PTF User's Guide

PTF Version: 2.0

Michael Gerndt, Anamika Chowdhury

18.03.2016

Contents

Chapter 1

Introduction

Periscope is a scalable automatic performance tuning tool currently under development at Technical University of Munich and is part of the Periscope Tuning Framework (PTF), along with tools like Pathway and tuning plugins.

Periscope provides two main functionalities for Fortran and C/C++ applications: automatic *performance tuning* and *performance analysis*.

Performance tuning is provided through a set of tuning plugins. Periscope tuning plugins support aspect specific tuning. Each plugin uses expert knowledge to, for example, find the best configuration of the tuning parameters of the MPI library. Periscope offers the necessary support for measurements, search logic, and automatic execution of experiments for selected configurations. The best configuration is provided as an advice at the end of the tuning.

Automaric performance analysis is performed at runtime, using an iterative approach. There is a starting set of performance properties, which is then refined based on the measurements and the chosen search strategy. In the end, the appropriate set of performance properties is provided for the application being analyzed. The search threshold, the confidence value, and the severity are defined by means of a formal specification of the properties.

Based on expert knowledge, Periscope uses several strategies to identify possible performance issues. Such strategies provide profile information about program regions and performance properties for MPI and OpenMP.

Periscope consists of four main components: the *frontend*, the *hierarchy of communication and analysis agents*, and the *monitoring library*.

- *The frontend* is responsible for starting both the application to be analyzed, as well as all the internal components of Periscope. All settings regarding the execution of Periscope can be selected by means

of command-line parameters of the frontend process. The frontend the loads the tuning plugins and executes the tuning.

- The *agent hierarchy* is transparent for the common users. At the bottom layer of the hierarchy there are the analysis agents. They control and configure the tuning actions and measurements for each application process. They can start, halt, or resume the application execution, and they also retrieve the performance data. Required tuning actions are communicated by the frontend and at the end of the local search, the tuning objective value is communicated back to the frontend and finally the tuning plugin.
- The *monitoring library* is also transparent to the user and it provides the measurement and communication layer between the application being tested and the performance tool. Periscope relies on the Score-P monitoring system.

Chapter 2

Quick Start

2.1 Installation

2.1.1 Access to pre-installed version

PTF might have been already installed on your system. In order to use it, you have to add to your `.bashrc` file:

```
$ module load periscope
```

and then issue in your home directory:

```
$ source .bashrc
```

Note: Please make sure to add the command for loading the periscope module into your .bashrc. Just issuing the command at the command line is not going to work properly.

If Periscope is not available as a module, it can be installed from the source files, following the common process of configuring and building using Autotools.

Please check the *PTF Installation Manual* for a thorough guide on how to install Periscope on your machine.

2.1.2 .periscope configuration file

Before using Periscope, the `.periscope` setup file has to be created in your home directory. You may create a new one, or copy it from the Periscope installation directory:

```
$ cp $PSC_ROOT/templates/.periscope ~
```

The setup file contains a list of `<option>=<value>` pairs, as follows:

```
MACHINE         = localhost
SITE            = Site
REGSERVICE_HOST = localhost
REGSERVICE_PORT = 50001
REGSERVICE_HOST_INIT = localhost
REGSERVICE_PORT_INIT = 50001
APPL_BASEPORT   = 51000
AGENT_BASEPORT  = 50002
```

It defines the machine where Periscope is running, the system where the registry is started that enables Periscope to connect to the application processes, and the first port used by the application and the analysis agents for communication.

Please refer to the *PTF Periscope Installation Manual* for a detailed description on how to choose the proper option values for your particular system.

2.1.3 SSH access

In order to run Periscope, a private key based `ssh` access has to be provided on the machine running the tool. If not already configured, you can do so in few steps:

1. `$ mkdir ~/.ssh`
2. `$ cd ~/.ssh`
3. `$ ssh-keygen -t rsa -N '' -f id_rsa`
4. `$ cat id_rsa.pub >> authorized_keys`
5. `$ chmod 600 authorized_keys`

The `ssh` access is not required if running on your local machine, i.e. the `MACHINE` option is set to `localhost` in your `.periscope` file.

2.2 First compiler flags tuning

Having Periscope properly installed, there are only few steps required for tuning a test application:

1. specify a *phase region* by instrumenting the source code of the application with the Score-P pragma;
2. modify the `Makefile` to enable instrumentation;

3. build the application;
4. start the tuning;
5. inspect the tuning advice.

For the remainder of this section we consider as the test application the NPB-MZ BT benchmark[1]. A directory with a prepared installation is provided in the source directory of Periscope under `examples/NPB3.2-MZ-MPI`.

2.2.1 Specify the phase region

The PTF uses tuning plugins to determine candidate configurations for the tuning parameters of a plugin-specific tuning aspect. The search space is reduced by expert knowledge coded into the plugins. The configurations are then tested by running an experiment, and the objective value is measured. To not have to restart the application for each experiment, experiments make use of the iterative behavior of the application. Most scientific application do have a progress loop, and in each iteration a different configuration can be tested.

In order to do so, the repetitive region has to be marked in the source code as *phase region*.

For the BT application, the phase region can be defined in file `bt.f` by inserting Score-P pragma as shown in Figure 2.1.

The file is already prepared for you in the test directory.

2.2.2 Modify the makefile

To enable instrumentation with Score-P, one has to substitute the compile/link commands usually defined in the `Makefile` and prepend it with the `scorep` command.

For NPB-MZ BT, one should edit the `config/make.def` file and update the `F77` variable as shown in Figure 2.2. This change was already done in the provided `make.def`. The `scorep` command will instrument the code. The `--user` argument triggers the instrumentation of user regions. Since the phase region is a special user region, the pragmas will be replaced with calls to the monitoring library. The `--online-access` argument will allow online tools like Periscope to connect to the Score-P monitor linked to the application processes, when the phase region is touched the first time.

[1] See `http://www.nas.nasa.gov/publications/npb.html` for download and documentation.

```
      SCOREP_USER_REGION_DEFINE( phase_handle )

c---------------------------------------------------------------------
c    start the benchmark time step loop
c---------------------------------------------------------------------

      do step = 1, niter
c---------- lines omitted here ...

      SCOREP_USER_OA_PHASE_BEGIN( phase_handle, "PhaseRegion",
      SCOREP_USER_REGION_TYPE_COMMON)
       call exch_qbc(u,qbc,nx,nxmax,ny,nz)

      do zone = 1, num_zones
         call adi(rho_i(start1(zone)),us(start1(zone)),
     $           vs(start1(zone)),ws(start1(zone)),
     $           qs(start1(zone)),square(start1(zone)),
     $           rhs(start5(zone)),forcing(start5(zone)),
     $           u(start5(zone)),
     $           nx(zone),nxmax(zone),ny(zone),nz(zone))
      end do
      SCOREP_USER_OA_PHASE_END(phase_handle)
      end do
```

Figure 2.1: The phase region has to be marked with Score-P pragmas.

```
#------------------------------------------------------------
# This is the fortran compiler used for fortran programs
#------------------------------------------------------------
F77=scorep --online-access --user ../bin/bt-mz.$(CLASS).$(NPROCS)
-mpif77
# This links fortran programs; usually the same as $(F77)
FLINK=$(F77)
```

Figure 2.2: Enabling instrumentation of the phase region by Score-P in the makefile.

2.2.3 Build the application

After the phase region was defined and the build command was adjusted, one can continue with the common build process of the test application.

For the NPB-MZ BT example, one should go to the root directory of the NPB-MZ series and issue:

```
$ make clean
$ make bt-mz CLASS=C NPROCS=16
```

2.2.4 Start MPI parameters tuning

Periscope can be started via its frontend `psc_frontend`. Upon calling the executable with proper parameters, both Periscope's internal components as well as the test application are started and the tuning is carried out.

For the NPB-MZ BT example, one should go to the `bin` directory and then call `psc_frontend` as follows:

```
$ psc_frontend --apprun=./bt-mz.C.16 --phase=PhaseRegion
--mpinumprocs=16 --tune=mpiparameters
```

This example demonstrates the tuning of parameters of the MPI library. The tuning plugin is selected via the `--tune` argument. It is setup for Intel MPI. Make sure to use the appropriate modules. The plugin reads a provided configuration file and explores different configurations of MPI parameter settings. A detailed description of the tuning plugin is provided in the *MPI Parameters Plugin User's Guide*.

2.2.5 Explore the results

Upon successful termination, Periscope generates an advice file covering the information about the best configuration and the search path. This is a standard XML file and can be opened using any text editor. Periscope provides the script `psc_result` to pretty print the result.

```
$ module load python
$ psc_result advice_xxxxx.xml
```

2.3 First MPI analysis

After having prepared the application for tuning, it can be used for performance analysis as well. Here, we will perform an automatic MPI analysis.

For the NPB-MZ BT example, one should go to the `bin` directory and then call `psc_frontend` as follows:

```
$ psc_frontend --apprun=./bt-mz.C.16 --phase=PhaseRegion
--mpinumprocs=16 --strategy=MPI
```

Here the `--strategy` option is used instead of `--tune`. The results are provided in form of the found properties in the `properties_*.psc` file. One can use the `psc_result` script again to check the found performance properties. Just pass the properties file as an argument.

Chapter 3

Preparing Applications

3.1 Specification of a phase region

The testing of tuning configurations is carried out within one **experiment**, i.e., a *phase* which is a single execution of the **phase region**. The phase region has to be marked as a special Score-P `user region`. Periscope requires the specification of the phase region.

The best example for a phase region is the body of the main loop of an application. It is common that scientific applications have a main loop iterating through time steps or grid elements. The application is suspended at the beginning of the phase region and new measurements and tuning actions are requested. The application is then released and the analysis is started. When the application encounters the end of the region, it is suspended and the measured values are retrieved.

Figure 3.1 outlines the specification of the phase region via Score-P pragmas. The pragmas are the same for FORTRAN and C/C++.

3.2 Enabling instrumentation

Measuring performance of an application is commonly based on the ability of the performance tool to "communicate" with the application at runtime. This can be achieved through the instrumentation of the application, i.e. inserting tool-specific calls inside the source code to enter the monitoring library, i.e., for Periscope this is Score-P.

In order to enable instrumentation with Score-P, one needs to *prepend the compiling and linking commands* with the call to the `scorep` script. This

```
//include "scorep/SCOREP_User.h"
...
SCOREP_USER_REGION_DEFINE(phase_handle)
//Loop control of the progress loop}
  SCOREP_OA_PHASE_BEGIN(phase_handle, "PhaseRegion",
                        SCOREP_USER_REGION_TYPE_COMMON)
    //Loop body
    ...
  SCOREP_OA_PHASE_END(phase_handle)
//End of the body of the progress loop
...
```

Figure 3.1: Marking the phase region via Score-P pragmas.

can usually be done by editing the `Makefile` of the application.

For example, one should replace

```
mpif90 -c <args>
```

with

```
scorep --online-access --user <scorep_options> mpif90 -c <args>
```

for a Fortran code, and

```
mpicc -c <args>
```

with

```
scorep --online-access --user <scorep_options> mpicc -c <args>
```

for a C/C++ code.

Do not forget to change both the compiling **and** the linking commands.

Please note that the script recognizes the `-c` argument passed to the compiler itself and uses it to decide between the instrumentation and the linking steps. It is thus required that the respective test application is built in two distinct steps: compilation and linking.

The Score-P instrumenter command `scorep` automatically takes care of compilation and linking to produce an instrumented executable, and should be prefixed to compile and link commands. Often this only requires prefixing definitions for CC or MPICC (and equivalents) in Makefiles.

Score-P supports the instrumentation of a wide spectrum of programming models. Please check the Score-P User Guide for details. Table 3.2 summa-

rizes the instrumentation of user-level functions, user regions, MPI functions, and OMP regions.

Region Type	Instrumenter switch (on/off)	Default value
MPI	`--mpp=mpi/` `--mpp=none`	enabled
OpenMP	`--thread=omp/` `--thread=none`	enabled
Routines	`--compiler/` `--nocompiler`	enabled
User Regions	`--user/` `--nouser`	disabled

MPI instrumentation and instrumentation of user regions has to be enabled for PTF. OpenMP and user-level routines can be instrumented. Be aware of the instrumentation overhead. Score-P provides a special filtering mechanism to reduce the instrumentation overhead for fine granular regions. To filter MPI routines, Score-P provides an additional mechanism different from the standard runtime filtering mechanism.

The `--online-access` argument has to be given for the instrumentation to enable Periscope to connect to the application's MPI processes.

Chapter 4

Performance Tuning with Periscope

Performance tuning using Periscope is based on the collaborative work performed by customized tuning plugins on the one side and Periscope as the host application of the plugins on the other side. Users select a tuning plugin by calling `psc_frontend` with the option `--tune=plugin-name`.

For example, the following will run the Compiler Flag Selection (CFS) plugin on the BT application:

```
psc_frontend --apprun="./bt-MZ.W" --mpinumprocs=1
--tune=compilerflags --cfs-config="cfs_config.cfg"
```

Depending on each particular plugin, there might be also other options available for configuration. Please consult the corresponding *User's Guide* for details specific to each of the plugins.

The mandatory parameters which are required for a tuning run are:

Option	Description
--apprun=<command line>	Specify the command line to start the application. It will be passed to the mpirun command. The executable specified in the command line must exist when Periscope is started.

--mpinumprocs=<np>	Number of MPI processes for the application. For serial applications, please set this value to `1`. Periscope treats serial applications as 1-process MPI applications.
--phase=<phase-name>	Name of the phase region provided by the instrumentation.
--tune=<tuning plugin name>	Name of the tuning plugin to run.

Other frequently used options are:

Option	Description
--ompnumthreads=<threads>	Number of OpenMP threads (default: 1).
--info=<level>	Verbosity level (default: 0).
--delay=<number>	Number of phases to be skipped at the beginning.

Please see table 6.2 for a complete list of options accepted by `psc_frontend` or run `psc_frontend --help`.

4.1 Tuning plugins

For the current version, PTF provides the following tuning plugins:

CFS (Name: compilerflags): the *Compiler Flags Selection* plugin finds the combination of compiler flags with which the best execution time is achieved.

DVFS (Name: dvfs): the *Dynamic Voltage and Frequency Scaling* plugin tunes the energy consumption of an application.

MPI Parameters (Name: mpiparameters): automatically optimizes the values of a user selected subset of MPI configuration parameters.

4.2 Tuning advice

As a result of the tuning process, Periscope generates an XML file describing:

- the final *tuning advice* to be applied to the application
- the *tuning scenarios* which were used in searching the best advice
- other information specific to the tuning plugin, like, for example, the tuning parameters, the execution times, or the energy consumption.

4.3 The tuning flow

Being the host of the tuning plugins, Periscope provides several services to build a standard tuning flow.

Data model

The main components of the tuning data model are:

tuning parameters: represent the parameters based on which a tuning of the application can be done. These are plugin dependent and their semantics is strictly defined in each plugin. For example, the CFS plugin uses compiler flags as tuning parameters, while the MPI Parameters plugin uses MPI related switches and parameters.

For most plugins, the tuning parameters are given by user input through a configuration file.

tuning scenario: represents a combination of tuning parameters. The application is analysed by Periscope using one scenario at a time.

Scenarios are computed internally based on a chosen search algorithm. Users can choose between different search algorithms.

tuning space: the set of all valid tuning scenarios.

analysis result: the analysis result associated with one specific tuning scenario. Results are partially displayed in the final tuning advice provided by Periscope.

Operations

On the functional side, the tuning flow is supported by means of two main operations:

search algorithm: the search algorithm generates the tuning space and delivers the next scenario to be evaluated. For most tuning plugins, users can choose the preferred search algorithm.

There are several search algorithms available: exhaustive search, individual search, random search and GDE3 search (one genetic algorithm).

pre-analysis: some plugins require an analysis step before the tuning process can start. The Periscope performance analysis feature is being used in this case.

Required pre-analysis is very much plugin specific. Please consult the given *User's Guide* to see whether user input is possible for each particular case.

4.4 Tuning uninstrumented applications

The CFS plugin also allows tuning of uninstrumented applications, but this is strongly not encouraged. The measurement of the execution time for the objective function includes the overhead of starting the application. For long running applications this is not a problem, but short running once might suffer significant overhead. If one does want to use the uninstrumented version, this can be done by passing the `--uninstrumented` option to the `psc_frontend` process at the command line.

Chapter 5

Performance Analysis with Periscope

Periscope follows an **iterative analysis** approach: it determines performance properties based on measurements, decides on possible new candidate properties, and then it performs again new experiments to measure the data required to check whether the candidate properties hold.

The number of experiments carried out in one run of Periscope depends on the performance issues it might detect. Thus the total execution time of one Periscope analysis will depend on both the execution time of the application itself, as well as the amount and severity of detected performance issues.

5.1 Starting performance analysis

The Periscope performance measurement and analysis process can be started via the `psc_frontend` executable. For example, the following command starts an MPI analysis. The analysis strategy is selected via the `--strategy` option.

```
$ psc_frontend --apprun=./bt-mz_C.16 --mpinumprocs=16
--strategy=MPI --info=1
```

On startup, a hierarchy of analysis and communication agents is first created, then the application to be measured is started and the analysis agents attach to the application nodes. The performance data are gathered by means of the monitoring library and communicated to the low-level agents. There it is analysed using the strategy established at the beginning within the frontend and based on the results, the next step of the iterative analysis is established.

The final results are propagated through the agent hierarchy up to the frontend, which then stores them in the properties file.

Chapter 6

Configuration Options

6.1 Environment Variables

Option	Description
PSC_ROOT	Root directory of the Periscope installation.
PERISCOPE_INFO	0..2 0=quiet 1=startup, found properties in each search 2=candidate properties and found properties in each strategy step

6.2 The frontend - `psc_frontend`

The frontend starts up the application and the agent hierarchy.

Option	Description
--apprun=<appl cmdline>	This is the command line used to start the application. It should be the same as in `mpirun -np <procs> <appl cmdline>`. This value is also used to determine the name of the SIR file, when `--sir` is missing. The executable specified in the command line must exist when Periscope is started. This is true also for the cases where the tuning feature of Periscope is used in combination with plugins which by themselves re-build the application from its source files (e.g. the CFS plugin).
--info=level	Verbosity level. Default: PERISCOPE_INFO or 0
--selective-info=list	Individual debugging level names separated by comma
--delay=<n>	Number of phase executions that are skipped before the search is started. This is useful for applications that have a different behaviour at the beginning.
--duration=<n>	Search delay in seconds of phase
--dontcluster	Do not use online clustering for the detected bottlenecks.
--help	Help information
--maxcluster=<n>	Maximum number of MPI processes analyzed by a single analysis agent. Default: 64
--maxfan=<n>	Determines the fan-out of the tree of high-level agents in interactive mode. Default: 4
--mpinumprocs=<n>	Number of MPI processes to be started.

--nprops=<n>	Specifies the number of properties the frontend prints to standard output. Regardless of this value, all properties are output to the properties file. Default: 50.
--ompnumthreads=<n>	Number of OMP threads to be started per MPI process. Default: 1.
--pedantic	Shows all detected properties.
--phase=<name>	Specifies the phase region via the region name given in the Score_P pragma.
--propfile=<filename>	Specify the file to use when exporting the properties. Default: properties.psc
--cfs-config=filename	Relative path to the CFS plugin configuration file
--strategy=<strategyname>	Strategy used by analysisagent. Currently one of MPI - MPI Communication analysis OMP - OpenMP analysis
--timeout=<secs>	Timeout for startup of the agent hierarchy. Default: varying depending on the number of processes
--uninstrumented	**Autotuning only**: instructs Periscope to tune an uninstrumented application. Use with caution. See also Section 4.2.
--name	defines the name of the application
--version	Displays the version of Periscope.
--starter=name	Specifies resource manager name. E.g. Fast Interactive, Interactive, SLURM
--statemachine-trace	Collects and prints state-machine transitions
--registry=<host:port>	This option defines the address of the registry service
--port=<n>	Local port number
--maxthreads=<n>	Maximum number of threads assigned to a node agent

--masterhost=<hostname>	Name of the host where the frontend starts
--hlagenthosts=<list>	List with host names of HL agents, e.g. –hlagenthost=h1,h2,...
--nodeagenthosts=<list>	List with host names of analysis agents, e.g. –nodeagenthosts=h1,h2,...
--agenthostfile=<name>	File containing host configuration
--manual	Run Periscope in manual mode
--property=<name>	Name of a property to export

Chapter 7

Advanced user information - technical details

The application and the agent network are started through the `psc_frontend` process. First the set of available processors is analysed and based on this the mapping of application and analysis agent processes are determined. Both the application and the agent hierarchy are then started and a command is propagated from the frontend down to the analysis agents to start the search. The search is performed according to a search strategy selected when the frontend is started.

Each of the analysis agents, i.e. the nodes of the agent hierarchy, searches autonomously for inefficiencies in a subset of the application processes.

The application processes are linked with a monitoring system that provides the *Monitoring Request Interface* (MRI). The agents attach to the monitor via sockets. The MRI allows the agent to configure the measurements, to start, to halt, to resume the execution, and to retrieve the performance data. The monitor currently only supports summary information.

At the end of the local search, the detected performance properties are reported back via the agent hierarchy to the frontend.

7.1 Agent hierarchy

The layout of the agent hierarchy can be controlled by the user by means of the specific parameters of the `psc_frontend` executable:

maxfan: determines the fan-out of the tree of high-level agents. By default this is set to 4.

maxcluster: gives the maximum number of MPI processes analysed by a single analysisagent. The default number is 64.

Further information on how the agents work within a specific run of PTF can be gathered by using the `--selective-debug` parameter of the same `psc_frontend` executable:

```
--selective-debug= <level1>,<level2>...
```

with the following *levels* being relevant for the agent hierarchy:

AgentApplComm: displays information regarding the communication between the agents and the application nodes.

AutotuneAgentStrategy: displays information regarding the analysis strategy used in the analysis agent for tuning. To be used only when the tuning feature of PTF is being used.

Using a proper layout of the agent hierarchy is very important especially when performing analysis and tuning of applications on large systems.

Examples

You can find one example with the adapted makefile in ~/Periscope/testcases/add.

Example on SuperMUC

Periscope can be used in batch jobs.

Example batch script:

```
#@ wall_clock_limit = 00:30:00
#@ job_name = mytest
#@ job_type = MPICH
#@ class = test
#@ island_count = 1
#@ node = 1
#@ total_tasks = 1
#@ node_usage = not_shared
#@ initialdir = .
#@ output = out.txt
#@ error = out.txt
#@ notification = never
#@ queue
. /etc/profile
. /etc/profile.d/modules.sh

export OMP_NUM_THREADS=1
psc_frontend –apprun=../add.exe –mpinumprocs=1 –tune=compilerflags –
phase="mainRegion"
```