Automatic Tuning of HPC Applications with Periscope

Michael Gerndt, Michael Firbach, Isaias Compres
Technische Universität München

Agenda

15:00 – 15:30 Introduction to the Periscope Tuning Framework (PTF)

15:30 – 16:00 Tuning plugins

- Compiler Flag tuning
- MPI parameter tuning
- Energy efficiency tuning

16:00 – 17:00 Hands-on

- Application of PTF with the tuning plugins to a prepared benchmark.

Background

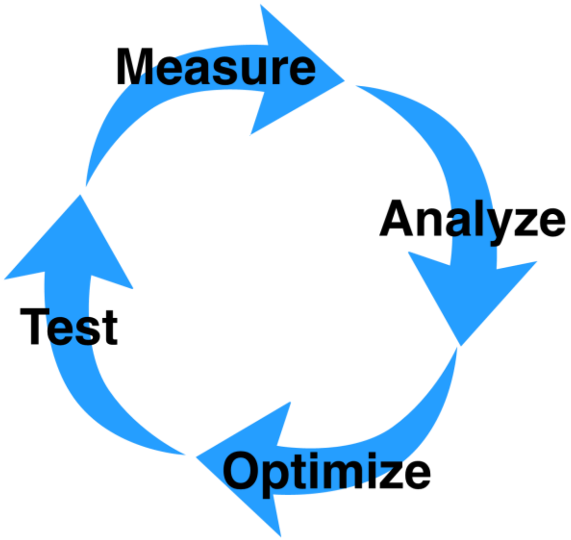
- Given
  - Multicore, Accelerators, HPC Cluster
  - Many programming models and applications
- Problem
  - How to tune application for a given architecture?
- Targeted Users
  - Improve performance : HPC Application Developers
  - Faster tuning : HPC Application Users
  - Reduce energy cost : Supercomputing Centers

Motivation

- Why tune applications?
  - Increasing complexity of HPC architectures
  - Frequently changing HPC hardware
  - Compute time is a valuable resource
- Manual tuning
  - Large set of tuning parameters with intricate dependencies
  - Diverse set of performance analysis tools
  - Several iterations between analysis and improvements

Tuning Stages

- Measure
  - Application tuning runs
  - Performance data collection
  - Identify metrics
- Analyze
  - Paradigm and programming model
  - Search space strategies
- Optimize
  - Apply identified optimizations
  - User knowledge
- Test
  - Re-evaluate



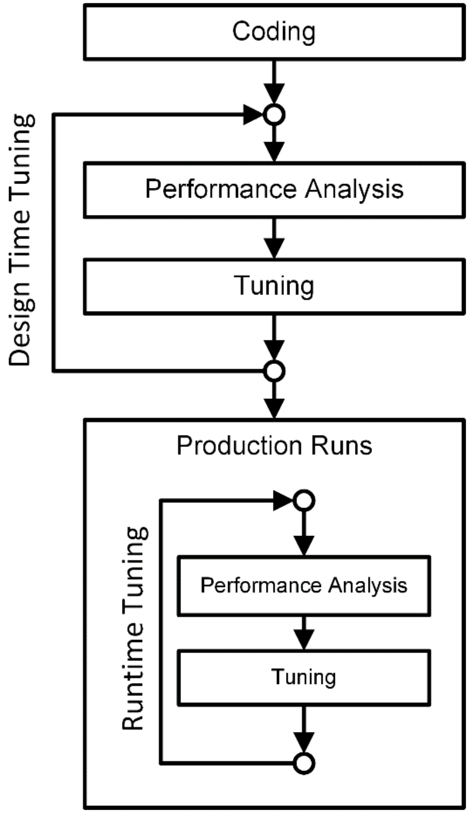
An Ideal Solution

- **Productivity**
  - Removes burden of tuning from developers
- **Portability**
  - Portable tuning techniques across different environments
- **Reusability**
  - Same techniques applied across different applications
- **Flexibility**
  - Re-evaluate optimizations for different scenarios
- **Performance**
  - Provides performance improvements

Periscope Tuning Framework

AutoTune
www.autotune-project.eu

- Objective - Single tool for **performance analysis** and **tuning**
- Extends Periscope with a tuning framework
- **Tuning plugins** for performance and energy efficiency tuning
- **Online** tuning
- Combine multiple tuning techniques



AutoTune Consortium

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

AutoTune Goals

- Automatic application tuning
  - Tune performance and energy
  - Create a scalable and distributed framework
  - Evaluate the alternatives online
  - Tune for Multicore and GPU accelerated systems

- Variety of parallel paradigms
  - MPI, OpenMP, OpenCL, Parallel pattern, HMPP

Review

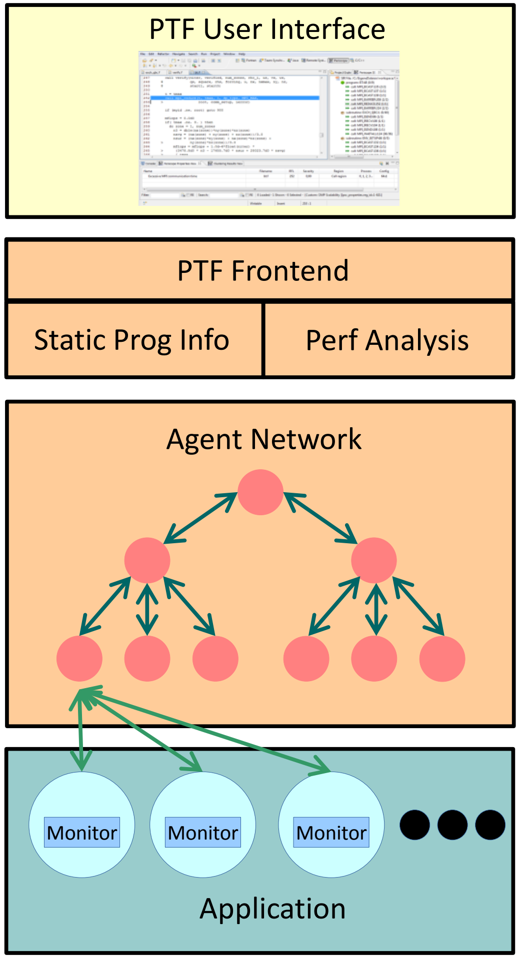
- Tune application
  - Automatic tuning is necessary because manual tuning is a time consuming and cumbersome process
- Ideal tool
  - Should be able to perform tuning automatically with improved productivity, flexibility, reusability and performance.
- AutoTune Project
  - AutoTune project is addressing

Progress

- Motivation
- AutoTune
- Periscope Tuning Framework (PTF)
  - Architecture
  - Plugins
  - Installation & Setup
- Tuning Plugins
  - CFS
  - MPI Parameters
  - DVFS
- Hands-on

Performance Analysis with Periscope

- Online
  - no need to store trace files
- Distributed
  - reduced network utilization
- Scalable
  - Up to 100000s of CPUs
- Multi-scenario analysis
  - Single-node Performance
  - MPI Communication
  - OpenMP
- Portable
  - Fortran, C with MPI & OMP
  - Intel Itanium2, x86 based systems
  - IBM Power6, BlueGene P, Cray

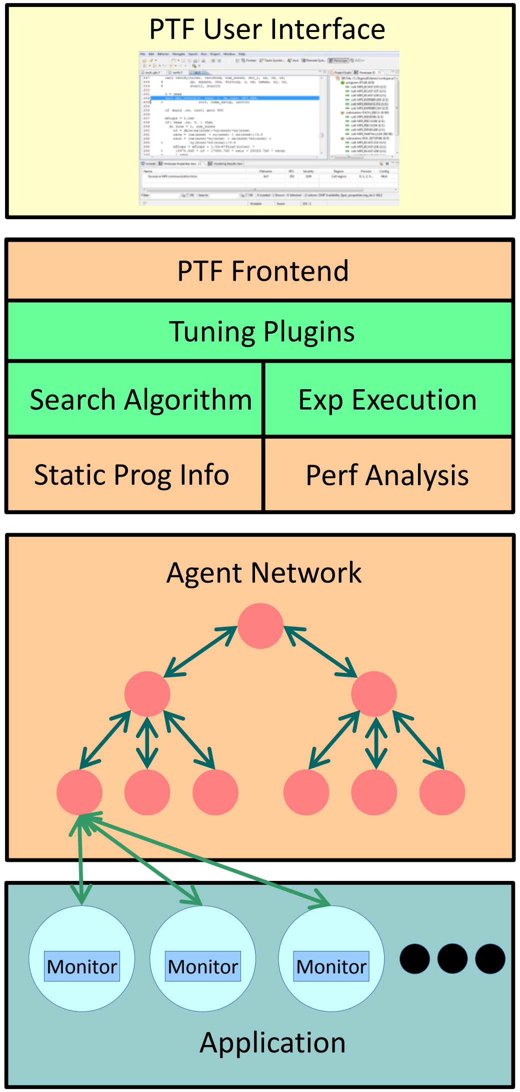


AutoTune Approach

- AutoTune will follow Periscope principles
  - Predefined tuning strategies combining performance analysis and tuning, online search, distributed processing.
- Plugins (online and semi-online)
  - Compiler based optimization
  - MPI tuning
  - Energy efficiency tuning
  - OpenCL tuning
  - Parallel pattern tuning
  - Master/Worker pattern tuning

Periscope Framework

- Extension of Periscope
- Online tuning process
  - Application phase-based
- Extensible via tuning plugins
  - Single tuning aspect
  - Combining multiple tuning aspects
- Rich framework for plugin implementation
- Automatic and parallel experiment execution



Plugin Overview

Name	Target	Objective
Compiler Flag Selection	Compiler flag combinations	Optimize performance
Dynamic Voltage and Frequency Scaling	DVFS settings	Reduce energy consumption with minimal impact
MPI Parameters	Parameters of MPI library	Optimize MPI application performance

Review

- Periscope Tool
  - Periscope performs the performance analysis
- Periscope Tuning Framework (PTF)
  - PTF extends the Periscope to use the performance data and allows developers to write tuning plugins
- Tuning Plugins
  - Tuning plugins uses PTF infrastructure and automate the tuning of applications using on or more tuning techniques

Progress

- Motivation
- AutoTune
- Periscope Tuning Framework (PTF)
  - Architecture
  - Plugins
  - Installation & Setup
- Tuning Plugins
  - CFS
  - MPI Parameters
  - DVFS
- Hands-on

Setting up PTF

- periscope.in.tum.de
- Version: 1.1
- Supported systems:
  - x86-based clusters
  - Bluegene
  - IBM Power
- License: New BSD

Setting up PTF

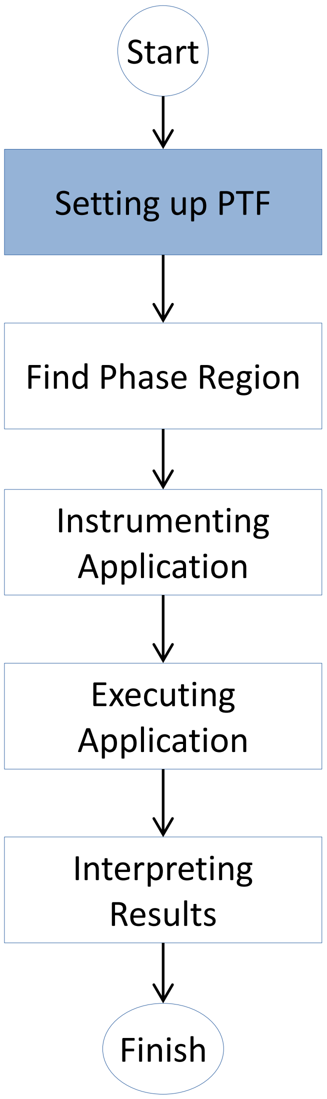
- Download PTF
  - http://periscope.in.tum.de/releases/latest/tar/PTF-latest.tar.bz2
- Installation of PTF
  - Uses AutoTools
- Third party library requirements
  - ACE (version >= 6.0)
  - Boost (version >= 1.47)
  - Xerces (version >= 2.7)
  - Papi (version >= 5.1)

Setting up PTF

- Plugin specific libraries
  - Enopt library for the DVFS plugin
  - Vpattern library for the Patterns plugin
- Doxygen documented code
- Plugin specific user guides
- Sample applications repository

Using PTF

- Identifying a Phase Region
  - Code region with repetitive computationally intensive part
  - PTF measurements focus on phase region
  - Optional step
  - Entire application is default phase region
- Instrumenting the application
  - Use `psc_instrument` command
- Executing application
  - Use periscope frontend for execution
  - Use `psc_frontend` command
- Interpreting the results

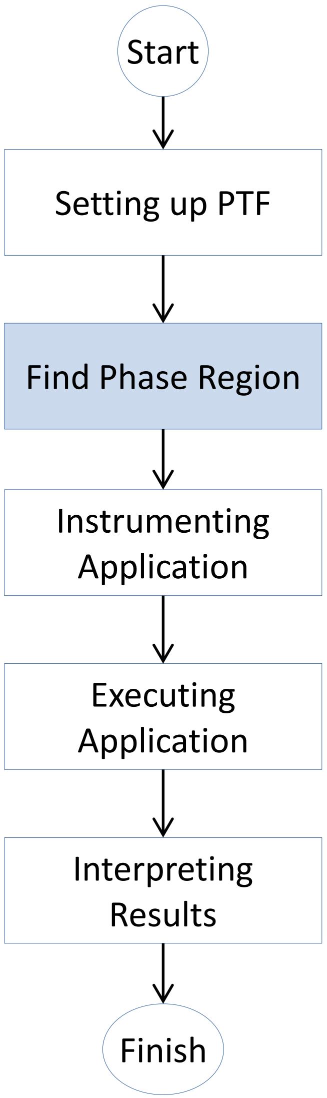


Finding Phase Region

If not known then find computationally intensive part using the output of gprof or equivalent tool

e.g. Serial NPB BT-MZ benchmark

```
-----------------------------------------------------------------
                                                   <spontaneous>
[2]     100.0     0.00   867.87                main [2]
                  0.02   867.85      1/1           MAIN__ [1]
-----------------------------------------------------------------
                  0.02   859.19  51456/51456       MAIN__ [1]
[3]      99.0     0.02   859.19  51456         adi_ [3]
                130.83   135.30  51456/51456       z_solve_ [4]
                103.15   139.31  51456/51456       y_solve_ [5]
                 94.49   134.66  51456/51456       x_solve_ [7]
                116.97     0.00  51456/51712       compute_rhs_ [9]
                  4.48     0.00  51456/51456       add_ [14]
-----------------------------------------------------------------
```

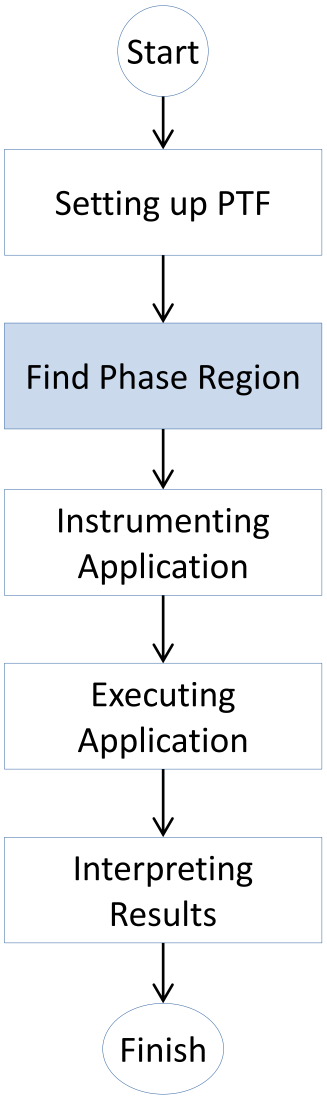


Finding Phase Region

Annotating the code within loop using pragmas

BT-MZ/bt.f

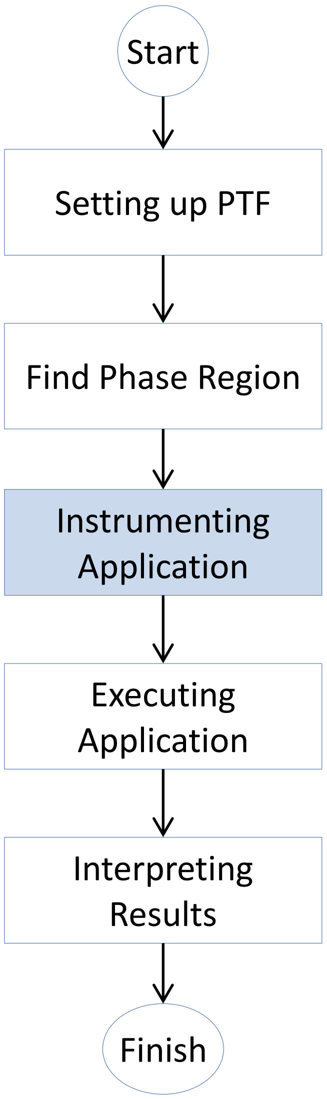
```
c---------------------------------------------------------------
c      start the benchmark time step loop
c---------------------------------------------------------------

       do  step = 1, niter

         if (mod(step, 20) .eq. 0 .or. step .eq. 1) then
            write(*, 200) step
 200        format(' Time step ', i4)
         endif

!$MON user region
         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
!$MON end user region

       end do
```



Adding Instrumentation

Changing the makefile to add instrumentation information

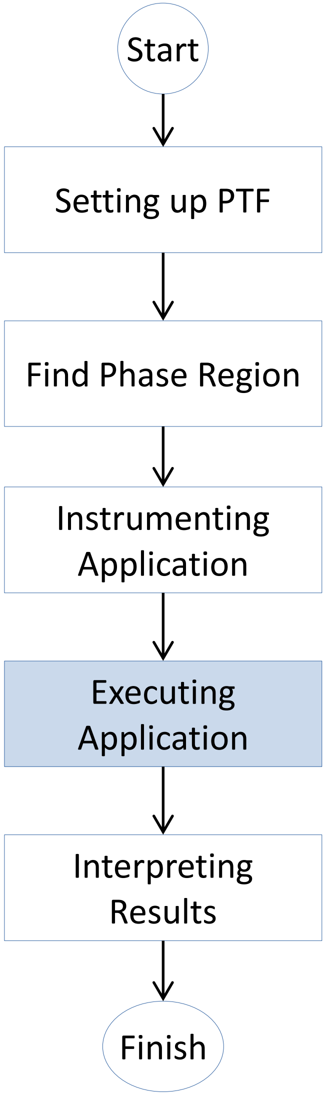
```
#-----------------------------------------------------------------------
# This is the fortran compiler used for fortran programs
#-----------------------------------------------------------------------

#F77 = gfortran -p
F77 = psc_instrument -v -d -s ../bin/bt-mz.C.x.sir -t user mpif90 -g
```



Prepare a config file

cfs_config.cfg

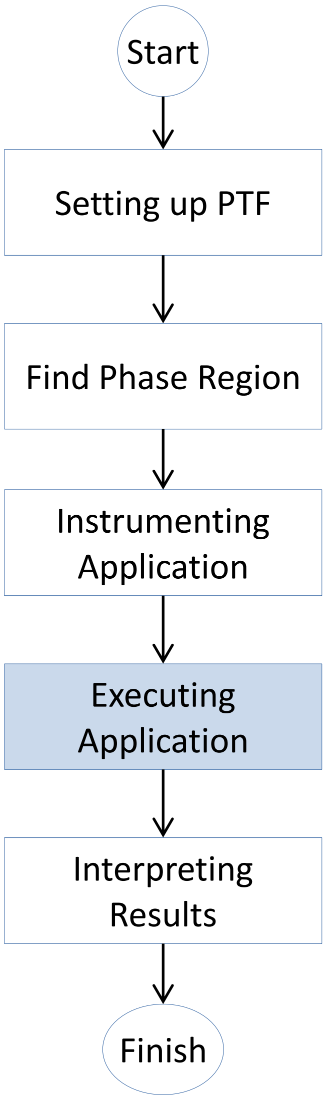
```
// ********** application related settings ***********
// the path to the Makefile
makefile_path="../";
// the variable containing the build flags
makefile_flags_var="FFLAGS";
// arguments for the make command
makefile_args="BT-MZ CLASS=C TARGET=BT-MZ";
// path to the source files of the application
application_src_path="../BT-MZ";
// **************************************************

// ********** plugin related settings ***************
//Details about the selective make
selective_file_list="x_solve.f y_solve.f z_solve.f";
make_selective="true";
// the desired search algorithm: exhaustive or individual
search_algorithm="exhaustive";
// the compiler flags to be considered in the search
tp "Opt" = "-" ["O1", "O2", "O3"];
// **************************************************
```



Executing Application

- psc_frontend command is used to execute PTF
- --tune=compilerflags to select the plugin

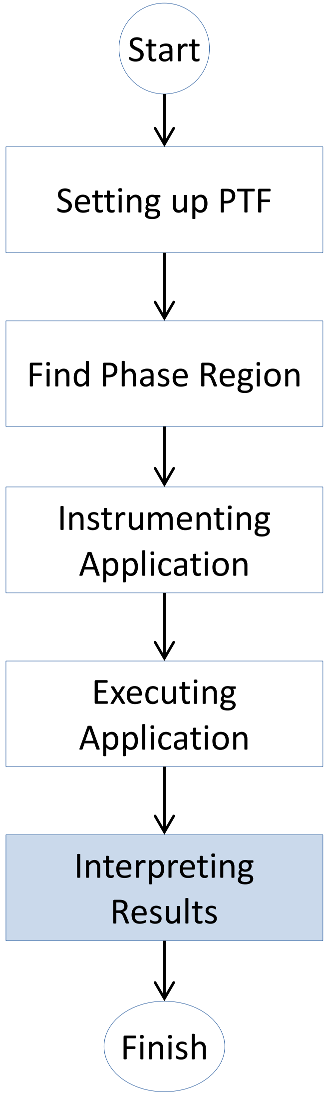
```
$ psc_frontend --apprun="./bt-mz.C.x"
 --starter=FastInteractive --delay=2
 --mpinumprocs=1 --tune=compilerflags
 --force-localhost
 --debug=2 --selective-debug=AutotuneAll
 --sir=bt-mz.C.x.sir
```



Interpreting the results

Combination executing in minimal time is reported as the optimal scenario

```
Optimum Scenario: 2

Compiler Flags tested:
Scenario 0 flags: "-O0 "
Scenario 1 flags: "-O2 "
Scenario 2 flags: "-O3 "


All Results:
Scenario          |  Severity
0                 |  4.9512
1                 |  3.98043
2                 |  3.95206

------------------------

----------------
 End Periscope run! Search took 69.3765 seconds
 ( 4.7041 seconds for startup  )
----------------
```



Review

- Installing Periscope Tuning Framework
- Using Periscope Tuning Framework
- Instrumenting and executing a sample application

Progress

- Motivation
- AutoTune
- Periscope Tuning Framework (PTF)
  - Architecture
  - Plugins
  - Installation & Setup
- Tuning Plugins
  - CFS
  - MPI Parameters
  - DVFS
- Hands-on

Compiler Flag Selection Plugin

- Goal:
  - Find best combination of compiler flags
- Features
  - Configuration file
  - Several search strategies
  - Selective make
  - Remote make
  - File-specific flag combinations

CFS Approach

- Search strategy specifies combinations to be evaluated. Called scenarios.
- For each scenario
  – Recompile the application
  – Restart the application
  – Measure phase region / main region
- Evaluate all scenarios
- Output the best configuration

CFS Configuration File

- Name
  - Default: cfs_config.cfg
  - Command line argument: --cfs-config=cfs_config.cfg
- Contents
  - Build information

```
makefile_path="../";
makefile_flags_var="PSC_FLAGS";
makefile_args="BT-MZ CLASS=C NPROCS=4";
application_src_path="../BT-MZ";
```

CFS Configuration File

- Selective make

```
make_selective="true";
selective_file_list="x_solve.f y_solve.f
                     z_solve.f";
```

- Search algorithm

```
search_algorithm="individual";
//gde3, exhaustive, random
individual_keep=1;
```

CFS Configuration File

- Flags

```
compiler="ifort";

tp "TP_OPT" = "-" ["O2", "O3"];
tp "TP_XHOST"  = " " ["-xhost", " "];
tp "TP_ALIAS" = "-f" ["no-alias","alias"];
tp "TP_UNROLL" = "-unroll=" [5,20,5];
tp "TP_PREFETCH" = " " ["-opt-prefetch", " "];
tp "TP_IP" = " " ["-ip", " "];
```

CFS Configuration File

- Remote make
  - Requires PPK

```
remote_make="true";
identity_path="~/.ssh/identity";
remote_make_machine_name="login05";
```

MPI Parameters

- Goals
  - Tuning MPI performance
  - Application specific setting of MPI library parameters
- Features
  - Configuration file
  - Template files for Intel MPI, IBM MPI, OpenMPI
  - Multiple search strategies
  - Automatic eager threshold tuning

MPI Parameters Approach

- Search strategy generates scenarios with different parameter combinations and settings.
- Application is restarted for each scenario.
- Parameter passing depends on the MPI flavor
- Measure execution time for phase region / main region
- Evaluate scenarios
- Output best setting and search path

MPI Parameters Configuration File

- Name
  - Default: param_spec.conf
  - Templates for MPI flavors at
    - $PERISCOPE_ROOT/templates
- Contents
  - MPI flavor
  - Search algorithm

```
MPIPO_BEGIN intel
    ...
    SEARCH=gde3;
MPIPO_END
```

MPI Parameters Configuration File

- Parameter specification

```
I_MPI_EAGER_THRESHOLD=4096:2048:65560;
I_MPI_INTRAN_EAGER_THRESHOLD=4096:2048:65560;
I_MPI_SHM_LMT=shm,direct,no;
I_MPI_SPIN_COUNT=1:2:500;
```

DVFS Plugin

- Goals
  - Reduce energy consumption
  - Limit application delay
- Features
  - Configuration via environment variables
  - Different objective functions
  - Frequency range specification
  - Individual region tuning

DVFS Plugin Approach

- Analysis run to determine application properties.
- Evaluation of energy model to predict best frequency
- Measurement of a configurable number of frequencies around the predicted frequency

DVFS Plugin Configuration

- Configuration via environment variables
- PSC_DVFS_MODEL

  1: Energy

  3: Energy-Delay Product

  4: Total Cost of Ownership
- PSC_FREQ_TO_ALL_NODE

  1: Frequency will be set for all cores
- PSC_FREQ_NEIGHBORS

  N: number of neighbor frequencies in both directions

Progress

- Motivation
- AutoTune
- Periscope Tuning Framework (PTF)
  - Architecture
  - Plugins
  - Installation & Setup
- Tuning Plugins
  - CFS
  - MPI Parameters
  - DVFS
- Hands-on

Hands-on Session

- Configure your account for hands-on
- Compile NAS Parallel Benchmark BT-MZ
  - NPB 3.3 Benchmark suite
  - Multiple problem classes A, B, C, …
  - Compiled for fixed number of processes
  - Hybrid parallelization
- Run
  - CFS plugin
  - MPI Parameters plugin
  - DVFS plugin
- Explain setup for heat code