Computing
DOI 10.1007/s00607-016-0532-7

The READEX formalism for automatic tuning for energy efficiency

Joseph Schuchart[1] · **Michael Gerndt**[2] · **Per Gunnar Kjeldsberg**[3] · **Michael Lysaght**[4] · **David Horák**[5] · **Lubomír Říha**[5] · **Andreas Gocht**[1] · **Mohammed Sourouri**[3] · **Madhura Kumaraswamy**[2] · **Anamika Chowdhury**[2] · **Magnus Jahre**[3] · **Kai Diethelm**[6] · **Othman Bouizi**[7] · **Umbreen Sabir Mian**[1] · **Jakub Kružík**[5] · **Radim Sojka**[5] · **Martin Beseda**[5] · **Venkatesh Kannan**[4] · **Zakaria Bendifallah**[7] · **Daniel Hackenberg**[1] · **Wolfgang E. Nagel**[1]

Received: 14 July 2016 / Accepted: 14 December 2016

Abstract Energy efficiency is an important aspect of future exascale systems, mainly due to rising energy cost. Although High performance computing (HPC) applications are compute centric, they still exhibit varying computational characteristics in different regions of the program, such as compute-, memory-, and I/O-bound code regions. Some of today's clusters already offer mechanisms to adjust the system to the resource requirements of an application, e.g., by controlling the CPU frequency. However, manually tuning for improved energy efficiency is a tedious and painstaking task that is often neglected by application developers. The European Union's Horizon 2020 project READEX (*Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing*) aims at developing a tools-aided approach for improved energy efficiency of current and future HPC applications. To reach this goal, the READEX project combines technologies from two ends of the compute spectrum, embedded systems and HPC, constituting a split design-time/runtime methodology. From the HPC domain, the Periscope Tuning Framework (PTF) is extended to perform

✉ Joseph Schuchart
joseph.schuchart@tu-dresden.de

1 Center for Information Services and High Performance Computing, Technische Universität Dresden, Dresden, Germany

2 Chair of Computer Architectures, Technische Universität München, Garching, Germany

3 Department of Electronics and Telecommunications, Norwegian University of Science and Technology, Trondheim, Norway

4 Irish Center for High-End Computing, Galway, Ireland

5 IT4Innovations, VŠB-Technical University of Ostrava, Ostrava, Czech Republic

6 Gesellschaft für numerische Simulation, Braunschweig, Germany

7 Intel ExaScale Labs, Paris, France

dynamic auto-tuning of fine-grained application regions using the systems scenario methodology, which was originally developed for improving the energy efficiency in embedded systems. This paper introduces the concepts of the READEX project, its envisioned implementation, and preliminary results that demonstrate the feasibility of this approach.

Keywords Automatic tuning · Energy efficiency · Dynamic behaviour · Dynamic tuning · Parallel computing

Mathematics Subject Classification 68M14 · 68M20

1 Introduction

The next milestone in supercomputing is the development of Exascale systems. However, challenges related to power and energy consumption are currently holding the construction of such systems back [1], consequently making energy efficiency an important research topic.

Nowadays, modern processing architectures provide features to control certain aspects of the hardware. Such features include the possibility to adjust the system to the actual resource requirements of an application to improve energy efficiency. However, hand-tuning such parameters is considered to be a tedious task, particularly for domain experts as it requires considerable understanding of the underlying hardware, and is thus often neglected. The task becomes even more challenging in applications with varying characteristics that change during execution. We refer to such a behaviour as *application dynamism*. Examples of characteristics may involve compute intensity, workload granularity (load balancing), and parallel efficiency. Applications exhibiting such dynamic behaviour are the main target for our methodology.

The READEX project aims at creating a tools-aided methodology for dynamic auto-tuning of high performance computing (HPC) applications for increased energy efficiency compared to the default system configuration. The methodology is divided into two parts: *design time analysis (DTA)* and *runtime application tuning (RAT)*. The role of the DTA is to perform a detailed analysis of the application to unveil application dynamism. Once dynamism has been identified, care is taken to find the best tuning configuration for the individual code region of interest. Typical examples of configurations may include the number of OpenMP threads and the CPU clock frequency. Every execution of a code region is called a *runtime situation (rts)*. At the end of the DTA, a tuning model is created, which contains a description of the best found configurations. In order to reduce overhead, similar or identical rts's are merged into *scenarios*, a technique for dynamic tuning found in the embedded systems domain [2,3]. During production runs (RAT), the tuning model is used by a lightweight runtime library called the *READEX Runtime Library (RRL)*. The role of the RRL is to perform adjustments according to the description found in the tuning model. The quality of the tuning model is evaluated at runtime by the RRL. A calibration mechanism can make adjustments to the tuning model to further refine configurations at runtime.

The remainder of this article is organised as follows: Sect. 2 surveys related work followed by a description of the target environment in Sect. 3. The READEX methodology and its definitions are presented in Sect. 4. A description of the tool suite is provided in Sect. 5 and the proposed implementation of the tool suite is discussed in Sect. 6. The results of first experiments applying dynamic tuning to a target application are presented in Sect. 7.

2 Related work

We divide auto-tuning into two areas: static and dynamic tuning. The former describes the adaptation of the application or the underlying system before the application is executed whereas the latter refers to adaptation at application runtime. Tools for both types of auto-tuning exist as described below.

César et al. [4] created a dynamic tuning tool for automatic scheduling of workload on different nodes. Their proposed master/worker framework solved the problem of taking optimal tuning decisions for different nodes during the program runtime. This approach is scalable for embarrassingly parallel jobs, as the master does not have to synchronise the workers for this kind of job. Moreover, this approach can lead to a significant reduction of the execution time. Although it might be possible to save energy by reducing execution time, the authors do not explicitly focus on energy reduction.

The work of Tiwari et al. [5] targets static tuning. The authors have created a framework that statically evaluates different "computation kernels" and compiler optimisation flags to find an optimum in terms of execution time. Their work is based on Active Harmony [6], which is a well known framework for static and dynamic tuning. The application can define a search space and retrieve new configurations to test from a central tuning server. Contrary to the READEX approach, it is based on standard optimisation algorithms and not on expert knowledge about the tuning aspect.

An approach taken by several projects is to overcome the problem of non-optimal programs by introducing new prototype languages or programming models, e.g., the ENCORE [7] prototype language where the OmpSs programming model [8] is used to save energy by reducing the compute time.

The PEPPHER project [9] implements a framework that compiles multiple variants of the code for different types of architectures such as CPU's and GPU's. During execution, a runtime system decides which code path to execute. Although the authors seem to have energy efficiency as a possible objective in mind, they choose to focus on execution time as the main objective.

The ANTAREX project [10] takes a similar approach. Using a Domain Specific Language (DSL) code can be distributed between multi-core CPUs and accelerators. An extra compilation step is introduced to translate the DSL into the intended programming language. While our work targets conventional HPC clusters, ANTAREX focuses on ARM-based systems.

A manual approach for dynamic tuning has been described by Schöne and Molka [11]. The authors extend the VampirTrace framework with a library that allows users to specify a configuration file with specific optimal configurations for individual regions, which are later applied at runtime.

The AutoTune project [12] implemented the Periscope Tuning Framework (PTF), which enables static tuning of different tuning aspects and is extensible through plug-ins. The READEX project will combine PTF with the system scenario methodology [3] known from embedded systems to support dynamic tuning. The system scenario methodology describes the classification of different runtime situations into so-called scenarios, which have similar optimal configuration.

To the best of our knowledge, no dynamic auto-tuning framework exists that is capable of tuning HPC applications for energy efficiency with possible scalability to future Exascale systems.

3 Target environment

The READEX methodology described in the following sections targets scalable scientific applications running on future large-scale systems. These applications commonly exhibit structured hierarchical parallelism, i.e., continuous iterations in a time-stepping loop with domain decomposition using a combination of MPI and OpenMP to exploit multiple levels of parallelism available. While this can be regarded as a limitation of the applicability of the READEX methodology it does allow for specific techniques and optimizations to be applied, e.g., exploiting the regular iterative nature of the application to reduce the effort for finding optimal configurations. Moreover, by targeting this specific type of application the READEX methodology can still be applied to a large number of codes running on large-scale HPC machines.

The READEX methodology mainly focuses on a large-scale homogeneous system architecture comprised of a large number of nodes with a single or multiple CPU sockets. This limitation can later be lifted by extending the set of parameters described in Sect. 4.1.2 to include parameters specific to these platforms. However, heterogeneous performance characteristics among these nodes, as observed with recent Intel processor generations [13], can be taken into account by the methodology.

4 Formal definitions

The READEX methodology splits the application life-cycle into a design-time and a runtime part. At design-time, the application is analysed and a tuning model is created. We consider the design-time to be the time of development and performance tuning. The created tuning model is serialised and stored for later use at production time, called runtime. These steps are described in detail below.

4.1 Design-time

At design-time, the application is instrumented and analysed for dynamism and optimal configuration parameters are investigated. Regions that are deemed as beneficial targets for the dynamic tuning are identified and eventually stored in the tuning model together with their optimal system configurations.

4.1.1 Application execution

Our methodology builds on instrumentation of application and library functions as well as other regions of interest such as OpenMP parallel regions. We will use the term *region* to describe arbitrary code parts of the application.

The set of all *instrumented regions* is called R_{instr}, which denotes all regions that are instrumented and thus visible to the tuning system while the application is running. This set potentially contains a large number of regions that might run for only a short time, making a switch of parameters undesirable due to the cost associated with each switching operation. We thus need to reduce the set of regions to the set of *significant regions* $R \subseteq R_{instr}$. A region $r \in R$ is considered significant if it exhibits a certain execution time and thus is a suitable target for dynamic parameter switching. We expect a minimum runtime in the millisecond range to qualify a region as being significant.

During the application execution, a region might be executed multiple times where each execution is called an *instance of this region*.

We use so-called *identifiers* to predict the characteristics of the upcoming region instance. As identifiers, we initially consider the *region's name*, its *call-path*, and optionally *user-defined parameters*. The region call-path is the sequence of nested region instances. User-defined parameters are manually instrumented and may reflect any variable that has an influence on the computational characteristics of the region, such as function parameters. Additional identifiers can be added to the methodology if required.

The combination of an identifier and its value is called a *context element*. The context elements of a significant region describe a runtime situation rts. The sequence of runtime situations of a process p is considered its *execution* $exe_p := rts_1, rts_2, \ldots, rts_n$ with $n = len(exe)$. An application execution is the set of executions of all processes defined as $EXE := \{exe_p | \forall p \in P\}$ with P being the set of parallel application processes.

A process can have multiple threads with the same sequence of runtime situations. However, the READEX methodology only covers dynamic tuning on the level of processes to reduce the complexity of the tuning process.

4.1.2 System tuning

A *tuning parameter* tp is a parameter of the HPC system stack, e.g., the CPU frequency or tuning parameters in the OpenMP runtime library. READEX focuses on tuning parameters that have the potential of influencing the energy efficiency characteristics of extreme-scale applications, which can be controlled by the RRL at runtime.

A *system configuration cfg* describes the set of tuning parameters and their associated values. During the execution of the application, configurations can be switched at switching points sp at which the runtime library is entered and performs the switching based on the current rts. Switching points are enter and exit events of significant regions.

The best system configuration is determined using an *objective function o* that maps a given runtime situation and a system configuration onto a real number. The *objective* of the tuning process is to minimise or maximise a given objective function by varying the system configuration. Examples for objective functions are energy consumption, energy-delay-product, and time-to-solution.

A *static system configuration* cfg_{static} is considered a configuration that is either the default configuration or that was obtained through static auto-tuning. Eventually, both will be used as baselines for the evaluation of the results achieved by the READEX dynamic tuning. The best system configuration for each rts is stored in a tuning model that will be used at runtime to adapt the system to the application resource requirements. Due to the varying availability of tuning parameters on different HPC systems and application-specific impact of system configurations, a tuning model is specific to the combination of an application running on a given machine.

4.2 Tuning model

In order to reduce the number of configurations in the tuning model, the runtime situations (rts's) are partitioned into a set of *scenarios*. Multiple rts's are grouped into the same scenario if they have the same best-found configuration or if they have the same context. We use a *classifier cl* that maps each $rts \in RTS$ onto a unique scenario based on its context.

For each scenario, a *selector* chooses a configuration from a set of configurations that are optimal depending on the chosen objective. The selectors can have different implementations. They can either choose from a single configuration or select from a set of Pareto-optimal configurations based on runtime priorities for the objectives. Selectors may also probabilistically choose from a set of good configurations and thus enable dynamic adjustments to go beyond the limitations of the design-time analysis.

4.3 Tuning potential

Before the formalism described above can be successfully applied, we need to estimate the benefit of dynamically tuning an application, i.e., we need to determine the *tuning potential*. Tuning-relevant dynamism exists in an application if two region instances in the application execution exhibit computational characteristics that are different such that we can find different optimal configurations for the respective rts's. In this case, it is beneficial to switch configurations between these two regions to improve energy efficiency by adapting the system to actual needs of the application regions.

Once we know that tuning-relevant dynamism exists, we can infer the tuning potential by quantifying the improvement in the objective function compared to a static configuration cfg_{static}, such as the default configuration. The tuning potential must be determined for all rts's on all processes in order to estimate the tuning potential of the whole application.

The tuning potential can only be estimated through measurements, that is, by experimentally determining the computational characteristics of different regions. Whether this tuning potential can be achieved using the READEX methodology depends on the quality of the tuning model created at design-time, the switching overhead of the individual parameters, and mutual side effects between processes.

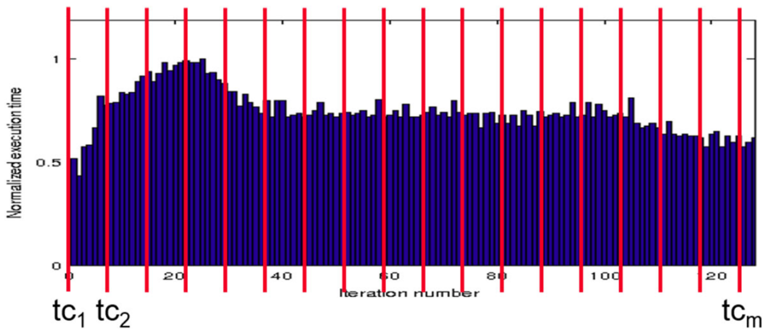


Fig. 1 Phase runtimes of an FEM application with tuning cycles marked *red*

4.4 Extension to inter-phase dynamism

Figure 1 displays the runtimes of different iterations of an industry-grade FEM application called Indeed. The figure shows that iteration runtimes are not uniform throughout the application run. The READEX methodology takes into account that the computational characteristics of the region instances might change across iterations, e.g., due to changing workload distribution. Thus, we extend the formalism by adding support for *inter-phase dynamism*, for which the definitions strongly align with the definitions presented above.

A *phase region* is a program region that defines the phases of an execution, e.g., the main time-stepping loop of a scientific application. Thus, all significant regions should be nested within this region. We initially assume that there is only exactly one phase region.

Similar to an rts, a *phase* ph is an instance of a phase region. We assume that the phases are executed collectively by all processes and that all processes go through the same *phase sequence* $ps = ph_1, ph_2, \ldots, ph_k$ with k denoting the number of iterations of the time-stepping loop.

Users may provide the tuning system with a so-called *phase identifier*, which should be chosen to reflect different behaviour across phases of a phase region. User-defined parameters can be used for this specification (see Sect. 4.1.1). This makes it possible to detect changing resource requirements over the course of an application execution due to changing computational load or a change in the employed algorithm based on the simulation progress.

For phases with different characteristics, the rts's of a region can be mapped onto different scenarios with different selectors. The partitioning into rts scenarios is now given by a *phase-aware classifier*, allowing the tuning model to cover the scope of inter-phase dynamism.

4.5 Extension for multiple inputs

The concepts described above can be extended by an input-aware classifier. Possible application inputs can be all external factors of an application that can potentially alter the computational characteristics, such as input data properties and the number of computing elements.

Similar to the previously introduced concepts, *input identifiers* can be exposed to the tuning system by the application developer using user-defined parameters. For each input configuration, an analysis run must be performed to determine the computational characteristics and scenarios of the input. Relevant inputs should be determined by the user. The partitioning for these scenarios will be performed by an input-aware classifier that maps rts's based on the input context, the phase context, and the rts context into scenarios.

5 READEX tool-suite approach

The approach of the READEX tool suite is introduced in the following sections. First the application is analysed for best configurations in the Design Time Analysis. The best-found configurations are stored as scenarios in a tuning model that guides the Runtime Application Tuning.

5.1 Design time analysis

The design time analysis (DTA) constitutes the first stage of the READEX methodology, which is used to analyse the target application and to generate a tuning model. The DTA stage uses PTF for experiment control and tuning model generation. Additionally, the READEX runtime library (RRL) is used for performance and energy measurement, instrumentation, and control of tuning parameters.

A high-level view of the DTA process is shown in Fig. 2. The DTA starts with the specification of the objective, i.e., the user can choose between different objective functions to optimise for, including energy efficiency, performance, or energy-delay-product. If available, the user may specify domain knowledge to further guide the tuning process. For example, by providing information on input parameters for the application or for individual functions.

Next, the instrumentation of the application is performed using automatic instrumentation techniques and potentially additional user instrumentation. In order to keep the runtime overhead low, the instrumentation has to be restricted to coarse granular program regions based on advanced automatic filtering techniques such as profile analysis for filter generation. The choice of a suitable instrumentation granularity is a common problem among performance analysis tools [14].

With the aid of the instrumentation, significant regions can be detected in the application. All insignificant regions will be ignored and later added to the instrumentation filter. Based on the selection of significant regions, the tuning potential analysis will be performed on the basis of profiling data. This actual selection will be carried out either using existing profiling data or by performing an additional profiling run of the application. The profiling data should contain function runtimes and hardware metrics used to estimate the computational characteristics of the individual regions. If no tuning potential can be detected, the DTA process aborts the tuning process with a warning.

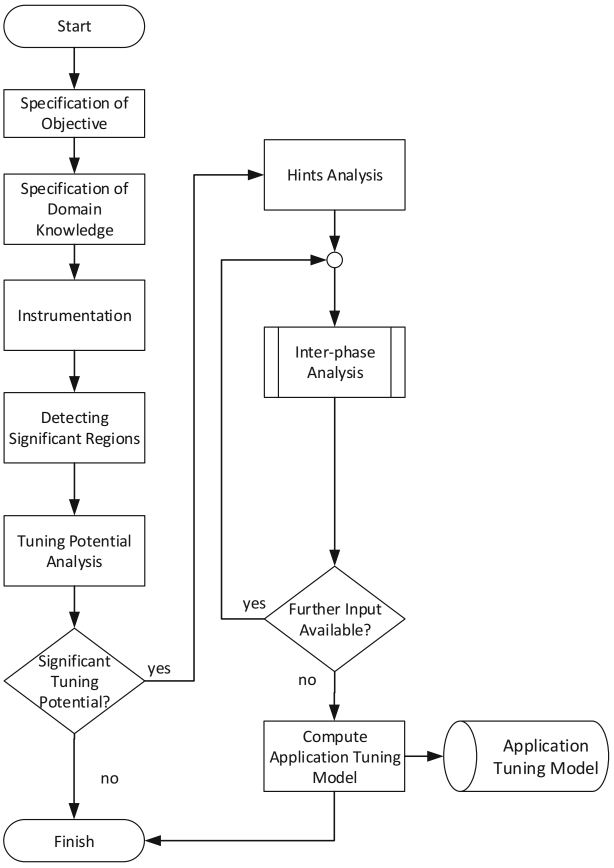

Start
Specification of Objective
Specification of Domain Knowledge
Instrumentation
Detecting Significant Regions
Tuning Potential Analysis
Significant Tuning Potential?
yes
no
Hints Analysis
Inter-phase Analysis
Further Input Available?
yes
no
Compute Application Tuning Model
Application Tuning Model
Finish

Fig. 2 High-level flowchart of the design-time analysis

If the tuning potential is significant, the inter-phase analysis starts by executing the application using PTF. Upon entering a phase region, a tuning cycle is started and the intra-phase analysis is performed, resulting in a tuning model for this phase. This process is repeated until the application finishes executing and no new tuning cycle is started. Upon exit of the application, the inter-phase tuning is computed from the different tuning models computed during intra-phase analysis iterations. If the user has provided different inputs to the DTA process, these additional inputs are used to re-execute the application so that an input-aware tuning model is created (also depicted in Fig. 2). After the tuning is complete, the overall application tuning model is created and serialised before the DTA process finishes.

The design-time analysis will be automated by the READEX tool-suite. The user is only responsible for providing for annotating the main iteration loop to identify different phases of the execution, re-compiling the application to insert the required instrumentation, and launching the application with the READEX tool-suite attached. However, he may provide the tuning system with additional manual instrumentation as described in Sects. 4.4 and 4.5.

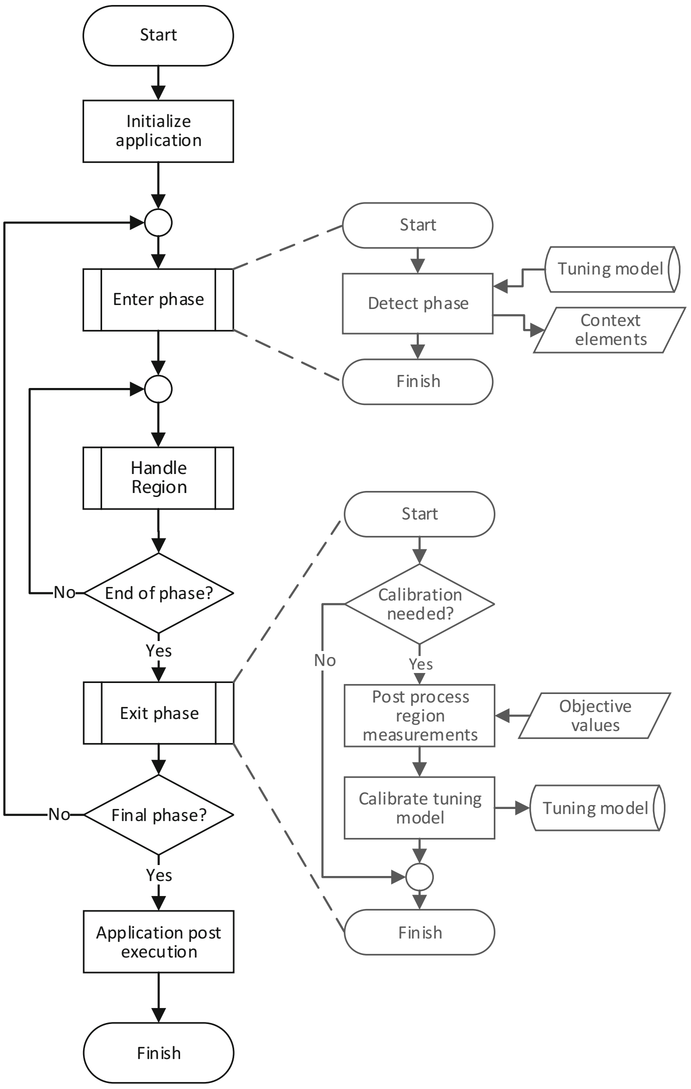


Fig. 3 Flowchart of the READEX runtime library (RRL) showing initialisation and phase handling. The *handle region* node is depicted in Fig. 4

5.2 Runtime application tuning

The runtime application tuning (RAT) ingests the tuning model created by the DTA process and dynamically adjusts the system while the application is running. The READEX Runtime Library (RRL) uses the same instrumentation as the DTA to trigger tuning actions based on the tuning model. The general control flow of the RAT process is depicted in Fig. 3.

During the start of the application, the measurement system is initialised by reading the tuning model and setting up the measurements required to determine the value of

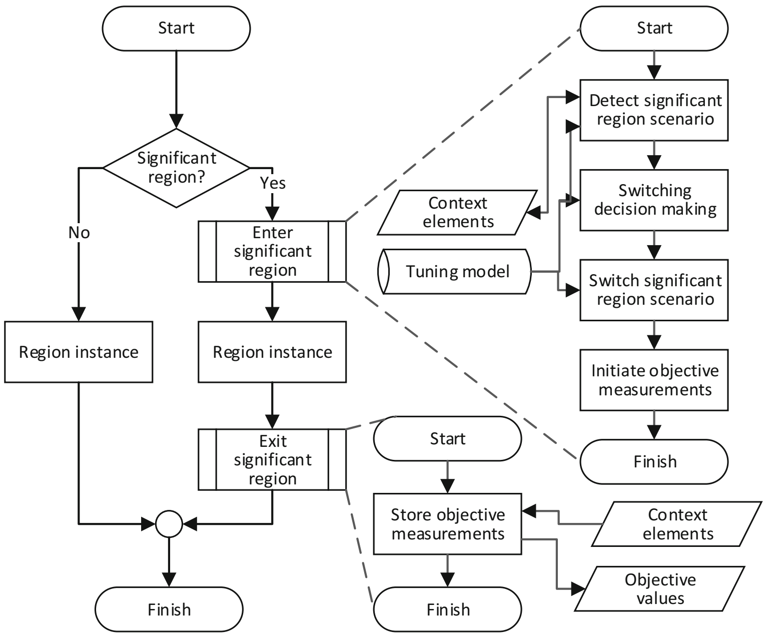


Fig. 4 Flowchart of the READEX runtime library (RRL) showing the handling of region enter and exit events

the objective function. This information is also stored in the tuning model. Additional measurements can be added to this set in order to feed the calibration process, as described later. Moreover, input parameters are evaluated, such as the number of CPUs available and input parameters provided by the application. All phase-external computation, i.e., code executed outside of a phase region, is ignored by the RRL and thus no tuning will be performed for these regions.

Upon entering a phase region, the RRL becomes active and closely monitors the events triggered by the application instrumentation. Moreover, the RRL identifies the phase entered to generate context elements for the tuning model of this phase. This lookup can be done without adding much overhead and is an essential step for the phase-aware tuning of the READEX approach. During the execution of a phase, all events generated by the instrumentation are inspected by the RRL and used to classify the current rts as shown in Fig. 4.

A region can either be significant or insignificant. For insignificant regions, no action is taken by the RRL except for book keeping, e.g., storing the call-path, and the region is executed immediately. However, for significant regions, the chosen classifier maps the rts onto a scenario by considering the current region, its call-path, and user-defined parameters. Depending on the information from the tuning model, the RRL may switch the current configuration. Lastly, the measurement of the objective value for this rts is initiated by storing the current value of an energy counter or storing the current timestamp. Consequently, the region is executed as usual until an event for entering or exiting a significant region is triggered. Currently, our approach does not allow for nested significant regions so only leaf nodes of the call tree can be considered as significant regions. We do not believe that this will limit the applicability of our approach since the granularity of significant regions can still be chosen to reflect changes in the computational characteristics at design-time.

Upon leaving a significant region, the measurements of the objective values are stored by reading the required metrics. For insignificant regions, no action is required by the RRL.

Figure 3 shows how the handling of regions is repeated until the end of a phase is reached. When a phase ends, the calibration step is executed, if required. This may be the case for every ten or hundred phase iterations or if a previously unseen rts is encountered during the phase. The objective measurements of all significant regions encountered during the last phase are post-processed to determine the objective value of each encountered rts.

In case of a previously unseen rts, the calibration either guesses a configuration based on existing measurements or triggers the recording of additional metrics the next time this rts is encountered. These measurements can then be used to create a suitable configuration by comparing the computational characteristics of this region with regions already stored in the tuning model. For known regions, the calibration step can adjust the configuration of specific scenarios and evaluate the impact of these changes on the objective function during the next phase. In all cases, the results of the calibration are written back to the tuning model for use in the next phase.

Eventually, if no further phase is to be executed the application will perform its final processing, which again will not be covered by the RRL. Upon exit of the application, the RRL will perform a tear-down, which may include the serialisation of the updated tuning model for post-mortem analysis of the calibration performed during the RAT.

6 Implementation

The implementation of the READEX Tool Suite will be based on existing tools that have proven scalability. As mentioned in Sect. 5.1, the DTA will be based on the PTF. The RRL will be based on the Score-P instrumentation and measurement infrastructure [15]. Both tools have been used in production on HPC systems for years and are thus well tested and stable. Moreover, the user's awareness of these tools lowers the entry barrier for the READEX Tool Suite. The overall envisioned architecture of the tool suite with the required extensions is depicted in Fig. 5 and will be further discussed below.

6.1 Design-time analysis

PTF will be extended to perform the DTA with the extensions depicted on the left side of Fig. 5. Most importantly, the *DTA Management* will control the overall execution of the DTA and includes two components: the *DTA Process Management*, which interacts with the PTF components to control the overall tuning workflow described in Sect. 5.1. The *RTS Management* will handle the *RTS Database*, which will eventually contain all rts's and their selectors. Based on this database, the *Scenario Identification* component

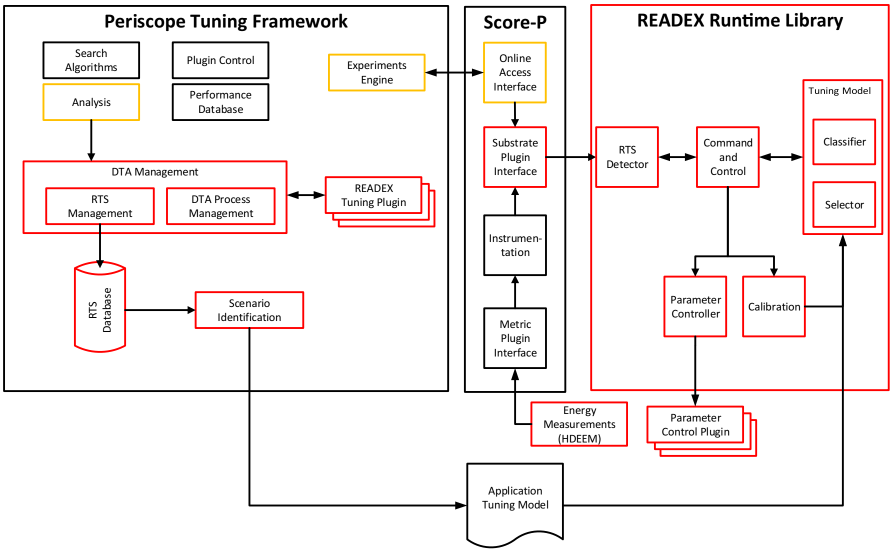


Fig. 5 Architecture of the READEX Tool Suite (*black* existing components; *red* new components to be developed; *yellow*: components to be extended) (Color figure online)

will construct the final application tuning model by aggregating the rts's into scenarios with corresponding classifiers according to their context and selector.

The *READEX tuning plugins* are controlled by the *plugin control* and select suitable tuning parameter values based on expert knowledge applied to performance characteristics measurements. The plugins make use of different search strategies and expert knowledge to shrink the search space. PTF provides multiple search strategies including *exhaustive search*, *individual search*, *genetic search*, and *random search*. The individual strategy optimizes tuning parameters assuming that they are independent. The genetic and random strategies are powerful search techniques for complex search spaces. Both can be enhanced in PTF with machine learning techniques to guide the selection of individuals based on previous tuning results. PTF's *Analysis Component* will be extended to provide hints to the design-time analysis, for example, the frequency of tuning cycles. The analysis component accesses performance data that is gathered in the *Performance Database*. The RRL (described below) is used to gather the performance data as well as to control the tuning parameters on the machines the parallel application is running on. The communication between PTF and the RRL is performed by the *Experiments Engine*, which communicate through the *Online Access* interface to start and stop the application execution, configure tuning parameters, and collect tuning results.

6.2 READEX runtime library

The READEX Runtime Library (RRL) will be based on Score-P and will be implemented as a substrate plugin [16]. Substrate plugins may access all management information and events generated from the instrumentation, e.g., through compiler, library, or user instrumentation. The plugins are loosely coupled with the Score-P infrastructure through a stable plugin API and thus can be maintained separately, which reduces the maintenance efforts for both the Score-P core developers as well as the plugin developers.

The architecture of the RRL is depicted on the right side of Fig. 5. Events generated by the Score-P instrumentation are passed through the substrate plugin interface to the *RTS detector* in the RRL, which determines whether an event constitutes the enter into a significant region or not based on the information from the *tuning model manager* (TMM), which holds the information contained in the *tuning model*. If a significant region has been detected, the RTS Detector collects all necessary information and passes it to the *command and control* (C&C) module, which passes it on to the TMM where the rts is mapped onto the upcoming scenario. The C&C receives the optimal configuration for the upcoming rts and applies it by sending the settings for the individual parameters to the *parameter controller*. The Parameter Controller controls the parameter settings through *parameter control plugins*, which provide a unified interface to the different parameters on the hardware and software layer.

The input to the TMM can be twofold:

1. During DTA, the configuration of the TMM is provided by PTF through the *Online Access Interface* (OAI) by sending tuning requests to the RRL that contain information on rts's and the configurations to be tried in the current phase for each rts. The tuning requests are also received through the substrate plugin interface and handled by the *OA event receiver*.
2. At runtime, the *application tuning model* is used as input to the TMM. The tuning model contains the classifier that maps rts's onto scenarios and the configurations that are used by the scenario specific selectors in the TMM to choose the optimal configuration with regards to the chosen objective function.

The *Calibration* component is only used at runtime to calibrate and further refine the tuning model as described in Sect. 5.2.

7 Use case study

The READEX project employs a co-design approach in which benchmarks and applications are manually tuned to gain insight into their computational characteristics. The results will influence the development of the READEX Tool Suite and vice versa. This section presents an application case study, which provides a supportive statement for the READEX approach. Using a model cube benchmark, we discuss the energy consumption evaluation of a Finite Element Tearing and Interconnecting (FETI) solver. These solvers are used to solve extremely large systems of linear equations on HPC systems. The measured characteristics illustrate the behaviour of various pre-processing and solve phases related mainly to the CPU frequency.

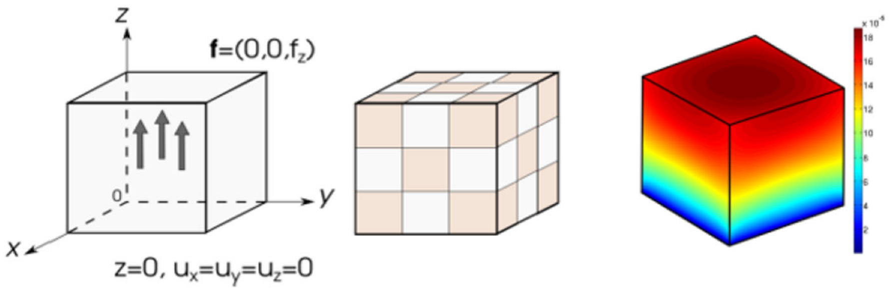


Fig. 6 The 3D linear elastic cube used as model problem

7.1 FETI methods and their implementations

Partial differential equations (PDEs) are often used to describe phenomena such as sheet metal forming, fluid flow, and climate modelling. The computational approaches taken to find solutions to such PDEs typically involve solving a large system of linear equations. When scientific applications solve PDEs that are too big to fit in the memory of a single machine or demand more processing power than a single machine can deliver, Domain Decomposition Methods (DDMs) are used to divide the original problem into smaller sub-domains that are distributed across the compute nodes of an HPC cluster. The FETI method forms a subclass of DDM, efficiently blending conjugate gradient (CG) iterative solvers and direct solvers.

The FETI method has both the parallel and numerical scalability to scale to tens of thousands of processors [17]. The PERMON [18] software package is an MPI-parallel implementation of the FETI method focusing on engineering applications and will be used as a target application throughout this section.

The two main phases of the FETI method are *pre-processing* and *solve*. In the pre-processing stage, the stiffness matrix K is factorised and the natural coarse space matrix G and coarse problem matrix GG^T are assembled. The latter matrix is also factorised. Both of these operations are among the most time consuming and thus the most energy consuming operations. The solver employs the CG algorithm, which consists of Sparse Matrix-Vector Multiplications (SpMV), vector dot products, or AXPY functions. For each iteration, it is necessary to apply the direct solver twice, i.e., forward and backward solves for the so-called pseudoinverse action and the coarse problem solution.

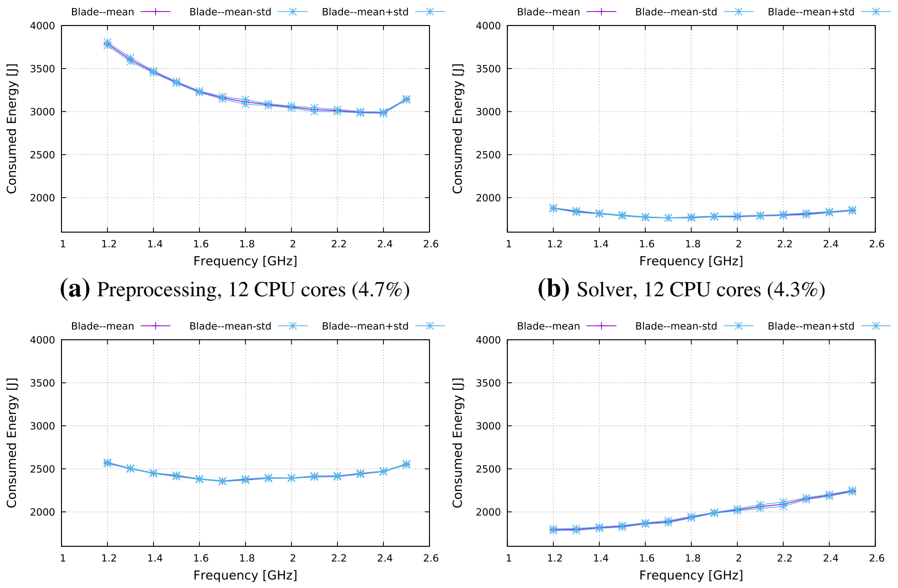
7.2 Model problem

As a benchmark, the 3D linear elastic cube was used with the bottom face fixed and the top one loaded with a surface force, as depicted in Fig. 6. For these computations, a mesh is generated and decomposed into subdomains by the PermonCube benchmark generator. The parallel mesh generation is controlled by two groups of parameters. The number of subdomains $N_S = XYZ$ is given by input parameters X, Y, Z (number of subdomains in each direction) and similarly the number of elements per subdomain edge is given by x, y, z. Decomposition into a large number of subdomains favourably affects the time of factorization of K and the subsequent solve. However, it also increases the size of the coarse problem, whose solution becomes a critical part of this method for a large number of subdomains.

7.3 Experimental setup

An initial set of tuning parameters has been identified for the READEX project. Although several parameters have been investigated so far, we concentrate on the CPU frequency and number of CPU cores used for this early-stage case study. As part of this study, we instrumented the application and manually selected individual significant regions. The instrumentation uses the HDEEM [19] infrastructure for accurate energy measurements with 1000 Sa/s on the Taurus system installed at TU Dresden. The Taurus system consists of over 1400 instrumented compute nodes each equipped with two 12-core Intel Xeon E5-2680v3 (Haswell-EP) processors. The default clock frequency is 2.5 GHz with turbo frequencies disabled. The instrumentation also adds switching points to dynamically switch the CPU frequency and measure the objective value. Thus, the function names identifying the regions implicitly serve as identifiers for the instances of the function executions, the so-called rts's. We will focus on two rts's, the pre-processing step and the solve step.

The measured energy consumption of the particular rts's under the given system configurations are depicted in Fig. 7. The measurements clearly indicate the existence of tuning potential since tuning-relevant dynamism can be observed, as described in more detail in the next section.



(a) Preprocessing, 12 CPU cores (4.7%)

(b) Solver, 12 CPU cores (4.3%)

(c) Preprocessing, 24 CPU cores (7.8%)

(d) Solver, 24 CPU cores (20%)

Fig. 7 Energy consumption of the PERMON preprocessing and solver stage for the model problem shown in Fig. 6 using different number of CPU cores and frequencies. Relative maximum energy savings presented in brackets

7.4 Manual tuning of PERMON

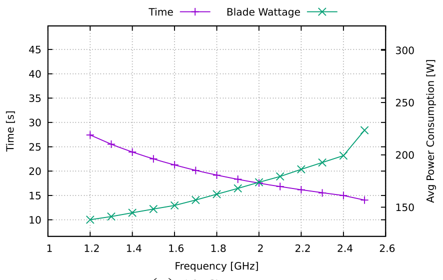
The measurements of the energy consumption of PERMON's FETI solver on a single computational node were performed to demonstrate the feasibility of the READEX dynamic tuning approach. Since the READEX Tool Suite is still under development, we can only provide manual tuning results at this time.

The relation of the CPU frequency, the number of CPU cores used, and the consumed energy for the complete solution of the problem is shown in Fig. 7 for the preprocessing stage and the CG solver stage. The figures show standard deviations and the mean values computed from 10 repetitions.

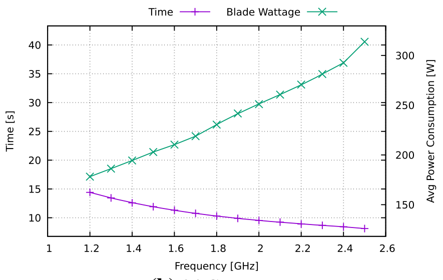
Significant differences in the energy consumption characteristics of the different application regions in relation to the number of used CPU cores can be seen. With 12 MPI processes, the processes are distributed in a cyclic fashion among the two sockets of the node. For 24 used CPU cores, the optimal frequency is lower than for 12 CPU cores indicating that the application benefits from the higher per-core memory bandwidth available for 12 processes.

Nevertheless, the data indicates that PERMON exhibits tuning-relevant dynamism: When using 12 processes, the optimal frequency for the preprocessing stage has been found to be 2.4 GHz while for the solver stage it is 1.7 GHz. Compared to the default frequency, these frequencies provide a reduction in energy consumption by 4.7 and 4.3%, respectively. For 24 processes, the differences are more significant and show optimal frequencies at 1.7 GHz for the preprocessing and 1.2 GHz for the solver stage. Here we can achieve 7.8 and 20% savings in energy consumption compared to the default frequency.

It is worth noting that the energy consumption for 24 CPU cores is lower than for 12 cores. For 12 CPU cores, the higher per-core memory bandwidth results in higher optimal frequencies compared to using 24 CPU cores. However, the cyclic distribution of the processes among the two sockets required to make use of the full memory bandwidth prohibits any energy-saving effects from idle states of one of the socket. Only individual cores may make use of idle states, slightly reducing the average power consumption, as depicted in Fig. 8. However, the additional computing performance available from 24 CPU cores eventually leads to improved energy efficiency (2350



(a) 12 CPU cores



(b) 24 CPU cores

Fig. 8 Runtimes and average node power consumption for different frequencies when using 12 and 24 CPU cores for the preprocessing stage of PERMON

vs 3000 kJ) due to lower time-to-solution (10.9 vs. 15 s for optimal frequencies). As expected, the default frequency of 2.5 GHz provides the lowest time-to-solution for both configurations. The READEX methodology will incorporate additional objective functions, e.g., time-to-solution or energy-delay-product, to account for different optimal parameter settings of different target objectives.

This simple example demonstrates the complexity of dynamic tuning and motivates the need for a tool suite for automatic tuning for energy efficiency.

8 Conclusion and outlook

This paper introduced the READEX tools-aided methodology for dynamic tuning of scalable HPC applications. The methodology consists of a design-time analysis part, in which the target application is analysed for dynamism. Based on this analysis a tuning model is created, which contains best-found system configurations for individual application regions. This tuning model is later used at runtime when the application is running in production to dynamically adapt the system to the actual needs of the application.

This paper has introduced the formal definitions that form the basis of the methodology and described the envisioned implementation of the tool suite. The design-time analysis will be based on the Periscope Tuning Framework to perform parameter studies and create the tuning model. At runtime, the lightweight READEX Runtime Library will be used to detect upcoming system scenarios and adapt the necessary system parameters dynamically based on the information provided by the tuning model. The runtime library will be based on Score-P, which has proven to be scalable on some of the largest HPC systems available today. By building on top of existing scalable software tools, we ensure both stability and usability of the READEX Tool Suite. On the one hand, these tools have been well tested by developers and users and thus provide a solid basis for the READEX software. On the other hand, they are already known to end users, which significantly reduces the entry barrier and allows for reuse of existing documentation.

The READEX project employs a co-design approach in which selected applications are manually tuned for energy efficiency. The results and insights of this manual tuning will guide the development of the tool suite and vice versa. This paper presents early results of these manual tuning efforts, highlighting the feasibility of the READEX approach for dynamic application tuning by demonstrating the tuning potential of a FETI solver and the achieved reduction in energy consumption for a model problem. The complexity of the manual tuning using two parameters clearly motivates the need for an automatic approach as is being developed by the READEX project.

By developing a tools-aided methodology for dynamic tuning for energy efficiency, the READEX project contributes to the development of energy-efficient scalable HPC applications. The tool suite is currently work in progress and first early results are to be expected soon. The manual tuning efforts will continue to investigate the READEX target applications, adding more fine-grained instrumentation and measurement to the applications to improve the tuning potential and reduction in energy consumption.

Acknowledgements The research leading to these results has received funding from the European Union's Horizon 2020 Programme under grant agreement number 671657. The authors would like to thank Yury Oleynik for his contributions.

References

1. Bergman K, Borkar S, Campbell D, Carlson W, Dally W et al (2008) Exascale computing study: technology challenges in achieving exascale systems
2. Gheorghita SV, Palkovic M, Hamers J, Vandecappelle A, Mamagkakis S, Basten T, Eeckhout L, Corporaal H, Catthoor F, Vandeputte F (2009) System-scenario-based design of dynamic embedded systems. ACM Trans Des Automation of Electronic Systems (TODAES)
3. Filippopoulos I, Catthoor F, Kjeldsberg PG (2013) Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios. Design Automation for Embedded Systems
4. César E, Moreno A, Sorribes J, Luque E (2006) Modeling Master/Worker applications for automatic performance tuning. Parallel Computing
5. Tiwari A, Chen C, Chame J, Hall M, Hollingsworth JK (2009) A Scalable Auto-Tuning Framework for Compiler Optimization. In: International Symposium on Parallel & Distributed Processing (IPDPS). IEEE
6. Tiwari A, Hollingsworth J (2011) Online adaptive code generation and tuning. In: International Parallel Distributed Processing Symposium (IPDPS). IEEE
7. ENabling technologies for a programmable many-CORE (ENCORE), last accessed June 15, 2016. [Online]. Available: http://cordis.europa.eu/project/rcn/94045_de.html
8. Duran A, Ayguaé E, Badia RM, Labarta J, Martinell L, Martorell X, Planas J (2011) OmpSs: a proposal for programming heterogeneous multi-core architectures. Parallel Processing Letters
9. Benkner S, Pllana S, Träf JL, Tsigas P, Dolinsky U, Augonnet C, Bachmayer B, Kessler C, Moloney D, Osipov V (2011) PEPPHER: Efficient and productive usage of hybrid computing systems," *IEEE Micro*
10. Silvano C, Agosta G, Cherubin S, Gadioli D, Palermo G, Bartolini A et al (2016) The ANTAREX Approach to Autotuning and Adaptivity for Energy Efficient HPC Systems. In: Proceedings of the ACM International Conference on Computing Frontiers
11. Schöne R, Molka D (2014) Integrating performance analysis and energy efficiency optimizations in a unified environment. Computer Science-Research and Development
12. Benkner S, Franchetti F, Gerndt M, Hollingsworth JK (2014) Automatic Application Tuning for HPC Architectures (Dagstuhl Seminar 13401) Dagstuhl Reports
13. Schuchart J, Hackenberg D, Schöne R, Ilsche T, Nagappan R, Patterson MK (2016) The Shift from Processor Power Consumption to Performance Variations: Fundamental Implications at Scale, Computer Science-Research and Development
14. Malony AD (1991) Event-based performance perturbation: a case study, SIGPLAN Not
15. Knüpfer A, Rössel C, an Mey D, Biersdorff S, Diethelm K et al (2012) Score-P: a joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In: Tools for High Performance Computing. Springer
16. Schöne R, Tschüter R, Ilsche T, Schuchart J, Hackenberg D (2006) Score-P as an extendable measurement environment, submitted for review
17. Říha L, Brzobohatý T, Markopoulos A, Meca O, Kozubek T (2016) Massively Parallel Hybrid Total FETI (HTFETI) Solver. In: Proceedings of the Platform for Advanced Scientific Computing Conference. ACM
18. Hapla V, Horak D, Pospisil L, Cermak M, Vasatova A, Sojka R (2016) Solving Contact Mechanics Problems with PERMON. Springer International Publishing
19. Hackenberg D, Ilsche T, Schuchart J, Schöne R, Nagel W, Simon M, Georgiou Y (2014) HDEEM: High Definition Energy Efficiency Monitoring. In: Energy Efficient Supercomputing Workshop (E2SC)