A Novel Algorithm for Bi-objective Performance-Energy Optimization of Applications with Continuous Performance and Linear Energy Profiles on Heterogeneous HPC Platforms

Hamidreza Khaleghzadeh[1], Ravi Reddy Manumachu[2(✉)], and Alexey Lastovetsky[2]

[1] School of Computing, University of Portsmouth, Portsmouth, UK
hamidreza.khaleghzadeh@port.ac.uk
[2] School of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland
{ravi.manumachu,alexey.lastovetsky}@ucd.ie

Abstract. Performance and energy are the two most important objectives for optimization on heterogeneous HPC platforms. This work studies a mathematical problem motivated by the bi-objective optimization of a matrix multiplication application on such platforms for performance and energy. We formulate the problem and propose an algorithm of polynomial complexity solving the problem where all the application profiles of objective type one are continuous and strictly increasing, and all the application profiles of objective type two are linear increasing. We solve the problem for the matrix multiplication application employing five heterogeneous processors that include two Intel multicore CPUs, an Nvidia K40c GPU, an Nvidia P100 PCIe GPU, and an Intel Xeon Phi. Based on our experiments, a dynamic energy saving of 17% is gained while tolerating a performance degradation of 5% (a saving of 106 J for an execution time increase of 0.05 s).

Keywords: Bi-objective optimization · Min-max optimization · Min-sum optimization · Performance optimization · Energy optimization

1 Introduction

Performance and energy are the two most important objectives for optimization on modern parallel platforms such as supercomputers, heterogeneous HPC clusters, and cloud infrastructures [3,5,7,18]. State-of-the-art solutions for the bi-objective optimization problem for performance and energy on such platforms can be broadly classified into *system-level* and *application-level* categories.

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

R. Chaves et al. (Eds.): Euro-Par 2021, LNCS 13098, pp. 166–178, 2022.
https://doi.org/10.1007/978-3-031-06156-1_14

System-level solution methods aim to optimize the performance and energy of the environment where the applications are executed. The methods employ application-agnostic models and hardware parameters as decision variables. The dominant decision variable in this category is Dynamic Voltage and Frequency Scaling (DVFS) [3,6,7,10,19,22].

The application-level solution methods proposed in [2,8,14,15] use application-level parameters as decision variables that include the number of threads, number of processors, loop tile size, and workload distribution. The solution methods proposed in [14,15] solve the bi-objective optimization problem of an application for performance and energy on homogeneous clusters of modern multicore CPUs. The solution method [2] considers the effect of heterogeneous workload distribution on bi-objective optimization of data analytics applications by simulating heterogeneity on homogeneous clusters.

Khaleghzadeh et al. [8] discover that moving from the single-objective optimization for performance or energy to the bi-objective optimization for performance and energy on heterogeneous processors results in a drastic increase in the number of optimal solutions in the case of linear performance and energy profiles, with practically all the solutions load imbalanced. They prove that for two processors with *linear* execution time and energy functions, the Pareto front is linear and contains an infinite number of solutions, out of which one solution is load balanced while the rest are load imbalanced. They then propose an algorithm that solves the bi-objective optimization problem for *discrete* execution time and dynamic energy functions with any arbitrary shape and returns the Pareto front of load imbalanced solutions and best load balanced solutions.

This work introduces a mathematical problem motivated by the bi-objective optimization of a matrix multiplication application on heterogeneous HPC platforms for performance and energy.

Consider the bi-objective optimization of a highly optimized matrix multiplication application on a heterogeneous computing platform for performance and energy. The application computes the matrix product, $C = \alpha \times A \times B + \beta \times C$, where A, B, and C are matrices of size $M \times N$, $N \times N$, and $M \times N$, and α and β are constant floating-point numbers. The application uses Intel MKL DGEMM for CPUs and Intel Xeon Phi and CUBLAS for Nvidia GPUs. The Intel MKL and CUDA versions used are 2017.0.2 and 9.2.148. Workload sizes range from 64×10112 to 19904×10112 with a step size of 64 for the first dimension m.

The platform consists of five processors: Intel Haswell E5-2670V3 multi-core CPU (CPU_1), Intel Xeon Gold 6152 multi-core CPU (CPU_2), NVIDIA K40c GPU (GPU_1), NVIDIA P100 PCIe GPU (GPU_2), and Intel Xeon Phi 3120P (XeonPhi_1).

Figure 1 shows the execution time functions $\{f_0(x), \ldots, f_4(x)\}$ and the dynamic energy functions $\{g_0(x), \ldots, g_4(x)\}$ of the processors against the workload size (x). Briefly, the total energy consumption during an application execution is the sum of dynamic and static energy consumptions. The static energy consumption is the idle power of the platform (without application execution) multiplied by the application's execution time. The dynamic energy consumption is the total energy consumed by the platform during the application execution

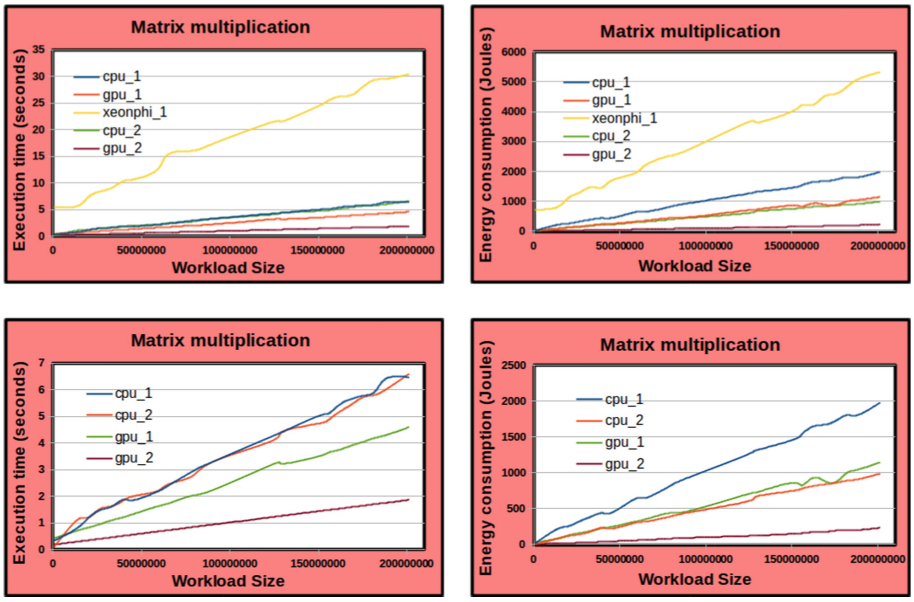

Fig. 1. The top two plots contain the execution time and energy profiles of the five heterogeneous processors employed in the matrix multiplication application. The bottom two plots do not contain the profiles for Xeon Phi. While the execution time profiles of the two CPUs are close to each other, the energy profile of CPU_1 is significantly higher than that of the CPU_2.

minus the static energy consumption. The dynamic energy consumption during an application execution is obtained using power meters, which is considered the most accurate method of energy measurement [9].

The execution time function shapes are continuous and strictly increasing. The energy function shapes can be approximated accurately by linear increasing functions. The execution time profiles of the two CPUs are close to each other but the energy profile of CPU_1 is significantly higher than that of the CPU_2. The optimization goal is to find workload distributions of the workload size n $(\{x_0, \ldots, x_4\}, \sum_{i=0}^{4} x_i = n)$ minimizing the execution time $(\max_{i=0}^{4} f_i(x_i))$ and the total dynamic energy consumption $(\sum_{i=0}^{4} g_i(x_i))$ during the parallel execution of the application. We solve the optimization problem for such shapes of performance and dynamic energy functions in this work.

We first formulate the mathematical problem, which for a given positive real number n aims to find a vector $X = \{x_0, \cdots, x_{k-1}\} \in \mathbb{R}^k_{\geq 0}$ such that $\sum_{i=0}^{k-1} x_i = n$, minimizing the max of k-dimensional vector of functions of objective type one and the sum of k-dimensional vector of functions of objective type two. We then propose an algorithm solving the case where all the functions of objective type one are continuous and strictly increasing, and all the functions of objective type two are linear increasing. The algorithm exhibits polynomial complexity.

We employ the algorithm to solve the problem for the matrix multiplication application using the five heterogeneous processors. Based on our experiments,

the maximum dynamic energy savings can be up to 17% while tolerating a performance degradation of 5% (an energy saving of 106 J for an execution time increase of 0.05 s).

The main original contributions of this work are:

- Mathematical formulation of the bi-objective optimization problem which for a given positive real number n aims to find a vector, $X = \{x_0, \cdots, x_{k-1}\} \in \mathbb{R}^k_{\geq 0}$, such that $\sum_{i=0}^{k-1} x_i = n$, minimizing the maximum of k functions of objective type one and the sum of k functions of objective type two.
- An exact algorithm of polynomial complexity solving the bi-objective optimization problem when all the functions of objective type one are continuous and strictly increasing, and all the functions of objective type two are linear increasing.

The rest of the paper is organized as follows. We discuss the related work in Sect. 2. The formulation of the bi-objective optimization problem is presented in Sect. 3. In Sect. 4, we propose our algorithm solving the bi-objective optimization problem. Section 5 contains the experimental results. Finally, we conclude the paper in Sect. 6.

2 Related Work

A bi-objective optimization problem can be mathematically formulated as [16,20]:

$$minimize \quad \{T(x), E(x)\}, \quad \text{Subject to} \quad x \in \mathcal{S}$$

where there are two objective functions, $T : \mathbb{R}^k \to \mathbb{R}$ and $E : \mathbb{R}^k \to \mathbb{R}$. We denote the vector of objective functions by $\mathcal{F}(x) = (T(x), E(x))^T$. The decision vectors $x = (x_1, ..., x_k)^T$ belong to the (non-empty) feasible region (set) $\mathcal{S}$, which is a subset of the decision variable space $\mathbb{R}^k$. We denote the image of the feasible region by $\mathcal{Z}$ $(=\mathcal{F}(\mathcal{S}))$, and call it a feasible objective region. It is a subset of the objective space $\mathbb{R}^2$. The elements of $\mathcal{Z}$ are called objective (function) vectors or criterion vectors and denoted by $\mathcal{F}(x)$ or $z = (z_1, z_2)^T$, where $z_1 = T(x)$ and $z_2 = E(x)$ are objective (function) values or criterion values.

The objective is to minimize both the objective functions simultaneously. The objective functions are at least partly conflicting or incommensurable, due to which it is impossible to find a single solution that would be optimal for all the objectives simultaneously. Furthermore, there is no natural ordering in the objective space because it is only partially ordered. Therefore, the concept of optimality is handled differently from a 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 $T(x) \leq T(x^*), E(x) \leq E(x^*)$ and either $T(x) < T(x^*)$ or $E(x) < E(x^*)$ or both* [16].

An objective vector $z^* \in \mathcal{Z}$ is Pareto optimal if there is not another objective vector $z \in \mathcal{Z}$ such that $z_1 \leq z_1^*, z_2 \leq z_2^*$ and $z_j < z_j^*$ for at least one index j.

There are several classifications for methods solving bi-objective optimization problems [16,20]. Since the set of Pareto optimal solutions is partially ordered, one classification is based on the involvement of the decision-maker in the solution method to select specific solutions. There are four categories in this classification, *No preference*, *A priori*, *A posteriori*, *Interactive*. The algorithms solving bi-objective optimization problems can be divided into two major categories, *exact methods* and *metaheuristics*. While branch-and-bound (B&B) is the dominant technique in the first category, genetic algorithm (GA) is popular in the second category.

Bi-objective Optimization on High Performance Computing Platforms. There are two principal categories of methods for optimizing applications on high performance computing (HPC) platforms for performance and energy. The first category of system-level solution methods aims to optimize the performance and energy of the executing environment of the applications. The dominant decision variable in this category is Dynamic Voltage and Frequency Scaling (DVFS). DVFS reduces the dynamic power consumed by a processor by throttling its clock frequency. The methods proposed in [6,19,22] optimize for performance under a energy budget or optimize for energy under an execution time constraint. The methods proposed in [3,7,10] solve bi-objective optimization for performance and energy with no time constraint or energy budget.

The second category of application-level solution methods [2,8,11,14,15,17] use application-level decision variables and models. The most popular decision variables include the loop tile size, workload distribution, number of processors, and number of threads.

Reddy et al. [15,17] study bi-objective optimization of data-parallel applications for performance and energy on homogeneous clusters multicore CPUs employing only one decision variable, the workload distribution. They propose an efficient solution method. The method accepts as input the number of available processors, the discrete function of the processor's energy consumption against the workload size, the discrete function of the processor's performance against the workload size. It outputs a Pareto-optimal set of workload distributions. Khaleghzadeh et al. [8] propose exact solution methods solving bi-objective optimization problem for hybrid data-parallel applications on heterogeneous computing platforms for performance and energy.

Tarplee et al. [21] consider optimizing two conflicting objectives, the make-span and total energy consumption of all nodes in a HPC platform. They employ linear programming and divisible load theory to compute tight lower bounds on the make-span and energy of all tasks on a given platform. Using this formulation, they then generate a set of Pareto front solutions. The decision variable is task mapping. Aba et al. [1] present an approximation algorithm to minimize both make-span and the total energy consumption in parallel applications running on a heterogeneous resources system. The decision variable is task scheduling. Their algorithm ignores all solutions where energy consumption exceeds a given constraint and returns the solution with minimum execution time.

3 Formulation of the Bi-objective Optimization Problem

Given a positive real number $n \in \mathbb{R}_{>0}$ and two sets of k functions each, $F = \{f_0, f_1, \cdots, f_{k-1}\}$ and $G = \{g_0, g_1, \cdots, g_{k-1}\}$, where $f_i, g_i \colon \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}, i \in \{0, \cdots, k-1\}$, the problem is to find a vector $X = \{x_0, \cdots, x_{k-1}\} \in \mathbb{R}^k_{\geq 0}$ such that $\sum_{i=0}^{k-1} x_i = n$, minimizing the objective functions $T(X) = \max_{i=0}^{k-1} f_i(x_i)$ and $E(X) = \sum_{i=0}^{k-1} g_i(x_i)$. We use $T \times E$ to denote the objective space of this problem, $\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$.

Thus, the problem can be formulated as follows:

BOPGVEC(n, k, F, G):

$$\begin{aligned} & T(X) = \max_{i=0}^{k-1} f_i(x_i), E(X) = \sum_{i=0}^{k-1} g_i(x_i) \\ \underset{X}{\text{minimize}} \quad & \{T(X), E(X)\} \\ \text{s.t.} \quad & x_0 + x_1 + \cdots + x_{k-1} = n \end{aligned} \tag{1}$$

We aim to solve BOPGVEC by finding both the Pareto front containing the optimal objective vectors in the objective space $T \times E$ and the decision vector for a point in the Pareto front. Thus, our solution finds a set of triplets $\Psi = \{(T(X), E(X), X)\}$ such that X is a Pareto-optimal decision vector, and the projection of Ψ onto the objective space $T \times E$, $\Psi \downarrow_{T \times E}$, is the Pareto front.

4 Bi-objective Optimization Problem for Max of Continuous Functions and Sum of Linear Functions

In this section, we solve BOPGVEC for the case where all functions in the set F are continuous and strictly increasing, and all functions in the set G are linear increasing, that is, $G = \{g_0, \cdots, g_{k-1}\}, g_i(x) = b_i \times x, b_i \in \mathbb{R}_{>0}, i = 0, \ldots, k-1$. Without loss of generality, we assume that the functions in G are sorted in the decreasing order of coefficients, $b_0 \geq b_1 \geq \cdots \geq b_{k-1}$.

Our solution consists of two algorithms, Algorithm 1 and Algorithm 2. The first one, which we call LBOPA, constructs the Pareto front of the optimal solutions in the objective space $\Psi \downarrow_{T \times E}$. The second algorithm finds the decision vector for a given point in the Pareto front.

The inputs to LBOPA (see Algorithm 1 for pseudo-code) are two sets of k functions each, F and G, and an input value, $n \in \mathbb{R}_{>0}$. LBOPA constructs a Pareto front, consisting of $k-1$ segments $\{s_0, s_1, \cdots, s_{k-2}\}$. Each segment s_i has two endpoints, (t_i, e_i) and (t_{i+1}, e_{i+1}), which are connected by curve $P_f(t) = b_i \times n - \sum_{j=i+1}^{k-1} (b_i - b_j) \times f_j^{-1}(t)$ $(0 \leq i \leq k-2)$. Figure 2 illustrates the functions in the sets, F and G, when all functions in F are linear, $f_i(x) = a_i \times x$. In this particular case, the Pareto front returned by LBOPA will be piece-wise linear, $P_f(t) = b_i \times n - t \times \sum_{j=i+1}^{k-1} \frac{b_i - b_j}{a_j}$ $(0 \leq i \leq k-2)$, as shown in Fig. 2.

The main loop of the Algorithm 1 computes k points (Lines 3–7). In an iteration i, the minimum value of objective T, t_i, is obtained using the algorithm, solving the single-objective min-max optimization problem, $\min_X \{\max_{j=i}^{k-1} f_j(x_j)\}$.

Algorithm 1. Algorithm constructing the Pareto front of the optimal solutions.

1: **function** LBOPA(n, k, F, G)
2: $S \leftarrow \varnothing$
3: **for** $i \leftarrow 0, k-1$ **do**
4: $t_i \leftarrow \min_X \{ \max_{j=i}^{k-1} f_j(x_j) \}$
5: $e_i \leftarrow b_i \times n - \sum_{j=i+1}^{k-1}(b_i - b_j) \times f_j^{-1}(t_i)$
6: $S \leftarrow S \cup (t_i, e_i)$
7: **end for**
8: **for** $i \leftarrow 0, k-2$ **do**
9: Connect (t_i, e_i) and (t_{i+1}, e_{i+1}) by curve $b_i \times n - \sum_{j=i+1}^{k-1}(b_i - b_j) \times f_j^{-1}(t)$
10: **end for**
11: **end function**

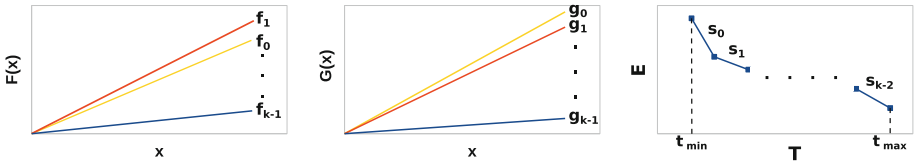

Fig. 2. Sets F and G of k linear increasing functions each. Functions in G are arranged in the decreasing order of slopes. LBOPA returns a linear piece-wise Pareto front shown in the bottom plot comprising a chain of $k-1$ linear segments.

We do not present the details of this algorithm. Depending on the shapes of functions, $\{f_0, \ldots, f_{k-1}\}$, one of the existing polynomial algorithms solving this problem can be employed [12,13].

The end point $(t_{min}, e_{max}) = (t_0, e_0)$ represents decision vectors with the minimum value of objective T and the maximum value of objective E, while the end point $(t_{max}, e_{min}) = (t_{k-1}, e_{k-1})$ represents decision vectors with the maximum value of objective T and the minimum value of objective E (as illustrated for the case of all linear increasing functions in Fig. 2).

Given an input $t \in [t_0, t_{k-1}]$, Algorithm 2 finds a decision vector $X = \{x_0, x_1, \cdots, x_{k-1}\}$ such that $\sum_{i=0}^{k-1} x_i = n$, $\max_{i=0}^{k-1} f_i(x_i) = t$, and $\sum_{i=0}^{k-1} g_i(x_i)$ is minimal. The algorithm first initialises X with $\{x_0, x_1, \cdots, x_{k-1} \mid x_i = f_i^{-1}(t)\}$ (Line 2) so that $f_i(x_i) = t$ for all $i \in [0, k-1]$. For this initial X the condition $\max_{i=0}^{k-1} f_i(x_i) = t$ is already satisfied but $\sum_{i=0}^{k-1} x_i$ may be either equal to n or greater than n. If $\sum_{i=0}^{k-1} x_i = n$, then this initial X will be the only decision vector such that $\sum_{i=0}^{k-1} x_i = n$ and $\max_{i=0}^{k-1} f_i(x_i) = t$ and hence the unique (Pareto-optimal) solution. Otherwise, $\sum_{i=0}^{k-1} x_i = n + n_{plus}$ where $n_{plus} > 0$. In that case, this initial vector$X$ will maximize both $\sum_{i=0}^{k-1} x_i$ and $\sum_{i=0}^{k-1} g_i(x_i)$ in the set $\mathcal{X}_t$ of all vectors in the decision space satisfying the condition $\max_{i=0}^{k-1} f_i(x_i) = t$. The algorithm then iteratively reduces elements of vector X until their sum becomes equal to n. Obviously, each such reduction will also reduce $\sum_{i=0}^{k-1} g_i(x_i)$. To achieve the maximum reduction of $\sum_{i=0}^{k-1} g_i(x_i)$, the algorithm starts from vector element x_i, the reduction of which by an arbitrary amount Δx will result in the maximum reduction of $\sum_{i=0}^{k-1} g_i(x_i)$. In our case, it will be x_0 as the functions in G are sorted in the decreasing order of coefficients b_i. Thus, at the first reduction step, the algorithm will try to reduce x_0 by n_{plus}. If $x_0 \geq n_{plus}$, it will succeed

and find a Pareto-optimal decision vector $X = \{x_0 - n_{plus}, x_1, \cdots, x_{k-1}\}$. If $x_0 < n_{plus}$, it will reduce n_{plus} by x_0, set $x_0 = 0$ and move to the second step. At the second step, it will try to reduce x_1 by the reduced n_{plus}, and so on. This way the algorithm minimizes $\sum_{i=0}^{k-1} g_i(x_i)$, preserving $\max_{i=0}^{k-1} f_i(x_i) = t$ and achieving $\sum_{i=0}^{k-1} x_i = n$.

Algorithm 2 . Algorithm finding a Pareto-optimal decision vector $X = \{x_0, x_1, \cdots, x_{k-1}\}$ for the problem $BOPGVEC(n, k, F, G)$.

```
1:  function PARTITION(n, k, F, G, t)
2:      X = {x_0, ··· , x_{k-1} | x_i ← f_i^{-1}(t)}
3:      n_plus ← Σ_{i=0}^{k-1} x_i − n
4:      if n_plus < 0 then
5:          return (0, 0, ∅)
6:      end if
7:      i ← 0
8:      while (n_plus > 0) ∧ (i < k − 1) do
9:          if x_i ≥ n_plus then
10:             x_i ← x_i − n_plus
11:             n_plus ← 0
12:         else
13:             n_plus ← n_plus − x_i
14:             x_i ← 0
15:             i ← i + 1
16:         end if
17:     end while
18:     if n_plus > 0 then
19:         return (0, 0, ∅)
20:     end if
21:     e ← Σ_{i=0}^{k-1} b_i × x_i
22:     return (t, e, X)
23: end function
```

The correctness of these algorithms is proved in Theorem 1.

Theorem 1. *Consider bi-objective optimization problem $BOPGVEC(n, k, F, G)$ where all functions in F are continuous and strictly increasing and $G = \{g_i(x) \mid g_i(x) = b_i \times x, b_i \in \mathbb{R}_{>0}, i \in \{0, \cdots, k-1\}\}$. Then, the piecewise function S, returned by* LBOPA*(n,k,F,G) (Algorithm 1) and consisting of $k-1$ segments, is the Pareto front of this problem, $\Psi \downarrow_{T \times E}$, and for any $(t, e) \in \Psi \downarrow_{T \times E}$, Algorithm 2 returns a Pareto-optimal decision vector X such that $T(X) = t$ and $E(X) = e$.*

Proof. First, consider Algorithm 2 and arbitrary input parameters $n > 0$ and $t > 0$. If after initialization of X (Line 2) we will have $\sum_{i=0}^{k-1} x_i < n$, it means that t is too small for the given n, and for any vector $Y = \{y_0, y_1, \cdots, y_{k-1}\}$ such that $\sum_{i=0}^{k-1} y_i = n$, $\max_{i=0}^{k-1} f_i(y_i) > t$. In this case, there is no solution to the optimization problem, and the algorithm terminates abnormally.

Otherwise, the algorithm enters the *while* loop (Line 8). If $i < k - 1$ upon exit from this loop, then the elements of vector X will be calculated as

$$x_j = \begin{cases} 0 & j < i \\ n - \sum_{m=j+1}^{k-1} f_m^{-1}(t) & j = i \\ f_j^{-1}(t) & j > i \end{cases} \tag{2}$$

and therefore satisfy the conditions $\sum_{j=0}^{k-1} x_j = n$ and $\max_{j=0}^{k-1} f_j(x_j) = t$. Moreover, the total amount of n will be distributed in X between vector elements with higher indices, which have lower G cost, $g_i(x)$, because $b_i \geq b_{i+1}, \forall i \in \{0, \cdots, k-2\}$. Therefore, for any other vector $Y = \{y_0, y_1, \cdots, y_{k-1}\}$ satisfying these two conditions, we will have $\sum_{i=0}^{k-1} g_i(y_i) \geq \sum_{i=0}^{k-1} g_i(x_i)$. Indeed, such a vector Y can be obtained from X by relocating certain amounts from vector elements with higher indices to vector elements with lower indices, which will increase the G cost of the relocated amounts. Thus, when the algorithm exits from the *while* loop with $i < k-1$, it returns a Pareto-optimal vector X.

If the algorithm exits from the *while* loop with $i = k-1$, it will mean that t is too big for the given n. We would still have $n_{plus} > 0$ to take off the last vector element, x_{k-1}, but if we did it, we would make $\max_{j=0}^{k-1} f_j(x_j) < t$. This way we would construct for the given n a decision vector, which minimizes $\sum_{i=0}^{k-1} g_i(x_i)$ but whose $\max_{j=0}^{k-1} f_j(x_j)$ will be less than t, which means that no decision vector X such that $\max_{j=0}^{k-1} f_j(x_j) = t$ can be Pareto optimal. Therefore, in this case the algorithm also terminates abnormally.

Thus, for any $t \in T$, Algorithm 2 either finds a Pareto-optimal decision vector X such that $T(X) = t$ and $E(X) = \sum_{i=0}^{k-1} b_i \times x_i = e$, or returns abnormally if such a vector does not exist. Let Algorithm 2 return normally, and the loop variable i be equal to s upon exit from the loop. Then, according to formula 2, $e = \sum_{i=0}^{k-1} b_i \times x_i = b_s \times (n - \sum_{i=s+1}^{k-1} f_i^{-1}(t)) + \sum_{i=s+1}^{k-1} (b_i \times f_i^{-1}(t)) = b_s \times n - \sum_{i=s+1}^{k-1} (b_s - b_i) \times f_i^{-1}(t)$, where s, n, b_i, b_s, a_i are all known constants. Therefore, the Pareto front $e = P_f(t)$ can be expressed as follows:

$$
\begin{aligned}
& e = P_f(t) = b_s \times n - \sum_{i=s+1}^{k-1} (b_s - b_i) \times f_i^{-1}(t) \\
& t_{min} = \min_{X} \{ \max_{j=i}^{k-1} f_j(x_j) \}, t_{max} = f_{k-1}(n) \\
& t \in [t_{min}, t_{max}], \quad s \in \mathbb{Z}_{[0,k-2]},
\end{aligned}
$$

which is the analytical expression of the piece-wise function constructed by Algorithm 1 (LBOPA). *End of Proof.*

Theorem 2. *LBOPA (Algorithm 1) and PARTITION (Algorithm 2) have polynomial time complexities.*

Proof. The *for* loop in LBOPA (Algorithm 1, Lines 3–7) has k iterations. At each iteration i, the computation of t_i has a time complexity of $\mathcal{O}(k^2 \times \log_2 n)$ [12], the computation of e_i has a time complexity of $\mathcal{O}(k)$, and the insertion of the point in the set $\mathcal{S}$ has complexity $\mathcal{O}(1)$. Therefore, the time complexity of the loop is $\mathcal{O}(k^3 \times \log_2 n)$. The time complexity of the loop (Lines 8–10) is $\mathcal{O}(k)$. Therefore, the time complexity of the Algorithm 1 is $\mathcal{O}(k^3 \times \log_2 n)$.

Let us consider the PARTITION Algorithm 2. The initialization of X (Line 2) and computation of n_{plus} has time complexity $\mathcal{O}(k)$ each. The while loop

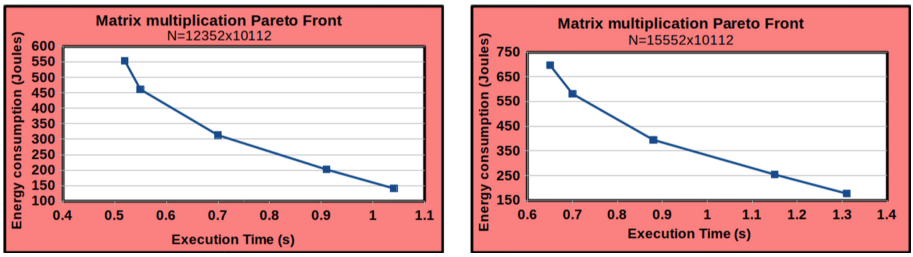

Fig. 3. Pareto front for the matrix multiplication application using five heterogeneous processors described earlier for two workloads. Each Pareto front contains four linear segments.

(Lines 8–17) iterates as long as $n_{plus} > 0$ and $i < k - 1$, of which $i < k - 1$ is the worst case scenario. The time complexity of the loop is, therefore, $\mathcal{O}(k)$. The time complexity of computation of e in Line 21 is $\mathcal{O}(k)$. Therefore, the time complexity of the Algorithm 2 is bounded by $\mathcal{O}(k)$. *End of Proof.*

5 Experimental Results

We employ the LBOPA and PARTITION algorithms to obtain the Pareto fronts for the matrix multiplication application using the five heterogeneous processors mentioned earlier. An automated tool, HCLWATTSUP [4], is used to determine the dynamic and total energy consumptions using system-level physical power measurements using power meters. HCLWATTSUP has no extra overhead and, therefore, does not influence the energy consumption of the kernel. The HCLWATTSUP interface is explained in the supplemental. Several precautions are taken in computing energy measurements to eliminate the potential disturbance due to components such as SSDs and fans. The input performance and dynamic energy functions, (F, G), to LBOPA and PARTITION are linear approximations of the profiles shown in the Fig. 1.

To obtain an experimental data point, the application is executed 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 using Pearson's chi-squared test.

Figure 3 shows the Pareto fronts for two workloads, 12352×10112 and 15552×10112. Each Pareto front contains four linear segments. Each segment is connected by two endpoints. All the points lying on a segment are the performance-energy optimal solutions in the objective space.

For the workload 12352×10112, 17% dynamic energy saving is gained while allowing 5% performance degradation. Similarly, for the workload 15552×10112, 13% energy saving is achieved while tolerating 5% performance degradation.

The first linear segment has a steep slope signifying a significant dynamic energy saving for a slight increase in execution time. The energy savings are 93 J and 106 J for execution time increases of 0.03 s and 0.05 s for the two workloads. The energy-performance tradeoff (that is, the gain in energy saving for a corresponding increase in execution time) decreases with each next linear segment.

Based on an input user-specified energy-performance tradeoff, one can selectively focus on a specific segment to return the Pareto-optimal solutions (workload distributions). The shapes of the two Pareto fronts are similar, suggesting that the qualitative conclusions apply for all workloads for this application.

6 Conclusion

Performance and energy are the two most important objectives for optimization on heterogeneous HPC platforms. This work introduced a mathematical problem motivated by the bi-objective optimization of a matrix multiplication application on heterogeneous HPC platforms for performance and energy. The application exhibits performance functions that are continuous and strictly increasing and energy functions that are linear increasing.

We first formulated the problem, which for a given positive real number n aims to find a vector $X = \{x_0, \cdots, x_{k-1}\} \in \mathbb{R}^k_{\geq 0}$ such that $\sum_{i=0}^{k-1} x_i = n$, minimizing the max of k-dimensional vector of functions of objective type one and the sum of k-dimensional vector of functions of objective type two. We then proposed an algorithm of polynomial complexity solving the problem for the case where all the functions of objective type one are continuous and strictly increasing, and all the functions of objective type two are linear increasing.

We solved the bi-objective optimization problem using the algorithm for the matrix multiplication application employing five heterogeneous processors, two Intel multicore CPUs, an Nvidia K40c GPU, an Nvidia P100 PCIe GPU, and an Intel Xeon Phi. Based on our experiments, 17% dynamic energy saving can be achieved while tolerating a performance degradation of 5% (a saving of 106 J for an execution time increase of 0.05 s).

References

1. Ait Aba, M., Zaourar, L., Munier, A.: Approximation algorithm for scheduling a chain of tasks on heterogeneous systems. In: Heras, D.B., Bougé, L. (eds.) Euro-Par 2017. LNCS, vol. 10659, pp. 353–365. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75178-8_29
2. Chakrabarti, A., Parthasarathy, S., Stewart, C.: A pareto framework for data analytics on heterogeneous systems: implications for green energy usage and performance. In: 2017 46th International Conference on Parallel Processing (ICPP), pp. 533–542. IEEE (2017)
3. Durillo, J.J., Nae, V., Prodan, R.: Multi-objective energy-efficient workflow scheduling using list-based heuristics. Futur. Gener. Comput. Syst. **36**, 221–236 (2014)

4. Fahad, M., Manumachu, R.R.: HCLWattsUp: energy API using system-level physical power measurements provided by power meters. Heterogeneous Computing Laboratory, University College Dublin, April 2021. https://csgitlab.ucd.ie/manumachu/hclwattsup
5. Fard, H.M., Prodan, R., Barrionuevo, J.J.D., Fahringer, T.: 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), CCGRID 2012, pp. 300–309. IEEE Computer Society (2012)
6. Gholkar, N., Mueller, F., Rountree, B.: Power tuning HPC jobs on power-constrained systems. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, pp. 179–191. ACM (2016)
7. Kessaci, Y., Melab, N., Talbi, E.G.: A pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. Clust. Comput. **16**(3), 451–468 (2013)
8. Khaleghzadeh, H., Fahad, M., Shahid, A., Manumachu, R.R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution. IEEE Trans. Parallel Distrib. Syst. **32**(3), 543–560 (2021)
9. Khaleghzadeh, H., Fahad, M., Reddy Manumachu, R., Lastovetsky, A.: A novel data partitioning algorithm for dynamic energy optimization on heterogeneous high-performance computing platforms. Concurr. Comput.: Pract. Exper. **32**(21), e5928 (2020)
10. Kołodziej, J., Khan, S.U., Wang, L., Zomaya, A.Y.: Energy efficient genetic-based schedulers in computational grids. Concurr. Comput.: Pract. Exper. **27**(4), 809–829 (2015)
11. Lang, J., Rünger, G.: An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration. J. Parallel Distrib. Comput. **74**(9), 2884–2897 (2014)
12. Lastovetsky, A., Reddy, R.: Data partitioning with a realistic performance model of networks of heterogeneous computers. In: 2004 Proceedings of 18th International Parallel and Distributed Processing Symposium, p. 104 (2004)
13. Lastovetsky, A., Reddy, R.: Data partitioning with a functional performance model of heterogeneous processors. Int. J. High Perform. Comput. Appl. **21**, 76–90 (2007)
14. Lastovetsky, A., Reddy, R.: New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. IEEE Trans. Parallel Distrib. Syst. **28**(4), 1119–1133 (2017)
15. Manumachu, R.R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Trans. Comput. **67**(2), 160–177 (2018)
16. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer (1999)
17. Reddy Manumachu, R., Lastovetsky, A.L.: Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution. Concurr. Comput.: Pract. Exper. **31**(4), e4958 (2019)
18. Rossi, F.D., Xavier, M.G., De Rose, C.A., Calheiros, R.N., Buyya, R.: E-eco: performance-aware energy-efficient cloud data center orchestration. J. Netw. Comput. Appl. **78**, 83–96 (2017)
19. Rountree, B., Lowenthal, D.K., Funk, S., Freeh, V.W., de Supinski, B.R., Schulz, M.: Bounding energy consumption in large-scale MPI programs. In: SC 2007: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, pp. 1–9 (2007)

20. Talbi, E.G.: Metaheuristics: from Design to Implementation, vol. 74. Wiley, Hoboken (2009)
21. Tarplee, K.M., Friese, R., Maciejewski, A.A., Siegel, H.J., Chong, E.K.: Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques. IEEE Trans. Parallel Distrib. Syst. **27**(6), 1633–1646 (2016)
22. Yu, L., Zhou, Z., Wallace, S., Papka, M.E., Lan, Z.: Quantitative modeling of power performance tradeoffs on extreme scale systems. J. Parallel Distrib. Comput. **84**, 1–14 (2015)