Introduction to the Periscope Tuning Framework

Michael Firbach[1] <firbach@in.tum.de>
Yury Oleynik[1] <oleynik@in.tum.de>

[1] Technische Universität München

Outline

In this presentation:

- Presentation PTF (20 min)
  - What is the Periscope Tuning Framework?
  - The plug-in system
  - Recent developments
- The CFS plug-in (40 min)
  - Explanation
  - Hands-on exercise
  - Additional features
- Questions / Discussion / Coffee break

What is the Periscope Tuning Framework?

The PTF is a tool that runs, analyses and optimizes your application (auto-tuning).

- Developed as part of the Autotune project (EU- FP7)
- Now being continued within the Score-E project (BMBF).

What is the Periscope Tuning Framework?

- Automatic application tuning
  - Tune performance and energy
  - Create a scalable and distributed framework
  - Evaluate the alternatives online
  - Tune for Multicore and GPU accelerated systems
- Support variety of parallel paradigms
  - MPI, OpenMP, OpenCL, Parallel pattern, HMPP

Technische Universität München, Germany

Universität Wien, Austria

CAPS Entreprise, France

Universitat Autònoma de Barcelona, Spain

Leibniz Computing Centre, Germany

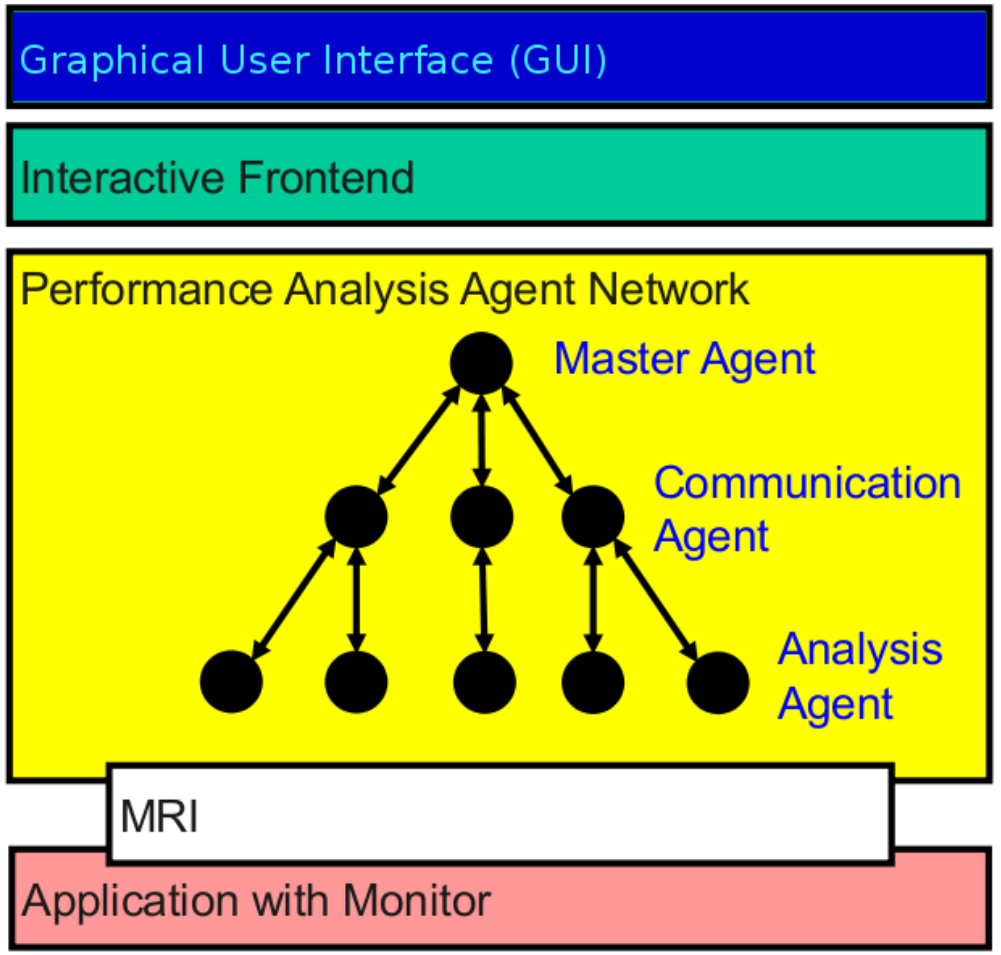
Irish Centre for High-End Computing, Ireland

What is the Periscope Tuning Framework?

The PTF is a runtime system that runs, analyses and optimizes your application (auto-tuning).

It consists of a frontend and a hierarchical network of agents.

- Measurement results are propagated up to the frontend
- Tuning decisions are passed down to the analysis agents

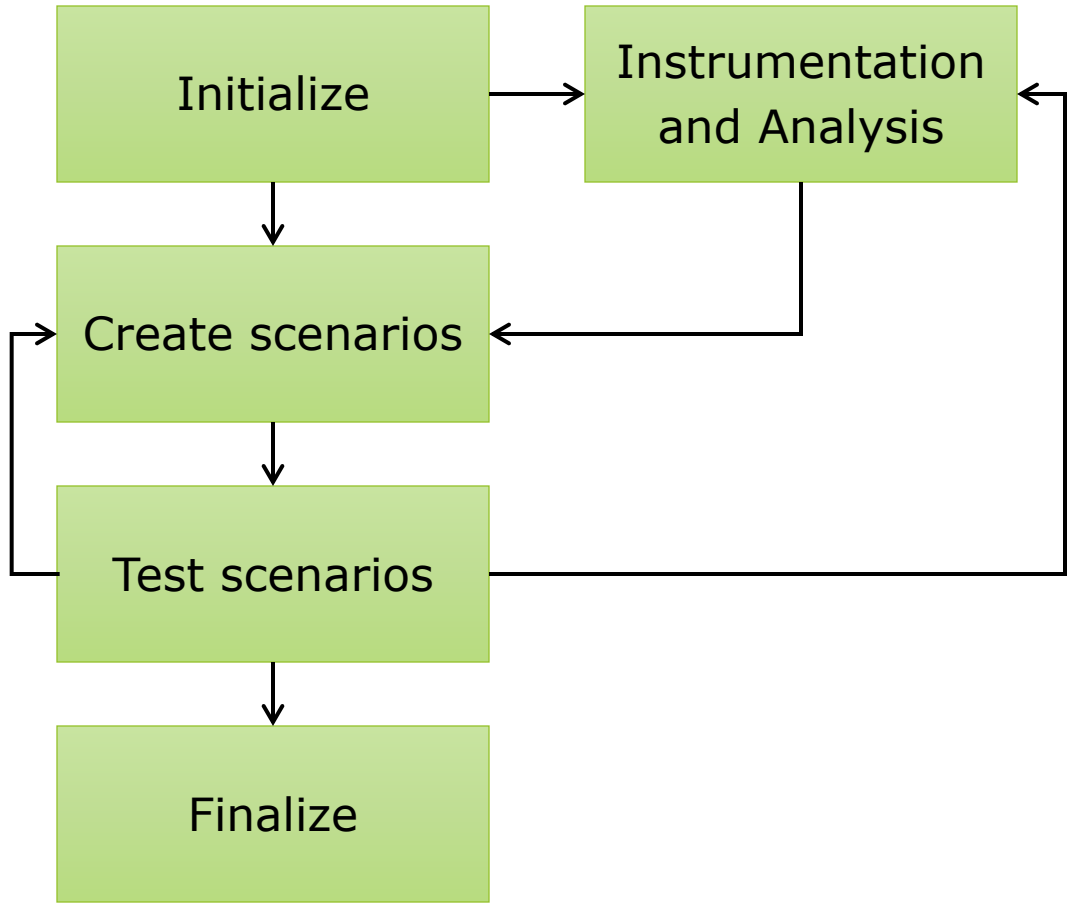


The plug-in system

The front-end loads a *Tuning Plug-in* that will tune certain parameters of the application (compiler settings, runtime settings, system/hardware settings, …)

- All tuning plug-ins have to follow a specific lifecycle
- Plug-ins can request measurements and instrumentation from the runtime system
- Plug-ins can make decisions, such as to change runtime values or re-compile or re-run the application

Please note: This is a very simplified picture!

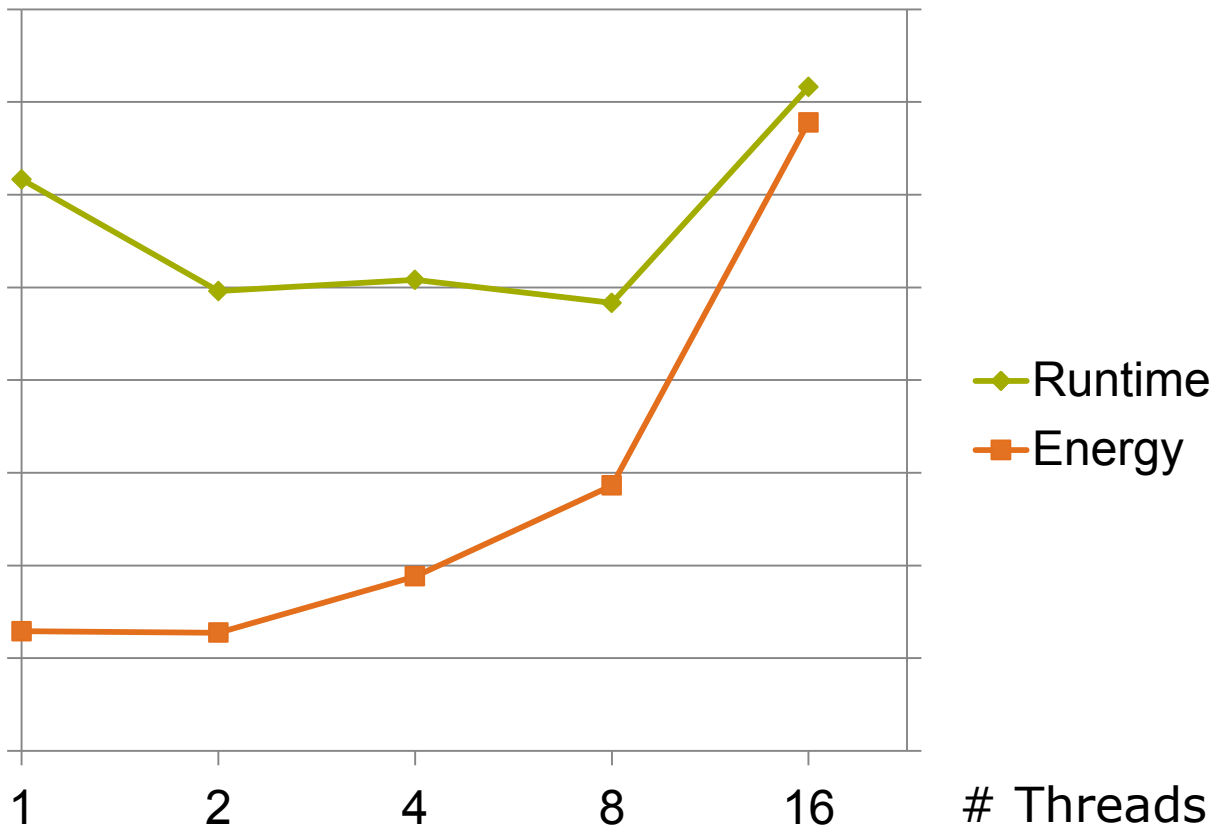


Recent developments

Being part of Score-E, the current focus lies on energy efficiency.

Tuning-Plugins to optimize energy consumption:

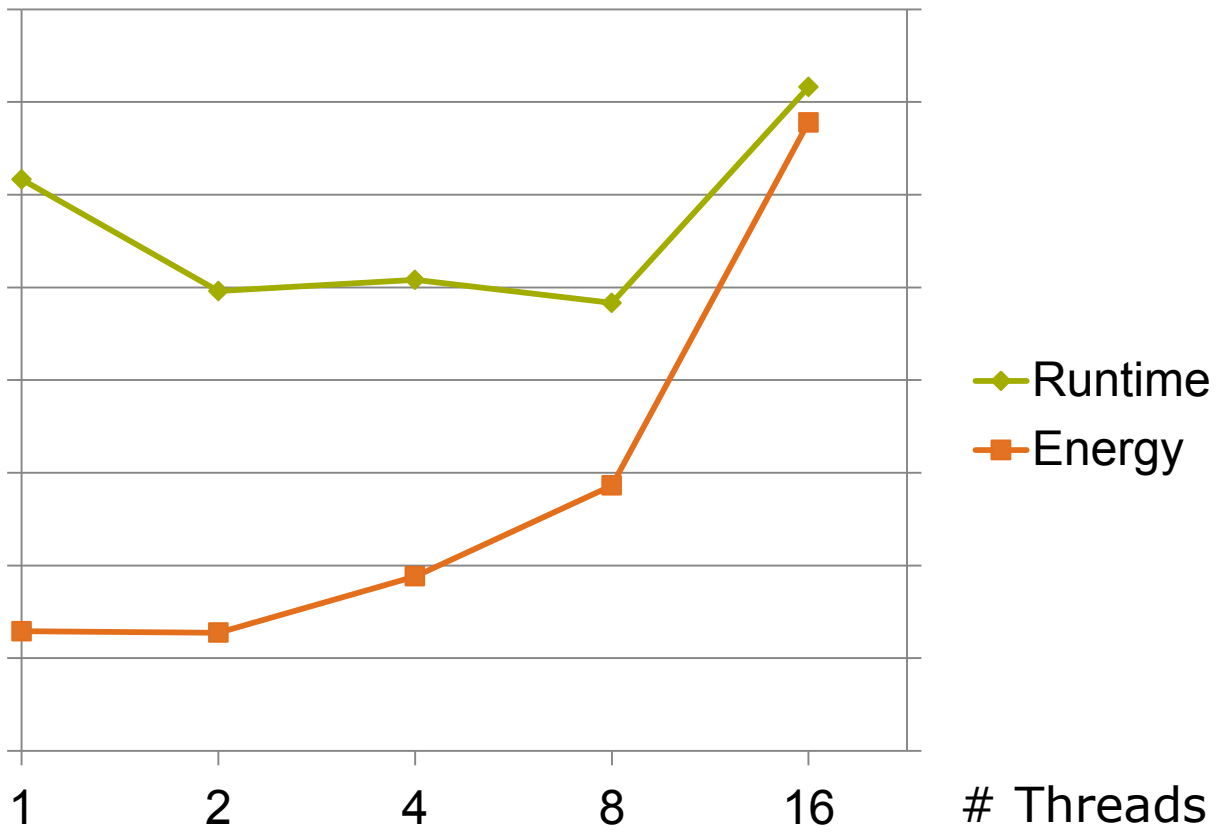
- **PCAP:** Determines the optimal number of threads in each OpenMP parallel region
- **MPI-Procs:** Determines the optimal number of processes to use for an MPI application
- **DVFS:** Voltage and frequency scaling



Recent developments

Alas, energy measurements currently only possible on SuperMUC using proprietary library.

- Ongoing work to use Score-P as underlying measurement infrastructure.
  - Will enable energy measurements everywhere
- Not doing any energy-tuning in today's exercises



Plugin overview

Name	Target	Objective
Compiler Flag Selection (CFS)	Compiler flag combinations	Optimize performance
PCAP	OMP parallel sections	Optimize energy consumption
MPI-Procs	# of MPI processes	Optimize energy consumption
Dynamic Voltage and Frequency Scaling (DVFS)	CPU Frequency and power states	Optimize energy consumption
High Level Pipeline Patterns	Pipeline stages	Optimize throughput
MPI Runtime	MPI parameters	Optimize SPMD MPI application performance
Master – Worker MPI	Workload imbalance in MPI applications	Balance application load by optimizing communication

Exercise: The CFS Plug-in

Finding the best combination of compiler flags

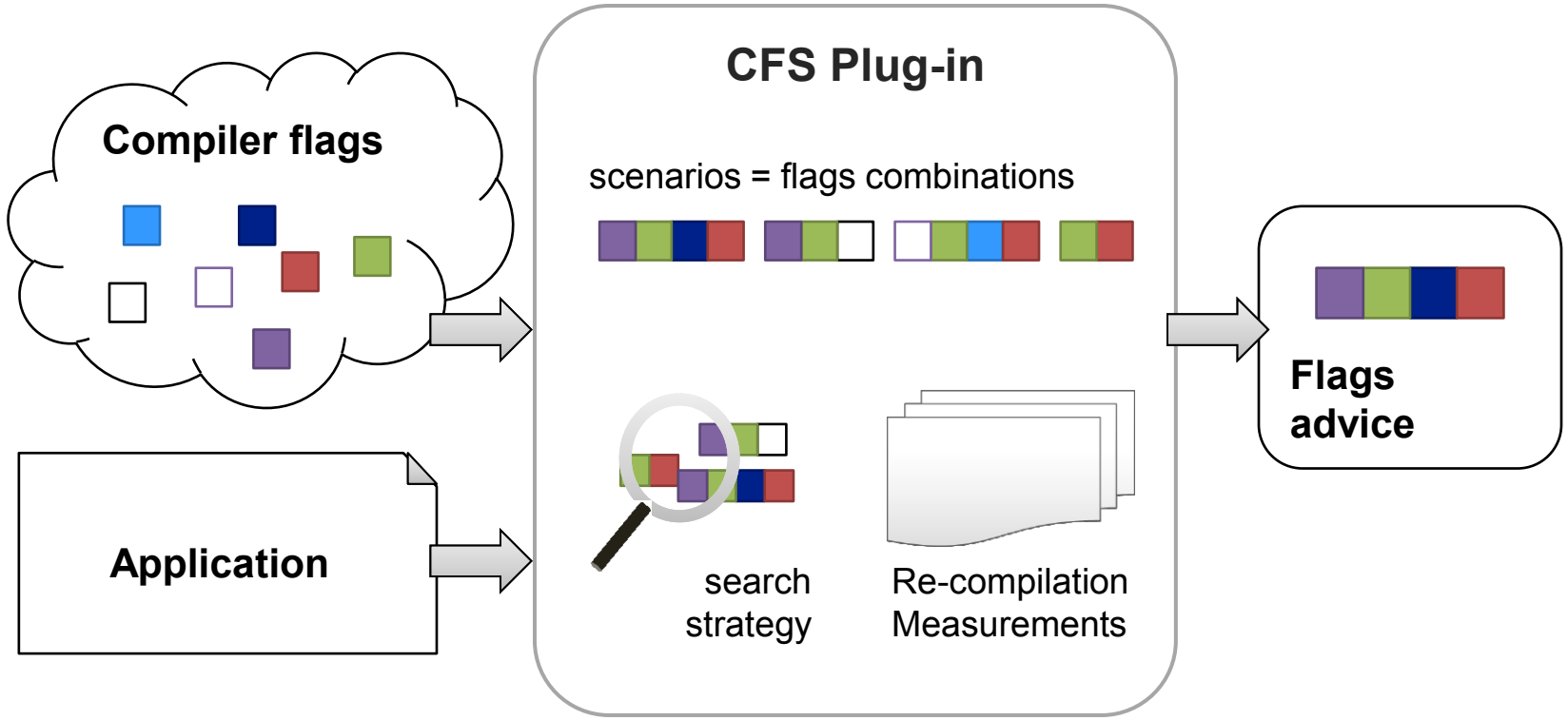
The CFS Plug-in

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 CFS Plug-in

1. Please use the GNU programming environment in this exercise!

```
module unload PrgEnv-cray
module load PrgEnv-gnu
module load chem/boost
```

2. Add Periscope to your PATH:

```
PATH=$PATH:/zhome/academic/HLRS/xhp/xhpmf/periscope/build/install/bin
```

You can use another environment for your own application in the afternoon.

First steps with the CFS Plug-in

1. Copy the test application and the documentation over to your home:
   ```
   cp -r /zhome/academic/HLRS/xhp/xhpmf/copy-me ~
   ```

2. Copy the Periscope config file to your home:
   ```
   cp ~/copy-me/config/.periscope ~
   ```

3. Open the CFS config file:
   ```
   cd ~/copy-me/NPB3.3-MZ-MPI/bin
   vim cfs_config.cfg
   ```

First steps with the CFS Plug-in

Contents of the cfs_config.cfg:

```
makefile_path="../";
makefile_flags_var="FFLAGS";
makefile_args="BT-MZ CLASS=B NPROCS=4";
application_src_path="../BT-MZ";
make_selective="false";

search_algorithm="exhaustive";

tp "OPT" = "-" ["O2", "O3", "O4"];
```

Build instructions

Search strategy

Flags

Try to understand the contents of this file: It contains build instructions and the set of compiler options (along with possible values) for examination.

First steps with the CFS Plug-in

1. Command line for starting Periscope with CFS plug-in would be:
   ```
   psc_frontend --apprun=./bt-mz_B.4 --uninstrumented --
   mpinumprocs=4 --tune=compilerflags
   ```

2. You can use the prepared job script – submit from bin directory:
   ```
   qsub ../../jobscripts/job_cfs.pbs
   ```

3. Check the output:
   ```
   tail –F my_output_file
   ```

4. Which scenario is the fastest?

First steps with the CFS Plug-in

```
Periscope Performance Analysis Tool (ver. 1.1.0)
[psc_frontend][INFO:fe] Preparing to start the performance analysis...

Loaded Autotune components:

Plugin:           Compiler Flag Selection Plugin
Version:          1.5
Description:      The Compiler Flags Selection Plugin (CFS) determines the best selection of compiler switches.

Search Algorithm: Exhaustive Search
Version:          1.0
Description:      Explores the full space spanned by all tuning parameters.

Significant region in process:

touch add.f

...

dir: cannot access *.f90: No such file or directory
dir: cannot access *.c: No such file or directory

...

make: Entering directory `/zhome/academic/HLRS/xhp/xhpyo/copy-me/NPB3.3-MZ-MPI'
   ========================================
   =      NAS PARALLEL BENCHMARKS 3.3       =
   =      MPI+OpenMP Multi-Zone Versions    =
   =      F77                               =
   ========================================

cd BT-MZ; make CLASS=B NPROCS=4 VERSION=
make[1]: Entering directory `/zhome/academic/HLRS/xhp/xhpyo/copy-me/NPB3.3-MZ-MPI/BT-MZ'
make[2]: Entering directory `/zhome/academic/HLRS/xhp/xhpyo/copy-me/NPB3.3-MZ-MPI/sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory `/zhome/academic/HLRS/xhp/xhpyo/copy-me/NPB3.3-MZ-MPI/sys'
../sys/setparams bt-mz 4 B
make[2]: Entering directory `/zhome/academic/HLRS/xhp/xhpyo/copy-me/NPB3.3-MZ-MPI/BT-MZ'
ftn  -c  -O2  bt_scorep_user.F
```

PTF Intro

Re-compilation

```
...

[psc_frontend][INFO:fe] Starting non-instrumented application ./bt-mz_B.4 using 4 MPI procs and 1 OpenMP threads...


 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

 Number of zones:   8 x   8
 Iterations: 200    dt:   0.000300
 Number of active processes:     4

 Use the default load factors with threads

 Calculated speedup =      3.99

 Time step    1

 ...


 BT-MZ Benchmark Completed.
 Class           =                        B
 Size            =            304x  208x 17
 Iterations      =                      200
 Time in seconds =                    32.83
 Total processes =                        4
 Total threads   =                        4
 Mop/s total     =                 18314.20
 Mop/s/thread    =                  4578.55
 Operation type  =            floating point
 Verification    =                SUCCESSFUL
 Version         =                     3.3.1
 Compile date    =               26 Feb 2015
Application 694057 resources: utime ~132s, stime ~1s, Rss ~50536, inblocks ~318, outblocks ~205
[psc_frontend][INFO:fe] Scenario pool not empty, still searching...
```

Execution of scenario

First steps with the CFS Plug-in

```
Optimum Scenario: 0
Compiler Flags tested:
Scenario 0 flags: "-O2 "
Scenario 1 flags: "-O3 "
Scenario 2 flags: "-O4 "


All Results:
Scenario          |  Severity
0                 |  35.2373
1                 |  40.2997
2                 |  40.4733
```

The CFS Plug-in

More advanced features (see User's Guide):

- Other search strategies, like individual search:
  - Creates scenarios with only one flag altered at a time
  - Might miss the optimal combination
  - Much faster (linear complexity)
- Selective make:
  - Periscope can determine relevant source files and re-build only those
  - Or, user provides list of files
  - Selected files are touched, then the application is re-built
- Periscope can suggest flags to test for a specific compiler

The CFS Plug-in

What you can expect:

- Performance increase will be moderate (maybe 5% to 10%)
- However, you don’t invest a lot of time
  - Only configure it once
  - Plug-in runs without user interaction
- Probably a good ratio of time spent and improvement
- Other plug-ins will be available in future workshops
  - MPI-tuning
  - Energy-tuning
  - …

Thank You

In the afternoon, you can tune your own application.
We will be happy to assist you.