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

EnA-HPC @ ISC’17

Robert Schöne – TUD
Energy Efficiency Tuning Types

1. Static or dynamic tuning?
   - Uniform or changing behavior 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 synchronization?
Energy Efficiency Tuning Types

![Diagram showing energy efficiency tuning types with processes and time iterations.]

- **Process 0**: Iteration N and Iteration N+1 with computation and synchronization.
- **Process 1**: Iteration N and Iteration N+1 with computation and synchronization.
- **Process 2**: Iteration N and Iteration N+1 with computation and synchronization.
- **Process 3**: Iteration N and Iteration N+1 with computation and synchronization.

**Key**:
- Green: Computation
- Red: Synchronization
- Blue: Slowed down Computation
Energy Efficiency Tuning with READEX

- Balancing based tuning
- Region based tuning
- Power based tuning
- Runtime based tuning

READEX
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
Overview

Co-design approach

• Manual tuning for energy efficiency as a baseline
• Automatic tuning for comparison

• Applications
  • PERMON and ESPRESO (FETI tools from IT4Innovations)
  • Indeed (GNS)
  • CORAL benchmark suite
  • ProxyApps
Project Partners

- Grant agreement No 671657
- Officially started September 1st, 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
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;
}
Terminology: Tuning Parameter and Intra-Phase Dynamism

```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
        // End phase region
    }

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

    MPI_Finalize();
    return 0;
}
```

Tuning Parameter FREQ=2 GHz
Tuning Parameter FREQ=1.5 GHz
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
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
  - Score-P Substrate Plugins
  - Use READEX Runtime Library (RRL) to change parameters
  - Support tools to lower measurement overhead via filtering
Toggling Parameters
Toggling Parameters

- **Hardware parameters**
  - Core frequency, uncore frequency
  - Clock modulation, Energy Performance Bias, prefachers

- **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 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
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
- Determine best configuration for significant regions
- Current work in progress:
  - Cluster regions in scenarios
  - Application tuning parameters
Tuning Model Visualization

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
ESPRESO: 12.3% + 9.1% = 20.3%

- Structural mechanics code
- Finite element + sparse FETI solver

**Tuning Potential**

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Value</th>
<th>Best dynamic configuration</th>
<th>Value</th>
<th>Dynamic savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler-</td>
<td>14.32</td>
<td>18 threads</td>
<td>733.73 J</td>
<td>20 threads</td>
<td>731.22 J</td>
<td>2.51 J</td>
</tr>
<tr>
<td>AssembleStiffMat</td>
<td></td>
<td>1.8 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>2.0 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>(0.34%)</td>
</tr>
<tr>
<td>Assembler-</td>
<td>2.23</td>
<td>20 threads</td>
<td>114.30 J</td>
<td>2 threads</td>
<td>94.15 J</td>
<td>20.15 J</td>
</tr>
<tr>
<td>Assemble-B1</td>
<td></td>
<td>1.8 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>2.2 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>(17.63%)</td>
</tr>
<tr>
<td>Cluster-</td>
<td>0.17</td>
<td>6 threads</td>
<td>8.71 J</td>
<td>1.6 GHz UCF, 2.5 GHz CF</td>
<td>6.90 J</td>
<td>1.80 J</td>
</tr>
<tr>
<td>CreateFstFactF0</td>
<td></td>
<td>1.8 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>1.2 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>(20.73%)</td>
</tr>
<tr>
<td>Assembler-</td>
<td>3.10</td>
<td>18 threads</td>
<td>158.81 J</td>
<td>2 threads</td>
<td>146.66 J</td>
<td>11.16 J</td>
</tr>
<tr>
<td>SaveResults</td>
<td></td>
<td>1.8 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>1.2 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>(7.03%)</td>
</tr>
<tr>
<td>Cluster-</td>
<td>0.17</td>
<td>8 threads</td>
<td>8.59 J</td>
<td>2 threads</td>
<td>7.03 J</td>
<td>1.56 J</td>
</tr>
<tr>
<td>CreateSa-SaReg</td>
<td></td>
<td>1.8 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>2.0 GHz UCF, 2.5 GHz CF</td>
<td></td>
<td>(18.15%)</td>
</tr>
</tbody>
</table>

**Total value for static tuning for significant regions**


**Total savings for dynamic tuning for significant regions**

2.51 J + 20.15 J + 1.80 J + 11.16 J + 47.01 J + 16.41 J + 5.31 J + 28.45 J + 29.03 J + 19.08 J + 0.16 J + 2.49 J + 288.21 J + 0.77 J + 1.88 J + 23.24 J + 1.56 J = 499.22 J of 5124.60 J (9.74%)

**Dynamic savings for application runtime**

499.22 J of 5438.35 J (9.09%)

**Total value after savings**

4994.33 J (79.72% of 6265.18 J)

**Review Meeting, 23.05.2017, Brussels – WP5**
Discussion

EnA-HPC - 22.06.2017
Backup
Tuning Parameters

READEX READEX EAB Meeting

Robert Schöne – TUD
WP1 Implementation
Hardware Parameters

TUD, INTEL (M01 – M12)

Existing Hardware parameters
• Dynamic Voltage and Frequency Scaling (DVFS)
• Uncore Frequency
• Energy Performance Bias (EPB)

Discarded parameters
• Dynamic Duty Cycle Management (DDCM, T-states)
// C example

// register parameters at READEX
ATP_PARAM_DECLARE("PARAMETER1", ATP_PARAM_TYPE_RANGE, 1, "Domain1");

// declare set of 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")
TUD, IT4I, INTEL (M01-12)

- **Message Passing Interface**
  - Short message size threshold
  - SMP-awareness
  - Relevant for MPI_AlltoAll and MPI_Reduce

- **OpenMP Threading**
  - Dynamic Concurrency Throttling (change number of threads)
  - Workload scheduling algorithm
  - Chunksize
Design Time Analysis

READEX READEX EAB Meeting

Robert Schöne – TUD
WP2 Implementation

Periscope Tuning Framework

Score-P

Online Access Interface

Substrate Plugin Interface

Energy Measurements (HDEEM)

Application Tuning Model

READEX Runtime Library

Parameter Control Plugin
Terminology: Region and Region Instance

```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
    }

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

    MPI_Finalize();
    return 0;
}
```

**Phase**

**Significant region**

**Runtime situation**

**Scenario**

Review Meeting, 23.05.2017, Brussels – WP2
Design Time Analysis

Task 2.3
- Instrumentation
  - Dynamism Detection
    - READEX Configuration File
      - scorep_autofilter
        - readex_dyn_detect

Task 2.2
- Design Time Analysis
  - PTF and RRL

Task 2.1
- Tuning Model

Task 2.4
- TM visualization
  - Vampir
Periscope Tuning Framework

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

Developed in the AutoTune EU-FP7 project
Pre-Computation of Configurations

Periscope Tuning Framework

- Search Algorithms
- Analysis
- Performance Database
- Plugin Control
- Experiments Engine

Score-P

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

READEX Runtime Library

- Application Tuning Model

RTS

- RTS Database
- RTS Management
- DTA Process Management
- DTA Management
- Scenario Identification

Experiments

Algorithms

Database

Analysis

Performance

Plugin Control

Experiments

Engine

Score-P

READEX

Runtime

Library

Review Meeting, 23.05.2017, Brussels – WP2
Runtime Tuning

READEX READEX EAB Meeting

Robert Schöne – TUD
WP3 Implementation
Task 3.2: Runtime Scenario Detection ... (2)

- During DTA
- Tuning request from PTF as JSON string
- Parsed by OA Event Receiver
- Stored by TMM
Task 3.2: Runtime Scenario Detection ...

- During RAT
- Identifier value detection
- Scenario classification
Task 3.4: Efficient Switching Decision making ... (2)

Switching decision component

Manipulation of tuning parameters
Task 3.3: Runtime Calibration Mechanism ...

Identifier value detection

Unseen rts

Calibrate

Configure

Update ATM
Integration

READEX READEX EAB Meeting

Robert Schöne – TUD
Initial Results

READEX READEX EAB Meeting

Robert Schöne – TUD
WP5 Implementation

Review Meeting, 23.05.2017, Brussels – WP5
WP5 Objectives

Focus of WP5 till M18

- Objective 1: State-of-the-art static tuning
- Objective 2: Manual dynamic tuning
- Objective 3: Automatic dynamic tuning with READEX
- Objective 4: Programming paradigm

Energy efficiency

Project progress

Default energy-efficiency of current and future exascale HPC applications
Task 5.1: Evaluating dynamism in HPC applications

Tools for Manual Evaluation

- **MERIC tool**
  - Based on manual annotation of significant regions
  - Search the space of tuning parameters to find optimal settings for each significant region
  - Support tool for energy measurements from HDEEM and RAPL
- **READEX Application Dynamism Analysis Report (RADAR) & RADAR generator**
  - Evaluates and reports the dynamic behaviour of the application

The dynamism observed in applications can be caused by the following factors:

- Floating point computations – variation in computational intensity (example - next slide)
- Memory read/write access patterns - variation in the sparsity of matrices in sparse linear algebra
- Inter-process communication patterns
- I/O operations performed during the application's execution
- Different inputs to regions in the application
Task 5.1: Evaluating dynamism in HPC applications

Goal: Investigate techniques to detect and evaluate dynamic behaviour in HPC applications

Hardware tuning parameters:
• core frequency (DVFS), uncore frequency (UFS)
• number of OpenMP threads

Example: Effect of Computational Intensity (CI)

<table>
<thead>
<tr>
<th>Two kernels with 1:1 workload ratio</th>
<th>Energy consumption</th>
<th>Energy savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default settings</td>
<td>2017J</td>
<td>-</td>
</tr>
<tr>
<td>Static optimal</td>
<td>1833J</td>
<td>179J 9%</td>
</tr>
<tr>
<td>Dynamic optimal</td>
<td>1612J</td>
<td>221J 12%</td>
</tr>
<tr>
<td>Total savings</td>
<td>-</td>
<td>400J 20%</td>
</tr>
</tbody>
</table>

Note: runtime of both kernels was equal for default settings
Task 5.1: Evaluating dynamism in HPC applications

Investigation of application parameters

Objective 1: State-of-the-art static tuning
- +10%

Objective 2: Manual dynamic tuning
- +20%

Objective 3: Automatic dynamic tuning with READEX
- +15%

Objective 4: Programming paradigm
- +22.5%

Project progress

Energy-efficiency

Default energy-efficiency of current and future exascale HPC applications
Approach and methodology for manual dynamism evaluation

1. **Identify significant regions** as the most time consuming (profiler – Alinea MAP)
2. **Manually annotate the significant regions** in the code – no compiler instrumentation overhead
3. **Apply tools developed in T5.1** to detect potential savings of an application
4. Using MERIC **run application for all possible configurations of tuning parameters** (parameters are set statically before each execution) and measure energy consumption
5. Using RADAR **find the best configuration for entire application** – static tuning potential
6. Using RADAR **find the best configuration for each significant region** – dynamic tuning potential
7. **Combine both static and dynamic savings** to define the potential for total energy savings
**OpenFOAM: 15.9% + 1.8% = 17.4% energy savings**

- Computational fluid dynamics
- Finite volume + multigrid solver

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Value</th>
<th>Best dynamic configuration</th>
<th>Value</th>
<th>Dynamic savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>init-createTime</td>
<td>0.03</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>3.35 J</td>
<td>1.4 GHz UCF, 1.4 GHz CF</td>
<td>2.64 J</td>
<td>0.71 J (21.06%)</td>
</tr>
<tr>
<td>init-createFields</td>
<td>4.28</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>506.91 J</td>
<td>2.4 GHz UCF, 2.0 GHz CF</td>
<td>474.80 J</td>
<td>32.11 J (6.33%)</td>
</tr>
<tr>
<td>init-createMesh</td>
<td>2.26</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>267.33 J</td>
<td>1.4 GHz UCF, 1.4 GHz CF</td>
<td>194.38 J</td>
<td>72.96 J (27.29%)</td>
</tr>
<tr>
<td>UEqn</td>
<td>40.71</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>4820.82 J</td>
<td>2.2 GHz UCF, 1.6 GHz CF</td>
<td>4810.03 J</td>
<td>10.79 J (0.22%)</td>
</tr>
<tr>
<td>pEqn</td>
<td>19.15</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>2268.19 J</td>
<td>1.4 GHz UCF, 1.4 GHz CF</td>
<td>2268.19 J</td>
<td>0.00 J (0.00%)</td>
</tr>
<tr>
<td>trans-And-Turbulence</td>
<td>25.70</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>3042.91 J</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>3042.91 J</td>
<td>0.00 J (0.00%)</td>
</tr>
<tr>
<td>write</td>
<td>7.88</td>
<td>2.0 GHz UCF, 1.6 GHz CF</td>
<td>932.59 J</td>
<td>1.2 GHz UCF, 1.4 GHz CF</td>
<td>841.62 J</td>
<td>90.97 J (9.75%)</td>
</tr>
</tbody>
</table>

Total value for static tuning for significant regions: 3.35 + 506.91 + 267.33 + 4820.82 + 2268.19 + 3042.91 + 932.59 = 11842.12 J

Total savings for dynamic tuning for significant regions: 0.71 + 32.11 + 72.96 + 10.80 + 0.00 + 0.00 + 90.97 = 207.54 J of 11842.12 J (1.75%)

Dynamic savings for application runtime: 207.54 J of 11966.36 J (1.73%)

Total value after savings: 11758.82 J (82.03% of 14231.30 J)

---

**static**  |  **dynamic**  |  **total**

**Task 5.2 Manually exploiting application dynamism**
Task 5.2 Manually exploiting application dynamism

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

- Structural mechanics code
- Finite element + sparse FETI solver

### Table: Region and Static/Dynamic Configuration

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Value</th>
<th>Best dynamic configuration</th>
<th>Value</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>733.73 J</td>
<td>20 threads, 2.0 GHz UCF, 2.5 GHz CF</td>
<td>731.22 J</td>
<td>2.51 J (0.34%)</td>
</tr>
<tr>
<td>Assembler-AssembleB1</td>
<td>2.23</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>114.30 J</td>
<td>2 threads, 2.2 GHz UCF, 2.5 GHz CF</td>
<td>94.15 J</td>
<td>20.15 J (17.63%)</td>
</tr>
<tr>
<td>Cluster-CreateF0-Fact0</td>
<td>0.17</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>8.71 J</td>
<td>6 threads, 1.6 GHz UCF, 2.5 GHz CF</td>
<td>6.90 J</td>
<td>1.80 J (20.73%)</td>
</tr>
<tr>
<td>Assembler-SaveResults</td>
<td>3.10</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>158.81 J</td>
<td>2 threads, 1.2 GHz UCF, 2.5 GHz CF</td>
<td>147.66 J</td>
<td>11.16 J (7.03%)</td>
</tr>
<tr>
<td>Cluster-CreateSa-SaReg</td>
<td>0.17</td>
<td>18 threads, 1.8 GHz UCF, 2.5 GHz CF</td>
<td>8.59 J</td>
<td>8 threads, 2.0 GHz UCF, 2.5 GHz CF</td>
<td>7.03 J</td>
<td>1.56 J (18.15%)</td>
</tr>
</tbody>
</table>

Total value for static tuning for significant regions:

733.73 J + 114.30 J + 8.71 J + 158.81 J + 278.39 J + 113.87 J + 14.23 J + 658.07 J + 325.69 J + 99.93 J + 74.70 J + 641.88 J + 1578.06 J + 1.28 J + 24.20 J + 278.22 J + 8.59 J = 5124.60 J

Total savings for dynamic tuning for significant regions:

2.51 J + 20.15 J + 1.18 J + 11.16 J + 47.01 J + 16.41 J + 5.31 J + 28.45 J + 20.03 J + 19.08 J + 0.16 J + 2.49 J + 288.21 J + 0.77 J + 1.88 J + 23.24 J + 1.56 J = 499.22 J of 5124.60 J (9.74%)

Dynamic savings for application runtime:

499.22 J of 5493.53 J (9.09%)

Total value after savings:

4994.33 J (79.72% of 6265.18 J)

---

ESPRESO:

**12.3%**

**+ 9.1%**

**= 20.3%**

---

**• Structural mechanics code**

**• Finite element + sparse FETI solver**

---

**Figure: Hybrid FETI Method**

- Green regions denote iterative solver (conjugate gradient (CG)) and provide opportunity for inter-phase dynamism.
- Orange regions are called just once per iteration and therefore are used only for intra-phase dynamism evaluation.
- White regions are ignored because there are other significant regions nested in them.
- Regions with names highlighted in bold are called only if Hybrid Total FETI method is used.

---

**Review Meeting, 23.05.2017, Brussels – WP5**

---

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

---

**Review Meeting, 23.05.2017, Brussels – WP5**

---

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

---

**Review Meeting, 23.05.2017, Brussels – WP5**
Indeed: \(17.6\% + 3.9\% = 20.9\%\)

- Structural mechanics code (Simulation of sheet metal forming)
- Finite Element solver

### Static vs Dynamic Results

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Value</th>
<th>Best dynamic configuration</th>
<th>Value</th>
<th>Dynamic savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm-matrix</td>
<td>17.07</td>
<td>16th, 2.1 GHz UCF, 2.5 GHz CF</td>
<td>119.19 J</td>
<td>24th, 2.1 GHz UCF, 2.3 GHz CF</td>
<td>113.18 J</td>
<td>6.00 J (5.04%)</td>
</tr>
<tr>
<td>output</td>
<td>32.95</td>
<td>16th, 2.1 GHz UCF, 2.5 GHz CF</td>
<td>230.05 J</td>
<td>24th, 1.2 GHz UCF, 2.5 GHz CF</td>
<td>226.61 J</td>
<td>3.43 J (1.49%)</td>
</tr>
<tr>
<td>recovery</td>
<td>9.16</td>
<td>16th, 2.1 GHz UCF, 2.5 GHz CF</td>
<td>63.95 J</td>
<td>24th, 2.1 GHz UCF, 2.3 GHz CF</td>
<td>61.88 J</td>
<td>2.06 J (3.23%)</td>
</tr>
<tr>
<td>solver</td>
<td>40.81</td>
<td>16th, 2.1 GHz UCF, 2.5 GHz CF</td>
<td>284.91 J</td>
<td>12th, 2.4 GHz UCF, 2.5 GHz CF</td>
<td>269.36 J</td>
<td>15.55 J (5.46%)</td>
</tr>
</tbody>
</table>

**Total value for static tuning for significant regions:**

\[119.19 + 230.05 + 63.95 + 284.91 = 608.09\ J\]

**Total savings for dynamic tuning for significant regions:**

\[6.00 + 3.43 + 2.06 + 15.55 = 27.05\ J\] of 608.09 J (3.87%)

---

**Review Meeting, 23.05.2017, Brussels – WP5**
Indeed: Specific look at solver region – intra-phase dynamism

Total sum of values from dynamic savings from all phases

Energy consumption: from 50.7kJ to 47.6kJ

6.3% energy savings
Evaluation of HPC codes ranging from basic kernels to very complex applications

**Key results**

- Highly optimized applications tend to provide higher static and lower dynamic savings
- Complex applications, such as ESPRESO, which contains variation on workload (not only compute) shows opportunity for dynamic tuning

<table>
<thead>
<tr>
<th>Application</th>
<th>Static savings [%]</th>
<th>Dynam. savings [%]</th>
<th>Total Savings [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel OpenMP I/O</td>
<td>56</td>
<td>—</td>
<td>56</td>
</tr>
<tr>
<td>Dense BLAS - DGEMV - without NUMA</td>
<td>5.6</td>
<td>—</td>
<td>5.6</td>
</tr>
<tr>
<td>Dense BLAS - DGEMM - without NUMA</td>
<td>10.4</td>
<td>—</td>
<td>10.4</td>
</tr>
<tr>
<td>Compute only kernel</td>
<td>12.8</td>
<td>—</td>
<td>12.8</td>
</tr>
<tr>
<td>Sparse BLAS Routines - without NUMA</td>
<td>3.1-12.3</td>
<td>—</td>
<td>3.1 - 12.3</td>
</tr>
<tr>
<td>Sparse BLAS Routines - with NUMA</td>
<td>4.2-66.2</td>
<td>—</td>
<td>4.2 - 66.2</td>
</tr>
<tr>
<td>ProxyApps 1 - AMG2013, configuration 1</td>
<td>6.53</td>
<td>2.89</td>
<td>9.23</td>
</tr>
<tr>
<td>ProxyApps 1 - AMG2013, configuration 2</td>
<td>25.66</td>
<td>2.80</td>
<td>27.47</td>
</tr>
<tr>
<td>ProxyApps 2 - Kripke, configuration 1</td>
<td>28.16</td>
<td>1.56</td>
<td>29.28</td>
</tr>
<tr>
<td>ProxyApps 2 - Kripke, configuration 2</td>
<td>12.63</td>
<td>7.04</td>
<td>18.78</td>
</tr>
<tr>
<td>ProxyApps 3 - LULESH, configuration 1</td>
<td>28.58</td>
<td>0.55</td>
<td>28.88</td>
</tr>
<tr>
<td>ProxyApps 3 - LULESH, configuration 2</td>
<td>25.81</td>
<td>1.23</td>
<td>26.72</td>
</tr>
<tr>
<td>ProxyApps 4 - MCB, configuration 1</td>
<td>4.13</td>
<td>1.42</td>
<td>5.51</td>
</tr>
<tr>
<td>ProxyApps 4 - MCB, configuration 2</td>
<td>3.40</td>
<td>4.18</td>
<td>7.44</td>
</tr>
<tr>
<td>ESPRESO - configuration 0</td>
<td>5.6</td>
<td>8.7</td>
<td>14.3</td>
</tr>
<tr>
<td>ESPRESO - configuration 1</td>
<td>12.3</td>
<td>9.1</td>
<td>21.4</td>
</tr>
<tr>
<td>ESPRESO - configuration 2</td>
<td>7.8</td>
<td>4.7</td>
<td>12.5</td>
</tr>
<tr>
<td>ESPRESO - configuration 3</td>
<td>7.8</td>
<td>5.4</td>
<td>13.1</td>
</tr>
<tr>
<td>OpenFOAM (Motorbike benchmark)</td>
<td>15.9</td>
<td>1.8</td>
<td>17.7</td>
</tr>
<tr>
<td>Indeed</td>
<td>17.6</td>
<td>to be evaluated</td>
<td>17.6</td>
</tr>
<tr>
<td>MiniMD</td>
<td>21.92</td>
<td>0.00</td>
<td>21.92</td>
</tr>
</tbody>
</table>
WP5: New applications

**BEM4I** - Boundary element method: $12.4\% + 11.7\% = 22.6\%$

- High CI for matrix assembling & Low CI for linear solver

### Overall application evaluation

<table>
<thead>
<tr>
<th></th>
<th>Default settings</th>
<th>Default values</th>
<th>Best static configuration</th>
<th>Static Savings</th>
<th>Dynamic Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy consumption [J]</td>
<td>24 threads, 3.0 GHz UCF, 2.5 GHz CF</td>
<td>5011.13 J</td>
<td>24 threads, 1.6 GHz UCF, 2.1 GHz CF</td>
<td>619.81 J (12.37%)</td>
<td>512.81 J of 4391.32 J (11.68%)</td>
</tr>
<tr>
<td>Runtime of function [s]</td>
<td>24 threads, 3.0 GHz UCF, 2.5 GHz CF</td>
<td>17.87 s</td>
<td>24 threads, 3.0 GHz UCF, 2.5 GHz CF</td>
<td>0.00 s (0.00%)</td>
<td>1.37 s of 17.87 s (7.64%)</td>
</tr>
<tr>
<td>Job info - hdeem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Region

<table>
<thead>
<tr>
<th>Region</th>
<th>% of 1 phase</th>
<th>Best static configuration</th>
<th>Value</th>
<th>Best dynamic configuration</th>
<th>Value</th>
<th>Dynamic savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmres.solve</td>
<td>16.82</td>
<td>24 threads, 1.6 GHz UCF, 2.1 GHz CF</td>
<td>737 J</td>
<td>12 threads, 2.2 GHz UCF, 1.3 GHz CF</td>
<td>326.87 J</td>
<td>410.15 J (55.65%)</td>
</tr>
<tr>
<td>print.vtu</td>
<td>0.51</td>
<td>24 threads, 1.6 GHz UCF, 2.1 GHz CF</td>
<td>22.37 J</td>
<td>12 threads, 1.2 GHz UCF, 2.3 GHz CF</td>
<td>16.28 J</td>
<td>6.09 J (27.24%)</td>
</tr>
<tr>
<td>assemble.K</td>
<td>40.81</td>
<td>24 threads, 1.6 GHz UCF, 2.1 GHz CF</td>
<td>1788 J</td>
<td>24 threads, 1.2 GHz UCF, 2.3 GHz CF</td>
<td>1747.57 J</td>
<td>40.59 J (2.27%)</td>
</tr>
<tr>
<td>assemble.V</td>
<td>41.85</td>
<td>24 threads, 1.6 GHz UCF, 2.1 GHz CF</td>
<td>1833 J</td>
<td>24 threads, 1.2 GHz UCF, 2.3 GHz CF</td>
<td>1777.74 J</td>
<td>55.98 J (3.05%)</td>
</tr>
</tbody>
</table>

**Dynamic savings for application runtime**

| Dynamic savings for application runtime | 512.81 J of 4391.32 J (11.68%) |
| Total value after savings              | 3878.51 J (77.40 % of 5011.13 J) |