Periscope Tuning Framework

Robert Mijaković<robert.mijakovic@in.tum.de>

Technische Universität München

Outline

- The Periscope Tuning Framework
- Performance Analysis with PTF
  - Concepts
  - OpenMP and MPI analyses
  - Hands-on exercise
- Performance Tuning with PTF
  - Tuning Plugins
  - Compiler Flag Selection Plugin (CFS)
  - MPI Parameters Plugin (MPIParams)
  - Parallelism Capping Plugin (PCAP)
  - Hands-on exercise
- Questions / Discussion / Coffee break

The Periscope Tuning Framework (PTF)

- PTF provides automatic performance analysis and tuning
  - It is a distributed online tool
  - Based on expert knowledge
- It consists of
  - Periscope: Analysis and tuning system
  - Periscope Tuning Plugins: Aspect-specific tuning
  - Pathway: Automatic Performance Engineering Workflow system
- PTF
  - was implemented in Autotune (EU-FP7)
  - is being continued in READEX (EU-FP7)
  - is open source
  - is downloadable at periscope.in.tum.de

PTF Home Page: periscope.in.tum.de

NAVIGATION

- Home
- Pathway
- Periscope
- HPC Tuning Plugins
- Documents
- Tutorials
- Downloads
- Projects
- Impressum

Search...

PERISCOPE TUNING FRAMEWORK

PERISCOPE TUNING FRAMEWORK

About

This is the official web presence of the Periscope Tuning Framework. The PTF is a suite of tools designed to assist the HPC application developer in the optimization of their application. Below is a quick overview of our main areas of contribution.

The Periscope Tuning Framework received significant funding from the European Commission in the FP7 project AutoTune.

All tools are available free of charge. The best way to get in touch with our tools is to visit us during one of our workshops or contact us directly.

We also maintain a YouTube channel for introduction and video tutorials.

Areas of contribution

Performance analysis

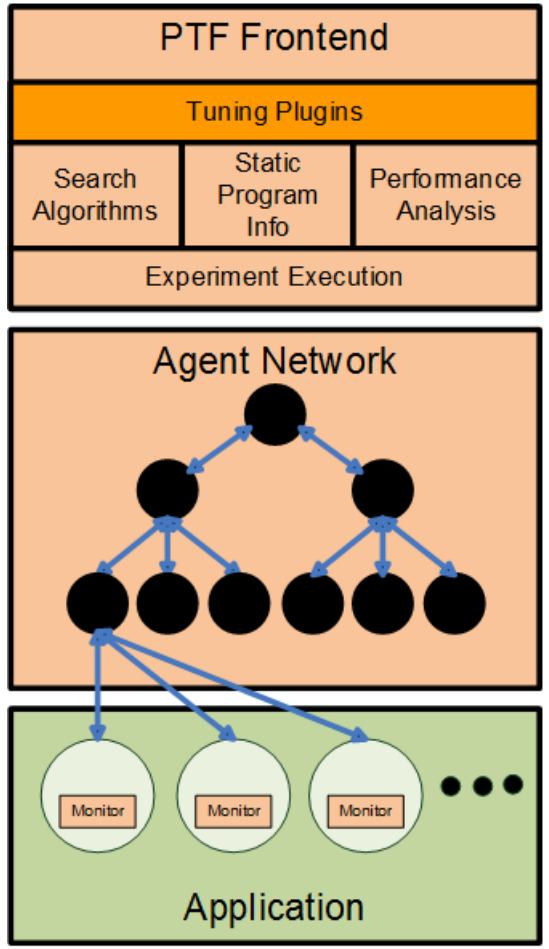
Using Periscope, an online instrumentation and measurement tool, the developers can more easily identify the bottlenecks of their application. Periscopes develops hypotheses about the bottlenecks of an application automatically and verifies them during run-time.

Performance analysis with automatically verified hypothesis can give the developer a good starting point and is especially advisable, when the exact bottlenecks are not yet known.

Auto-tuning

Periscope Architecture

- Periscope Frontend
  - Controls the analysis and tuning process
  - Performs a sequence of experiments
    - While the application is executing
    - Based on application phases
    - Automatically starting/restarting the application if required
- Agent Network for scalability
  - Leave agents responsible for a subset of the MPI processes
  - Intermediate agents aggregate performance properties
- Online Access to the monitoring system
  - Configuration of measurements and tuning actions
  - Retrieval of performance data



Version

- PTF 2.0
  - Based on Score-P
  - Analysis strategies work with the standard Score-P
  - Tuning plugins require an extended Score-P. This is used in this course.

Performance Analysis with Periscope

- Performance Properties
  - Formalize expert knowledge about typical performance problems
  - What to measure?
  - How to verify that the problem exists?
  - How severe is the problem?
- Analysis Strategies
  - Searches performance properties for a programming model
  - PTF 2.0 supports
    - OpenMP
    - MPI
- Significance
  - Overall threshold of 5% of execution time
  - Pedantic switch

Hands-on: Performance Analysis

Finding Performance Bottlenecks for OpenMP and MPI

Hands-on Performance Analysis: MPI

- Copy the test folder:

```
% cd; cp -r /home/hpc/a2c06/lu23bik/PTF-Demo .
```

- Load the PTF environment from the demo folder:

```
% cd PTF-Demo; source load-ptf
```

- Change the directory

```
% cd bin.scorep
```

- Submit job file and check job.out.mpi

```
% sbatch psc_analysis_mpi.slurm
```

Hands-on Performance Analysis: OMP

- (Skip if already done) Copy the test folder:

```
% cd; cp -r /home/hpc/a2c06/lu23bik/PTF-Demo .
```

- (Skip if already done) Load the PTF environment from the demo folder:

```
% cd PTF-Demo; source load-ptf
```

- (Skip if already done) Change the directory

```
% cd bin.scorep
```

- Submit job file and check job.out.omp

```
% sbatch psc_analysis_omp.slurm
```

Hands-on Performance Analysis: OMP

- Periscope is started via its frontend. It automatically starts the application and the agent hierarchy.
- Run `psc_frontend --help` for brief usage information

```
% psc_frontend --help
Usage: psc_frontend <options>
  [--help]                (displays this help message)
  [--quiet]               (do not display debug messages)
  [--appname=name]
  [--apprun=commandline]
  [--mpinumprocs=number of MPI processes]
  [--ompnumthreads=number of OpenMP threads]
  [--strategy=name]
  [--tune=name]
  [--phase=name]
  [--info=level]
```

Hands-on Performance Analysis: OMP

```
ALL FOUND PROPERTIES
-------------------------------------------------------------------------------
 Procs        Region       Location              Severity        Description
-------------------------------------------------------------------------------
P  24;  !$omp parallel  10;    z_solve.f:43;     8.317;       Load Imbalance in parallel region
P   9;  !$omp parallel  10;    y_solve.f:43;     8.029;       Load Imbalance in parallel region
P  15;  !$omp parallel  10;    z_solve.f:43;     7.917;       Load Imbalance in parallel region
P  23;  !$omp parallel  10;    y_solve.f:43;     7.336;       Load Imbalance in parallel region
P  28;  !$omp parallel  10;    z_solve.f:43;     7.197;       Load Imbalance in parallel region
P   1;  !$omp parallel  10;    y_solve.f:43;     7.020;       Load Imbalance in parallel region
P   2;  !$omp parallel  10;    y_solve.f:43;     6.935;       Load Imbalance in parallel region
P  13;  !$omp parallel  10;    z_solve.f:43;     6.915;       Load Imbalance in parallel region
P   5;  !$omp parallel  10;    y_solve.f:43;     6.781;       Load Imbalance in parallel region
P   3;  !$omp parallel  10;    y_solve.f:43;     6.669;       Load Imbalance in parallel region
…
```

Hands-on Performance Analysis: OMP

- Other important command-line arguments:
  - **`--delay=n`** skip n phase, e.g., application warm-up
  - **`--maxcluster=n`** assign at most n processes to an agent
  - **`--iterations=n`** combine n iterations of progress loop into a phase

- Run with multiple agents

```
psc_frontend --apprun=$EXE
             --mpinumprocs=32
             --ompnumthreads=8
             --strategy=OMP
             --phase="ITERATION"
             --info=1
             --pedantic
             --maxcluster=16
```

Hands-on Performance Analysis: OMP

- Results:

```
ALL FOUND PROPERTIES
-------------------------------------------------------------------------------
Procs              Region       Location       Severity       Description
-------------------------------------------------------------------------------
P(24,28,15);   !$omp parallel @z_solve.f:43;   7.846; Load Imbalance in parallel region
P(...);        !$omp parallel @y_solve.f:43    7.145; Load Imbalance in parallel region
P(...);        !$omp parallel @x_solve.f:46    6.293; Load Imbalance in parallel region
P(11,18,20,14,22,23,26,9,29,17);
               !$omp parallel @z_solve.f:43;   5.154; Load Imbalance in parallel region
```

Performance Tuning

- Tuning aspects
  - Compiler flags, MPI library parameters, MPI IO tuning hints, Energy efficiency through DVFS, Degree of parallelism
  - Have unique parameters with non-intuitive best settings
- Tuning objectives
  - Execution time or performance
  - Energy consumption
  - Energy delay product
  - Memory consumption
  - Costs
- Tuning plugins
  - Expert knowledge how to shrink the typically large search space
  - Deploy aspect-specific tuning strategy
  - Make use of predefined search strategies: exhaustive, individual, random, machine learning enhanced random, …
  - Come with configuration file and user‘s guide

Status

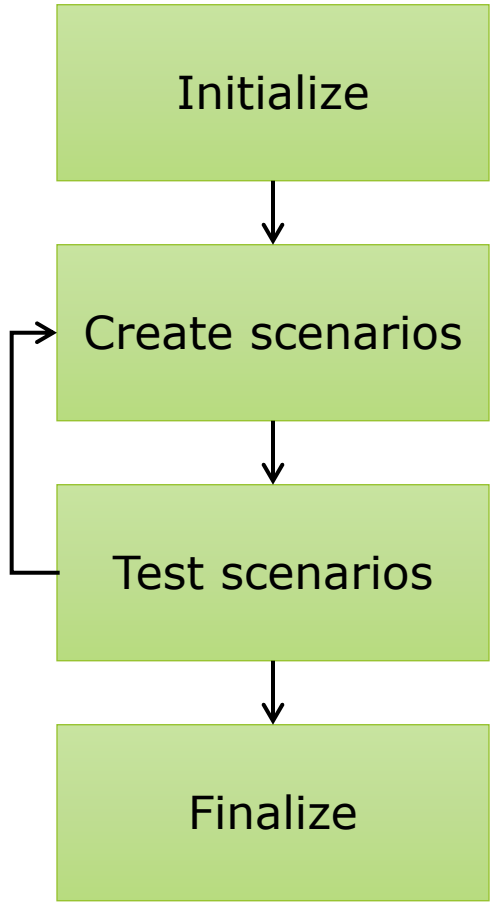
- PTF 2.0
  - Compiler Flag Selection plugin
  - MPI Parameters plugin
  - Energy tuning with DVFS plugin (SuperMUC only)
  - Parallelism Capping plugin

The Plugin System

The frontend loads a *Tuning Plugin* that will tune certain parameters of the application (compiler settings, runtime settings, system/hardware settings, …)

- All tuning plugins have to follow a specific lifecycle
- Plugins can request performance properties
- Plugins can request tuning actions, such as to change runtime values or re-compile or re-run the application

Please note: This is a very simplified picture!



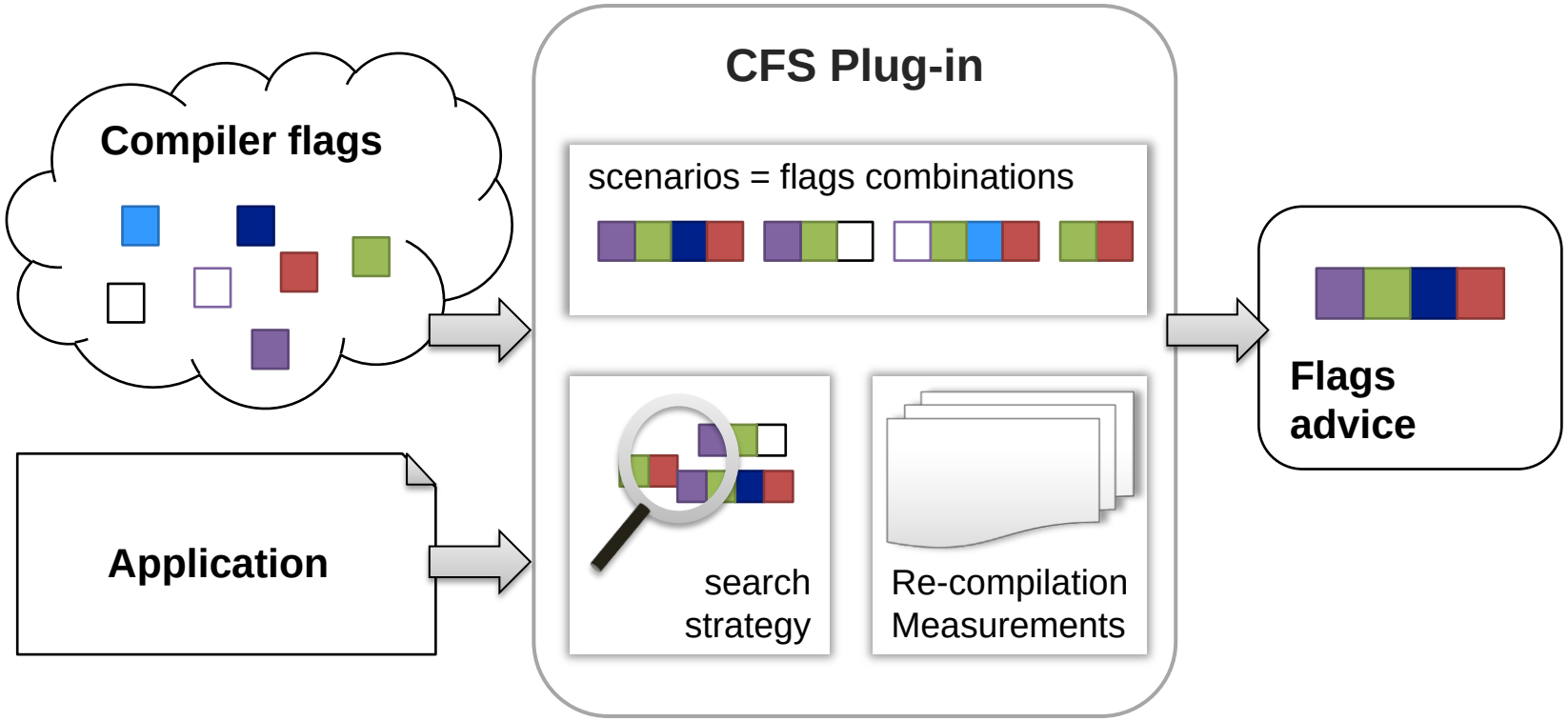
The CFS plugin

Keys:

- We are interested in compiler flags that control *code generation*
- All the possible combinations of flags form a *search space*
- For every trial, the application is re-built and re-run

Applicable to:

- Compute-bound applications
- Single-core optimization



The MPI Parameters plugin

Keys:

- We are interested in run-time tuning of MPI parameters
- All the possible combinations of flags form a *search space*
- Evaluate various protocol, e.g., eager or rendezvous
- For every trial, the application is re-run

Applicable to:

- Communication-bound MPI applications
- Hybrid applications

Hands-on: The CFS Plugin

Finding the best combination of compiler flags

First steps with the CFS Plugin

1. Command line for starting Periscope with pcap-speedup plugin:

```
psc_frontend --apprun=$EXE --mpinumprocs=$PROCS --ompnumthreads=$OMP_NUM_THREADS
--tune=compilerflags --phase="ITERATION" --cfs-config="cfs_config.cfg" --info=1
```

2. Uncomment the appropriate line in the job script

```
vim psc_tuning_cfs.slurm
```

3. Submit the job

```
sbatch psc_tuning_cfs.slurm
```

4. Check the output

```
vim job.out.cfs
```

CFS Plugin Configuration File

```
makefile_path="~/PTF-Demo";
makefile_flags_var="COMPFLAGS";
makefile_args=" bt-mz CLASS=B NPROCS=32";
application_src_path="..";
make_selective="false";
remote_make="false";
//identity_path="~/.ssh/id_rsa";
remote_make_machine_name="mpp3-login8 . /etc/profile";

search_algorithm="exhaustive";

tp "OPT" = "-" ["O1", "O2", "O3"];
tp "AVX" = " " ["-xMIC-AVX512", " "];
```

Hands-on: The MPI Parameters Plugin

Finding the best combination of MPI parameters

First steps with the MPI Parameters Plugin

1. Command line for starting Periscope with pcap-speedup plugin:

   ```
   psc_frontend --apprun=$EXE --mpinumprocs=$PROCS --ompnumthreads=$OMP_NUM_THREADS
   --tune=mpiparameters --phase="ITERATION" --info=1
   ```

2. Uncomment the appropriate line in the job script

   ```
   vim psc_tuning_mpiparams.slurm
   ```

3. Submit the job

   ```
   sbatch psc_tuning_mpiparams.slurm
   ```

4. Check the output

   ```
   vim job.out.mpiparams
   ```

MPI Parameters Plugin Configuration File

```
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
```

Thank You