Using the Intel® Trace Analyzer and Collector on Hybrid MPI Programs

James Tullos
Technical Consulting Engineer
Intel® Cluster Tools
james.a.tullos@intel.com
Agenda

Introduction to the Intel® Trace Analyzer and Collector
- Data collection considerations for hybrid programs

Interpretation of results

Demonstration

Additional Resources
Objectives of the lesson

Increase familiarity with Intel® Trace Analyzer and Collector capabilities

Learn to collect trace data for hybrid programs

Learn to interpret trace to improve application performance via increasing parallelism and load balancing
Agenda

Introduction to the Intel® Trace Analyzer and Collector
- Data collection considerations for hybrid programs
Interpretation of results
Demonstration
Additional Resources
Intel® Trace Analyzer and Collector

Helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots

Features

- Event based approach
- Low overhead, excellent scalability
- Comparison of multiple profiles
- Powerful aggregation and filtering functions
- Fail-safe MPI tracing
- API to instrument user code
- MPI Correctness Checker
- Idealizer and Application Imbalance Diagram
- Coming in 9.0: Performance Assistant
## Multiple Methods for Data Collection

<table>
<thead>
<tr>
<th>Collection Mechanism</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run with <code>–trace</code> or preload trace collector library.</td>
<td>Automatically collects all MPI calls, requires no modification to source, compile, or link.</td>
<td>No user code collection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link with <code>–trace</code>.</td>
<td>Automatically collects all MPI calls.</td>
<td>No user code collection. Must be done at link time.</td>
</tr>
<tr>
<td>Compile with <code>–tcollect</code>.</td>
<td>Automatically instruments all function entries/exits.</td>
<td>Requires recompile of code.</td>
</tr>
<tr>
<td>Add API calls to source code.</td>
<td>Can selectively instrument desired code sections.</td>
<td>Requires code modification.</td>
</tr>
</tbody>
</table>
Data Location

Stored in a set of stf (structured tracefile) files

For large runs, data can quickly grow unmanageable
- Really depends on number of instrumented calls
- Filters available to reduce collected data

Files are stored by default in launching folder
Considerations for Hybrid Programs

Can only trace instrumented calls

- If non-master threads have no MPI calls, using –trace will not detect the threads
- If using –tcollect, threads must call a subroutine in order to be correctly found.
  - Still detectable, but with no function entry/exit, data is less useful
- If using the Trace Collector API, must instrument inside threaded region

Does not correlate its thread numbering with thread IDs.

For best results, ensure that something in each thread is instrumented

- If using the API, you can record actual thread ID if desired
Agenda

Introduction to the Intel® Trace Analyzer and Collector
  - Data collection considerations for hybrid programs

Interpretation of results

Demonstration

Additional Resources
Full View Event Timeline
Identifying Ranks and Threads

Each execution unit has an identifier P<rank> T<thread>

- If only one thread detected in a rank, appears as P<rank> only
- T<thread> does not necessarily correspond to the thread ID used by threading models
  - But thread 0 will always match

All messages (point-to-point and collective) will have identifying data included in message data
Analyzing Load Balance in the Function Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>TSelf</th>
<th>TSelf</th>
<th>TTotal</th>
<th>#Calls</th>
<th>TSelf/Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Waitall</td>
<td>5.98177</td>
<td>5.98177</td>
<td>2886</td>
<td>2.07268e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 0</td>
<td>526.502e-3</td>
<td>526.502e-3</td>
<td>62</td>
<td>8.49197e-3</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 3</td>
<td>1.30751</td>
<td>1.30751</td>
<td>413</td>
<td>3.16588e-3</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 2</td>
<td>1.63967</td>
<td>1.63967</td>
<td>634</td>
<td>2.58623e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 10</td>
<td>83.939e-3</td>
<td>83.939e-3</td>
<td>37</td>
<td>2.26862e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 8</td>
<td>6.325e-3</td>
<td>6.325e-3</td>
<td>3</td>
<td>2.10833e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 2</td>
<td>1.81285</td>
<td>1.81285</td>
<td>929</td>
<td>1.9514e-3</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 18</td>
<td>39.624e-3</td>
<td>39.624e-3</td>
<td>24</td>
<td>1.651e-3</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 5</td>
<td>46.21e-3</td>
<td>46.21e-3</td>
<td>31</td>
<td>1.49065e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 12</td>
<td>6.433e-3</td>
<td>6.433e-3</td>
<td>5</td>
<td>1.2866e-3</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 7</td>
<td>50.584e-3</td>
<td>50.584e-3</td>
<td>43</td>
<td>1.7637e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 4</td>
<td>120.227e-3</td>
<td>120.227e-3</td>
<td>112</td>
<td>1.07346e-3</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 6</td>
<td>152.497e-3</td>
<td>152.497e-3</td>
<td>161</td>
<td>947.186e-6</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 11</td>
<td>20.29e-3</td>
<td>20.29e-3</td>
<td>22</td>
<td>922.273e-6</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 8</td>
<td>62.196e-3</td>
<td>62.196e-3</td>
<td>72</td>
<td>863.833e-6</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 3</td>
<td>20.453e-3</td>
<td>20.453e-3</td>
<td>66</td>
<td>309.894e-6</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 14</td>
<td>219e-6</td>
<td>219e-6</td>
<td>3</td>
<td>73e-6</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 23</td>
<td>71e-6</td>
<td>71e-6</td>
<td>1</td>
<td>71e-6</td>
<td></td>
</tr>
<tr>
<td>Process 0 Thread 25</td>
<td>142e-6</td>
<td>142e-6</td>
<td>2</td>
<td>71e-6</td>
<td></td>
</tr>
<tr>
<td>Process 1 Thread 21</td>
<td>340e-6</td>
<td>340e-6</td>
<td>5</td>
<td>68e-6</td>
<td></td>
</tr>
</tbody>
</table>
Analyzing Load Balance in the Quantitative Timeline
Agenda

Introduction to the Intel® Trace Analyzer and Collector

- Data collection considerations for hybrid programs

Interpretation of results

Demonstration

Additional Resources
Demonstration Highlights

Watch for the Quantitative Timeline, what does it show?

Given the following information about the code and the trace shown, what would you do to improve the code?

- 3D code, two arrays are sent in each axis to neighboring ranks
- All MPI calls occur in OMP SINGLE regions without NOWAIT
- All MPI_Irecv calls occur at the beginning of the parallel section
- All MPI_Isend calls occur immediately after the arrays for that direction are computed
- All MPI_Waitall calls occur almost immediately before the received data is used (followed by OMP BARRIER, then data is used)

Questions are good!
Ideas from Demonstration

The Quantitative Timeline shows

- Very few threads are in simultaneous MPI calls

Possible code improvements:

- Allow multiple threads to make simultaneous MPI calls where safe
  - Make the MPI_IRecv and MPI_Isend sections NOWAIT
  - Split the MPI_IRecv and MPI_Isend sections into multiple sections, one for each call
  - Move the MPI_WaitAll calls earlier, add to a SINGLE NOWAIT, and put an OMP BARRIER to synchronize threads
Agenda

Introduction to the Intel® Trace Analyzer and Collector
- Data collection considerations for hybrid programs

Interpretation of results

Demonstration

Additional Resources
Additional Resources

Intel® Trace Analyzer and Collector Product Page
- [http://www.intel.com/go/itac](http://www.intel.com/go/itac)

Intel® Clusters and HPC Technology Forum

Intel® Developer Zone
- [http://software.intel.com](http://software.intel.com)

Intel® Premier Support
- [https://businessportal.intel.com](https://businessportal.intel.com)
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804