Specification of Periscope Tuning Framework Plugins

Robert Mijaković¹, Antonio Pimenta Soto², Isaías A. Comprés Ureña¹, Michael Gerndt¹, Anna Sikora², Eduardo César²

¹Technische Universität München
²Universitat Autònoma de Barcelona
Overview

• AutoTune goals and partners

• Periscope and Periscope Tuning Framework (PTF)

• Tuning Plugin Workflow

• MPI Parameters Plugin

• Evaluation and Conclusion
AutoTune

- Tackle complexity of HPC architectures
  - Multicore, multisocket, accelerators, DVFS
- Enable higher productivity via auto-tuning
- Focus on static tuning in pre-production phase
  - Produce tuning recommendations
- Leverage state of the art performance analysis
- Implement an extensible environment
  - Support open and propriety plugins
- Focus of available tuning plugins:
  - Tuning of High-Level Patterns for GPGPU
  - Tuning of HMPP Codelets
  - Tuning of Energy Consumption via CPU frequency
  - Tuning of MPI Runtime
  - Tuning of Compiler Flag Selection
Partners

Technische Universität München

Universität Wien

CAPS Entreprises

Universitat Autònoma de Barcelona

Leibnitz Computing Centre

National University of Galaway, ICHEC
Periscope

- Automated search
  - Based on formalized performance properties
- Online analysis
  - Search performed while application is executing
  - No need to store trace files
- Distributed search
  - Reduced network overhead
  - Base on autonomous cooperating agents
- Analyzes:
  - MPI Communication
  - Single-node Performance
  - OpenMP Performance
- Supports: Fortran, C++
Periscope Tuning Framework (PTF)

• Goals
  – Combine performance analysis and tuning to speedup the tuning process
  – Tune codes to improve performance and energy efficiency
  – Support multicore and GPU accelerated parallel systems
  – The whole tuning process executed online

• Idea:
  – Automatically evaluate optimization space
  – Produce tuning recommendation
  – Use it to improve production runs
Terminology

- **Tuning point (TP):**
  - Features for influencing the execution of a region, e.g. frequency
- **Tuning space \((TS_P)\):**
  - Definition: Tuning Space of a plugin \(P\) define a multidimensional tuning space
- **Variant of a program region \(r (v_r)\):**
  - Concrete vector of values for the region’s tuning points
- **Variant space of a program region \((VS_r)\):**
  - A subset of the overall tuning space
- **Objective**
  - A function objective: \(REG_{app} \times TS_P \rightarrow \mathbb{R}\)
- **Tuning scenario**
  - A tuple \(sc_r = (r, v_r, \{obj_1, ..., obj_n\})\)
- **Tuning action \((TA_i)\):**
  - Enforces for each tuning point \(TP_i\) during the execution of a scenario
  - Runtime tuning actions: variables and functions
  - Pre-runtime tuning actions: Compilation, \#MPI tasks, MPI Parameters
Combining tuning and analysis

• Speeds up the tuning process
• Use case:
  – Shrinking the search space
    • Reducing tuning regions
    • Reducing a range of values for a tuning point
  – Checking the influence of a scenario on other regions
  – Guiding the selection of a tuning plugin
Tuning Model

- Tuning point: Type: variable, Name: TEST, Range: 1..10
- Tuning space: TS = 1..10
- Variant space = TS
- Objective: ExecTime
- Tuning scenario: ((1, 485), (1), {ExecTime}) to ((1, 485), (10), {ExecTime})
- Tuning action: Assign 1 to variable variant in start_region(1, 485)

```fortran
do k = 1, 20
    variant = k
    !$MON USERREGION TP name(Test) variable(variant) variants(10)
    tstart=MPI_Wtime()
    mm=5-variant+1
    if (mm<=0)
        m=-1*mm+1
    endif
    call sleep(mm)
    tend=MPI_Wtime()
    write (*,*), myrank, variant, tend-tstart
    !$MON END USERREGION
endo
```
Tuning Plugin Workflow

- **Scenario pools:**
  - Accessible by all plugins and the frontend
  - Created Scenario Pool (CSP)
    - Scenarios created by a search algorithm
  - Prepared Scenario Pool (PSP)
    - Scenarios already prepared for execution
  - Experiment Scenario Pool (ESP)
    - Scenarios selected for the next experiment
  - Finished Scenario Pool (FSP)
    - Scenarios executed

- **Tuning Plugin Interface:**
  - The functions that have to be implemented

Control flow

Data flow
MPI Parameters Plugin: Introduction

• Tuning Objective
  – Reduce wall time of the user region execution based on finding the right combination of user provided MPI parameters

• Tuning Points
  – MPI environment parameters
    • MPI application mapping
  – MPI communication buffer/protocol
    • Adapting the sending/receiving buffer size – in the paper
    • Analyzing the size pattern of the messages
    • Adapting the communication protocol (eager/randevouz) – in the paper
  – Code variants for MPI communication

• Tuning Actions
  – Set a variant of the parameters in the preparation phase of a scenario before reexecuting
MPI Parameters Plugin: Control Flow

- Initialize Tuning Points from a configuration file
  - Parameter_Name=<comma separated list>
  - Parameter_Name=<initial:step:final>
- Create new command line
  - Before application restarts
- Use the command line built in the preparation
- Restarts the application
- Process Results
  - Retrieve the best variant from the search algorithm
  - Give it as tuning recommendation
MPI Parameters Plugin: Evaluation

- Testing application
  - FSSIM, i.e. biological simulator that models the movement of large fish schools
  - 256k individuals and 64 threads
- Search strategy
  - Exhaustive search strategy
- Objective function
  - Minimum function to select the scenario that executed in less time
- Tuning points
  - Adapting the sending/receiving buffer size, i.e. BUFFER_MEM, configurable up to 256 MB
  - Adapting the communication protocol (eager/randevouz), i.e. EAGER_LIMIT, configurable from 0..64 kB
MPI Parameters Plugin: Evaluation

- Analysis reduced ranges of tuning points:
  - BUFFER_MEM: 32 MB, 64 MB
  - EAGER_LIMIT: 1 kB..8 kB, step 2x
  - Tuning space of 8 points
- Histogram of message sizes
MPI Parameters Plugin: Evaluation: Results

- AutoTune Results:
  - Optimum scenario 3
    - BUFFER_MEM: 64 MB
    - EAGER_LIMIT: 2 kB
    - Execution time: 39.07 s
  - Low impact of memory buffer sizes
  - The eager limit represents a valid tuning point
- Execution time for different EAGER_LIMIT
THANK YOU

QUESTIONS?