READEX – Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing

EnA-HPC @ ISC’17

Robert Schöne – TUD
Project Motivation

Applications exhibit dynamic behaviour
- Changing resource requirements
- Computational characteristics
- Changing load on processors over time
Overview

READEX creates a tools-aided methodology for automatic tuning of parallel applications

- Dynamically adjust system parameters to actual resource requirements

Join technologies from embedded systems and HPC

- HPC: PTF, Score-P, and HDEEM
- ES: System scenario methodology
Background and Project Partners

- Grant agreement No 671657
- Officially started September 1\textsuperscript{st}, 2015

- Technische Universität Dresden/ZIH (Coordinator)
- Norwegian University of Science and Technology
- Technische Universität München
- IT4Innovations, VSB-Technical University of Ostrava
- NUI Galway, Irish Centre for High-End Computing
- Intel France
- Gesellschaft für numerische Simulation mbH
1. Static or dynamic tuning?
   - Uniform or changing demand of programs
   - Dynamic: sampling or instrumentation

2. Reducing power or runtime?
   - Power:
     - Frequencies
     - C-States
     - Speculative execution (e.g., prefetchers)
   - Runtime:
     - Frequencies (Turbo, various resources share single power budget)
     - Select optimal code paths
     - Optimize code

3. Tackling regions or balancing?
Energy Efficiency Tuning Types

<table>
<thead>
<tr>
<th>Process 0</th>
<th>Iteration N</th>
<th>Iteration N+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Clock down Synchronization</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

Time
Energy Efficiency Tuning Types

Process 0

Process 1

Process 2

Process 3

Iteration N

Iteration N+1

Time

Computation

Synchronization

Synchronization Marker
Energy Efficiency Tuning Types

<table>
<thead>
<tr>
<th>Process 0</th>
<th>Iteration N</th>
<th>Iteration N+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Synchronization</td>
<td>Slowed down Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>
Energy Efficiency Tuning Types

<table>
<thead>
<tr>
<th>Process 0</th>
<th>Iteration N</th>
<th>Iteration N+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>Synchronization</td>
<td>Synchronization Marker</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
</tr>
</tbody>
</table>

Time
Energy Efficiency Tuning with READEX

Region based tuning

READEX

Power based tuning ← Runtime based tuning

Balancing based tuning
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
  }

  // Post-processing:
  // Write noise matrices to disk for visualization
  // Terminate application

  MPI_Finalize();
  return 0;
}
Workflow

1. Instrument application
   Score-P provides different kinds of instrumentation

2. Detect dynamism
   Check whether runtime situations could benefit from tuning

3. Detect energy saving potential and configurations (DTA)
   Use tuning plugin and power measurement infrastructure to search for optimal configuration
   Create tuning model

4. Runtime application tuning (RAT)
   Apply tuning model, use optimal configuration
Instrumentation via Score-P

- HPC performance measurement infrastructure
  - Creates CUBEx profiles or OTF2 traces
  - Instrumentation and sampling
  - Supports most HPC programming paradigms
  - Mechanism for online usage of data – Periscope
  - Efficient implementation
  - Power measurement plugins (see talk by T. Ilsche)

- Re-use and extend existing infrastructure
  - Parse CUBEx profiles to find significant regions
  - Support tools to lower measurement overhead via filtering
  - Score-P Substrate Plugin Interface for alternative use-cases
  - READEX Runtime Library (RRL) to change parameters
Toggling Parameters

Periscope Tuning Framework

Score-P

Online Access Interface

Substrate Plugin Interface

Energy Measurements (HDEEM)

Application Tuning Model

READEX Runtime Library

Parameter Control Plugin

READEX Tuning Plugin
Toggling Parameters

- Hardware parameters
  - Core frequency, uncore frequency
Toggling Parameters

- Hardware parameters
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefetchers
Toggling Parameters

- **Hardware parameters**
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefellers
- **Runtime parameters**
  - Message Passing Interface, e.g., message size threshold
  - OpenMP parameters, e.g., loop scheduler, number of threads
Toggling Parameters

• Hardware parameters
  • Core frequency, uncore frequency
  • Clock modulation, Energy Performance Bias, prefetchers
• Runtime parameters
  • Message Passing Interface, e.g., message size threshold
  • OpenMP parameters, e.g., loop scheduler, number of threads
• Application tuning parameters
// C example
// register parameters at READEX
ATP_PARAM_DECLARE("PARAMETER1", ATP_PARAM_TYPE_RANGE, 1, "Domain1");
// declare set of possible values for the parameter
ATP_PARAM_ADD_VALUES("PARAMETER1", values_array, num_values, "Domain1")
// getting parameter setting from READEX, store in variable app_param
ATP_PARAM_GET("PARAMETER1",&app_param,"Domain1")
// ... usage of app_param, e.g., switch (app_param) {

Application Tuning Parameters (Work in Progress)

Application parameters example: different preconditioners in ESPRESO solver

- Full Dirichlet preconditioner is usually the preferred one (the best numerical properties)
- Depends on input dataset / problem that is solved
- All preconditioners have been evaluated with the optimal hardware parameter settings

<table>
<thead>
<tr>
<th>Preconditioner type</th>
<th>Number of iterations</th>
<th>Single iteration cost</th>
<th>Total solution cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time and energy</td>
<td>Time and energy</td>
</tr>
<tr>
<td>No preconditioner</td>
<td>172</td>
<td>130 + 0 ms</td>
<td>32.3 + 0.00 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>Full Dirichlet (default)</td>
<td>30</td>
<td>130 + 80 ms</td>
<td>32.3 + 20.6 J</td>
</tr>
</tbody>
</table>

11.3% energy savings against the default full Dirichlet preconditioners

Note: 130 ms and 32.3 J – is a baseline for single iteration cost without preconditioner
Design Time Analysis

![Diagram showing the interaction between Periscope Tuning Framework, Score-P, and READEX Runtime Library.]

Periscope Tuning Framework
- READEX Tuning Plugin

Score-P
- Online Access Interface
- Substrate Plugin Interface
- Energy Measurements (HDEEM)
- Application Tuning Model

READEX Runtime Library
- Parameter Control Plugin
Design Time Analysis

- Periscope Tuning Framework
- Pre-computation of dynamicity and significant regions
- Different objectives (e.g., runtime, energy, EDP)
- Different search strategies (complete, random, genetic)
- Uses RRL to switch parameters per phase
- Determine intra- and inter-phase dynamism
- Determine best configuration for significant regions
- Current work in progress:
  - Cluster regions in scenarios
  - Application tuning parameters
Force graph for scenarios

Vampir visualization of parameter changes
Runtime Tuning

• READEX Runtime Library
• Reads and applies Tuning Model
• Sets and resets configuration at runtime
• Current work in progress:
  • Application tuning parameters
  • Online calibration mechanism
  • Advanced switching decision making
Tuning Potential

ESPRESO: **12.3 %** + **9.1 %** = **20.3 %**

- Structural mechanics code
- Finite element + sparse FETI solver

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Best dynamic configuration</th>
<th>Dynamic savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler-AssembleStiffMat</td>
<td>14.32</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>20 threads, 2.0 GHz UCF, 2.5 GHz CF</td>
<td>2.51 J (0.34%)</td>
</tr>
<tr>
<td>Assembler-Assemble-B1</td>
<td>2.23</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>20 threads, 2.0 GHz UCF, 2.5 GHz CF</td>
<td>2.15 J (17.63%)</td>
</tr>
<tr>
<td>Cluster-CreateFEB-Fact0</td>
<td>0.17</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>20 threads, 2.0 GHz UCF, 2.5 GHz CF</td>
<td>1.80 J (20.73%)</td>
</tr>
<tr>
<td>Assembler-SaveResults</td>
<td>3.10</td>
<td>1588.81 J</td>
<td>1746.66 J</td>
<td>1.16 J (7.03%)</td>
</tr>
</tbody>
</table>

Total value for static tuning of significant regions: 733.73 + 114.30 + 8.71 + 158.81 + 278.39 + 278.39 + 113.87 + 14.23 + 688.97 + 325.69 + 99.93 + 74.70 + 641.88 + 1578.06 + 132.28 + 24.20 + 278.22 + 8.50 = 5124.66 J

Total savings for dynamic tuning of significant regions: 2.51 + 20.15 + 1.80 + 11.16 + 47.61 + 16.41 + 5.31 + 28.45 + 29.03 + 19.08 + 0.16 + 2.49 + 288.21 + 0.77 + 1.88 + 23.24 + 1.56 = 499.22 J of 5124.66 J (9.74 %)

Dynamic savings for application runtime: 499.22 J of 5124.66 J (9.74 %)

Total value after savings: 4994.33 J (79.72% of 6265.18 J)

ESPRESO:
- Structural mechanics code
- Finite element + sparse FETI solver
Alpha Prototype Results – Kripke

Master thread: 6, Master thread: 1, Master thread: 0, and 21 more, Values of Metric "PAPI_TOT_CYC" over Time

node taurusi282, Values of Metric "hdeem/BLADE" over Time
Alpha Prototype Results – Kripke

Values of Metric "PAPI_TOT_CYC" over Time

Values of Metric "hdeem/BLADE" over Time
Summary

• Region-based offline and online energy efficiency tuning

• Four steps:
  • Instrumentation
  • Preparation
  • Design Time Analysis
  • Runtime Tuning

• Alpha Prototype available

• Current work:
  • Application Tuning Parameters
  • Online Calibration
Discussion