October 2017

Design-time Analysis for the READEX Tool Suite[1]

Madhura KUMARASWAMY, Anamika CHOWDHURY and Michael GERNDT
Technische Universität München
Institut für Informatik
85748 Garching, Germany
E-mail: {kumarasw, chowdhua, gerndt}@in.tum.de

Abstract. Energy efficiency and consumption are now the most important and challenging issues in current Petascale and in designing future Exascale computing systems. The European Union Horizon 2020 READEX project uses an online approach to exploit application dynamism and tune large-scale HPC applications to improve energy efficiency and performance. The paper presents the READEX methodology, consisting of the Design-Time Analysis and Runtime Application Tuning, and describes the pre-analysis steps involving application dynamism and significant region detection. During design-time, the READEX tuning plugin evaluates configurations of hardware and software tuning parameters to determine the best settings for instances of application regions. The runtime tuning dynamically switches to the best configuration for an application region during production runs. Finally, the energy savings obtained for LULESH on the Taurus supercomputer highlight the effectiveness of this methodology.

1. Introduction

High Performance Computing requires significant electrical energy, and energy efficiency remains a key challenge in today's HPC landscape. As developers typically lack the platform and hardware knowledge, and tools to influence the energy consumption, improvements to the energy efficiency of applications have been rarely targeted. To overcome this challenge, READEX aims to deliver the first standalone auto-tuning framework that dynamically tunes large-scale HPC applications at design- and runtime for Exascale computing.

READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) is a European Union Horizon 2020 project and extends previous works that only statically tune the energy consumption by setting tuning parameters for the entire application run [1]. Instead, in READEX, the tuning parameter switching occurs dynamically during the application execution. READEX leverages dynamically varying application characteristics, such as compute intensity and algorithm granularity. READEX targets applications that exhibit iterative behavior, typically in the form of a main progress loop, called a *phase region*, whose individual time steps are called *phases*.

[1]The final publication is available at IOS Press through http://dx.doi.org/10.3233/978-1-61499-843-3-307

We extend an approach developed for tuning embedded systems, called *scenario based tuning* to the HPC environment by grouping *runtime situations*, which are executions of program regions, into *scenarios* if they have similar characteristics. Finally, the best tuning parameter setting for each scenario is determined.

READEX explores tuning parameters that have the potential to influence the energy consumption of an HPC application, such as the CPU frequency, which can be lowered or raised for compute or memory bound applications during Dynamic Voltage and Frequency Scaling (DVFS). READEX leverages the features of the Intel Haswell processor family, which allow the userspace governor to set the frequency for individual cores as opposed to full sockets as seen in previous processor lines. Unlike other related works, READEX also explores the uncore frequency, which can be controlled by setting machine specific registers (MSRs). READEX also leverages the domain-knowledge of the application developer, who can specify application-level tuning parameters, such as the types of solvers, access strides, or number of subdomains for exploiting the dynamicity.

The READEX tool suite is a two-staged approach, consisting of Design-Time Analysis (DTA) and Runtime Application Tuning (RAT). First, the *readex_dyn_detect* tool (Section 3.1.2) determines *significant regions*, a subset of all instrumented regions that are coarse-granular enough for dynamic setting of the tuning parameters. The READEX Tuning Plugin (Section 4.2) then determines the best settings of the tuning parameters for runtime situations of the significant regions. Runtime situations that have similar characteristics are grouped into scenarios, and a *selector* chooses a best *configuration*, or a tuning parameter setting for each scenario. This knowledge is encapsulated in a *tuning model* at the end of DTA and is forwarded to the runtime tuning.

During runtime, the READEX Runtime Library (RRL) uses the tuning model to switch between system configurations that are best for upcoming runtime situations. RRL is integrated into Score-P via the so-called *Score-P Substrate Plugin Interface*. The monitoring and measurements are done via Score-P [2,3]. Energy measurements are provided on the Taurus supercomputer at the ZIH in Dresden via the HDEEM measurement hardware [4], which allows processor as well as blade energy measurements.

This paper presents in detail the preparatory steps in Section 3, and the Design-Time Analysis in Section 4. It also provides an overview of the tuning model generation in Section 4.3 and the Runtime Application Tuning in Section 5. The static and dynamic energy savings obtained after DTA for LULESH [5], a proxy application from the CORAL benchmark suite, are discussed in Section 6. An evaluation of results obtained for different node configurations for the exhaustive search strategy is also presented. In the end, the paper will draw the concluding remarks.

2. Related Works

Currently, there are various approaches that employ DVFS tuning in HPC to improve the energy-efficiency. The AutoTune Project [6] implemented a DVFS tuning plugin that interfaces with the *enopt* library to change the core frequency and governor for different application regions, as well as predict the frequency for the minimum energy consumption. Guillen et al. [7] developed a DVFS tuning plugin to set the CPU frequency based on a model that predicts energy consumption for different CPU frequencies.

Laros et al. [8] showed the effect of both CPU frequency and network bandwidth scaling on the Cray XT architecture to achieve improved energy consumption with no

performance degradation. Etinski et al. [9] present a model that gives a risk factor value of performance loss due to frequency scaling based on application/performance characteristics. Kimura et al. [10] developed an offline approach to hook DVFS directives into the instrumented code for better energy savings. While previous research works are based on static tuning, READEX configures different tuning knobs (hardware, software, application parameters) dynamically for program regions based on dynamic application characteristics. It also applies a holistic approach by exploring additional new hardware parameters in combination with system and application-level tuning parameters.

The ANTAREX project [11] uses a Domain Specific Language (DSL) approach to specify mapping and adaptivity strategies of the application at runtime. The DSL template interacts with the runtime resource manager to configure software knobs for the application regions. The project focuses only on ARM-based multi-cores and GPGPUs, while READEX targets all HPC systems.

3. Preparatory Steps

The READEX tool suite is based on the instrumentation of the application with Score-P. Instrumentation inserts measurement probes into the application source code. Section 3.1.1 describes how the *scorep-autofilter* tool helps to reduce the measurement overhead, after which, the application is analyzed to determine if there is tuning potential that can be exploited by the READEX methodology (Section 3.1.2).

3.1. Dynamism Detection

The READEX tool suite computes an application's tuning potential for two aspects: *intra-phase dynamism*, resulting from variations due to the execution of different algorithms or control flow within a single phase, and *inter-phase dynamism* due to changing behavior between phases. First, performance data is collected by Score-P [2] for the instrumented application regions and stored in a `profile.cubex` file in the *cube* format [12] as a call-tree. Next, coarse-granular instrumented regions are identified, and the dynamism is computed using the execution time and compute intensity for those regions.

3.1.1. The scorep-autofilter Tool

Automatic compiler instrumentation with Score-P causes a high overhead due to the instrumentation of frequently executed fine granular regions. To reduce their impact, a filter file containing the names of too fine-granular regions that should be omitted from measurement collection may be provided to Score-P. The so-called filter file is generated by the `scorep-autofilter` tool, which accesses the profile data and computes the *granularity*[2] for application regions. If the granularity for an application region is lower than a given threshold and it is neither an MPI nor an OpenMP parallel region, it is added to the Score-P filter file.

[2] Average execution time of the instances of a region

Table 1. Statistics for the significant regions produced by *readex-dyn-detect* for the BT-MZ benchmark

Region name	Min (t)	Max (t)	Exec time	Time Dev. (%Reg)	Ops/L3miss	Weight (%Phase)
compute_rhs	0.0	0.001	23.839	0.0	8	10
exch_qbc	0.0	0.037	10.802	0.0	4	4
x_solve	0.0	0.005	61.448	0.0	6664	27
y_solve	0.0	0.006	62.739	0.0	1328	28
z_solve	0.0	0.005	65.676	0.0	522	29

3.1.2. The readex-dyn-detect Tool

`readex-dyn-detect` executes a selection algorithm to generate a list of coarse-granular regions, called significant regions. The tool also detects the dynamism by computing the tuning potential for the detected significant regions.

Significant Region Detection

`readex-dyn-detect` specifies three constraints for the significant region selection algorithm. First, the candidate region must be coarse enough to overcome the switching overhead. Next, significant regions cannot be nested. Finally, the selected regions should cover a significant amount of the application execution time.

First, the tool analyzes all program regions that are called inside a phase region, and marks them as candidates for the significant region analysis. To satisfy the second constraint, the algorithm marks only non-nested and non-recursive regions, and removes all the cycles from the call graph. It then iterates over the candidate regions in the call graph and considers only the leaf nodes for the selection algorithm, since they cover a significant amount of the application execution time. Then, a leaf node is selected as a significant region if its execution time is more than the summed up exclusive execution times of its parents. Otherwise, the parents are selected as significant regions during the next iterations when they become leaf nodes. To satisfy the second constraint, the current region and its predecessors are removed from the candidate list. After all the leaf nodes were visited, dynamism analysis is performed for the selected significant regions.

Table 1 lists the significant regions detected for the BT-MZ benchmark from the NAS Parallel Benchmarks suite, along with some statistical information for each significant region. The statistics are based on the execution time and compute intensity. In Table 1, columns 2, 3 and 4 are the minimum, maximum and absolute values of the execution time for each significant region. Column 5 shows the standard deviation relative to the mean execution time of the significant region. Column 6 represents the absolute value for compute intensity, and column 7 shows the weight of each significant region with respect to the phase region.

Dynamism Analysis

After detecting the significant regions, `readex-dyn-detect` computes intra-phase and inter-phase dynamism from the statistical information shown in Table 1. The tool detects dynamism for variations in both the execution time and compute intensity. For this, the user may specify thresholds for the variation in the execution time v_t of the instances of the significant regions, the minimal standard deviation of the compute intensity v_i, and the minimal weight of the execution time of a significant region with respect to the phase

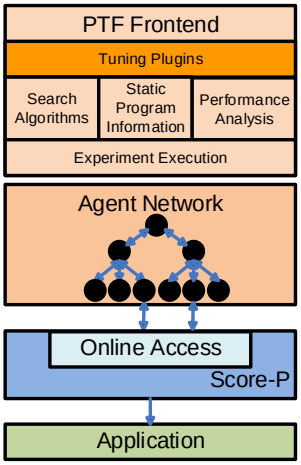


Figure 1. Components of PTF and their interaction with Score-P.

region v_w. For BT-MZ, a threshold of 10% for v_t, v_i and v_w was considered. To qualify for dynamism analysis, each significant region's weight must be above v_w.

If v_t for a significant region is above the specified threshold, `readex-dyn-detect` reports that there is intra-phase dynamism due to variation in the execution time. If the deviation of the compute intensity across significant regions is above v_i, there is intra-phase dynamism due to variation in the compute intensity. For BT-MZ, `readex-dyn-detect` reports that there is intra-phase dynamism due to variation in the compute intensity as the variation is above 10%, but no intra-phase dynamism due to the variation in the execution time, as it is below 10% for all the significant regions.

Finally, if there is dynamism due to variation in the minimum and maximum execution time for the phase region, and the deviation is larger than v_t , `readex-dyn-detect` reports inter-phase dynamism. The dynamism results for each significant region are exported to a general configuration file called the READEX configuration file in the *.xml* format. The generated file contains the tuning potential summary, the list of significant regions, and the intra-phase and inter-phase dynamism for variation in the execution time and compute intensity per significant region. The user may then insert the tuning parameters, the objectives and the search algorithm into the file, which will be used by the READEX tuning plugin to tune significant regions (Section 4.2).

4. Design-Time Analysis

4.1. Overview of the Periscope Tuning Framework (PTF)

The Periscope Tuning Framework (PTF) [6] performs DTA, which is the first stage of the READEX methodology. It was developed at Technische Universität München, Garching, Germany, and is a distributed framework consisting of the frontend, the tuning plugins, the experiment execution engine and a hierarchy of analysis agents. The PTF agents connect to the application via monitors that can send measurement requests and receive measurement results to and from Score-P via the Online Access interface. Figure 1 shows

the overall architecture of PTF and the interaction between PTF and Score-P. PTF frontend executes a tuning plugin to tune an application for a given objective and applies an online search by running experiments with different configurations within a single program run. Each experiment is an individual execution of the phase region in which, performance properties collected for the application's run enable the plugin to determine the best setting of the tuning parameters.

4.2. READEX Tuning Plugin

PTF was extended with the READEX tuning plugin to perform DTA. The READEX tuning plugin is a modified Dynamic Voltage Frequency Scaling (DVFS) plugin that determines the best configuration for runtime situations of the significant regions by switching specific tuning parameters. The following steps describe the sequence of actions performed by the plugin:

1. Initialization: First, the plugin reads and initializes the ranges (minimum, maximum and the step size) of the tuning parameters that are provided in the READEX configuration file. The READEX tuning plugin currently supports three tuning parameters: CPU frequency, uncore frequency and the number of OpenMP threads. It then reads the search algorithm. If no search strategy is specified, exhaustive search is used. The plugin then reads the objective to tune the application for. If no objective is specified, the default objective is set to Energy, which is the energy consumption of the entire node.
2. Experiment Creation: The plugin uses the search algorithm to create a search space, which is the cross-product of the tuning parameters. The exhaustive search strategy creates a search space of the cross-product of the CPU frequency, uncore frequency, and OpenMP threads tuning parameters. Then, it creates individual experiments for combinations of runtime situations of the significant regions and the tuning parameter settings.
3. Experiment Execution: The plugin then executes an experiment and requests the node energy, the CPU energy and the execution time from Score-P for each runtime situation in the form of a tuning request. Each analysis agent stores the measurements returned by Score-P for those MPI processes controlled by it, which are then gathered for all the runtime situations in the frontend.
   To compute the consumed energy per experiment, the plugin aggregates the values returned by the designated processes of all the nodes for an MPI application. Since energy is a global metric of a node, its value is only returned to Score-P for a designated single process in each node.
4. Process Results: When there are no more experiments left to be executed, the plugin determines the best setting of the tuning parameters for the selected objective. It outputs the optimal configuration for both the phase and the runtime situations of the significant regions by selecting the tuning parameter settings for the least energy consumption value. The static energy saving for the phase is computed as the improvement in the energy consumption for the best setting over the default setting of the tuning parameters. The static energy saving for each runtime situation is the reduction in the energy consumed by it for the best setting for the phase over the energy consumed for the default setting for that runtime situation. The dynamic savings are computed as the improvement in the energy consumption

for the best configuration for a runtime situation over the energy consumption for the best static configuration.
Finally, the plugin creates three additional experiments, each of which sets the tuning parameters to the best configuration for the phase and the individual runtime situations. The experiments are used to determine the dynamic energy saving obtained by taking the dynamic switching overhead into account, as well as verify the reproducibility of the measured objective values.

4.2.1. Variations Supported by the READEX Tuning Plugin

The READEX tuning plugin also supports other search strategies and objective functions, as described below:

1. Objective functions: The READEX plugin allows the user to specify different objective functions to be measured in addition to Energy, such as Execution Time, CPU Energy, Energy Delay Product and Energy Delay Product Squared.
2. Multiple search algorithms: The READEX tuning plugin supports search algorithms other than the default exhaustive search, such as the individual, random and genetic search algorithms. Each of these search algorithms generates the search space differently.
   For the individual search strategy, the plugin explores the tuning parameters individually assuming independence. Before switching to the next parameter, the found best setting for the previous parameter is fixed. The user can also specify the *keep* factor, which is the number of best values of the already evaluated tuning parameter to keep for the next search step [6].
   If the search space becomes large due to the cross-product of the tuning parameters, the random search strategy can be specified, along with the number of *samples* (experiments) that should be performed. This prevents the search space from exploding. The algorithm may pick a tuning parameter value using a uniform probability distribution or a user-specified probability distribution [6].
   The GDE3 (Generalized Differential Evolution 3) genetic search algorithm can be used to specify multiple tuning objectives. This results in a pareto-curve, and the algorithm returns a set of best configurations depending on the tradeoff between the objectives. The user may specify the population size to limit the number of experiments in the next generations, the maximum number of generations, and the timer to set an upper limit on the tuning execution time.

4.3. Tuning Model Generation

After the READEX tuning plugin finishes, runtime situations that have the same best configurations are clustered into a scenario, and best configurations for those scenarios are selected. The knowledge obtained during DTA, such as the best configurations for individual scenarios is encapsulated in a tuning model. For production runs, this tuning model is forwarded to the READEX Runtime Library (RRL).

5. Runtime Application Tuning

The second stage of the READEX methodology is the Runtime Application Tuning, which is performed by RRL. During the production run of an application, the event han-

Table 2. Results of DTA for LULESH using the READEX tuning plugin for 8 nodes and 64 tasks.

Significant regions	Energy for default configuration (J)	Energy for best static configuration (J) (1.8, 1.2)	READEX energy (J)	CPU frequency (GHz)	Uncore frequency (GHz)
ApplyMaterialPropertiesForElems	216.33	229.13	209.64	2.4	2.8
CalcFBHourglassForceForElems	293.85	256.20	251.34	2.2	1.2
CalcKinematicsForElems	338.70	314.31	300.32	2.2	1.4
CalcMonotonicQForElems	205.18	201.67	184.49	1.8	2.2
CalcMonotonicQGradientsForElems	181.01	190.49	169.34	2.4	1.2
IntegrateStressForElems	250.70	274.41	233.54	2.4	1.6
SUM	**1485.77**	**1466.21**	**1348.67**		
Energy consumed by Phase (J)	6592.62	5685.62			

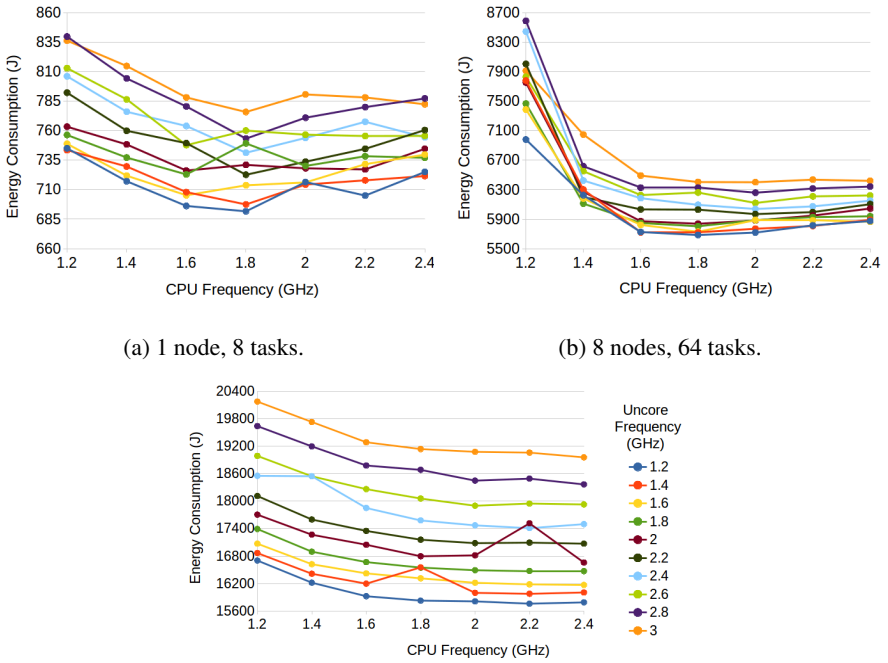
dler uses the tuning model generated at the end of DTA to look for the best configuration upon encountering a runtime situation. It then switches to this configuration if there is significant expected saving in the energy consumption over the switching overhead. For switching, RRL employs separate *parameter control plugins* to control the setting of each tuning parameter. RRL may also apply a calibration mechanism at runtime to handle unseen runtime situations that may pop up when there are changes in the control flow between the design-time and the runtime.

6. Evaluation

This section presents the results obtained from `readex-dyn-detect` and DTA for the Livermore Unstructured Lagrange Explicit Shock Hydro (LULESH) [13] proxy application from the CORAL benchmark suite. LULESH is used for hydrodynamics modeling to simulate the interaction of materials when subject to forces by solving a Sedov blast problem. The simulation is run by applying a time stepping algorithm followed by calculating the time constraint to limit how far the simulation advances. The hybrid MPI+OpenMP version of LULESH 2.0 [5] was executed with problem size 80 (grid size 80x80x80) and 64 MPI processes on 8 nodes of the Taurus HPC system. Each node on Taurus contains two 12-core Intel Xeon CPUs E5-2680 v3 (Intel Haswell family) running with a default CPU frequency of 2.50 GHz.

After applying `readex-dyn-detect`, the significant regions that were obtained for LULESH are: `ApplyMaterialPropertiesForElems`, `CalcFBHourglassForceForElems`, `CalcKinematicsForElems`, `CalcMonotonicQForElems`, `CalcMonotonicQGradientsForElems` and `IntegrateStressForElems`. These regions are communicated through the READEX configuration file to the READEX tuning plugin.

The plugin executed a total of 70 experiments using the exhaustive search strategy and returned the best CPU frequency and uncore frequency setting for each significant region, as shown in Table 2. Column 2 shows the energy consumption (node energy) in Joules for the significant regions for the default setting of the CPU frequency at 2.5 GHz. Column 3 shows the total energy consumed by the significant regions for the best static configuration for the phase, which is a CPU frequency of 1.8 GHz and an uncore frequency of 1.2 GHz. Column 4 presents the energy consumption for each significant



(a) 1 node, 8 tasks.

(b) 8 nodes, 64 tasks.

(c) 32 nodes, 64 tasks.

Figure 2. Energy consumption measured for different node and task configurations of LULESH during DTA.

region obtained for the best region-specific settings of the CPU frequency and the uncore frequency, as shown in columns 5 and 6. As it can be seen, the best setting for the phase is not necessarily the best setting for all the significant regions. Also, the best configuration varies for different significant regions. This is where the READEX dynamic tuning can be applied. The static energy saving amounts to a total of 1.32% for all the significant regions and 13.76% for the phase region as compared to the default energy values in column 2. The improvement obtained using the READEX tuning for the significant regions is 8.02% over the energy for the best static configuration for these regions (column 3).

Figure 2 shows the trend in the energy consumption obtained for three different node configurations (nodes=1 and tasks=8, nodes=8 and tasks=64, nodes=32 and tasks=64) under different settings for the CPU and uncore frequency. As seen, extremely low values of CPU frequency lead to a higher energy consumption for the phase for all values of the uncore frequency. This trend is also seen for higher values of the uncore frequency. We observe that the least amount of energy is consumed for a lower setting for the uncore frequency, and a medium-high setting for the CPU frequency. Thus, the best configuration for: 1 node and 8 tasks is $\{CPU\_Freq=1.8,\ Uncore\_Freq=1.2\}$ and 8 nodes and 64 tasks is $\{CPU\_Freq=1.8,\ Uncore\_Freq=1.2\}$. For 32 nodes and 64 tasks, since the application is not memory-bound, the best configuration $\{CPU\_Freq=2.2,\ Uncore\_Freq=1.2\}$ has a higher setting for the CPU frequency. This suggests a dependence on the input parameters, and will be explored in the future work of the project.

7. Conclusion

Energy consumption is a major challenge on the road to Exascale computing. READEX aims to address this by providing an online approach to dynamically tune HPC applications to significantly improve the energy-efficiency and performance. This paper presented in detail the preparatory steps and described how the tuning potential is leveraged during Design-Time Analysis in the READEX methodology. In contrast to previous approaches, READEX has a runtime tuning stage, which is guided by a tuning model that is pre-computed during application design-time. This paper also demonstrated the expected static and dynamic savings for LULESH, as well as the trend in the energy consumption for different node and task configurations using the READEX approach.

8. 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] Y. Oleynik, M. Gerndt, J. Schuchart, P. G. Kjeldsberg, and W. E. Nagel, "Run-time exploitation of application dynamism for energy-efficient exascale computing (READEX)," in *Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on*, C. Plessl, D. El Baz, G. Cong, J. M. P. Cardoso, L. Veiga, and T. Rauber, Eds. Piscataway: IEEE, Oct 2015, pp. 347–350.

[2] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "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.

[3] "Score-E: Scalable tools for the analysis and optimization of energy consumption in HPC," http://www.vi-hps.org/projects/score-e.

[4] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. Nagel, M. Simon, and Y. Georgiou, "HDEEM: High Definition Energy Efficiency Monitoring," in *Energy Efficient Supercomputing Workshop (E2SC)*, Nov 2014, http://dx.doi.org/10.1109/E2SC.2014.13DOI: 10.1109/E2SC.2014.13.

[5] I. Karlin, J. Keasler, and R. Neely, "Lulesh 2.0 updates and changes," Tech. Rep. LLNL-TR-641973, August 2013.

[6] M. Gerndt, E. César, and S. Benkner, Eds., *Automatic Tuning of HPC Applications - The Periscope Tuning Framework*. Aachen: Shaker Verlag, 2015.

[7] C. Guillen, C. Navarrete, D. Brayford, W. Hesse, and M. Brehm, "Dvfs automatic tuning plugin for energy related tuning objectives," in *Green High Performance Computing (ICGHPC), 2016 2nd International Conference on*. IEEE, 2016, pp. 1–8.

[8] J. H. Laros III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan, "Energy based performance tuning for large scale high performance computing systems," in *Proceedings of the 2012 Symposium on High Performance Computing*. Society for Computer Simulation International, 2012, p. 6.

[9] M. Etinski, J. Corbalán, J. Labarta, and M. Valero, "Understanding the future of energy-performance trade-off via dvfs in hpc environments," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 579–590, 2012.

[10] H. Kimura, T. Imada, and M. Sato, "Runtime energy adaptation with low-impact instrumented code in a power-scalable cluster system," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 378–387.

[11] C. Silvano, G. Agosta, S. Cherubin, D. Gadioli, G. Palermo, A. Bartolini, L. Benini, J. Martinovič, M. Palkovič, K. Slaninová *et al.*, "The antarex approach to autotuning and adaptivity for energy efficient

hpc systems," in *Proceedings of the ACM International Conference on Computing Frontiers*. ACM, 2016, pp. 288–293.

[12] P. Saviankou, M. Knobloch, A. Visser, and B. Mohr, "Cube v4: From performance report explorer to performance analysis tool," *Procedia Computer Science*, vol. 51, pp. 1343–1352, 2015.

[13] "Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory," Tech. Rep. LLNL-TR-490254.