Application Performance Snapshot

Performance Overview at Your Fingertips
Application Performance Snapshot (APS)

High-level overview of application performance
Merges previous Application Performance Snapshot with MPI Performance Snapshot
Identify primary optimization areas and next steps in analysis
Extremely easy to use
Informative, actionable data in clean HTML report
  • Also includes recommendations for next steps in analysis
Detailed reports available via command line
Scales to large jobs
APS Usage

Setup Environment
$ source <APS_Install_dir>/apsvars.sh

Run Application
$ mpirun <mpi options> aps <application and args>

Generate Report on Results
$ aps --report <result folder>

Generate advanced CL reports on Results
$ aps report --option <result folder>
APS HTML Report

Application Performance Snapshot

**Elapased Time**

121.39s

**SP FLOPS**

50.98

**CPI**

0.68

*MAX 0.81, MIN 0.65*

**MPI Time**

53.74% of Elapased Time

(65.23s)

**MPI Imbalance**

11.03% of Elapased Time (13.39s)

**TOP 5 MPI Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waitall</td>
<td>37.35</td>
</tr>
<tr>
<td>Send</td>
<td>6.48</td>
</tr>
<tr>
<td>Barrier</td>
<td>5.52</td>
</tr>
<tr>
<td>recv</td>
<td>3.70</td>
</tr>
<tr>
<td>Scatterv</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**I/O Bound**

0.00%

*AVG 0.00*, *PEAK 0.00*

**OpenMP Imbalance**

0.43% of Elapsed Time (6.7s)

**Memory Footprint**

**Memory Stalls**

14.70% of pipeline slots

**FPU Utilization**

0.39%

*SP FLOPS per Cycle 0.08 Out of 32.00*

*Vector Capacity Usage 25.84%*

*FP Instruction Mix % of Total FP Insts: 3.54% *% of L2 Read: 3.54% *% of L1D Read: 0.00% *% of L1I Read: 96.46%*

*FP ArithMem Red Inst Ratio 0.00%*

*FP ArithMem WW Inst Ratio 0.00%*
Overview shows all areas and relative impact on code performance

Provides recommendation for next step in performance analysis

Your application is MPI bound.
This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Current Run</th>
<th>Target</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Time</td>
<td>53.74%</td>
<td>&lt;10%</td>
<td></td>
</tr>
<tr>
<td>OpenMP Imbalance</td>
<td>0.43%</td>
<td>&lt;10%</td>
<td></td>
</tr>
<tr>
<td>Memory Stalls</td>
<td>14.70%</td>
<td>&lt;20%</td>
<td></td>
</tr>
<tr>
<td>FPU Utilization</td>
<td>0.30%</td>
<td>&gt;50%</td>
<td></td>
</tr>
<tr>
<td>I/O Bound</td>
<td>0.00%</td>
<td>&lt;10%</td>
<td></td>
</tr>
</tbody>
</table>
HTML Report Breakdown – Parallel Runtimes

- **MPI Time**
  - How much time was spent in MPI calls
  - Average by ranks with % of Elapsed time
  - Available for MPICH-based MPIs

- **MPI Imbalance**
  - Unproductive time spent in MPI library waiting for data
  - Available for Intel MPI

- **OpenMP Imbalance**
  - Time spent at OpenMP Synchronization Barriers normalized by number of threads
  - Available for Intel OpenMP
HTML Report Breakdown - Memory

- Memory stalls measurement with breakdown by cache and DRAM
- NUMA ratio
- KNL: back-end stalls with L2-demand access efficiency
HTML Report Breakdown – Vectorization

- FPU Utilization based on HW-event statistics with
  - Breakdown by vector/scalar instructions
  - Floating point vs memory instruction ratio

- KNL: SIMD Instr. per Cycle
  - Scalar vs. vectorized instructions
Intel® VTune™ Profiler

Profiling MPI and Hybrid MPI+Threads Applications
Using Intel® VTune™ Profiler on MPI programs

- Run VTune underneath MPI
  - Results are grouped into one result per node
    - <result folder>.<node name>
  - Within result, ranks indicate rank number

$ mpirun <mpi args> amplxe-cl <vtune args> -- <application and args>
Easier Multi-Rank Analysis of MPI + OpenMP

Tune hybrid parallelism using ITAC + VTune Profiler

Tune OpenMP performance of high impact ranks in VTune Profiler

<table>
<thead>
<tr>
<th>Top OpenMP Processes by MPI Communication Spin Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>This section lists processes sorted by MPI Communication Spin time. The lower MPI Communication Spin time, the critical path of MPI application execution. Explore OpenMP efficiency metrics by MPI processes laying on the critical path.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process</th>
<th>PID</th>
<th>MPI Communication Spinning</th>
<th>(%)</th>
<th>OpenMP Potential Gain</th>
<th>(%)</th>
<th>Serial Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>heart_demo (rank 7)</td>
<td>32394</td>
<td>5.122s</td>
<td>8.1%</td>
<td>19.923s</td>
<td>31.3%</td>
<td>2.875s</td>
</tr>
<tr>
<td>heart_demo (rank 10)</td>
<td>32397</td>
<td>5.463s</td>
<td>8.6%</td>
<td>19.482s</td>
<td>30.6%</td>
<td>2.867s</td>
</tr>
<tr>
<td>heart_demo (rank 11)</td>
<td>32398</td>
<td>5.593s</td>
<td>8.8%</td>
<td>20.183s</td>
<td>31.7%</td>
<td>2.873s</td>
</tr>
<tr>
<td>heart_demo (rank 6)</td>
<td>32393</td>
<td>6.264s</td>
<td>9.8%</td>
<td>19.429s</td>
<td>30.5%</td>
<td>2.869s</td>
</tr>
<tr>
<td>heart_demo (rank 5)</td>
<td>32395</td>
<td>6.595s</td>
<td>10.4%</td>
<td>19.379s</td>
<td>30.5%</td>
<td>2.864s</td>
</tr>
</tbody>
</table>

Ranks sorted by MPI Communication Spins – ranks on the critical path are on the top.

Process names link to OpenMP metrics.

Detailed OpenMP metrics per MPI ranks.

Per-rank OpenMP Potential Gain and Serial Time metrics.
Intel® Inspector

- Analysis Process
- Launch Intel® Inspector
  - Use mpirun
  - List your app as a parameter
- Results organized by MPI rank
- Review results
  - Graphical user interface
  - Command line report

Find errors earlier when they are less expensive to fix
Using Intel® Inspector with MPI

Use the command-line tool under the MPI run script to gather report data

```bash
$ mpirun -n 4 inspxe-cl -r my_result -collect mi1 -- ./test
```

Argument Sets can be used for more control

- Only collect data on certain ranks
- Different collections or options on different ranks

A unique results directory is created for each analyzed MPI rank
Launch the GUI and view the results for each rank
Hands-On

Build and Run a Simple MPI Program
Simple MPI Program - Setup

- Intel MPI provides a simple example in C, C++, Fortran, fortran90 and Java under the directory `<impi_install_dir>/test/`

- Copy the file to your working directory:
  
  ```bash
  $ cp /opt/intel/impi/<version>/test/test.c .
  ```

- Ensure your environment is set to use the Intel MPI compiler wrappers:
  
  ```bash
  $ source /opt/intel/impi/<version>/bin64/mpivars.sh
  ```

- You can check that the environment is set properly by checking if the variable `I_MPI_ROOT` is set:
  
  ```bash
  $ echo $I_MPI_ROOT
  ```
Simple MPI Program – Build and Run

- Compile the test program using the provided wrapper:
  - $ mpiicc ./test.c -o test.x

- Note that you can see exactly what the compiler wrapper will use by typing:
  - $ mpiicc --show

- Run the program on a single node. How many cores are available?
  - $ mpirun ./test.x

- Run the program on at least two nodes, and set **I_MPI_DEBUG=4** to check on which node and core each MPI rank run.
Simple MPI Program – Review Questions

- How do compiler scripts match to languages, e.g. which compile script is used for Fortran and which one for C?
  - What is the difference between mpicc and mpiicc?
  - Does mpiifort/mpiicc always take the ifort/icc compiler contained in the same package? Can you force mpiifort to take a different version?
  - What is the strategy Intel MPI Library uses to fill up nodes? What is the default process pinning when we have less ranks than physical core?
  - How would tasks be distributed if we had 8 physical cores per node, 2 nodes, and we only used 12 MPI ranks total?
Hands-On

ITAC Trace of Simple MPI Program
Simple MPI Trace - Setup

- Intel MPI provides a simple example in C, C++, Fortran, fortran90 and Java under the directory `<impi_install_dir>/test/`
- Copy the file to your working directory:
  - `$ cp /opt/intel/impi/<version>/test/test.c`.
- Ensure your environment is set to use Intel MPI and ITAC:
  - `$ source /opt/intel/impi/<version>/bin64/mpivars.sh`
  - `$ source /opt/intel/itac/<version>/bin/itacvars.sh`
- You can check that the environment is set properly by checking if the variables `I_MPI_ROOT` and `VT_ROOT` are set.
Simple MPI Trace – Build and Run (I)

- Compile the test program using the provided wrapper:
  - `$ mpiicc ./test.c -o test_1.x`
- Run the program with a small number of tasks (2 or 4) and collect the trace:
  - `$ mpirun -np 2 -trace ./test_1.x`
Simple MPI Trace – Build and Run (II)

- Copy the executable to change its name:
  - `$ cp ./test_1.x ./test_2.x`

- Set `LD_PRELOAD` to the `libVT.so` location and run with a different number of tasks but without the `–trace` flag:
  - `$ export LD_PRELOAD=$VT_SLIB_DIR/libVT.so`
  - `$ mpirun -np 4 ./test_2.x`
Simple MPI Trace – Build and Run (II)

- Compile the test program using the provided wrapper and the –trace flag:
  - `$ mpiicc -trace ./test.c -o test_3.x`
- This time, make sure we collect to a single trace file:
  - `$ export VT_LOGFILE_FORMAT=STFSINGLE`
- Run the program again:
  - `$ mpirun -np 2 ./test_3.x`
Simple MPI Trace – Visualize Traces

- There are several ways of viewing your results using the ITAC GUI
  - From the system that collected the data – using an X window
  - Copying the trace files to a local Linux* or Windows system
- If you have an X-Window or VNC connection running:
  - `$ traceanalyzer test_3.x.stf`
- If you moved the file to a local system just double-click the stf file
- The Summary opens first – what is the most expensive MPI call?
- Proceed to the next window, which rank uses the most MPI time? [Hint: Load Balance tab, look at “Children of all processes”]
- Make sure to check out the “Event Timeline” within the “Charts” menu
Hands-On

Produce APS Report with Simple MPI Program
Simple MPI APS report - Setup

- Intel MPI provides a simple example in C, C++, Fortran, fortran90 and Java under the directory `<impi_install_dir>/test/`
- Copy the file to your working directory:
  - `$ cp /opt/intel/impi/<version>/test/test.c`.
- Ensure your environment is set to use Intel MPI and APS:
  - `$ source /opt/intel/impi/<version>/bin64/mpivars.sh`
  - `$ source /opt/intel/performance_snapshots/apsvars.sh`
- You can check that the environment is set properly by checking if the variable `I_MPI_ROOT`. 


Simple MPI APS report - Build and Run

- Compile the test program using the provided wrapper:
  - $ mpiicc ./test.c -o test.x

- Run the program with a small number of tasks (2 or 4) and using APS:
  - $ mpirun -np 4 aps ./test.x

- Notice that a new directory named `aps_report_<yyymmdd>` has been generated

- Notice also that instructions are given regarding how to generate a full report

-
Simple MPI APS report - Build and Run

- Generate a text summary report from the command line:
  - `$ aps-report -s ./aps_results_<yyymmmdd>`

- Generate an html report following the instructions provided:
  - `$ aps --report=<working_dir>/aps_result_<yyymmmdd>`

  This will generate the text summary again by default

- Open the html report with any browser and check that all the expected sections are included.
Hands-On

Simple MPI VTune Example
Simple MPI VTune™ Example - Setup

- Obtain the example tarball from the website:
  - https://mantevo.org/downloads/miniFE_ref_2.0.htm

- Ensure your environment is set to use Intel MPI and VTune™ Profiler:
  - $ source /opt/intel/impi/<version>/intel64/bin/mpivars.sh
  - $ source /opt/intel/vtune/amplxe-vars.sh

- You can check that the environment is set properly by checking if the variables `I_MPI_ROOT` and `VTUNE_PROFILER_2020_DIR` are set.
Simple MPI VTune™ Example - Build

- Untar the example:
  - $ tar xzvf sources/miniFE-2.0_ref.tgz
- Change to the new directory:
  - $ cd miniFE-2.0_ref/src
- Build the executable:
  - $ make CC=mpiicc CXX=mpiicpc
- Copy the generated executable miniFE.x to the top-level directory:
  - $ cp ./miniFE.x ../
Simple MPI VTune™ Example - Run

- Now change directory to the top-level directory and perform an hpc-performance collection:
  - `$ cd ../`
  - `$ mpirun -np 4 amplxe-cl -collect hpc-performance -r hpc -- ./miniFE.x nx=100`
- Open newly generated results in the GUI:
  - `$ amplxe-gui ./hpc.<hostname>`
- What is the MPI imbalance for this job?
- Can you identify the slowest MPI rank? [Hint: Bottom-up, Grouping by Process/Function/Threads/Call Stack]
Additional ITAC Capabilities
Zooming
Grouping and Aggregation

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

- Functions and threads can be grouped hierarchically
  - Process Groups and Function 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 and 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
Ideal Interconnect Simulator (Idealizer)

- Helps to figure out application’s imbalance simulating its behavior in the “ideal communication environment”

Actual trace

Idealized Trace

Easy way to identify application bottlenecks
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

P1: MPI_Send
P2: MPI_Recv

Late Send / Early Receive

P1: MPI_Send
P2: MPI_Recv

Load imbalance
Tuning Methods

- Library Tuning (algorithms, fabric parameters)
  - mpitune
  - MPI statistics

- Application Tuning (load balance, MPI/thashed/serial performance)
  - Intel® Trace Analyzer and Collector
  - Intel® VTune™ Profiler XE
  - Application Performance Snapshot
Library Tuning: mpitune

Use the automatic tuning facility to tune the Intel® MPI Library for your cluster or application (done once, may take a long time)

Modes (see mpitune –h for options)

- Cluster-wide tuning

- Application-specific tuning

Creates options settings which are used with the –tune flag
Lightweight Statistics

- Set I_MPI_STATS to a non-zero integer to gather MPI communication statistics (max. 10)
- Change scope with I_MPI_STATS_SCOPE
- Define output file with I_MPI_STATS_FILE
- Bin results using I_MPI_STATS_BUCKET
Lightweight Statistics

- $ export I_MPI_STATS=3
- $ export I_MPI_SCOPE=coll
- $ mpirun -np 8 ./exe

... Communication Activity by actual args

<table>
<thead>
<tr>
<th>Collectives</th>
<th>Operation</th>
<th>Context</th>
<th>Algo</th>
<th>Comm size</th>
<th>Message size</th>
<th>Calls</th>
<th>Cost(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allreduce</td>
<td>1</td>
<td>248</td>
<td>1</td>
<td>8</td>
<td>64</td>
<td>1</td>
<td>0.27</td>
</tr>
<tr>
<td>Barrier</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>Bcast</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>32</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>72</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>56</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>Reduce</td>
<td>1</td>
<td>248</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>248</td>
<td>1</td>
<td>8</td>
<td>72</td>
<td>1</td>
<td>0.32</td>
</tr>
</tbody>
</table>

=================================================================================