Leveraging Inter-Phase Application Dynamism for Energy-Efficiency Auto-Tuning

Madhura Kumaraswamy[1] **and Michael Gerndt**[1]
[1]Chair for Computer Architectures and Parallel Systems, Department of Informatics,
Technical University of Munich, Garching, Bavaria, Germany

Abstract— *Energy efficiency and consumption are currently the challenging issues in current Petascale and in designing future Exascale systems. The European Union Horizon 2020 project READEX (Runtime Exploitation of Application Dynamism for Energy-efficient Exascale computing) develops a tools-aided online approach to analyze and auto-tune HPC applications for energy efficiency on Exascale systems. It exploits dynamism that occurs due to the variation in the application behavior between iterations of the time loop as well as changing control flow within the time loop. This paper describes the readex_interphase tuning plugin, which analyzes the inter-loop dynamism. The plugin performs clustering using DBSCAN for normalized PAPI metrics, and computes the best tuning parameter settings for each cluster. It verifies the cluster analysis results, and finally computes static and dynamic savings. The inter-phase tuning strategy was evaluated for miniMD and INDEED, and the energy savings obtained validate the effectiveness of this methodology.*

Keywords: Automatic tuning, HPC, energy-efficiency, DVFS, DBSCAN, clustering

1. Introduction

As we move towards Exascale computing, designing new energy-efficient systems with an Exaflop capability becomes a challenge, especially when HPC systems have a power demand of several MW [1]. The European Union Horizon 2020 project READEX (Runtime Exploitation of Application Dynamism for Energy-efficient Exascale computing) aims to deliver the first standalone auto-tuning framework to tune large-scale HPC applications for energy-efficiency. While previous works perform static tuning, READEX performs dynamic tuning by switching tuning parameters at runtime. It targets applications that exhibit an iterative behavior in the form of a main progress loop, called a *phase region*. Individual time steps of the phase region are called *phases*.

The READEX methodology consists of *Design Time Analysis* (DTA) and *Runtime Application Tuning*. DTA is performed by the Periscope Tuning Framework (PTF) [2]. First, coarse granular program regions, called *significant regions* having a tuning potential are selected for tuning. PTF then calls a tuning plugin, which performs one or more tuning steps, and uses a search algorithm to explore the multi-dimensional space of *system configurations*, each of which is a *tuning parameter*. READEX currently supports three tuning parameters: CPU frequency to perform DVFS (Dynamic Voltage and Frequency Scaling), uncore frequency to perform UFS (Uncore Frequency Scaling) and the number of OpenMP threads.

READEX exploits the inherent dynamism in the execution characteristics of an application to determine the potential for energy reduction. It measures application dynamism w.r.t. two aspects: intra-phase and inter-phase. The *readex_intraphase* [3] tuning plugin exploits dynamism that arises from variations in the execution of instances of significant regions, called *runtime situations* (rts's). The tuning plugin then selects the best configurations for individual rts's. Since the plugin has no notion of the phase behavior, it is limited to tuning for rts's, and simply selects one best configuration for all the phases even if groups of phases behave differently.

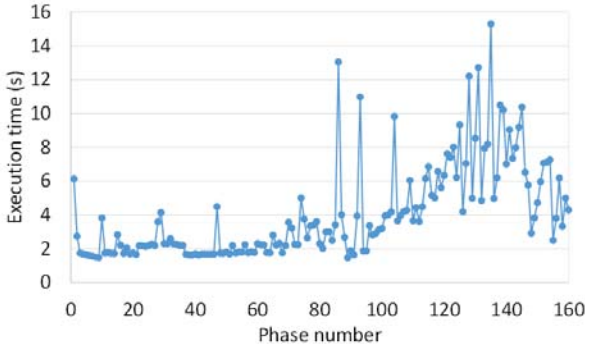


Fig. 1: Variation of the execution time with respect to the phase number of the time loop in INDEED.

A novel tuning plugin, called *readex_interphase* was developed to extend the READEX methodology to exploit inter-phase dynamism arising from variations in the characteristics between phases. Figure 1 illustrates the trend of the execution time across the phases of the INDEED [4] time loop. The execution time is highly dynamic due to the the process of adaptive mesh refinement when the tool comes in contact with the workpiece. The *readex_interphase* tuning plugin leverages this behavior by grouping application phases based on these similarities, thus enabling the selection of different best configurations for groups of phases.

After DTA, rts's with identical/similar best configurations are clustered into a *scenario*, and a *selector* determines the best configuration for a scenario. This information is encapsulated in a tuning model file, which guides the Runtime Application Tuning. A lightweight substrate plugin of Score-P [5], called the *READEX Runtime Library* reads the tuning model and dynamically switches to the specific best configuration upon encountering an rts at runtime. Both DTA and runtime tuning use Score-P as the common measurement and monitoring infrastructure.

The paper outlines the related work in Section 2, and provides an overview of DTA in Section 3. Section 4 presents the tuning steps performed by the *readex_interphase* plugin to analyze application dynamism, select the best configurations and generate the tuning model. Section 5 presents the results of the cluster analysis and the savings obtained for miniMD [6], a mini molecular dynamics application from the Mantevo benchmark suite, and INDEED [4], a production application used for sheet metal forming simulations. Section 6 finally summarizes the important results and provides an insight to the possibilities for future work.

2. Related Work

There are many approaches that employ DVFS tuning in HPC to improve the energy-efficiency. Eastep et al. [7] were able to increase the energy efficiency using offline and online analysis in the GEOPM framework. Laros et al. [8] experimented with both CPU frequency and network bandwidth scaling on the Cray XT architecture and reported improved energy consumption with no performance degradation. The AutoTune project [2] implemented a DVFS tuning plugin that used the enopt library to vary the core frequency for different application regions. It also implemented a model based frequency prediction for minimum energy consumption. Gonzalez et al. [9] implemented an adaptive power-capping technique to optimize the performance of threaded applications. Active threads are dynamically packed onto variable number of cores, and DVFS is used to optimize the performance within the power constraints. While the above methods are static, READEX implements a dynamic tuning approach.

The ANTAREX project [10] specifies adaptivity strategies of the application at runtime by using a Domain Specific Language approach to configure software knobs for the application regions. This is specialized for ARM-based multi-cores and GPGPUs, while READEX targets all HPC systems.

3. Design-Time Analysis

DTA is performed by PTF, which is a distributed framework consisting of the frontend, the tuning plugins, the experiment execution engine, and a hierarchy of analysis agents. First, *scorep-autofilter* filters out fine-granular program regions to avoid measurement overheads due to automatic compiler instrumentation. The *readex-dyn-detect* tool then selects significant regions that constitute most of the application execution time based on a granularity threshold. It then computes the application's tuning potential for: a) intra-phase dynamism, resulting from variations due to the execution of rts's or in the control flow within a single phase, and b) inter-phase dynamism due to changing application characteristics between phases. The dynamism results for each significant region are then exported to a READEX configuration file in the XML format.

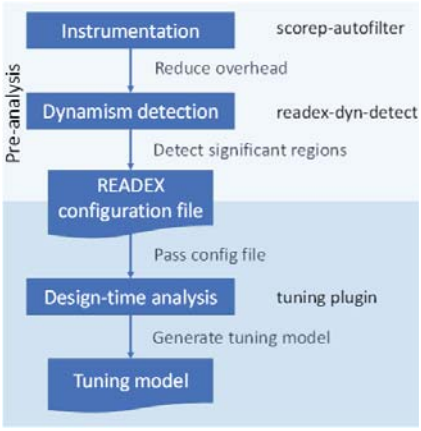


Fig. 2: The workflow of the Design-Time Analysis stage of the READEX framework.

The *readex_interphase* plugin is configured via the configuration file by specifying the objective function to tune for, and the values for the system-level tuning parameters. The plugin then executes experiments, during which the PTF analysis agents send measurement requests and receive results to and from Score-P. The plugin generates a tuning model, which is propagated to the runtime tuning stage.

3.1 Leveraging Application Dynamism

READEX quantifies the tuning potential using two dynamism metrics: execution time and compute intensity. Intra-phase dynamism exists for execution time if there is a variation in the execution time of the rts's, and for compute intensity due to the deviation of the compute intensity across significant regions. Finally, inter-phase dynamism for execution time exists if there is a variation in the minimum and maximum execution time for the phase region.

To exploit inter-phase dynamism, phases with similar characteristics or behavior are grouped together for the selection of different best configurations for each group of phases. This also allows individual rts's to be distinguished in the tuning model based on the behavior of the phase from which it was called. The following sections describe how the *readex_interphase* tuning plugin performs cluster analysis to leverage inter-phase dynamism.

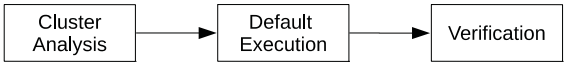


Fig. 3: The *readex_interphase* plugin executes three tuning steps: Cluster analysis, default execution and verification.

4. Inter-phase Tuning

The *readex_interphase* plugin is a modified Dynamic Voltage Frequency Scaling (DVFS) plugin that performs a cluster analysis of the phases based on the similarity in their behavior, and determines the best configuration for each cluster. The following sections describe the sequence of steps performed by the *readex_interphase* tuning plugin, as illustrated in Figure 3.

4.1 Cluster Analysis

The cluster analysis step uses the random search strategy [2] to create a number of experiments to request measurements for randomly selected configurations. It then clusters the phases based on the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [11] algorithm using normalized features (clustering aspects). Then, cluster-best configurations are determined for the phase as well as individual rts's.

4.1.1 Initialization

The *readex_interphase* plugin first reads the tuning parameters and the objective for tuning. If no objective is specified, energy is set as the default objective, and is measured as the energy consumption of the entire node.

The plugin allows the user to specify different objective functions to be measured in addition to energy, such as execution time, CPU energy, Energy Delay Product, Energy Delay Product Squared, and Total Cost of Ownership (TCO). TCO determines the overall costs of a job as the sum of the energy costs plus the execution time dependent fraction of the HPC system costs.

4.1.2 Experiment Execution

The plugin reads the list of significant regions from the configuration file, and uses the random search strategy to create a search space of the tuning parameters. This prevents the search space from exploding. The plugin then performs experiments, i.e., an execution of a single phase by measuring the effect of a random system configuration selected based on a uniform distribution [2]. The number of experiments that are executed is determined by a user specified number of *samples* in the configuration file.

In each experiment, the plugin requests the objective values for the phase and the rts's in the form of a tuning request to Score-P. Each PTF analysis agent stores the measurements returned by Score-P for those MPI processes controlled by it. The plugin computes the consumed energy by aggregating the values returned by the designated processes of all the nodes for an MPI application. In addition to the objective values, the plugin collects PAPI [12] hardware metrics, such as the number of AVX instructions, L3 cache misses, and the conditional branch instructions. These are used to derive the phase features for clustering in later steps.

At the end of every experiment, each PTF analysis agent generates a partial *Calling Context Graph* (CCG) [1] to build the sequence of calls from the phase to different regions. Individual CCGs are propagated to the frontend and then aggregated to produce the complete CCG.

4.1.3 Process Results

The plugin performs clustering using a carefully selected set of features, and determines the best configuration for each cluster, as described in the following sections.

a) Clustering: Phase features, such as arithmetic intensity, capture the characteristics of phases. The mapping of phases into clusters enables PTF to select different best configurations for each cluster. The features for clustering should be chosen carefully since the resulting tuning model heavily influences the runtime tuning stage. Hence, the following considerations were taken into account while selecting the clustering method and the features:

- Since the dynamism in many applications arises from the variation in the compute intensity or memory read/write access patterns, compute intensity and the number of conditional branch instructions were chosen as the features for clustering. Compute intensity is determined by $\frac{\#AVX\ Instructions}{\#L3\ Cache\ Misses}$.
- DBSCAN was chosen for clustering, since it does not make any assumptions about the shape of the clusters. Moreover, it is robust against noise or measurement outliers, and prevents them from being assigned to a cluster.

The plugin normalizes the features using the *min-max* method. Feature normalization ensures that while using features with different data ranges, all the features have a similar weight, and that there is no bias during clustering. The *min-max* method is a range normalization method that scales the numeric range of a feature to a [0,1] range, as shown in the following formula:

$$\forall x_i \in X, \quad x^{'} = \frac{x_i - min(X)}{max(X) - min(X)} \tag{1}$$

The plugin uses the normalized features to perform clustering using DBSCAN, and groups points that are closely packed together, resulting in high density regions. It marks points that lie in low-density regions as noise. The algorithm requires two parameters to cluster the data points:

[1] A context sensitive version of a call graph.

1) minPts: *minPts* determines the minimum number of points that must lie in the neighborhood to define a cluster. The *minPts* parameter was chosen to be 4 [13], which means that a neighborhood should have a minimum of 4 data points.
2) eps: *eps* is the maximum distance between any two points for them to be considered to be in the same neighborhood. This ensures that $\forall p \in C_i$, the Euclidean distance between any pair of points is less than or equal to *eps*. *eps* is automatically determined using the *elbow method* [14]. The plugin computes the average 3-NN (3-Nearest-Neighbor) Euclidean distances for all the data points, and the distances are arranged in the ascending order. The elbow is a sharp change in the average 3-NN distance curve, as shown in Figure 4. It is computed as the average 3-NN distance of the point that has the maximum distance to the line formed by the points with the minimum and the maximum 3-NN distance (the first and the last points on the curve).

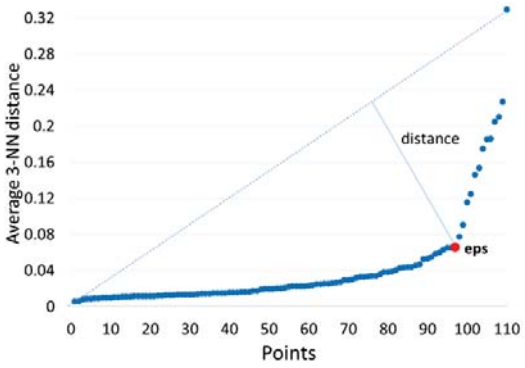


Fig. 4: *eps* for DBSCAN is automatically determined as the point that has the greatest distance from the line formed by the first and last points on the curve.

b) Computation of cluster-best configurations: The plugin uses the objective normalized by the AVX instructions to compute the best configuration for the clusters and the rts's. This allows the plugin to tune phases with different amounts of work but of the same kind, such as more iterations of an iterative solver. For each newly created cluster, the plugin determines the best setting of the tuning parameters for all the phases as well as individual rts's to achieve the lowest normalized objective value. The cluster-best configuration is then applied for all the phases of a particular cluster during the next application run.

4.2 Default Execution

Before starting the default execution step, PTF restarts the application, and the plugin creates the same number of experiments as in the previous step. Each experiment measures the objective value for the default configuration, i.e., the execution with the default tuning parameter settings provided by the batch system. The measurements obtained for each phase as well as the rts's are then used for computing the savings at the end of the plugin.

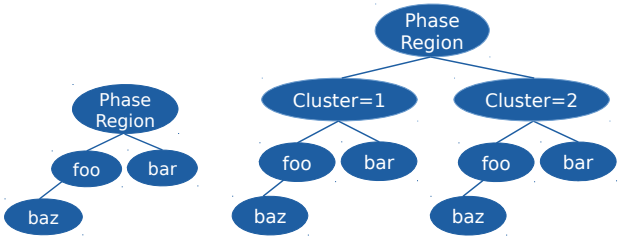
4.3 Verification

In the verification step, PTF restarts the application, and the plugin creates the same number of experiments as in the previous steps. This step determines if the theoretical savings computed in the cluster analysis step match the actual savings incurred after switching the configurations.

PTF first configures the *READEX Runtime Library* at the start of the phase with the corresponding cluster-best configuration. If the phase was determined to be a noise point in the cluster analysis step, it is executed using the default configuration. Similarly, the configurations for the rts's are set to the rts-specific best configuration for the current phase's cluster. The runtime system thus enforces the static phase configuration for the phase, and dynamically switches system configurations for the individual rts's. The measurements recorded for these experiments are used to determine the true static and dynamic energy savings obtained by taking the dynamic switching overhead into account.

4.3.1 Cloning the Calling Context Graph (CCG)

Once all the experiments are completed, the plugin clones the children of the phase region of the Calling Context Graph (CCG) to correctly depict the formed clusters.



(a) Before cloning. (b) After cloning.

Fig. 5: Calling Context Graph (CCG) for a dummy application consisting of three regions foo, bar and baz.

Figure 5a shows the initial CCG of a dummy application containing regions foo, bar and baz. The callpath of an rts starts with the phase region and includes the names of the regions that are called on the way to this region. For the phase, the callpath would be */PhaseRegion*. Figure 5b illustrates the case when the plugin creates new nodes for the clusters, and clones all the nodes under the phase region. After this step, the cluster ID is used to represent all the phases belonging to a cluster. Thus, the callpath becomes */PhaseRegion/Cluster=1*. Doing this has an advantage of reducing the memory overhead because a single cluster node can be used to represent all the phases belonging to it. The

tuning results for each cluster are then inserted into the cluster nodes.

4.3.2 Computing the savings

The plugin determines the following three values to characterize the savings:

1) Static savings for the whole phase: The improvement in the objective value for the best static configuration of the phase over the objective value for the default configuration accumulated over all the clusters.

$$S^{static}_{phase} = \sum_{cluster=1}^{n} (o^{default}_{phase,cluster} - o^{opt}_{phase,cluster}) \Big/ \sum_{cluster=1}^{n} o^{default}_{phase,cluster} * 100 \quad (2)$$

2) Static savings for the rts's: The improvement in the objective value for the rts's for the best setting for the cluster (static best) over the objective value for the rts's for the default setting accumulated over all the clusters.

$$S^{static}_{RTS} = \sum_{cluster=1}^{n} \sum_{rts \in RTS} (o^{default}_{rts,cluster} - o^{static}_{rts,cluster}) \Big/ \sum_{cluster=1}^{n} \sum_{rts \in RTS} o^{default}_{rts,cluster} * 100 \quad (3)$$

3) Dynamic savings for the rts's: The improvement in the objective value for rts-specific best configuration over the objective value for rts's for the static best setting for the cluster accumulated over all the clusters.

$$S^{dyn}_{RTS} = \sum_{cluster=1}^{n} \sum_{rts \in RTS} (o^{static}_{rts,cluster} - o^{opt}_{rts,cluster}) \Big/ \sum_{cluster=1}^{n} \sum_{rts \in RTS} o^{static}_{rts,cluster} * 100 \quad (4)$$

where,

$o^{default}_{phase,cluster}$ = objective value of the phase for the default configuration for a cluster

$o^{opt}_{phase,cluster}$ = objective value of the phase for the static best configuration for a cluster

$o^{default}_{rts,cluster}$ = objective value of the rts for the default configuration for a cluster

$o^{static}_{rts,cluster}$ = objective value of the rts for the static best configuration for a cluster

$o^{opt}_{rts,cluster}$ = objective value of the rts for the rts-specific optimal configuration for a cluster

4.3.3 Tuning Model Generation

As the last step of DTA, the plugin calls the tuning model generation. Rts's that have identical best configurations are grouped into a unique scenario. Rts's with similar system configurations can also be clustered using a similarity score that determines if the objective values are close to each other. This ensures that two rts's with similar best configurations can be merged into one scenario to reduce the runtime switching overhead. A selector then returns the best configuration for that scenario with respect to the chosen objective. The tuning model encapsulates this knowledge as a JSON file. For production runs, the READEX Runtime Library reads the tuning model and dynamically switches to the best configuration for the cluster ID of the currently executing phase.

5. Evaluation

The evaluation of the clustering analysis by the *readex_interphase* plugin was performed on two applications: miniMD and INDEED.

miniMD is a lightweight, parallel molecular dynamics simulation code written in C++, and performs molecular dynamics simulation of a Lennard-Jones Embedded Atom Model (EAM) system. It provides an input file for users to specify the problem size, temperature, the number of timesteps, and the timestep size among others. The evaluation of DTA was done for the hybrid (MPI+OpenMP) AVX vectorized version, with a problem size of 50 for the Lennard-Jones system.

INDEED is a finite element software that performs sheet metal forming simulations, and has an implicit time integration. The simulation involves a stationary workpiece and a number of tools with different geometries that move towards this workpiece. When there is a contact between the tool and the workpiece, an adaptive mesh refinement takes place, and the number of finite element nodes increases with every time step. This results in an increasing computational cost. For every change in the number of contact points between the tools and the workpiece, the time loop computes the solution to a system of equations until equilibrium is reached.

Experiments were conducted on the Taurus HPC system at the ZIH in Dresden. Each node on Taurus contains two 12-core Intel Xeon CPUs E5-2680 v3 (Intel Haswell family), running with a default CPU frequency of 2.5 GHz and an uncore frequency of 3 GHz. Energy measurements are provided on Taurus via the HDEEM [15] measurement hardware, which allows processor as well as blade energy measurements.

Figures 8 and 9 show the trend in the compute intensity for the executions of the time loop of miniMD and INDEED respectively. As can be seen, both applications show dynamic behavior during the course of the execution.

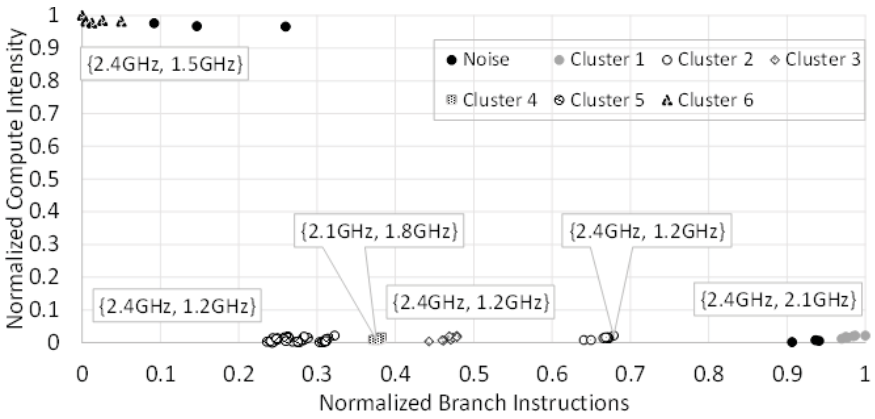


Fig. 6: Results of the cluster analysis performed on the phases of miniMD. Six clusters are produced, and the best configuration in the form {CPU_freq,Uncore_freq} for each cluster is depicted. Noise points are marked in black.

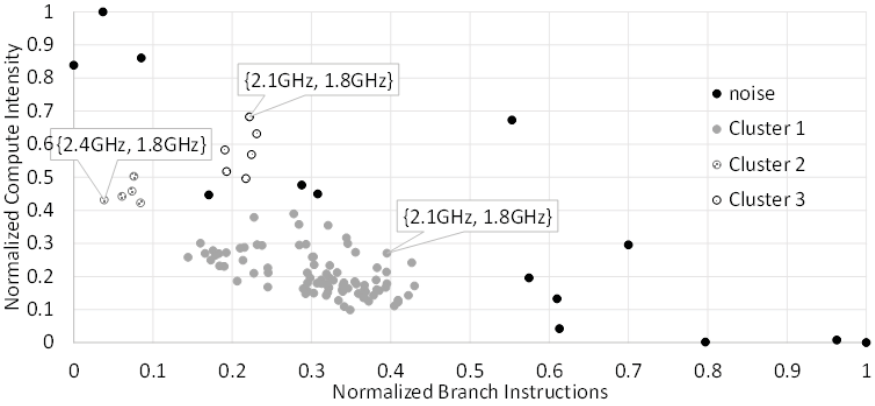


Fig. 7: Results of the cluster analysis performed on INDEED phases. Three clusters are produced, and the best configuration in the form {CPU_freq,Uncore_freq} for each cluster is depicted. Noise points are marked in black.

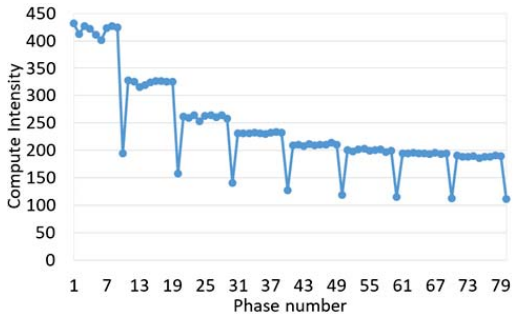


Fig. 8: Variation of the compute intensity with respect to the iteration number of the time loop in miniMD.

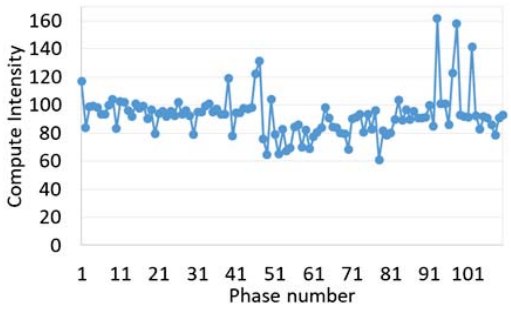


Fig. 9: Variation of the compute intensity with respect to the iteration number of the time loop in INDEED.

For miniMD, a sharp change occurs at every tenth phase, while for INDEED, it is non-deterministic due to the adaptive mesh refinement.

The *readex_interphase* tuning plugin was run on both applications, and the results are illustrated in Figures 6 and 7 for miniMD and INDEED respectively. The plugin used the normalized compute intensity and the normalized conditional branch instructions to perform the clustering. Six clusters were produced for miniMD, while three clusters were produced for INDEED. The points that were not assigned to any cluster were marked as noise, and are colored in black. Both figures illustrate that a low normalized compute intensity coupled with a low value of normalized conditional branch instructions requires a high CPU frequency, while a low

normalized compute intensity value coupled with a high value of conditional branch instructions requires high values of both CPU and uncore frequencies.

Table 1 presents the static and dynamic savings in percentages obtained for miniMD and INDEED as a result of applying DTA using the *readex_interphase* tuning plugin.

Table 1: Static savings for the phase and the rts's, and the dynamic savings for the rts's for miniMD and INDEED obtained after applying the *readex_interphase* plugin.

Application	Static savings for the whole phase (%)	Static savings for the rts's (%)	Dynamic savings for the rts's (%)
miniMD	13.74	14.51	0.03
INDEED	5.75	9.24	10.45

Static savings of 13.74% for miniMD and 5.75% for INDEED were observed for the phase. This value is measured as the overall savings for the phases over all clusters obtained as a result of the inter-phase tuning. While the static and dynamic savings for the rts's of INDEED show a relatively good improvement of 9.24% and 10.45% respectively, miniMD records low dynamic savings. This is because miniMD has only two significant regions, and one of the regions is called only once during the entire application run. On the other hand, INDEED has nine significant regions, providing more potential for dynamism, and hence records better dynamic savings.

6. Summary and Future Work

Energy efficiency optimization is now one of the major challenges on the way to Exascale computing. A novel *readex_interphase* tuning plugin enables READEX to exploit inter-phase dynamism, and select best configurations for groups for phases. The tuning plugin implements a three-step tuning strategy that handles dynamism across phases via experiments for randomly chosen system configurations. It clusters phases using DBSCAN and uses normalized PAPI metrics (compute intensity and conditional branch instructions) as cluster features. A verification step is performed to compute the true savings by taking into account the dynamic switching overhead. Static savings of 13.74% obtained for miniMD, and dynamic savings of 10.45% for INDEED highlight the effectiveness of the tuning plugin.

For future improvements, a planned selective search for specific configurations can be performed in cases when a cluster has too few points, resulting in low confidence on the results. Another step is the development of a cluster prediction mechanism similar to a branch prediction scheme for dynamic cluster prediction at runtime.

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

References

[1] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, "Top 500," Nov. 2017. [Online]. Available: https://www.top500.org/lists/2017/11/

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

[3] M. Kumaraswamy, A. Chowdhury, and M. Gerndt, "Design-time analysis for the READEX tool suite," in *Parallel Computing is Everywhere, Proceedings of the International Conference on Parallel Computing, ParCo 2017, 12-15 September 2017, Bologna, Italy*, 2017, pp. 307–316. [Online]. Available: https://doi.org/10.3233/978-1-61499-843-3-307

[4] "Highly accurate finite element simulation for sheet metal forming," http://gns-mbh.com/en/produkte/indeed/.

[5] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-p: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin: Springer, 2012, pp. 79–91.

[6] P. S. Crozier, H. K. Thornquist, R. W. Numrich, A. B. Williams, H. C. Edwards, E. R. Keiter, M. Rajan, J. M. Willenbring, D. W. Doerfler, and M. A. Heroux, "Improving performance via mini-applications." Sandia National Laboratories, Tech. Rep., 2009.

[7] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, *Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions*. Cham: Springer International Publishing, 2017, DOI: 10.1007/978-3-319-58667-0_21.

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

[9] J. Gonzalez, J. Gimenez, and J. Labarta, "Automatic detection of parallel applications computation phases," in *2009 IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–11.

[10] C. Silvano, G. Agosta, S. Cherubin, D. Gadioli, G. Palermo, A. Bartolini, L. Benini, J. Martinovič, M. Palkovič, K. Slaninová, J. a. Bispo, J. a. M. P. Cardoso, R. Abreu, P. Pinto, C. Cavazzoni, N. Sanna, A. R. Beccari, R. Cmar, and E. Rohou, "The ANTAREX approach to autotuning and adaptivity for energy efficient hpc systems," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '16. New York, NY, USA: ACM, 2016, pp. 288–293. [Online]. Available: http://doi.acm.org/10.1145/2903150.2903470

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[12] "Performance Application Programming Interface," http://icl.cs.utk.edu/papi/.

[13] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 169–194, June 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1009745219419

[14] M. N. Gaonkar and K. Sawant, "Autoepsdbscan: Dbscan with eps automatic for large dataset," *International Journal on Advanced Computer Theory and Engineering*, vol. 2, no. 2, pp. 11–16, 2013.

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