Intel® Trace Analyzer and Collector

James Tullos
james.a.tullos@intel.com
Technical Consulting Engineer
Intel® Cluster Tools
What is the Intel® Trace Analyzer and Collector?
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
- Powerful aggregation and filtering functions
- Idealizer
Intel® Trace Analyzer and Collector Overview

Source Code → Compiler → Objects → Linker → Binary → Runtime → Output

API and -tcollect

Intel® Trace Collector

Trace File (.stf)

Intel® Trace Analyzer
Collecting a Trace
Intel® Trace Analyzer and Collector Prerequisites

Set the Intel® Trace Analyzer and Collector environment (per user)

```bash
# source /opt/intel/itac/8.1.2.033/bin/itacvars.sh

  – Identical for Host and coprocessor

For Intel® Xeon Phi™:

• Ensure that the library is accessible on the coprocessor
  – This can be via NFS or via manually copying

```bash
# scp /opt/intel/itac/8.1.2.033/mic/slib/libVT.so node0-mic0:/lib64/libVT.so
```
Intel® Trace Analyzer and Collector Usage with Intel® Xeon Phi™ coprocessor

Recommended

Run with -trace flag (without linkage) to create a trace file

- MPI+Offload
  
  # mpirun -trace -n 2 ./test

- Coprocessor only and Symmetric
  
  # export I_MPI_MIC=enable
  
  # mpirun -trace -f mpi_hosts -n 2 ~/test_hello[.MIC]

Flag “-trace” will implicitly pre-load the libVT.so (which finally calls libmpi.so to execute the MPI call)
Intel® Trace Analyzer and Collector Usage with Intel® Xeon Phi™ coprocessor

Compilation Support

Compile and link with the “–trace” flag

```
# mpiicc -trace -o test_hello test.c
# mpiicc -trace -mmic -o test_hello.MIC test.c
- Linkage of tracing library
```

or, Compile with –tcollect flag

```
# mpiicc -tcollect -o test_hello test.c
# mpiicc -tcollect -mmic -o test_hello.MIC test.c
- Linkage of tracing library
- Will do full instrumentation of your code, i.e. All user functions will be visible in the trace file
- Maximal insight, but also maximal overhead
```

or, Use the tracing API to manually instrument your code

Run your Intel® MPI program as per usual, without “–trace” flag

```
# mpirun -f mpi_hosts -n 2 ~/test_hello[.MIC]
```
Analyzing the Trace
Intel® Trace Analyzer and Collector

Compare the event timelines of two communication profiles
Blue = computation
Red = communication

Chart showing how the MPI processes interact
A Chart is a numerical or graphical diagram.
**Timelines: Event Timeline**

- Get impression of program structure
- Display functions, messages and collective operations for each process/thread along time-axis
- Retrieval of detailed event information

<table>
<thead>
<tr>
<th>P0</th>
<th>Message: COM/MPI</th>
<th>Application: COM/MPI</th>
<th>Application: COM/MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P2</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P3</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P4</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P5</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P6</td>
<td>Message: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
<tr>
<td>P7</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
<td>Application: COM/MPI</td>
</tr>
</tbody>
</table>
Timelines: Qualitative Timeline

• Find patterns and irregularities

• Display attributes of functions, messages or collective operations as they occur for any process/thread
**Timelines: Quantitative Timeline**

- Get impression on parallelism and load balance
- Show for every function how many threads/processes are currently executing it
Profiles: Flat Function Profile

• Statistics about functions
Profiles: Call-Tree and Call-Graph

- Function statistics including calling hierarchy
  - Tree: call-stack
  - Graph: calling dependencies
Communication Profiles

- Statistics about point-to-point or collective communication

- Matrix supports grouping by several attributes in each dimension
- Axes: Sender, Receiver, Data volume per msg, Tag, Communicator, Type
- Attributes: Count, Bytes transferred, Time, Transfer rate
**View**

Helps navigating through the trace data and keep orientation

Every View can contain several Charts

A View on a file is defined by a triplet of
- time-span
- set of threads
- set of functions

All Charts follow changes to View (e.g. zooming)

Timelines are correctly aligned along time
View - zooming
Grouping and Aggregation

Allow analysis on different levels of detail by aggregating data upon group-definitions

Functions and threads can be grouped hierarchically
- Function Groups and Thread Groups

Arbitrary nesting is supported
- Functions/threads on the same level as groups
- User can define his/her own groups

Aggregation is part of View-definition
- All charts in a View adapt to requested grouping
- All charts support aggregation
Aggregation Example
Tagging & Filtering

Help concentrating on relevant parts

Avoid getting lost in huge amounts of trace data

Define a set of interesting data
  • E.g. all occurrences of function x
  • E.g. all messages with tag y on communicator z

Combine several filters:
  Intersection, Union, Complement

Apply it
  • Tagging: Highlight messages
  • Filtering: Suppress all non-matching events
Tagging Example
Filtering Example
Load Balance
Improving Load Balance:
Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
8 MPI procs x 28 OpenMP threads

Too high load on Host = too low load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
24 MPI procs x 8 OpenMP threads

Too low load on Host = too high load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
16 MPI procs x 12 OpenMP threads

Perfect balance
Host load = Coprocessor load
Ideal Interconnect Simulator
Ideal Interconnect Simulator (Idealizer)
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive

Load imbalance
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
- 257 = MPI_Alltoallv
- 506 = User_Code
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
257 = MPI_Alltoallv
506 = User_Code

Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
257 = MPI_Alltoallv
506 = User_Code

Same MPI_Alltoallv

Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Online Resources

Intel® Trace Analyzer and Collector product page
www.intel.com/go/traceanalyzer

Intel® Clusters and HPC Technology forums

Intel® Xeon Phi™ Coprocessor Developer Community
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 © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk 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