Tutorial: Analyzing MPI Applications with MPI Performance Snapshot

MPI Performance Snapshot

Legal Information

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

Contents

1. Overview

MPI Performance Snapshot is a scalable lightweight performance tool for MPI applications. It collects the MPI application statistics, such as communication, activity, load balance, and presents it in an easy-to-read format. Use the collected information for in-depth analysis of the application scalability and performance.

About This Tutorial	This tutorial demonstrates an end-to-end workflow you can ultimately apply to your own applications: • Detect performance issues in your application • Find communication hotspots • Review your application This tutorial uses the `poisson` application as an example. The application source code is available at: `<installdir>/examples/poisson`
Estimated Duration	10-15 minutes.
Learning Objectives	After you complete this tutorial, you should be able to: • Conduct the quick analysis of your application using MPI Performance Snapshot • Improve your application performance
More Resources	Learn more about MPI Performance Snapshot in the [MPI Performance Snapshot User's Guide]

1.1. Prerequisites

Before you start using MPI Performance Snapshot, make sure to install the necessary software and libraries and set up the environment:

1. Install the Intel® Fortran Compiler version 15.0.1 or higher and set up the environment:

```
$ source <compiler_installdir>/bin/compilervars.sh
```

2. Install the Intel® MPI Library version 5.0.3 or higher and set up the environment:

```
$ source <IMPI_installdir>/intel64/bin/mpivars.sh
```

3. Set up the environment for MPI Performance Snapshot, source the script:

```
$ source <ITAC_installdir>/bin/mpsvars.sh
```

2. Analyzing an MPI Application

Step 1: **Prepare for analysis**	Build an application and generate statistics files.
Step 2: **Detect performance issues**	• See the Summary Page • See the Function Summary
Step 3: **Resolve the issues**	• Replace the blocking `SendRecv` function with the non-blocking `Icomm`. • Tune the `Allreduce` function
Step 4: **Check your work**	Rebuild the application and view the results.

2.1. Preparing for Analysis

Complete the steps described in the Prerequisites section.

Copy the contents `<installdir>/examples/poisson` into your working directory. Edit the `inp` file as follows:

```
3200
2 16
```

Build the application by running the `make` command and run the application on two nodes of the cluster:

```
$ make
$ mpirun -mps -n 32 -hosts <node1>,<node2> ./poisson
```

Two statistics files will be generated: `stats.txt` and `app_stat.txt`.

2.2. Viewing the Application Statistics

2.2.1. Viewing the Summary Page

Display the application summary by processing the generated files:

```
$ mps ./stats.txt ./app_stat.txt
```

The output will look as follows:

```
| Summary information
|--------------------------------------------------------------
  Application    : ./poisson
  Number of ranks: 32
```

```
  Used statistics: app_stat.txt, stats.txt
|
  WallClock time :              6.37 sec
| Total application lifetime. The time is elapsed time for the slowest process.
| This metric is the sum of the MPI Time and the Computation time below.
|
      MPI Time:                     2.36 sec            37.56%
|     Time spent inside the MPI library. High values are usually bad.
|     This value is HIGH. The application is Communication-bound.
|     This might be caused by:
|     - High wait times inside the library - see the MPI Imbalance metric below.
|     - Active communications - see the diagrams 'MPI Time per Rank' (key '-m'
|       or '-m -D' for per MPI-function details) & 'Collective Operations Time
|       per Rank' (key '-t' or '-t -D' for per MPI-function details).
|     - Unoptimized settings of the MPI library. You can tune Intel(R) MPI
|       Library for your application and cluster configuration using the mpitune
|       utility available as part of the library package.
|
          MPI Imbalance:                2.34 sec            37.24%
|         Mean unproductive wait time per-process spent in the MPI library calls
|         when a process is waiting for data. This time is part of the MPI time
|         above. High values are usually bad.
|         This value is HIGH. The application workload is NOT well balanced
|         between MPI ranks.
|         For more details about the MPI communication scheme use Intel(R) Trace
|         Analyzer and Collector available as part of Intel(R) Parallel Studio
|         XE Cluster Edition.
...
```

We can observe that MPI Time and MPI Imbalance Time values are very close, which indicates that the application does little productive work and should be optimized.

The next step is to see the function summary to detect the most time-consuming functions.

Key Terms

MPI Imbalance

2.2.2. Viewing the Function Summary

To see the function summary, process the statistics files with the -f option:

```
$ mps -f ./stats.txt ./app_stat.txt
```

The output may look as follows:

```
| Function summary for all ranks
|--------------------------------------------------------------------------------------------------------
|            Function       Time(sec)          Time(%)       Volume(MB)        Volume(%)            Calls
|--------------------------------------------------------------------------------------------------------
             SendRecv           50.37            59.96            39.06            99.96             9200
            Allreduce           24.20            28.81             0.01             0.03             1600
                 Init            8.83            10.51             0.00             0.00               32
                Bcast            0.55             0.66             0.00             0.00               32
               Gather            0.05             0.06             0.00             0.01               32
```

```
          Finalize                0.00            0.00            0.00            0.00             32
|===================================================================================================
| TOTAL                          84.01          100.00           39.08          100.00          10928
```

We can observe that the `SendRecv` and `Allreduce` functions are potential hotspots and should be optimized.

Key Terms

Hotspot

2.3. Resolving the Issues

2.3.1. Optimizing the SendRecv function

To improve the performance of the `poisson` application we can replace the blocking `SendRecv` function with the non-blocking `Icomm` function.

The source code with the necessary changes is also available at the `poisson` folder. Rename the `pardat.f90_icomm` file into `pardat.f90`

2.3.2. Optimizing the Allreduce function

We can tune the `Allreduce` function with the Intel® MPI Library parameters. To do this set the suitable value for the `I_MPI_ADJUST_ALLREDUCE` environment variable. The `poisson` application shows the best results with the value 2. However, you should check manually which value suits best for your application and your configuration.

2.4. Viewing the Results

To view the results, rebuild your application, run it and process the generated statistics files once again.

As a result of the optimization, we get the following:

- application lifetime reduced from 6.37 seconds to 5.72 seconds (~9%)
- MPI time rate reduced from 37.56% to 30.28% (7.28%)

See the diagrams below.

```
| Summary information
|--------------------------------------------------------------------
  Application    : ./poisson
  Number of ranks: 32
  Used statistics: app_stat.txt, stats.txt
|
  WallClock time :             5.72 sec
| Total application lifetime. The time is elapsed time for the slowest process.
| This metric is the sum of the MPI Time and the Computation time below.
|
      MPI Time:                    1.70 sec            30.28%
|     Time spent inside the MPI library. High values are usually bad.
|     This value is HIGH. The application is Communication-bound.
```

```
|      This might be caused by:
|      - High wait times inside the library - see the MPI Imbalance metric below.
|      - Active communications - see the diagrams 'MPI Time per Rank' (key '-m'
|        or '-m -D' for per MPI-function details) & 'Collective Operations Time
|        per Rank' (key '-t' or '-t -D' for per MPI-function details).
|      - Unoptimized settings of the MPI library. You can tune Intel(R) MPI
|        Library for your application and cluster configuration using the mpitune
|        utility available as part of the library package.
|
          MPI Imbalance:               1.66 sec           29.49%
|         Mean unproductive wait time per-process spent in the MPI library calls
|         when a process is waiting for data. This time is part of the MPI time
|         above. High values are usually bad.
|         This value is HIGH. The application workload is NOT well balanced
|         between MPI ranks.
|         For more details about the MPI communication scheme use Intel(R) Trace
|         Analyzer and Collector available as part of Intel(R) Parallel Studio
|         XE Cluster Edition.
```

```
| Function summary for all ranks
|-----------------------------------------------------------------------------------------------------------
|           Function         Time(sec)          Time(%)        Volume(MB)        Volume(%)             Calls
|-----------------------------------------------------------------------------------------------------------
                Recv             26.94            42.94            138.53            78.00              9200
           Allreduce             25.76            41.06              0.01             0.01              1600
                Init              9.06            14.44              0.00             0.00                32
               Bcast              0.61             0.97              0.00             0.00                32
                Send              0.31             0.49             39.06            21.99              9200
              Gather              0.04             0.07              0.00             0.00                32
| [skipped 2 lines]
|===========================================================================================================
| TOTAL                          62.74           100.00            177.60           100.00             23328
```

3. Key Terms

MPI Imbalance: The unproductive time a process spends in the MPI calls while waiting for data.

Hotspot: A section of code that took a long time to execute. Some hotspots may indicate bottlenecks and can be removed, while other hotspots inevitably take a long time to execute due to their nature.