Receives</th>
<th>OOS Messages</th>
<th>% OOS</th>
<th>Wall Time (us)</th>
<th>Match Time (us)</th>
<th>OOS Match Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>601,773.54</td>
<td>56,320</td>
<td>16,633</td>
<td>29.53%</td>
<td>85,598</td>
<td>9,634</td>
<td>9,875</td>
</tr>
<tr>
<td>4</td>
<td>476,174.73</td>
<td>112,640</td>
<td>47,216</td>
<td>41.92%</td>
<td>218,807</td>
<td>34,312</td>
<td>51,196</td>
</tr>
<tr>
<td>8</td>
<td>162,458.95</td>
<td>225,280</td>
<td>112,813</td>
<td>50.08%</td>
<td>1,260,863</td>
<td>96,465</td>
<td>729,187</td>
</tr>
</tbody>
</table>
LAMMPS Example

• *indent* Example
  • Has more communication than most examples

• Used the mmap interface
  • Record snapshots every 0.5 seconds

• Run parameters
  • 2 nodes
  • 2 CPU sockets per node
  • 40 processes (20 procs/node)
LAMMPS Example (Bytes User)
LAMMPS Example (Match Time)
LAMMPS Example (Unexpected)
Current/Future Work Open for Collaboration

• Program phase detection
  • Looking at differences in counters over time to identify program phases

• Post-mortem analysis and report
  • Give suggestions for improvements

• Tool adoption of SPCs
  • Anyone interested?

• Survey of CORAL mini-apps
  • Show the MPI usage over time
  • Analyze queue usage (unexpected messages, matching time, etc...)