Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy

Ravi Reddy, and Alexey Lastovetsky, *Member, IEEE*

Abstract—Performance and energy are now the most dominant objectives for optimization on modern parallel platforms composed of multicore CPU nodes. The existing intra-node and inter-node optimization methods employ a large set of decision variables but do not consider problem size as a decision variable and assume a linear relationship between performance and problem size and between energy consumption and problem size. We demonstrate using experiments of real-life data-parallel applications on modern multicore CPUs that these relationships have complex (non-linear and even non-convex) properties and, therefore, that the problem size has become an important decision variable that can no longer be ignored. This key finding motivates our work in this paper.
In this paper, we first formulate the bi-objective optimization problem for performance and energy (BOPPE) for data-parallel applications on homogeneous clusters of modern multicore CPUs. It contains only one but heretofore unconsidered decision variable, the problem size. We then present an efficient and exact global optimization algorithm called *ALEPH* that solves the *BOPPE*. It takes as inputs, discrete functions of performance and dynamic energy consumption against problem size and outputs the globally Pareto-optimal set of solutions. The solutions are the workload distributions, which achieve inter-node optimization of data-parallel applications for performance and energy. While existing solvers for *BOPPE* give only one solution when the problem size and number of processors are fixed, our algorithm gives a diverse set of globally Pareto-optimal solutions. The algorithm has time complexity of $O(m^2 \times p^2)$ where m is the number of points in the discrete speed/energy function and p is the number of available processors.
We experimentally study the efficiency and scalability of our algorithm for two data parallel applications, matrix multiplication and fast Fourier transform, on a modern multicore CPU and homogeneous clusters of such CPUs. Based on our experiments, we show that the average and maximum sizes of the globally Pareto-optimal sets determined by our algorithm are 15 and 34 and 7 and 20 for the two applications respectively. Comparing with load-balanced workload distribution solution, the average and maximum percentage improvements in performance and energy respectively demonstrated for the first application are (13%,97%) and (18%,71%). For the second application, these improvements are (40%,95%) and (22%,127%). Assuming 5% performance degradation from the optimal is acceptable, the average and maximum improvements in energy consumption demonstrated for the two applications respectively are 9% and 44% and 8% and 20%.
Using the algorithm and its building blocks, we also present a study of interplay between performance and energy. We demonstrate how *ALEPH* can be combined with *DVFS*-based Multi-Objective Optimization (MOP) methods to give a better set of (globally Pareto-optimal) solutions.

Index Terms—homogeneous multicore CPU clusters, data partitioning, load balancing, performance, energy, bi-objective optimization, DVFS

1 Introduction

Performance and energy are now the most dominant objectives for optimization on modern parallel platforms composed of multicore CPUs.

State-of-the-art methods solving the bi-objective optimization problem for performance and energy (*BOPPE*) can be broadly classified as follows:

- *System-level*: Methods that aim to optimize several objectives of the system or the environment (for example: clouds, data centers, etc) where the applications are executed. The leading objectives are performance, energy consumption, cost, and reliability. A core characteristic of the methods is the use of application-agnostic models for predicting the performance of applications and energy consumption of resources in the system.
- *Application-level*: Methods focusing mainly on optimization of applications for performance and energy. These methods use application-level models for predicting the performance and energy consumption of applications.

We present below notable works in the *System-level* category. Mezmaz et al. [1] propose a parallel bi-objective genetic algorithm to maximize the performance and minimize the energy consumption in cloud computing infrastructures. Fard et al. [2] present a four-objective case study comprising performance, economic cost, energy consumption, and reliability for optimization of scientific workflows in heterogeneous computing environments. Beloglazov et al. [3] propose heuristics that consider twin objectives of energy efficiency and Quality of Service (QoS) for provisioning data center resources. Kessaci et al. [4] present a multi-objective genetic algorithm that minimizes the energy consumption, CO2 emissions, and maximizes the generated profit of a cloud computing infrastructure. Durillo et al. [5] propose

- *R. Reddy and A. Lastovetsky are with the School of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland.*
E-mail: ravi.manumachu@ucd.ie,alexey.lastovetsky@ucd.ie

a multi-objective workflow scheduling algorithm that maximizes performance and minimizes energy consumption of applications executing in heterogeneous high-performance parallel and distributed computing systems.

Application-level methods solving *BOPPE* can be further categorized as follows:

- Methods targeting intra-node optimization.
- Methods targeting both intra-node and inter-node optimizations. To the best of our knowledge, there are no efforts that specifically target inter-node optimization of applications for performance and energy. We address this shortcoming in the current work.

Before we summarize the notable efforts in each of the categories, we would like to clearly define the meaning of the terms "problem size" and "workload size" used in our work. These two terms are used synonymously in the literature. The problem size is defined as a set of one, two or more parameters characterizing the amount and layout of data stored and processed during the execution of a computational task [6], [7]. It also represents the size of a computational task that is allocated to a processor during the parallel execution of a data-parallel application. It is the key decision variable in our work. The workload size is defined as the size of the workload of the data-parallel application that is executed using one or more processors. It is a multiple of one or more computational tasks, whose size is defined to be the problem size. It is not a decision variable but an application input. Wherever possible, we use "data-parallel application workload size" to clearly differentiate it from the problem size.

The methods for intra-node optimization employ several decision variables (for example, number of threads, DVFS etc), which model the intra-node performance and energy of their applications. Since they employ node-level parameters, their main target is intra-node optimization of applications for performance and energy. The methods that target both intra-node and inter-node optimization use a large set of intra-node and inter-node decision variables (for example, number of processors, etc.). Along with decision variables, they consider several parameters such as cost of floating-point operations, cost of memory operations, latencies and bandwidths of the communication links, etc., which impact the performance and energy consumption of their applications but which have fixed values in their solution methods. To reduce the complexity of interplay between performance and energy arising from combined effects of many decision variables, they consider the influence individually of only a subset of them. We summarize below works in these categories.

Freeh et al. [8] propose an intra-node optimization approach that analyzes the performance-energy trade-offs of serial and parallel applications on a cluster of DVFS-capable AMD nodes. In their study, they consider three intra-node parameters to characterize the performance and energy of serial and parallel applications. Ishfaq et al. [9] formulate a bi-objective optimization problem for power-aware scheduling of tasks onto heterogeneous and homogeneous multicore processor architectures. Their solution method targets intra-node optimization. They consider intra-node parameters such as DVFS, computational cycles, and core architecture type. Subramaniam et al. [10] use multi-variable regression to study the performance-energy trade-offs of the high-performance LINPACK (HPL) benchmark. They consider three HPL parameters, one intra-node and two inter-node, to study the bi-objective optimization problem. Song et al. [11] propose an iso-energy-efficiency model to study the performance-energy trade-offs. In their model targeted for both intra-node and inter-node optimizations, they use twenty-eight parameters to characterize the performance and energy of applications. Demmel et al. [12] present an intra-node and inter-node optimization approach that studies energy savings at the algorithmic level. In their analysis, they use performance and energy models containing six and twelve parameters respectively. Choi et al. [13] present an energy roofline model at the node level based on the time-based roofline model [14]. Choi et al. [15] extend the roofline model by adding an extra intra-node parameter, power caps, to their execution time model. Both works present an intra-node optimization approach. Drozdowski et al. [16] propose a concept called an iso-energy map, which represents points of equal energy consumption in a multi-dimensional space of system and application parameters. They study three analytical models, two intra-node and one inter-node. For the inter-node model, they consider eight parameters. From all the possible combinations of these parameters, they study twenty-eight combinations and their corresponding iso-energy maps. However, one of the key assumptions in their model is that the energy consumption is constant and independent of problem size. Marszakowski et al. [17] analyze the impact of memory hierarchies on performance-energy trade-off in parallel computations. They study the effects of twelve intra-node and inter-node parameters on performance and energy. In their problem formulations, they represent performance and energy by two linear functions of problem size, one for in-core computations and the other for out-of-core computations.

To summarize, there are no application-level methods that specifically target inter-node optimization of applications for performance and energy. Existing methods do not consider problem size as a decision variable. They consider problem size as an application parameter and assume a linear relationship between performance and problem size and between energy consumption and problem size. However, we demonstrate in our motivation section using experiments of real-life data-parallel applications on modern multicore CPUs that the relationships between performance and problem size and between energy and problem size have complex (non-linear and even non-convex) properties. Therefore, we believe the problem size has become an important decision variable that can no longer be ignored.

This key finding motivates our work in this paper. We first formulate the *BOPPE* for data-parallel applications on homogeneous clusters of modern multicore CPUs. It contains only one but heretofore unconsidered decision variable, the problem size. We then present an efficient and exact global optimization algorithm called *ALEPH* that solves *BOPPE*. The inputs to the algorithm are: (a). Data-parallel application workload size, (b). Number of available processors, (c). Discrete function (functional model) of performance against problem size, (d). Discrete function (functional model) of dynamic energy consumption

against problem size, and (e) User-specified preferences for performance and energy to select a subset of the globally Pareto optimal set. Its output is the globally Pareto-optimal set of solutions, which are the workload distributions that achieve inter-node optimization of data-parallel applications for performance and energy. The algorithm has time complexity of $O(m^2 \times p^2)$ where m is the number of points in the discrete speed/energy function and p is the number of available processors. Its building blocks are the algorithms, called *POPTA* and *EOPTA* (brief overviews in section 9, supplemental). These are proposed in [18] for single-objective optimization of data-parallel applications on homogeneous multicore CPU clusters for performance and energy respectively. While existing solvers for *BOPPE* give only one solution when the problem size and number of processors are fixed, we show that *ALEPH* gives a diverse set of globally Pareto-optimal solutions.

Based on our experiments with two data parallel applications on a modern multicore CPU and simulations on clusters of such CPUs, we demonstrate the efficiency of *ALEPH*. Using the globally Pareto-optimal front determined by *ALEPH* and the paths taken by its fundamental building blocks, *POPTA* and *EOPTA*, we study the interplay between performance and energy. We also demonstrate how *ALEPH* can be combined with *DVFS*-based MOP methods to give a better set of (globally Pareto-optimal) solutions.

To summarize, our main contributions in this paper are:

- Illustration of the new challenges introduced to bi-objective optimization problem for data-parallel applications on homogeneous multicore CPU clusters for performance and energy (*BOPPE*) by the brand new complexities of resource contention and NUMA present in modern multicore CPUs. We demonstrate why problem size has now become an important decision variable that can not be ignored.
- First application-level bi-objective optimization study for data-parallel applications on homogeneous multicore CPU clusters for performance and energy that is not based on DVFS but based on a single decision variable, problem size.
- An efficient and exact global optimization algorithm, *ALEPH*, for solving *BOPPE* that specifically targets inter-node optimization of data-parallel applications for performance and energy. Unlike existing solvers, which assume linear relationship between performance and problem size and energy and problem size, *ALEPH* takes as inputs, discrete functions of performance and dynamic energy consumption against problem size. We show that these functions realistically represent the performance and energy profiles of data-parallel applications on modern multicore CPUs. We show that *ALEPH* gives a diverse set of globally Pareto-optimal solutions whereas existing solvers give only one solution when the problem size and number of processors are fixed. We also show that *ALEPH* can be combined with *DVFS*-based MOP methods to give a better set of (globally Pareto-optimal) solutions.
- We demonstrate how the Pareto-optimal front (determined by *ALEPH*) and paths of optimization algorithms for performance and energy (*POPTA* and *EOPTA*) can be used together to analyze the interplay between performance and energy.

The rest of the paper is organized as follows. Section 2 presents the challenges posed to solve *BOPPE* by the inherent complexities in modern multicore CPUs. Section 3 presents related work on bi-objective optimization problems for parallel platforms. Section 4 contains theory and notation of multi-objective optimization and the concept of optimality. We then formulate the bi-objective optimization of data-parallel applications on homogeneous clusters of multicore CPU nodes for performance and energy and analyze classical methods to solve the problem. We discuss why the state-of-the-art solvers are inadequate for direct solution of the bi-objective optimization problem. Section 5 presents *ALEPH*, an efficient exact global optimization algorithm solving *BOPPE*. Section 6 contains experimental analysis of the algorithm and study of interplay between performance and energy using *ALEPH* and its building blocks, *POPTA* and *EOPTA*. Section 7 concludes the paper.

2 Motivation: Performance and Energy of Data-parallel Applications on Homogeneous Clusters of Multicore CPUs

In this section, we present the dramatic changes observed in performance and energy profiles of real-life scientific data-parallel applications executing on parallel platforms with multicore CPUs compared to parallel platforms composed of uniprocessors. Using this presentation, we specifically highlight the challenges posed to solving *BOPPE* by the new complexities in modern multicore CPUs. At the same time, we demonstrate why problem size has now become an important decision variable that can not be ignored.

In parallel platforms with uniprocessors, the performance and energy profiles of real-life scientific data-parallel applications were smooth and exhibited certain properties, which essentially rendered the bi-objective optimization problem of performance and energy (*BOPPE*) mono-objective in the sense that optimizing for performance also optimized for energy. This meant that users had to use hardware-level intra-node parameters such as DVFS to tackle *BOPPE* where they reported noteworthy improvements in energy consumption while causing minimal degradation of performance.

However, due to new formidable challenges imposed by the inherent complexities in modern multicore CPUs, *BOPPE* has become very difficult to solve. This is because the cores in a modern multicore CPU are very tightly integrated and arranged in a highly hierarchical fashion. Due to this tight integration, they contend for various shared on-chip resources such as Last Level Cache (LLC) and interconnect (For example: Intel's Quick Path Interconnect, AMD's Hyper Transport), thereby causing severe resource contention and non-uniform memory access (NUMA). These inherent complexities have posed the new challenges to solve *BOPPE*.

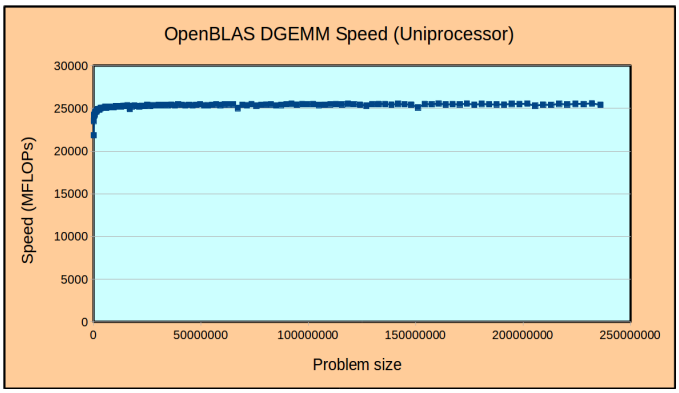
To elucidate these challenges, we compare the typical shapes of real-life scientific data-parallel applications on platforms consisting of uniprocessors and modern multicore

TABLE 1
Specification of the Intel Haswell workstation used to build the uniprocessor speed and energy models.

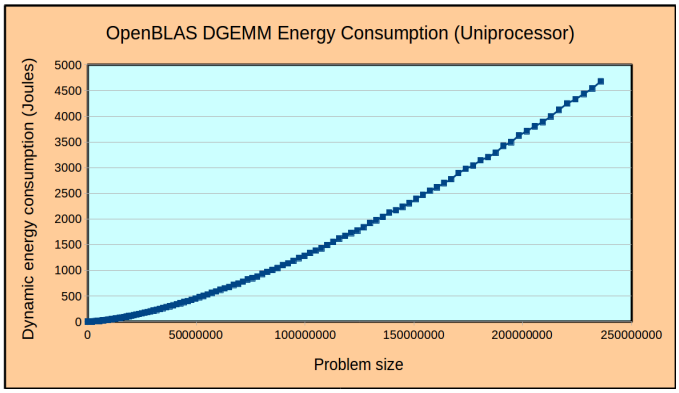
Technical Specifications	Intel Haswell i5-4590
Processor	Intel(R) Core(TM) i5-4590 3.3 GHz
Microarchitecture	Haswell
Memory	8 GB
Socket(s)	1
Core(s) per socket	4
L1d cache, L11 cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 6144 KB
TDP, Base Power	84 W, 22.3 W
Max Turbo Frequency	3.7 GHz

TABLE 2
Specification of the Intel Haswell server used to build the speed and energy functions for multithreaded OpenBLAS DGEMM and FFTW applications.

Technical Specifications	Intel Haswell Server
Processor	Intel E5-2670 v3 @ 2.30GHz
OS	CentOS 7
Microarchitecture	Haswell
Memory	64 GB
Socket(s)	2
Core(s) per socket	12
NUMA node(s)	2
L1d cache, L11 cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
TDP, Base Power	240 W, 58 W

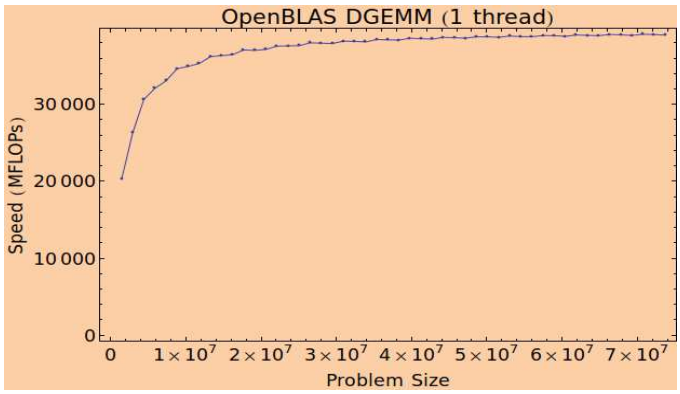


(a)

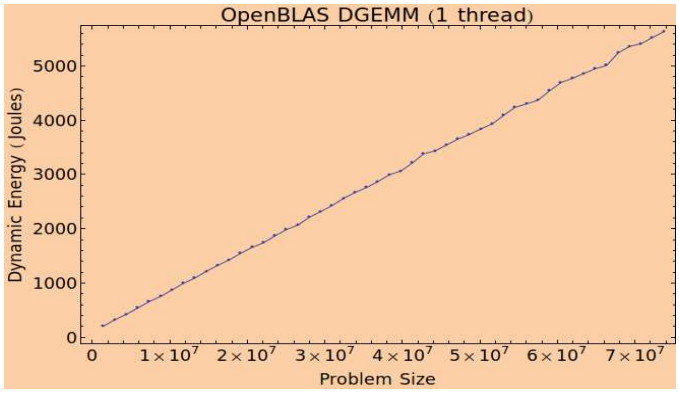


(b)

Fig. 1. (a). Speed function of OpenBLAS DGEMM application executed on a single core on the Intel Haswell workstation. (b). Dynamic energy consumption of OpenBLAS DGEMM application executed on a single core on the Intel Haswell workstation.



(a)



(b)

Fig. 2. (a). Speed function of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server. (b). Dynamic energy consumption of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server.

CPUs. For this purpose, we select two widely known and highly optimized scientific routines, OpenBLAS DGEMM [19] and FFTW [20].

Consider the profiles of the speed and dynamic energy consumption functions of the OpenBLAS DGEMM application built experimentally by executing it on a single core of an Intel Haswell workstation (specification shown in Table 1). The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2). In these experiments, the *numactl* tool is used to bind the application to one core. The dynamic energy consumptions are obtained using Watts Up Pro power meter. We must mention that there are two types of energy consumptions, dynamic energy and static energy. We define the static energy consumption as the energy consumption of the platform without the application execution. The static energy consumption of the platform during the application execution is calculated by multiplying the idle power (static power) of the platform by the execution time of the application. Dynamic energy consumption is calculated by subtracting this static energy consumption from the total energy consumption of the platform during the application execution measured using the Watts Up Pro. In this work, we consider only the dynamic energy consumption due to several reasons, which are explained in Section 2 of the supplemental. However, we would like to mention that static power consumption can be easily incorporated in our problem formulations and algorithms.

Fig. 1a and 1b respectively show the shapes, whose properties can be summarized below:

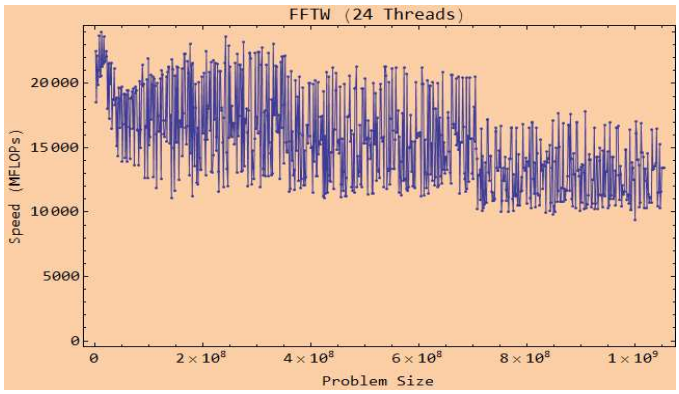
- The functions are smooth.
- The speed function satisfies the following properties:
  - Monotonically increasing.
  - Concave.
  - Any straight line coming through the origin of the coordinate system intersects the graph of the function in no more than one point.
- The dynamic energy consumption is a monotonically increasing convex function of problem size.

Lastovetsky et al. [21], [22], [18] prove that for such shapes, the solutions determined by the traditional and the state-of-the-art load-balancing algorithms [23], [7], [6], [24], [25], simultaneously minimize the execution time and dynamic energy consumption of computations in the parallel execution of the application. Fig. 2a and 2b respectively show the shapes of the speed and dynamic energy consumption functions of the same application built experimentally by executing it on a single core of an Intel Haswell server (specification shown in Table 2). It can be seen that while the shape of the speed function is the same as before, the shape of the dynamic energy consumption is linear. This implies that all workload distributions will result in same dynamic energy consumption and therefore parallelization has no effect on the dynamic energy consumption of computations in the parallel execution of the application.

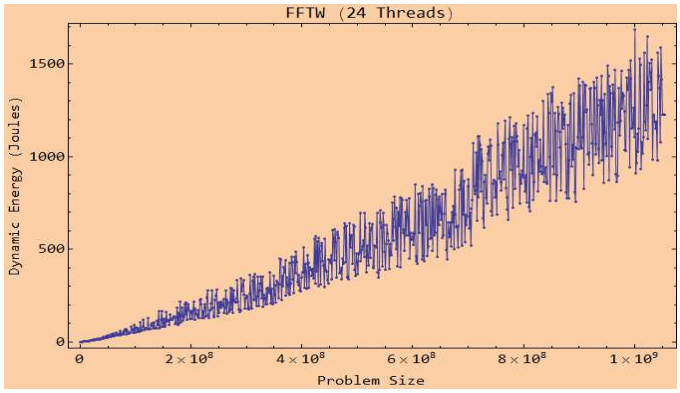
Therefore, on platforms composed of uniprocessors, one can see that solutions determined by load-balancing algorithms that minimize the execution time either minimized the energy consumption or did not have any effect on it. Owing to this, methods solving *BOPPE* used energy-efficiency techniques such as DVFS, where the clock frequency of the processor is dynamically adjusted, to determine any trade-offs between execution time and energy. The main objective of these methods was to minimize the energy consumption but at the same time causing minimal performance degradation.

Now, on modern homogeneous clusters composed of multicore CPUs, due to the newly introduced complexities such as resource contention and NUMA, the performance and energy profiles of real-life scientific applications executing on these platforms are not smooth and may deviate significantly from the shapes observed before. This is illustrated in Fig. 3 and 4, which respectively show the speed function and dynamic energy consumption graphs for multi-threaded OpenBLAS DGEMM and FFTW applications executed with 24 threads on the Intel Haswell server. The FFTW application performs a 2D FFT of size $n \times n$ (the problem size being n^2).

To make sure the experimental results are reliable, we follow a detailed methodology, which is described in Section 3 in the supplemental. It contains the following main steps: 1). We make sure the server is fully reserved and dedicated to our experiments and is exhibiting clean and normal behavior by monitoring its load continuously for a week. 2). For each data point in the speed and energy functions of an application, the sample mean is used, which is calculated by executing the application repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations.



(a)

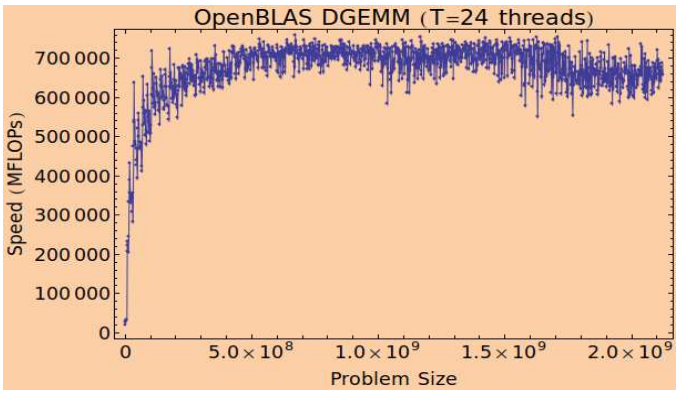


(b)

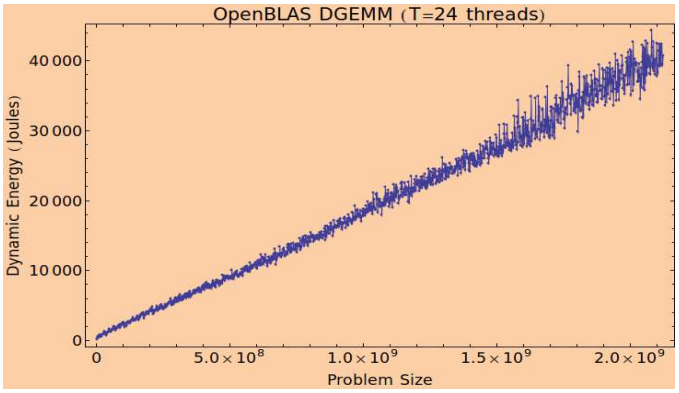
Fig. 3. (a). Speed function of FFTW executing 24 threads on the Intel Haswell server. (b). Function of dynamic energy consumption against problem size for FFTW executing 24 threads on the Intel Haswell server.

Therefore, the variation observed is not noise but is an inherent trait of applications executing on multicore servers with resource contention and NUMA. Other interesting properties about the variations are discussed in [18] and summarized below:

- They can be quite large. This is evident from the speed and energy functions shown in Fig. 3a and 3b respectively. From the speed function plot, one can observe performance drops of around 70% for many problem sizes.
- The variations presented in the paper cannot be explained by the constant and stochastic fluctuations due to OS activity or a workload executing in a node in common networks of computers. In such networks, a node is persistently performing minor routine computations and communications by being an integral part of the network. Examples of such routine applications include e-mail clients, browsers, text editors, audio applications, etc. As a result, the node will experience constant and stochastic fluctuations in the workload. This changing transient load



(a)



(b)

Fig. 4. (a). Speed function of OpenBLAS DGEMM executing 24 threads on the Intel Haswell server. (b). Function of dynamic energy consumption against problem size for OpenBLAS DGEMM executing 24 threads on the Intel Haswell server.

will cause a fluctuation in the speed of the node in the sense that the speed will vary for different runs of the same workload. One way to represent these inherent fluctuations in the speed is to use a speed band rather than a speed function. The width of the band characterizes the level of fluctuation in the speed due to changes in load over time [23], [7], [6]. For a node with uniprocessors, the width of the band has been shown to decrease as the problem size increases. For a node with a very high level of network integration, typical widths of the speed bands were observed to be around 40% for small problem sizes and narrowing down to 3% for large problem sizes. Therefore, as the problem size increases, the width of the speed band is observed to decrease. Therefore, for long running applications, one would observe the width to become quite narrow (3%). However, this is not the case for variations in the presented graphs. The dynamic energy consumption in Fig. 3b and 4b (for the number of threads equal to 24) show the widths of the variations increasing as problem size increases. These widths reach a maximum of 125% and 70% respectively for large problem sizes. The speed functions in Fig. 3a and 4a (for the number of threads equal to 24) demonstrate that the widths are bounded with the averages around 60% and 17% respectively. This suggests therefore that the variation is largely due to the newly introduced complexities and not due to the fluctuations arising from changing transient load.

Although we use two standard scientific kernels to illustrate the drastic variations in performance and energy profiles, these variations have been the central research focus in [21], [22] where the authors study them for a real-life scientific application, Multidimensional Positive Definite Advection Transport Algorithm (MPDATA). MPDATA is a core component of the EULAG (Eulerian/semi-Lagrangian fluid solver) geophysical model [26], which is an established computational model developed for simulating thermo-fluid flows across a wide range of scales and physical scenarios.

Therefore, these variations are not singular and will become natural because chip manufacturers are increasingly favoring and thereby rapidly progressing towards tighter integration of processor cores, memory, and interconnect in their products.

It is these variations that have now made the *BOPPE* difficult to solve. To demonstrate why this is the case, we zoom into the energy function of the OpenBLAS DGEMM application to analyze its properties. The speed functions also exhibit these properties. Figure 5 shows the energy function between two arbitrarily chosen points A and B. Assume, for the sake of simplicity, that we are allowed to use only this partial energy function in our algorithms.

- One can observe that the energy function is characterized by many local minima $(Q_1, Q_2, ...)$ and many local maxima $(P_1, P_2, ...)$. There is one global maximum P and one global minimum Q.
- The function (feasible region) is non-linear and non-convex.
- The function is not differentiable. It lacks first-order and second-order derivatives. Hence, solvers that rely on the existence of derivatives for efficient search cannot be directly used. There are two approaches to deal with this problem. One is to estimate first-order derivatives by finite-differences and in the other, solvers must choose search directions purely based on functional values. One popular approach is to use Nelder-Mead algorithm [27], which is however designed to solve the classical unconstrained optimization problem without derivatives.

We describe in the supplemental (section 6) why state-of-the-art optimization softwares are not directly amenable to tackle optimization problems where objective functions (in this case, execution time and energy) exhibit such complex properties.

To summarize, the new inherent complexities introduced in modern multicore CPUs have made the *BOPPE* difficult to solve. We also show that problem size has become an important decision variable.

3 Related Work

In this section, we survey the state-of-the-art solution methods solving *BOPPE*.

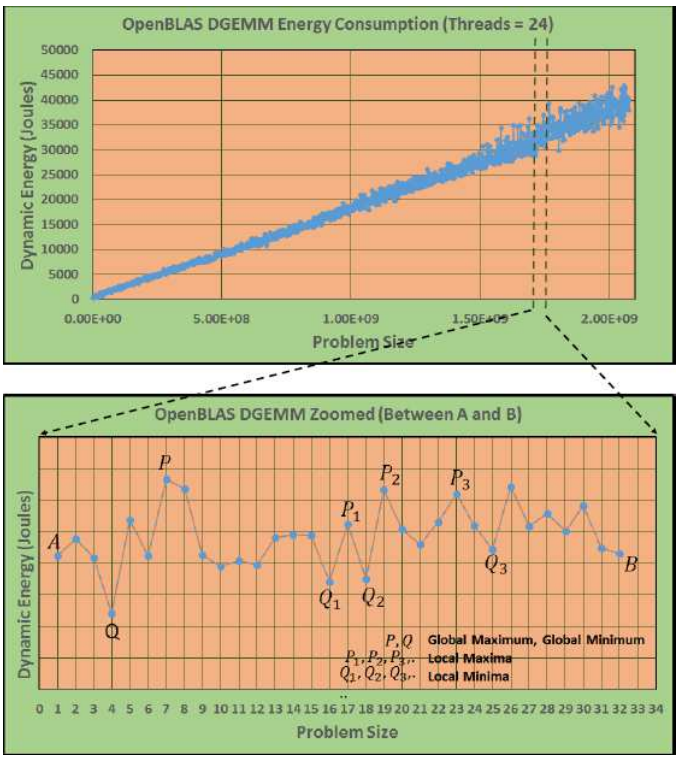


Fig. 5. Zoomed energy function of OpenBLAS DGEMM application between two arbitrarily chosen points A and B. The points are connected by dashed lines for clarity.

3.1 System-level Methods

In this section, we present system-level solution methods for solving *BOPPE*. These are methods that aim to optimize several objectives of the system or the environment (for example: clouds, data centers, etc) where the applications are executed. We will focus on works that consider performance and energy consumption as two prominent objectives. All these methods propose heuristics. Our summary of each work contains the parameters and decision variables that are used in it and the relationships between the objectives and parameters (and decision variables).

Mezmaz et al. [1] propose a parallel bi-objective genetic algorithm to maximize the performance and minimize the energy consumption in cloud computing infrastructures. The parameters used in their method are the computation cost of a task (w) and the communication costs between two tasks. The decision variable is the supply voltage (V) of the processor. Energy consumption of computations is modeled as a function of $V^2 \times w$.

Fard et al. [2] present a four-objective case study comprising performance, economic cost, energy consumption, and reliability for optimization of scientific workflows in heterogeneous computing environments. The parameters are the computation speeds of the processors and the bandwidths of the communication links connecting a pair of processors. The decision variable is the task assignment or mapping. The energy consumption of computations is modeled as cube-root of clock frequency.

Beloglazov et al. [3] propose heuristics that consider twin objectives of energy efficiency and Quality of Service (QoS) for provisioning data center resources. The decision variables are the number of VMs and clock frequencies. The energy consumption is modeled as a linear function of CPU utilization.

Kessaci et al. [4] present a multi-objective genetic algorithm that minimizes the energy consumption, CO2 emissions and maximizes the generated profit of a cloud computing infrastructure. The parameters are the execution time of an application, the number of processors used in the execution of an application, and the deadline for completion of the application. The decision variable is the arrival rate. The energy consumption is calculated as a product of execution time and power consumption, which is modeled using the formula $\alpha \times f^3 + \beta$, where f is the clock frequency.

Durillo et al. [5] propose a multi-objective workflow scheduling algorithm that maximizes performance and minimizes energy consumption of applications executing in heterogeneous high-performance parallel and distributed computing systems. A machine is characterized using nine parameters (from technology(nm) to TDP). They study the impact of different decision variables: number of tasks, number of machines, DVFS levels, static energy, and types of tasks. The execution time and energy consumption are predicted using neural networks.

Kolodziej et al. [28] propose multi-objective genetic algorithms that aim to maximize performance and energy consumption of applications executing in green grid clusters and clouds. The performance is modeled using computation speed of a processor. The decision variable is the DVFS level. Energy consumption is modeled using the equation, $\gamma \times V^2 \times f \times t_e$, where γ is a constant for a processor, V is the supply voltage, f is the clock frequency, and t_e is the estimated completion time.

3.2 Application-level Methods

In this section, we present application-level solution methods for solving *BOPPE* for parallel platforms. We focus exclusively on three aspects of each method: a). Type of optimization achieved, b). Parameters and decision variables used, and c). The relationship of performance and energy consumption with the parameters and decision variables.

It should be noted the surveyed efforts do not consider problem size as a decision variable. In our work, we propose an efficient inter-node optimization method that is based purely on one decision variable, the problem size.

3.2.1 Intra-node Methods

Freeh et al. [8] is an intra-node optimization method that analyzes the performance-energy trade-offs of serial and parallel applications on a cluster of DVFS-capable AMD nodes. They use three parameters in their study: β, which compares the application slowdown to the CPU slowdown, *memory pressure*, which is determined using hardware performance counters such as memory of operations retired and L2 cache misses, and *slack*, which predicts communication bottlenecks.

Ishfaq et al. [9] formulate a bi-objective optimization problem of power-aware scheduling of tasks onto heterogeneous and homogeneous multicore processor architectures. The twin objectives in their problem formulation are minimization of energy consumption and the makespan of computationally intensive scientific problems. Their approach

aims to achieve intra-node optimization by considering node-level parameters such as DVFS, computational cycles, and core architecture type in their optimization problem. They mention a solution that combines a classical game-theoretic approach and Karush-Kuhn-Tucker (KKT) conditions to simultaneously optimize both the objectives.

Choi et al. [13] present an energy roofline model based on the time-based roofline model [14]. Choi et al. [15] extend the roofline model by adding an extra parameter, *power caps*, to their execution time model. These two works [13], [15] present an intra-node optimization approach to study the performance-energy trade-offs.

Balaprakash et al. [29] is an intra-node optimization approach that explores trade-offs among power, energy, and performance using various application-level tuning parameters such as number of threads and hardware parameters such as DVFS.

3.2.2 Intra-node and Inter-node Methods

Subramaniam et al. [10] use multi-variable regression to study the performance-energy trade-offs of the high-performance LINPACK (HPL) benchmark. Their model contains four parameters: N, the problem size, NB, the block size, P, Q, the rows and columns respectively of the process grid. If the problem size and number of processors are fixed, their approach gives a single solution whereas our algorithm gives a diverse set of globally Pareto-optimal solutions. They study performance-energy tradeoffs using the following decision variables separately: a). Threads, b). Number of nodes, and c). DVFS levels.

Song et al. [11] propose an iso-energy-efficiency model to quantify the improvements in energy consumption of parallel applications. It is based on lines similar to performance iso-efficiency function. The energy improvement (of parallel over sequential application) is studied using pairs of decision variables: level of parallelism, clock frequency, and problem size.

Demmel et al. [12] present an intra-node and inter-node optimization approach that studies energy savings at the algorithmic level. They prove that a region of perfect strong scaling in energy exists for matrix multiplication (classical and Strassen) and the direct ($O(n^2)$) n-body problem. This means that for a given problem size n, the energy consumption remains constant as the number of processors p increases and the runtime decreases proportionally to p. The performance is modeled as a linear function of parameters representing costs of computations and communications. The energy consumption is modeled as a linear function of parameters representing costs of computations, communications, and leakage (static power).

Drozdowski et al. [16] propose a concept called an iso-energy map, which represent points of equal energy consumption in a multi-dimensional space of system and application parameters. They use iso-energy maps to study performance-energy trade-offs. They present iso-energy maps for three models of parallel computations, Amdahl's law, Gustafson's law, and divisible loads representing data-parallel computations in distributed systems. For the models for Amdahl's law and Gustafson's law, three intra-node parameters are used: f, the size of the parallel portion of the application, m, the number of processors in the system, and k, which represents the ratio of powers in active and idle states. They do not consider problem size as a decision variable. One of the key assumptions in their model is that the energy consumption is constant and independent of problem size. This they confirm to be true for the applications and the platform used in their experiments.

Marszakowski et al. [17] analyze the impact of memory hierarchies on time-energy trade-off in parallel computations, which are represented as divisible loads. They represent execution time and energy by two linear functions on problem size, one for in-core computations and the other for out-of-core computations. They use this formalization in their bi-objective optimization problem of minimizing time and energy in parallel processing of divisible loads. Due to large parameter space, they restrict their study of performance-energy trade-offs where they just have one parameter, the number of processors (all the other parameters have fixed values).

The works presented in this section do not consider problem size as a decision variable. In our work, we consider only one decision variable, the problem size, and the performance and dynamic energy consumption are represented by highly non-linear non-convex functions of problem size. We show using experiments on a modern multicore CPU that these functions are highly non-linear and non-convex and cannot be represented by piece-wise linear functions.

4 Bi-Objective Optimization for Performance and Energy on Homogeneous Multicore Clusters: Problem Formulation

In this section, we present the theory of multi-objective optimization and the concept of optimality. We then formulate the bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy.

4.1 Multi-Objective Optimization (MOP): Background

A multi-objective optimization (MOP) problem may be defined as follows [30], [31]:

$$\begin{aligned} &minimize \quad \{\mathcal{F}(x) = (f_1(x), ..., f_k(x))\} \\ &\text{Subject to} \quad x \in \mathcal{S} \end{aligned}$$

where there are $k(\geq 2)$ objective functions $f_i : \mathbb{R}^p \rightarrow \mathbb{R}$. The objective is to minimize all the objective functions simultaneously.

$\mathcal{F}(x) = (f_1(x), ..., f_k(x))^T$ denotes the vector of objective functions. The decision (variable) vectors $x = (x_1, ..., x_p)$ belong to the (non-empty) feasible region (set) $\mathcal{S}$, which is a subset of the decision variable space $\mathbb{R}^p$. We call the image of the feasible region represented by $\mathcal{Z}$ $(= f(\mathcal{S}))$, the feasible objective region. It is a subset of the objective space $\mathbb{R}^k$. The elements of $\mathcal{Z}$ are called objective (function) vectors or criterion vectors and denoted by $\mathcal{F}(x)$ or $z = (z_1, ..., z_k)^T$, where $z_i = f_i(x), \forall i \in [1, k]$ are objective (function) values or criterion values.

If there is no conflict between the objective functions, then a solution x^* can be found where every objective function attains its optimum [31].

$$\forall x \in \mathcal{S}, f_i(x^*) \leq f_i(x), \quad i = 1, ..., k$$

However, in real-life multi-objective optimization problems, the objective functions are at least partly conflicting. Because of this conflicting nature of objective functions, it is not possible to find a single solution that would be optimal for all the objectives simultaneously. In multi-objective optimization, there is no natural ordering in the objective space because it is only partially ordered. Therefore we must treat the concept of optimality differently from single-objective optimization problem. The generally used concept is *Pareto-optimality*.

Definition 1. *A decision vector $x^* \in \mathcal{S}$ is Pareto-optimal if there does not exist another decision vector $x \in \mathcal{S}$ such that $f_i(x) \leq f_i(x^*), \forall i = 1, ..., k$ and $f_j(x) < f_j(x)^*$ for at least one index j [30].*

An objective vector $z^* \in \mathcal{Z}$ is Pareto-optimal if there does not exist another objective vector $z \in \mathcal{Z}$ such that $z_i \leq z_i^*, \forall i = 1, ..., k$ and $z_j < z_j^*$ for at least one index j.

Definition 2. *A decision vector $x^* \in \mathcal{S}$ is weakly Pareto-optimal if there does not exist another decision vector $x \in \mathcal{S}$ such that $f_i(x) < f_i(x^*), \forall i = 1, ..., k$ [30].*

An objective vector $z^* \in \mathcal{Z}$ is Pareto-optimal if there does not exist any other vector for which all the component objective vector values are better.

Definition 3. *A decision vector $x^* \in \mathcal{S}$ is properly Pareto-optimal if it is Pareto-optimal and if there is a real number $M > 0$ such that for each f_i and each $x^* \in \mathcal{S}$ satisfying $f_i(x) < f_i(x^*)$, there exists at least one f_j such that $f_j(x^*) < f_j(x)$ and $\frac{f_i(x^*) - f_i(x)}{f_j(x) - f_j(x^*)} \leq M$ [30].*

An objective vector $z^* \in \mathcal{Z}$ is properly Pareto-optimal if the decision vector corresponding to it is properly Pareto-optimal. A solution is properly Pareto-optimal if there is at least one pair of objectives for which a finite decrement in one objective is possible only at the expense of some reasonable increment in the other objective. Essentially speaking, unbounded trade-offs are not allowed between objectives in properly Pareto-optimal set of solutions [30].

Mathematically speaking, every Pareto-optimal point is an equally acceptable solution of the multi-objective optimization problem. Therefore, user preference relations (or preferences of decision maker) are provided as input to the solution process to select one or more points from the set of Pareto-optimal solutions [30].

In Figure 6, a feasible region $\mathcal{S} \subset \mathbb{R}^3$ and its image, a feasible objective region $\mathcal{Z} \subset \mathbb{R}^2$, are shown. The thick blue line in the figure showing the objective space contains all the Pareto-optimal objective vectors. The vector z^* is one of them.

4.2 Bi-Objective Optimization for Performance and Energy on Homogeneous Multicore Clusters (*BOPPE*): Problem Formulation

Consider a data-parallel application workload of size n executed using p available identical processors where the speed function of a processor executing a problem size x is represented by $s(x)$ and the dynamic energy consumption of the execution of a problem size x by a processor is represented by $e(x)$. Then the bi-objective optimization problem for minimization of execution time (maximization of performance) and minimization of total dynamic energy of computations during the execution of the workload can be formulated as follows:

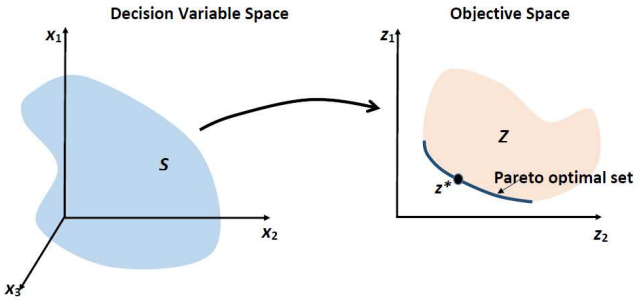


Fig. 6. An example showing the set $\mathcal{S}$ of decision variable vectors, the set $\mathcal{Z}$ of objective vectors, and Pareto-optimal objective vectors.

$$\begin{aligned} &BOPPE(n, p, s, e, q): \\ &\quad minimize \quad \{\max_{i=1}^{q} \frac{x_i}{s(x_i)}, \sum_{i=1}^{q} e(x_i)\} \\ &\quad \text{Subject to} \quad x_1 + x_2 + ... + x_q = n \\ &\qquad\qquad x_i \geq 0 \qquad i = 1, ..., q \\ &\qquad\qquad x_i \leq n \qquad i = 1, ..., q \\ &\qquad\qquad 1 \leq q \leq p \\ &\quad \text{where} \quad p, q, n, x_i \in \mathbb{Z}_{>0}, \\ &\qquad\qquad s(x), e(x) \in \mathbb{R}_{>0} \end{aligned}$$

The output of a solution method solving *BOPPE* is a set of Pareto-optimal solutions represented by workload distributions. It is important to note that the optimal number of processors (q) that are selected in a Pareto-optimal solution satisfies the constraint, $1 \leq q \leq p$.

In the supplemental, we present several classifications for methods solving bi-objective optimization problems. We show the formulations of *BOPPE* in some of the methods. We also discuss why the state-of-the-art solvers are inadequate for direct solution of *BOPPE*.

5 ALEPH: Efficient Algorithm for Solving *BOPPE*

In this section, we present an efficient algorithm solving *BOPPE* called *ALEPH*, which stands for ALgorithm for Bi-Objective Optimization for Energy and Performance on Homogeneous Multicore Clusters. The problem can be described as follows: Let p identical processors be used to execute the data-parallel application workload of size n. Let $s(x)$ be the speed of execution of the workload of size x by a processor. Let $e(x)$ be the dynamic energy consumption of execution of the workload of size x by a processor. Let Δx be the minimum granularity of workload so that each processor is allocated a multiple of Δx only. The speed function of a processor, $s(x)$, and the energy function of a processor,

$e(x)$, are represented by a discrete set of experimental points separated by Δx. The problem is to find the set of globally Pareto-optimal solutions for performance and energy.

The inputs to the algorithm are data-parallel application workload size n given as multiple of Δx, the number of processors p, the speed function represented by two discrete sets, $\mathcal{X}$ and $\mathcal{S}$ respectively containing problem sizes and speeds, the dynamic energy function represented by two discrete sets, $\mathcal{X}$ and $\mathcal{E}$ respectively containing problem sizes and dynamic energy consumptions, and the user preference, $\mathcal{U}$, which specifies the tolerance (in fractional values of optimal) for either performance or energy. The user preference parameter, $\mathcal{U}$, has two sub-parameters (Obj, δ). If $Obj = TIME$, then the output set of globally Pareto-optimal solutions have execution times that do not exceed $(1+\delta) \times t_{opt}$ where t_{opt} represents the optimal execution time. This signifies the case where $100 \times \delta$ percent performance degradation is acceptable. If $Obj = ENERGY$, then the output set of globally Pareto-optimal solutions have dynamic energy consumptions that do not exceed $(1+\delta) \times e_{opt}$ where e_{opt} represents the optimal dynamic energy consumption. This signifies the case where $100 \times \delta$ percent increase in energy consumption is acceptable.

The outputs from the algorithm are the set of globally Pareto-optimal solutions for performance and energy, $\mathcal{P}_{pareto}$, and the corresponding workload distributions, $\mathcal{D}_{pareto}$.

The core of *ALEPH* contains two invocations to the algorithm, *ALSOO* (Algorithm 2). The algorithm *ALSOO* is used to determine the optimal workload distribution solving single objective optimization problem of performance or energy.

Algorithm 1 Algorithm determining globally Pareto-optimal solutions for performance and energy of computations in the parallel execution of workload of size n.

procedure ALEPH($n, p, \Delta x, \mathcal{X}, \mathcal{S}, \mathcal{E}, \mathcal{U}$)
Input:
Data-parallel application workload size, $n \in \mathbb{Z}_{>0}$
Number of processors, $p \in \mathbb{Z}_{>0}$
$\mathcal{X} = \{x_1, ..., x_m\}, x_1 < ... < x_m, x_i \in \mathbb{Z}_{>0}, \forall i \in [1, m]$
Speed function represented by two sets $(\mathcal{X}, \mathcal{S})$,
$\mathcal{S} = \{s(x_1), ..., s(x_m)\}, s(x_i) \in \mathbb{R}_{>0}, \forall i \in [1, m]$
Energy function represented by two sets $(\mathcal{X}, \mathcal{E})$,
$\mathcal{E} = \{e(x_1), ..., e(x_m)\}, e(x_i) \in \mathbb{R}_{>0}, \forall i \in [1, m]$
User Preference, $\mathcal{U} = \{Obj, \delta\}, Obj \in \{TIME, ENERGY\}, \delta \in \mathbb{R}_{>0}$
Output:
Set of trade-off solutions optimizing performance and energy,
$\mathcal{D}_{pareto} = \{(x_1^i)_{i=1}^p, ..., (x_{|\mathcal{D}_{pareto}|}^i)_{i=1}^p)\}, x_i^j \in \mathbb{Z}_{>0}, \forall i \in [1, |\mathcal{D}_{pareto}|], \forall j \in [1, p]$
$\mathcal{P}_{pareto} = \{(t_1, e_1), ..., (t_{|\mathcal{P}_{pareto}|}, e_{|\mathcal{P}_{pareto}|})\}, t_i, e_i \in \mathbb{R}_{>0}$

  $\forall I \in [\frac{n}{p}, |\mathcal{X}|], J \in [1, p], K \in [1, p]$
    $memoized[I][J][K] \leftarrow (I, 0, 0)$
  $(\mathcal{D}_{pareto}, \mathcal{P}_{pareto}) \leftarrow$ ALSOO($TIME$,n,p,$|\mathcal{X}|$,$\mathcal{X}$,$\mathcal{S}$,$\mathcal{E}$,$memoized$,1)
  $(\mathcal{D}_{pareto}, \mathcal{P}_{pareto}) \leftarrow$ ALSOO($ENERGY$,n,p,$|\mathcal{X}|$,$\mathcal{X}$,$\mathcal{S}$,$\mathcal{E}$,$memoized$,1)
  if ($\mathcal{U} = \Phi$) **then**
    return $(\mathcal{D}_{pareto}, \mathcal{P}_{pareto})$
  end if
  return SELECTTRADEOFFS($\mathcal{U}, \mathcal{D}_{pareto}, \mathcal{P}_{pareto}$)
end procedure

Algorithm 2 Algorithm determining optimal distribution of workload of size n for maximizing performance or minimizing energy.

function ALSOO($ObjType, n, p, F,$ $\mathcal{X}, \mathcal{S}, \mathcal{E}, memoized, rLevel, \mathcal{D}_{pareto}, \mathcal{P}_{pareto}$)
  if ($p = 1$) **then return** $(\{n\}, \{\frac{n}{\mathcal{S}[n]}, \mathcal{E}[n]\})$ **end if**
  $d_{opt}^i \leftarrow \frac{n}{p}, \forall i \in [1, p]$
  $d_{opt}^i \leftarrow d_{opt}^i + 1, \forall i \in [1, n \bmod p]$
  if $ObjType = TIME$ **then**
    $f_{opt} \leftarrow \max_{1 \le i \le p} (\frac{d_{opt}^i}{S[d_{opt}^i]})$
  else
    $f_{opt} \leftarrow \sum_{i=1}^{p} \mathcal{E}[d_{opt}^i]$
  end if
  for $L \leftarrow memoized(\frac{n}{p}, p, n \bmod p, 1), F$ **do**
    for $r \leftarrow 1, p-1$ **do**
      $n_r \leftarrow \mathcal{X}[L]; n_l \leftarrow n - r \times n_r$
      if $n_l < 0$ **then break end if**
      if $ObjType = TIME$ **then**
        $f_r \leftarrow \frac{n_r}{\mathcal{S}[n_r]}$
      else
        $f_r \leftarrow r \times \mathcal{E}[n_r]$
      end if
      if $n_l = 0$ and $f_r < f_{opt}$ **then**
        $d_i \leftarrow n_r, \forall i \in [1, r]$
        $d_i \leftarrow 0, \forall i \in [r+1, p]$
        $f \leftarrow f_r$
        break
      end if
      if $memoized(\frac{n_l}{p-r}, p-r, n \bmod p, 1) > L$ **then**
        $f_l \leftarrow memoized(\frac{n_l}{p-r}, p-r, n \bmod p, 3)$
        if $ObjType = TIME$ **then**
          $f_{tmp} \leftarrow max(f_l, f_r)$
        else
          $f_{tmp} \leftarrow e_l + e_r$
        end if
        if $f_{tmp} < f_{opt}$ **then**
          $\forall i \in [r+1, p], x_i \leftarrow$
          $memoized(\frac{n_l}{p-r}, p-r, extra, 2)$
        end if
      else
        $((x_{r+1}, ..., x_p), f_{tmp}, \mathcal{D}_{pareto}, \mathcal{P}_{pareto}) \leftarrow$ ALSOO($ObjType$, n_l,$p-r$,L, $\mathcal{X}$,$\mathcal{S}$,$\mathcal{E}$,$memoized$,$rLevel$)
      end if
      if ($rLevel = 1$) **then**
        UPDATEPARETO($ObjType$, $((n_r)_{i=1}^r, (x_i)_{i=r+1}^p), f_{tmp}$, $\mathcal{D}_{pareto}$,$\mathcal{P}_{pareto}$)
      end if
      if $f_{tmp} < f_{opt}$ **then**
        $d_{opt}^i \leftarrow n_r, \forall i \in [1, r]$
        $d_{opt}^i \leftarrow x_i, \forall i \in [r+1, p]$
        $f_{opt} \leftarrow f_{tmp}$
      end if
    end for
  end for
  $memoized(\frac{n}{p}, p, n \bmod p) \leftarrow (L, d_{opt}, f_{opt})$
  return $(d, f, \mathcal{D}_{pareto}, \mathcal{P}_{pareto})$
end function

The *ALSOO* algorithms for performance and energy respectively are variants of the *POPTA* and *EOPTA* algorithms, which are presented in detail in [18]. Unlike *POPTA*, which examines a subset of points in the functional performance model $(\mathcal{X}, \mathcal{S})$, and *EOPTA*, which examines only the convex points in the energy model $(\mathcal{X}, \mathcal{E})$, *ALSOO* examines all the points in the performance and energy models, which is bounded by $O(m)$ where m is the cardinality of the sets $(\mathcal{X},\mathcal{S},\mathcal{E})$.

We present a summary of the operation of the *ALSOO* algorithm. It starts from a balanced workload distribution,

$x_i = \frac{n}{p}, \forall i \in [1, p]$. A key optimization in the algorithm is the 3D array, $memoized$, of size $O(m \times p^2)$, which memorizes the points that have already been visited during the recursive invocations of *ALSOO* in the invocation of *ALEPH*. Briefly speaking, for the execution of the problem size n using p processors, the array value $memoized(\frac{n}{p}, p, n \bmod p)$ contains the ending index of the range of points examined during the previous invocation, the optimal workload distribution, and the optimal value of the objective function. The array entry $memoized(\frac{n}{p}, p)$ is of size p where the index $n \bmod p$ represents a problem size $(\frac{n}{p} + n \bmod p)$ in the range $[\frac{n}{p}, \frac{n}{p} + p]$. This memorization ensures that there are only $O(m \times p^2)$ recursive invocations of the core kernel (Algorithm 2) to solve a problem size of n using p processors where m is the cardinality of the sets ($\mathcal{X}$,$\mathcal{S}$,$\mathcal{E}$).

Line 2 of the procedure, *ALSOO*, deals with the simple case of solving the problem size n using one processor. The execution time to solve the problem is $\frac{n}{\mathcal{S}[n]}$ and the energy consumption is $\mathcal{E}[n]$ using the functional performance models and energy models. Lines 3-9 initialize the optimal workload distribution, d_{opt}, and the optimal value of the objective function, f_{opt}. *ALSOO* always starts with evaluation of the workload distribution given by the traditional load-balancing algorithm, $x_i = \frac{n}{p}, \forall i \in [1, p]$.

Lines 10-49 contains the kernel of *ALSOO*. The points between $B = \frac{n}{p}$ and $F = |\mathcal{X}|$ are sequentially examined (Line 10). For each point A, there are $p - 1$ main execution steps in the nested for loop (Line 11). In a main step, each of the r processors is allocated the problem size n_r to the right of B. If the remaining problem size n_l is less than 0, that means there is excessive allocation to the right of B and so we break from the loop. This is because subsequent allocations to the right of B will always result in negative remaining problem size to the left of B to be solved using a recursive invocation of *ALSOO*. If the remaining problem size n_l is equal to 0, then we save this distribution if ($f_r < f_{opt}$) (Lines 14-24).

Now to solve the problem size n_l to the left of B using $p - r$ processors, we check if the range of points ($[\frac{n_l}{p-r}, L]$) have already been examined (Line 25). If yes, then they will not be re-examined due to the memorization and recursive invocation of *ALSOO* is avoided. The memorized optimal objective function value solving the problem size n_l to the left of B using $p - r$ processors is retrieved in f_l. If the combined objective function value (f_{tmp}) is less than f_{opt}, then we save the memorized workload distribution and avoid recursion (32-35).

For a main step, if the objective function value of the parallel execution (f_{tmp}) is lesser than the previously saved value, f_{opt}, then we save the improved solution (Lines 41-45). For each problem size n_l solved using $p - r$ processors, the ending index L, which contains the range of points already examined, is saved (Line 48). So, if an invocation for solving this problem size recurs, then recursion is avoided using the memorized arrays (Lines 32-35). Therefore, this memorization ensures that the total number of points (including those in the recursive invocations) for a point in the interval $[\frac{n}{p}, |\mathcal{X}|]$ is not more than $O(m \times p^2)$.

The globally Pareto-optimal set of solutions is updated using the call, $UpdatePareto$ (Lines 38-40). This call is invoked only when the recursion level is 1. This level essentially represents the different workload distributions for workload of size n using p possible combinations of number of processors.

We would like to mention that the algorithm *ALEPH* can be easily extended for the case where the user can specify input tolerances (δ_1, δ_2) for performance and energy respectively and it would determine the set, which has solutions where the execution times and dynamic energy consumptions do not exceed $(1+\delta_1) \times t_{opt}$ and $(1+\delta_2) \times e_{opt}$ respectively. This set will also contain the globally Pareto-optimal set of solutions.

The proof of *ALEPH* is given in Section 7 of the supplemental. The complexity of *ALEPH* is $O(m^2 \times p^2)$ and is presented in Section 8 of the supplemental.

5.1 Application of *ALEPH*

We summarize below how a user would apply *ALEPH*:

- The user takes a data-parallel application that has been optimized for *load-balanced* execution on a cluster of identical multicore nodes.
- This application is now treated as a black-box and executed on one node for a range of problem sizes to construct its speed and energy functions. When it is executed to obtain an experimental point of a function, the optimal values of the application tunables (such as the number of threads) that have already been determined are used. The goal here is not to perform multi-parameter optimization where the user finds the optimal values of these tunables. It is assumed that this optimization has already been done and the optimal values of the tunables have already been determined before this step. *So, essentially, we consider only one decision variable in our optimization problem, which is the problem size.*
- The speed and energy functions are then input to *ALEPH*, which determines the globally Pareto-optimal set of solutions for execution time and the energy consumption of computations in the parallel execution of the application.

6 Experimental Results and Discussion

In this section, we analyze the efficiency and scalability of *ALEPH* and study the interplay between performance and energy using two data-parallel applications on a Intel multicore CPU server containing 24 physical cores.

6.1 Applications and Platform

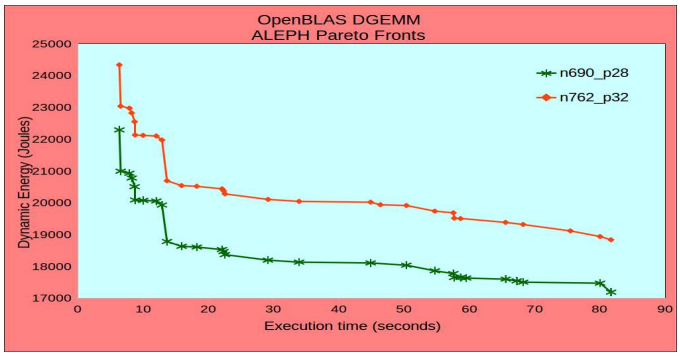
Our experimental platform is a Intel Haswell server containing 24 physical cores. Its specification is shown in Table 2. The two data-parallel applications are OpenBLAS DGEMM [19] and FFTW [20], [32], which are executed on the multicores composing the Intel Haswell server. The algorithms *ALEPH* and its building blocks, *POPTA* and *EOPTA*, are sequential algorithms, which are executed on just one core of the server. The experiments are a combination of actual measurements conducted on the server and simulations for clusters containing replicas of the server. We would

like to mention that the results from actual experiments on clusters would be no different from the results of our simulations. This is because *ALEPH* does not take into account the communication overheads. Since the problems sizes ($x_i, \forall i \in [1, p]$) in the workload distributions (Pareto-optimal solutions) are members of the set X in the input functions that have been built experimentally, the execution time and energy consumption from actual measurements can not differ from simulations.

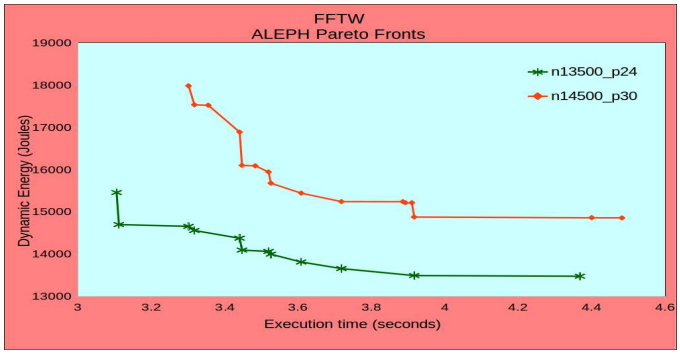
6.2 Speed and Energy Functions

The speed and the energy functions that are input to *ALEPH* are experimentally built only once and only for one node (in this case, the Intel Haswell server). To make sure the experimental results are reliable, we automated this build procedure and we follow an experimental methodology, which is described in detail in Section 3 in the supplemental. The inputs to the automation API function are the application and application parameters (problem size, number of threads, etc), range of problem sizes, and granularity. To obtain a data point for each function, the software executes the application repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Students t-test is used. In the end, the software returns a set of points, which represents the function. Figures 3a and 3b respectively show the speed and dynamic energy consumption functions of FFTW application. Figures 4a and 4b respectively show the speed and dynamic energy consumption functions of OpenBLAS DGEMM application. The total dynamic energy consumption during the application execution is obtained using Watts Up Pro power meter.

The cardinality of the discrete sets representing the speed and dynamic energy functions of OpenBLAS DGEMM application is 1440. The granularity (Δx) selected for the OpenBLAS DGEMM application is 1478656 (representing DGEMM of a 1216×1216 matrix). The cardinality of the discrete sets representing the speed and dynamic energy functions of FFTW application is 766. The granularity (Δx) selected for the FFTW application is 1327104 (representing 2D Discrete Fourier Transform of size 1152×1152). There is no specific reason why we picked the particular granularities for OpenBLAS DGEMM and FFTW. Lesser granularity would unveil larger fluctuations in the functional models but would also mean more experimental points thereby increasing the time to build the functional models as well as the execution times of *ALEPH*. As the granularity increases, the functional models become smooth and will resemble those for uniprocessors therefore disallowing any opportunity for optimization. The starting point in the speed and energy functions contains a problem size that is so selected as to exceed the L2 cache. The last point in the discrete set $\mathcal{X}$ (representing DGEMM of a 46080×46080 matrix) for the OpenBLAS DGEMM application contains a problem size, which occupies the whole main memory. For the FFTW application, we restrict the number of points because we observed that very large problem sizes were taking erroneously large execution times to complete possibly due to a software limitation. This was unduly lengthening the experimental building times of the functional models. The last point contains the problem size representing 2D DFT of size 32768×32768.



(a)



(b)

Fig. 7. Globally Pareto-optimal set of solutions with maximum sizes determined by *ALEPH* for OpenBLAS DGEMM and FFTW applications. Each curve shown as nX_pY represents results for data-parallel application workload size given by n (in multiples of granularity) and number of available processors, p.

6.3 Experimental Dataset

For all our experiments, we use a dataset constructed as follows:

1) Select all the problem sizes from the set $\mathcal{X}$.
2) For each problem size x from Step 1, set $\frac{n}{q}$ equal to x. Then for each $\frac{n}{q}$, q is iterated from 2 to 1024.
3) Fill the dataset with workloads of size, $n = \frac{n}{q} \times q$, solved using p processors in the range, $[q, 1024]$.

6.4 Experimental Analysis of *ALEPH*

In this section, we study the efficiency and scalability of *ALEPH* using four sets of experiments.

In the first set, we determine the minimum, average, and maximum sizes of the globally Pareto-optimal set of solutions determined by *ALEPH*. For the OpenBLAS DGEMM, these are 1, 15, and 34 and for the FFTW application, these are 1, 7, and 20. For workloads where the minimum size of globally Pareto-optimal set is 1, the solution minimizes both execution time and dynamic energy consumption. Figure 7 shows graphs for the combinations, (n, p), where the size of the globally Pareto-optimal set is maximum.

In the second set of experiments, we compare the improvements provided by the optimal solutions for performance and energy over those resulting from using the homogeneous workload distribution.

The performance improvement is calculated as follows: Performance Improvement (%) $= \frac{t_{homo}-t_{opt}}{t_{opt}} \times 100$, where t_{homo} is the execution time obtained using traditional homogeneous workload distribution ($x_i = \frac{n}{p}, \forall i \in [1,p]$) and t_{opt} is the optimal execution time. The percentage energy reduction is calculated as follows: Energy reduction (%) $= \frac{e_{homo}-e_{opt}}{e_{opt}} \times 100$, where e_{homo} is the dynamic energy consumption using traditional homogeneous workload distribution and e_{opt} is the optimal dynamic energy consumption. Negative values of these percentages represent performance degradation and increase in energy consumption. The average percentage improvement in performance and energy is calculated by averaging the percentages obtained from solving all the workloads in the dataset. The maximum percentage improvement is calculated by computing the maximum of all these percentages.

For the OpenBLAS DGEMM application, the average and maximum percentage improvements in performance are found to be 13% and 97% respectively. The average and maximum percentage reductions in energy are 18% and 71% respectively. For the FFTW application, the average and maximum performance percentage improvements are found to be 40% and 95% respectively. The average and maximum percentage reductions in energy are 22% and 127% respectively.

For the third and fourth sets of experiments, we determine the performance-energy trade-offs. In the third set, we determine how much performance can be gained by considered a 5% increase in energy consumption over the optimal ($U = \{ENERGY, .05\}$) to be acceptable. We then find the average and maximum improvements in performance. In the fourth set, we determine how much energy savings can be obtained by considering 5% performance degradation over the optimal ($U = \{TIME, .05\}$) to be acceptable. We then find the average and maximum improvements in energy consumption. It should be noted that this 5% represents how far we go from the time-optimal or energy-optimal endpoints along the Pareto-optimal front and should not be confused with the width of band of solutions determined by *ALEPH* (based on input tolerances for performance and energy). The performance improvement is calculated as follows: Performance Improvement (%) $= \frac{t_{e_{opt}\times 1.05}-t_{e_{opt}}}{t_{e_{opt}\times 1.05}} \times 100$, where $t_{e_{opt}}$ and $t_{e_{opt}\times 1.05}$ represent the execution time associated with energy-optimal endpoint and execution time associated with 5% increase in energy consumption over the optimal. The percentage energy reduction is calculated as follows: Energy reduction (%) $= \frac{e_{t_{opt}\times 1.05}-e_{t_{opt}}}{e_{t_{opt}\times 1.05}} \times 100$, where $e_{t_{opt}}$ and $e_{t_{opt}\times 1.05}$ represent the dynamic energy consumption associated with time-optimal endpoint and dynamic energy consumption associated with 5% performance degradation compared to the optimal. Therefore, these set of experiments also help determine the average and maximum slopes of the Pareto-optimal front close to the time-optimal and the energy-optimal solutions (or the endpoints of the front).

The average and maximum improvements in performance for the OpenBLAS DGEMM application (considering 5% increase in energy consumption) are 41% and 94% respectively. The average and maximum improvements in energy consumption (considering 5% performance degradation) are 9% and 44% respectively. The average and maximum improvements in performance for the FFTW application (considering 5% increase in energy consumption) are 19% and 73% respectively. The average and maximum improvements in energy consumption (considering 5% performance degradation) are 8% and 20% respectively. From the results, we can conclude that there exist decent trade-offs between performance and energy. One can also conclude that the Pareto-optimal front, on the average, has steeper slope closer to the time-optimal solution but starts flattening out immediately as we proceed towards the energy-optimal solution. This can be observed for the Pareto-optimal fronts shown in Figures 7 and 8.

For all the experimental runs, *ALEPH* demonstrated average quadratic polynomial complexity of $O(p^2)$. Its execution times ranged from few seconds to few minutes. Our conclusions are valid for homogeneous clusters containing arbitrary number of such processors suggesting large positive implications for extreme-scale parallel platforms.

6.5 Interplay Between Performance and Energy Using *ALEPH*, *POPTA*, and *EOPTA*

In this section, we study the interplay between performance and energy using *ALEPH* and its building blocks, *POPTA*, and *EOPTA*. *POPTA* solves the performance optimization problem by examining a subset of points in the functional performance model $(\mathcal{X}, \mathcal{S})$. It gives only one solution (workload distribution), which results in maximum performance. *EOPTA* solves the energy optimization problem by examining only the convex points in the energy model $(\mathcal{X}, \mathcal{E})$. It gives only one solution (workload distribution), which results in minimum dynamic energy consumption. *ALEPH*, however, examines all the points in the performance and energy models to determine the globally Pareto-optimal front of solutions.

First, we determine the improvements in performance obtained when using workload distribution minimizing dynamic energy consumption as well as the improvements in energy obtained when using workload distribution maximizing performance.

We observed that optimizing for performance alone can lead to good reduction in dynamic energy consumption. For OpenBLAS DGEMM, the average and maximum percentage reductions in energy were 12% and 68% respectively. For FFTW, the average and maximum percentage reductions in energy were 23% and 55% respectively.

We observed that optimizing for energy alone can cause major performance degradation. For OpenBLAS DGEMM, the average and maximum performance degradations were 95% and 100%. For FFTW, the average and maximum performance degradations were close to 100% and 100%.

Figure 8 illustrates the interplay for select combinations of (n, p). Each graph has the following plots: a). The globally Pareto-optimal set of solutions determined by *ALEPH* (points in the green plot). b). The path taken by *POPTA* towards the optimal solution for execution time (points in the blue plot), and c). The path taken by *EOPTA* towards the optimal solution for dynamic energy consumption (points in the red plot).

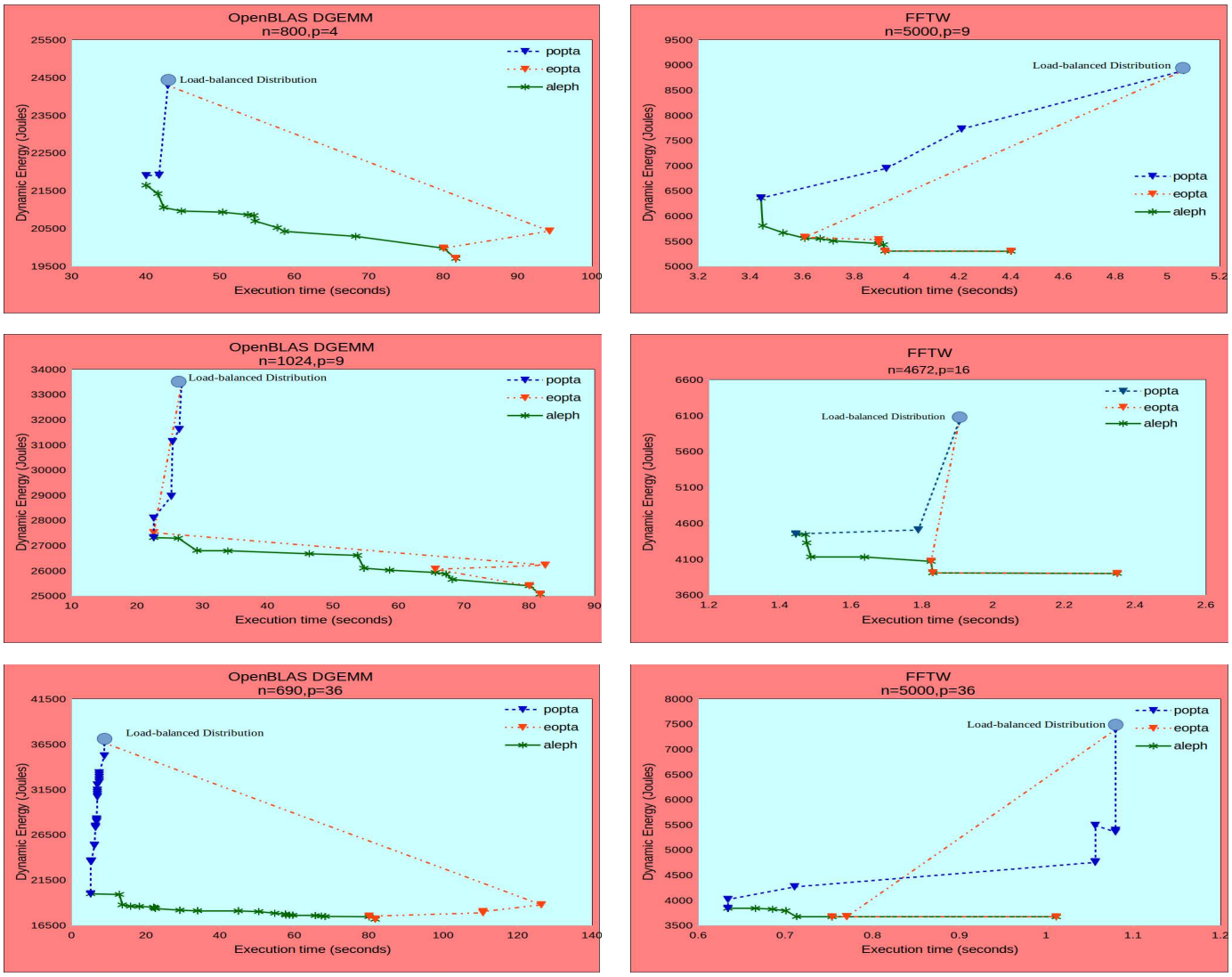


Fig. 8. Graphs for OpenBLAS DGEMM and FFTW applications for different workloads (size given by n in multiples of granularity) and number of available processors, p. Globally pareto-optimal set of solutions determined by *ALEPH* is shown by green points. Paths of *POPTA* and *EOPTA* are connected by blue and red points respectively. All the algorithms start from the homogeneous load-balanced distribution.

We define the path of an optimization algorithm as the sequence of points, which are ordered by non-increasing/non-decreasing values of the objective function if the function is minimized/maximized and which terminates in a point that represents the optimal value of the objective function. For example, the path of POPTA starts from the load-balanced distribution point and contains only the points with non-increasing values of execution time, with the last point representing the time-optimal solution. Similarly, the path of EOPTA starts from the load-balanced distribution point and contains only the points with non-increasing values of dynamic energy consumption, with the last point containing the energy-optimal solution. During the course of their execution, POPTA and EOPTA may veer off their paths (but will return to them soon) and evaluate points, which do not lead to decrease in execution time or dynamic energy consumption, but these points are not considered in their paths by definition.

Apart from better illustration of the interplay between performance and energy, the combinations, (n, p), are chosen to demonstrate the diversity in the paths taken by *POPTA* and *EOPTA*. The starting point for all the three algorithms is the balanced workload distribution, $x_i = \frac{n}{p}, \forall i \in [1, p]$. It should be noted that this distribution is the end point (solution) of the existing solvers. For a problem size n and number of processors p, each point in a plot represents the execution time and dynamic energy consumption of a workload distribution (or a partitioning of a workload).

We can conclude the following from the graphs: 1). The paths for *POPTA* show why optimizing for performance leads to reduction in energy consumption. For both the applications, it can be seen that the dynamic energy of the load-balanced distribution lies outside (and well above) the interval bounded by the projections of the endpoints of the Pareto-optimal front on the y-axis (dynamic energy axis). It can also be seen, that for OpenBLAS DGEMM, the execution time of the load-balanced distribution is very close to projection of the time-optimal endpoint than the

projection of the energy-optimal endpoint on the x-axis (execution time axis). Therefore, since *POPTA* descends from the load-balanced distribution towards the time-optimal endpoint of the Pareto-optimal front, minor improvements in performance result in major reduction in dynamic energy consumption.

2). The paths for *EOPTA* demonstrate why optimizing for energy can lead to significant performance degradation. For the OpenBLAS DGEMM application, it can be seen that the execution time of the load-balanced distribution is very close to projection of the time-optimal endpoint of the Pareto-optimal front on the x-axis than the energy-optimal endpoint. Therefore, since *EOPTA* moves from the load-balanced distribution towards the energy-optimal endpoint of the Pareto-optimal front, minor improvements in dynamic energy consumption result in major performance degradation. For the FFTW application, for almost all the experiments, the paths for *EOPTA* show the same behavior as observed for the OpenBLAS DGEMM application. However, we show two but rare examples ($(n = 5000, p = 9)$, $(n = 5000, p = 36)$) where the load-balanced distribution lies outside the intervals bounded by the projections of the endpoints of the Pareto-optimal front on the x-axis and the y-axis. Also, the execution time of the load-balanced distribution is very close to projection of the energy-optimal endpoint of the Pareto-optimal front on the x-axis than the time-optimal endpoint. In these cases, major improvements in dynamic energy consumption result in minor performance improvements.

3). For almost all the cases, the path for *POPTA* moves from the load-balanced distribution to touch the Pareto-optimal front at the time-optimal endpoint. However, there are few cases (first graph for OpenBLAS DGEMM where $n = 800, p = 4$) where *POPTA* terminates at a time-optimal solution, which is not the time-optimal endpoint on the Pareto-optimal front but just above it. That is, the time-optimal solution determined by *POPTA* has larger energy consumption (but same execution time) compared to the time-optimal solution determined by *ALEPH*.

4). Unlike *POPTA*, the path for *EOPTA* terminates in the energy-optimal point on the Pareto-optimal front for all the cases.

5). For a significant number of cases, the path for *EOPTA* moves from the load-balanced distribution to meet the Pareto-optimal front and then moves along it towards the energy-optimal solution. However, we have not observed this behavior for *POPTA*.

Our objective in presenting this study of interplay between performance and energy is to propose use of the globally Pareto-optimal set determined by *ALEPH* and the paths taken by *POPTA* and *EOPTA* as a guide to decide which objective (performance or energy) to optimize. The guidelines are as follows:

1). If the dynamic energy of the load-balanced distribution lies outside the interval bounded by the projections of the endpoints of the Pareto-optimal front on the y-axis, then optimizing for performance will also result in reduction in dynamic energy consumption. It should be noted that the endpoints of the Pareto-optimal front can be determined using *POPTA* and *EOPTA*. If the execution time of the load-balanced distribution is very close to projection of the time-optimal endpoint on the x-axis than the energy-optimal endpoint, then minor improvements in performance will result in major reduction in dynamic energy consumption.

2). If the load-balanced distribution lies outside both the intervals bounded by the projections of the endpoints of the Pareto-optimal front on the x-axis and y-axis, then optimizing for energy will also result in performance improvement. However, if the execution time of the load-balanced distribution is very close to projection of the energy-optimal endpoint on the x-axis than the time-optimal endpoint, then major improvements in dynamic energy consumption will only result in minor performance improvements. Similarly, if the dynamic energy of the load-balanced distribution is very close to projection of the time-optimal endpoint on the y-axis than the energy-optimal endpoint, then major improvements in execution time will only result in minor improvements in dynamic energy consumption.

3). If the load-balanced distribution lies inside both the intervals bounded by the projections of the endpoints of the Pareto-optimal front on the x-axis and y-axis, then minimizing one objective will definitely result in degradation of the other objective.

From the experiments, we observed that the practical time complexity of *ALEPH* is large compared to that of *POPTA* and *EOPTA*. This cost is few minutes for large values of p (thousands). This may not be acceptable, especially when it comes to using it for determining optimal workload distributions (with given accuracy) in self-adaptable applications. However, it should be noted that *ALEPH* and its building blocks are sequential algorithms, which execute on just one core of the Intel Haswell server. One approach to reduce its time complexity is to parallelize it. Along with developing parallel versions of *ALEPH* and its building blocks, we propose to pursue in our future work development of faster algorithms along the following lines of research.

1). If the goal is to maximize energy savings while ensuring performance degradation (with respect to optimal) is within specified tolerance, then there are two approaches to achieve it. If the starting point is close to the projection of the time-optimal endpoint on the x-axis than the energy-optimal endpoint, optimize for performance by using *POPTA*. From the time-optimal solution, start optimizing for energy by using *EOPTA* and stop once the tolerance is exceeded. If the starting point is close to the projection of the energy-optimal endpoint on the x-axis than the time-optimal endpoint, start optimizing for energy by using *EOPTA*. Once *EOPTA* meets the straight line joining the time-optimal and energy-optimal endpoints (determined by *POPTA* and *EOPTA*), start optimizing for performance by using *POPTA* until the tolerance is satisfied. Which approach to select depends on how fast either of the algorithms reach the Pareto-optimal front, which depends on the starting point and the slope of the front. One can develop similar algorithms if the goal is to maximize performance while ensuring energy consumption does not exceed specified fraction of the optimal energy consumption.

2). *ALEPH* determines the globally Pareto-optimal front of exact solutions. However, as we said before, its practical time complexity in the order of minutes for large values of n and p may not be acceptable for its use in self-adaptable applications, which require use of data parti-

tioning algorithms that can determine optimal workload distributions (with given accuracy) at low runtime cost. Therefore, one approach is to find approximate solutions quickly at runtime that are closer to the Pareto-optimal front (within certain distance specified by the user) by designing approximation algorithms, which intelligently combine *POPTA* and *EOPTA*. The initial slope of the Pareto-optimal front can be approximated by a straight line joining the time-optimal and energy-optimal solutions determined by *POPTA* and *EOPTA* respectively. From the experiments on trade-offs, we observe that the slope is steeper immediately closer to the time-optimal endpoint but is quite flat as we progress towards the energy-optimal endpoint. Therefore, we find an approximate solution closer to the midpoint of the front using the initial straight-line approximation for the front. Then, all the points lying on the line connecting this midpoint approximation to the energy-optimal endpoint will lie close to the globally Pareto-optimal front. We then recursively apply this procedure for the region between the time-optimal endpoint and the estimated midpoint approximation. To summarize, we would consider in our future work design and implementation of algorithms on these lines that determine approximate solutions (or front) that are ideal for use in self-adaptable applications.

6.6 *ALEPH* and DVFS-based MOP Methods

In this section, we demonstrate how *ALEPH* can be combined with MOP methods [8], [9], [10], [29], that use DVFS as one of the key decision variables. For this purpose, we use the same experimental platform, the Intel Haswell server containing 24 physical cores (48 logical cores) (Table 2), and the same two data-parallel applications (OpenBLAS DGEMM [19] and FFTW [20], [32]). Each core can be set to one of the twelve supported DVFS frequencies: {1.2 GHz, 1.4 GHz,..., 2.3 GHz}. This gives us 12^{48} frequency combinations. To keep the analysis tractable, we study only twelve combinations where we set all the cores to a same frequency from the supported DVFS set. Other details related to employing DVFS in our server are provided in Section 10, Supplemental.

We keep all the application-level parameters (*Workload size = n*, *Number of processors = p*, *Number of threads = t*) fixed while we study the impact of two decision variables, *DVFS* and *problem size*. MOP methods that employ *DVFS* as the primary decision variable fix the problem size parameter and use the workload distribution given by the traditional load-balancing algorithm, $x_i = \frac{n}{p}, \forall i \in [1, p]$. To determine a load-balanced solution for a frequency combination, the frequencies of all the cores are set and the execution time and dynamic energy consumption corresponding to the load-balanced workload distribution are determined.

Each Pareto-optimal point (load-balanced solution) determined by the *DVFS*-based MOP methods can be used as the starting point for *ALEPH*, which then finds a better set of (globally Pareto-optimal) solutions. To be more precise, for each Pareto-optimal load-balanced solution, the frequency combination is determined, the frequencies of all the cores are set, the speed and energy functions are constructed using the automated experimental methodology (Section 3, Supplemental), and *ALEPH* is executed to determine the better set of (globally Pareto-optimal) solutions.

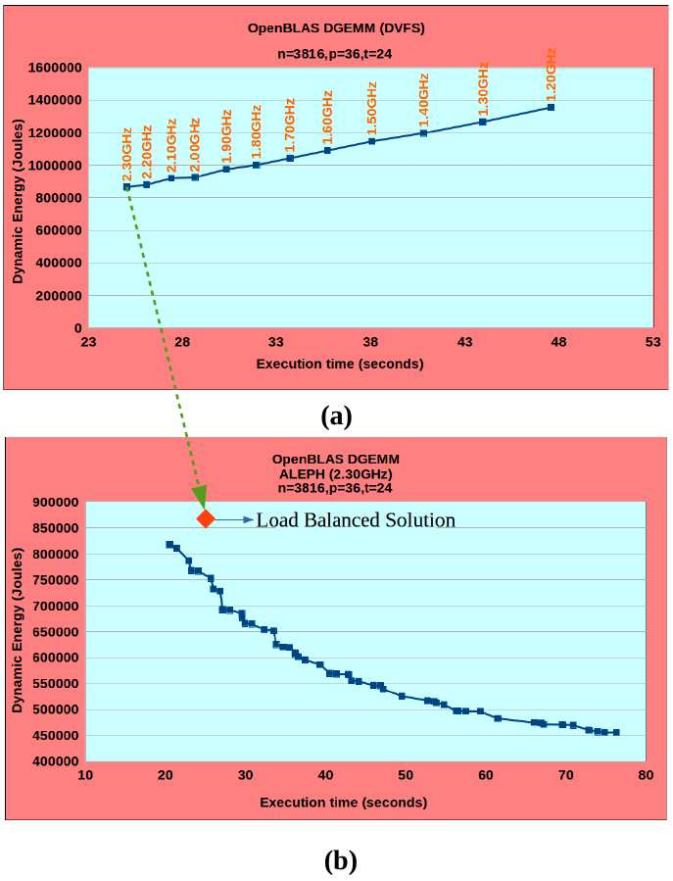


Fig. 9. (a). Load-balanced solutions for OpenBLAS DGEMM application determined by MOP methods using *DVFS* as the key decision variable. The data labels for the points are the DVFS frequencies. The optimal load-balanced solution maximizing performance and minimizing dynamic energy consumption corresponds to the frequency 2.30GHz. (b). Pareto-optimal front determined by *ALEPH* for the DVFS frequency 2.30GHz. The load-balanced solution is shown by the red rhombus symbol.

Figure 9(a) shows the load-balanced solutions determined by *DVFS*-based MOP methods for the OpenBLAS DGEMM application executed using the inputs, $(n = 3392, p = 36, t = 24)$. The optimal load-balanced solution maximizing performance and minimizing dynamic energy consumption corresponds to the frequency 2.30GHz. Using this solution as a starting point and for the corresponding frequency, *ALEPH* determines the Pareto-optimal front as shown in Figure 9(b). One can see that *ALEPH* gives a better set of solutions. There are altogether 53 solutions in the set. The solution with the best performance gives 25% and 6% improvements respectively in performance and dynamic energy compared to the load balanced solution.

Figure 10(a) shows the load-balanced solutions determined by *DVFS*-based MOP methods for the Intel MKL FFT application executed using the inputs, $(n = 3960, p = 36, t = 24)$. The optimal load-balanced solution maximizing performance and minimizing dynamic energy consumption corresponds to the frequency 2.30GHz. Using this solution as a starting point and for the corresponding frequency, *ALEPH* determines the Pareto-optimal front as shown in Figure 10(b). One can see that *ALEPH* gives a better set of solutions. There are 5 solutions in the set. The solution with

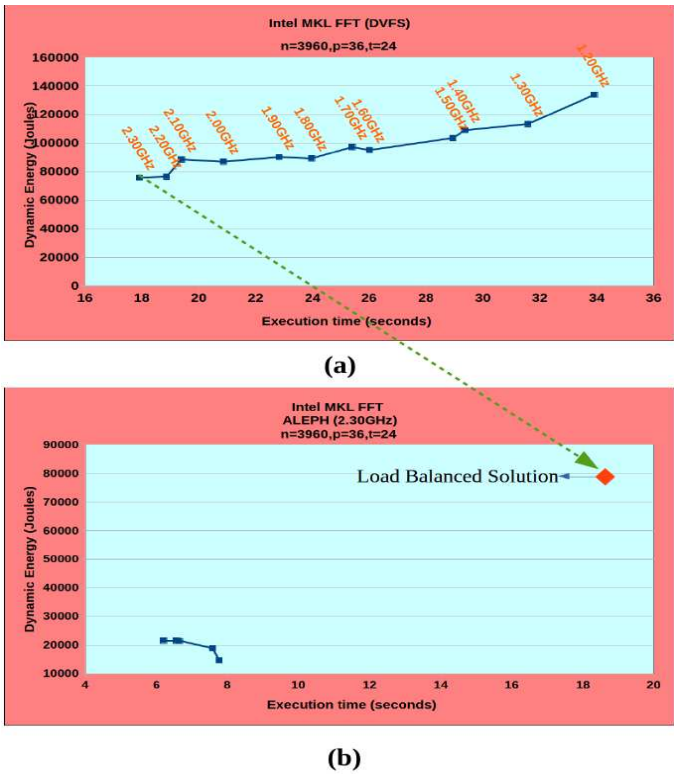


Fig. 10. (a). Load-balanced solutions for Intel MKL FFT application determined by MOP methods using *DVFS* as the key decision variable. The optimal load-balanced solution maximizing performance and minimizing dynamic energy consumption corresponds to the frequency 2.30GHz. (b). Pareto-optimal front determined by *ALEPH* for the DVFS frequency 2.30GHz. The load-balanced solution is shown by the red rhombus symbol.

the best performance gives 200% and 265% improvements respectively in performance and dynamic energy compared to the load balanced solution.

To conclude, we show how *ALEPH* can be combined with *DVFS*-based MOP methods to determine a better set of (globally Pareto-optimal) solutions.

Due to time constraints, we could not construct a full three-dimensional bi-objective optimization picture for the three dimensions (*Execution time*, *Dynamic energy*, *DVFS*) and also the combined Pareto-optimal front using both the decision variables, *DVFS* and *problem size*. We would consider it in our future work. We will also study in-depth the bi-objective optimization problem with more decision variables apart from *DVFS* and *problem size* such as *Number of processors* and *Number of threads*.

7 Conclusion

Performance and energy are now the most dominant objectives for optimization on modern parallel platforms composed of multicore CPU nodes. State-of-the-art application-level methods solving the bi-objective optimization problem for performance and energy (*BOPPE*) chiefly focused on either intra-node optimization or both intra-node and inter-node optimizations. These methods employed a large set of intra-node and inter-node decision variables, which include number of threads, number of processors, DVFS, etc. They analyzed the impact of each decision variable individually on both performance and energy. However, they did not consider problem size as a decision variable. When the problem size was considered as a parameter instead of a decision variable, they also assumed a linear relationship between performance and problem size and between energy and problem size. We demonstrated using experiments of real-life data-parallel applications on modern multicore CPUs that the relationships demonstrated between performance and problem size and between energy and problem size have complex (non-linear and even non-convex) properties. Therefore, we believe that problem size has become an important decision variable that can no longer be ignored.

Motivated by this important finding, we formulated the *BOPPE* for data-parallel applications on homogeneous clusters of modern multicore CPUs, which is based on only one but heretofore unconsidered decision variable, the problem size. We then presented an efficient and exact global optimization algorithm called *ALEPH* that solved the *BOPPE*. It takes as inputs, discrete functions of performance and dynamic energy consumption against problem size, and outputs the globally Pareto-optimal set of solutions. These solutions are the workload distributions, which achieve inter-node optimization of data-parallel applications for performance and energy. While existing solvers give a single solution for performance and energy when problem size and the number of processors are fixed, *ALEPH* gives a diverse set of globally Pareto-optimal solutions. We proved the complexity of the algorithm to be $O(m^2 \times p^2)$ where m is the number of points in the discrete speed/energy function and p is the number of available processors.

We studied the interplay between performance and energy of two data-parallel applications using the Pareto-optimal front (determined by *ALEPH*) and paths of optimization algorithms for performance and energy (*POPTA* and *EOPTA*). We showed that, for the two applications, optimizing for performance alone led to significant reduction in energy. However, optimizing for energy alone caused major degradation in performance. We also demonstrated the efficiency of *ALEPH*.

Based on our study of interplay, we proposed collective use of the Pareto-optimal front and paths of *POPTA* and *EOPTA* as optimization guide for design of optimization algorithms for performance and energy.

We demonstrated how *ALEPH* can be combined with *DVFS*-based MOP methods to give a better set of (globally Pareto-optimal) solutions.

The software implementations of the algorithms presented in this paper can be found at [33].

In our future work, we would find and study applications and platforms where optimization of energy would lead to improvements in performance. We would also focus on bi-objective optimization problems for performance and energy on heterogeneous parallel platforms.

Acknowledgments

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474.

REFERENCES

[1] M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E.-G. Talbi, A. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497 – 1508, 2011.

[2] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. IEEE Computer Society, 2012, pp. 300–309.

[3] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755 – 768, 2012, special Section: Energy efficiency in large-scale distributed systems.

[4] Y. Kessaci, N. Melab, and E.-G. Talbi, "A pareto-based metaheuristic for scheduling hpc applications on a geographically distributed cloud federation," *Cluster Computing*, vol. 16, no. 3, pp. 451–468, Sep. 2013.

[5] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, pp. 221 – 236, 2014.

[6] A. Lastovetsky and J. Twamley, "Towards a realistic performance model for networks of heterogeneous computers," in *High Performance Computational Science and Engineering*. Springer, 2005, pp. 39–57.

[7] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *International Journal of High Performance Computing Applications*, vol. 21, no. 1, pp. 76–90, 2007.

[8] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, Jun. 2007.

[9] I. Ahmad, S. Ranka, and S. U. Khan, "Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–6.

[10] B. Subramaniam and W. C. Feng, "Statistical power and performance modeling for optimizing the energy efficiency of scientific computing," in *2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010.

[11] S. Song, C. Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, "Iso-energy-efficiency: An approach to power-constrained parallel computation," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 128–139.

[12] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz, "Perfect strong scaling using no additional energy," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013.

[13] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 661–672.

[14] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, Apr. 2009.

[15] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, "Algorithmic time, energy, and power on candidate HPC compute building blocks," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 447–457.

[16] M. Drozdowski, J. M. Marszalkowski, and J. Marszalkowski, "Energy trade-offs analysis using equal-energy maps," *Future Generation Computer Systems*, vol. 36, pp. 311–321, 2014.

[17] J. M. Marszalkowski, M. Drozdowski, and J. Marszalkowski, "Time and energy performance of parallel systems with hierarchical memory," *Journal of Grid Computing*, vol. 14, no. 1, pp. 153–170, 2016.

[18] A. Lastovetsky and R. Reddy, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, April 2017.

[19] OpenBLAS, "OpenBLAS: An optimized BLAS library," 2016. [Online]. Available: http://www.openblas.net/

[20] FFTW, "FFTW: A fast, free c FFT library," 2016. [Online]. Available: http://www.fftw.org/

[21] A. L. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of MPDATA on Intel Xeon Phi through load imbalancing," *CoRR*, vol. abs/1507.01265, 2015. [Online]. Available: http://arxiv.org/abs/1507.01265

[22] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing," *IEEE Transactions on Parallel and Distributed Systems*, 08/2016.

[23] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 104.

[24] A. L. Lastovetsky and R. Reddy, "Data distribution for dense factorization on computers with memory heterogeneity," *Parallel Computing*, vol. 33, no. 12, Dec. 2007.

[25] D. Clarke, A. Lastovetsky, and V. Rychkov, "Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models," in *Euro-Par 2011: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, vol. 7155. Springer-Verlag, 2012.

[26] P. K. Smolarkiewicz and W. W. Grabowski, "The multidimensional positive definite advection transport algorithm: Nonoscillatory option," *J. Comput. Phys.*, vol. 86, no. 2, Feb. 1990.

[27] S. Singer and J. Nelder, "Nelder-Mead algorithm," *Scholarpedia*, vol. 4, no. 7, p. 2928, 2009.

[28] J. Kołodziej, S. U. Khan, L. Wang, and A. Y. Zomaya, "Energy efficient genetic-based schedulers in computational grids," *Concurr. Comput. : Pract. Exper.*, vol. 27, no. 4, pp. 809–829, Mar. 2015.

[29] P. Balaprakash, A. Tiwari, and S. M. Wild, *Multi Objective Optimization of HPC Kernels for Performance, Power, and Energy*. Springer International Publishing, 2014, pp. 239–260.

[30] K. Miettinen, *Nonlinear multiobjective optimization*. Kluwer, 1999.

[31] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[32] PFFTW, "Parallel FFTW," 2016. [Online]. Available: http://www.fftw.org/fftw2_doc/fftw_4.html

[33] R. Reddy and A. Lastovetsky, "ALEPH: ALgorithms for performance and energy optimization of data-parallel applications on homogeneous multicore clusters," 2017. [Online]. Available: https://git.ucd.ie/manumachu/aleph

Ravi Reddy Manumachu received a B.Tech degree from I.I.T, Madras in 1997 and a PhD degree from the School of Computer Science, University College Dublin in 2005. His main research interests include high performance heterogeneous computing, energy efficient computing, high performance data analytics, and distributed computing.

Alexey Lastovetsky received a PhD degree from the Moscow Aviation Institute in 1986, and a Doctor of Science degree from the Russian Academy of Sciences in 1997. His main research interests include algorithms, models, and programming tools for high performance heterogeneous computing. He has published over a hundred technical papers in refereed journals, edited books, and international conferences. He authored the monographs Parallel computing on heterogeneous networks (Wiley, 2003) and High performance heterogeneous computing (with J. Dongarra, Wiley, 2009).