Code Auto-Tuning with the Periscope Tuning Framework

Renato Miceli, SENAI CIMATEC
renato.miceli@fieb.org.br

Isaías A. Comprés, TUM
compresu@in.tum.de
Project Participants

• **Michael Gerndt**, TUM
  Coordinator

• **Renato Miceli**, SENAI CIMATEC
  WP5: Evaluation Leader at ICHEC

• **Isaías A. Comprés**, TUM
  Periscope Plugin Developer

• **Siegfried Benkner**, University of Vienna
  WP2, WP4: Work package Leader
Tutorial Overview

• Users of PTF
  – Motivation
  – The AutoTune Project
  – Periscope Tuning Framework (PTF)
    • Architecture
    • Plugins
    • Installation & Setup
  – Demo
  – Case Study & Best Practices
    • NPB, SeisSol, LULESH
  – AutoTune Demonstration Center

• Plugin Developers
Background

• Given
  – Multicore, Accelerators, HPC Cluster
  – Many programming models and applications

• Problem
  – How to tune application for a given architecture & application?

• 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

- **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
• 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
• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH
• AutoTune Demonstration Center
Periscope Performance Analysis Toolkit

- **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
  - HMPP tuning for GPUs
  - Parallel pattern tuning
  - MPI tuning
  - Energy efficiency tuning
  - User defined plugins - info in developer session
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 Lifecycle

Static Analysis and Instrumentation

Analysis using Periscope Frontend

Tuning Strategy

Analysis Strategy

Plugin Strategy

Generate Tuning Report
Progress

• Motivation
• AutoTune

• Periscope Tuning Framework (PTF)
  – Architecture
  – Plugins
  – Installation & Setup

• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH

• AutoTune Demonstration Center
## Plugin Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Target</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Flag Selection (CFS)</td>
<td>Compiler flag combinations</td>
<td>Optimize performance</td>
</tr>
<tr>
<td>Dynamic Voltage and Frequency Scaling (DVFS)</td>
<td>CPU Frequency and power states</td>
<td>Reduce energy consumption with minimal impact</td>
</tr>
<tr>
<td>High Level Pipeline Patterns</td>
<td>Pipeline stages</td>
<td>Optimize throughput</td>
</tr>
<tr>
<td>MPI Runtime</td>
<td>MPI parameters</td>
<td>Optimize SPMD MPI application performance</td>
</tr>
<tr>
<td>Master – Worker MPI</td>
<td>Workload imbalance in MPI applications</td>
<td>Balance application load by optimizing communication</td>
</tr>
<tr>
<td>OpenCL Plugin</td>
<td>Kernel tuning based on NDRange</td>
<td>Improve utilization of many core architecture</td>
</tr>
</tbody>
</table>

May be more detailed information already has for the plugins.
Compiler Flag Selection (CFS)

• **Goal**
  – Optimize application performance by guiding machine code generation using compiler flags

• **Tuning Technique**
  – Selection of compiler flags and corresponding values
    • -O1, O2, -floop-unroll
  – Search through combination of flags
Dynamic Voltage and Frequency Scaling (DVFS)

• **Goal**
  – Reduce energy consumption with the objective of optimal performance to energy ratio

• **Tuning Technique**
  – Select CPU governor and frequency, processor specific power states
  – Energy prediction model avoids evaluating all frequency governor combinations
High Level Pipeline Patterns

• **Goal**
  – Optimizing throughput of pipeline patterns by exploiting CPU and GPU effectively

• **Tuning Technique**
  – Selection of stage replication factor
  – Sizes of intermediate buffers
  – Choice of hardware for executing a task
MPI Runtime

• **Goal**
  – Tweaking MPI Parameters to improve SPMD code performance

• **Tuning Technique**
  – MPI runtime parameters – application mapping and buffer/protocol
  – MPI communication parameters - Eager limit, buffer limit
Master-Worker MPI

• **Goal**
  – Optimize execution time by balancing application load and decreasing communication between workers

• **Tuning Technique**
  – Impact of varying partition size per worker
  – Number of workers
  – Size of partition per worker

![Performance Index Graph](image)

- Size
- Explanation

what is this?
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

www.autotune-project.eu
Progress

• Motivation
• AutoTune
• Periscope Tuning Framework (PTF)
  – Architecture
  – Plugins
  – Installation & Setup
• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH
• AutoTune Demonstration Center
Setting up PTF

- periscope.in.tum.de
- Version: 1.0 (1.1 coming in April 2015)
- 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

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>time</th>
<th>cumulative seconds</th>
<th>self seconds</th>
<th>calls</th>
<th>s/call</th>
<th>self s/call</th>
<th>total s/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.35</td>
<td>237.34</td>
<td>237.34</td>
<td>2110313472</td>
<td>0.00</td>
<td>0.00</td>
<td>binvcrhs_</td>
<td></td>
</tr>
<tr>
<td>15.08</td>
<td>368.18</td>
<td>130.83</td>
<td>51456</td>
<td>0.00</td>
<td>0.01</td>
<td>z_solve_</td>
<td></td>
</tr>
<tr>
<td>14.49</td>
<td>493.93</td>
<td>125.75</td>
<td>2110313472</td>
<td>0.00</td>
<td>0.00</td>
<td>matmul_sub_</td>
<td></td>
</tr>
<tr>
<td>12.55</td>
<td>611.48</td>
<td>117.55</td>
<td>51712</td>
<td>0.00</td>
<td>0.00</td>
<td>compute_rhs_</td>
<td></td>
</tr>
<tr>
<td>11.63</td>
<td>729.11</td>
<td>103.15</td>
<td>51456</td>
<td>0.00</td>
<td>0.00</td>
<td>y_solve_</td>
<td></td>
</tr>
<tr>
<td>10.89</td>
<td>809.12</td>
<td>94.49</td>
<td>51456</td>
<td>0.00</td>
<td>0.00</td>
<td>x_solve_</td>
<td></td>
</tr>
<tr>
<td>3.70</td>
<td>841.24</td>
<td>32.11</td>
<td>2110313472</td>
<td>0.00</td>
<td>0.00</td>
<td>matvec_sub_</td>
<td></td>
</tr>
</tbody>
</table>

Find Phase Region

Instrumenting Application

Interpreting Results

Finish
Finding Phase Region

Annotating the code within loop using pragmas

```c
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)),
                 gs(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
```

Start

- Setting up PTF
- Find Phase Region
- Instrumenting Application
- Executing Application
- Interpreting Results
- Finish
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
```
Executing Application

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" = ";" ["01", "02", "03"];

// ***********************************************
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: "-00"
Scenario 1 flags: "-02"
Scenario 2 flags: "-03"

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 )
-----------------------------
Demo
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

• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH

• AutoTune Demonstration Center
Tuning Results

• Comparison of manual tuning against automatic tuning

![Chart showing time comparison](chart.png)

- Baseline (With -O2) Time: 872.4 seconds
- Manual Tuning Time: 797.8 seconds
- PTF Suggested Flags Time: 797.5 seconds
Selecting Search Strategy

• Search strategy influencing the number of scenarios executed and total execution time of the plugin.
Tuning Results

• Improvement in the PTF execution time to find the optimal flags for BT-MZ class C
Progress

• Motivation
• AutoTune
• Periscope Tuning Framework ( PTF )
  – Architecture
  – Plugins
  – Installation & Setup
• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH
• AutoTune Demonstration Center
SeisSol Use Case

• Developed by the Department of Earth and Environmental Sciences at the Ludwig-Maximilian University.

• Application simulates realistic earthquakes, propagation of seismic waves e.g. viscoelastic attenuation, strong material heterogeneities and anisotropy.

• Code Details
  – FORTRAN Code
  – MPI
SeisSol Use Case

• Execution of SeisSol with DVFS plugin

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Governor</th>
<th>Freq (MHz)</th>
<th>Energy (J)</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Userspace</td>
<td>1200</td>
<td>840.000</td>
<td>21.487</td>
</tr>
<tr>
<td>1</td>
<td>Userspace</td>
<td>1300</td>
<td>1140.000</td>
<td>19.290</td>
</tr>
</tbody>
</table>

• Energy Prediction Model predicted the optimal frequency as 1.2GHz
• Plugin tests +/- 100MHz step for the plugin
• Here 1.2GHz is the minimum frequency supported by hardware
• ~26% energy is saved at the cost of ~10% increased execution time
Progress

• Motivation
• AutoTune
• Periscope Tuning Framework (PTF)
  – Architecture
  – Plugins
  – Installation & Setup

• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH

• AutoTune Demonstration Center
LULESH Use Case

• Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)
• Modelling hydrodynamics
• DOE proxy applications

• Code Details
  – C++ Code
  – Serial, OMP, MPI, MPI+OMP
LULESH Use Case

• CFS Tuning Results

![Chart showing comparison between Baseline and PTF](chart.png)
## LULESH Use Case

- **Execution Results with the DVFS**

  - **1 iteration**

    | Scenario | Governor   | Freq (MHz) | Energy (J) | Runtime (s) |
    |----------|------------|------------|------------|-------------|
    | 0        | Userspace  | 1200       | 4.000      | 0.302       |
    | 1        | Userspace  | 1300       | 4.000      | 0.371       |

  - **1000 iterations**

    | Scenario | Governor   | Freq (MHz) | Energy (J) | Runtime (s) |
    |----------|------------|------------|------------|-------------|
    | 0        | Userspace  | 1200       | 2709.000   | 115.537     |
    | 1        | Userspace  | 1300       | 2740.000   | 106.674     |

More clear: How related to production runs.
Meta-plugins

• Meta Plugins
  – Fixed Sequence Plugin
    • Executes selected set of plugins in the given order
  – Adaptive Sequence Plugin
    • Uses the optimal scenario found by the previous plugin
Progress

• Motivation
• AutoTune
• Periscope Tuning Framework (PTF)
  – Architecture
  – Plugins
  – Installation & Setup
• Case Study & Best Practices
  – NPB
  – SeisSol
  – LULESH
• AutoTune Demonstration Center
AutoTune Demonstration Center

- The AutoTune Demonstration Centre (ADC)
  - Established by AutoTune partners

- The ADC objective are to:
  - Spread the use of AutoTune software
  - Further exploit energy save and power steering
  - Disseminate Best Practice Guides
  - Provide developers with a platform to test/validate
  - Become a platform for exchange of information
  - To extend the PTF developer community

- The ADC will offer the following services:
  - Online documentations, Best Practice Guides
  - Discussion forums
  - AutoTune training events
  - Individual support for applications
  - Education and training of students
  - AutoTune website as central hub
  - Service Desk and Issue Tracking

Structure of the ADC

www.autotune-project.eu
Conclusion

- Automatic Tuning - Motivation and AutoTune
- Periscope Tuning Framework (PTF)
- Tuning Plugins
- Instrumenting and executing sample application
- Demo: NPB, SeisSol, LULESH
- Best Practices
- AutoTune Demonstration Center