Exploiting Inter-Phase Application Dynamism to Auto-tune HPC Applications for Energy-efficiency

Madhura Kumaraswamy and Michael Gerndt
Technical University of Munich
Garching, Bavaria
{kumarasw|gerndt}@in.tum.de

ABSTRACT
The EU Horizon 2020 project READEX provides a novel readex_interphase tuning plugin to exploit the variation in the application behavior between individual time loop iterations, or interloop dynamism to auto-tune HPC applications for energy consumption. It clusters phases with similar behavior using DBSCAN for normalized PAPI metrics and computes the best tuning parameter settings. The methodology is evaluated for miniMD and INDEED.

CCS CONCEPTS
• Hardware → Power estimation and optimization; • Computer systems organization → Multicore architectures; • Software and its engineering → Application specific development environments;

KEYWORDS
Automatic tuning, HPC, DVFS, energy-efficiency, DBSCAN

ACM Reference Format:

1 INTRODUCTION
Optimizing energy consumption has become a challenging issue for current HPC systems and in designing new Exascale systems. To overcome this challenge, the EU Horizon 2020 project READEX (Runtime Exploitation of Application Dynamism for Energy-efficient Exascale computing) provides an auto-tuning framework to tune HPC applications for energy-efficiency by switching tuning parameters dynamically at runtime.

The READEX methodology consists of Design Time Analysis (DTA) and Runtime Application Tuning (RAT). During DTA, the Periscope Tuning Framework (PTF) [6] calls a tuning plugin, which performs tuning steps to explore the multi-dimensional space of system configurations, each of which is a tuning parameter. Measurements are requested for instances (rts’s) of coarse-granular instrumented program regions, called significant regions. The best system configurations for the rts’s are stored in a tuning model, which is read during the RAT stage to dynamically switch the system configuration on encountering an rts during production runs.

READEX determines the potential for tuning energy consumption by leveraging the variation (dynamism) in the execution characteristics of an application. It computes: a) intra-phase dynamism, which is caused due to the variation in the execution of rts’s within a single instance (phase) of the main progress loop w.r.t. execution time and compute intensity, and b) inter-phase dynamism resulting from the variation in the execution characteristics between phases w.r.t. execution time.

A novel tuning plugin, called readex_interphase exploits the inter-phase dynamism, for example, resulting from variations in the execution time of the time loop of a metal-forming simulation application that performs an adaptive mesh refinement, as illustrated in Figure 1. The readex_interphase plugin can group similar application phases, and select different best system configurations for each phase group. This also allows individual rts’s called from different phase groups to be distinguished in the tuning model.

2 INTER-PHASE TUNING PLUGIN
The readex_interphase plugin performs the following three tuning steps to cluster similarly behaving phases:

1) Cluster Analysis: The plugin reads the tuning parameters (CPU frequency, uncore frequency and OpenMP threads), the objective (energy, execution time, CPU energy, Energy Delay Product, Energy Delay Product Squared, or Total Cost of Ownership) for tuning, and the significant regions from a configuration file. It uses the random search strategy [6] to create a search space of the tuning parameters, and performs experiments to measure the effects of a randomly selected system configuration. In each experiment, the plugin requests the objective values for the phase and the rts’s,
Table 1: Energy savings obtained for the phase and the rts’s for miniMD and INDEED for the readex_interphase plugin.

<table>
<thead>
<tr>
<th>Application</th>
<th>Static savings for the whole phase (%)</th>
<th>Static savings for the rts’s (%)</th>
<th>Dynamic savings for the rts’s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>miniMD</td>
<td>13.74</td>
<td>14.51</td>
<td>0.03</td>
</tr>
<tr>
<td>INDEED</td>
<td>5.75</td>
<td>9.24</td>
<td>10.45</td>
</tr>
</tbody>
</table>

as well as PAPI [2] hardware metrics (AVX instructions, L3 cache misses, and the conditional branch instructions).

It uses DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [4] to cluster phases using compute intensity (AVX Instructions), and conditional branch instructions as the features (clustering aspects). DBSCAN takes minPts, which is the minimum number of points in the neighborhood of a point, and eps, which is the maximum distance between two points in the same neighborhood as the parameters for clustering. eps is automatically determined as the point corresponding to a sharp change in the average 3-NN distance curve using the elbow method [5]. The plugin normalizes the cluster features, and groups closely packed points into clusters, and marks the other points as noise.

For each cluster, the plugin determines the best system configuration that results in the lowest objective value normalized by the AVX instructions. This allows to tune phases with different amounts of work but of the same kind. The cluster-best configuration is then applied for all the phases and rts’s of a particular cluster during the application run.

(2) Default Execution: Here, the plugin creates the same number of experiments as in the previous step. Each experiment measures the objective value for the default system configuration (execution with the default tuning parameter settings provided by the batch system). These measurements are used to compute the savings at the end of the plugin.

(3) Verification: The plugin determines if the computed savings match the true savings by executing each phase with the best configuration of its cluster, and the rts’s with the rts-specific best configuration of the current phase’s cluster. If the phase is a noise point, it is executed using the default configuration.

The plugin then computes: (a) static savings for the whole phase, which is the improvement in the objective value for the best static configuration of the phase over the default configuration accumulated over all the clusters, (b) static savings for the rts’s, which is the improvement in the objective value for the rts’s for the cluster-best setting over the default setting accumulated over all the clusters, and (c) dynamic savings for the rts’s, which is the improvement in the objective value for rts-specific best configuration over the static cluster-best setting accumulated over all the clusters.

4 CONCLUSIONS

To overcome the challenge of optimizing energy efficiency, a novel readex_interphase tuning plugin was developed to exploit inter-phase dynamism, and select best system configurations for groups of phases. A three-step tuning strategy handles dynamism across phases by performing experiments for randomly chosen system configurations. The plugin clusters phases using DBSCAN and uses normalized PAPI metrics (compute intensity and conditional branch instructions) as clustering features. It performs a verification step 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 this plugin.

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

REFERENCES