PTF MPI Parameters Plugin User's Guide

PTF Version: 1.1

MPI Parameters Plugin Version: 1.1

Gertvjola Saveta, Eduardo Cesar, Anna Sikora

13.04.2015

Contents

Chapter 1

Introduction

MPI is the de facto standard for interprocess communications in distributed parallel programs and thus it represents a key factor in the optimization of the MPI-based applications. However, a library setup for a specific system might not perform equally in a different environment (e.g. different architecture or interconnection network), so to increase portability most MPI implementations provide multiple configuration parameters. These parameters are usually set by experienced users who have a deep understanding of a specific MPI application and how it might behave on the target architecture. Changing the MPI parameters allows for improving the application performance.

The plugin for MPI Parameter Tuning aims to automatically optimize the values of a user selected subset of MPI configuration parameters. The integration with PTF provides the plugin with online measurements in the form of high level properties allowing it to make tuning decisions based on the actual performance of the application. The plugin generates the scenarios to represent specific MPI configurations in the form of tuples of parameter-value pairs (i.e. specific combinations of values for the selected subset of MPI parameters). These scenarios are executed as experiments via PTF and evaluated using the resulting properties. Experiments are performed directly on the target application, but we expect to need only a fraction of the program execution (for example a small number of iterations) to be able to analyse its behaviour.

The MPIParameters Plugin gives the opportunity to choose between an exhaustive search space or an evolutionary search to reduce the search space. The exhaustive search searches all the environment variables and list/ranges of values specified by the user. Using evolutionary search the plugin generates the search space that is the crossproduct of all possible combinations of values of the parameters or an initial population of configurations in the

search space, executes them all, or applies a genetic algorithm to define a new population using the configurations with the best results (elitism) and new configurations obtained by crossover and mutations, repeating this process a configurable number of times.

The combination of values leading to the lowest execution time is provided to the user as an advice.

Chapter 2

Quick Start

2.1 Quick installation

The MPIParameters Plugin is being installed along with the Periscope Tuning Framework. Please refer to the *PTF Installation Guide* for a complete description of the installation process.

2.2 Basic configuration

In order to use the MPIParameters plunin, the set of MPI parameters to be tuned must be specified in a configuration file. The name of this file can be indicated using the `PSC_PARAM_SPEC_FILE` environment variable or using the default file `param_spec.conf`. Depending on what flavor of MPI is being used, the user can start from one of the three configuration files provided with the plugin:

```
$Periscope/autotune/mpiparameters/src/MPIparam_ibm.conf
$Periscope/autotune/mpiparameters/src/MPIparam_intel.conf
$Periscope/autotune/mpiparameters/src/MPIparam_openmpi.conf
```

Copy and edit the proper configuration file to reflect the current context of your application. The general syntax for the configuration file is the following:

```
# ********* plugin related settings ***************
# Configuration File Start and Library Implementation
# the desired tuning parameters and search
MPI_PIPO BEGIN [openmpi || ibm || intel]
SEARCH=[exhaustive || individual || gde3];
```

```
<parameter-name>= initial-value:  [[+|*] step:  final-value>;
<parameter-name>=<comma-separated-list-of-values>;
# Configuration File End
MPIPO_END
# ***********************************************
```

2.3 Running MPI Parameters Plugin

MPIParameters runs as a plugin within the Periscope Tuning Framework. It can be started using `psc_frontend` (see also *PTF User's Guide*) by setting the `tune` flag to `mpiparameters`.

```
--tune=mpiparameters
```

For an application, one would call from within the folder:

```
psc_frontend --apprun="./Application" --mpinumprocs=4 --tune=mpiparameters
```

This will start the measurements and the MPIParameters tuning strategy for the application using four processes.

2.4 Execution results

Upon successful completion of the tuning measurements, the MPIParameters plugin displays at the standard output the list of all parameter combinations (scenarios) that were used in the search along with the corresponding execution times (severity), and the flags' values used for each scenario. It also outputs the scenario with the best execution time.

For example, this is the output of the above call to `psc_frontend` for the BT-MZ benchmark for two tuning parameters:

```
 Found best scenario:  8
Parameter combination:
--mca mp_buffer_mem 64M --mca mp_eager_limit 16
All Results:
-----------------------

Scenario | Runtime | Flags

0 | 2.359510 | --mca mp_buffer_mem 16M --mca mp_eager_limit 12
1 | 2.251537 | --mca mp_buffer_mem 16M --mca mp_eager_limit 14
```

```
2 | 2.243806 | --mca mp_buffer_mem 16M --mca mp_eager_limit 16
3 | 2.224361 | --mca mp_buffer_mem 32M --mca mp_eager_limit 12
4 | 2.240855 | --mca mp_buffer_mem 32M --mca mp_eager_limit 14
5 | 2.226031 | --mca mp_buffer_mem 32M --mca mp_eager_limit 16
6 | 2.221029 | --mca mp_buffer_mem 64M --mca mp_eager_limit 12
7 | 2.230055 | --mca mp_buffer_mem 64M --mca mp_eager_limit 14
8 | 2.218171 | --mca mp_buffer_mem 64M --mca mp_eager_limit 16
```

Chapter 3

MPIParameters AutoTuning Approach

MPIParameters follows the general PTF plugin approach (see also *PTF User's Guide*).

3.1 Tuning parameters

Tuning parameters are depended on the MPI Library. Users should create the configuration file specifying the MPI library parameters to be tuned and a range of valid values for each of them. Depending on the parameter, the valid values may vary; some of them require a Boolean value (e.g. `pe_affinity` (yes,no)), while others need a string of characters (e.g. `task_affinity` (core, cpu)), or a range of integers (e.g. `eager_limit` (from a few bytes to 64KB), `buffer_mem` (from 4KB to 2GB), `polling_interval` (from 1 to 2 billion microseconds)). In addition, users can specify the kind of search strategy the plugin should apply, choosing between an exhaustive, individual or genetic strategy `SEARCH=[exhaustive || individual || gde3];`.

Under `$PERISCOPE_ROOT/autotune/mpiparameters/src/` the user can find sample configuration files for the supported libraries.

3.2 Search strategy

In order to find the best tuning of an application, a search through the tuning space has to be performed. For the MPIParameter plugin, the search strategies that can be used are, exhaustive individual and evolutionary.

3.3 Tuning scenario

Tuning scenarios are being generated at run time, and the performance of the application is being evaluated for each of these scenarios. In the MPIParameters plugin, one scenario represents either a tuning parameter or a combination of them.

3.4 Tuning action

Applying a scenario to the application means setting the tuning parameters before running the application. Thus, the tuning action is used to set the tuning parameters. Note that the application is automatically retested for each of the created scenarios.

Chapter 4

Configuration

4.1 Configuration files

All configuration settings for the MPIParameters plugin are read at execution time from the configuration file. The default name of the configuration file is `param_spec.conf`, but a different file can be specified using the `PSC_PARAM_SPEC_FILE` environment variable.

The configuration file is being searched in the folder from which the `psc_frontend` was started. Hence, if the name also includes a relative path to the file, it has to be relative to that folder.

4.2 MPIParameters tuning parameters

The tuning parameters and the search desired for the MPIParameters plugin are defined in the configuration file as follows:

```
# ********* plugin related settings ***************
# Configuration File Start and Library Impementation
# the desired tuning parameters and search
MPI_PIPO BEGIN [openmpi || ibm || intel]
SEARCH=[exhaustive || individual || gde3];
<parameter-name>= initial-value:  [[+|*] step:  final-value>;
<parameter-name>=<comma-separated-list-of-values>;
# Configuration File End
MPIPO_END
# ***********************************************
```

Tuning parameters included in the configuration file are build depending

on the MPI flavor used by the application. There are three flavors directly supported by the plugin: IBM MPI, Intel MPI and OpenMPI. In any of these cases the user specifies the desired command line flags and the values that should be tested. The following templates are provided with the plugin for each of these flavors:

```
MPIPO_BEGIN ibm
eager_limit=4096:2048:65560;
buffer_mem=8388608:2097152:134217728;
use_bulk_xfer=yes,no;
bulk_min_msg_size=4096:4096:1048576;
pe_affinity=yes,no;
cc_scratch_buf=yes,no;
wait_mode=nopoll,poll;
css_interrupt=yes,no;
polling_interval=100000:10000:1000000;
SEARCH=gde3;
MPIPO_END

MPIPO_BEGIN intel
I_MPI_EAGER_THRESHOLD=4096:2048:65560;
I_MPI_INTRANODE_EAGER_THRESHOLD=4096:2048:65560;
I_MPI_SHM_LMT=shm,direct,no;
I_MPI_SPIN_COUNT=1:2:500;
I_MPI_SCALABLE_OPTIMIZATION=yes,no;
I_MPI_WAIT_MODE=yes,no;
I_MPI_USE_DYNAMIC_CONNECTIONS=yes,no;
I_MPI_SHM_FBOX=yes,no;
I_MPI_SHM_FBOX_SIZE=2048:512:65472;
I_MPI_SHM_CELL_NUM=64:4:256;
I_MPI_SHM_CELL_SIZE=2048:1024:65472;
SEARCH=gde3;
MPIPO_END

MPIPO_BEGIN openmpi
mpi_paffinity_alone=0,1;
btl_openib_eager_limit=1024:2048:65560;
btl_openib_free_list_num=2:4:128;
btl_openib_use_eager_rdma=0,1;
btl_openib_eager_rdma_num=1:2:32;
btl_sm_eager_limit=1024:2048:65560;
btl_sm_num_fifos=1:1:10;
btl_sm_fifo_size=2048:512:65472;
btl_sm_free_list_num=2:4:128;
```

`MPIPO_END`

A detailed description of these parameters and the complete list of parameters for each flavor can be found at:

- IBM: `https://www.lrz.de/services/software/paralell/mpi/ibmmpi/`
- Intel: `https://software.intel.com/en-us/mpi-refman-lin-5.0.3-html`
- OpenMPI: `http://www.open-mpi.org/faq/`

For these three cases, the plugin will build the proper command line, for example:

```
mpiexec -n <numprocs> <application> <application-args> <MPI-args>
```

However, the plugin can also support any other MPI flavor, but in this case users must specify the corresponding environment variables and the plugin will build the proper `export` command in order to set their values for each scenario.

4.3 Improved tuning time

To guide the MPIParameters plugin to speedup the tuning process, run an instrumented version of the application.

4.3.1 Instrumented applications

If a region is defined within the application, then the Periscope Tuning Framework will only perform measurements for that region (see also *PTF User's Guide*). This means, for example, that if the application has a main iterative loop, one could define as a region the body of the loop, thus always measuring only one iteration instead of the entire execution. By default, PTF runs on instrumented applications.

Please note that, in the uninstrumented mode, the execution time is measured as the wall clock time of the system command which executes the application. This means that reliable results can be achieved only if the execution time of the application is not too small.

Chapter 5

How To Use the Tuning Advice

The best combination of values is selected based on the value of the `ExecTime` property. These values are returned to the user as the recommended configuration for the selected MPI library parameters. This advice can be applied by assigning the values to the corresponding environment variables, for example `set MP_EAGER_LIMIT = 16384` for IBM MPI, or by passing the value as an option in the `mpirun` command, for example `-eager_limit 16384` for IBM MPI.