Periscope Tuning Framework: Automatic Performance Engineering for HPC

Prof. Dr. Michael Gerndt
Technische Universität München
gerndt@in.tum.de
Performance Analysis and Tuning is Essential
AutoTune FP7 Project Goals

• Extend Periscope for automatic tuning
  – Performance and energy

• Support wide spectrum of HPC systems
  – Homogeneous and heterogeneous
  – Focus on SuperMUC

• Provide an easily extensible tuning framework
  – Tuning plugins
  – Interface hides Periscope details but provides support by Periscope’s rich infrastructure
Partners

Technische Universität München

Universität Wien

CAPS Entreprises

Universitat Autònoma de Barcelona

Leibniz Computing Centre

National University of Galaway, ICHEC
• Automated search
  – Based on formalized performance properties
• Online analysis
  – Search performed while application is executing
• Distributed search
  – User specified number of analysis agents
  – Additional cores for agents
• Profile data only
  – even for MPI Waittime analysis
Properties

• StallCycles(Region, Rank, Thread, Metric, Phase)
  – Condition: Percentage of lost cycles >30%
  – Confidence: 1.0
  – Severity: Percentage of lost cycles

• StallCyclesIntegerLoads
  – Requires access to two counters

• L3MissesDominatingMemoryAccess
  – Condition: Importance of L3 misses (theoretical latencies)
  – Severity: Importance multiplied by actual stall cycles
Periscope Design

Frontend

Performance Analysis Agent Network
- Master Agent
- Communication Agent
- Analysis Agent

MRI

Application with Monitor
Agent Search Strategies

• Application phase is a period of program’s execution
  – Phase regions
    • Full program
    • Single user region assumed to be repetitive
  – Phase boundaries have to be global (SPMD programs)

• Search strategies
  – Determine hypothesis refinement
    • Region nesting
    • Property hierarchy-based refinement
  – Single and multi step strategies
Where is the problem?

What is the most severe problem?

Filter problems for region
Autotune Approach

• Predefined tuning plugins combining performance analysis and tuning

• Plugins
  – Compiler based optimization
  – HMPP tuning for GPUs
  – Parallel pattern tuning
  – MPI tuning
  – Energy efficiency tuning
Periscope Tuning Framework

• Online
  – Analysis and evaluation of tuned version in single application run
  – Multiple versions in single step due to parallelism in application

• Result
  – Tuning recommendation
  – Adaptation of source code and/or execution environment
  – Impact on production runs
Extensions to Periscope

Frontend
- Tuning Plugin
- Search Strategies
- Scenario Execution Engine

Analysis Agent
- Tuning Strategy

Monitor Request Interface
- Tuning Action Requests

Monitor
- Tuning Actions
Tuning Plugin

• Defines tuning space
  – Crossproduct of tuning parameters

• Goes through single/multiple tuning steps
  – Selection of a variant space
  – Find best variant in this space by generating and executing tuning scenarios

• Searching the variant space can make use of predefined search algorithms.

• Provides functions that can be called by
  – Frontend
  – Meta Tuning Plugins
Tuning Objectives

• Tuning searches for variant(s) with best value for a single or multiple objectives

• Objectives are implemented as Periscope properties.
  – Properties specify measurements and return a severity, i.e. the objective value.
  – They are automatically evaluated by the analysis agents based on the AA Tuning Strategy
Tuning Scenarios

• Specify a single variant
  – Region to be tuned
  – Tuning action/value pairs
  – Properties for objective function

• Life cycle
  1. Creation by search algorithm -> Scenario Pool
  2. Preparation -> Prepared Scenario Pool
  3. Selection for experiment -> Experiment Scenario Pool
  4. Evaluation -> Finished Scenario Pool

• Steps 1-3 provided by plugin functions
• Step 4 executed by Scenario Execution Engine
Tuning Actions

• Monitor Request Interface (MRI)
  – Configuration of monitor
  – Application control

• MRI tuning actions
  – Variable tuning action
  – Function tuning action

• General tuning actions
  – During preparation of scenarios by tuning plugin
  – During restart of the application
  – During execution
Development of Plugins

• Determine tuning points with tuning actions
• Define (intelligent) search algorithm
  – Predefined search algorithm
  – Plugin-specific search algorithm
  – Combination of both
• Provide functions for
  – Creation and preparation of scenarios
  – Optional recompilation
  – Optional restart parameters
  – Selection of scenarios for next experiment
  – Evaluation of experiment results
Compiler Flag Selection Plugin

- Single node performance is of utmost importance for overall performance and energy efficiency.
- Modern compilers have heuristics for application tuning that might not be optimal.
- The plugin allows to automatically evaluate compiler flags based on expert knowledge.
- Features:
  - Automatic recompilation
  - Multiple plugin search strategies: Exhaustive and Individual
  - Selective compilation of compute intensive files
  - Flexible specification of compiler flags to explore
OpenACC Tuning Plugin for GPUs

- **Goal**
  - Optimization of HMPP Codelet computations on accelerators
  - Adjust kernel execution time to the many-core architecture

- **Tuning Parameters**
  - Selection of a static codelet variant
  - Modification of runtime conditions
    - GPU grid size, ...
  - Tuning points are set at an HMPP callsite location

- **Tuning Actions**
  - Select codelet variant

- **Prepare for tuning**
  - The user provides statically a set of codelet variants
  - Generate information on the location of callsite regions in the code & related tuning points for codelet variants and runtime configuration

- **Tuning results**
  - Retrieve the codelet variant and the runtime configuration with the minimal execution time
Tuning Plugin of High-Level Pipeline Patterns for CPU/GPU

- High-Level Component-based Parallelization
  - Multi-architectural components
  - Asynchronous, task-based execution model
  - Intelligent runtime system (StarPU)
  - cf. European PEPPHER project

- Tuning of pipeline structure
  - OO Coordination Layer
  - Replication factors, buffer sizes

- Tuning of runtime properties
  - StarPU scheduling policies
  - Number of CPUs and GPUs

Image processing pipeline with user-provided hints

Integration of the Pipeline Coordination Layer with PTF

PERISCOPE TUNING FRAMEWORK

Tuning Plugin for High-Level Parallel Patterns
MPI Tuning Plugin

- MPI parameters
  - Easy to automate tuning
  - Many parameters available
    MP_COLLECTIVE_OFFLOAD
    MP_USE_BULK_XFER
    MP_EAGER_LIMIT

- Master/Worker applications
  - Easy to automate tuning
  - Number of workers
  - Data partition factor
Energy Efficiency Tuning Plugin

Tuning Plugin for Energy Consumption via CPUFreq using enopt library (libenopt)

• Aim
  – Optimize the energy consumption of an arbitrary application, by choosing the best combination of CPUFreq parameters for each code region.

• Integration with Periscope
  – The start of each code region calls (per callback) the corresponding libenopt function to change:
    • The CPU governor
    • The CPU frequency
  – The code is executed for each combination of frequencies and governors, looking for the minima energy consumption.
Pathway Features

• **Formal performance engineering workflows**
  – Pre-defined workflows available
  – Can be customized with graphical editor
  – Can be explained to new team members

• **Transparently serve different HPC systems**
  – Define connection parameters for each system
  – PAThWay automatically generates batch scripts for different schedulers
  – Integrates with Eclipse’s Parallel Tools Platform (PTP)
Pathway Features

• Creates a snapshot of your application source
  – Does not interfere with your development repository

• Automatic tool invocations
  – Several tools from VI-HPS are preconfigured
    • Scalasca
    • Periscope
    • Score-P
  – Others can be added through configuration dialogs
  – More preconfigurations are planned
## Experiment Browser

### Experiments overview, filtered and sorted

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>HPC System</th>
<th>Application</th>
<th>MPI Process</th>
<th>OMP Threads</th>
<th>Performance Tool</th>
<th>Job status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 22, 2013</td>
<td>Oct 15, 2013 3:40:41 PM CEST</td>
<td>Local</td>
<td>omp_prime_sum</td>
<td>1</td>
<td>1</td>
<td>Uninstrumented</td>
<td>Completed</td>
</tr>
</tbody>
</table>

### Review results, environment settings, performance data, etc.

- Number of threads: 5
- Sum: 7261954630277
- Time: 17.138833
PATHWAY_NOTE: 20130829

Experiments for 29. August 2013

- Cross-Platform Memory Analysis - PATHWay:Exp:ff80818140ca86920140ca87702d0000
  - Application: my_mpi_test - default
  - Tool: Uninstrumented
  - Number of sub-experiments: 2
    - MPI: 2 / OpenMP: 1 (LRZ Linux Cluster New): ff80818140ca86920140ca87702d0000
    - MPI: 4 / OpenMP: 1 (LRZ Linux Cluster New): ff80818140ca86920140ca87a5330001

- Cross-Platform Memory Analysis - PATHWay:Exp:ff80818140ca62750140ca65a06b0000
  - Application: my_mpi_test - default
  - Tool: Uninstrumented
  - Number of sub-experiments: 2
    - MPI: 2 / OpenMP: 1 (LRZ Linux Cluster New): ff80818140ca62750140ca65a06b0000
    - MPI: 4 / OpenMP: 1 (LRZ Linux Cluster New): ff80818140ca62750140ca65d7ca0001

- Scalability Analysis - PATHWay:Exp:ff80818140cac69b0140cac6d7f70000
  - Application: my_mpi_test - local
  - Tool: Uninstrumented
  - Number of sub-experiments: 8
    - MPI: 1 / OpenMP: 1 (Local): ff80818140cac69b0140cac6d7f70000
    - MPI: 2 / OpenMP: 1 (Local): ff80818140cac69b0140cac6dc330001
Scalability Analysis Workflow

Local System

Start New Analysis
- Create Run Configurations
- Select Desired Configurations
- Select Number of Processes
- Compare & Display Results
- Select Target System
- Store Results (Location), Environment & Other Info
- Select Performance Analysis Tool
- Create Exec Script(s) (Batch/Interactive)
- Get Inputs, Tools, Systems
- H2 Database
- Start Experiments
- Get Running Environment
- Perform Experiment
- More Experiments?
  - YES
  - NO
- End Experiments
- End Analysis

Remote System
Status

- Infrastructure and Tuning Plugin Interface available
- Prototype tuning plugins are available
- Additional plugins under development
  - Energy tuning via thread throttling
  - MPI IO tuning
  - MPI parameter tuning via MPIT
- Integration with other projects
  - InvasIC (TRR 89)
  - Score-E
THANK YOU
do k=1,20
  variant=k
  !$MON USERREGION TP name(Test) variable(variant) variants(10)
  
  tstart=MPI_Wtime()
  call sleep(5-variant+1)
  tend=MPI_Wtime()
  
  write (*,*) myrank, variant, tend-tstart
  
  !$MON END USERREGION
enddo
Search Algorithms

• **Purpose**
  – Find the best variant in a multidimensional tuning space

• **Input**
  – Search space: region to be tuned, variant space, objectives

• **Output**
  – Best variant according to given objective
  – Search path with objective values

• **Algorithms**
  – Single or multistep algorithms
  – Exhaustive search, simplex algorithm from Active Harmony
AA Tuning Strategy

1. Select relevant scenarios
   - Scenario assigned to an MPI process that is controlled by the analysis agent.
2. Request tuning actions via MRI
3. Request objective measurements
4. Collect measurements (single/multiple phases)
5. Evaluate the objectives
Scenario Execution Engine

- Runs the experiment scenarios
  - Forwards the scenarios to the analysis agents
  - Starts the AA tuning strategy

- Provides the objective values for the scenarios