Received: 11 January 2022 | Revised: 9 June 2022 | Accepted: 16 July 2022

DOI: 10.1002/spe.3139

RESEARCH ARTICLE

WILEY

DEPO: A dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing

Adam Krzywaniak[1] | **Paweł Czarnul**[1] | **Jerzy Proficz**[2]

[1]Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland

[2]Centre of Informatics—Tricity Academic Supercomputer and networK (CI TASK), Gdańsk University of Technology, Gdańsk, Poland

Correspondence
Adam Krzywaniak, Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland.
Email: adam.krzywaniak@pg.edu.pl

Funding information
None

Abstract

In the article we propose an automatic power capping software tool DEPO that allows one to perform runtime optimization of performance and energy related metrics. For an assumed application model with an initialization phase followed by a running phase with uniform compute and memory intensity, the tool performs automatic tuning engaging one of the two exploration algorithms—linear search (LS) and golden section search (GSS), finds a power cap optimizing a given metric and sets it for the remaining computations. The considered metrics include energy (E), energy-delay sum, energy-delay product. We present experimental results obtained for a set of benchmarks that differ in compute and memory intensity—parallel custom built OpenMP implementations of: numerical integration, heat distribution simulation (HEAT), fast Fourier transform (FFT), and additionally NAS parallel benchmarks: CG, MG, BT, SP, and LU. Tests were performed using multi-core CPUs that are representatives of modern servers and the desktop family: 2 × Intel Xeon E5-2670 v3 CPU (Haswell-EP) and Intel i7-9700K CPU (Coffee Lake). The results show that our approach enabled considerable improvements for the tested metrics, for example, for HEAT and Coffee Lake we minimized energy by 50% at the cost of a 15% increase in execution time (LS), for FFT energy was minimized by 40% at a 25.5% increase in execution time (GSS), for SP and Haswell energy was minimized by 25% at the cost of an 18.5% time increase and for Coffee Lake energy was decreased by 56% with a 12% time increase.

KEYWORDS

automatic power capping, green computing, HPC, performance-energy trade-off, software tools

1 | INTRODUCTION

Nowadays, providing high-performance computing (HPC) resources can be expensive, especially when the power required by computing centers exceeds megawatts. Under such circumstances, every method that allows users to decrease power consumption is extremely desirable, and even low energy savings are multiplied by the effects of scale. Thus, new

Abbreviations: EDP, energy-delay product; EDS, energy-delay sum; GSS, golden section search; HPC, high-performance computing; PDU, power distribution unit.

techniques enabling the control of computing equipment's electric characteristics are developed and deployed in almost all new CPU/GPU devices.

Power capping is one of the recently introduced mechanisms, operating at the hardware level, intending to limit the current power level of performed computations. Using this functionality seems to be a straightforward method to save the costs of electricity. However, quite often power limitations can cause an unacceptable performance decrease, or even opposite effect to the energy consumption.[1]

The main goal of our research is to provide means for optimization of energy consumption applicable to a range of CPU devices. We assumed the usage of power capping and focused on the solutions either giving the highest energy saving or being a compromise between performance degradation and energy consumption reduction. The contribution of the article is as follows:

1. A new method of optimizing energy consumption at application runtime, using the power capping functionality provided by modern CPUs, and considering both single objective (energy) and multi-objective (energy and performance) optimization (power capping as a new API available for a range of CPUs as well as GPUs can exploit various power management techniques and is available for various system domains as described further).
2. Review of existing static target metrics (considering performance in a time domain) for multi-objective energy-performance optimization, along with the proposition of new dynamic (considering performance as the number of instructions executed within a fixed period of time) versions of these metrics.
3. An automatic tool for an HPC environment—published and available online as a part of the Software Power Limiting Tools (SPLiT) suite*, implementing both the aforementioned method with two exploration algorithms (linear search (LS) and golden section search (GSS)) and dynamic versions of selected metrics (optimizing energy (E), energy-delay sum (EDS), or energy-delay product (EDP)).
4. An empirical proof of the method's effectiveness by experiments performed on both desktop and server CPUs using the proposed tool, and considering typical HPC applications including our custom developed (code available online* as "minibenchmarks") numerical integration (INT), fast Fourier transform (FFT), heat distribution (HEAT) as well as the well-known conjugate gradient (CG), multi-grid (MG), block tri-diagonal solver (BT), scalar penta-diagonal solver (SP), and lower-upper Gauss–Seidel solver (LU) from the NAS parallel benchmarks suite.[2]

The following Section 2 presents related works, Section 3 shows our motivations along with the foundation for the dynamic power capping and the DEPO tool concept and design. Section 4 discusses dynamic versions of selected target metrics, Section 5 presents the implementation details of the proposed tool. Section 6 describes the testbed environments and applications, experiments and their results, performed using the aforementioned workload types representative for HPC. Finally, conclusions and future works are outlined in Section 7. Appendix A discusses and justifies the approach using the Intel Running Average Power Limit (RAPL) driver verified against power distribution unit (PDU) measurements for the FFT, HEAT, and INT applications on the Haswell system, especially taking the measurement window and averaging used in the proposed implementation.

2 | RELATED WORK

In the first part of this section we characterize the current landscape of energy-aware HPC,[3,4] including tools for energy management and control, optimization techniques and methods of power/energy control, energy and power efficiency metrics, power and energy modeling approaches, placing this work in that context. Additionally, we outline the details of the most relevant works, again denoting key differences and comparison to our approach. In the second part of this section we review currently existing target metrics for multi-objective energy-performance optimization and we justify why there is a need for using any single choice metric when optimizing automatically the energy consumption with performance considerations.

2.1 | Current approaches and classification of energy-aware HPC

The currently available software tools for energy management of the hardware components are usually used for power monitoring and/or power controlling. Depending on the vendor, the most popular ones are as follows: RAPL[5] (for Intel

*https://projects.task.gda.pl/akrz/split/

CPUs), APM[6] (for AMD CPUs), energy scale[7] (for IBM CPUs), and NVML/nvidia-smi[8] (for NVIDIA GPUs). In this work we adopt RAPL as an underlying layer of our configuration and optimization software.

The currently available optimization techniques utilize different target metrics, we can distinguish the several main groups, see Table 1. The first group represents methods related to maintain a specified limited power, but maximizing performance of the system. Such an approach is useful in the case of limited auxiliary resources, such as the capacity of energy lines or air conditioning. The second group provides techniques targeting optimization of both energy and performance. They are important for solutions, where energy is an important factor, but the performance needs to be maintained on some reasonable level. Finally, the third group covers methods for energy minimization, without consideration of other aspects of system functioning. In this work we target our optimization on an energy minimization and on performance and energy trade-offs represented by three target metrics: E, EDP, and EDS, as described in Section 4.

From the point of view of system complexity, we can distinguish various hardware levels that are considered in the context of energy-aware parallel computing.[4] The most basic one is a single device, with the focus on used frequencies and core number, the next is a multiprocessor system where more than one CPU/GPU is involved in computations, then clusters (also in the context of big data processing[26]) and clouds (including the context of data centers, virtual machines, and resource provisioning policies[27]) where multiple compute nodes are taken under consideration, and finally grids being collections of clusters. In this work we focus on shared memory systems with multicore CPUs.

For the above metrics and hardware levels, Table 2 presents a review of tools dedicated for power/energy consumption control. In this work we adopt an approach for automatic search through various energy-power configurations using CPU power capping for optimization of the considered metrics.

TABLE 1 Main groups of power/energy-aware criteria and related, state-of-the-art optimization solutions

Criteria	Description of the solution
Performance with power limit	Selection of devices and scheduling for minimization of application execution time with an upper bound for the total power consumption of CPU and GPU compute devices[9]
	Integration of power limitation into a job scheduler and SLURM implementation[10] also including RAPL[11]
	A hybrid software/hardware power capping system which is based on a decision framework that makes decisions on the desired configuration[1]
	Scheduling of sporadic real-time tasks for thermal management with minimization of the peak temperature, using homogeneous ARM processors[12]
	Minimizing peak temperature in a multicore system by selecting core speeds[13]
	Task scheduling with thermal consideration applicable to a heterogeneous real-time multiprocessor system[14]
	Temperature-aware workload placement a in data center[15]
Performance and energy trade-off	Concurrent kernel scheduling/execution on a GPU and frequency scaling[16]
	A framework detecting recurrent workload patterns to reconfigure multiple subsystems, including NIC and HDD, dynamically and reduce overall energy consumption[17]
	Fine-grained autotuning of an HPC application combined with DVFS scaling[18]
	Bi-objective optimization with makespan and average energy consumption[19]
Energy minimization	Reducing power consumption of CPUs on which processes perform I/O operations or are idle[20]
	Energy-aware selection of a subset of differently-clocked cores in a heterogeneous chip to perform a specific HPC application[21]
	DVFS optimization during time of lower activity[22] especially on particular nodes[23]
	Prediction modeling of potential optimization, based on various application and system parameters (performance counters), using a non-negative multivariate regression[24]
	Product of energy and execution time (EDP) optimization by computation partial offloading to GPU[25]

TABLE 2 Software tools for power/energy-aware management.

Power management method	Description of the tool
Selection of the devices/scheduling	Scheduling a parallel application into a heterogeneous CPU and CPU+GPU cluster nodes[9]
	Energy-Aware-Multi-Cluster (EAMC) job scheduling policy integrated into the SLURM scheduler[28]
DVFS (dynamic voltage and frequency scaling), DFS (dynamic frequency scaling) or DCT (dynamic concurrency throttling)	READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) OpenMP and MPI code instrumenting tools for optimization of energy-aware HPC computing[29]
	A multi-agent based intelligent energy management framework for a reduction of power of idle or partially loaded CPUs[20]
	LEO (learning for energy optimization) a framework based on a probabilistic graphical model for obtaining Pareto-optimal power and performance trade-offs[30]
	A framework implementing two EDP-optimizing (energy delay product) algorithms: SEA and SPRA[31]
	An extension to SLURM scheduler to implement a "uniform frequency" in different configuration modes[10]
Power capping	CoPPer framework using power capping and adaptive control to approximate non-linearities in the power and performance relationship[32]
	PShifter: dynamic redistribution of power budged between cluster nodes using power limitation for faster processes[33]
Application optimizations	A framework modeling impact of optimization and providing recommendations for energy savings[24]
	Preparing best application configuration and settings on a GPU[25]
	Controlling CPU frequency, disk spinning and network speed scaling[34]
Hybrids of the above	A software/hardware approach with power capping based on a framework that makes decisions on configurations going through nodes[1]
	A reinforcement learning framework using power capping and uncore frequency scaling for optimization of the power consumption and run time[35]
	Scheduling/software as well as resource management with the use of RAPL[11]
	Scheduling kernels within a GPU as well as frequency scaling[16]

Subsequently, we provide a concise comparison of selected approaches presented in respective research works to energy-performance oriented optimization in high-performance computing and presentation of the contribution and differences presented by us within this article.

Within the READEX (Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing) approaches and tools for the optimization of energy-aware HPC computing have been developed.[29,36] Specifically, a MERIC tool allows one to instrument a parallel application and subsequently perform energy-aware optimization. MERIC inserts synchronization MPI and OpenMP barriers into an application code in order to make sure that all running processes and/or threads are synchronized. It requires an HDEEM or ×86 adapt library for accessing RAPL counters, Cpufreq or ×86 adapt library for CPU frequency management, and a PAPI and perf event for access to hardware counters. Then the programmer is responsible for code analysis, identification of regions such as compute, communication, I/O and instrumenting the application using the supported API. The CPU frequency, uncore frequency and number of OpenMP threads are set by MERIC for each region for optimization. The approach allows so-called static tuning for which all parameters are set before an application is started and dynamic tuning when some parameters can be set at runtime. The RADAR generator allows users to evaluate results from MERIC automatically while Score-P is said to provide automatic instrumentation. In terms of applications, for ESPRESO (combination of finite element, boundary element tools

and TFETI/HTFETI domain decomposition solvers) code considerable energy savings (4% for static tuning and additional 7.46% for dynamic tuning) were shown when run on a cluster with two Intel Xeon E5-2680v3 (Haswell-EP) processors per node,[29] for Lattice Boltzmann simulation code that is, (compared to default maximum frequency settings) 9% of CPU package energy for static tuning and additional an 3.5% for dynamic tuning at the cost of less than a 3% increased execution time run on 2 Intel Haswell E5-2630v3 CPUs. Compared to that, we focus on an automatic approach for runtime application of a power cap optimizing a given function involving execution time and energy without any changes to the source code. We collect application progress using hardware counters and explore performance-energy configurations automatically searching for optimization of one of several metrics incorporating execution time and energy used which should be free of overheads due to potential instrumentation.

Komoda et al. presented an interesting approach, dedicated to a heterogeneous CPU-GPU environment, where DVFS mechanisms were used to decrease power level.[37] The authors proposed a new model estimating performance/energy consumption, and used it to improve task assignment (either to CPU or GPU), which resulted in decreasing waiting times of kernel barrier synchronization, and in turn better performance of the system. Setting device frequencies and the task mapping in advance of the application execution in a CPU+GPU environment using empirical models was adopted. Using the above solution gave precise, flexible and efficient power capping functionality. The following applications were used: BFS—graph traversal, HOTSPOT—heat simulation, KMEANS—clustering, PF—model estimation, and SGEMM—BLAS library. Compared to that approach, we focus on experimental and automatic search for optimized configurations.

Mishra et al.[30] proposed an algorithm optimizing the energy-performance trade-off using the DCT technique for power control. The optimization is performed using a hierarchical Bayesian model, set up with historical (off-line) data and tuned to the current application execution in runtime (on-line).

Several benchmarks were used from different suites including PARSEC (blackscholes, bodytrack, fluidanimate, swaptions, ×264), Minebench (ScalParC, apr, semphy, svmrfe, Kmeans, HOP, PLSA, non fuzzy kmeans (Kmeansnf)), Rodinia (cfd, nn, lud, particlefilter, vips, btree, streamcluster, backprop, bfs), jacobi solver, filebound and the swish++ search web-server. In our approach we use power-capping mechanisms and provide automatic selection of the energy-performance ratio at application runtime using built-in metrics, without prior execution data gathering.

Berned et al.[31] proposed off-line heuristics optimizing an EDP metric for parallel applications, they use DCT mechanisms for power limitation and the results showed a significant decrease of energy consumption, with a moderate performance penalty. In this context, many benchmarks were used from the NAS Parallel suite: block tri-diagonal solver (BT) CG, embarrassingly parallel (EP), discrete 3d fast Fourier transform (FT), integer sort (IS), lower–upper Gauss–Seidel solver (LU) multi-grid on a sequence of meshes (MG), scalar penta-diagonal solver (SP), unstructured adaptive mesh (UA); from the Rodinia Benchmark Suite: Hotspot (HS), LavaMD2 (LM), Leukocyte (LT), Needleman–Wunsch (NW); fast Fourier transform (FFT), Jacobi (JA), n-body (NB), STREAM (ST), Poisson (PO). Our solution uses an on-line approach without prior application execution and is based on power-cap mechanisms. Moreover, our method supports EDP as well as other metrics.

Zhu et al.[38] presented a new scheduling method optimizing the performance of parallel programs execution on integrated CPU-GPU chips under a given power cap. The authors provided a new performance model based on a measured application memory throughput while executed on GPU or CPU cores. Then they proposed a scheduling algorithm reflecting device characteristics related to the performance and power capping. Experimental results, performed using a typical OpenCL based, Rodinia benchmark suite, showed the effectiveness of the method in comparison to random and default (Linux) scheduling algorithms. In this context, the results of our work can be combined with scheduling at a higher level in which various energy-performance configurations could be considered for each assignment.

Rountree et al.[39] presented the power capping functionality and its adoption for total power limitation of the used devices, with the assumption of maximizing the system performance. They proposed an idea of the data center bound by the maximum used electric power instead of compute power capacities. Under such circumstances, the power capping is going to play the main role in setting up the boundaries of intensity of possible electricity usage. Additionally, they presented performance differences between various instances of the same CPU type, measured under the power limitation, showing that in practice their compute power can vary significantly, which is not so visible for conditions when no power cap is introduced.

In this work we focus at a lower level that can potentially be used to allow various power caps for various users in an HPC center and allow per-user optimization as well as global constraint involvement.

Haidar et al. presented a study of the correlation between power usage and performance for various numerical kernels typical of real scientific applications,[40] gathered using the Intel Xeon Phi architecture, showing how power capping enables reduced energy consumption with a negligible performance loss. The analysis showed that the highest energy

saving can be achieved while using memory intensive applications, where the power limits do not significantly influence the performance.

A set of the benchmarks mirroring typical science applications in HPC, as well as having different compute/memory intensity were used including: Jacobi, LBM, HPCG, XSBench mini-app, Stream benchmark. We perform a similar work, although using a different set of applications and multi-core CPUs focusing on the automation of the optimization process.

Fukazawa et al.[41] in their research focused on analysis of performance/execution time versus power limit configurations, using Intel RAPL and Magnetohydrodynamic (MHD) simulation code. Specifically, CPU and DRAM power consumption values are measured for two parts of the code—calc-part and data-part under various CPU power caps between 100 and 260 W. It is shown that for the latter part, under power capping considerable CPU power consumption can be obtained with minor increases in the elapsed time while DRAM power consumption stays at a constant level. This allows visible energy optimization at potentially minor increase of execution time. A parallel system with Sandy Bridge Xeon E5 CPUs was used for tests. Tiwari et al.[42] focused on building a model that considers power allocation for CPU and DRAM domains and its impact on performance when power capping.[42] A function is defined that predicts performance change when using power caps for CPU and DRAM domains. Using low level tools for application instrumentation as well as rule-based machine learning, this impact is estimated. The model considers performance changes from CPU and DRAM domains with power caps as well as cache hits and misses, memory accesses, ratio of floating point operations, instruction mixes and data dependence metrics. The authors demonstrated an average absolute mean error of 6% with the error below 10% for more than 86% of the application hotspots. Several codes were used: miniGhost, CoMD, kernels from various computational domains—dense linear algebra (matrix-matrix, matrix-vector multiplication), stencil computations, linear algebra solvers (LU decomposition). Compared to those works, we adopt a different approach focusing on the automation of the configuration process and also algorithms that identify configuration fast and with good precision but we are planning to incorporate DRAM domain consideration in the future work extending the final algorithms.

Conoci et al. in their work[43] adopted performance and power limit aware optimization of a configuration with self-adaptation of parallel applications able to find a good configuration (frequency, cores, and threads placement) satisfying a user requirement on power consumption and performance. The approach obtains relevant performance and power models on-the-fly. The applications they used were: Blackscholes, Bodytrack, Canneal, Dedup, Facesim, Ferret, Fluidanimate, Freqmine, Raytrace, Streamcluster, Swaptions, Vips. Our solution picks up a slightly different problem, where on the left-hand side is total energy used by the application. We consider the maximal power constraint to be maintained by the hardware supported power-capping mechanisms.

CoPPer library, proposed by Imes et al.,[32] uses power-capping mechanisms for performance management supporting the power minimization, for soft real-time applications. It allows the use of hardware power capping to meet application performance requirements with high energy efficiency, avoiding over-allocating power when not beneficial. Applications from the PARSEC benchmark suite, MineBench, STREAM and SWISH++ were tested. Similarly to our approach, that solution controls the performance by applying power limits. However, it uses its own performance indicators being implemented in the additional code dependent on the application, and does not provide flexible metrics referring to an energy-performance trade-off.

Conoci et al.[44] proposed techniques for optimization of throughput of a multithreaded application under a given power cap where two variables are considered: P-state and the number of threads. The authors tested two strategies: exploration as well as model (with performance and power-usage model construction) based. Both models resulted in errors below 5%. As a conclusion following tests, one of the two strategies can be recommended as follows: exploration one for workloads with unimodality of the scalability profile, and a model-based one for workloads with distinctly different profiles. Three groups of applications were distinguished: with highest performance for the highest number of available cores, middle as well as just one core, respectively. In our approach the number of threads is imposed when running an application and considered fixed, not a variable. Power settings are set with a power cap and are meant to optimize a function corresponding to an acceptable/desired trade-off between application execution time and energy used.

2.2 | Review of existing target metrics

For multi-objective optimization, all possible optimal solutions may form a Pareto-optimal front, where one of the solutions might be arbitrarily selected. This indeed was our first approach.[45]

Running any application for different power limits generates a set of results consisting of total execution time and total energy consumption. Many of the result points were typically included in the Pareto-optimal front. Figure 1 presents

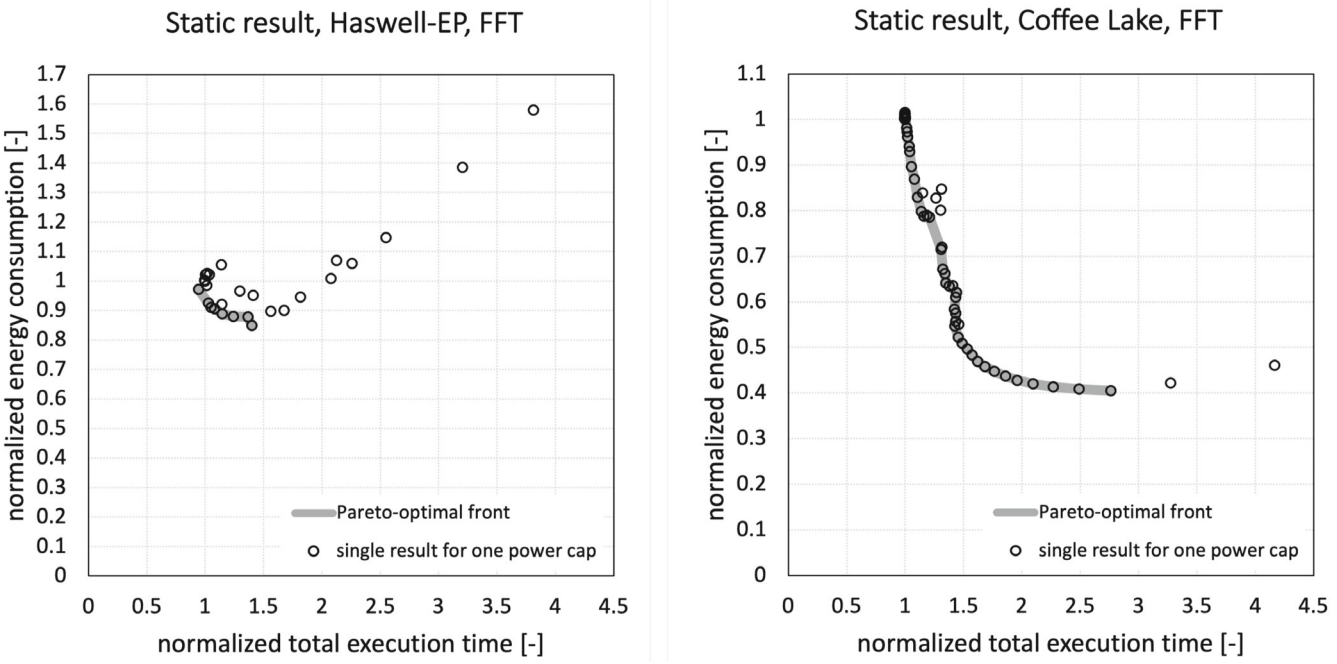

FIGURE 1 Pareto-optimal front plotted for an exemplary static result of FFT application execution on Haswell-EP (left) and Coffee Lake (right) systems for different power caps. The results are normalized with the result obtained for default system configuration as a base.

one of the application's (FFT—fast Fourier transform) results obtained on two considered CPUs with the Pareto-optimal front highlighted from all available results. On the horizontal axis we put normalized execution time and on the vertical axis the normalized energy consumption was placed. All the results are normalized with a base result (total execution time or total energy consumption) obtained for a default system settings (no power caps applied). Any value less than 1 represents a better than default result (less energy consumed or shorter execution time) and, consequently, any value above 1 is worse than the default result.

However, after in-depth analysis, we found out that a fully automatic implementation needs a more direct way to find desired optimization, as the whole procedure takes a lot of time to collect a proper set of measurements to build the front. Moreover, even having solutions, the tool needs to select one of them, using some additional criteria, to introduce a final, target power-cap. Thus, we decided to stick with a well-defined metric function, which can be easily exchanged for a different one if user requirements change.

In our case, the metric function, which value is going to be minimized, has the following form:

$$M : (E, t) \in \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+, \tag{1}$$

where E and t are respectively estimated energy and time required for the application execution.

The proposed tool is metric agnostic, that is, it can use any monotonically increasing function to define the goal of the energy/performance optimization. The simplest case represents a minimal energy (E) goal, where the execution time is not taken under consideration:

$$M_E(E, t) = E. \tag{2}$$

On the opposite side, where the performance is maximized, without any power considerations, the following metric can be used:

$$M_t(E, t) = t. \tag{3}$$

There are several metrics being in between these two extremes, and the usually used one is energy time product, a.k.a. EDP:[46]

$$M_{EDP}(E,t) = Et. \tag{4}$$

In some works,[47,48] it appears in a more general form:

$$M_{EDP}(E,t) = Et^w \tag{5}$$

with a range of used w parameter, that is, $w = 2$ or $w = 3$.

Roberts et al. argue that EDP is unsuitable for energy aware optimization,[49] and they propose two other metrics: EDS:

$$M_{EDS}(E,t) = \alpha E + \beta t, \tag{6}$$

and energy-delay distance (EDD):

$$M_{EDD}(E,t) = \sqrt{(\alpha E)^2 + (\beta t)^2}. \tag{7}$$

3 | DEPO TOOL—MOTIVATION, CONCEPT, AND DESIGN

The analysis of the state-of-the-art research can be summarized by conclusions that there is a lack of tools for automatic configuration of HPC systems especially the ones involving power caps mechanisms for both CPU and GPU based solutions. In order to advance this field of study even further, we propose and present an implementation of an automatic and dynamic tool for finding the desired performance-energy consumption trade-off configurations for a given application. In the following Section 3.1 we summarize our previous works and present initial experiments which are a foundation for the dynamic power capping concept. Then, in Section 3.2 we introduce the concept and the design of the DEPO tool.

3.1 | Dynamic power capping foundations

Our previous works explored power capping usability in HPC in a static way. The research was based on manual configuration of a testbed system where we performed a series of tests using different power limits.[45] We have observed that setting a power cap on a system results in non-trivial energy savings and performance degradation trade-offs. However, one cannot easily predict potential benefit and loss caused by a particular power limit level. We have observed that the Energy characteristic, defined as a total energy consumption level against power limit level, is specific for an application and testbed system pair.

Afterwards, we extended our experiments using a subset of NAS parallel benchmarks[2] applications, well known in the HPC environment, and proposed an automatic tool that we called EnergyProfiler that can execute a series of tests on any compatible modern Intel CPU, and produce an energy characteristic as an output.[50] After publishing the code in SPLiT suite we have renamed the EnergyProfiler into StaticEnergyProfiler (StEP). Additionally, we performed tests for various input data sizes per each computational workload type. This allowed us to conclude that at least for some classes of iterative applications, performing the same computation on the data with the same layout at each iteration, the energy characteristic is indeed specific for an application-system pair, but does not depend on the computational problem size. As long as the memory is not a limitation, we can assume that the workload, executing specific types of computations, will have its energy minimum for the same power limit level, no matter if it is executing for an hour or just a few seconds. This revealed the potential of automatic tools for finding the desired energy efficient system configurations, based on a series of short computational workload samples.

The main disadvantage of the static method for finding the best system configuration using the static StEP tool is that we have to measure the total execution time of a workload application. Without knowledge of total execution time and total energy consumption of the testbed workload the static method cannot apply any of the static target metrics evaluating the energy/performance impact of the particular power limit level. Therefore, the StEP tool based on static power capping exploration method is useful only when a tuned application is frequently repeated or allows for a short sample run before long target execution .

The new approach proposed in this article is dynamic power capping. The main assumption is that the power capping exploration phase, as well as the decision on selecting the best power limit level for any given workload, is performed

at runtime. In order to design such an approach we need to use a method which allows for estimating the progress of computations. This can be represented by the number of instructions executed on a CPU.

Figure 2 presents an exemplary static result obtained with the StEP tool exploring different power caps, extended by CPU counters monitoring. The figure presents normalized total energy consumption, as well as total execution time and total number of instructions, executed for an example testbed workload (FFT). The figure also shows evaluation of static target metrics: EDP and EDS ($k = 2$).

The results confirm the intuitive assumption that, regardless of the current power cap level and the performance impact of it, the total number of instructions required for completing a computational problem is constant for an application. This is a foundation for dynamic power capping design as it solves the problem of the runtime application progress estimation.

3.2 | DEPO tool—concept and design

The main contribution of this article is the concept of a dynamic power capping mechanism, its design and implementation, followed by a series of experiments proving its usability and effectiveness. We have implemented a new tool allowing for quick exploration of available power caps range for an unknown application. Examination of software power limits does not require the application to finish its execution. All the measurements are performed at runtime. The general idea is to find the best suited power cap at the beginning of the application execution and use it for the rest of the computations. We call the new tool DEPO which can be expanded as *dynamic energy-performance optimization*.

Figure 3 presents a component communication diagram, describing the general idea of our DEPO tool. A tuned application is executed simultaneously with the tool, notifying it about exact times of start and end of the computations. The power monitoring and controlling interface is implemented using the Intel RAPL driver. The tool automatically detects all available packages (CPUs) and reports the current energy consumption for a system as a sum of energy readings from each available package. The power limits are applied in the form of a power budget, which means that the limit assigned by the DEPO tool is split equally between available packages. Specifically, for a single CPU system the whole power budget is assigned to a single package. Consequently, for a system with two CPUs the power budget is split in half and each half of a power budget is applied as a power cap for each of the two packages (CPUs).

Power capping using the RAPL driver allows for defining long term and short term power limits. Using RAPL it is possible to set both power limit values, as well as the corresponding time windows. The time window defines the time period within which the average power consumption is forced to not exceed the requested power cap.

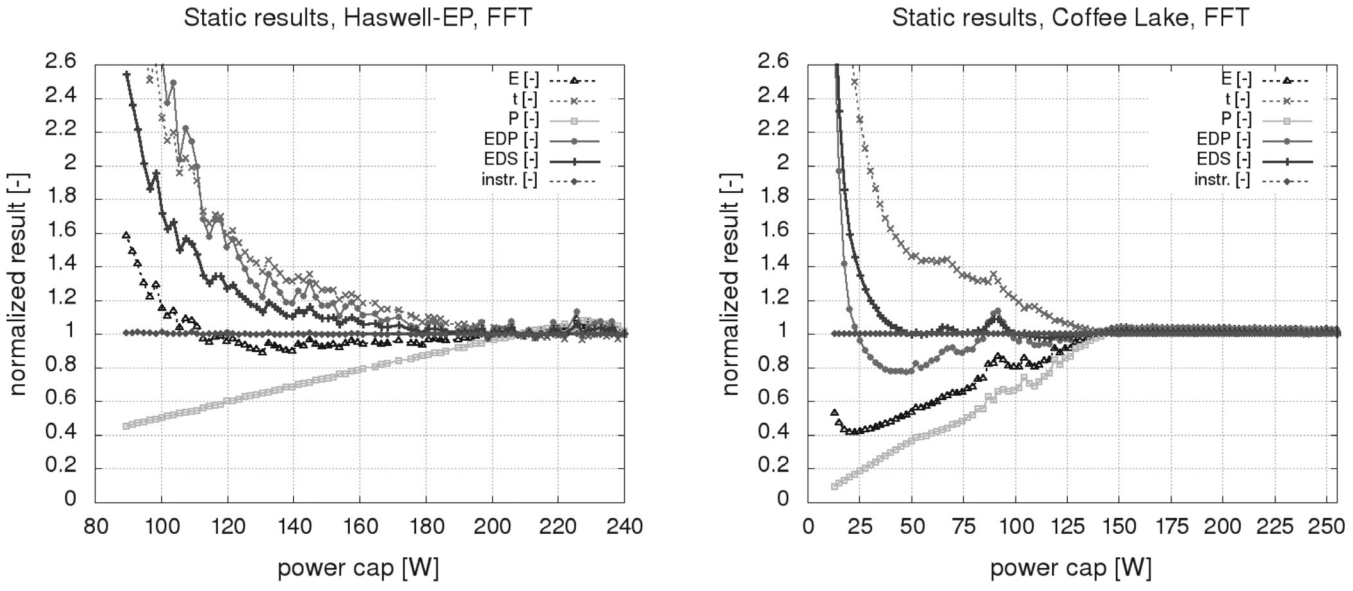

FIGURE 2 Static result obtained with StEP tool with CPU counters on Haswell-EP (left) and Coffee Lake (right).

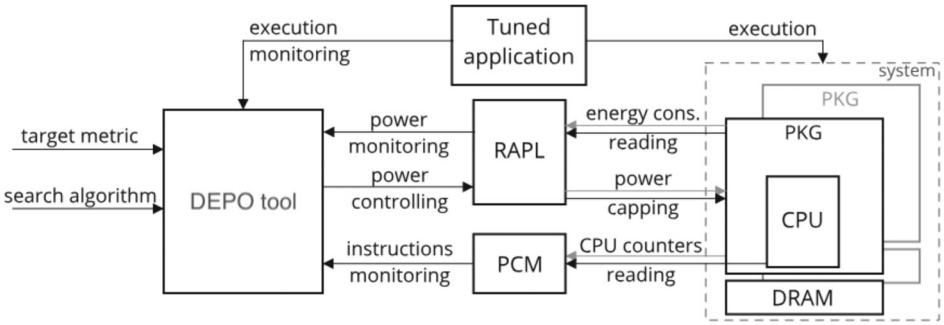

FIGURE 3 DEPO general component communication diagram

The DEPO tool applies the power caps using only the long term power limit, and allowing for power consumption peaks exceeding the requested power limit value. The long term time window is set to 100 ms. The short term power limit has always a default value specific for a CPU model. The short limit time windows in the systems tested in this article had a 2 ms value. We do not apply limits to the DRAM domain.

The Intel PCM API is used to monitor the number of executed instructions while running the application. The DEPO tool executes a series of measurements for different power caps and searches the most appropriate one. The used metric (E, EDP, or EDS) and the search type (linear or GSS) can be chosen by the user according to his/her preferences.

The DEPO tool is designed to be API agnostic so that it may be easily modified to work with other platforms for example, NVIDIA GPUs. The Intel specific RAPL API might be then replaced by NVML API calls for power consumption monitoring and controlling. We have already applied such an approach[51] for the StEP tool. When we replace another Intel specific PCM API with for example, NvBit[52] instruction or kernel counting, it would make DEPO tool able to be used for dynamic power capping with NVIDIA GPUs.

The implementation details of the DEPO tool are summarized in Section 5.

4 | DYNAMIC VERSIONS OF SELECTED TARGET METRICS

In Section 2.2 we described several target metrics allowing for evaluation of energy-performance trade-offs while using power caps. For the research described in this article we have selected the following metrics: simple energy (E, Equation 2), energy-delay product (EDP, Equation 4), and energy-delay sum (EDS, Equation 6). However, this set can be easily extended with other ones.

Figure 4 presents graphical visualization of the E, EDP, and EDS metrics on the total energy consumption and total execution time plane. The axes represent normalized energy consumption (vertical axis) and normalized total execution time (horizontal axis). The result obtained for default system setup (no power caps) was used as a reference and is represented by point with values (1,1). The energy (E) metric is just a simple horizontal line which divides the plane into the upper and lower subplanes. Any result on the lower subplane is better than the default one in terms of optimization of the E metric.

The EDP metric is represented by a hyperbole passing through point (1,1). Any result below the hyperbole is better than the default one in terms of EDP metric.

The EDS metric is presented for three different values of the k parameter which represents the maximal acceptable proportional execution time increase. The k parameter is directly correlated with Equation (6)'s α and β coefficients as described in Section 4.3. Any result below the EDS line is better than the default one in terms of EDS metric.

The general interpretation of the graphical visualization of these metrics is that any result represented by a normalized energy-time pair that is below the metric line is accepted by the metric and any other is rejected. The optimal (minimal) value for each metric is the point with maximal Euclidean distance between the result point and the metric line.

All the metrics reviewed above are useful only when total energy consumption and/or total execution time of the computational problems under examination are known, which makes these metrics static. To be able to evaluate the impact of dynamically changing power limit levels at runtime, we need to substitute total energy consumption and total execution time with their dynamic equivalents.

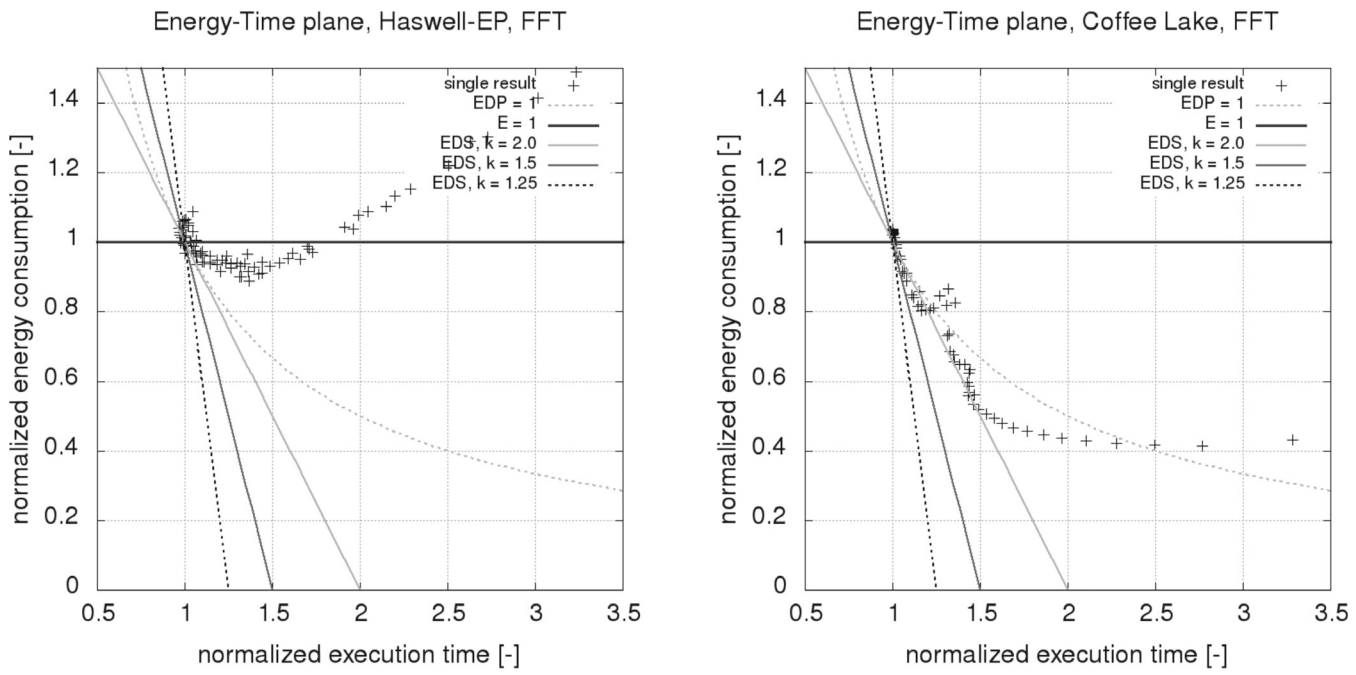

FIGURE 4 Graphical visualization of E, EDP, and EDS (for three different k values) metrics for an example workload (FFT) executed on Haswell-EP (left) and on Coffee Lake (right).

In the static approach,[45] for the comparison purposes, we used the total time of the application execution. However, the dynamic version of the evaluation needs to be performed at runtime, before the application execution is finished, therefore the total time has to be substituted by some other metric. Since, as described in Section 3.1, we observed that any application requires the same total number of instructions I to complete the computation regardless of the power cap, we decided to use instructions per second ratio ips_T instead. This approach, after the adjustment of the application startup (see Section 5.2), provides appropriate results, see Section 4.4 for further analysis.

4.1 | Dynamic minimum of energy (E)

Starting with general energy equation which defines total energy consumption as a product of average power and total execution time:

$$E_{\text{total}} = P_{avg} \cdot t_{\text{total}}. \tag{8}$$

We can replace total execution time with total number of instructions I_{total} divided by instructions per second ratio ips_T measured for the time period T:

$$t_{\text{total}} = \frac{I_{\text{total}}}{ips_T}. \tag{9}$$

We also know that average power P_{avg} in a time period T is a ratio of energy measured for a fixed period of time E_T and the period itself T:

$$E_{\text{total}} = \frac{E_T}{T} \cdot \frac{I_{\text{total}}}{ips_T}. \tag{10}$$

Using the number of instructions I_T measured within the time period T we can replace ips_T and simplify to the following form:

$$E_{\text{total}} = \frac{E_T}{I_T} \cdot I_{\text{total}}. \tag{11}$$

Simplifying and combining the above equations with Equation (2) we can write:

$$M_E(E, t) = \frac{E_T}{I_T} \cdot I_{\text{total}}. \tag{12}$$

Since the total number of instructions I_{total} is constant for a given application, we can simplify the minimum energy metric to its dynamic version:

$$E_{\text{dyn}} = \frac{E_T}{I_T}, \tag{13}$$

which may be interpreted as average energy per instruction ratio within a fixed period T. Thus, in order to find minimal energy consumption we can seek for minimum of energy per instruction or maximum of instructions per Joule.

4.2 | Dynamic energy delay product (EDP)

The second target metric that we have selected is energy consumption and execution time product which is known in literature as energy delay product.[46] Static version of this metric was defined in Equation (4). Using Equations (12) and (9) we can define EDP as:

$$M_{EDP}(E, t) = I_{\text{total}}^2 \cdot \frac{E_T}{I_T} \cdot \frac{1}{\frac{I_T}{T}}. \tag{14}$$

Remembering that total number of instructions I_{total} is constant and has no impact on the above optimization target formula we obtain final form of EDP_{dyn}:

$$EDP_{\text{dyn}} = \frac{E_T \cdot T}{I_T^2}, \tag{15}$$

where E_T represents energy consumption within period T.

4.3 | Dynamic energy delay sum (EDS)

The third target metric that we selected for dynamic power capping is energy delay sum (*EDS*) for which a general idea was presented in Equation (6). *EDS* is a weighted sum of total energy consumption and total execution time. Its geometric interpretation is a linear function on an energy and time plane which divides the surface into two subplanes. The line is constructed based on two points: a reference result and some assumed abstract point which considers zero energy consumption and maximal accepted execution time degradation. Usually the potential increase of execution time is defined as $k \cdot t_{\text{ref}}$ where k is a coefficient determining the potential performance loss ratio.

Any result point obtained for some different system configuration which is exactly on the *EDS* line is equivalent to the reference result. Any point below the line is better than the default and, consequently, any result point above the *EDS* line is rejected by this metric. Figure 4 presents a graphical interpretation of all static metrics including three potential *EDS* lines with different k values.

Based on the aforementioned two points $(t_{\text{ref}}, E_{\text{ref}})$ and $(k \cdot t_{\text{ref}}, 0)$ on a time-energy plane we can build a linear function equation:

$$E - E_{\text{ref}} = \frac{0 - E_{\text{ref}}}{k \cdot t_{\text{ref}} - t_{\text{ref}}} \cdot (t - t_{\text{ref}}), \tag{16}$$

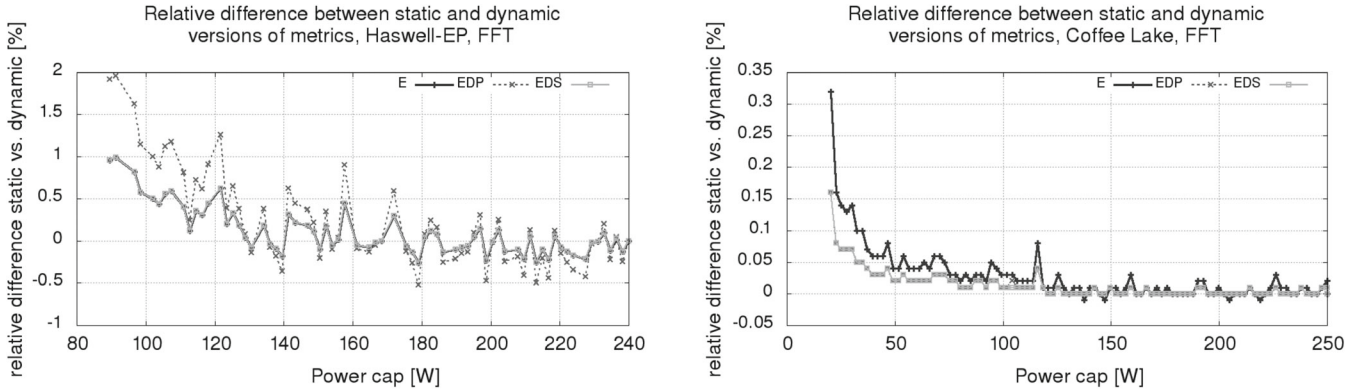

FIGURE 5 Comparison of relative static metrics (E, EDP, EDS) values against their dynamic equivalents calculated for the same series of static experiments run with the StEP tool on Haswell-EP (left) and Coffee Lake (right) for the FFT application.

which, after combining with Equation (6) provides α and β weights as:

$$\alpha = \frac{k-1}{k \cdot E_{\text{ref}}}, \beta = \frac{1}{k \cdot t_{\text{ref}}}. \tag{17}$$

Using coefficients from Equation (17) and dynamic equivalents of energy from Equation (11) and time from Equation (9) we can replace static energy and time in Equation (6) and result in:

$$EDS = \frac{k-1}{k \cdot \frac{I_{\text{total}} \cdot E_{T_{\text{ref}}}}{I_{T_{\text{ref}}}}} \cdot \frac{I_{\text{total}} \cdot E_T}{I_T} + \frac{1}{k \cdot \frac{I_{\text{total}}}{ips_{T_{\text{ref}}}}} \cdot \frac{I_{\text{total}}}{ips_T}. \tag{18}$$

After reductions and simplification of the above equation dynamic version of energy delay metric EDS_{dyn} is:

$$EDS_{\text{dyn}} = \frac{1}{k} \cdot \frac{I_{T_{\text{ref}}}}{I_T} \cdot \left((k-1) \cdot \frac{E_T}{E_{T_{\text{ref}}}} + 1 \right). \tag{19}$$

4.4 | Evaluation of dynamic metrics on static results obtained with StaticEnergyProfiler

Before implementation of dynamic power capping we compared our theoretical dynamic version of target metrics to static results obtained with the StEP tool running the FFT application. We have selected the FFT application as a representative for the initial test as it is the most complex problem in terms of its combination of computation and communication. Also, its power characteristic is the most variable. Thus, the FFT application shall be the biggest challenge for the tested dynamic approach. We used total execution time t and total number of instructions I ratio to calculate the average value of instructions per second ips_T measured within a fixed time period T. Also, dividing total energy E by total number of instructions I we modeled a dynamic version of energy measurements. The value of the k coefficient in the *EDS* metric for the purpose of this experiment was set to 2.0 which means that we accept execution time doubling at most.

Figure 5 presents comparison of static and dynamic metrics *E*, *EDP*, and *EDS* calculated for the same series of static experiments run with StEP tool for the FFT application on both Haswell-EP and Coffee Lake systems. The same experiments series was also a base for previous Figures 4 and 2.

We can observe that values of dynamic versions of *E*, *EDP*, and *EDS* metrics almost perfectly match the values of their static equivalents for the Coffee Lake system. The relative errors between static and dynamic versions of considered metrics calculated for static results obtained on the Coffee Lake system are smaller than 0.4%. For the server type Haswell-EP system the relative errors are less than 1% for E and EDS metrics and less than 2% for the EDP metric. This means that replacing the total execution time domain by the total number of instructions retired domain in proposed target metrics does not have significant impact on the final target metric value for the Coffee Lake system and has a negligible impact

for the Haswell-EP system. These results constitute a strong foundation for the proposed design and implementation of a tool for dynamic power capping.

5 | IMPLEMENTATION DETAILS OF DEPO TOOL

This section describes the implementation details of the proposed DEPO tool. First, in Section 5.1 we refer to our previous work and present the new open sourced set of tools for power monitoring and controlling called SPLiT. We also summarize all the Intel specific APIs that have been used in the DEPO tool. In Section 5.2 we introduce the idea of dynamic power capping phases. Search algorithms for the tuning phase are described in Section 5.3 and, in Section 5.4, we select the dynamic versions of target metrics.

5.1 | SPLiT: Software power limiting tools—The ECO library extension

In our previous work[50] we introduced a tool named EnergyProfiler which was built on our original energy consumption optimization (ECO) library. The ECO library aimed at defining a simple application programming interface (API) for power and energy consumption monitoring as well as software power limits controlling. The main goal for the ECO library is to allow easy building of simple custom applications for energy efficient (controlling) and energy aware (monitoring) computing. We have published the tools for software power monitoring and controlling as the Software Power Limiting Tools suite, available online*.

The underlying layer of ECO library is an Intel Running Average Power Limit (RAPL) driver which uses model specific registers to monitor the energy consumption of CPU and control its software power limits. The RAPL driver is able to monitor and control several power domains such as PP0—representing CPU cores, DRAM—representing memory and PKG—representing the whole CPU socket. The availability of particular domains depends on the CPU model. Since the SandyBridge Intel CPU family release, the PKG power domain is always available in modern Intel CPUs. The power capping API introduced with Intel's RAPL driver creates an abstract layer allowing for setting the power limits intuitive for the user power units (microwatts). The underlying technology is vendor specific and, according to Haidar et al.,[40] for Intel's RAPL it is a combination of Dynamic Voltage and Frequency Scaling (DVFS) and clock cycling modulation at low power levels.

Our previous work was focused on exploring the advantages of static power capping. The power limits were applied only to PKG domain (which refer to CPU cores) with no impact on DRAM domain. Based on the foundation of the ECO library, we have developed a set of applications for energy aware high-performance computing (HPC) such as: SetPowerLimit, GetPowerConsumption, and EnergyProfiler (now renamed to StEP: StaticEnergyProfiler).

In this article we extend the ECO library with new features and a new application for HPC. First of all, we have added the ability to monitor CPU counters such as executed instructions and CPU cycles. We have upgraded GetPowerConsumption (GPC) and StaticEnergyProfiler (StEP) tools to report the number of the instructions for each testbed application execution. The extended version of StEP was already useful for preliminary research for this article as described in Section 3.1.

Monitoring of CPU counters is realized with a processor counter monitor (PCM) Intel open source API.[53] In the ECO library we use a PCM object programmed for `PCM::DEFAULT_EVENTS`, through which we collect the counter states, using a dedicated `getAllCounterStates()` method. Finally, we read the number of instructions actually executed by CPU to complete the monitored application with `getInstructionsRetired()`. The PCM library was compiled without `-DPCM_USE_PERF` flag, which indicates that the performance measurements do not rely on Linux perf support.

For generating graphical reports online we used `gnuplot-iostream`,[54] which is basically a header-only library wrapping gnuplot application for C++ developers.

The ECO library, as well as any application within the Software Power Limiting Tools (SPLiT) collection, are written using the C++17 programming language and were compiled with the gcc version 7.5.0 compiler.

5.2 | Dynamic power capping phases

The DEPO tool performs power cap tuning and monitors the power consumption after applying the best power cap. Therefore, in the DEPO tool we can distinguish a clear preparation phase before the execution phase with desired system setup. HPC applications rarely represent a purely homogeneous workload type and usually there is some startup time required before the application starts to perform at the target power consumption level. Therefore, we need to add some *wait-for-steady-state* time before testing the power caps. Otherwise, we would base our decisions on the energy readings for the low power consumption startup phase while the main computations require much more power. For this reason, we split the preparation phase into a *wait phase* and a *tuning phase*. Figure 6 presents the DEPO phases scheme in parallel with the testbed application on a timeline.

Before the tuning phase starts, there is a need for detection if the tuned application is already running at its target power consumption level. A solution might assume some fixed startup time and introduce some sleep time before tuning phase. Unfortunately, such an approach would not be universal and might not be good enough for many HPC applications with non-trivial power characteristics.

As a more generic solution we decided to base the steady state detection on the filtered power readings. We used simple moving average (SMA) to get the current average power value. We analyze the variability of the current power and when the filtered power deviation is lower than the assumed fixed ϵ the tuning phase is triggered.

Technically, we calculate SMA50(P) for 50 recent power samples. For better accuracy with steady state detection we calculate SMA100(SMA50) for 100 recent SMA50 values. Finally, we select the minimal and maximal sample for each SMA100 value and calculate the relative $err = \frac{(\max - \min)}{SMA100}$ error. When $err < \epsilon$ the tuning phase is triggered. The ϵ value was selected experimentally and equals 0.03 for our experimental setup.

5.3 | Search algorithms in tuning phase

The tuning phase aims for finding the best power cap for a current workload according to the given target metric (one of: E, EDP, or EDS). The power cap is selected after a series of energy and power measurements obtained for different power limit levels, and performed within a fixed time period T which is defined by the user. The DEPO tool supports two algorithms for exploring the power caps: LS and GSS.

LS is a simple method of a gradual increase of the current power limit. The search starts from the lowest value close to idle consumption power demand and increases to reach either the double value of application maximal power consumption or the maximal available power cap—whichever is greater. After each gradual step the target metric is evaluated and referred to the current best value. The best power cap is stored each time the metric evaluation results with better values. The full power cap range is always explored from the maximal available power limit to the minimal power demand determined by system idle power consumption.

The second algorithm (GSS) implements a well known technique for finding function extreme points in a specific interval, first introduced by Kiefer in 1953.[55] The GSS is also an iterative method but its goal is to narrow the considered range until it is lower than assumed fixed parameter δ. When the interval is small enough, the assumption is that the desired point is found in the middle of the final subrange.

The main difference between LS and GSS is that in the former the whole available power range is always explored. On the other hand, in the latter the next step, which narrows the power caps range, is always based on the previous two steps. The decision about rejection of some unexplored subrange is final so there are always some power cap subranges which were not tested in GSS.

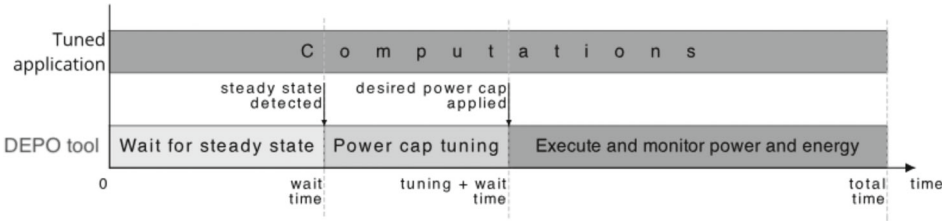

FIGURE 6 Dynamic power capping phases

Another intuitive observation is that, assuming the same time period T for a single power cap test, the GSS might be much faster in finding global optimum than LS. Obviously, LS also might be faster than the GSS method because it is possible that one might choose some large (more than 10%) power decrement value so the algorithm finishes in just a few iterations. However, in such a case, the selected power cap would probably result in the target metric value being more off the globally optimal value.

Exemplary experimental results for a single testbed application run with DEPO with both search algorithms are presented in Section 6.3.

5.4 | Dynamic target metrics

The DEPO separates the method of power cap search from the chosen optimized target metric. The tool might then work with any user defined target function, that might be evaluated at runtime, based on available system readings. The tool currently implements three target metrics as described in Section 4 so the optimization functions implement Equation (13) (E), Equation (15) (EDP), and Equation (19) (EDS).

6 | EXPERIMENTAL RESULTS

In this section we present the configuration and the results of our experiments. Section 6.1 describes the testbed systems used in the experiments. Section 6.2 presents all the tuned applications used in the experiments. Before the actual evaluation of the DEPO tool we performed a series of preliminary experiments in order to find proper configuration parameters for the tool, especially the time window for probing current power usage during execution of the application. Section 6.3 presents the results of these tests. Then, in Section 6.4, we present a comparison of used dynamic power cap searching algorithms: LS and GSS. Eventually, in Section 6.5, we presented the results of the extensive evaluation of the tool with the finally selected search algorithm (GSS) using the well-known NAS parallel benchmarks suite.[2]

6.1 | Testbed systems

The following tests were conducted in two environments, specified in Table 3, representative of modern CPUs for desktop (Testbed 1) and server (Testbed 2) systems respectively. The platforms differ specifically in: HyperThreading technology (not present in Testbed 1 and present in Testbed 2), numbers of physical processors (1 in Testbed 1, 2 in Testbed 2), number of cores versus clock frequency (higher frequency and fewer cores in Testbed 1, lower frequency and considerably more cores in Testbed 2).

From the power consumption perspective the two considered platforms differ in thermal design power (TDP) as well as default and maximal available power cap level. Testbed 1 has TDP at 95 W but max power cap is 255 W with default

TABLE 3 Testbed configurations

Testbed system	Testbed 1—Coffee Lake	Testbed 2—Haswell-EP
CPU model	Intel® Core™ i7-9700K CPU 3.60 GHz Coffee Lake	2 × Intel® Xeon® E5-2670 v3 CPU 2.30 GHz Haswell-EP
Number of physical/logical cores	8/8	24/48 (2 × 12/2 × 24)
System memory size (RAM)	32 GB (4 × 8 GiB)	128 GB (8 × 16 GiB)
Operating system	Ubuntu 18.04.4 LTS	Ubuntu 16.04.7 LTS
Compiler/version	gcc v. 7.5.0 (Ubuntu 7.5.0-3ubuntu1 18.04)	gcc v. 7.5.0 (Ubuntu 7.5.0-3ubuntu1 16.04)
Thermal design power (TDP)	95 W	240 W (2 × 120 W)
Default power cap	255 W	240 W (2 × 120 W)
Max available power cap	255 W	480 W (2 × 240 W)

power cap set to the maximal value of 255 W. Testbed 2 has TDP at 240 W (2 CPUs × 120 W), but the default power cap is set to 240 W (2 × 120 W) while the maximal available power constraint is 480 W (2 × 240 W).

6.2 | Testbed applications

For the testing purposes we have used eight different workloads which are representative, parallel applications. Three of the applications are custom developed and five of them are taken from the well-known in HPC benchmarking NAS parallel benchmarks suite.[2]

Similarly to our previous work,[45] the first three custom build applications solve typical computational problems with the support of commonly used in HPC, open source tools and libraries, like OpenMP or GCC. Their implementation is based on the shared memory processing paradigm with threads used for providing parallelization and scalability. The minibenchmarks' source code is published and available online*.

In this article, we assume the application execution to be uniform, that is, during the application execution the compute load is approximately the same: intensity of the usage of the system components like the memory or the compute cores does not change significantly in time. We consider an exception for a beginning phase, when due to initialization (e.g., data loading or preparation) instructions differ from the following computational phase. The proposed testbed applications follow this assumption, however we are aware of other possible models of the application execution, for example, with iterative computations/communication phases, and consider their analysis in future work.

The first application (denoted INT) solves a numerical integration over a given range, specified by the input arguments and for an arbitrary chosen function: $f(x) = \frac{1}{1+x}$. The range is split between the threads, and each thread calculates its own part. Afterwards, the total sum is computed by invoking a reduction operation, and the final result is printed out. The precision of the calculations is controlled by a number of subpartitions to be integrated, provided as an input argument.

Figure 7 presents power consumption measurements for the proposed numerical integration testbed application executed on both testbed systems: Haswell-EP and Coffee Lake. The application starts performing at the target power consumption level almost immediately after it has been launched. The power consumption is steady during the whole running time.

The second application (denoted HEAT) simulates heat distribution in a square room using a simplified 2D model, based on the example presented by Sanders et al.[56] The area is split into $N \times N$ cells, where one cell in a corner is a heater. The simulation is performed in iterations where the current iteration uses the input data produced as the output of the previous one. Thus, each cell state can be calculated independently by a separate computation unit (i.e., core). Such an approach can be used to achieve high scalability, however it also requires keeping an additional memory buffer with the previous iteration's output data.

Figure 8 presents power consumption measurements for the proposed heat distribution simulation testbed application executed on both testbed systems: Haswell-EP and Coffee Lake. The HEAT application's power profile is more complex than the previous numerical integration. Although on the server Haswell-EP system the application starts to perform at the target power consumption level almost immediately after launch, the power profile of the same application run on the desktop Coffee Lake system varies in its power demand during the whole execution period. The HEAT application

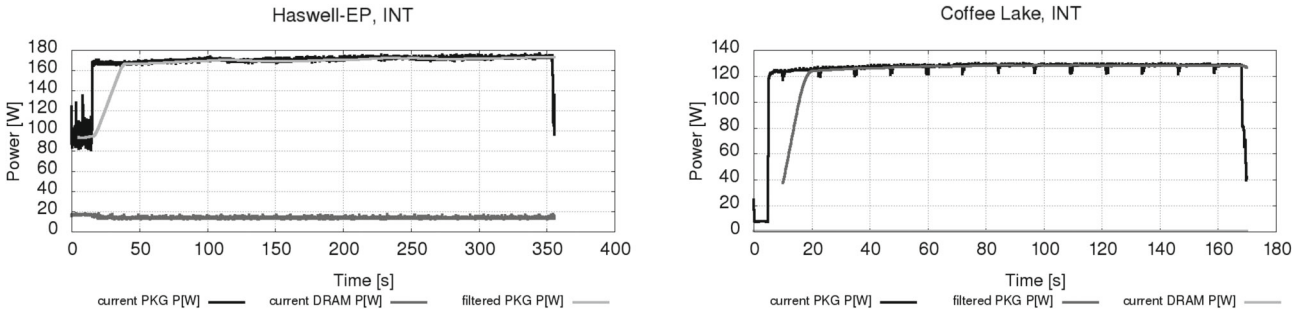

FIGURE 7 Exemplary power profile of INT testbed application on Haswell-EP (left) and Coffee Lake (right) systems without power caps.

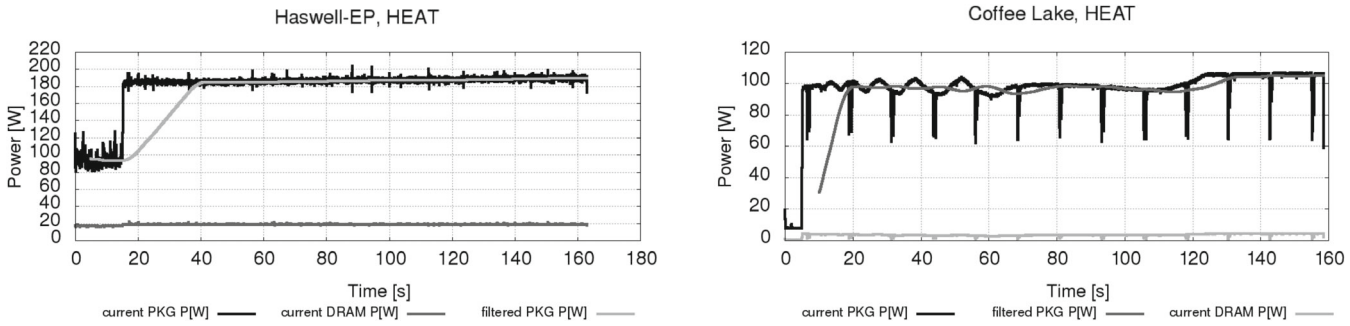

FIGURE 8 Exemplary power profile of HEAT testbed application on Haswell-EP (left) and Coffee Lake (right) systems without power caps.

executed on the latter system seems to reach target power demand close to 100 W immediately after launch but there are clear low power moments during the computations when the power consumption drops to 60 W.

The third application (denoted FFT) is a typical fast Fourier transform implementation, with the Radix-2 parallel algorithm supporting decimation-in-time approach.[57] The input data, consisting of a vector with N complex numbers, is automatically generated, shuffled and then used to perform $\log_2 N$ parallel iterations. For the testing purposes the computations are repeated multiple times, according to a given parameter and the input data is overwritten by the results, thus the computations use only one memory buffer of size N.

Figure 9 presents power consumption measurements for the proposed FFT implementation testbed application executed on both testbed systems: Haswell-EP and Coffee Lake. The third application presents a non-trivial power consumption profile on both testbed systems. The FFT application has a clear startup phase before the power demand for the computations reaches target level. FFT also presents periodical power demand drops which can differ from the regular power demand up to 75%.

The above applications were implemented in the C language v. C99, using OpenMP with the GCC v. 4.88 compiler and with double precision floating number support. The execution code was optimized with the −O3 parameter and default OpenMP configuration (for the thread number, affinity and the computation partitioning) was used, without further tuning in the testbed environment.

As an additional representative HPC workload we selected five applications from the well known NAS parallel benchmarks (NPB) suite which is a set of programs designed to evaluate the performance of parallel supercomputers.[2] From the kernels subset we selected CG which is characterized by irregular memory access and communication and also multi-grid on a sequence of meshes (MG) which realizes long- and short-distance communication and is memory intensive. We did not use the EP and discrete 3D fast Fourier transform (FT) kernels as they are basically the same as two of our tuned applications considered earlier in this section: INT and FFT respectively. On the other hand, the execution time of the integer sort (IS) kernel was too short to make use of the DEPO tool on both testbed systems.

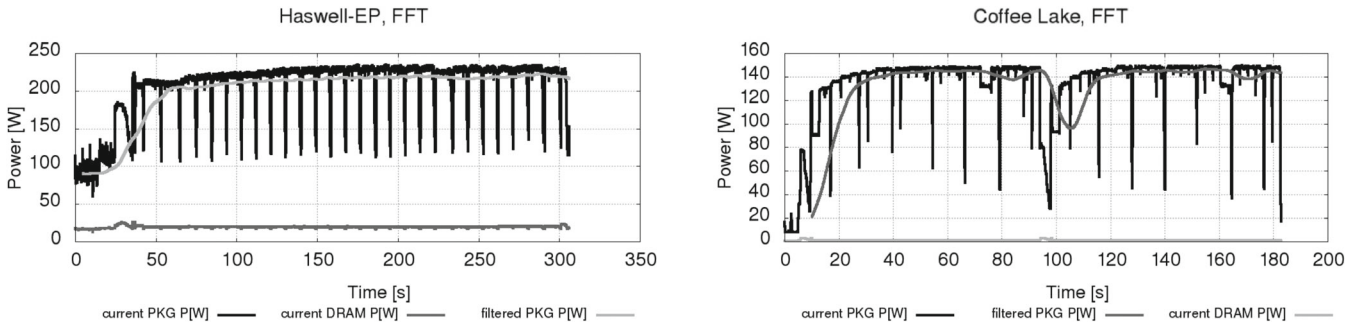

FIGURE 9 Exemplary power profile of FFT testbed application on Haswell-EP (left) and Coffee Lake (right) systems without power caps.

From the subset of pseudo applications we used all three available programs which were block tri-diagonal solver (BT), scalar penta-diagonal solver (SP), and lower-upper Gauss–Seidel solver (LU).

All five selected NPB applications were written in Fortran using OpenMP. The code was compiled with a `gfortran` compiler, version 7.5.0 with default for the NPB `-O3` optimization level and memory model (`-mcmodel`) set to `medium`. Power profiles for these applications and the two systems are shown in Figures 10–14.

The NPB applications can be run with different input data sizes which are defined as benchmark classes. For most of the tests we used class D which is the first class of large test problems group. For the purpose of the DEPO tool validation we considered only the total execution time of the tuned application and for almost all of the tests class D satisfied our minimal requirement of total execution time longer than 250 s. Only for the MG kernel executed on the Haswell-EP we had to make an exception and accept the application's execution time for the class D smaller than 138 s as the use of class

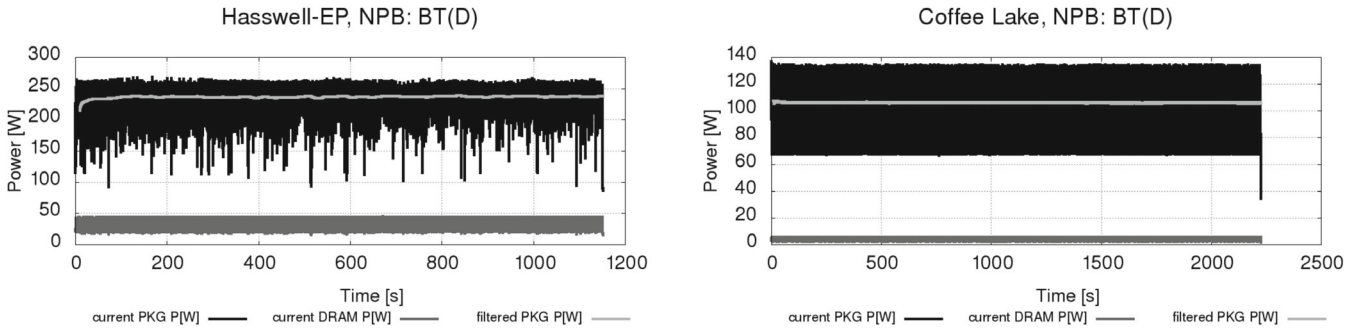

FIGURE 10 Exemplary power profile of NPB: BT application executed for class D on Haswell-EP (left) and Coffee Lake (right) systems without power caps.

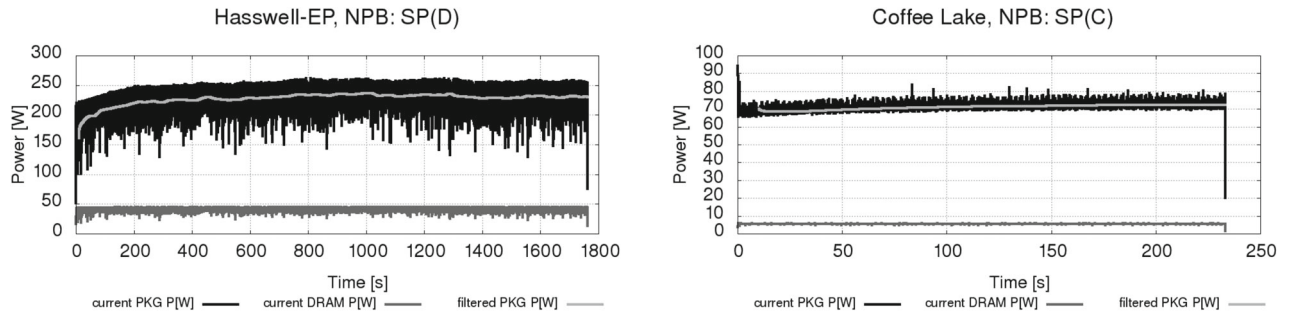

FIGURE 11 Exemplary power profile of NPB: SP application executed for class D on Haswell-EP (left) and for class C on Coffee Lake (right) systems without power caps.

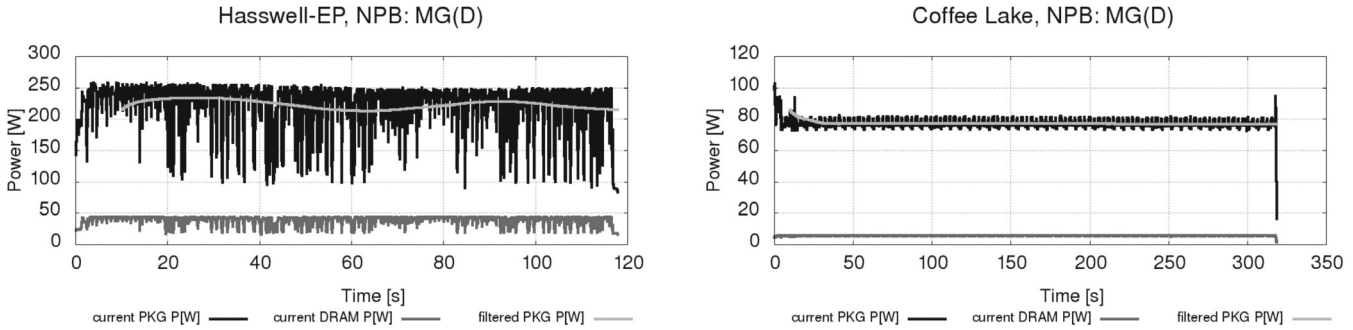

FIGURE 12 Exemplary power profile of NPB: MG application executed for class D on Haswell-EP (left) and on Coffee Lake (right) systems without power caps.

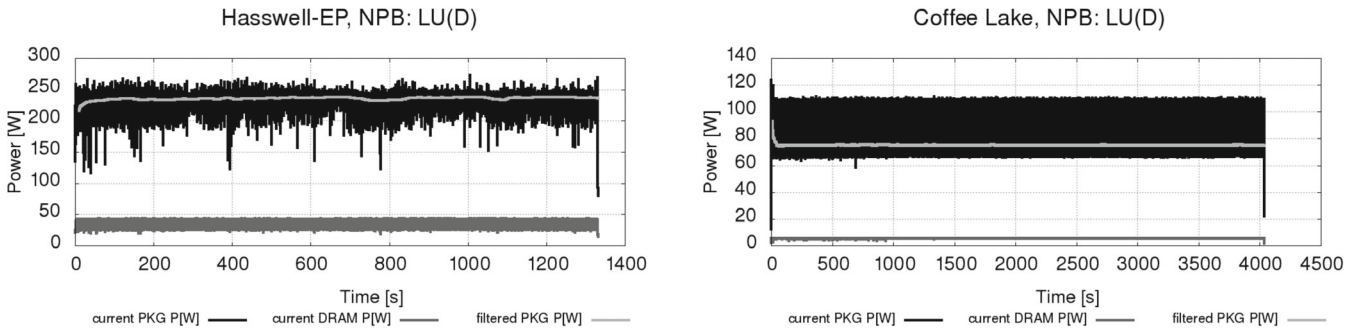

FIGURE 13 Exemplary power profile of NPB: LU application executed for class D on Haswell-EP (left) and on Coffee Lake (right) systems without power caps.

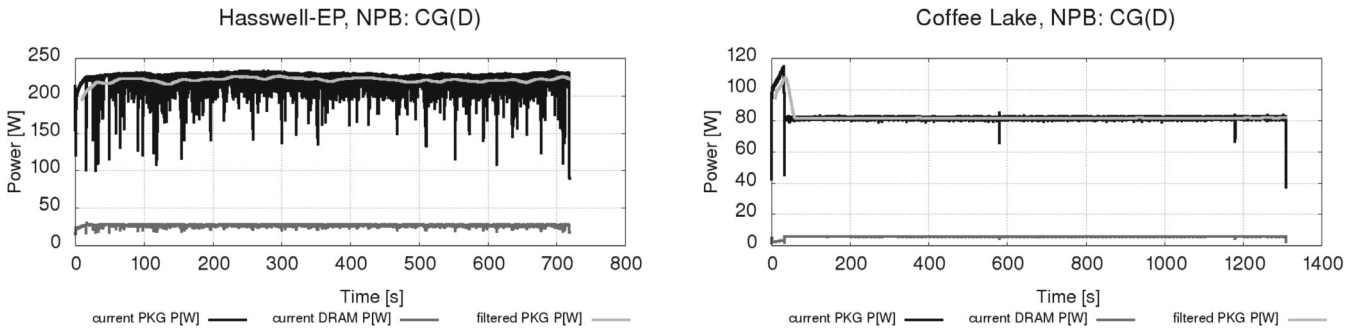

FIGURE 14 Exemplary power profile of NPB: CG application executed for class D on Haswell-EP (left) and on Coffee Lake (right) systems without power caps.

E problem size was blocked by insufficient memory size available on Haswell-EP system. On the other hand, for the SP application run on the Coffee Lake system we had to decrease the problem size to class C which resulted in 253 s of the total execution time while the same SP application executed with class D problem size ran for over 5200 s for the default system configuration. Since the difference in execution times of the SP application on the Coffee Lake system were so wide for the C and D classes we decided to use the smaller class as the total execution time for C class is closer to the average execution time of every other application for both systems.

6.3 | Preliminary experiments for tool parameters configuration setup

Figure 15 presents a typical execution of the FFT application on both Haswell-EP and Coffee Lake testbed systems with GSS control algorithm while Figure 16 presents the same application and systems setup for the LS algorithm. Both DEPO executions evaluated the EDP target metric so the goal was to minimize the energy consumption and execution time product.

Both algorithms detect the available power caps range properly. For the Haswell-EP the available power caps range was between 95 and 245 W while for the Coffee Lake system the range started with 15 W (idle CPU demand) and 200 W (maximal available power cap). The 300 W power cap value on each of the figures represent the default systems configuration.

A GSS's characteristic feature is that it starts the search using two points between the given [min, max] range. The two points in the middle are determined by the formula $\min + (\max - \min) \div \phi$ and $\max - (\max - \min) \div \phi$ where $\phi = 1.618$. The $\phi = 1.618$ is the famous *Golden ratio* number. Using *phi*, with its unique feature, allows for smart selection[55] of the points between given [min, max] range in a way that the new range calculated in the next iteration will reuse one of the previous inner points as either new min or new max. Therefore, one of the next examined power caps is always based on

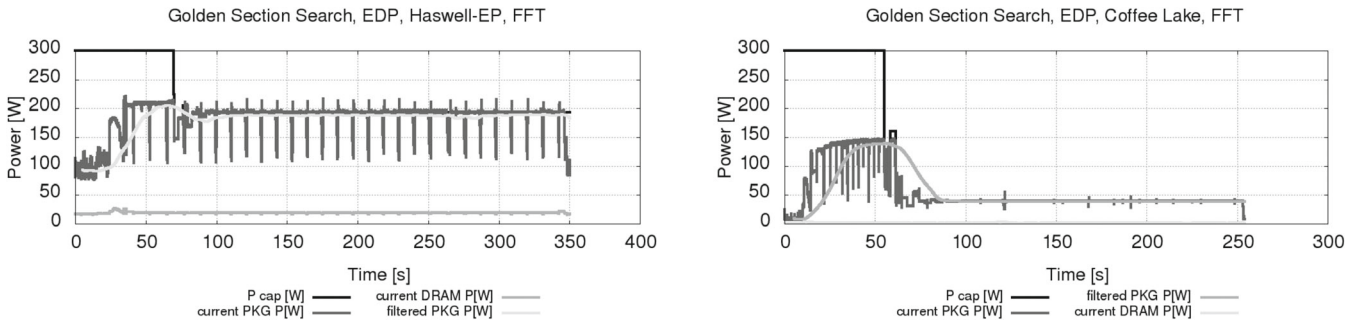

FIGURE 15 Typical power log for GSS with EDP target metric recorded for FFT application on Haswell-EP (left) and Coffee Lake (right) systems.

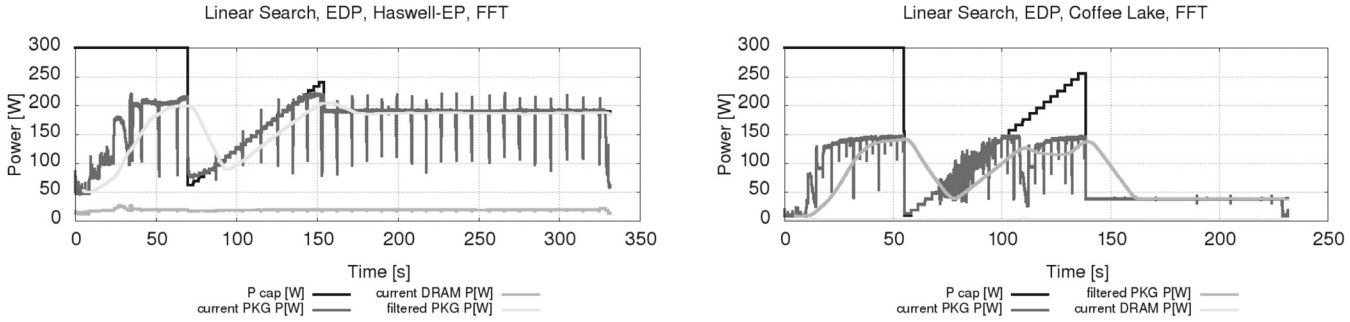

FIGURE 16 Typical power log for LS with EDP target metric recorded for FFT application on Haswell-EP (left) and Coffee Lake (right) systems.

one of the two previous candidates being the inner subrange constraints. The powercap subrange is iteratively narrowed down until it reaches the width less than 1% of the available power caps range.

On the other hand, LS evaluates the target metric for the whole available power caps range with fixed steps increasing the power cap by 5%. Exploring the whole range with a relatively small increment allows for finding the power cap that better fits the target metric's global optimum, but the tuning phase duration takes a significant amount of time.

Results of preliminary experiments motivated us to propose the tool configuration parameters that we used in all following experiments that we present in Section 6. We decided to use time window $T = 6400$ ms, which means that each power cap is set for 6400 ms, and based on the energy consumption and the number of instructions measured within this period the target metric is evaluated. The values of T that we tested for were: 100, 200, 400, 800, 1600, 3200, and 6400 ms. The longer time window T we set, the more stable the results are but also the longer the tuning period takes. The value of 6400 ms is the shortest T for which we have observed stable results with standard deviation from 5 test runs smaller than 10% for most of the tuned applications.

We set the power cap fixed step for LS to 2% and the final subrange width for GSS was chosen as 1% of the available power limits range.

Preliminary tests have also shown that the k coefficient, needed to evaluate *EDS* target metric, does not have a significant impact on the DEPO tool algorithm, since the tool searches the minimal value of the target metric anyway. We confirmed that performing preliminary tests with k values: 1.25, 1.5, and 2. The test showed that the minimal value for each version of dynamic *EDS* metric is found always for the same power cap so we decided to stay with $k = 2$.

6.4 | Comparing linear search with golden section search

The proposed tool supports two approaches for searching for the best power configuration: LS and GSS. Each of them can be used to optimize the following target metrics: E, EDP, EDS. The tool supports these search algorithms and metrics out-of-the-box, however is not limited to them and can be easily extended with additional approaches.

The initial evaluation of the proposed dynamic power capping solution was performed for two testbed systems: Haswell-EP and Coffee Lake Intel architectures, where we have run experiments using three self-prepared applications (INT, HEAT, FFT). For each system-application pair we have run the DEPO tool with a tuned application for both built-in search types: LS and GSS, and for all currently implemented target metrics: E, EDP, EDS.

Figures 17–22 present a series of results for INT, HEAT, and FFT applications run on both systems for both search types and all three target metrics. The results are normalized based on the reference application run executed with the default system power cap settings. One result group represents one search type-target metric pair and contains the following measurements: average power, energy (E), execution time result, total energy and total time product (EDP), and EDS evaluation for the $k = 2$ coefficient. Each bar on the charts represents the average result of 10 experiment runs. Related values of the standard deviation are presented as T signs over the bars.

First experiments were performed for the numerical integration (INT) testbed application. In Figure 17, for the Haswell-EP testbed system, we can observe interesting results with non-default system setup for minimization of all three target metrics using the GSS algorithm, with energy savings 10%, 7%, 8% and the execution times extended by 11%, 2.5%, 2% for E, EDP, EDS metrics, respectively. On the other hand, LS provided better than default (no power cap) configuration, only for the total energy consumption (E), where savings were on 6%–7% level. In Figure 18, for the Coffee Lake testbed system, we can observe that for all three target metrics the DEPO tool was able to find such system settings that

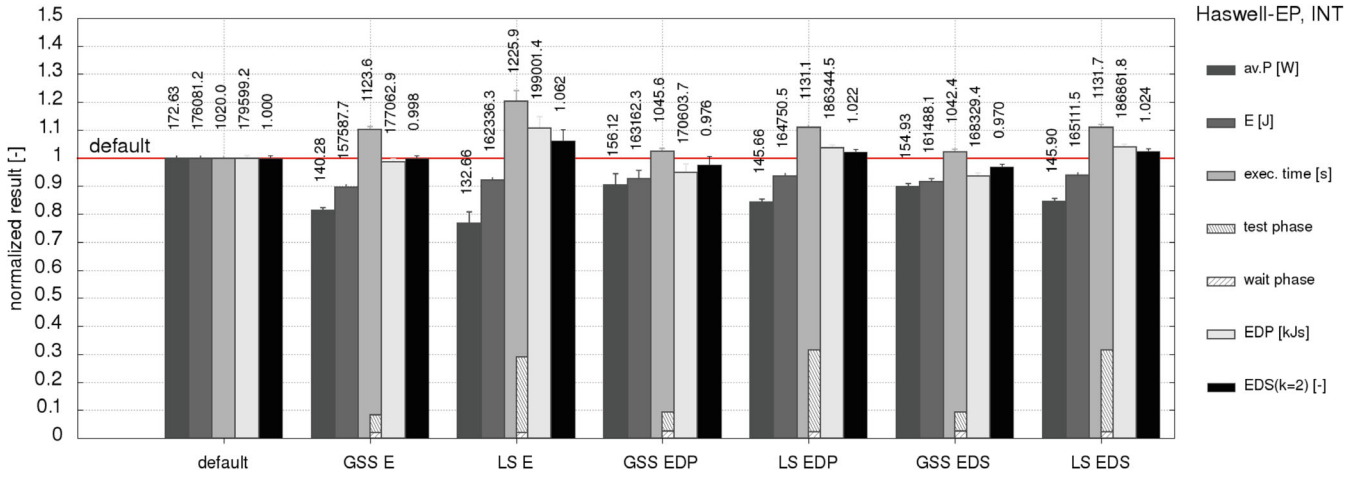

FIGURE 17 Results for INT application run on Haswell-EP system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

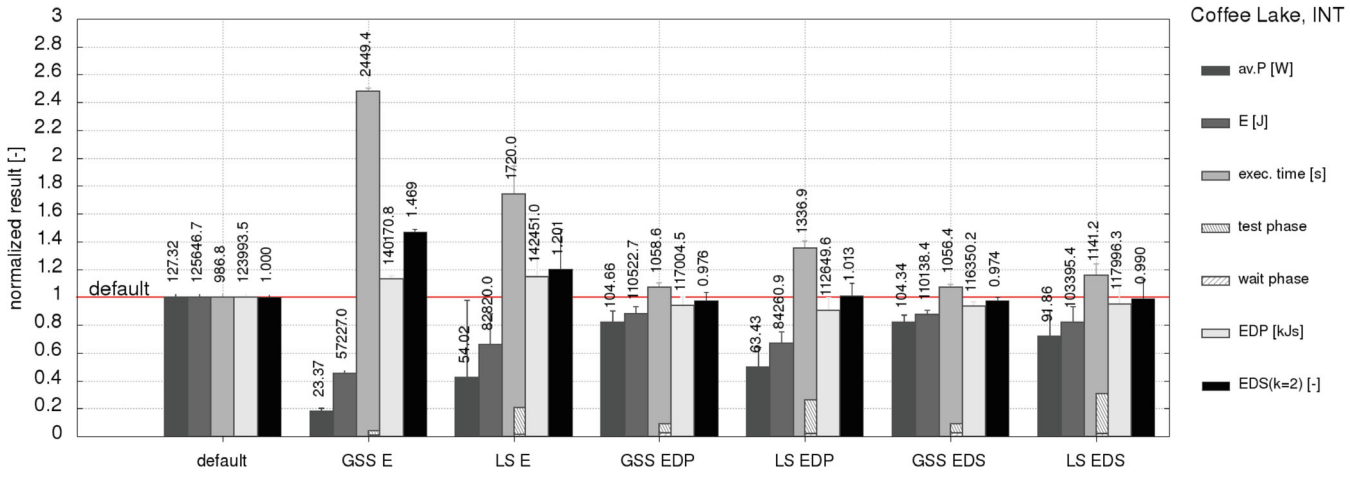

FIGURE 18 Results for INT application run on Coffee Lake system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

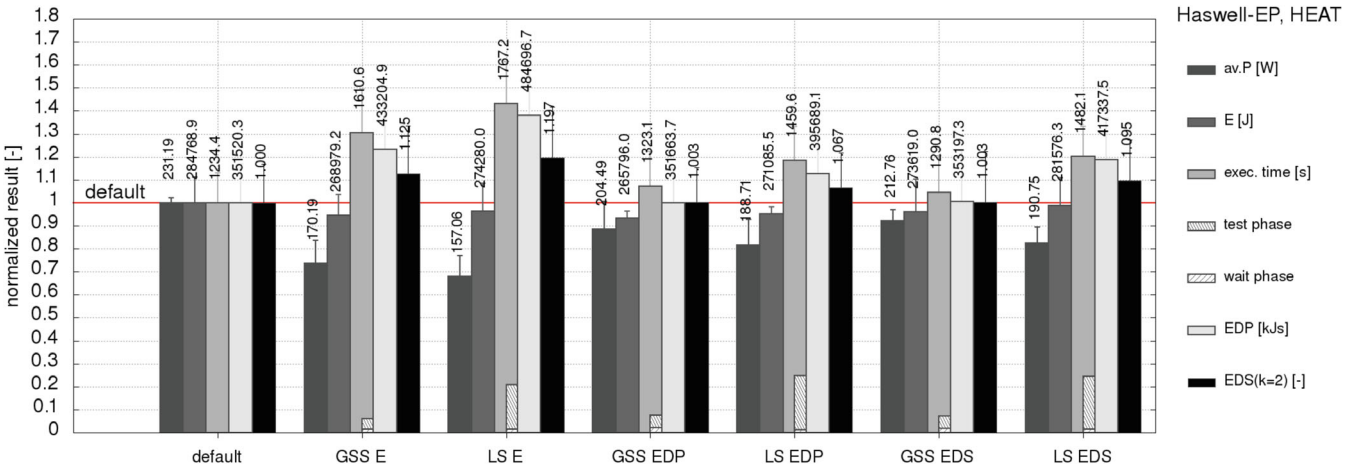

FIGURE 19 Results for HEAT application run on Haswell-EP system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

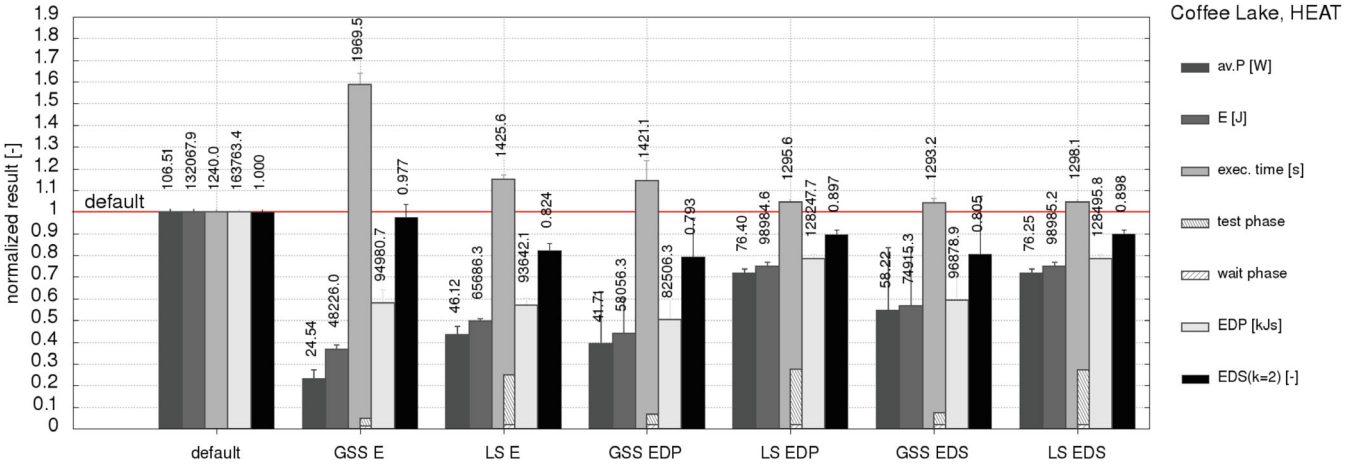

FIGURE 20 Results for HEAT application run on Coffee Lake system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

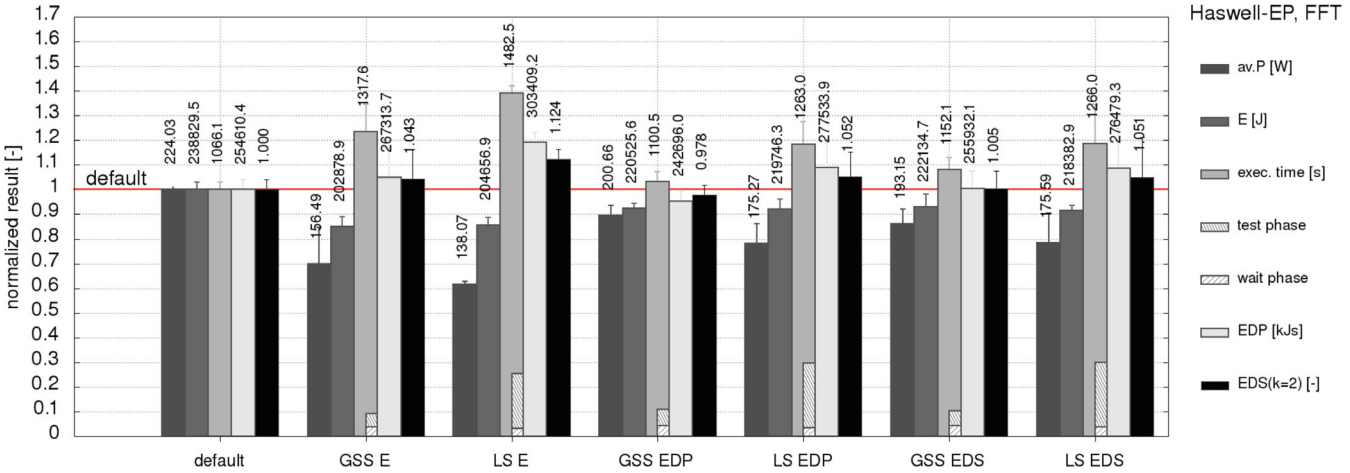

FIGURE 21 Results for FFT application run on Haswell-EP system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

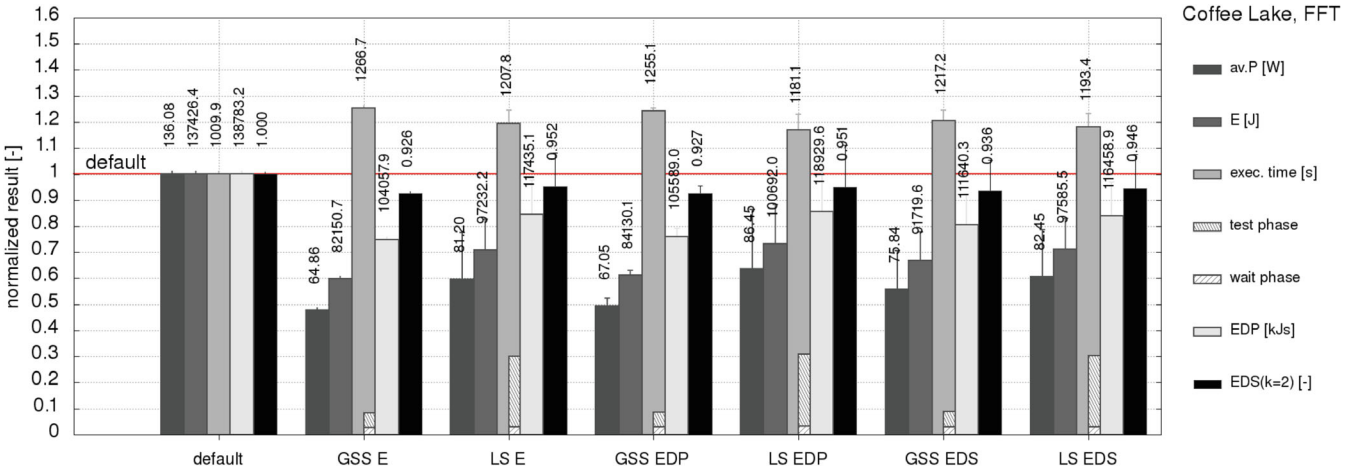

FIGURE 22 Results for FFT application run on Coffee Lake system with DEPO for LS and GSS algorithms and three target metrics: E, EDP, EDS.

the resulting parameters measured for dynamically tuned application execution were better than the default system settings. Using the GSS method we reach up to 54.5%, 12%, 12% energy savings with execution times extended by 148%, 7%, 7% for E, EDP, EDS metrics, respectively. While again with LS the savings are much lower, up to 34%.

Figure 19 and 20 present results of simple heat distribution simulation (HEAT) for Haswell-EP and Coffee Lake testbed systems respectively. For the former, the minimization of E metric results in 4% and 5.5% of energy consumption reduction, while execution time increase is close to 30% and almost 43% for LS and GSS, respectively. Similarly for EDP and EDS, GSS shows better results providing lower energy consumption and performance drops than LS. For the Coffee Lake testbed system, the energy savings caused by both search methods are even larger, for example, for E metric, LS and GSS respectively provides 50% and 63% energy savings, with 15% and 59% increase of the execution time. Thus, in this the GSS seems to work better than LS.

Finally, Figures 21 and 22 present results of an application performing FFT calculations for Haswell-EP and Coffee Lake testbed systems respectively. For the FFT application we can observe that the final results have the highest standard deviation error (up to 16%) which means that the result of dynamic power capping is the least stable of all three testbed applications. This is caused by the FFT application power profile characterized by variable power consumption levels which implies that the steady state detection is not deterministic for every execution and can affect the start time of the tuning phase. However, the results show the significant decrease of the energy, up to 23% (LS) and 39% (GSS), or even 29% (LS) and 40% (GSS) for Haswell-EP and Coffee Lake systems, respectively.

We would like to emphasize, that the testbed applications were selected and set up to generate a stress over different system components. The INT (the numerical integration) application, with the applied configuration, represents a so-called EP computational problem. Each thread executes compute intensive code and works almost independently, using designated data, which is stored in the CPU cache memory. Thus, any larger limitations of the power level cause a great impact on the performance, for example, for the Coffee Lake CPU, gaining 54.5% energy savings with an 18% cap of the initial power level, increases the execution time up to 248% of the original value. Thus, for such a case, the majority of the multi-objective optimized results are found around the default power setup.

FFT is also a compute intensive application, but it requires many more synchronizations between the cooperating threads. Thus, even when the data stays in the cache memory, possible waiting cycles can lead to much better trade-off opportunities than in the INT case. And indeed, the experimental results show, that the applied power limitations provide quite high energy savings (up to 40%) with moderately low performance drop (16%–25%). In this case, the two objective optimization, with various metrics, provides a wide spectrum of the in-between solutions.

Finally, the HEAT application with its focus on memory intensive operations, shows the highest flexibility for the optimization. The low occupation of the CPU, while waiting for main (DRAM) memory data transfers, causes that the even high power limitations provide acceptable trade-offs. We can observe the energy savings reaching over 56%, while the performance loss is below 14% (using the EDP metric on the Coffee Lake CPU). For such applications the DEPO tool presents the greatest opportunities to facilitate green computing oriented implementations.

Summarizing the results, we can notice that the more compute intensive cases are harder to improve using our method, and the ones with lower focus on calculations provide more possibilities to introduce energy-performance trade-offs. Moreover, comparing the LS algorithm with GSS we can observe and conclude:

1. For most of the cases LS with a 2% power cap step and GSS with a 1% target power cap range width lead to similar power-cap settings. This might change in favor of any of them if we decided to use different DEPO settings.
2. There were just a few cases where we observed what we expected after preliminary research—the advantage of LS due to full power cap range exploration, for example, EDP minimization for the INT application on Coffee Lake where LS had a chance to select a lower power cap with an even better EDP value than GSS.
3. For LS with a relatively small power cap step of 2% and time window $T = 6400$ ms the tuning phase takes over 300 s which is almost 30% of the selected testbed applications' execution times. Such a long tuning phase with an additional wait-for-steady-state phase might impact the overall result of dynamic power capping.

Thus for further usage we recommend GSS, as a default choice.

Additionally, Appendix A justifies using RAPL against PDU measurements for the FFT, HEAT, and INT applications on the Haswell system, taking into consideration the measurement window and averaging proposed in our method.

6.5 | Validation of the proposed tool using NAS parallel benchmarks workloads with golden section search algorithm

This section presents experiments that extend the validation of the proposed tool implementing dynamic power capping method described in this article with a well-known NAS Parallel Benchmark suite.[2] The rationale for the initial set of tests was investigation of tuning phase parameters, especially deriving the time window length as well as performing comparison of LS and GSS. Based on the comparison of the latter summarized above, in particular the generally shorter test phase of GSS/fast exploration of the power limits configurations, GSS was selected for an extended set of following tests. The GSS algorithm finds the power cap performing the final selection in less than 10 steps which results in the shorter test phase duration in comparison to LS (less than 65 s when using the time window of $T = 6400$ ms). With the above exploration method and the time window, along with the minimum 250 s total execution time requirement we obtain, in the worst case scenario (for the shortest applications), the ratio of the tuning phase to the total execution time close to 25% which seems to be acceptable considering the potential impact of the tuning phase on the total energy consumption and performance.

For each tested NBP kernel (application): CG, MG, BT, SP, and LU, run on both Haswell-EP and Coffee Lake systems we evaluate all three target metrics (E, EDP, EDS) considered in previous sections. Figures 23–27 present a series of results for each of the five NPB applications run on both testbed systems with the GSS algorithm and all three target metrics. Each result is an average of 10 test runs. The results are presented as bars normalized with default system configuration results as a base. Each bar has its absolute value placed above it. The standard deviation for each average result is marked on the top of each result bar.

6.5.1 | CG application

Figure 23 presents the results obtained for both Haswell-EP and Coffee Lake systems for the tuned CG application. For both systems class D problem size was used which resulted in 910 s of total execution time on Haswell-EP and 1305 s on Coffee Lake both in default system configurations. This implies the tuning phase to total execution time ratio below 7.5% for the Haswell-EP system and below 5% for the Coffee Lake system.

We can observe that for Haswell-EP minimization of energy (E) results in 12% savings on energy with almost 36% of performance degradation. The two other target metrics (EDP, EDS) minimization result in similar system configuration selection (target power cap in range 189–193 W) and consequently similar energy savings close to 5% with 3%–4% performance drop in both cases. We suspect that the minimal differences in target power cap selection associated with specific metric minimization might result in some noticeable diversity in the final results and possible improvements of the EDP and EDS metrics if the application's total execution time was for example, 3× longer.

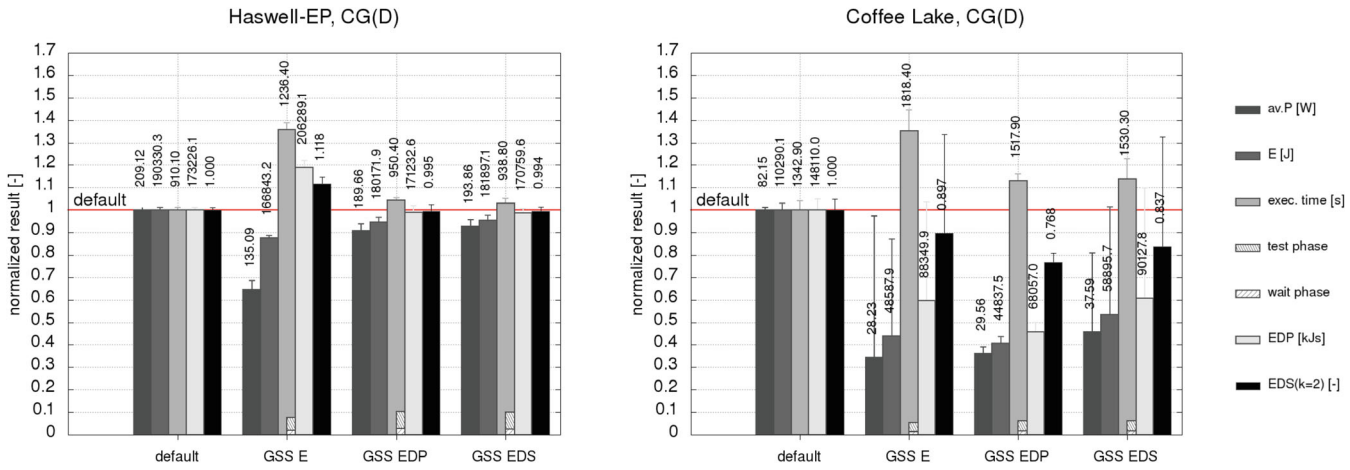

FIGURE 23 Results for CG NPB application run on Haswell-EP (left) and Coffee Lake (right) system with DEPO for GSS algorithm and three target metrics: E, EDP, EDS.

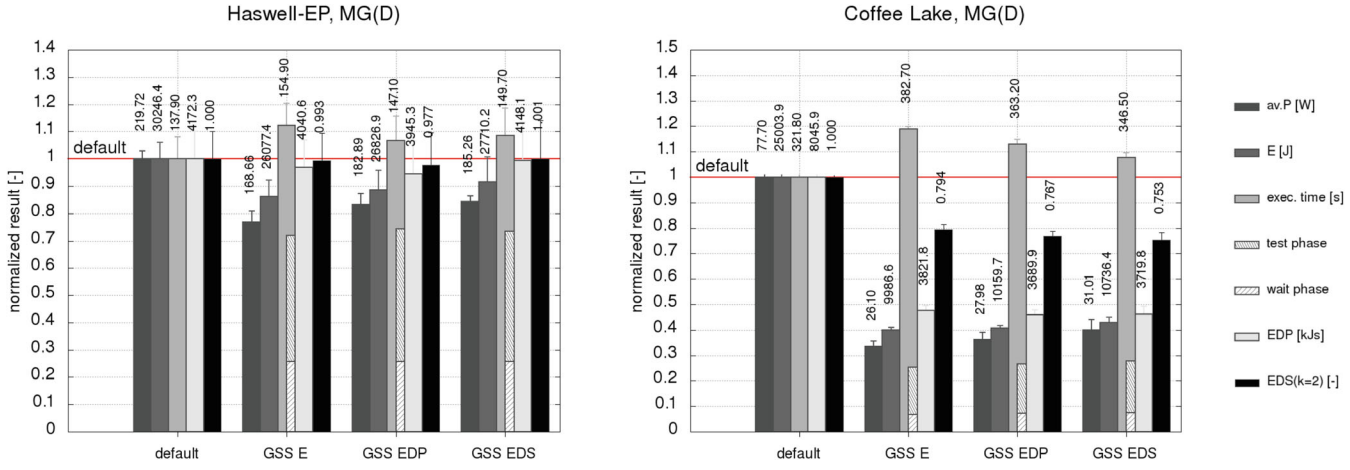

FIGURE 24 Results for MG NPB application run on Haswell-EP (left) and Coffee Lake (right) system with DEPO for GSS algorithm and three target metrics: E, EDP, EDS.

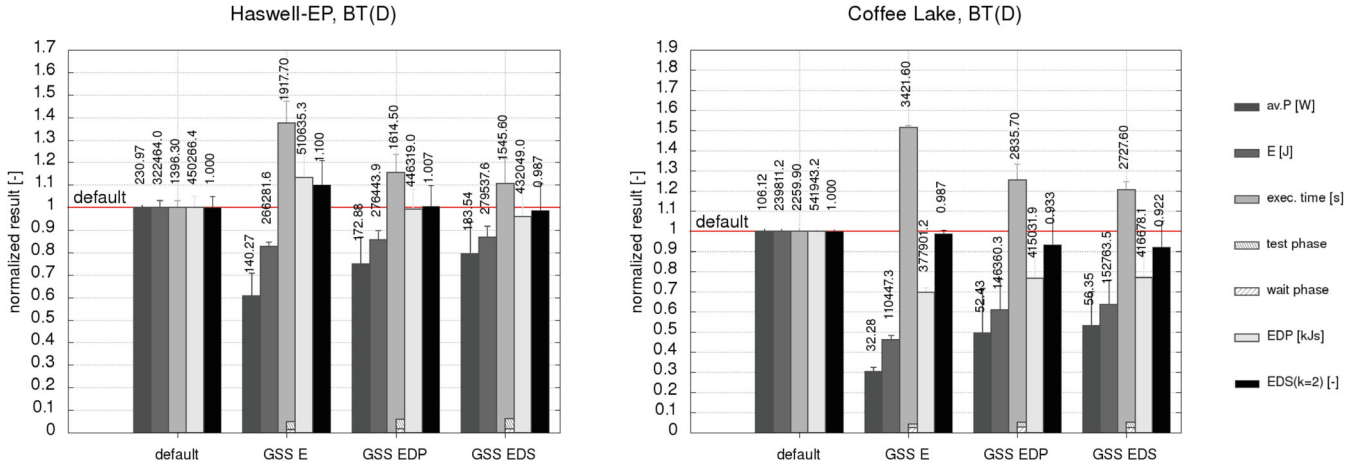

FIGURE 25 Results for BT NPB application run on Haswell-EP (left) and Coffee Lake (right) system with DEPO for GSS algorithm and three target metrics: E, EDP, EDS.

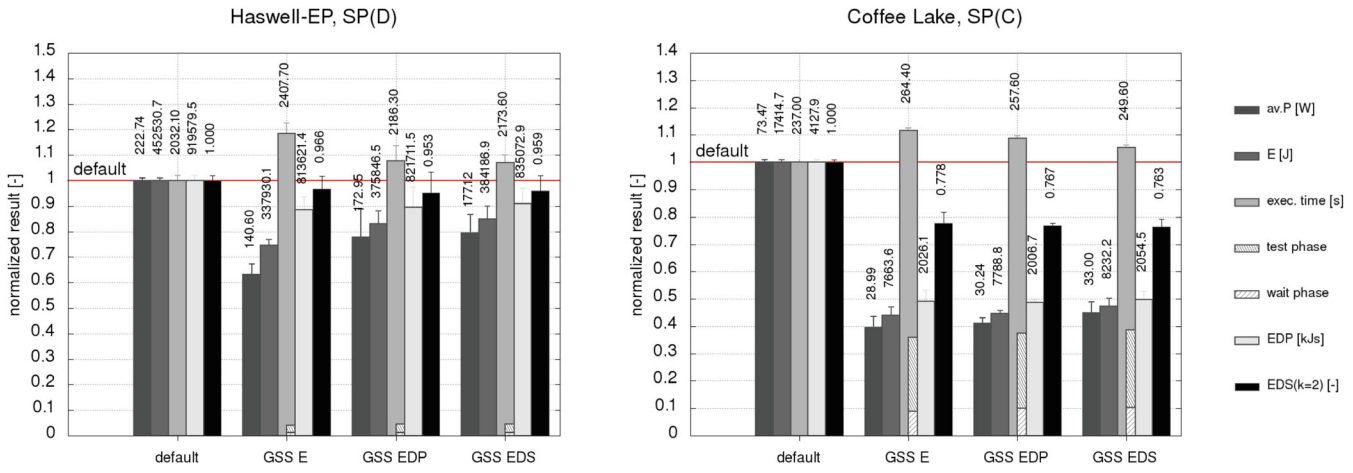

FIGURE 26 Results for SP NPB application run on Haswell-EP (left) and Coffee Lake (right) system with DEPO for GSS algorithm and three target metrics: E, EDP, EDS.

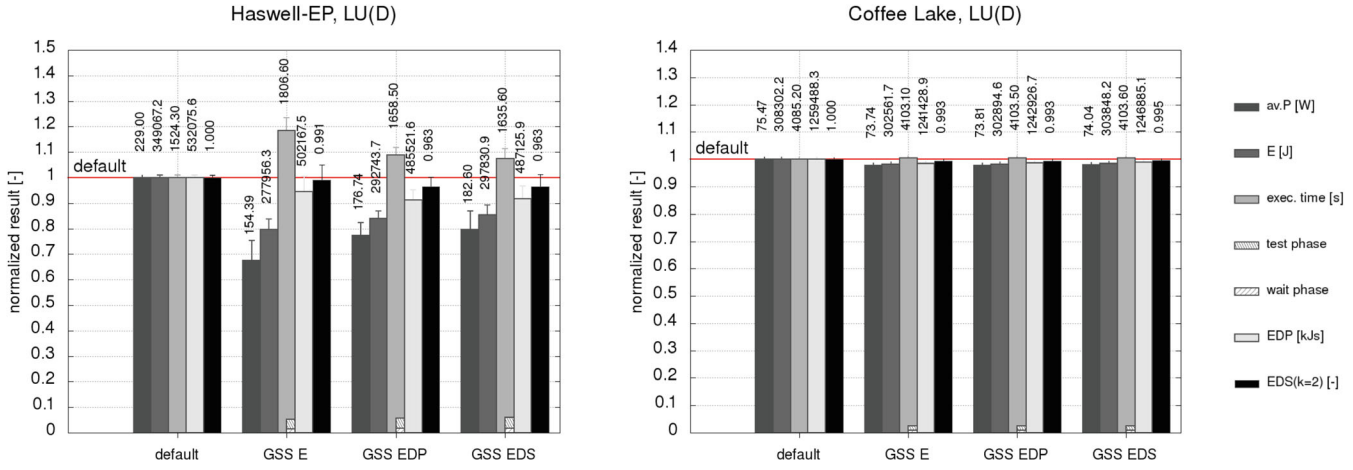

FIGURE 27 Results for LU NPB application run on Haswell-EP (left) and Coffee Lake (right) system with DEPO for GSS algorithm and three target metrics: E, EDP, EDS.

For the Coffee Lake system we can observe much more interesting results than for Haswell-EP. Minimization of the E metric for the CG application resulted in 56% of energy savings with only 35% of the performance drop. EDP metric minimization leads to 54% lower value of energy-time product which implies 59% of energy consumption reduction with only 13% of total execution time increase. Minimization of the EDS metric results in 23% improvement of its value which indicates 46.5% of energy savings with only 14% of performance drop. The relatively high standard deviations ($\sigma = 43\%–48\%$) for the target power cap, total energy consumption and E/EDS metrics with only $\sigma = 9\%$ for total execution time were probably caused by only one extreme result. In one of ten test runs when minimizing the E/EDS target metric the GSS algorithm selected the target power cap close to the default one which did not cause any performance degradation but impacted the power and energy consumption average value.

6.5.2 | MG application

The results obtained for the MG NPB kernel run on both Haswell-EP and Coffee Lake systems are presented in Figure 24. The MG application was run on both Haswell-EP and Coffee Lake systems with class D problem size which resulted in average total execution time for default system setups equal to 137.9 s for the server system and 321.8 s for the desktop

CPU. This implies test phase duration to total execution time ratio around 47% and 20% for Haswell-EP and Coffee Lake respectively.

For the MG kernel run on Haswell-EP we can observe that for the first two target metrics (E and EDP) the results of their minimization are quite good even despite the relatively long tuning phase impacting the overall result. For the E metric we were able to save almost 14% of energy with 12% of performance drop. The EDP metric value was improved by 5% which implied 11% lower energy consumption with 6.5% longer total execution time. The EDS metric minimization ended with its average value exactly the same as the default reference value but with 14% standard deviation and 8% of energy savings. This allows us to expect that for the same application executed for a much longer time the overall results with selected power cap (185 W) might be much more impressive.

For the Coffee Lake system the DEPO tool run with MG kernel allowed for significant improvements of each of the three target metrics (E, EDP, EDS). Minimizing the E metric resulted in over 60% lower total energy consumption with performance degradation less than 19%. For EDP minimization the energy-time product reduction reached almost 55% which was followed by over 59% of energy savings and 13% of total execution time increase. The EDS metric minimization resulted in almost 25% lower value of this metric which indicated over 57% of energy consumption reduction with only 8% of performance drop.

6.5.3 | BT application

Figure 25 presents results obtained for the BT NPB application run on both Haswell-EP and Coffee Lake testbed systems. The problem size used for both systems was class D which resulted in average total execution time for default system configurations equal to 1396 and 2259 s for Haswell-EP and Coffee Lake respectively. The average test phase to total execution time ratios were close to 5% for the first testbed system and less than 3% for the second system. However, the BT application is an unique workload type among all five tested NPB programs as the power characteristic is variable and unstable. Similarly to the FFT application considered in previous section, the BT NPB program characterized with a high amplitude of power demand during whole application execution which makes finding the steady state of power consumption difficult. This indicates that the steady state detection is not consistent among all 10 test runs and also the wait phase is much longer than the test phase. As a consequence of that, the results obtained for the BT application have noticeably higher standard deviation values than for any other workload type.

For the Haswell-EP system minimization of the E metric results in over 17.5% of energy consumption reduction with a total execution time increase of less than 37%. The EDP metric minimization finds a system configuration allowing for only 1% reduction of energy-time product which is followed by almost 14% of energy savings and less than 16% of performance drop. The EDS metric minimization reduced its value by almost 1.5% which allowed for saving over 13% of energy with only 10.5% of total execution time increase.

Evaluation of the DEPO tool on the Coffee Lake system for the BT NPB application showed interesting results for all three target metrics (E, EDP, EDS). For minimization of the E metric the energy savings reached 54% with almost 51% of total execution time increase. On the other hand, the EDP metric minimization allowed for almost 24% of energy-time product reduction which indicated 39% decrease of energy consumption and 25.5% of performance drop. The EDS metric minimization reduced its total value by almost 8% with the following 36% energy consumption reduction and 21% of performance drop.

6.5.4 | SP application

Results for the SP NPB application are presented in Figure 26. The tests on Haswell-EP system were performed using the class D problem size which resulted in average total application execution time equal to 2032 s for their default system configuration. The tests on the Coffee Lake system were run for the class C problem size and the average total execution time for the default system setup was 237 s. This implied the test phase to total execution time ratio less than 3.5% and close to 27% for the Haswell-EP and Coffee Lake systems respectively.

For the Haswell-EP minimization of E metric resulted in 25% of energy savings with 18.5% of performance falloff. The EDP metric minimization allowed for 10% reduction of energy-time product which was followed by 17% of energy consumption decrease and 7.5% of total execution time increase. The EDS minimization reduced its overall value by 4% with corresponding 15% energy reduction and 7% of performance degradation.

For the Coffee Lake system the results are much better in terms of minimization of all the three target metrics (E, EDP, EDS). Even in spite of short total execution time of the SP application with the class C problem size on the Coffee Lake system which results in relatively long test phase duration the results of using the DEPO tool for the target metrics optimization are noticeably better than the default result. Minimization of the E metric allowed for almost 56% of energy reduction with less than 12% of the total execution time increased. On the other hand, the EDP metric minimization reduced the energy-time product by over 51% with 55% less energy consumption and 8.5% of performance falloff. The EDS metric minimization reduced its value by 23% so that energy consumption decreased by 53% and the execution time increased by 5.5%.

6.5.5 | LU application

Figure 27 visualizes the results obtained for the tuned LU NPB application. For both Haswell-EP and Coffee Lake systems the problem sizes were set to class D which resulted in average total execution time for the default system configuration equal to 1524 and 4085 s for the Haswell-EP and Coffee Lake systems respectively. The test phase duration to total execution time ratio was less than 4.5% and less than 2% respectively.

Minimizing the E metric for the NPB LU application executed on the Haswell-EP system resulted in 20% of energy consumption reduction with total execution time increased by 18.5%. The EDP target metric minimization reduced the energy-time product by 9% with corresponding 16% energy savings and 9% of performance drop. The EDS minimization reduced its value by 4% which was followed by almost 15% of energy consumption reduction and 7% of performance drop.

Minimization of all the three target metrics (E, EDP, EDS) on the Coffee Lake testbed system resulted in similarly small outcomes. For that system the target power cap was set relatively close to the default power consumption level. This caused that the values of total execution time, total energy consumption and evaluated EDP and EDS metrics were similar to values obtained for the default system configurations. The maximum energy savings were close to 1.5% and were obtained for minimization of the E metric.

The extended validation of the DEPO tool with GSS algorithm only showed that for a typical HPC workload simulated by five representative applications from the well known NAS parallel benchmarks, the tool is able to automatically find appropriate system configurations allowing for total energy consumption reduction or optimization of any multi-objective target metric which combines energy consumption with total execution time. The additional tests proved that the proposed dynamic power capping method implemented by the proposed DEPO tool is able to automatically explore and select the optimal target power cap according to the given target metric for an unknown workload type.

The tests showed that for the Haswell-EP testbed system we are able to find more interesting configurations with the compute intensive workload types, which require more synchronization between threads (such as the BT application), which results in variable power characteristics. When the workload type is compute intensive (CG or MG kernels) with no bottleneck on synchronization or memory the 2-CPU (24-core) Haswell-EP system shows that the power cap close to the default power demand is optimal for EDP and EDS metrics.

On the other hand, the tests performed on the 1-CPU (8-core) Coffee Lake desktop system showed a lot of potential in optimization of any target metric that we used in the DEPO tool. The best results for selected NPB kernels allowed to find such configurations that the energy consumption savings reach 50% with corresponding performance falloff less than 20%.

We emphasize that the results for every test presented in this article were obtained for exactly the same DEPO tool configuration. We are sure that adjusting some of the parameters like test phase time window T might result in better optimization outcome for some of the tested workloads. Specifically, the results for the LU NPB application on the Coffee Lake system might be much more similar to other applications' results if we modified T or changed the exploration method to LS. Such experiments with a different test phase time window T and preferably automatic adjustment of this time window will remain within our future work in the dynamic power capping area.

7 | CONCLUSION AND FUTURE WORKS

The article presented a new approach for automatic management of performance-energy trade-offs in parallel HPC applications. The objective was defined as minimization of one of several independent, dynamic metrics, which were analyzed and experimentally evaluated. The proposed tool implementing the optimization uses various exploration methods along

with the power capping mechanism. The experimental results, provided for a selection of modern CPUs, showed a wide range of possibilities to save energy while preserving high performance of the evaluated applications.

The solution proved to be effective for finding configurations significantly optimizing selected metrics versus default settings for the assumed model of an application. The considered model consists of an initialization phase followed by the computational phase in which compute and memory usage intensity is uniform across the execution time. It should be noted, that this uniformity shall be considered in the context of the measurement window and can potentially be extended using for example, averaging using more samples acting as a low pass filter resulting in deriving power caps for averaged phases.

In terms of algorithmic comparison of the approach adopted by us to the other solutions described in Section 2 we shall mention that some of the latter differ in adopting an off-line approach,[31] using off-line data,[30] adopt additional code[32] and the majority employ various performance and/or energy models[30,37,38,42-44] for optimization. We propose an on-line approach using auto measurements for runtime tuning without a need for code instrumentation at the cost of a specific application model described above.

Specifically, we can formulate conclusions on how effective the solution proved to be in acquiring substantial improvements over energy and performance results of the default power cap settings. Generally, we can conclude consistent behavior of 5%–20% gains in energy with 0%–10% for EDP and smaller 0%–5% for EDS for Haswell and much larger gains in energy up to 60% for Coffee Lake with 5%–50% for EDP and 0%–25% for EDS.

In terms of the GSS to LS comparison, we can generally summarize that GSS was able to achieve slightly better energy savings and did it faster than LS. Gains for EDP and EDS were typically either similar or around 5%–10% better for GSS, with over 20% for EDP only for the heat application.

We expect the proposed solution to be widely used for many HPC applications, especially in cases where specific green computing requirements need to be applied. Thus, in the future we plan to continue the development of the proposed approach by considering the following topics:

1. Introduction of an automatic time window adjustment during the sampling period to optimize the initial setup time and precision of the proper power-cap level.
2. Consideration of power caps for the DRAM domain.
3. Analysis and adaptation of more complex application models, where the compute load changes significantly at runtime, for example, with iterative computations/communication phases. Such an approach has been proposed within the already mentioned MERIC tool that allows to instrument an application with specific API[29] for example, for distinguishing phases such as propagate and collide for Lattice Boltzmann simulation code[36] and subsequent energy-aware optimization using DVFS. However, this requires additional effort from the programmer.
4. Energy versus performance trade-off for GPUs, can also be used for the automatic energy consumption optimization.[58] However, similarly to CPU, an additional metric reflecting the performance needs to be considered.
5. Modeling[59,60] and simulation[61] of using power-cap functionality for energy consumption and performance trade-off in multi-node, parallel HPC systems.
6. Development of more sophisticated, automatic mechanisms of power consumption control for hybrid and heterogeneous systems, where each node or even subsystem has different trade-off characteristics, which can grant a potential chance for higher savings in energy and execution times.

AUTHOR CONTRIBUTIONS

Adam Krzywaniak: Conceptualisation, Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data Curation, Writing - Original Draft, Writing - Review and Editing, Visualisation. **Paweł Czarnul:** Conceptualisation, Methodology, Investigation, Resources, Writing - Original Draft, Writing - Review and Editing, Supervision. **Jerzy Proficz:** Conceptualisation, Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data Curation, Writing - Original Draft, Writing - Review and Editing, Visualisation, Supervision.

ACKNOWLEDGMENT

Computations were carried out partially at the Centre of Informatics Tricity Academic Supercomputer & networK (CI TASK)[†] as well as partially using facilities of the Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology.

[†]https://www.task.gda.pl/

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Adam Krzywaniak https://orcid.org/0000-0003-1904-2510

REFERENCES

1. Zhang H, Hoffmann H. Maximizing performance under a power cap: a comparison of hardware, software, and hybrid techniques. *SIGPLAN Not*. 2016;51(4):545-559. doi:10.1145/2954679.2872375
2. Bailey DH, Barszcz E, Barton JT, et al. The NAS parallel benchmarks. *Int J Supercomput Appl*. 1991;5(3):63-73. doi:10.1177/109434209100500306
3. Liu Y, Zhu H. A survey of the research on power management techniques for high-performance systems. *Softw Pract Exp*. 2010;40(11):943-964.
4. Czarnul P, Proficz J, Krzywaniak A. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. *Sci Program*. vol. 2019;2019:1-19. doi:10.1155/2019/8348791
5. David H, Gorbatov E, Hanebutte UR, Khanna R, Le C. RAPL: memory power estimation and capping. Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design - ISLPED '10; 2010:189; ACM Press, New York.
6. AMD. εBios and kernel developer's guide (BKDG) bios and kernel developer's guide for AMD family 15h models 00h-0fh processorsε; 2015.
7. Ware M, Rajamani K, Floyd M, et al. Architecting for power management: The IBM® POWER7™ approach. Proceedings of the HPCA - 16 16th International Symposium on High-Performance Computer Architecture; 2010:1-11.
8. Ge R, Vogt R, Majumder J, Alam A, Burtscher M, Zong Z. Effects of dynamic voltage and frequency scaling on a K20 GPU. Proceedings of the 2013 42nd International Conference on Parallel Processing; 2013:826-833.
9. Czarnul P, Rościszewski P. Optimization of execution time under power consumption constraints in a heterogeneous parallel system with GPUs and CPUs. In: Chatterjee M, Cao J-n, Kothapalli K, Rajsbaum S, eds. *Distributed Computing and Networking*. Springer; 2014:66-80.
10. Bodas D, Song J, Rajappa M, Hoffman A. Simple power-aware scheduler to limit power consumption by HPC system within a budget. Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing; 2014:21-30; IEEE Press.
11. Rajagopal D, Tafani D, Georgiou Y, Glesser D, Ott M. A Novel approach for job scheduling optimizations under power cap for ARM and intel HPC systems. Proceedings of the 2017 IEEE 24th International Conference on High Performance Computing (HiPC); 2017:142-151.
12. Wang Z, Ranka S, Mishra P. Efficient task partitioning and scheduling for thermal management in multicore processors. Proceedings of the International Symposium on Quality Electronic Design; 2015.
13. Fisher N, Chen JJ, Wang S, Thiele L. Thermal-aware global real-time scheduling on multicore systems. Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium; 2009:131-140.
14. Zhou J, Wei T, Chen M, Yan J, Hu XS, Ma Y. Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems. *IEEE Trans Comput Aided Des Integr Circuits Syst*. 2016;35(8):1269-1282. doi:10.1109/TCAD.2015.2501286
15. Moore J, Chase J, Ranganathan P, Sharma R. Making scheduling εcoole: temperature-aware workload placement in data centers. Proceedings of the Annual Conference on USENIX Annual Technical Conference; 2005:5; USENIX Association, Berkeley.
16. Li T, Narayana VK, El-Ghazawi T. Symbiotic scheduling of concurrent GPU kernels for performance and energy optimizations. Proceedings of the 11th ACM Conference on Computing Frontiers; 2014:36:1-36:10; ACM, New York, NY.
17. Chetsa GLT, Lefevre L, Pierson JM, Stolf P, Costa GD. Application-agnostic framework for improving the energy efficiency of multiple HPC subsystems. Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing; 2015:62-69.
18. Sourouri M, Raknes EB, Reissmann N, et al. Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2017:41:1-41:12; ACM, New York, NY.
19. Kołodziej J, Khan SU, Wang L, Byrski A, Min-Allah N, Madani SA. Hierarchical genetic-based grid scheduling with energy optimization. *Clust Comput*. 2013;16(3):591-609. doi:10.1007/s10586-012-0226-7
20. Unni B, Parveen N, Kumar A, Bindhumadhava BS. An intelligent energy optimization approach for MPI based applications in HPC systems. *CSI Trans ICT*. 2013;1(2):175-181. doi:10.1007/s40012-013-0012-6
21. Langer A, Totoni E, Palekar US, Kalé LV. Energy-efficient computing for HPC workloads on heterogeneous manycore chips. Proceedings of the 6th International Workshop on Programming Models and Applications for Multicores and Manycores; 2015:11-19; ACM, New York, NY.
22. Tiwari A, Laurenzano M, Peraza J, Carrington L, Snavely A. Green queue: customized large-scale clock frequency scaling. Proceedings of the 2012 2nd International Conference on Cloud and Green Computing; 2012:260-267.
23. Peraza J, Tiwari A, Laurenzano M, Carrington L, Snavely A. PMaC's green queue: a framework for selecting energy optimal DVFS configurations in large scale MPI applications. *Concurr Comput Pract Exp*. 2016;28(2):211-231. doi:10.1002/cpe.3184
24. Wu X, Taylor V, Cook J, Mucci PJ. Using performance-power modeling to improve energy efficiency of HPC applications. *Computer*. 2016;49(10):20-29. doi:10.1109/MC.2016.311

25. Huang S, Xiao S, Feng WC. On the energy efficiency of graphics processing units for scientific computing. Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing; 2009:1-8; IEEE.
26. Tiwari N, Bellur U, Sarkar S, Indrawan M. Optimizing MapReduce for energy efficiency. *Softw Pract Exp*. 2018;48(9):1660-1687. 10.1002/spe.2599
27. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exp*. 2011;41(1):23-50. doi:10.1002/spe.995
28. D'Amico M, Gonzalez JC. Energy hardware and workload aware job scheduling towards interconnected HPC environments. *IEEE Trans Parallel Distrib Syst*. 2021;9219(c):1. doi:10.1109/TPDS.2021.3090334
29. Vysocky O, Beseda M, Říha L, Zapletal J, Lysaght M, Kannan V. MERIC and RADAR generator: tools for energy evaluation and runtime tuning of HPC applications. In: Kozubek T, Čermák M, Tichý P, et al., eds. *High Performance Computing in Science and Engineering*. Springer International Publishing; 2018:144-159.
30. Mishra N, Zhang H, Lafferty JD, Hoffmann H. A probabilistic graphical model-based approach for minimizing energy under performance constraints. *ACM SIGARCH Comput Arch News*. 2015;43(1):267-281. doi:10.1145/2786763.2694373
31. Berned GP, Rossi FD, Luizelli MC, Souza SX, Beck Antonio Carlos S, Lorenzon AF. Low learning-cost offline strategies for EDP optimization of parallel applications. *J Syst Archit*. 2021;114(December 2020):101959. doi:10.1016/j.sysarc.2020.101959
32. Imes C, Zhang H, Zhao K, Hoffmann H. CoPPer: soft real-time application performance using hardware power capping. Proceedings of the 2019 IEEE International Conference on Autonomic Computing (ICAC); 2019:31-41; IEEE.
33. Gholkar N, Mueller F, Rountree B, Marathe A. PShifter: feedback-based dynamic power shifting within HPC jobs for performance. Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing; 2018:106-117; ACM, New York, NY.
34. Tsafack CGL, Lefèvre L, Pierson J-M, Stolf P, Da Costa G. Exploiting performance counters to predict and improve energy performance of HPC systems. *Futur Gener Comput Syst*. 2014;36:287-298. doi:10.1016/j.future.2013.07.010
35. Wang Y, Zhang W, Hao M, Wang Z. Online power management for multi-cores: a reinforcement learning based approach. *IEEE Trans Parallel Distrib Syst*. 2022;33(4):751-764. doi:10.1109/TPDS.2021.3092270
36. Calore E, Gabbana A, Schifano SF, Tripiccione R. Energy-efficiency tuning of a lattice boltzmann simulation using MERIC. In: Wyrzykowski R, Deelman E, Dongarra J, Karczewski K, eds. *Parallel Processing and Applied Mathematics*. Springer International Publishing; 2020:169-180.
37. Komoda T, Hayashi S, Nakada T, Miwa S, Nakamura H. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD); 2013:349-356.
38. Zhu Q, Wu B, Shen X, Shen L, Wang Z. Co-run scheduling with power cap on integrated CPU-GPU systems. Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2017:967-977.
39. Rountree B, Ahn DH, Supinski BR, Lowenthal DK, Schulz M. Beyond DVFS: a first look at performance under a hardware-enforced power bound. Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum; 2012:947-953; IEEE.
40. Haidar A, Jagode H, Vaccaro P, YarKhan A, Tomov S, Dongarra J. Investigating power capping toward energy-efficient scientific applications. *Concurr Comput Pract Exp*. 2019;31(6):e4485. doi:10.1002/cpe.4485
41. Fukazawa K, Ueda M, Aoyagi M, et al. Power consumption evaluation of an MHD simulation with CPU power capping. Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2014:612-617.
42. Tiwari A, Schulz M, Carrington L. Predicting optimal power allocation for cpu and dram domains. Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop; 2015:951-959.
43. De Sensi D, Torquati M, Danelutto M. A reconfiguration algorithm for power-aware parallel applications. *ACM Trans Arch Code Optim*. 2016;13(4):1-25. doi:10.1145/3004054
44. Conoci S, Di Sanzo P, Pellegrini A, Ciciani B, Quaglia F. On power capping and performance optimization of multithreaded applications. *Concurr Comput Pract Exp*. 2021;33(13):e6205. doi:10.1002/cpe.6205
45. Krzywaniak A, Proficz J, Czarnul P. Analyzing energy/performance trade-offs with power capping for parallel applications on modern multi and many core processors. Proceedings of the 2018 Federated Conference on Computer Science and Information Systems (FedCSIS); 2018:339-346.
46. Gonzalez R, Horowitz M. Energy dissipation in general purpose microprocessors. *IEEE J Solid State Circuits*. 1996;31(9):1277-1284. doi:10.1109/4.535411
47. Martin AJ, Nyström M, Pénzes PI. Et2: a metric for time and energy efficiency of computation; 2002:293-315; Springer.
48. Laros III, James H, Pedretti K, et al. *Energy Delay Product*. Springer; 2013:51-55.
49. Roberts SI, Wright SA, Fahmy SA, Jarvis SA. Metrics for energy-aware software optimisation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10266 LNCS; 2017:413-430; Springer Verlag.
50. Krzywaniak A, Czarnul P, Proficz J. Extended investigation of performance-energy trade-offs under power capping in HPC environments. Proceedings of the 2019 International Conference on High Performance Computing and Simulation (HPCS); 2019:440-447.
51. Krzywaniak A, Czarnul P, Proficz J. GPU power capping for energy-performance trade-offs in training of deep convolutional neural networks for image recognition. Lecture Notes in Computer Scienceaccepted for Publication in Proceedings of ICCS 2022; 2022.
52. Villa O, Stephenson M, Nellans D, Keckler SW. NVBit: a dynamic binary instrumentation framework for NVIDIA GPUs. *MICRO '52*. Association for Computing Machinery; 2019:372-383.

53. Intel Corporation. Processor counter monitor. Accessed May 11, 2020. https://github.com/opcm/pcm
54. Stahlke D, Mueller J, Morrison R, Marco DD, Arabas S. Gnuplot iostream. Accessed May 11, 2020. https://github.com/dstahlke/gnuplot-iostream
55. Kiefer J. Sequential minimax search for a maximum. *Proc Am Math Soc*. 1953;4:502-506. doi:10.2307/2032161
56. Sanders J, Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. 1st ed. Addison-Wesley Professional; 2010.
57. Balducci M, Choudary A, Hamaker J. Comparative analysis of FFT algorithms in sequential and parallel form; 1996.
58. Krzywaniak A, Czarnul P. Performance/energy aware optimization of parallel applications on GPUs under power capping. In: Wyrzykowski R, Deelman E, Dongarra J, Karczewski K, eds. *Parallel Processing and Applied Mathematics*. Springer International Publishing; 2020:123-133.
59. Proficz J, Czarnul P. Performance and power-aware modeling of MPI applications for cluster computing. Parallel Processing and Applied Mathematics - 11th International Conference, PPAM; 2015; September 6-9, 2015. Revised Selected Papers, Part II:199-209; Krakow, Poland.
60. Czarnul P, Kuchta J, Rościszewski P, Proficz J. Modeling energy consumption of parallel applications. Proceedings of the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS); 2016:855-864.
61. Czarnul P, Kuchta J, Matuszek M, et al. MERPSYS: An environment for simulation of parallel application execution on large scale HPC systems. *Simul Model Pract Theory*. 2017;77:124-140. doi:10.1016/j.simpat.2017.05.009
62. Khan KN, Hirki M, Niemi T, Nurminen JK, Ou Z. RAPL in action: experiences in using RAPL for power measurements. *ACM Trans Model Perform Eval Comput Syst*. 2018;3(2):1-26. doi:10.1145/3177754
63. Hähnel M, Völp M, Döbel B, Härtig H. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Perform Eval Rev*. 2012;40:13-17. doi:10.1145/2425248.2425252
64. Hackenberg D, Schöne R, Ilsche T, Molka D, Schuchart J, Geyer R. An energy efficiency feature survey of the intel Haswell processor. Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop; 2015:896-904.
65. Szustak L, Wyrzykowski R, Olas T, Mele V. Correlation of performance optimizations and energy consumption for stencil-based application on Intel Xeon scalable processors. *IEEE Trans Parallel Distrib Syst*. 2020;31(11):2582-2593. doi:10.1109/TPDS.2020.2996314
66. Desrochers S, Paradis C, Weaver VM. A validation of DRAM RAPL power measurements. Proceedings of the Second International Symposium on Memory Systems, vol. 03, 2016:455-470; ACM, New York, NY.
67. Fahad M, Shahid A, Manumachu RR. Lastovetsky alexey. a comparative study of methods for measurement of energy of computing. *Energies*. 2019;12(11):2204. doi:10.3390/en12112204
68. HPE metered power distribution units (PDU), QuickSpecs. Accessed January 12, 2021. https://h20195.www2.hpe.com/v2/GetDocument.aspx?docname=c04229509
69. HPE flexible slot power supplies, quickspecs. Accessed January 12, 2021. https://h20195.www2.hpe.com/v2/GetDocument.aspx?docname= c04346217
70. Krawczyk H, Nykiel M, Proficz J. Tryton supercomputer capabilities for analysis of massive data streams. *Polish Maritime Res*. 2015;22(3):99-104. doi:10.1515/pomr‐2015‐0062

How to cite this article: Krzywaniak A, Czarnul P, Proficz J. DEPO: A dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing. *Softw Pract Exper*. 2022;52(12):2598-2634. doi: 10.1002/spe.3139

APPENDIX A. ACCURACY OF THE POWER MEASUREMENTS

In this article, we use Intel CPUs for the experiments; thus, we decided to employ the RAPL (Running Average Power Limit) library. This is a mature tool, with over 10 years of development and improvement history, since its introduction.[5] Many related works show that despite these disadvantages RAPL is a robust approach for both CPU[62-65] and DRAM[66] hardware components. However, there are also some researches putting in doubt its general reliability for power measurements.[67]

Thus, in our specific case, we performed a separate validation of the accuracy of the power/energy RAPL measurements provided to the proposed tool. The validation was realized using two gauges installed in two separated physical locations. Figure A1 presents a configuration deployed along with the server containing Haswell CPUs, used for the earlier described and analyzed tool validation experiments. The gauges are placed in the power supply units (PSU), being a part of the server, and in an power distribution unit (PDU), completely external to the evaluated computer. Results are gathered by an external server using SNMP messaging over Ethernet network, and the measurements are performed every 100 ms.

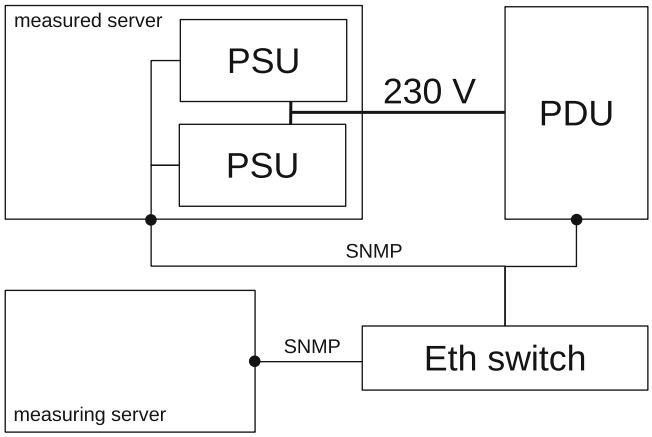

FIGURE A1 Configuration of the Haswell server used for power measurement validation

The used PDU is HPE Metered 3Ph 22 kVA/60309 5-wire 32 A/230 V Outlets (30) C13 (3) C19/Vertical INTL PDU (D9N56A),[68] with inlet and load segment monitoring, Ethernet interface and security features: local/LDAP authentication. The PDU meets IEC 62053-21, Class 1 standards for ±1% or better accuracy in power monitoring, including amperage, voltage, wattage and kilowatts per hour.

The used power supply units' type is HPE 1400 W Flex Slot Platinum Plus Hot Plug Power Supply Kit (720620-B21), supporting power efficiency of 89.9% for 230 V (Titanium power efficiency certification from 80Plus program) and providing multiple operating efficiency modes for redundant power supplies.[69] Each PSU contains a power meter (wattage), unfortunately there are no details about its accuracy; thus, we used it as a double check of the measurements.

The above infrastructure is a part of a GreenLab facility, being a part of the Tryton supercomputer[70] (a cluster, consisting of 1600 nodes, with the total compute power 1.5 PFLOPS) located in Centre of Informatics—Tricity Academic Supercomputer & networK (CI TASK) at Gdansk University of Technology.

Figures A2 and A3 present the measurements of the FFT and HEAT applications, respectively, where they were executed multiple times with decreasing power-cap values. We can notice that the observed RAPL values are lower than PDU ones and visually are at a constant distance, confirmed by the difference chart. The closer examination confirms the high linear correlation between values provided by these meters, with Pearson correlation coefficient equals 0.9750 and

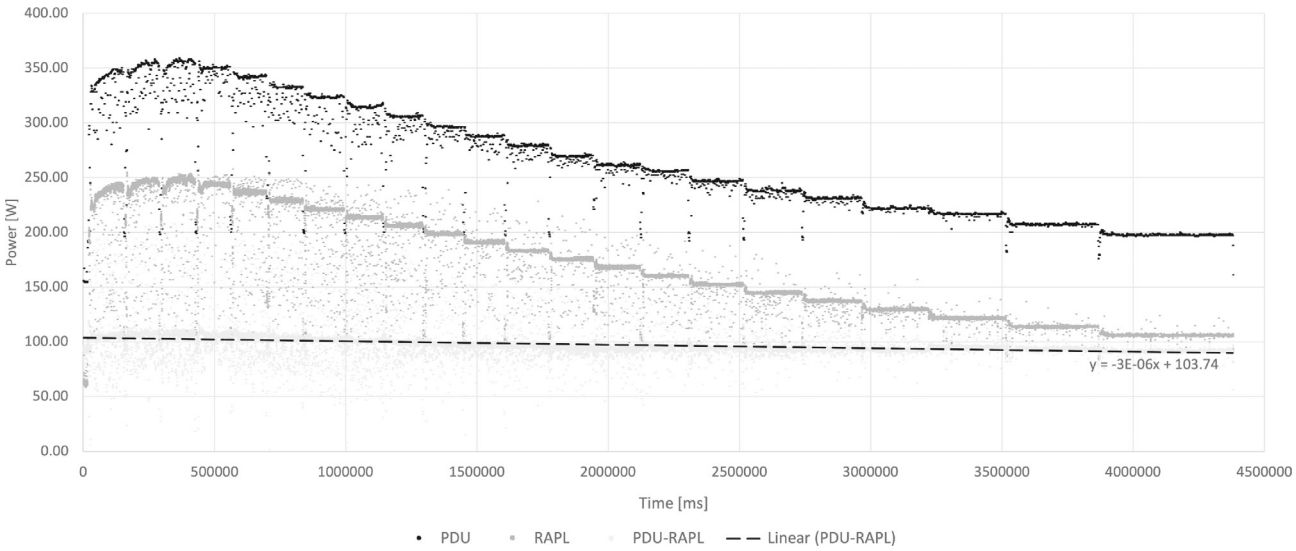

FIGURE A2 Power measurements over time performed on PDU and RAPL meters, and their difference for multiple FFT application executions with the decreasing power-cap values, on the Haswell server.

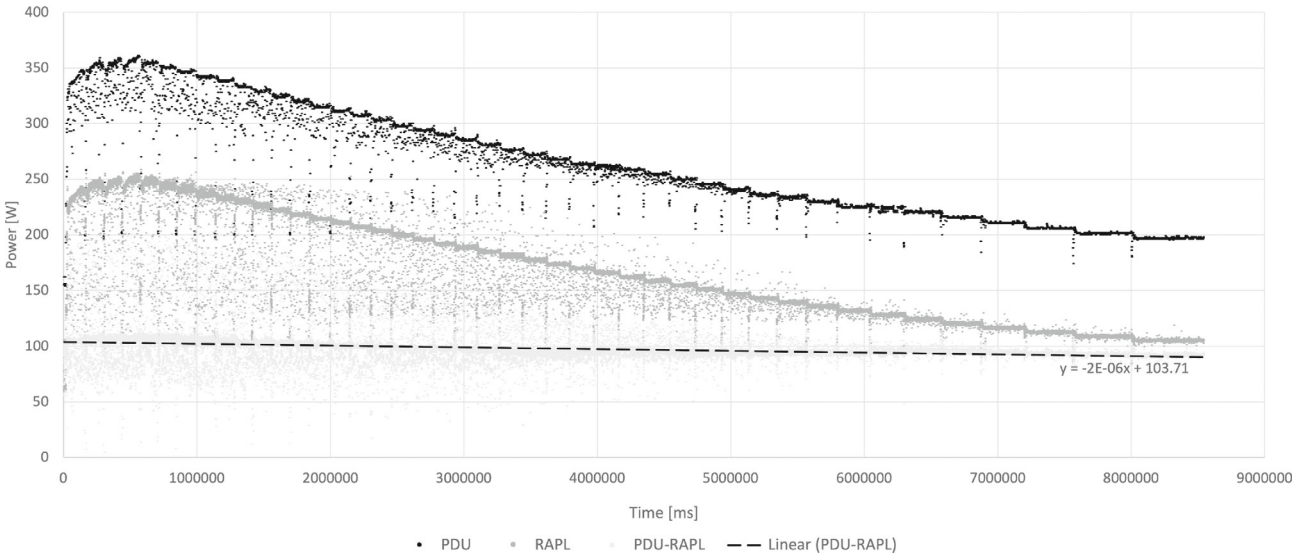

FIGURE A3 Power measurements over time performed on PDU and RAPL meters, and their difference for multiple HEAT application executions with the decreasing power-cap values, on the Haswell server.

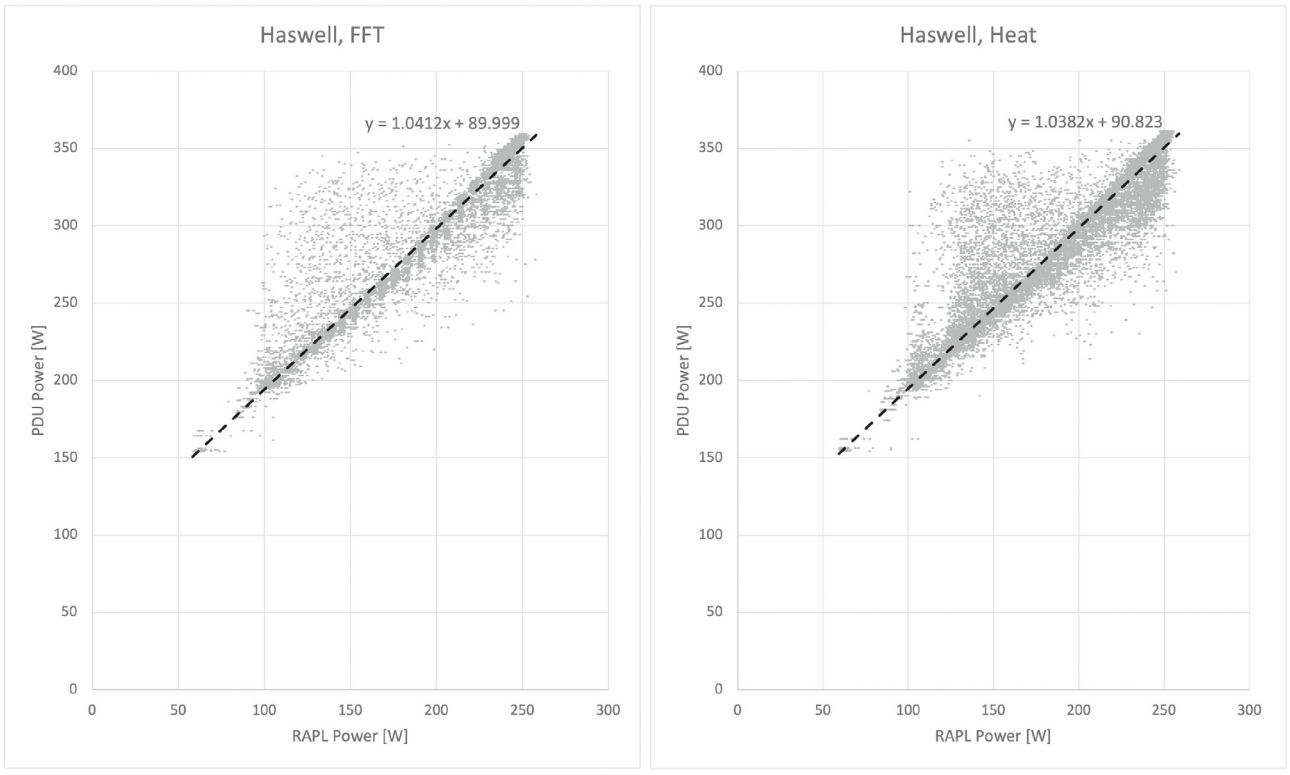

FIGURE A4 Power measurements on PDU versus RAPL meters for FFT and HEAT applications on the Haswell server

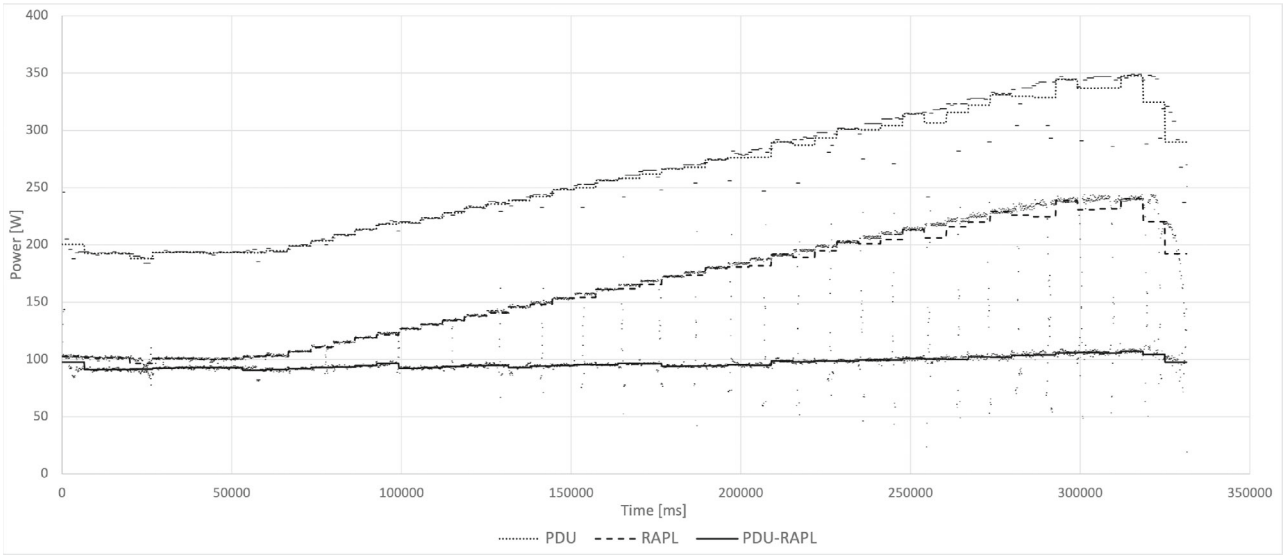

FIGURE A5 Power measurements during the tuning phase performed on PDU and RAPL meters, and their difference, with the averaged values for each tuning window, for the FFT application on the Haswell server.

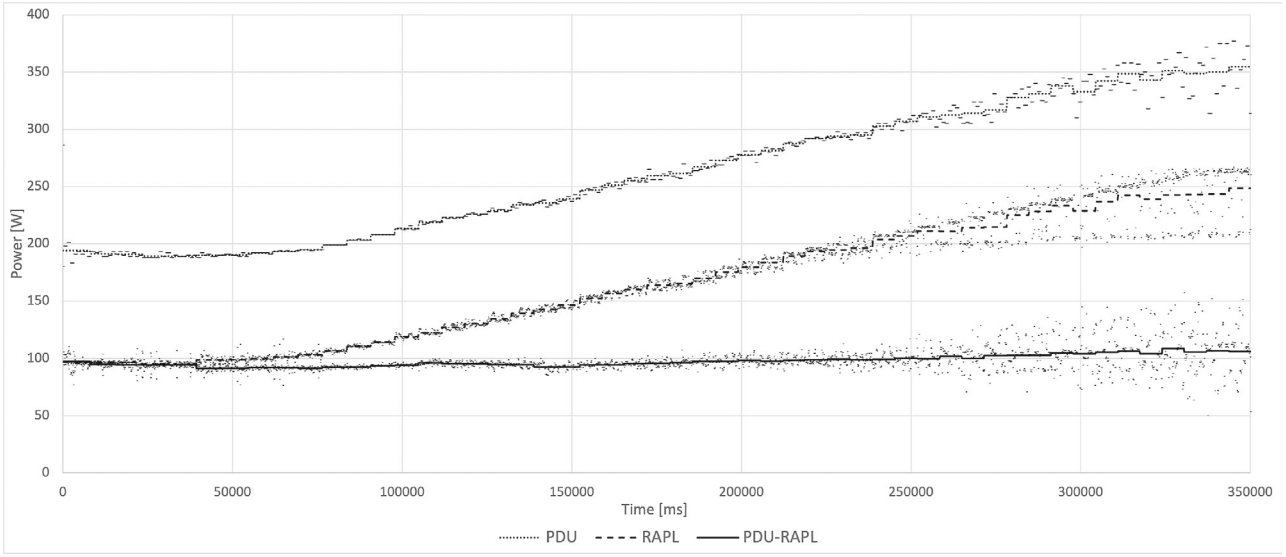

FIGURE A6 Power measurements during the tuning phase performed on PDU and RAPL meters, and their difference, with the averaged values for each tuning window, for the HEAT application on the Haswell server.

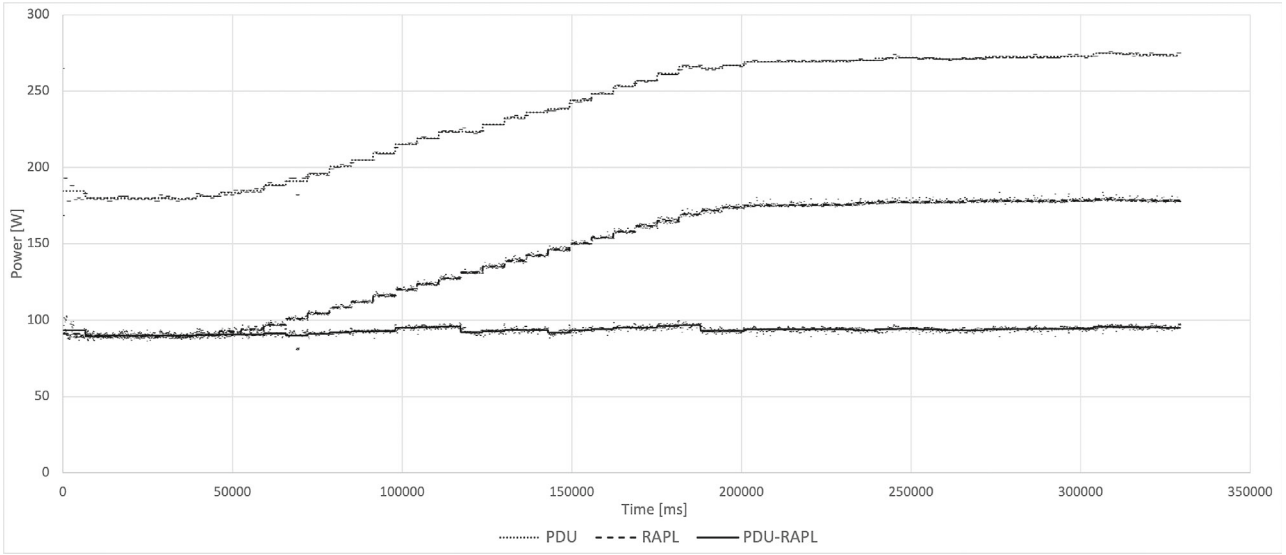

FIGURE A7 Power measurements during the tuning phase performed on PDU and RAPL meters, and their difference, with the averaged values for each tuning window, for the INT application on the Haswell server.

0.9742 respectively. The linear regressions performed over time shows a flat slope being close to 0: -3.2131×10^{-6} and -1.5568×10^{-6}, with the low standard errors: 4.7449×10^{-8} and 1.7252×10^{-8}, implying the low uncertainty: 1.48% and 1.11%.

A similar examination was performed for these results without the time component, see Figure A4. The linear regressions show the slope being close to 1.0 or 45°: 1.0412 and 1.0382, with the low standard errors: 1.3360×10^{-3} and 9.7849×10^{-4}, implying the low uncertainty: 0.13% and 0.09%, and in general the mean offset between RAPL and PDU measurements equals 96.73 and 97.07 W for FFT and HEAT applications respectively.

Let us consider a specific case which occurred in our tool. During the tuning phase the decision about the final power-cap value is taken; thus, the power measurements performed in this period are especially important. Despite the high correlation and almost fixed bias of the RAPL measurements, we can still notice some noise, with values scattered over and under the leading line, see Figures A2 and A3. However, the tool averages the measurements taken during each tuning window: a 6400 ms time frame. Figures A5–A7 present visualization of the tuning phase for the FFT, HEAT, and INT applications, respectively. We can notice that the noise is completely smoothed, and further statistical analysis shows high correlation of the RAPL and PDU measurements, with values of Pearson coefficient equal 0.9992, 0.9995, and 0.9994 for the FFT, Heat, and INT applications, respectively.

The above analysis confirms that the RAPL power measurements reflect the real power levels with a (nearly) constant bias, especially for the case of our tool. The results measured by the PSU meter, and also with the other applications, seem to be compliant with the presented observations. Thus, we can conclude that the proposed tool provides the proper support of power evaluation in the scope of our interests.