# Multi-Objective Auto-Tuning with Insieme: Optimization and Trade-Off Analysis for Time, Energy and Resource Usage

Philipp Gschwandtner, Juan J. Durillo, and Thomas Fahringer

University of Innsbruck, Institute of Computer Science, Austria
{philipp,juan,tf}@dps.uibk.ac.at

**Abstract.** The increasing complexity of modern multi- and many-core hardware design makes performance tuning of parallel applications a difficult task. In the past, auto-tuners have been successfully applied to minimize execution time. However, besides execution time, additional optimization goals have recently arisen, such as energy consumption or computing costs. Therefore, more sophisticated methods capable of exploiting and identifying the trade-offs among these goals are required. In this work we present and discuss results of applying a multi-objective search-based auto-tuner to optimize for three conflicting criteria: execution time, energy consumption, and resource usage. We examine a method, called RS-GDE3, to tune HPC codes using the Insieme parallelizing and optimizing compiler. Our results demonstrate that RS-GDE3 offers solutions of superior quality than those provided by a hierarchical and a random search at a fraction of the required time (5%) or energy (8%). A comparison to a state-of-the-art multi-objective optimizer (NSGA-II) shows that RS-GDE3 computes solutions of higher quality. Finally, based on the trade-off solutions found by RS-GDE3, we provide a detailed analysis and several hints on how to improve the design of multi-objective auto-tuners and code optimization.

## 1 Introduction

The performance of a software application crucially depends on the quality of its source code. The increasing complexity and multi/many-core nature of hardware design have transformed code generation, whether done manually or by a compiler, into a complex, time-consuming, and error-prone task which additionally suffers from a lack of performance portability. To mitigate these issues, a new research field, known as *auto-tuning*, has gained increasing attention. *Auto-tuners* are an effective approach to generate high-quality portable code. They are able to produce highly efficient code versions of libraries or applications by generating many code variants which are evaluated on the target platform, often delivering high performance code configurations which are unusual or not intuitive.

Whilst earlier auto-tuning approaches were mainly targeted at execution time [1], other optimization criteria such as energy consumption or computing costs are gaining interest nowadays. In this new scenario, a code *configuration* that

is found to be optimal for low execution time might not be optimal for another criterion. Therefore, there is no single solution to this problem that can be considered optimal, but a set, namely the Pareto set, of solutions (i.e. code configurations) representing the optimal trade-off among the different optimization criteria. Solutions within this set are said to be non-dominated: any solution within it is not better than the others for all the considered criteria.

This multi-criteria scenario requires a further development of auto-tuners, which must be able to capture these trade-offs and offer the user either the whole Pareto set or a solution within it. Although there is a growing amount of related work considering the optimization of several criteria [2, 3, 4, 5, 6], most of them consider two criteria simultaneously at most, and many fail in capturing the trade-off among the objectives.

In this paper we investigate the auto-tuning of parallel codes using the Insieme compiler to optimize three different criteria: execution time, resource usage and energy consumption. For tuning the codes, we consider as optimization knobs: dynamic concurrency throttling (DCT, later on referred to as used cores), loop tiling, and frequency and voltage scaling (DVFS). We examine the obtained results in detail to analyze and illustrate the complex interactions between optimized software and hardware. To the best of our knowledge, this is the first work exploring an auto-tuner to optimize parallel programs for more than two objectives and analyzing trade-offs among these objectives. Our main findings of this work demonstrate that: (1) RS-GDE3 can be successfully applied to a three-objective optimization problem without any modifications or restrictions and (2) the trade-off between execution time and energy consumption, dependent on efficient parallelization, can be explained by investigating resource usage. Furthermore, we compare RS-GDE3 with a state-of-the-art multi-objective optimizer (NSGA-II) that has been adjusted to deal with three objectives. The results show that RS-GDE3 derives solutions with better quality than an NSGA-II-based solution.

The paper is structured as follows: Section 2 describes the auto-tuning infrastructure used for this work. The experiment design, the objectives of interest, the target codes and hardware platform are outlined in Section 3. Section 4 presents our results and their detailed analysis. Finally relevant related work is listed in Section 5 and Section 6 concludes.

## 2 Insieme Compiler

### 2.1 Auto-Tuning Infrastructure

Our work is based on the Insieme compiler, a multi-objective auto-tuning optimizing compiler and runtime system for parallel codes [7].

Figure 1 illustrates the overall architecture of Insieme. An input code is loaded by the compiler (1), analyzed and prepared (2) to be tuned prior to execution. During this process, a set of tunable *parameters* are identified, encompassing loop tile sizes, number of cores involved in the computation as well as the frequency setting of the CPUs. Afterwards, the optimizer conducts auto-tuning (hence we use the terms *auto-tuner* and *optimizer* interchangeably) by

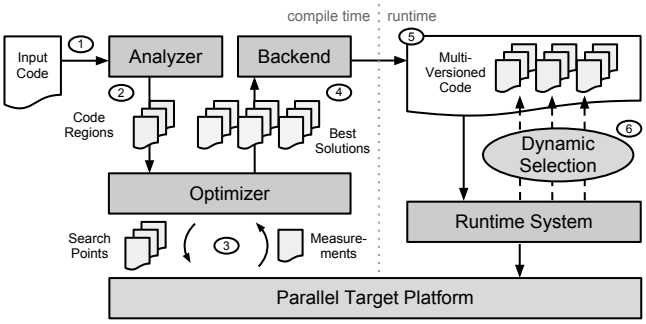


Fig. 1: Overview of the Insieme compiler, adapted from [7].

iteratively selecting sets of *configurations* for each code to be evaluated (executed) on the target system (3). At the end, the optimizer derives a Pareto set consisting of the best configurations found. These are passed to the backend (4) and compiled into multi-versioned code (5). The runtime system can then dynamically select the preferred code version to be executed (6).

### 2.2 Optimizers

The main search engine of Insieme, described in previous work of the authors [7], is called RS-GDE3 and aims at computing the Pareto set of code configurations. RS-GDE3 combines an approximation technique from the class of Differential Evolution (DE) and a search space reduction mechanism based on Rough Set theory. The goal of this latter technique is to reduce the search to a small area where RS-GDE3 assumes the location of the optimal configurations. This method was successfully applied to an optimization problem with two conflicting objectives in [7], whereas we apply it for the first time to three objectives in this work. However, RS-GDE3 is a true multi-objective optimizer that can handle an arbitrary number of objectives within the scope of Pareto optimality.

In addition to RS-GDE3, the Insieme compiler includes two other search engines, which are used in this paper to compare with, based on a hierarchical and a random search. The hierarchical search evaluates points on an equidistant grid defined over each tunable parameter. Random search generates a set of code configurations by randomly setting the values of each tunable parameter.

## 3 Experiment Design

### 3.1 Objectives

In this work we try to optimize parallel programs for three objectives and investigate the trade-offs between them: **execution time**, **resource usage**, and **energy consumption**.

*Execution time* is inherently an objective of interest, as providing results within the shortest possible time is desirable for most programs.

We furthermore include *resource usage*, denoted by $r(x) = x \cdot t_p(x)$ with $x$ being the number of cores involved in executing the program and $t_p(x)$ denoting

Table 1: Code Characteristics.

| Code | Problem Size | Computation | Memory | Tile Sizes | No. of Cores | CPU Freq. (Ghz) | Total No. of Configurations |
|---|---|---|---|---|---|---|---|
| mm | $1200^2$ | $\mathcal{O}(N^3)$ | $\mathcal{O}(N^2)$ | $(1–600)^3$ | 1–32 | 1.2–2.7 + Turbo Boost | $1.11 \cdot 10^{11}$ |
| dsyrk | $1200^2$ | $\mathcal{O}(N^3)$ | $\mathcal{O}(N^2)$ | $(1–600)^3$ | | | $1.11 \cdot 10^{11}$ |
| jacobi-2d | $10000^2$ | $\mathcal{O}(N^2)$ | $\mathcal{O}(N^2)$ | $(1–5000)^2$ | | | $1.28 \cdot 10^{10}$ |
| 3d-stencil | $600^3$ | $\mathcal{O}(N^3)$ | $\mathcal{O}(N^3)$ | $(1–300)^2$ | | | $4.61 \cdot 10^{7}$ |
| n-body | 500000 | $\mathcal{O}(N^2)$ | $\mathcal{O}(N)$ | 1–1000, 1–500000 | | | $2.56 \cdot 10^{11}$ |

the parallel execution time, as an objective to reflect computing costs. Most economic cost models that focus on computational resources, such as the ones used by cloud providers, are based on CPU hours [8]. Similarly, many academic computing centers base their accounting on CPU hours even if users are not charged. Hence, we believe that resource usage (reflecting computing costs – economic or otherwise) is an important optimization goal for parallel applications.

As a third objective of interest we consider *energy consumption*. Reducing it is of interest to both HPC center operators and users (as future cost models might include energy consumption due to its increasing workload dependence). To optimize also for energy, we require information about the energy consumption of parallel programs. The CPU is the largest contributor of the overall energy consumption of a non-accelerated HPC node that can also be influenced the most by the workload executed. Hence, we focus our energy optimization efforts on this component and rely on the Intel RAPL interface. It offers estimations with a resolution of 15.3 microjoules at a rate of 1 KHz for the entire CPU package. Recent related work showed RAPL to be accurate enough for purposes such as ours [9]. It should be noted that we use RAPL due to its wide availability, however the Insieme compiler can use any energy measurement/modeling system that meets the necessary accuracy and resolution requirements.

Let $E_i$ be the energy consumption of a code executed on any number of cores of CPU socket $i \in P$ where $P$ denotes the set of all sockets that have cores participating in the execution of a parallel program. Then $E_{total} = \sum_{i \in P} E_i$ denotes the overall energy consumption of the code. For brevity, we refer to execution time only as time and to energy consumption as energy throughout the rest of the paper.

### 3.2 Benchmarks and Target Platform

Our benchmarks consist of a matrix multiplication kernel (*mm*, using an ijk loop order), a BLAS-3 linear algebra kernel (*dsyrk*, computing $B = A * A^T + B$), two stencil codes (*jacobi-2d* and a generic 3x3x3 *3d-stencil*) and an implementation of an *n-body* simulation. Except for the *mm* and *dsyrk* codes, all of them exhibit distinct computation and memory complexities as listed in Table 1 and hence considerably different memory reuse and access patterns. Furthermore, although identically categorized in terms of complexity, the memory access patterns of *mm* and *dsyrk* are very different since the (on-the-fly) transposition of $A$ eliminates the unaligned matrix access conducted within the *mm* code. Table 1 also lists the tunable parameters and their ranges for each code.

Table 2: Parameter Settings of the Optimizer.

| Algorithm | Parameters |
|---|---|
| **RS-GDE3** | $\|C\| = 30$, $CR = 0.5$, $F = 0.5$ |
| **Hierarchical Search** | 21 values per tiling parameter (2D tiling problems)<br>8 values per tiling parameter (3D tiling problems)<br>6 different numbers of cores<br>6 different frequencies |

The target platform is a quad-socket shared-memory system equipped with Intel Xeon E5-4650 Sandy Bridge EP processors, each offering 8 cores clocked at 1.2–2.7 GHz (up to 3.3 GHz with Turbo Boost). Each core features private L1 and L2 caches of 64 and 256 KB each in addition to the CPU-wide shared L3 cache of 20 MB. The system provides 128 GB of main memory, uses a Linux operating system with a 3.5.0 kernel and our backend compiler is GCC 4.6.3. Hyper-Threading was not used in any of our experiments.

### 3.3 Configuration of the Optimizers

We have run the three optimizers available within the Insieme framework: RS-GDE3, hierarchical search, and random search. The parameters for RS-GDE3 and hierarchical search are described in the following and summarized in Table 2. In the case of RS-GDE3, we need to set the size of set $C$ of code configurations (processed by RS-GDE3), the parameters $CR$ and $F$ required by the DE method, and the termination condition of the algorithm. These values have been determined during a preceding tuning phase , have an impact on the optimization results and may differ for different architectures. As termination condition, RS-GDE3 stops when it does not generate a better code configuration for $m$ consecutive iterations (to be set by the user, 5 in our case).

For the hierarchical search only the sampling grid needs to be defined. It depends on the number of tunable parameters and defines the total number of configurations to be evaluated. We have configured the hierarchical search with a grid such that at least 15000 different configurations are examined. For generating the grid we only need to specify how many equidistant values we consider for every tunable parameter (note that for the number of cores, we only select powers of 2).

Finally, for the random search, we need to specify the number of configurations to be examined (also 15000 for this work) and the probability distribution to be used (uniform probability distribution).

### 3.4 Comparison Criteria

To systematically compare different search-based optimization strategies we use two different metrics: (1) the *efficiency* of each strategy, and (2) the *quality* of the configuration set obtained.

**Efficiency.** $N$ denotes the total number of configurations evaluated and reflects the effort of the auto-tuner. Furthermore, time-to-solution and energy-to-solution respectively refer to the amount of time and energy spent by a search method to arrive at a final configuration set $S$.

Table 3: Performance Comparison of the Different Evaluated Algorithms.

| | Hierarchical Search | | | | Random | | | | RS-GDE3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Code | $N$ | $\|S\|$ | $\|S\|'$ | $V(S)$ | $\overline{N}$ | $\overline{\|S\|}$ | $\overline{\|S\|'}$ | $\overline{V(S)}$ | $\overline{N}$ | $\overline{\|S\|}$ | $\overline{\|S\|'}$ | $\overline{V(S)}$ |
| mm | 18432 | 18 | 2% | 0.00 | 15000 | 4.4 | 0% | 0.33 | 956.2 | 23.4 | 98% | 0.48 |
| dsyrk | 18432 | 21 | 5% | 0.00 | 15000 | 2.2 | 11% | 0.17 | 1149.6 | 24.8 | 98% | 0.31 |
| jacobi-2d | 15876 | 31 | 78% | 0.69 | 15000 | 17.2 | 5% | 0.55 | 1243.6 | 29.8 | 75% | 0.76 |
| 3d-stencil | 15876 | 30 | 22% | 0.75 | 15000 | 24.8 | 60% | 0.61 | 981.4 | 28.2 | 77% | 0.76 |
| n-body | 15876 | 26 | 0% | 0.50 | 15000 | 30 | 17% | 0.70 | 1801.4 | 29.6 | 87% | 0.77 |

**Quality.** Assessing the quality of a configuration which optimizes only one objective can be achieved by simply analyzing its value in that objective. However, comparing configurations of a multi-objective optimization problem is more complex since it requires comparing sets –the computed trade-offs– instead of single values. The *hypervolume* $V(S)$ of a set of non-dominated configurations $S$ is a metric proposed in [10] that solves this problem. It consists of the normalized volume –an area in case of a dual-objective problem– containing configurations that are worse than those contained in $S$. In other words, for any configuration enclosed by that volume there is a configuration in $S$ with better values for all the considered objectives. Obviously, the larger the hypervolume the better the quality of the configurations in $S$. The largest hypervolume value ($V(S) = 1$) belongs to the utopia point (unattainable optimal configuration), i.e. the point consisting of the optimum value for each criterion.

We also propose another metric to evaluate the quality of $S$: the freedom in selection. The metric aims to quantify how many different high quality configurations a technique exposes to the user. Simply using $|S|$ to measure this does not completely address the problem: e.g. a configuration set obtained by strategy A could contain a lot of points dominated by the single point computed with strategy B. For this reason, we also employ $|S|'$, denoting the relative amount of configurations which are not dominated by the configurations computed by any other of the auto-tuners used. Hence, the higher the percentage, the higher the quality of the configurations contained within $S$.

Since random search and RS-GDE3 are stochastic algorithms, results of a single run are not sufficient for a meaningful comparison. In our evaluation we use the arithmetic means $\overline{N}$, $\overline{|S|}$, $\overline{|S|'}$ and $\overline{V(S)}$, derived over five runs, as directly comparable substitutes.

## 4 Experimental Results

### 4.1 RS-GDE3 Evaluation

Table 3 gives an overview of the performance of RS-GDE3 compared to hierarchical and random search with respect to the three considered metrics. It shows that RS-GDE3 needs only 5–12% of the number of evaluations compared to the hierarchical and random search strategies to provide configurations that dominate between 77% and 100% of the configurations offered by the other two. In addition, the configuration sets offered by RS-GDE3 span larger hypervolumes than the configuration sets provided by hierarchical and random search.

Beyond the already low number of evaluations compared to hierarchical and random search, RS-GDE3 requires even less time and energy for finding the final configuration set since it quickly converges on good solutions during the search. Hence, only 0.7–7.2% of the time and 1.2–8% of the energy are required by RS-GDE3 compared to hierarchical and random search. It should be noted that the optimization problem cannot be simplified by sequentially optimizing parameters (e.g. finding an optimal tile size first and then tuning the number of cores), as the optimal choices for these settings are inter-dependent [7].

### 4.2 Energy-Time Trade-off as a Function of Resource Usage

Related work has already shown the existence of a trade-off between time and power consumption [5]. It is easily explained by different levels of CPU usage: faster configurations commonly use a higher number of cores, naturally demanding a higher power budget. Trade-offs between time and energy have been less studied in literature and are more difficult to obtain/explain since energy also depends on time. Thus, any optimization providing a trade-off between time and energy must in-/decrease power consumption disproportionally high compared to the de-/increase in time. Our experiments show that the trade-off between time and energy varies with the resource usage and can expose different behaviors. In the rest of this section, we analyze these results and describe which parameters/situations are responsible for such trade-offs.

For the sake of clarity, we summarize our results using a graphical representation as the one presented by Figure 2a. It shows the time, energy, and resource usage behavior of the set of code configurations computed by RS-GDE3 for different benchmark codes (in this case *mm*). These configurations (described in Table 4) are first ranked according to the number of sockets used; configurations using the same number of sockets are further sorted by increasing resource usage.

In all our evaluated problems (see Figure 2) we can observe two different parts: a part where time and energy are highly positively correlated, and a second one indicating a trade-off between the two. In all the cases, the first part always corresponds to configurations using a single CPU socket. As a consequence, we structure our discussion in two blocks: the *single-socket* and the *multi-socket* case. It should be noted that RS-GDE3 computed configurations that use up to four sockets for all problems except for *jacobi-2d*. This is explained by an average scaling behavior of the *jacobi-2d* code, which reaches its minimal execution time by using 10 cores instead of the maximum of 32. The remaining four codes scale well on our target hardware.

**The single-socket case.** The results show that the configurations using only one socket can be further divided into a subset where reducing time also reduces the energy, and a subset where reducing time increases the energy. Without loss of generality we focus our discussion on the example of matrix multiplication (Figure 2a). When taking resource usage into consideration, we observe that time and energy are highly correlated when resource usage is low; however, this only holds until the resource usage reaches a critical point (configuration no. 5

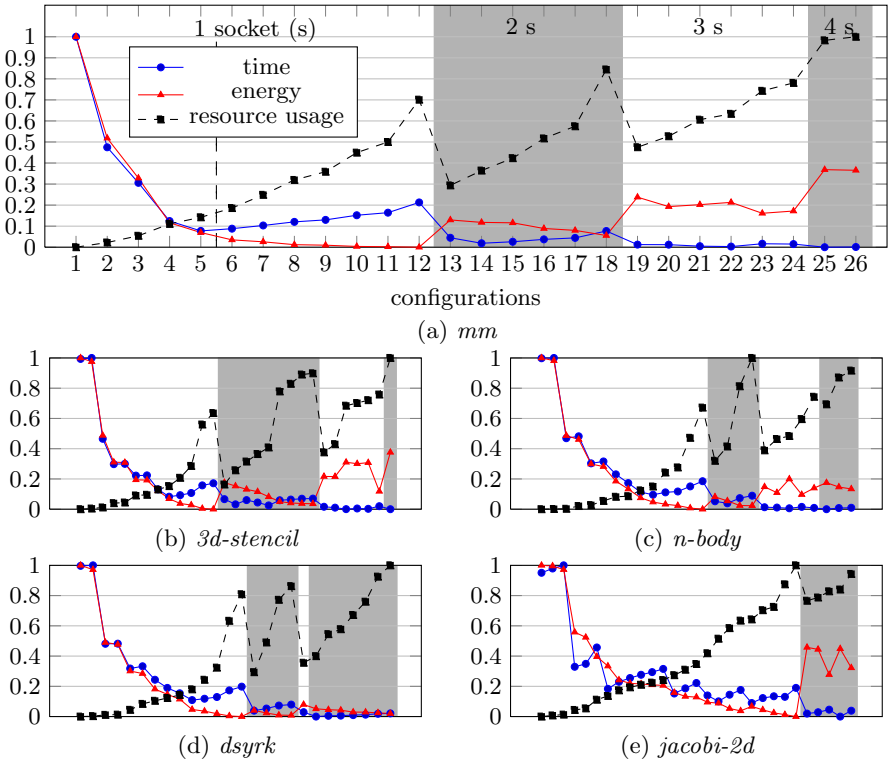


Fig. 2: RS-GDE3 computed trade-offs among time, energy and resource usage.

in Figure 2a), when both, energy and time, become conflicting objectives (i.e. energy can be further reduced from that point onwards while time increases).

A detailed analysis of the computed configurations (listed in Table 4) reveals that they use almost identical tile sizes. These values correspond to an optimal (local or global) tile size configuration found by the auto-tuner. Thus, once this optimal tile size configuration has been found, there are only two tunable parameters influencing the behavior of a code: the number of cores and the clock frequency.

Due to our sorting, the left-most configuration in Figure 2a is the one with the lowest resource usage (only one core in use, at the highest frequency). From this point, increasing the number of used cores reduces the time, and at the same time also the energy. The reason for this behavior can be explained with the power consumption breakdown of the CPU: using a single core requires most off-core entities of a socket to be active, such as the last level cache or the memory controller. Generally, increasing the number of used cores does not require providing additional power to activate those entities. Hence, doubling the number of used cores for example does not usually require double the power. Thus, as both time and power per used core decrease, the overall energy is also reduced. In fact, our experiments show that configurations no. 1–5 in Figure 2a, where time and energy do not conflict, only differ in the number of used cores. Note

Table 4: Details of all *mm* configurations depicted in Figure 2a.

| Conf. No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tile Size A | 37 | 30 | 24 | 31 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 21 |
| Tile Size B | 248 | 248 | 248 | 248 | 236 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 |
| Tile Size C | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| No. of Cores | 1 | 2 | 3 | 6 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 12 |
| CPU Freq. (GHz) | 2.7 | 2.7 | 2.7 | 2.7 | 2.7 | 2.7 | 2.5 | 2.3 | 2.2 | 2.0 | 1.9 | 1.6 | 2.7 |

| Conf. No. | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tile Size A | 18 | 30 | 18 | 30 | 32 | 31 | 25 | 21 | 30 | 15 | 24 | 21 | 24 |
| Tile Size B | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 | 248 |
| Tile Size C | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 6 |
| No. of Cores | 16 | 16 | 16 | 16 | 16 | 19 | 20 | 23 | 23 | 24 | 24 | 32 | 32 |
| CPU Freq. (GHz) | 2.7 | 2.6 | 2.3 | 2.2 | 1.7 | 2.7 | 2.6 | 2.7 | 2.3 | 2.7 | 2.3 | 2.7 | 2.7 |

that this holds only for scalable codes such as the ones used in our experiments. If a code does not scale sufficiently, parallelization may lead to a disproportionally low decrease in time compared to the increase in power, and the overall energy will increase as well. Since we target HPC codes, we assume scalability for the rest of the analysis. Our first observation can then be stated as follows: *1. Assuming scalable codes, parallelism is a way of reducing both time and energy when using a single socket computing system if the other parameters are kept invariable.*

The second way of modifying the behavior with regard to the left-most configuration is via frequency tuning. Lowering the frequency – despite possibly decreasing the energy – increases time. The results of RS-GDE3 show that frequency tuning leads to dominated configurations if it is applied before fully exploiting parallelism. The reason explaining this is very simple. For every other configuration, the optimizer finds a configuration with increased parallelism reducing the time and obtaining a higher energy reduction than by using lower frequencies. Our second observation can be stated then as: *2. In a single-socket scenario, parallelism allows for higher rates of energy reduction than frequency tuning and, in addition, reduces time.*

Once the maximum number of cores has been reached, the auto-tuner exploits frequency tuning. These configurations correspond to the second part of the graph, where energy and time are conflicting objectives. As follows from our previous discussion, decreasing the time is no longer possible since parallelism has been already exploited and all cores are working at their maximum frequencies. Decreasing the frequency will naturally increase the execution time but energy reductions can be achieved, caused by the cube root rule [11]: the power consumption of a CPU scales cubically as long as its voltage changes with the frequency in a correlated fashion; however, the performance of a code usually scales at most linearly with the CPU clock frequency. Hence, a trade-off between time and energy is formed and continues up to the energy-optimal frequency setting. This energy-optimal setting is workload-dependent and was found to be around 1.5 GHz on our target platform by our auto-tuner, as lower frequencies show an increase in energy (because the CPU voltage cannot be scaled down accordingly by the hardware). Thus, as lower frequencies would worsen all three objectives, such configurations are rejected by the optimizer. Our third observation in this

case is: *3. When parallelism has been already exploited, energy can still be further reduced by the sake of slightly increasing time, via applying frequency tuning.*

**The multi-socket case.** Again, without loss of generality we focus on the results depicted by Figure 2a. According to the results illustrated in that graph, moving to a configuration using an increased number of sockets has been successfully exploited by the auto-tuner. In such situations, RS-GDE3 has always found a configuration which reduces the time compared to configurations using a lower number of cores (see for example the first configurations using two, three, or four sockets in Figure 2a). However, this jump to a higher number of sockets always comes with an increase in energy. Thus, our observation (1) in the previous section does not hold in the case of using multiple sockets due to the required energy to operate additional sockets. This fact allows us to state our fourth observation: *4. Multiple sockets can be exploited to decrease the execution time of an application but not to further reduce its energy.*

Our experiments also reveal that, when using more than one socket, the number of cores leading to optimal trade-off configurations does not gradually increase as in the single socket case, but almost instantly reaches the maximum number. This results in our fifth observation: *5. Optimal trade-off configurations using more than one socket span over the maximum number of available cores.*

We also observe that the energy can be reduced by the sake of increasing the time. This situation corresponds to observation (3), where the auto-tuner reduced the frequency for energy savings. Therefore, that observation also applies to the case of configurations involving several sockets at a full utilization level.

In addition to the results presented so far, we investigated whether Turbo Boost might have any effect on our observations. Our experiments showed that, while Turbo Boost allows RS-GDE3 to generate additional solutions (with lower execution time and higher energy compared to not using Turbo Boost, therefore extending the solution set in one direction), all our observations are valid whether Turbo Boost is enabled or disabled.

### 4.3 Comparison of RS-GDE3 with NSGA-II

We have shown the potential of our RS-GDE3 method for three-objective auto-tuning compared to a hierarchical and a random search. The aim of this section is to empirically evaluate RS-GDE3 when compared to other multi-objective optimizers that have been adjusted to deal with three objectives. Nevertheless, it should be noted that without such modification, none of them can be used for auto-tuning with three conflicting objectives. To that end, we chose NSGA-II [12], the most popular algorithm for multi-objective optimization. For a fair comparison, we configured NSGA-II to evaluate the same number of configurations as RS-GDE3. Table 5 lists the results of this comparison for each of our benchmark codes. It shows that the Pareto sets obtained by RS-GDE3 span larger hypervolumes than the ones achieved by NSGA-II, hence providing better solutions. Furthermore, RS-GDE3 offers at least the same number of solutions as NSGA-II. Thus, overall, RS-GDE3 outperforms NSGA-II.

Table 5: Performance Comparison of RS-GDE3 with NSGA-II.

| | RS-GDE3 | | NSGA-II | |
|---|---|---|---|---|
| Code | $\|S\|$ | $V(S)$ | $\|S\|$ | $V(S)$ |
| mm | 17 | 0.65 | 17 | 0.64 |
| dsyrk | 20 | 0.93 | 8 | 0.78 |
| jacobi-2d | 30 | 0.83 | 30 | 0.74 |
| 3d-stencil | 25 | 0.93 | 20 | 0.87 |
| n-body | 30 | 0.88 | 30 | 0.82 |

## 5 Related Work

There is a wide range of related work in the field of auto-tuning. One possible approach is machine learning (ML), however it has never been used in a truly multi-objective fashion. Search-based methods as used in Active Harmony [1] pose an alternative to ML. They have been successfully applied for computing the whole set of Pareto efficient solutions for up to two criteria, (e.g. execution time and efficiency [7] or execution time and compilation time [4]).

The recent concern for power and energy consumption is reflected in the growing amount work applying auto-tuning to optimize them. Whether they consider power or energy, in addition to execution time, most of them fail to capture the full trade-off and only compute a single solution. Some works use models for power/energy and execution time and apply dynamic programming for optimization [2], while others obtain real power measurements [3]. Similar efforts include exploiting slack time for example in OpenMP [6]. However, hardly any of these approaches compute the full Pareto set of solutions. Reducing this trade-off to a predefined number of solutions may limit the freedom of selecting a solution and render detailed trade-off analyses impossible. To the best of our knowledge, [13, 14] are two of the few works investigating that trade-off.

To the best of our knowledge, this is the first application of an auto-tuner to optimize three objectives. We also provide a detailed analysis of the identified trade-offs. While present in several related works, they do not directly deal with optimization or auto-tuning. They rather analyze trade-offs for changing hardware or software configurations. Predominantly using manually preselected solutions, instead of automatically obtained ones, many investigate DVFS or DCT [15], while some evaluate application model changes [16].

## 6 Conclusion

In this work, we have shown the application of a multi-objective auto-tuner which optimizes for three conflicting criteria: execution time, resource usage and energy consumption. We compared RS-GDE3 with a hierarchical and a random search and showed that it requires at least 93% less time and 92% less energy to obtain solutions of equal or higher quality in a benchmark composed of five representative codes. A comparison to a modified state-of-the-art optimizer, NSGA-II, shows that RS-GDE3 offers solutions of higher quality. We identified the complex relationships between the three objectives and the effect of our tunable parameters on them. Our results have been outlined with clear observations to be used to guide the development of auto-tuners and code optimization.

**Acknowledgements** This research has been partially funded by the Austrian Research Promotion Agency under contract 834307 (AutoCore) and by the FWF Austrian Science Fund under contracts I01079 (GEMSCLAIM) and W 1227-N16 (DK-plus CIM).

## References

[1] Tapus, C., Chung, I., Hollingsworth, J.: Active harmony: Towards automated performance tuning. In: Supercomputing, 2002 Conference, IEEE (2002)
[2] Li, D., de Supinski, B.R., Schulz, M., et al.: Strategies for energy-efficient resource management of hybrid programming models. Parallel and Distributed Systems, IEEE Transactions on **24**(1) (2013) 144–157
[3] Tiwari, A., Laurenzano, M., Carrington, L., et al.: Auto-tuning for energy usage in scientific applications. In: Euro-Par 2011: Parallel Processing Workshops, Springer (2012)
[4] Hoste, K., Eeckhout, L.: Cole: compiler optimization level exploration. In: Proc. of the 6th Intl. Symposium on Code generation and optimization, ACM (2008)
[5] Rahman, S., Guo, J., Bhat, A., et al.: Studying the impact of application-level optimizations on the power consumption of multi-core architectures. In: Proc. of the 9th conference on Computing Frontiers, ACM (2012)
[6] Dong, Y., Chen, J., Yang, X., et al.: Energy-oriented openmp parallel loop scheduling. In: Parallel and Distributed Processing with Applications, 2008. ISPA'08. International Symposium on, IEEE (2008)
[7] Jordan, H., Thoman, P., Durillo, J., et al.: A multi-objective auto-tuning framework for parallel codes. In: Supercomputing, 2012 Conference, IEEE (2012)
[8] Fox, A., Griffith, R., Joseph, et al.: Above the clouds: A berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS **28** (2009)
[9] Hähnel, M., Döbel, B., Völp, M., et al.: Measuring energy consumption for short code paths using RAPL. SIGMETRICS Perform. Eval. Rev. **40**(3) (January 2012)
[10] Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation **3**(4) (1999)
[11] Flynn, M., Hung, P., Rudd, K.: Deep submicron microprocessor design issues. Micro, IEEE **19**(4) (1999)
[12] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. Evolutionary Computation, IEEE Transactions on **6**(2) (2002) 182–197
[13] Freeh, V., Lowenthal, D.: Using multiple energy gears in mpi programs on a power-scalable cluster. In: Proc. of the 10th ACM SIGPLAN PPoPP, ACM (2005)
[14] Balaprakash, P., Tiwari, A., Wild, S.: Multi-objective optimization of hpc kernels for performance, power, and energy. In: 4th International Workshop on Performance Modeling, Benchmarking, and Simulation of HPC Systems (PMBS12). (2013)
[15] Freeh, V., Lowenthal, D., Pan, F., et al.: Analyzing the energy-time trade-off in high-performance computing applications. Parallel and Distributed Systems, IEEE Transactions on **18**(6) (2007)
[16] Lively, C., Wu, X., Taylor, V., et al.: Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems. International Journal of High Performance Computing Applications **25**(3) (2011)