Horizon 2020
European Union funding
for Research & Innovation

GA no. 671657

Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing

D5.3

Evaluation of the READEX Tool Suite using the READEX test-suite

Document type:	Report
Dissemination level:	Report
Work package:	WP5
Editor:	Lubomír Říha (IT4I-VSB)
Contributing partners:	IT4I-VSB, TUM, TUD, INTEL, ICHEC
Reviewer:	Michael Gerndt (TUM), Uldis Locans (INTEL)
Version:	1.0

Document history

Version	Date	Author/Editor	Description
0.1	27/01/17	Lubomír Říha, Jan Zapletal, Ondřej Vysocký (IT4I-VSB)	1st Draft
0.2	13/08/18	Lubomír Říha, Jan Zapletal, Ondřej Vysocký (IT4I-VSB) Robert Schone, Andreas Gocht (TUD), Michael Gerndt (TUM), Uldis Locans (INTEL)	2nd Draft
1.0	31/08/18	Lubomír Říha, Jan Zapletal, Martin Beseda, Ondřej Vysocký (IT4I-VSB) Robert Schone, Andreas Gocht (TUD), Venkatesh Kannan (ICHEC-NUIG), Kai Diethelm (GNS), Jan Christian Meyer, Per Gunnar Kjeldsberg (NTNU) Michael Gerndt (TUM), Uldis Locans (INTEL)	Final version

Contents

1 Introduction **4**

2 The READEX Test Suite **5**

2.1 Benchmark description 6

3 Test System Description **8**

4 Productivity Evaluation **10**

4.1 Manual Tuning Effort 10

4.2 READEX Tuning Effort 11

5 Hardware and runtime parameters tuning **13**

5.1 TU Dresden, Haswell Partition 13

5.2 IT4I Salomon 15

6 Application Parameters Tuning **17**

6.1 Effort Evaluation for Application Parameter Tuning 17

6.2 Test Applications 18

6.3 Summary of the Application Parameters Tuning 21

7 Collaboration with External Users **22**

7.1 CSC Finland - ELMER Application 23

7.2 PRACE - Alya Application 25

7.3 HLRS - FLEXI Application 26

7.4 AkerBP - OptEWE Application 27

1 Introduction

This deliverable presents the evaluation of the READEX Tool Suite based on the READEX Test Suite. This task is part of Work Package 5 and was handled in particular by following partners IT4I, TUD, NUIG, and GNS, in project months 19–36 of the project. We compared the READEX Tool Suite to a manual approach of tuning applications, which includes a step for static and dynamic manual tuning for hardware parameters. Furthermore, we evaluated application parameter tuning on a subset of applications. Our evaluation focuses on both performance and energy savings as well as productivity (effort required for manual tuning and tuning using the READEX Tool Suite).

To show that the READEX Tool Suite does not only fit a single system, we evaluated it on different parallel machines and hardware architectures. Furthermore, we tested the READEX Tool Suite at different sites to show that different software stacks can be used. We also show that the READEX Tool Suite also supports different compilers, by testing Intel and GNU compilers. For high precision power measurements, we used the HDEEM [2] measurement infrastructure at the TU Dresden Haswell partition. We used RAPL measurement infrastructure, which is part of all contemporary Intel processors for the other systems, i.e., the Salomon system at IT4I and the Broadwell partition at TU Dresden

2 The READEX Test Suite

As part of task 5.2, we defined the READEX test suite, whose repository is located at `git@acratus.ichec.ie:readex-apps.git`. The repository holds several benchmark and production applications, which are listed in Table 1.

application	**path**	**maintained by**
AMG2013	benchmark_apps/amg2013	IT4I
Blasbench	benchmark_apps/blasbench	IT4I
Kripke	benchmark_apps/kripke	IT4I
Lulesh	benchmark_apps/lulesh	TUM
NPB3.3	benchmark_apps/NPB3.3-MZ-MPI	TUD
BEM4I	production_apps/BEM4I	IT4I
ESPRESO	production_apps/ESPRESO/ espreso_readex_new	IT4I
INDEED	production_apps/Indeed_for_READEX	GNS
OpenFOAM	production_apps/OPENFOAM	IT4I

Table 1: List of applications in the readex-apps repository.

In addition to the source files, the directory of each application contains bash scripts to compile the application in several configurations. This includes:

1. the compilation of the uninstrumented (plain) version used as a reference and
2. the compilation for every tool in the READEX tool suite (scorep-autofilter, readex-dyn-detect, PTF, and RRL).

The compilation scripts have been prepared both for manual and automated tuning using the READEX tool suite. While we use the former approach for the evaluation of possible savings, we use the latter to evaluate the automated READEX tool suite in comparison to the manual effort. In addition to the compilation scripts, the repository contains scripts to launch the individual READEX tools and a brief `READEX_README.txt` help file that describes the usage. Please refer to Deliverable D5.2 [4] to find out how to use the READEX test suite and description of particular execution scripts.

During our collaboration with other HPC centers we have evaluated four additional complex HPC applications: ELMER, ALYA, FLEXI, and OptEWE. These are presented in Section 7.

application	HPC center, Country	maintained by
ELMER	CSC, Finland	IT4I
ALYA	BSC, Spain	ICHEC
FLEXI	HLRS, Germany	TUD
OptEWE	AkerBP, Norway	NTNU

Table 2: New applications identified during collaboration with European HPC centers.

2.1 Benchmark description

2.1.1 AMG2013

AMG2013 is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. It has been derived directly from the BoomerAMG solver in the hypre library, a large linear solver library that is being developed in the Center for Applied Scientific Computing (CASC) at LLNL. The driver provided in the benchmark can build various test problems. For our tests the Laplace type problem on an unstructured domain with various jumps and an anisotropy in one part is used.

2.1.2 Blasbench

Blasbench is a simple artificial benchmark designed to simulate different workloads including compute, memory, I/O, or communication bound regions. The users can define their own mixture of such regions to obtain a highly dynamic code.

2.1.3 Kripke

Kripke is a simple, scalable, 3D Sn deterministic particle transport code. Its primary purpose is to research how data layout, programming paradigms and architectures effect the implementation and performance of Sn transport. A main goal of Kripke is investigating how different data-layouts affects instruction, thread and task level parallelism, and what the implications are on overall solver performance.

2.1.4 Lulesh

Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is part of the ALE3D Software Infrastructure. According to the documentation, it is "one of five challenge problems in the DARPA UHPC program and has since become a widely studied proxy application in DOE co-design efforts for exascale." It is implemented in C++ and uses OpenMP and MPI for parallelization.

2.1.5 NPB3.3

The NAS Parallel Benchmarks are a set of micro benchmarks that mimic real applications used in HPC. In our test suite, we use the hybrid parallel benchmarks written in FORTRAN and parallelized with MPI and OpenMP. In particular the pseudo application BT-MZ-Block Tri-diagonal (Multi-Zone) solver from the benchmark suite is used. We used the READEX Domain-level Knowledge Specification Interface [1] for instrumentation.

2.1.6 BEM4I

BEM4I is a solver for partial differential equations based on the boundary element method and is under development at IT4Innovations. Contrary to finite element solvers, BEM4I produces dense matrices and due to the nature of boundary integral equations the assembly of system matrices is more or less compute bound. This is in contrast to the iterative solver used for the solution of the resulting system of linear equations which is usually memory bound due to matrix vector multiplications. BEM4I is thus a suitable library to be involved in the test apps repository.

2.1.7 ESPRESO

The ESPRESO library is a combination of Finite Element (FEM) and Boundary Element (BEM) tools and TFETI/HTFETI solvers. It supports FEM and BEM (uses BEM4I library) discretization for Advection-diffusion equation, Stokes flow and Structural mechanics. The ESPRESO solver is a parallel linear solver, which includes a highly efficient MPI communication layer designed for massively parallel machines with thousands of compute nodes. The parallelization inside a node is done using OpenMP.

2.1.8 INDEED

INDEED is a sheet metal forming simulation software that combines ease of use with high quality simulation. It is written in Fortran. Since it is a commercial tool, we do not provide the sources in the repository, but only the diff that has to be applied for manual instrumentation.

Note that, due to a change in the marketing strategy, the INDEED software system will in the future be known under the name OpenForm/Solv or, for short, OF/Solv.

2.1.9 OpenFOAM

OpenFOAM is an open source C++ toolbox for computational fluid dynamics (CFD). OpenFOAM does not have a generic solver applicable to all cases, but there is a long list of solvers each for specific class of problems e.g. compressible and incompressible flow, multiphase flow, combustion, particle-tracking flows heat transfer and many more.

3 Test System Description

In this section, we shortly describe the systems, which we used for the evaluation of the READEX Tool Suite. To demonstrate that READEX is capable of supporting different architectures and software stacks, we test it on the Intel Haswell and Intel Broadwell processors, as well as on different sites, i.e., the TU Dresden Top500 system **Taurus** and the IT4I Top500 system **Salomon**.

Taurus is a heterogeneous cluster listed in the Top500 list since its first installation at TU Dresden. It is regularly updated with new compute nodes and accelerators, and hosts various architectures. In previous projects, TU Dresden analysed the Haswell architecture in detail [3], and supported the development of a reliable power measurement infrastructure [2], which was available on the Haswell partition during the project life time and continues to be so. We used two different partitions for our tests, which host different processor architectures and software stacks. More information on taurus can be found online[1].

Salomon is a Top500 installation at IT4I. It hosts different architectures, including accelerator based systems. In this document, we use the 'grafton' partition, which is equipped with Intel Haswell based processors. Even though the processors are the same as in the Taurus Haswell partition, the software stack is completely different. More information on the Salomon test system can be found online[2].

In Table 3, we provide an overview on the used hardware and software stack. Whenever we describe energy savings in this document, we use measurements of the whole node. For the Taurus Haswell partition, this mean that we use the HDEEM node counter. On the other systems, we use the processor counters, add the package energy and the DRAM energy and add a static offset that was determined in a separate evaluation. Whenever we describe runtime measurements, we measure the runtime of the whole application. Our main test platform was the Taurus Haswell partition , using the Intel Compiler Suite.

[1] `https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus`
[2] https://docs.it4i.cz/salomon

Table 3: Comparison of test system used for evaluation.

Test System	Taurus (Haswell)	Taurus (Broadwell)	Salomon
Processor	Xeon E5-2680 v3	Xeon E5-2680 v4	Xeon E5-2680 v3
Nr. of cores	12	14	12
Frequency range (without turbo)	1.2-2.5 GHz	1.2-2.4 GHz	1.2-2.5 GHz
Uncore frequency range	1.2-3.0 GHz	1.2-2.7 GHz	1.2-3.0 GHz
DRAM	64 GB	64 GB	128 GB
Operating system	Bullx SCS4	Bullx SCS5	CentOS 6.9
Batch system	SLURM 16.05.7 Bull.1.1-20170512-1252	SLURM 17.02.11	PBS Pro 13.1.1
Linux kernel base	2.6.32	3.10.0	2.6.32
Linux kernel minor	696.30.1.el6.x86_64	693.21.1.el7.x86_64	696.20.1.el6.x86_64
Compiler suite (Intel)	17.0.2 20170213	18.0.1 20171018	18.0.1 20171018
MPI suite (Intel)	2017 Update 2 Build 20170125	2018 Update 1 Build 20171011	2018 Update 1 Build 20171011
Compiler suite (GCC)	6.3.0	7.3.0	7.1.0
MPI suite (GCC)	bullxmpi1.2.8.4	OpenMPI 3.1.0	OpenMPI/1.10.7
Power measurement infrastructure	Bull HDEEM	Intel RAPL + static offset	Intel RAPL + static offset

4 Productivity Evaluation

One of the objectives described in the project proposal targets evaluation of the tuning effort needed for READEX. To quantify the time needed to tune a software with READEX, we record the time spent with the instrumentation and tuning procedure.

4.1 Manual Tuning Effort

The first tasks for IT4I were dynamism analysis and manual evaluation of HPC applications. After initial work on the evaluation of the ESPRESO application, which is developed at IT4I and therefore well known to us, we realized that:

Pure manual tuning of complex applications, which contain tens of significant regions is significantly more time consuming than anticipated in the proposal.

To stick with the project plan, we have developed support routines, which make the manual approach feasible. While these helped us with the manual tuning evaluation, they cannot be compared with the extensive READEX tool suite for multiple reasons:

- They rely on manual instrumentation and do not provide the capabilities of Score-P.
- They do not support Runtime Calibration mechanisms.
- The number of supported systems is significantly lower since they support less interfaces for energy measurement and hardware parameter tuning.
- They do not scale well with threads due to OpenMP barriers in the code.

We can summarize the manual tuning effort requires following steps:

1. **Develop support routines:** These must be able to change CPU core and CPU uncore frequency, the number of OpenMP threads, and to measure energy consumption for each of the annotated regions on our test systems.
2. **Find significant regions:** To do so, we used a profiler to find significantly long regions in the application and annotate them.
3. **Run analysis:** for this report we have execute the applications for all combinations of Core Frequency (CF), Uncore Frequency (UCF) and number of threads (our support tools can only perform brute force search of the parameter space).
4. **Evaluate the measurements:** To do so, we had to find the best configuration for each region and write it into a configuration file
5. **Run the application:** Here, we used the support routines to perform the dynamic switching of the tuning parameters based on the previous measurements.

4.2 READEX Tuning Effort

The automatic instrumentation effort demonstrates how much time it takes to successfully apply the READEX Tool Suite to applications we have worked with. The steps that are considered for automatic instrumentation are as follows:

1. Compile application with Score-P
2. Run all steps of the READEX Tool Suite

Table 4 presents an effort needed to apply READEX approach on advanced and complex HPC applications. While some are part of the READEX Test Suite, others have been evaluated under collaboration with external users (ELMER from CSC, Finland: Alya from Barcelona Supercomputing Centre, Spain; and FLEXI from HLRS, Stuttgart).

Table 4: Effort spent in applying automatic tuning using the READEX Tool Suite for complex HPC applications.

Application	compile with Score-P	run analysis HW params.	total time
ESPRESO	1 D	2 H	**approx. 2 D**
BEM4I	1 D	18 H	**approx. 3 D**
OpenFOAM	7 D	2 H	**approx. 7 D**
INDEED	3 D	2 D	**approx. 5 D**
ELMER	6 D	6 H	**approx. 7 D**
ALYA	7 D	1 D	**approx. 8 D**
FLEXI	2 H	2 H	**approx. 4 H**

Note 1: D - day, H - hour

Table 4 shows that **applying the READEX tool suite to the test applications takes between two and eight days**. Most of the time is spent with compilation of the application with Score-P. This can be explained with the high complexity of the application's source codes.

As we stated in the beginning of this section, we skipped a pure manual approach for being too time consuming to finish Task 5.1 and 5.2 in time. Therefore, we had to implement support routines to find the manual tuning optima and to be able to compare READEX against these results. The development, which makes parameters accessible and enables energy measurements took a multiple of the 10 weeks that would relate to a 90 % lower effort as anticipated in the project proposal. This does not count the effort for analysing the runtimes and detect significant regions of the application, which is done automatically in READEX by the application of scorep_autofilter and readex_dyn_detect. Furthermore, the manual approach took way longer for the analysis of parameters, since it could not benefit from advanced search strategies. Even more, the manual approach was focused on a specific software and hardware stack, which means that it would have to be re-implemented for new architectures. For a domain developer, who is not an expert in processor hardware capabilities, energy measurement frameworks, and tuning strategies, the effort for a manual approach would certainly at least double over what we have seen

Based on this experience the READEX Tool Suite brings the key advantage to any application developer who is unaware of the concepts introduced by READEX and who want to reduce energy efficiency of his application and potentially even increase its performance.

5 Hardware and runtime parameters tuning

Results for most codes have been gathered using both Intel and GCC compilers. The exception to this is INDEED that has only been tested in combination with the Intel compiler because of its use of nonstandard language constructs that are not supported by GCC.

Tables 5 to 10 presents all the results and their respective columns mean:

- Application - name of the application,
- HW parameters - HW parameters that have been tuned for this application,
- Static tuning savings - energy and time savings achieved by static tuning only with respect to default settings,
- Manual dynamic tuning - energy and time savings achieved by manual dynamic tuning with respect to default settings,
- READEX tuning savings - energy and time savings achieved by READEX tuning (HW parameters only).

5.1 TU Dresden, Haswell Partition

Table 5: Hardware parameter tuning on TUD Taurus system - Haswell partition. Applications and READEX Tool Suite compiled with Intel compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	12.5% / −0.9%	N/A	7.0% / −14.0%
Blasbench	CF, UCF, threads	7.4% / −0.9%	15.3% / −18.1%	9.9% / −9.2%
Kripke	CF, UCF	11.5% / −28.3%	18.8% / − 18.7%	10.5% / −28.9%
Lulesh	CF, UCF, threads	17.6% / −8.9%	18.7% / −11.7%	18.2% / −25.7%
NPB3.3-BT-MZ	CF, UCF, threads	11% / −11.3%	N/A	10.8% / −12%
BEM4I	CF, UCF, threads	15.7% / −6.2%	34.1% / 10.9%	34.0% / 10.9%
INDEED	CF, UCF, threads	17.6% / −12.8%	19.5% / −14.2%	19.1% / −17.3%

Note: We were unable to obtain results for ESPRESO and OpenFOAM due to bug in Score-P when Intel compiler is used. Issue submitted.

Table 6: Hardware parameter tuning on TUD Taurus system - Haswell partition. Applications and READEX Tool Suite compiled with GCC compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	7.8% / −7.8%	N/A	3.5% / −17.5%
Blasbench	CF, UCF, threads	6.5% / −1.7%	14.3% / −11.4%	15.4% / −21.2%
Kripke	CF, UCF	11.5% / −28.3%	12.9% / -1.9%	8.7% / −14.0%
Lulesh	CF, UCF, threads	1.3% / −0.5%	7.2% / −8.5%	5.5% / −8.3%
NPB3.3-BT-MZ	CF, UCF, threads	9.8% / −23%	N/A	9.5% / −23%
BEM4I	CF, UCF, threads	8.2% / −1.7%	20.6% / 4.7%	1.1% / −8.2% [1]
ESPRESO	CF, UCF, threads	4.3% / −8.9%	8.2% / −10.1%	7.1% / −12.3%
OpenFOAM	CF, UCF	15.9% / −10.5%	20.1% / 11.5%	9.8% / −9.8%

[1] The outlier here is a result of the Score-P GNU compiler plugin, which does not allow for an instrumentation of header files. Unfortunately, BEM4I is based on using such files.

5.1.1 TU Dresden, Broadwell Partition

Table 7: Hardware parameter tuning on TUD Taurus system - Broadwell partition. Applications and READEX Tool Suite compiled with Intel compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	6.8% / −6.2%	10.3% / −8.2%	7.5% / −10.5%
Blasbench	CF, UCF, threads	5.6% / −2.5%	12.8% / −9.6%	12.0% / −19.0%
Kripke	CF, UCF	11.8% / −12.0%	18.0% / −10.0%	4.3% / −10.3%
Lulesh	CF, UCF, threads	6.7% / −9.0%	9.0% / −8.5%	10.0% / −9.2%
NPB3.3-BT-MZ	CF, UCF, threads	9.3% / −10.9%	N/A	8.9% / −11.3%
BEM4I	CF, UCF, threads	5.9% / −5.8%	26.0% / 3.4%	23.0% / −1.1%
INDEED	CF, UCF, threads	12.2% / −14.1%	14.6% / −16.2%	14.0% / −18.0%

Note: We were unable to obtain results for ESPRESO and OpenFOAM due to bug in Score-P when Intel compiler is used. Issue submitted.

Table 8: Hardware parameter tuning on TUD Taurus system - Broadwell partition. Applications and READEX Tool Suite compiled with GCC compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	5.5% / −1.1	7.6% / −9.7%	9.0% / −8.0%
Blasbench	CF, UCF, threads	7.4% / −5.6%	13.6% / −4.3%	14.0% / −4.0%
Kripke	CF, UCF	13.6% / −0.9%	17.8% / −10.1%	8.2% / −14.0%
Lulesh	CF, UCF, threads	1.6% / −1.0%	8.3% / −5.9%	6.2% / −8.1%
NPB3.3-BT-MZ	CF, UCF, threads	11.1% / −10.2%	N/A	10.4% / −10.8%
BEM4I	CF, UCF, threads	4.0% / −7.1%	19.8% / 6.5%	3.2% / 1.7%[1]
OpenFOAM	CF, UCF	17.2% / -10.1%	19.4% / −4.3%	7.5% / −7.6

[1] The outlier here is a result of the Score-P GNU compiler plugin, which does not allow for an instrumentation of header files. Unfortunately, BEM4I is based on using such files.

5.2 IT4I Salomon

Table 9: Hardware parameter tuning on IT4I Salomon system. Applications and READEX Tool Suite compiled with Intel compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	4.9% / −4.3%	N/A	3.4% / −23.2%
Blasbench	CF, UCF, threads	7.0% / −38.8%	13.5% / −20.5%	10.9% / −19.8%
Kripke	CF, UCF	8.9% / −8.3%	16.2% / −16.1%	10.3% / −22.2%
Lulesh	CF, UCF, threads	14.5% / −9.0%	N/A	7.3% / −20.6%
NPB3.3-BT-MZ	CF, UCF, threads	2.7% / −3.5%	N/A	3.1% / −2.8%
BEM4I	CF, UCF, threads	6.5% / −13.0%	29.8% / 4.8%	24.3% / 8.2%

Note: We were unable to obtain results for ESPRESO and OpenFOAM due to bug in Score-P when Intel compiler is used. Issue submitted.

Table 10: Hardware parameter tuning on IT4I Salomon system. Applications and READEX Tool Suite compiled with GCC compiler.

Application	HW parameters	Static tuning savings (node energy/ time)	Manual dynamic tuning savings (node energy/ time)	READEX tuning savings (node energy/ time)
AMG2013	CF, UCF, threads	2.7% / −5.5%	4.1% / −5.4%	1.8% / −9.3%
Blasbench	CF, UCF, threads	5.9% / −5.4%	9.9% / −24.4%	11.5% / −20.4%
Kripke	CF, UCF	10.3% / −2.0%	12.8% / −12.5%	4.8% / −15.4%
NPB3.3-BT-MZ	CF, UCF, threads	4% / −2.6%	N/A	4% / −2.5%
BEM4I	CF, UCF, threads	6.2% / −4.9%	13.5% / −2.1%	−0.3% / −0.2% [1]
ESPRESO	CF, UCF, threads	4.8% / −9.5%	9.4% / −8.4%	8.1 −7.0%
OpenFOAM	CF, UCF	4.2% / 2.1%	5.5% 9.3%	18.4% / −7.5%

[1] The outlier here is a result of the Score-P GNU compiler plugin, which does not allow for an instrumentation of header files. Unfortunately, BEM4I is based on using such files.

6 Application Parameters Tuning

During the development of the READEX tool suite and in collaboration with application designers, we learned that most of the application parameters are (1) static and (2) defined outside of the existing source code, i.e., in configuration files. To support this common pattern, we implemented a PTF plugin for tuning the application parameters by modifying such configuration files (using Application Configuration Parameters plugin for PTF) of the particular application. This is in line with the reviewers comment on application specific parameters, which have a higher saving potential, which arose during the first project review. Out of the selected applications in this section, ESPRESO uses the ATP library directly, the other applications use Application Configuration Parameters. ESPRESO can also use by this approach to increase the number of tuning knobs.

In case of application parameter tuning we do not perform the hardware parameter tuning and default hardware settings are used for all tests.

6.1 Effort Evaluation for Application Parameter Tuning

As mentioned in the previous section there are two approaches: (1) using the ATP library and (2) using Application Configuration Parameters (ACP). The integration of the ATP library requires developer knowledge of the application and therefore, IT4I implemented this support into the ESPRESO library. The effort greatly depends on the parameter and how its runtime tuning affects entire application workflow. For instance, for ESPRESO we have experienced following:

- Runtime tuning of FETI METHOD, PRECONDITIONERS, ITERATIVE SOLVERS, HFETI type, SCALING, BO_TYPE was relatively easy. A developer **needed approximately 1 hour per parameter** from the list.
- Runtime tuning of **domain decomposition** was quite difficult. A developer **needed 5 days** to implement the support for this parameter, since ESPRESO performs domain decomposition only during startup. For the READEX project, we developed enhanced ESPRESO to redo the decomposition after each time-step of a transient simulation.

Alternatively, static tuning of application configuration parameters is very simple and straight forward. Based on our results it is one of the most valuable plugins in terms of runtime and energy savings. However, since this is performed using modification of configuration files, one can also do this manually. The effort comparison between manual and READEX approach can described as follows.

- Both approaches start with reading the user manual to find all parameters that the user wants to test and identify all potential values of these parameters.

- Manual approach
  - Manually prepare set of configuration files for all configuration the user wants to test
    - For a small number of parameters this can be done purely manually by copying and editing configuration files.
    - For a high number of parameters, the researcher must write a script - this takes **approximately 3 hours** (tested on ESPRESO with 9 different parameters)
  - Run application for each configuration file and manually measure energy consumption (for instance from job scheduler or other technique available on cluster, even runtime in this case is reasonable indicator)
    - For a small number of jobs, this can be done purely manually
    - For a large number of jobs, developers can modify the script from the previous step to run the application - this takes **approximately 1 hour**
  - Manually analyze finished jobs and find one with optimal energy consumption
    - For a small number of jobs, can be done purely manually
    - For a large number of jobs, developers can modify the script from previous step to write configuration ID and energy consumption into single file - this takes **approximately 2 hours**
- With READEX using ACP, the effort is assembled as follows:
  - Prepare a single configuration file and for each parameter: annotate it with all values that should be tested - this takes **approximately 0.5 hour** (tested on all applications)
  - Run the READEX Tool Suite to find the optimal configuration and get a final configuration file

Following either of these approaches will provide the same set of results because identical configurations are executed.

Note: The manual approach as described will use brute-force search of the parameter space which for high number of parameters becomes impossible to search. READEX can use different advanced algorithms implemented in PTF which significantly reduce the time needed to search the parameter space.

Note: The *reasonable settings* are settings prepared by experienced user that uses the application for its work.

6.2 Test Applications

In this section, we shortly describe the tuned parameters for each application. We furthermore present the detailed results.

6.2.1 ESPRESO

The **ESPRESO library** is developed by one of the partners (IT4I) in the READEX project. It was updated during the project to support more advanced dynamic application parameter tuning using ATP library. However, it also supports static application configuration parameters tuning.

The key features for application parameter tuning are:

- List of application parameters that were evaluated:
  - FETI METHOD (2 options)
  - PRECONDITIONER (5 options)
  - ITERATIVE SOLVER TYPE (2 options)
  - HFETI type (2 options)
  - NON-UNIFORM PARTS (6 options)
  - REDUNDANT LAGRANGE (2 options)
  - SCALING (2 options)
  - B0_TYPE (2 options)
  - ADAPTIVE PRECISION (2 options)
  - DOMAIN DECOMPOSITION (10 options)
- total number of possible combinations: 3840
- support for static application configuration parameter tuning using config. file: yes
- support for dynamic application parameter tuning: yes

Savings:

- the worst case scenario: runtime **1320** seconds, energy consumption **230 kJ**
- the best case scenario: runtime **189** seconds, energy consumption **32.5 kJ**
- savings between the worst and the best case: **86%**
- savings between default and the best case: **N/A** (see notes bellow)
- savings between reasonable settings and best case: **50-66%**

Notes: There is no default configuration in ESPRESO. Based on the problem that is being solved, a user has to setup the FETI solver in ESPRESO based on his knowledge of the problem he wants to solve.

6.2.2 OpenFOAM

OpenFOAM tool is controlled by a text configuration file. Therefore, only application configuration parameters can be tuned.

The key features for application parameter tuning are:

- List of application parameters that were evaluated:
  - PHYSICAL SOLVER PROPERTY (2 options)
  - LINEAR SOLVER (6 options)
- total number of possible combinations: 12
- support for static application configuration parameter tuning using config. file: yes
- support for dynamic application parameter tuning: no

Savings:

- the worst case scenario: runtime **60.4 s** seconds, energy consumption **77.5 kJ**
- default scenario: runtime **50.1** seconds, energy consumption **63.8.7 kJ**
- the best case scenario: runtime **46.1** seconds, energy consumption **58.7 kJ**
- savings between the worst and the best case: **24%**
- savings between default and the best case: **8%**

Notes: For this test, we used the MotorBike benchmark that is part of the OpenFOAM download. Therefore, we used the default settings. These settings are hand tuned by the OpenFOAM developers with high level of knowledge of the OpenFOAM behaviour. Even in this case, we have found setting that provide 8% savings.

6.2.3 INDEED

The **INDEED tool** is developed by GNS and it is controlled by a text configuration file. Therefore, we could only tune for application configuration parameters.

The key features for application parameter tuning are:

- List of application parameters that were evaluated:
  - SOLVER (2 options)
  - PCG METHOD (2 options)
  - STIFFNESS MATRIX ACCESS (3 options)

- total number of possible combinations: 12
- support for static application configuration parameter tuning using config. file: yes
- support for dynamic application parameter tuning: no

Savings:

- the worst case scenario: runtime **114.4** seconds, energy consumption **30.9 kJ**
- the best case scenario: runtime **74.8** seconds, energy consumption **20.1 kJ**
- savings between the worst and the best case: **35.1%**
- savings between reasonable settings and best case: **24.5%**.

Notes: The optimal settings require the linking of INDEED against a special solver for linear systems. This solver is part of a proprietary library from an external vendor and hence not necessarily available on all platforms. Our definition of *reasonable settings* assumes that this library is not available.

6.3 Summary of the Application Parameters Tuning

Table 11 summarizes the savings of all evaluated applications. INDEED evaluation was done on Taurus HSW partition with Intel compiler. All remaining applications have been tested on Salomon machine with GGC compiler.

Table 11: Summary table of the application parameter tuning with energy savings. The details for each application are described in the following subsections of this chapter.

Application	# of parameters tested / total # of options	energy savings compared to worst setting	energy savings compared to default/reasonable settings
ESPRESO	9/3840	86%	50-66%
ELMER	1/40	97%	50-75%
OpenFOAM	2/12	24%	8% [1]
INDEED	3/12	35%	25%

[1] For OpenFOAM we have used the official MotorBike benchmark and therefore we have the default settings and these are used in this case instead of reasonable settings. Because the benchmark is prepared by developers the default settings are near optimal ones.

7 Collaboration with External Users

In addition to the tests performed using the READEX Test Suite, we also collaborated with external users and have supported them with applying our tool suite on their applications. We use this kind of collaboration to demonstrate the potential of the READEX software for general applications. In parallel to that, we discussed with the external users, which interfaces for energy measurement and hardware parameter access are present on their system. We used this information to extend our tool suite to support more interfaces then we anticipated in the proposal of the project. The results of these extension will be used in further experiments on HPC system of the external sites.

7.1 CSC Finland - ELMER Application

In this section we present the results of applying the READEX Tool Suite on a real application from CSC Finland HPC center named as ELMER.

7.1.1 ELMER

ELMER is an open source multiphysical simulation software mainly developed by CSC - IT Center for Science (CSC). Elmer development was started 1995 in collaboration with Finnish Universities, research institutes and industry. After it's open source publication in 2005, the use and development of Elmer has become international.

Elmer includes physical models of fluid dynamics, structural mechanics, electromagnetics, heat transfer and acoustics, for example. These are described by partial differential equations which Elmer solves by the Finite Element Method (FEM).

7.1.2 Hardware Parameter Tuning

Table 12: Hardware parameter tuning on TUD Taurus system - Haswell partition. Applications and READEX Tool Suite compiled with GCC compiler.

Application	HW parameters	READEX tuning savings (node energy/time)
ELMER (4 nodes)	CF, UCF	7.8% / -26.5%

7.1.3 Application Parameter Tuning

ELMER is controlled by a text configuration file. Therefore application configuration parameters can be tuned.

The key features for application parameter tuning are:

- List of application parameters that were evaluated:
  - LINEAR SOLVERS (40 options) + each linear solver has its own set of parameters
- total number of possible combinations: 40 linear solvers
- support for static application configuration parameter tuning using config. file: yes
- support for dynamic application parameter tuning: no

Savings:

- the worst case scenario: runtime **919** seconds, energy consumption **1522 kJ** 5 nodes

- the best case scenario: runtime **24** seconds, energy consumption **35 kJ** - 5 nodes
- savings between the worst and the best case: **43x / 97%**
- savings between default and the best case: **N/A** (see notes bellow)
- savings between reasonable settings and best case: **2-4x / 50-75%**

In ELMER there is no default configuration. Based on the problem that is being solved, the user has to pick and setup appropriate linear solver based on his knowledge of the problem he wants to solve with ELMER. We have created 40 configuration files (one per linear solver) with the help of ELMER developers and tested these configurations. Please note that linear solver configuration is included in the problem configuration file and can be tuned using ACP.

7.2 PRACE - Alya Application

This section is based on an ongoing collaborative work between the READEX project and EoCoE (Energy oriented Centre of Excellence) as a part of PRACE 5IP Task 7.2. We present the currently available results of applying the READEX Tool Suite on a production-scale application from Barcelona Supercomputing Centre (BSC) called Alya.

Alya is a simulation code for high performance computational mechanics. Alya solves coupled multiphysics problems using high performance computing techniques for distributed and shared memory supercomputers, together with vectorization and optimization at the node level.

Presently, READEX has been appled on Alya to tune hardware (processor core and uncore frequencies) and system software (number of active OpenMP threads) parameters. As a part of the ongoing PRACE task, we are in the process of deploying the READEX tool suite on the MareNostrum4 supercomputer at the Barcelona Supercomputing Centre (BSC) and will continue beyond the READEX project to include application parameter tuning using the ATP library.

The following subsections present the efforts and results for tuning Alya availble when preparing this deliverable.

7.2.1 Hardware and System Software Parameter Tuning

Application	Parameters tuned	READEX tuning savings (node energy/time)
Alya	CF, UCF, threads	9.8% / -13.4%

Table 13: Hardware parameter tuning on TUD Taurus system - Haswell partition. Applications and READEX Tool Suite compiled with Intel compiler.

7.3 HLRS - FLEXI Application

In this section we present the results of applying the READEX Tool Suite on a real application from High Performance Computing Centre Stuutgart (HLRS) named as FLEXI.

7.3.1 FLEXI

FLEXI is an open source framework that offers a complete CFD solution. Flexi is developed by the team of the Numerics Research Group hosted at the Institute of Aero- and Gasdynamics at the University of Stuttgart. It is used in READEX as part of the collaboration with High Performance Computing Center Stuttgart(HLRS).

FLEXI is a high-order numerical framework for solving PDEs, with a focus on Computational Fluid Dynamics. FLEXI is based on the Discontinuous Galerkin Spectral Element Method (DGSEM), which allows for high-order of accuracy and fully unstructured hexahedral meshes. it consists of the high-order mesh generator and preprocessor HOPR, the solver Flexi and a converter to the Paraview format for visualization and postprocessing. The solver is parallelized very efficiently and scales up to hundreds of thousand cores.

7.3.2 Hardware Parameter Tuning

The FLEXI experiments are performed on the Haswell partition with HDEEM energy measurement system on TUD Taurus cluster using the Intel compiler suite. The hardware parameters used to tune the application are CPU core frequency.

The SLURM accounting tool *sacct* is used to gather the energy values gathered with HDEEM and the elapsed time. Due to the structure of the batch systems, the runtimes can vary for overhead reasons.

Application	Parameters tuned	READEX tuning savings (node energy/time)
Flexi (2 nodes)	CFS	11.0 %/-29 %
Flexi (4 nodes)	CFS	11.7 %/-17.3 %

Table 14: Hardware parameter tuning on TUD Taurus system - Haswell partition. Applications and READEX Tool Suite compiled with Intel compiler.

Table 14 shows the results of FLEXI runs. Here, we can see a saving in energy consumption of 11% with an added execution time overhead of 29%.

When using the same tuning model and scaling up to 4 nodes[4] shows almost the same amount of savings in energy. The energy savings in this case is 11.72% (183.28kJ in comparison to 207.63kJ) with a runtime overhead of 21.5%.

[4]We could not measure higher scale-ups due to a bug in hopr

7.4 AkerBP - OptEWE Application

In this section we present the results of applying the READEX Tool Suite to an application from the oil exploration company Aker BP ASA called OptEWE.

7.4.1 OptEWE

OptEWE models the computationally expensive step of an imaging process which recovers 3D images of the Earth's subsurface from recordings of seismic waves. OptEWE solves the elastodynamic wave equation by an explicit finite difference method on a Cartesian 3D volume.

The workflow it is integrated in analyzes a numbers of seismic sources independently at the inter-node level, while the computation at the intra-node level utilizes OpenMP threads. The time integration step utilizes a combination of 25 numerical kernels which exhibit variable arithmetic intensity. READEX has been applied to tune hardware parameters in accord with these variations.

7.4.2 Hardware and System Software Parameter Tuning

Application	Parameters tuned	READEX tuning savings (node energy/time)
OptEWE	CF, UCF	9.7% / -7.0%

Table 15: Hardware parameter tuning on TUD Taurus system - Haswell partition. Application and READEX Tool Suite compiled with GNU compiler.

Table 15 shows energy savings and run time impact obtained by comparing tuning results to a baseline of running the application with all hardware parameters at maximal settings. The time integration step of the application contains a number of compute-bound kernels interspersed with some that exhibit more memory-intensive behavior. The READEX tool chain identified that this stage can conserve total energy by maximizing core frequencies and minimizing uncore frequencies, at the expense of increases in application run time.

References

[1] Michael Gerndt, Madhura Kumaraswamy, and Zakaria Bendifalla. D4.5: Final description of the READEX programming paradigm. Technical report, TUM, Intel, 2018.

[2] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W.E. Nagel, M. Simon, and Y. Georgiou. HDEEM: High Definition Energy Efficiency Monitoring. In *Energy Efficient Supercomputing Workshop (E2SC)*, Nov 2014. DOI: 10.1109/E2SC.2014.13.

[3] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An energy efficiency feature survey of the Intel Haswell processor. In *Parallel and Distributed Processing Symposium Workshop*, May 2015. DOI: 10.1109/E2SC.2014.13.

[4] Lubomir Riha, Jan Zapletal, Ondřej Vysocký, and Vojtěch Nikl. D5.2 extended readex test-suite with manually tuned applications. Technical report, IT4I-VSB, 2018.