READEX: Linking Two Ends of the Computing Continuum to Improve Energy-efficiency in Dynamic Applications

Per Gunnar Kjeldsberg*, Andreas Gocht†, Michael Gerndt‡,
Lubomir Riha§, Joseph Schuchart¶, and Umbreen Sabir Mian†
*Department of Electronics and Telecommunications
Norwegian University of Science and Technology, Trondheim, Norway
†Center for Information Services and High Performance Computing
Technische Universität Dresden, Germany
‡Chair of Computer Architectures
Technische Universität München, Germany
§IT4Innovations, Ostrava, Czech Republic
¶High Performance Computing Center Stuttgart
Universität Stuttgart, Germany

***Abstract*—In both the embedded systems and High Performance Computing domains, energy-efficiency has become one of the main design criteria. Efficiently utilizing the resources provided in computing systems ranging from embedded systems to current petascale and future Exascale HPC systems will be a challenging task. Suboptimal designs can potentially cause large amounts of underutilized resources and wasted energy. In both domains, a promising potential for improving efficiency of scalable applications stems from the significant degree of dynamic behaviour, e.g., runtime alternation in application resource requirements and workloads. Manually detecting and leveraging this dynamism to improve performance and energy-efficiency is a tedious task that is commonly neglected by developers. However, using an automatic optimization approach, application dynamism can be analysed at design time and used to optimize system configurations at runtime.**

The European Union Horizon 2020 READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) project will develop a tools-aided auto-tuning methodology inspired by the system scenario methodology used in embedded systems. Dynamic behaviour of HPC applications will be exploited to achieve improved energy-efficiency and performance. Driven by a consortium of European experts from academia, HPC resource providers, and industry, the READEX project aims at developing the first of its kind generic framework to split design time and runtime automatic tuning while targeting heterogeneous system at the Exascale level. This paper describes plans for the project as well as early results achieved during its first year. Furthermore, it is shown how project results will be brought back into the embedded systems domain.

I. Introduction

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 the two domains are different in many respects, techniques that have proven to be efficient in one may very well be beneficial also in the other. The methodology of system scenario based design are being applied with success in embedded systems both to increase performance and to reduce power and energy consumption [1]–[3]. It consists of a split design time and runtime approach to enable dynamic adaption to changing system requirements without a prohibitively large runtime overhead. In the HPC domain auto-tuning is used either at design time to statically tune the system configuration, or at runtime through compute intensive estimation of the dynamic requirements [4], [5]. Combining the approaches from embedded systems and HPC has potential of giving substantial synergies in both domains.

A constantly growing demand for data centre computing performance leads to the installation of increasingly powerful and ever more complex systems characterized by a rising number of CPU cores as well as increasing heterogeneity. This makes optimization of HPC applications a complex task demanding severe programming effort and high levels of expertise. With 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. Limitations to chip temperature and cooling capabilities can furthermore make the performance of Exascale HPC systems power-bound. Developers, however, commonly focus on the implementation and improvement of algorithms with regards to accuracy and performance, neglecting possible improvements to energy-efficiency. The fact that programmers in general also lack the platform and hardware knowledge required to exploit these measures is an important obstacle for their use both in the embedded and HPC domains.

The European Union READEX Project [6] (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) tackles the challenges above by embracing the significant potential for improvements to performance and energy-efficiency stemming from the fact that HPC applications commonly exhibit dynamic resource requirements similar to those seen in embedded systems. Examples are alternating application regions or load-changes at application runtime. We will refer to this as application *dynamism* throughout the

paper. Examples of dynamism can be found in many current HPC applications, including weather forecasting, molecular dynamics, or adaptive mesh-refinement applications.

These applications commonly operate in an iterative manner, e.g., using a timestep loop as the main control flow. Each iteration of such a program loop can be regarded as a phase of the application execution. 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. Inter-phase dynamism, on the other hand, describes the changes in application behaviour 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.

It is expected that applications running on future systems, both extreme-scale HPC and embedded, will exhibit even higher levels of dynamism. This will be mainly due to the increased demand on data movement between processing elements, both on intra- and inter-node levels, and more complex levels of the memory hierarchy. Furthermore, the rise of many-core co-processors and accelerators are introducing new degrees of freedom such as offloading and scheduling.

The READEX project will develop and implement a tools-aided methodology that enables HPC application developers to exploit dynamic application behaviour 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. This will be further outlined later in this paper.

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

The rest of this paper is organized as follows. After a review of related work in Section II, the project background in systems scenario based design and static auto-tuning is given in Section III. This is followed by a description of the READEX concepts (Section IV) as well as results from experiments with a realistic industrial HPC application (Section V). Finally, we show how results from READEX can be applied in the embedded systems domain in Section VI and conclude in Section VII.

II. Related Work

System scenario based design will be described in the next section. Other related work in the embedded systems domain is out of the scope of this paper, but a good overview of different approaches for scenario based dynamic system adaptation can be found in [7].

While a small number of dynamic auto-tuning methodologies and tools exist for run-time optimizations [8], [9], no single standalone dynamic auto-tuning framework currently exists 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.

Several leading 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 approach is the ENabling technologies for a programmable many-CORE (ENCORE) project [10], which aims to achieve massive parallelism relying on tasks and efficient task scheduling using the OmpSs programming model [11]. The READEX project takes a different approach by developing a new generic programming paradigm allowing 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 [12] 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 of the READEX project, we will go one step further and provide a framework that supports the heterogeneity of the system in the form of tuning parameters that allows large-scale heterogeneous applications to dynamically (and automatically) adapt heterogeneous resources according to runtime requirements.

The ANTAREX project [13] 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.

To the best of our knowledge, no dynamic auto-tuning framework exists yet that shows the potential to scale to future Exaflop machines. Auto-tuning frameworks typically follow a centralized approach, where the central agent will become a bottleneck when deployed on these systems. In contrast, READEX aims to develop the concept of (semi)-distributed dynamic tuning that minimises centralisation of control.

III. Background

READEX synergistically combines and extends two technologies from opposite ends of the computing continuum, namely the systems scenario methodology for dynamic tuning from the field of embedded systems with the automatic static tuning from the area of HPC.

A. System Scenarios Methodology

In the embedded systems domain a scenario-based methodology [1]–[3] has been developed to enable exploitation of application dynamism through fine-grained system tuning. The methodology exploits detailed knowledge about the application(s) to be run on the system, extracted through profiling and (semi-)automatic code inspection at design time. Using these techniques, different runtime situations (rts's) of the application are identified that have different costs related to them, e.g., execution time, energy consumption, and memory footprint. At the same time, identifiers that decide the upcoming rts, e.g., data and control variables, are determined.

At design time, rts's are grouped into scenarios with similar multidimensional system costs. Optimized platform configurations are generated for each scenario using parameters such as dynamic voltage and frequency scaling, task mapping, and data and memory reconfiguration. Furthermore, efficient and possibly application-specific scenario prediction and scenario switching mechanisms are developed. The prediction is based on the identifiers that decide the upcoming rts and hence the corresponding scenario. The switching mechanism compares the cost and the gain of reconfiguration and decides whether it is beneficial to switch. It also defines how the actual reconfiguration shall be performed, e.g., how to change from one voltage and frequency setting to another.

At runtime, the upcoming scenario is predicted and a platform configuration switch is performed based on the mechanisms developed at design time. Should the application experience an rts that was not seen at design time a backup scenario guaranteed to satisfy any rts is used. The methodology also opens for the inclusion of a calibration mechanism but this is typically too resource demanding for smaller embedded systems. Examples of more than 30 % reduction of energy consumption have been reported for the system scenarios methodology in the embedded systems field [2].

B. Static Auto-tuning of HPC Applications

Most of the current tools for performance engineering focus on collecting and presenting information for the users, while only few focus on the automation of the performance optimization process (auto-tuning), e.g., the Periscope Tuning Framework (PTF) developed in the EU FP7 ICT AutoTune project [4], [5]. PTF automatically finds optimized system configurations for whole application runs, effectively averaging the benefits of system adaptation over the whole runtime of the application (static tuning). With these static auto-tuning techniques, improvements in energy-efficiency of up to 10 % for application runs have already been achieved while keeping the performance degradation to a few percent [14].

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 runtime environment settings
- Dynamic Concurrency Throttling (DCT) in OpenMP

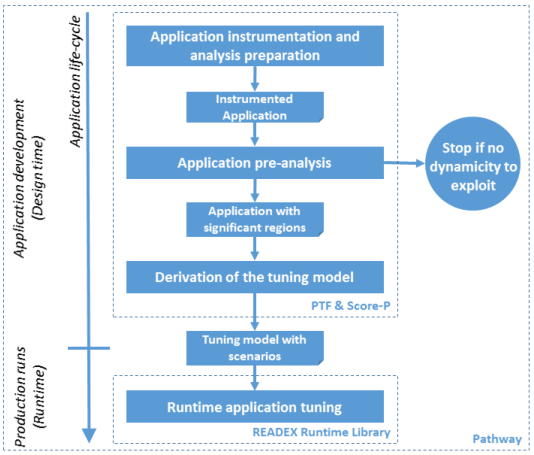

Fig. 1. Overview of the READEX methodology.

- MPI master-worker pattern settings

PTF also provides the Tuning Plugin Interface for the development of new plugins. It builds on the common performance measurement tools infrastructure Score-P [15], which has proven scalability on current petascale systems.

IV. The READEX Concept

The READEX concept combines the system scenarios methodology with automatic energy and performance tuning into a holistic tools-aided methodology, spanning over the major parts of the HPC application life-cycle, i.e., application development and performance tuning at design time and production runs at runtime. Figure 1 provides a high-level overview of the methodology.

A. Application Instrumentation and Analysis Preparation

During the first step of the methodology, the application is instrumented through insertion of 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 is both done automatically and by letting the user provide application-domain knowledge using a new programming paradigm being developed in the project.

The programming paradigm provides a possibility for exposing parameters that define dynamic behaviour of the application to the READEX tool-suite. These application domain parameters, referred to as additional identifiers, will enhance the distinguishing of different system scenarios and correspondingly adapting system configurations in order to improve overall application performance and energy characteristics.

An example for exposed parameters are different input data sets. Exposing these data to READEX will allow detecting different compute characteristics caused by varying input. The paradigm will also allow 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.

B. Application Pre-analysis

Based on the performance dynamics analysis capabilities of PTF, the READEX analysis strategy is being developed. It will automatically characterize present dynamism and indicate 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.

The application is run once with a representative data set, during which relevant timing information is recorded. This performance data will reveal all significant regions of the application with runtimes above a certain threshold. The instrumentation of insignificant regions with runtime 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 applications phases. According to the system scenario methodology, each region's execution, represented by a point in this space, corresponds to a runtime situation (rts). A first analysis will reveal whether relevant code regions exist that exhibit enough dynamism to make the following tuning steps worthwhile. The dynamism can, e.g., be differences in compute- and memory-bound operation. If no such dynamism is detected the tuning should be aborted due to homogeneous application behaviour. Since the runtime tuning necessarily cause an overhead, a threshold will be defined for the required dynamism.

C. 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 Runtime Library are used to perform an automatic search for optimal system configurations for the significant regions identified in the previous step.

Exploration of the space of possible tuning configurations is controlled by PTF tuning plugins. Each tuning plugin is responsible for a specific tuning aspect, which can include one or multiple related tuning parameters. READEX will support and develop 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 Figure 2. To facilitate the determination of optimal platform configurations, PTF will run the instrumented application. For the identified rts's, various system configurations are evaluated in terms of the requested objective functions and the results are stored in the RTS Database. The READEX Runtime Library (RRL) is used to configure the platform according to each experiment. Details regarding the RRL will be given in Section IV-D.

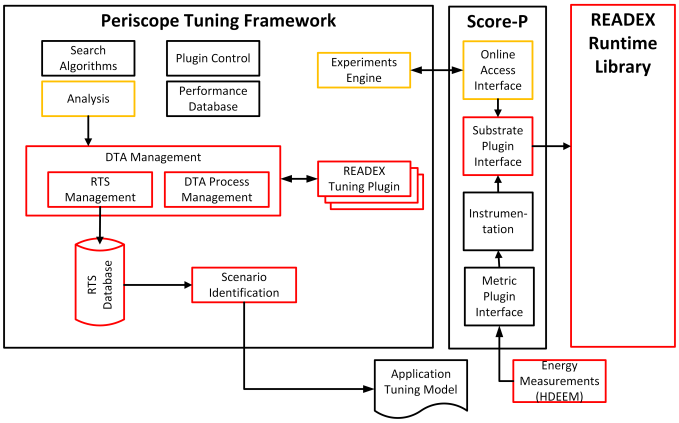

Fig. 2. Architecture of the design time part of the READEX Tool Suite (black: existing component; red: new ; yellow: extended).

Since the search space for optimal configurations is potentially large, new search strategies are being developed. A number of possible search strategies to explore have been identified, e.g., heuristics based parameter selection, interphase comparisons with an underlying approximation of the expected objective function, and a simple comparison with the objective function values taken during a baseline run.

After all relevant system configurations are evaluated for all rts's, a scenario identification module groups rts's with identical or similar best found configuration into scenarios. In order to predict upcoming system scenarios at runtime, it also builds a classifier based on the provided identifiers. The classifier maps an rts to a scenario based on the current identifier values, e.g., 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, e.g., directly be used to select the voltage and frequency setting to apply. More advanced selectors can also be envisioned, taking switching overhead into account as well as selection between Pareto optimal configurations. The set of scenarios, the classifier and the selector is stored in the application tuning model, in the form of a serialised text file, to be loaded and exploited during production runs at runtime.

D. Runtime Application Tuning

Once the pre-analysis is finished and the application tuning model is created, the optimized application can be used in production. For this, the READEX Run-time Library (RRL) is implemented. This library uses the previously obtained knowledge about application dynamism to optimize the application's energy consumption. For the already seen rts's, the optimal configuration is directly extracted from the tuning model. For the unseen rts's a calibration mechanism will be developed.

The architecture of the RRL is depicted in Figure 3. Note that even though included in the figure, PTF is not used during a production run. As mentioned in Section IV-C the RRL is used at design time to set the system configuration during the search for optimal configurations. At runtime, the RRL obtains the set of configurations, in the form of scenarios, classifiers

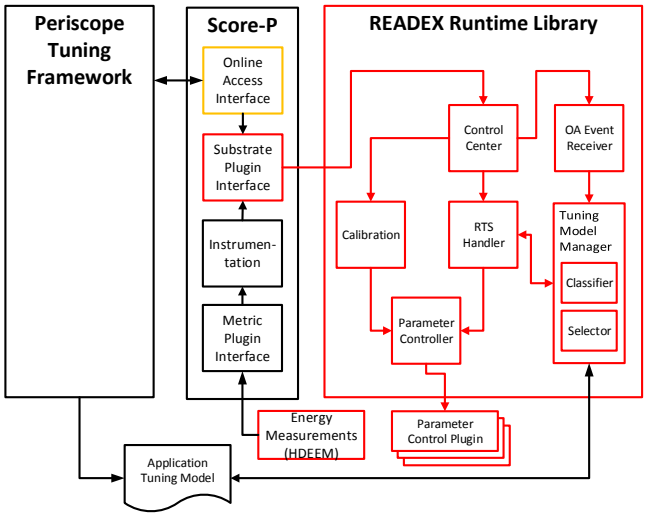

Fig. 3. Architecture of the READEX Run-time Library (RRL) (black: existing component; red: new ; yellow: extended).

and selectors, from the application tuning model generated by PTF at design time.

A production run of an application starts with a loading of the application tuning model into the Tuning Model Manager (TMM). When a new region is entered during the application execution, Score-P will notify 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 of OpenMP threads. They are employed both during the derivation of the tuning model at design time and during a production run.

In order to qualify for Exascale deployments, the architecture of the RRL relies on maximal decentralization of tuning decision making. If it is unavoidable, a scalable tree-like reduction network will be implemented to determine global decisions.

During the application run, a calibration mechanism will handle unseen rts's and changes to the environment. The calibration will be based on established Machine Learning approaches to be investigated in later phases of the project. Once the calibration detects a new optimal configuration for a certain scenario, the tuning model will be updated. This will lead to a constantly improving tuning model.

V. 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 the manual tuning.

For evaluation and validation of the project results, READEX employs a co-design process in which the auto-tuning methodology and the tool-suite are developed in parallel with a manual tuning effort of selected applications and computational libraries. The information exchange will benefit both the creation of the tools-aided methodology and the manual tuning efforts.

Since the READEX project is still in an early phase, this section presents a purely manual application case study. The goal is to show how application dynamism can be exploited to improve energy efficiency. The evaluation has been done on a 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. Each CPU has an L3 cache while each core has local L2 and L1 caches. These are used transparently. The tests are performed on a single compute node, and overall interconnect hence does not influence the results. 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) [16].

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 Finite Element Tearing and Interconnecting (FETI) method forms a subclass of DDM, efficiently blending Conjugate Gradient (CG) iterative solvers and direct solvers.

The FETI method exhibits both parallel and numerical scalability and can scale to tens of thousands of CPU cores as implemented in the ESPRESO library, which employs both MPI and OpenMP parallelization [17]. The OpenMP parallelization enables the resource manager to dynamically change number of threads and therefore number of utilized CPU cores for different code regions of the application.

The CG solver in FETI calls three key operations: (i) FETI operator F, (ii) preconditioner Pr, and (iii) projector P (for more detail see [17]) in every CG iteration. These three operations takes over 95% of the solver runtime and each of them uses different BLAS functions. In our experiments, we have dynamically modified both the number of OpenMP threads (from 1 to 12 per CPU socket) and frequency settings (from 1.2 to 2.5 GHz). The supply voltage is automatically set to the lowest level allowed for a given frequency. The results are shown in Table I. The *best static savings* value shows the best combinations of found frequencies and number of cores for the entire CG solver without dynamic tuning of the F, Pr and P regions. The energy saving is given as a percentage compared to using the default frequency 2.5 GHz and default number of threads 24. The *dynamic savings* values in the table show additional savings when we dynamically switch the frequency and number of threads for F, Pr and P regions.

TABLE I. OPTIMAL FREQUENCIES AND ENERGY SAVING FOR ESPRESO FETI SOLVER.

HW configuration	Phase	Opt. freq. / # of cores	Energy saving
Default:	CG solver	2.4 GHz / 12 cores	—
Best static:	CG solver	1.8 GHz / 10 cores	9 %
Dynamic tuning:	F region	1.2 GHz / 12 cores	
	P region	2.4 GHz / 12 cores	
	Pr region	1.8 GHz / 10 cores	
	all regions		+ 6 % = 15 % total savings

The table shows that the best static configuration provides 9 % energy savings, dynamic tuning adds extra 6 %, which in total provides 15 % energy savings. This is exactly the type of dynamic behaviour that READEX plans to exploit. Note that in this case we only experiment with two tuning parameters (DVFS and number of utilized cores) while later in the project we will work with additional types of platform configuration alternatives to further reduce the energy consumption.

VI. APPLICATION IN EMBEDDED SYSTEMS

In system scenario based design of embedded systems, the process of searching for optimal platform configurations are today a partly manual task based on code inspection and profiling. Important elements from the design time auto-tuning process can be adapted to embedded systems. Both the automatic code instrumentation and use of the new programming paradigm for application domain experts can be transferred. The concept of plugins that tune for specific tuning aspects, as well as the novel search strategies, can also be used as inspiration for future projects in the embedded domain.

For an average embedded system, the RRL will impose a too large overhead. In this case, a lightweight version compiled directly into the application at design time is more reasonable. This can be used to extend current work on scenario platform adaption units [19]. For high-end multi-processor systems-on-chip, an RRL-like solution, including the calibration mechanism, may also be explored.

VII. CONCLUSION

Energy-efficiency and extreme parallelism are the major challenges on the road to Exascale computing. The European Union Horizon 2020 READEX project will address these by providing application developers with a tools-aided methodology for a combined design time runtime approach for dynamic adaption to changing resource requirements. The aim is to 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 builds on two proven technologies, the static auto-tuning and the system scenarios methodology, to develop the first of its kind generic dynamic auto-tuner. This paper has presented the main concepts being used in the READEX project, as well as experimental results demonstrating the type of dynamic behaviour the project will exploit.

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

ACKNOWLEDGMENT

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

REFERENCES

[1] I. Filippopoulos, F. Catthoor, and P. G. Kjeldsberg, "Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios," *Design Automation for Embedded Systems*, vol. 17, no. 3-4, pp. 669–692, 2013.

[2] S. V. Gheorghita et al., "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 1, pp. 3:1–3:45, 2009.

[3] Z. Ma et al., *Systematic Methodology for Real-time Cost-effective Mapping of Dynamic Concurrent Task-based Systems on heterogenous Platforms*. Springer Science & Business Media, 2007.

[4] S. Benkner et al., "Automatic Application Tuning for HPC Architectures (Dagstuhl Seminar 13401)," *Dagstuhl Reports*, vol. 3, no. 9, pp. 214–244, 2014. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2014/4423

[5] 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, Eds. P. Manninen and P. Öster, Springer Berlin Heidelberg, 2013, vol. 7782, pp. 328–342.

[6] Run-time Exploitation of Application Dynamism for Energy-efficient Exascale computing (READEX), last accessed November 25, 2016. [Online]. Available: http://www.readex.eu

[7] W. Quan and A. D. Pimental, "Scenario-based run-time adaptive mpsoc systems," *Journal of Systems Architecture*, vol. 62, pp. 12–23, 2016.

[8] E. César et al., "Modeling Master/Worker applications for automatic performance tuning," *Parallel Computing*, vol. 32, no. 7, pp. 568–589, 2006.

[9] A. Tiwari et al., "A Scalable Auto-Tuning Framework for Compiler Optimization," *IEEE International Parallel & Distributed Processing Symposium. IPDPS 2009*, pp. 1–12.

[10] ENabling technologies for a programmable many-CORE (ENCORE), last accessed November 25, 2016. [Online]. Available: http://cordis.europa.eu/project/rcn/94045_en.html

[11] The OmpSs Programming Model, last accessed November 25, 2016. [Online]. Available: https://pm.bsc.es/ompss

[12] S. Benkner et al. "PEPPHER: Efficient and productive usage of hybrid computing systems," *IEEE Micro*, vol. 31, no. 5, pp. 28–41, 2011.

[13] 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*. New York, NY, USA: ACM, 2016, pp. 288–293.

[14] "Automatic Online Tuning (AutoTune), http://www.autotune-project.eu/," last accessed November 25, 2016. [Online]. Available: http://www.autotune-project.eu/

[15] 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*, H. Brunst, M. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin: Springer, 2012, pp. 79–91.

[16] D. Hackenberg et al., "HDEEM: High Definition Energy Efficiency Monitoring," in *Energy Efficient Supercomputing Workshop, E2SC 2014*.

[17] L. Riha et al., "Massively parallel hybrid total feti (htfeti) solver," *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '16*. New York, NY, USA: ACM, 2016.

[18] V. Hapla et al., *Solving Contact Mechanics Problems with PERMON*. Springer International Publishing, 2016.

[19] Y. Yassin et al., "Dynamic hardware management of the h264/avc encoder control structure using a framework for system scenarios," in *The 19th Euromicro Conference on Digital System Design, DSD 2016*.