READEX: A Tool Suite for Dynamic Energy Tuning

Michael Gerndt
Technische Universität München
SuperMUC: 3 Petaflops, 3 MW
Runtime Exploitation of Application Dynamism for Energy-efficient eXascale Computing
09/2015 to 08/2018
www.readex.eu
Objectives

• Tuning for energy efficiency
• Beyond static tuning: exploit dynamism in application characteristics
• Leverage system scenario based tuning
Systems Scenario based Methodology

Design-time

- Applications
- Platform

Identify system scenarios

Develop prediction mechanism

Develop switching mechanism

Develop calibration mechanism

Optimize scenario exploitation

System integration

Run-time

- Cost
- Time

Without scenarios

With scenarios

1. Exploitation and prediction
2. Switching
3. Calibration
Periscope Tuning Framework (PTF)

- Automatic application analysis & tuning
  - Tune performance and energy (statically)
  - Plug-in-based architecture
  - Evaluate alternatives online
  - Scalable and distributed framework
- Support variety of parallel paradigms
  - MPI, OpenMP, OpenCL, Parallel pattern
- AutoTune EU-FP7 project
Scalable Performance Measurement Infrastructure for Parallel Codes
Common instrumentation and measurement infrastructure

Score-P measurement infrastructure
Hardware counter (PAPI, rusage)

Application (MPI×OpenMP×CUDA)

Instrumentation wrapper

Vampir
Scalasca
TAU
Periscope

Event traces (OTF2)
Call-path profiles (CUBE4, TAU)

Online interface

PMPI
OPARI 2
CUDA
Compiler
PDT
User

MPI
POMP2
CUDA
Compiler
TAU
User
Tuning Plugin Interface

Search Space Exploration inside of Tuning Steps

Scenario execution
- Tuning actions
- Measurement requests
Tuning Plugins

• MPI parameters
  • Eager Limit, Buffer space, collective algorithms
  • Application restart or MPIT Tools Interface

• DVFS
  • Frequency tuning for energy delay product
  • Model-based prediction of frequency
  • Region level tuning

• Parallelism capping
  • Thread number tuning for energy delay product
  • Exhaustive and curve fitting based prediction
Dynamic Tuning with the READEX Tool Suite

• READEX extends the concept of tuning in Periscope

• Dynamic tuning
  • Instead of one optimal configuration, SWITCH between different best configurations.
  • Dynamic adaptation to changing program characteristics.
Scenario-Based Tuning

- **Design Time Analysis**
- **Tuning Model**
- **Runtime Tuning**

Periscope Tuning Framework (PTF)

READEX Runtime Library (RRL)
```c
int main(void) {

    // Initialize application
    Initialize experiment variables

    int num_iterations = 2;
    for (int iter = 1; iter <= num_iterations; iter++) {
        // Start phase region
        // Read PhaseCharct
        laplace3D();  // significant region
        residue = reduction(); // insignificant region
        fftw_execute();  // significant region
        // End phase region

        // Pre-processing:
        // Write matrices to disk for visualization of the application

        MPI_Finalize();
        return 0;
    }

    return 0;
}
```
Tuning plugin supporting

- Core and uncore frequencies, numthreads parameters, application tuning parameters
- Configurable search space via READEX Configuration File
- Several objective functions: energy, CPUenergy, EDP, EDP2, time
- Several search strategies: exhaustive, individual, random, genetic

Approach

1. Experiment with default configuration
2. Experiments for selected configurations
   - Configuration set for phase region
   - Energy and time measured for all runtime situations
3. Identification of static best for phase and rts specific best configurations
Pre-Computation of Tuning Model

Periscope Tuning Framework

- Search Algorithms
- Analysis
- Plugin Control
- Performance Database
- DTA Management
- RTS Management
- RTS Database
- DTA Process Management
- Scenario Identification
- READEX Tuning Plugin

Score-P

- Online Access Interface
- Substrate Plugin Interface
- Instrumentation
- Metric Plugin Interface
- Energy Measurements (HDEEM)

READEX Runtime Library

- Application Tuning Model
**READEX Runtime Library (RRL)**

- Runtime Application Tuning performed by the READEX Runtime Library.
- Tuning requests during Design Time Analysis are sent to RRL.
- A lightweight library
  - Dynamic switching between different configurations at runtime.
  - Implemented as a substrate plugin of Score-P.
- Developed by TUD and NTNU
Runtime Scenario Detection and Switching Decision during Production Run

- During Runtime Application Tuning
- Scenario classification

- Switching decision component
- Manipulation of tuning parameters
## BEM4I – Dynamic switching – Energy

http://bem4i.it4i.cz/

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>default settings</td>
<td>1467</td>
<td>1484</td>
<td>2733</td>
<td>1142</td>
<td>6872</td>
</tr>
<tr>
<td>static tuning only</td>
<td>1876</td>
<td>1926</td>
<td>1306</td>
<td>402</td>
<td>5537</td>
</tr>
<tr>
<td>dynamic tuning only</td>
<td>1348</td>
<td>1335</td>
<td>1150</td>
<td>268</td>
<td>4138</td>
</tr>
<tr>
<td>static + dynamic tuning</td>
<td>1343</td>
<td>1322</td>
<td>1161</td>
<td>265</td>
<td>4125</td>
</tr>
</tbody>
</table>

| static savings [%]            | -27.9%         | -29.8%         | 52.2%           | 64.8%        |
| dynamic savings [%]           | 8.4%           | 10.9%          | 57.5%           | 76.8%        |
| static + dynamic savings [%]  | 8.1%           | 10.0%          | 57.9%           | 76.5%        |

"static": {
  "FREQUENCY": "25",
  "NUM_THREADS": "12",
  "UNCORE_FREQUENCY": "22"
}

"assemble_k": {
  "FREQUENCY": "23",
  "NUM_THREADS": "24",
  "UNCORE_FREQUENCY": "16"
}

"assemble_v": {
  "FREQUENCY": "25",
  "NUM_THREADS": "24",
  "UNCORE_FREQUENCY": "14"
}

"gmres_solve": {
  "FREQUENCY": "17",
  "NUM_THREADS": "8",
  "UNCORE_FREQUENCY": "22"
}

"print_vtu": {
  "FREQUENCY": "25",
  "NUM_THREADS": "6",
  "UNCORE_FREQUENCY": "24"
}
**simpleFoam**
- strong scaling test
- Motorbike example
- optimum detected for every run
- Static: 11.7%
- Dynamic: 4.4%
- Total: 15.5%
- Dynamic savings increases with higher number of nodes

### Table: Scalability Tests – OpenFOAM – Analysis

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Default energy (J)</th>
<th>Default time (s)</th>
<th>Best static configuration</th>
<th>Static savings (J)</th>
<th>Dynamic savings (J)</th>
<th>Overall savings (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 node</td>
<td>37864.34</td>
<td>113.1</td>
<td>2.2 GHz UCF, 1.6 GHz CF</td>
<td>7344.46</td>
<td>105.85 of 30519.88 (19.40%)</td>
<td>7450.31 (19.68%)</td>
</tr>
<tr>
<td>2 nodes</td>
<td>37229.74</td>
<td>57.99</td>
<td>2.2 GHz UCF, 1.6 GHz CF</td>
<td>6175.44</td>
<td>118.24 of 31054.3 (16.59%)</td>
<td>6293.68 (16.9%)</td>
</tr>
<tr>
<td>4 nodes</td>
<td>38158.96</td>
<td>30.04</td>
<td>2.0 GHz UCF, 1.8 GHz CF</td>
<td>6223.96</td>
<td>107.28 of 31935.0 (16.31%)</td>
<td>6331.24 (16.59%)</td>
</tr>
<tr>
<td>8 nodes</td>
<td>41179.44</td>
<td>19.48</td>
<td>2.0 GHz UCF, 1.8 GHz CF</td>
<td>4493.2</td>
<td>945.12 of 36686.24 (10.91%)</td>
<td>5438.32 (13.21%)</td>
</tr>
<tr>
<td>16 nodes</td>
<td>57980.96</td>
<td>14.86</td>
<td>2.2 GHz UCF, 1.8 GHz CF</td>
<td>6780.96</td>
<td>2224.96 of 51200.0 (11.70%)</td>
<td>9005.92 (15.53%)</td>
</tr>
<tr>
<td>32 nodes</td>
<td>90374.4</td>
<td>16.498</td>
<td>2.2 GHz UCF, 1.4 GHz CF</td>
<td>22944.0</td>
<td>7113.6 of 67430.4 (25.39%)</td>
<td>30057.6 (33.26%)</td>
</tr>
</tbody>
</table>

*Does not scale anymore*
Inter-phase Dynamism

All-to-all Performance 2048 phases

PEPC Benchmark of the DEISA Benchmark Suite
Inter-Phase Analysis

• Variation of behavior among phases
• Group/cluster phases
• Select a best configuration for each cluster of phases

What do we need?
• Identifiers of phase characteristics (Phase Identifiers)
• Provided by application expert (??)
Inter-Phase Analysis – Approach

- Developed the `interphase_tuning` plugin
- 3 tuning steps:
  - **Analysis step:**
    - Random search strategy is used to create the search space
    - Don’t want to explore the whole tuning space
    - Cluster phases and find best configuration for each cluster
  - **Default step:**
    - Run the application for the default setting
  - **Verification step:**
    - Select the best configuration for each phase, as determined for its cluster.
    - Aggregate the savings over the phases
• 3 clusters identified
• Noise points marked in red
Cluster Prediction in RRL

• How to handle phase identifiers to predict clusters?
  • Call path of an rts now includes the cluster number

• Solution:
  • Add the cluster number as a user parameter
  • Add PAPI events to measure L3_TCM, TotalInstr and conditional branch instructions

```c
... 
SCOREP_OA_PHASE_BEGIN()
    SCOREP_USER_PARAMETER_INT64(cluster, predict_cluster())
... 
SCOREP_OA_PHASE_END()
```

• Predict the cluster of the upcoming phase
• If the cluster was mispredicted for the phase, correct it at the end of the phase
Evaluation of the readex_interphase plugin

- Performed on two applications: miniMD, INDEED
- Experiments conducted on the Taurus HPC system at the ZIH in Dresden
- Each node contains two 12-core Intel Xeon CPUs E5-2680 v3 (Intel Haswell family)
- Runs with a default CPU frequency of 2.5 GHz, uncore frequency of 3 GHz
- Energy measurements provided on Taurus via HDEEM measurement hardware
- Provides processor and blade energy measurements
miniMD

- Lightweight, parallel molecular dynamics simulation code
- Performs molecular dynamics simulation of a Lennard-Jones Embedded Atom Model (EAM) system
- Written in C++
- Provides input file to specify problem size, temperature, timesteps
- Evaluation of DTA:
  - Hybrid (MPI+OpenMP) AVX vectorized version
  - Problem size of 50 for the Lennard-Jones system.
• 6 clusters identified
• Noise points marked in red
• INDEED performs sheet metal forming simulations of tools with different geometries moving towards a stationary workpiece

• Contact between tool and workpiece causes:
  • Adaptive mesh refinement
  • Increase in number of finite element nodes
  • Increasing computational cost

• Time loop computes the solution to a system of equations until equilibrium is reached.

• OpenMP version evaluated
• 3 clusters identified
• Noise points marked in red
Energy Savings

<table>
<thead>
<tr>
<th>Application</th>
<th>Phase best for the rts’s (%)</th>
<th>rts best for the rts’s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>miniMD</td>
<td>14.51</td>
<td>0.03</td>
</tr>
<tr>
<td>INDEED</td>
<td>9.24</td>
<td>10.45</td>
</tr>
</tbody>
</table>

- miniMD records lower dynamic savings
  - miniMD has only two significant regions
  - One region is called only once during the entire application run
- Better static and dynamic savings for the rts’s of INDEED
  - INDEED has nine significant regions
  - Provides more potential for dynamism
Application Tuning Parameters (ATP)

- Exploit the dynamism in characteristics through the use of different code paths (e.g. preconditioners)
- Identify the control variables responsible for control flow switching.
  - provides APIs to annotate the source code
Evaluation of ATP: Espreso

- Finite Element (FEM) tools and domain decomposition based Finite Element Tearing and Interconnect (FETI) solver
  - Contains a projected conjugate gradient (PCG) solver.
  - Convergence can be improved by several preconditioners.
- Evaluated preconditioners on a structural mechanics problem with 23 million unknowns
  - On a single compute node with 24 MPI processes.

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th># iterations</th>
<th>1 iteration</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>172</td>
<td>125 ms</td>
<td>31.6 J</td>
</tr>
<tr>
<td>Weight function</td>
<td>100</td>
<td>130+2 ms</td>
<td>32.3+0.53 J</td>
</tr>
<tr>
<td>Lumped</td>
<td>45</td>
<td>130+10 ms</td>
<td>32.3+3.86 J</td>
</tr>
<tr>
<td>Light dirichlet</td>
<td>39</td>
<td>130+10 ms</td>
<td>32.3+3.74 J</td>
</tr>
<tr>
<td>Dirichlet</td>
<td>30</td>
<td>130+80 ms</td>
<td>32.3+20.62 J</td>
</tr>
</tbody>
</table>

15.9 s  4091.5 J
Configuration Variable Tuning

• Recently added \textit{readex\_configuration} tuning plugin
• Application Configuration Parameters with search space
• Replace selected value in input files and rerun the application
Input Identifiers

• What about different inputs?
• Annotation with *Input Identifiers* like problem size
• Apply Design Time Analysis and merge generated tuning models
• Selection at runtime based on the input identifiers
Summary

• Energy-efficiency tuning
  • Design Time Analysis – Tuning Model – Runtime Tuning

• Support for
  • Intra-phase dynamism
  • Inter-phase dynamism
  • Application Tuning Parameters
  • Application Configuration Parameters
  • Different input configurations

• Based on
  • Periscope Tuning Framework
  • Score-P Monitoring
Thank you! Questions?