Chapter 6
Run-Time Exploitation of Application Dynamism for Energy-Efficient Exascale Computing

Per Gunnar Kjeldsberg, Robert Schöne, Michael Gerndt, Lubomir Riha, Venkatesh Kannan, Kai Diethelm, Marie-Christine Sawley, Jan Zapletal, Andreas Gocht, Nico Reissmann, Ondrej Vysocky, Madhura Kumaraswamy, and Wolfgang E. Nagel

1 Introduction and Context

In the embedded systems domain, energy efficiency has been a main design constraint for more than two decades. More recently, this has also become a major concern in high performance computing (HPC). Even though these two domains

P. G. Kjeldsberg (✉) · N. Reissmann
Norwegian University of Science and Technology, NTNU, Trondheim, Norway
e-mail: pgk@ntnu.no; nico.reissmann@ntnu.no

R. Schöne · A. Gocht · W. E. Nagel
Technische Universität Dresden, Dresden, Germany
e-mail: robert.schoene@tu-dresden.de; andreas.gocht@tu-dresden.de; wolfgang.nagel@tu-dresden.de

M. Gerndt · M. Kumaraswamy
Technische Universität München, München, Germany
e-mail: gerndt@in.tum.de; kumarasw@in.tum.de

L. Riha · J. Zapletal · O. Vysocky
VSB - Technical University of Ostrava, Ostrava, Czech Republic
e-mail: lubomir.riha@vsb.cz; jan.zapletal@vsb.cz; ondrej.vysocky@vsb.cz

V. Kannan
Irish Center for High-End Computing, Galway, Ireland
e-mail: venkatesh.kannan@ichec.ie

K. Diethelm
Gesellschaft für numerische Simulation, Braunschweig, Germany

University of Applied Sciences Würzburg-Schweinfurt, Schweinfurt, Germany
e-mail: diethelm@gns-mbh.com

M.-C. Sawley
Intel ExaScale Labs, Paris, France
e-mail: marie-christine.sawley@intel.com

F. Catthoor et al. (eds.), *System-Scenario-based Design Principles and Applications*, https://doi.org/10.1007/978-3-030-20343-6_6

are different in many respects, techniques that have proven to be efficient in one may also be beneficial in the other. The split design-time and run-time approach of system-scenario-based design, which is described in this book, is being applied successfully in embedded systems, both to increase performance and to reduce power and energy consumption [7, 8, 12]. In the HPC domain, auto-tuning is used either at design-time to statically tune the system configuration or at run-time through compute intensive estimation of the dynamic requirements [2, 15]. Combining these approaches from embedded systems and HPC has the potential of giving substantial synergies in both domains.

A constantly growing demand for data center computing performance leads to the installation of increasingly powerful and complex systems, characterized by a rising number of CPU cores as well as increasing heterogeneity. This makes optimization of HPC applications a complex task, which demands significant programming effort and high levels of expertise. With a growing computational performance, there is typically also an increase in a system's energy consumption, which in turn is a major driver for the total cost of ownership of HPC systems. Furthermore, limitations to chip temperature and cooling capabilities can make the performance of Exascale HPC systems power-bound. However, developers commonly focus on the implementation and improvement of algorithms with regard to accuracy and performance, neglecting possible improvements to energy efficiency. Often, programmers lack the platform and hardware knowledge required to exploit these measures, which is an important obstacle for their use both in the embedded and HPC domains.

The European Union Horizon 2020 project READEX [6] (Run-time Exploitation of Application Dynamism for Energy-efficient eXascale computing) tackles these challenges by embracing the significant potential for improvements to performance and energy efficiency that result from dynamic resource requirements of HPC applications similar to those seen in embedded systems. Examples are alternating application regions and load-changes at application run-time. Such dynamism can be found in current HPC applications, including weather forecasting, molecular dynamics, or adaptive mesh-refinement applications.

These applications often operate in an iterative manner, e.g., using a time step loop as the main control flow. Each iteration of such a program loop can be regarded as a phase of the application execution. In this context, intra-phase dynamism describes the changes in resource requirements and computational characteristics between different code regions executed by a single iteration, e.g., the change between memory- and compute-bound kernels. Intra-phase dynamism can be exploited by adjusting the system to the resource requirements of the current code region. Alternatively, inter-phase dynamism describes the changes in application behavior between iterations or phases. As the execution progresses, the required computation can vary, either on single processes—causing imbalances—or on all processes with a homogeneous rise in computational complexity on all processing elements.

As discussed in Chap. 1, it is expected that applications running on future embedded systems platforms will exhibit even higher levels of dynamism. This will

be mainly due to the increased demand for data movement performance between processing elements, both on intra- and inter-node levels, and more complex multi-level memory hierarchies. Furthermore, the rise of many-core co-processors and accelerators introduces new degrees of freedom such as offloading and scheduling. For extreme-scale HPC systems, this trend will be even more critical.

The READEX project has developed and implemented a tools-aided methodology that enables HPC application developers to exploit dynamic application behavior when run on current and future extreme parallel and heterogeneous multi-processor platforms. READEX combines and extends state-of-the-art technologies in performance and energy efficiency tuning for HPC with dynamic energy optimization techniques for embedded systems. Many of the techniques developed for HPC systems can also be fed back to the embedded systems domain, in particular with the increasing performance seen in embedded multi-processor system-on-chip [10].

The general concept of the READEX project is to handle application energy efficiency and performance tuning by taking the complete application life-cycle approach. This is in contrast to other HPC approaches that regard performance and energy tuning as a static activity, which takes place in the application development phase. With inspiration from system-scenario-based design, READEX has developed a (semi-)automatic dynamic tuning methodology spanning the development (design-time) and production/maintenance (run-time) phases of the application life-cycle. Furthermore, a novel programming paradigm for application dynamism was developed that enables domain experts to pinpoint parts of the application and/or external events that influence the dynamic behavior. This can reduce the energy consumption even further, compared to a purely automatic approach.

The rest of this chapter is organized as follows: After a review of related work and project background in Sect. 2, a description of the READEX concepts is given in Sect. 3. This is followed by results from experiments with a realistic industrial HPC application in Sect. 4. Section 5 concludes this chapter with a summary.

2 Auto-tuning of HPC Systems

Given that system-scenario-based design is fully covered in other chapters of this book, this section focuses on how dynamic behavior is handled in HPC systems.

While a small number of dynamic auto-tuning methodologies and tools existed for run-time optimizations [3, 20], there is currently no single standalone dynamic auto-tuning framework with the capability to target the full breadth of large-scale HPC applications being used in academia and industry, both now and on the road to Exascale.

Still, several EU research projects are approaching the challenge of tuning for performance and energy efficiency by either introducing entirely new programming models or leveraging existing prototype languages. An example of the latter is the ENabling technologies for a programmable many-CORE (ENCORE) project [5],

which aims to achieve massive parallelism relying on tasks and efficient task scheduling using the OmpSs programming model [19]. The READEX project takes a different approach by developing a new generic programming paradigm, which allows to express and to utilize dynamism of applications in the automatic tuning process.

The Performance Portability and Programmability for Heterogeneous Many-core Architectures PEPPHER project [1] has developed a methodology and framework for programming and optimizing applications for single-node heterogeneous many-core processors to ensure performance portability. With Intel as a key partner in the project, READEX goes one step further and provides a framework that supports the heterogeneity of the system in the form of tuning parameters, which enable large-scale heterogeneous applications to dynamically (and automatically) adapt heterogeneous resources according to run-time requirements.

The ANTAREX project [18] creates a domain specific language (DSL), which distributes the code 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, the ANTAREX project focuses on ARM-based systems.

Nowadays, most performance engineering tools focus on collecting and presenting information to the users, while only a few focus on the automation of the performance optimization process (auto-tuning). One example of the latter is the periscope tuning framework (PTF) developed in the EU FP7 ICT AutoTune project [2, 15]. PTF automatically finds optimized system configurations for whole application runs, effectively averaging the benefits of system adaptation over the whole run-time of the application (static tuning). With these static auto-tuning techniques, improvements in energy efficiency of up to 10% for application runs have been achieved while keeping the performance degradation to a few percent [4].

PTF's main principles are the use of formalized expert knowledge and strategies, an extensible and modular architecture based on tuning plugins, automatic execution of experiments, and distributed, scalable processing. PTF provides a number of predefined tuning plugins, including:

- Dynamic voltage and frequency scaling (DVFS)
- Compiler flags selection
- MPI run-time environment settings
- Parallelism capping in OpenMP
- MPI master–worker pattern settings

PTF also provides an interface for the development of new plugins. It builds on the common performance measurement tools infrastructure Score-P [11], which has proven to be scalable on current petascale systems.

3 The READEX Concept

The READEX concept combines the system scenarios methodology with automatic energy and performance tuning into a holistic tools-aided methodology, spanning major parts of the HPC application life-cycle, i.e., application development (design-time) and production runs (run-time). Figure 6.1 provides a high-level overview of the methodology.

3.1 Application Instrumentation and Analysis Preparation

During the first step of the methodology, the application is instrumented by inserting probe functions around different regions in the application code. A region can be any arbitrary part of the code, for instance, a function or a loop nest. The instrumentation itself can be done automatically or by letting the user provide application domain knowledge using a new programming paradigm.

The programming paradigm enables users to expose parameters, which describe the dynamic behavior of the application to the READEX tool suite. These applica-

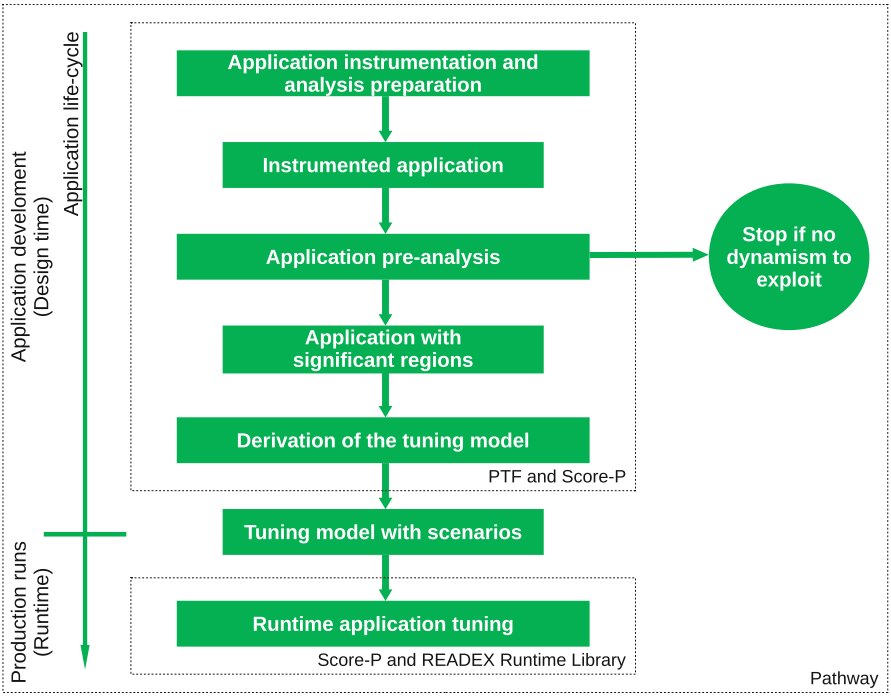

Fig. 6.1 Overview of the READEX methodology

tion domain parameters enhance the identification of different system scenarios and hence extend the adaptation of system configurations in order to improve overall application performance and energy characteristics.

One example of these parameters is the definition of different input data sets. Exposing such information to READEX enables the tools to attribute different compute characteristics to the varying input. The paradigm also enables developers to expose additional application-level tuning parameters to the tuning process, e.g., alternative code-paths that will be chosen based on the provided identifiers.

In addition to the optional information provided by the user, probe functions are automatically inserted around user code regions and by linking instrumented versions of relevant programming libraries using existing technologies from the Score-P infrastructure. This allows for fine-grained analysis and tuning.

3.2 Application Pre-analysis

The READEX analysis strategy is based on the performance dynamics analysis capabilities of PTF, which automatically characterizes present dynamism and indicates the optimization potential. The latter gives the user an estimate of the performance and energy efficiency gains that can be realized using the READEX methodology.

In the pre-analysis step, the application is run once with a representative data set. During this run, relevant timing information is recorded. The gathered performance data reveals application regions with run-times above a certain threshold. The instrumentation of fine-grained regions with run-time below the threshold is then removed, e.g., through the definition of an instrumentation filter, to reduce the perturbation of the application. This step prepares the application for all following steps of the methodology.

In a second iteration, the application is run again with the same, or extended, data set and relevant performance and energy metrics are collected. This results in time-series of measurements representing temporal evolution of each region's computational characteristics over multiple application phases. Similarly to what is described in the use-case scenario section of Chap. 2, each region's execution, which is represented by a point in this space, corresponds to a run-time situation (RTS). A first analysis reveals whether relevant code regions exist that exhibit enough dynamism to make the following tuning steps worthwhile. The dynamism can result from differences in compute- and memory-bound operation, either between regions or between different runs of a given region. Regions found to have such dynamism are defined to be significant, and will be tuned in the following design steps. If no such dynamism is detected the tuning should be aborted due to homogeneous application behavior. Since the run-time tuning necessarily causes an overhead, a minimum dynamism threshold is defined to ensure a satisfactory overall gain.

3.3 Derivation of the Tuning Model

In order to build a tuning model that guides the adaptation of the system (both application and platform) to the dynamically changing requirements, PTF, Score-P, and the later described READEX Run-time Library are used to perform an automatic search for optimal system configurations for the significant regions identified in the previous step. To configure the platform according to each experiment, the READEX Run-time Library (RRL) is used. Details regarding the RRL are given in Sect. 3.4.

Exploration of the space of possible tuning configurations is controlled by PTF tuning plugins. Each tuning plugin is responsible to tune a specific aspect, which can include one or multiple related tuning parameters. READEX has developed and supports a number of plugins for hardware, system software, and application aspects.

The architecture of the design-time tools required to explore optimal system configurations is depicted in Fig. 6.2. To determine the optimal platform configurations, PTF runs and measures the instrumented application. Here, various system configurations are evaluated in terms of the requested objective functions for the identified RTSs. Afterwards, the results are stored in the RTS database.

Since the search space for optimal configurations is potentially large, a number of possible search strategies were identified as part of the project. This includes heuristics based parameter selection, inter-phase comparisons with an underlying

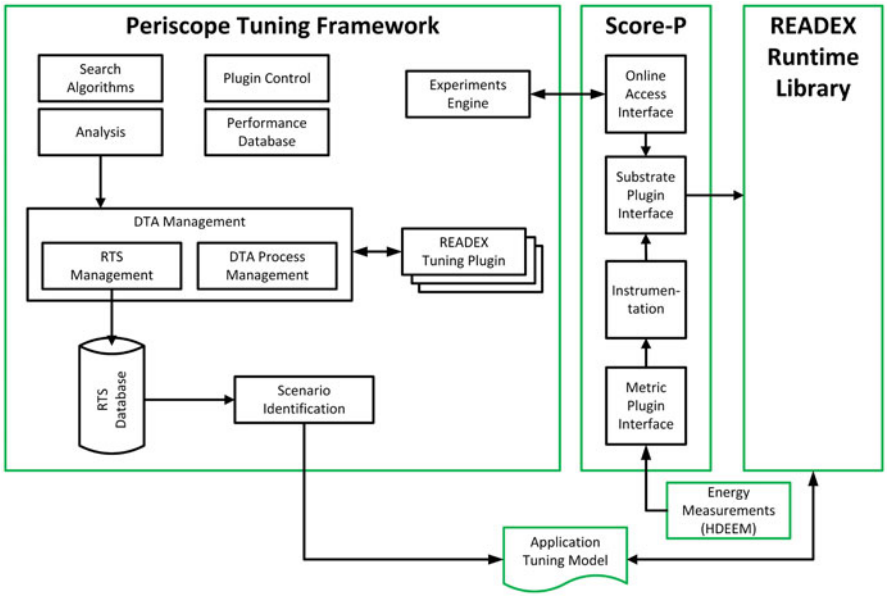

Fig. 6.2 Architecture of the design-time part of the READEX tool suite

approximation of the expected objective function, or a simple comparison with the objective function values taken during a baseline run.

After all relevant system configurations are evaluated for all RTSs, a scenario identification module groups RTSs into a limited number of scenarios, e.g., up to 20. Each scenario is represented with a common system configuration. The bound on the number of scenarios limits the frequency and associated overhead from configuration switching at run-time. As in all system scenario approaches, it is necessary to predict upcoming scenarios at run-time. This is done using the provided identifiers. An RTS is mapped to a scenario based on its signature, i.e., the current identifier values. An example of such an identifier is the call-path that has been followed to reach the current region. A configuration selector is also generated. In its simplest form, this is a function returning a set of tuning parameter values according to a one-to-one mapping between scenario and configuration. A given scenario can, for example, directly be used to specify the voltage and frequency setting to apply. The set of scenarios, the RTS signatures pointing to the scenarios, and the system configuration of each scenario, are stored in the form of a serialized text file as an application tuning model (ATM), to be loaded and applied during production runs at run-time.

3.4 Run-Time Application Tuning

Once the analysis is finished and the ATM is created, the application can be optimized in production. For this, the READEX Run-time Library (RRL) is used. It connects to the performance measurement infrastructure Score-P via a plugin interface [17], and reuses the existing code instrumentation from design-time tuning. The RRL reads the previously obtained knowledge about application dynamism, stored in the ATM, to optimize the application's energy consumption. For already seen RTSs, the optimal configuration is directly extracted from the tuning model. For unseen RTSs a calibration mechanism is used.

The architecture of the RRL is depicted in Fig. 6.3. Note that even though it is included in the figure, PTF is not used during production runs. As mentioned in Sect. 3.3, the RRL is used at design time to set the system configuration during the search PTF performs for optimal configurations. At run-time, the RRL obtains the scenarios with configurations from the ATM, which were generated by PTF at design-time.

A production run of an application starts with loading the ATM into the tuning model manager (TMM). When a new region is entered during the application execution, Score-P notifies the control center, which passes the information to the RTS Handler. The RTS Handler then checks if the region is significant and gets the best configuration for the current region from the TMM. Finally, the RTS Handler passes this configuration down to the parameter controller, which configures the different parameter control plugins (PCPs). PCPs are responsible for setting different system configurations like the CPU frequency or the number

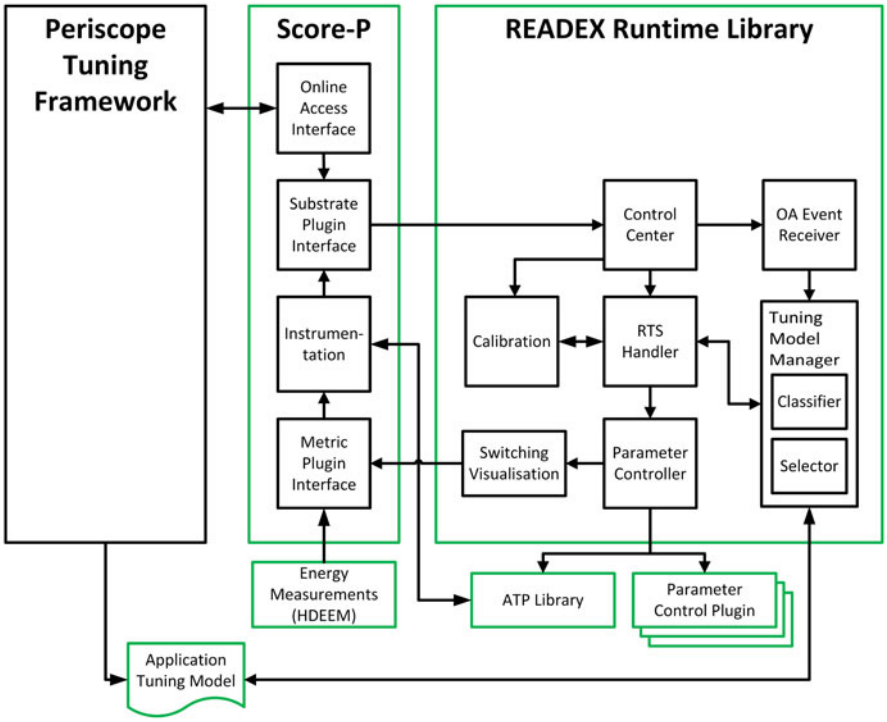

Fig. 6.3 Architecture of the READEX Run-time Library (RRL)

of OpenMP threads. They are employed both during the derivation of the tuning model at design-time and during a production run.

During the application run, a calibration mechanism can handle unseen RTSs and changes to the environment. The calibration is based on established machine learning approaches, the details of which are outside the scope of this chapter. Once the calibration detects a new optimal configuration for a certain scenario, the tuning model is updated. This leads to a constantly improving tuning model.

4 Experiments

The READEX project considers two different metrics for evaluation of the project success: the achieved improvement in energy efficiency compared to the default system configuration, measured in energy-to-solution, and the time and effort required to achieve this improvement compared to performing manual tuning.

For evaluation and validation of the project results, READEX has employed a co-design process in which the auto-tuning methodology and the tool suite have

```
int main( int argc, char ** argv ) {

    // Initialize application

    // insignificant region
    read_mesh( ... );
    // significant region, compute intensive
    assemble_Vh( ... );
    // significant region, compute intensive
    assemble_Kh( ... );
    // significant region, memory bound, MKL NUMA effects
    gmres_solve( ... );
    // significant region, I/O bound
    print_output( ... );

    // Finalize application

    return 0;
}
```

Listing 6.1 BEM4I sketch

been developed in parallel with a manual tuning effort of selected applications and computational libraries. The information exchange that resulted from this strategy has benefited both, the creation of the tools-aided methodology, and the manual tuning efforts.

The goal of this case study is to show how application dynamism can be exploited to improve energy efficiency. The evaluation was done on the *taurus* system installed at Technische Universität Dresden. The system is equipped with more than 1400 power-instrumented nodes with two 12-core Intel Xeon E5-2680v3 (Haswell-EP) processors each. The power-instrumentation allows for scalable and accurate energy measurements with a fine spatial and temporal granularity (CPU, memory, and whole node with up to 1000 Samples/s) [9]. The tests were performed on a single compute node. Hence, overall interconnect and distribution of processes in the network do not influence the results. To further validate the READEX methodology in the HPC environment the tool suite has also been tested on massively parallel applications. Scalability experiments with the ESPRESO library developed at IT4Innovations are provided in [16] but are outside the scope of this chapter.

Partial differential equations (PDEs) are often used to describe phenomena such as sheet metal forming, fluid flow, and climate modeling. One of the numerical approaches to solve PDEs is the boundary element method (BEM) as implemented, for example, in the BEM4I library [13]. In contrast to volume based methods, such as the finite element/differences/volume methods, BEM gives dense matrices whose assembly (V_h, K_h) results in compute bound code. This fact is even more pronounced when the assembly kernels are parallelized and vectorized as in the case of BEM4I [14, 21]. However, as shown in Listing 6.1, BEM4I also uses the iterative GMRES solver. This solver is based on the matrix–vector product as implemented

in the Intel Math Kernel Library (MKL), which tends to be less compute intensive and results in memory bound computation. Furthermore, printing the results for visualization leads to an I/O bound region. This dynamic behavior makes the BEM4I library suitable for testing the READEX methodology. The application was compiled with the Intel 2017 compiler with `-O3 -xcore-avx2` flags to enable the AVX2 vector instruction set available in the Haswell CPUs. The environment variable `KMP_AFFINITY` was set to `compact,granularity=core`.

For the experiment, both manual and READEX assisted tuning were performed. Listing 6.1 shows the relevant regions to be the assembly of two system matrices V_h, K_h, the iterative GMRES solver, and the I/O region, all included in a single non-iterative phase region. The HW and run-time parameters that were analyzed include the core/uncore frequency (ranging between 1.3 and 2.5 GHz/1.2 and 3.0 GHz, respectively) and the number of OpenMP threads (ranging from 4 to 24). The results that were obtained by automatic instrumentation with the READEX tool suite, i.e., application pre-analysis to detect significant regions, PTF (generation of the tuning model), and RRL (dynamic run-time switching), are compared with results obtained using the MERIC tool (tuning model and dynamic switching) developed at IT4Innovations for the purposes of manual application tuning. Note that READEX automatically identified all regions previously annotated manually for MERIC. With such an automatic instrumentation, tuning effort is decreased significantly.

Table 6.1 presents the HW and run-time configurations obtained by MERIC, namely the default configuration, the static optimum applied for the whole phase, and the optimal setting for each annotated region. It can be seen that the analysis leads to expected results with high core frequency, low uncore frequency, and a high number of threads for the compute bound assembly (V_h and K_h). For the memory bound solver (GMRES), the manual tuning results in a low core frequency, high uncore frequency, and the use of eight threads. The latter overcomes non-uniform memory access (NUMA) effects of the dual socket computational node. While static savings reach 15.7%, the dynamic switching among individual configurations increases the savings to 34.1%.

The results obtained by the READEX tools are given in Table 6.2. In comparison to the default, the READEX approach was able to achieve 34.0% energy savings.

Table 6.1 Optimal frequencies and energy saving for the BEM4I solver (MERIC)

HW conf. unit	Region –	Core freq. GHz	Uncore freq. GHz	Threads –	Energy saving %	Time saving %
Default	Phase	2.5	3.0	24	–	–
Static	Phase	2.5	2.2	16	15.7	−6.2
Dynamic	V_h	2.5	1.4	24	–	–
	K_h	2.1	1.4	24	–	–
	GMRES	1.7	2.2	8	–	–
	I/O	2.5	2.2	4	–	–
	All	–	–	–	34.1	10.9

Table 6.2 Optimal frequencies and energy saving for the BEM4I solver (PTF + RRL)

HW conf. unit	Region –	Core freq. GHz	Uncore freq. GHz	Threads –	Energy saving %	Time saving %
Default	Phase	2.5	3.0	24	–	–
Dynamic	V_h	2.5	1.4	24	–	–
	K_h	2.1	1.4	24	–	–
	GMRES	1.7	2.2	8	–	–
	I/O	2.5	3.0	8	–	–
	All	–	–	–	34.0	10.9

Note that the fully automatic READEX tool suite is able to reproduce the optimal settings from the manually tuned application.

Although this is not common for a general application, the tuning of energy also leads to the decrease in run-time in the case of BEM4I. This is caused by NUMA effects of the MKL solver—the tuned version runs on eight threads and due to the compact affinity all threads run on a single socket.

5 Conclusions

Energy efficiency and extreme parallelism are the major challenges on the road to Exascale computing. The European Union Horizon 2020 project READEX has addressed these by providing application developers with a tools-aided methodology for a combined design-time/run-time approach for dynamic adaptation to changing resource requirements. This will significantly improve energy efficiency and performance by exploiting the resources available to the application while reducing the programming effort through the automation.

In order to achieve its ambitious goals, the project has been based on two proven technologies: the system scenarios methodology as presented in this book and the static auto-tuning known from the HPC community. This chapter has presented the main concepts being used in the READEX project, as well as experimental results demonstrating the type of dynamic behavior the project will exploit.

Contemporary and future embedded systems experience continuously increasing computational capacities, e.g., through the use of heterogeneous many-core platforms. Therefore, techniques developed in the READEX project, e.g., for auto-tuning, parallel decision making, and run-time calibration, can be transferred back into this domain.

Acknowledgements The research leading to these results has received funding from the European Union's Horizon 2020 Programme under grant agreement number 671657.

References

1. S. Benkner et al., PEPPHER: efficient and productive usage of hybrid computing systems. IEEE Micro **31**(5), 28–41 (2011)
2. S. Benkner, F. Franchetti, H.M. Gerndt, J.K. Hollingsworth, Automatic application tuning for HPC architectures (Dagstuhl Seminar 13401), in *Dagstuhl Reports*, vol. 3, no. 9, pp. 214–244, 2014, http://drops.dagstuhl.de/opus/volltexte/2014/4423
3. E. César, A. Moreno, J. Sorribes, E. Luque, Modeling master/worker applications for automatic performance tuning. Parallel Comput. **32**(7), 568–589 (2006)
4. European Union FP7 project 248481, Automatic online tuning (AutoTune), http://www.autotune-project.eu/. Accessed 25 Nov 2016
5. European Union FP7 project 248647, ENabling technologies for a programmable many-CORE (ENCORE), http://cordis.europa.eu/project/rcn/94045_en.html. Accessed 26 Mar 2018
6. European Union Horizon 2020 project 671657, Run-time exploitation of application dynamism for energy-efficient exascale computing (READEX), http://www.readex.eu. Accessed 11 Feb 2019
7. I. Filippopoulos, F. Catthoor, P.G. Kjeldsberg, Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios. Des. Autom. Embed. Syst. **17**(34), 669692 (2013)
8. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, K. De Bosschere, System scenario based design of dynamic embedded systems. ACM Trans. Des. Autom. Embed. Syst. **14**(1), article 3 (2009)
9. D. Hackenberg et al., HDEEM: high definition energy efficiency monitoring, in *Energy Efficient Supercomputing Workshop, E2SC*, New Orleans, USA, 2014
10. P.G. Kjeldsberg, A. Gocht, M. Gerndt, L. Riha, J. Schuchart, U.S. Mian, READEX: Linking two ends of the computing continuum to improve energy efficiency in dynamic applications, in *Design Automation and Test in Europe Conference & Exhibition, DATE 2017*, Lausanne, Switzerland, March 2017
11. A. Knüpfer et al., Score-p: a joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir, in *Tools for High Performance Computing 2011*, ed. by H. Brunst, M. Müller, W.E. Nagel, M.M. Resch (Springer, Berlin, 2012), pp. 79–91
12. Z. Ma et al., *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms* (Springer, Dordrecht, 2007). ISBN 978-1-4020-6328-2
13. M. Merta, J. Zapletal, BEM4I, in *IT4Innovations National Supercomputing Center*, 2013, http://bem4i.it4i.cz/
14. M. Merta, J. Zapletal, J. Jaros, Many core acceleration of the boundary element method, in *High Performance Computing in Science and Engineering: Second International Conference, HPCSE 2015, Soláň, Czech Republic, May 25–28, 2015, Revised Selected Papers* (Springer, New York, 2016), pp. 116–125
15. R. Miceli et al., Autotune: a plugin-driven approach to the automatic tuning of parallel applications, in *Applied Parallel and Scientific Computing*. Lecture Notes in Computer Science, ed. by P. Manninen, P. Öster, vol. 7782, pp. 328–342 (Springer, Berlin, 2013)
16. L. Riha, M. Merta, R. Vavrik, T. Brzobohaty, A. Markopoulos, O. Meca, O. Vysoocky, T. Kozubek, V. Vondrak, A massively parallel and memory-efficient FEM toolbox with a hybrid total FETI solver with accelerator support. Int. J. High Perform. Comput. Appl. **33**(4), 660–677 (2019)
17. R. Schöne et al., Extending the functionality of score-p through plugins: interfaces and use cases, in *Tools for High Performance Computing 2016*, ed. by C. Niethammer et al. (Springer, Berlin, 2017), pp. 59–82

18. C. Silvano et al., The ANTAREX approach to autotuning and adaptivity for energy efficient HPC systems, in *Proceedings of the ACM International Conference on Computing Frontiers, CF '16* (ACM, New York, 2016), pp. 288–293
19. The OmpSs Programming Model, https://pm.bsc.es/ompss. Accessed 25 Nov 2016
20. A. Tiwari, C. Chen, J. Chame, M. Hall, J.K. Hollingsworth, A scalable auto-tuning framework for compiler optimization, in *IEEE International Parallel & Distributed Processing Symposium. IPDPS 2009*, pp. 1–12, 2009
21. J. Zapletal, M. Merta, L. Maly, Boundary element quadrature schemes for multi- and many-core architectures. Comput. Math. Appl. **74**(1), 157–173 (2016)