Received 21 February 2023, accepted 14 March 2023, date of publication 17 March 2023, date of current version 22 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3258684

RESEARCH ARTICLE

Acceleration of Bi-Objective Optimization of Data-Parallel Applications for Performance and Energy on Heterogeneous Hybrid Platforms

RAVI REDDY MANUMACHU[1], (Member, IEEE), HAMIDREZA KHALEGHZADEH[2], AND ALEXEY LASTOVETSKY[1], (Member, IEEE)
[1]School of Computer Science, University College Dublin, Belfield, Dublin 4, D04 V1W8 Ireland
[2]School of Computing, University of Portsmouth, PO1 2UP Portsmouth, U.K.

Corresponding author: Ravi Reddy Manumachu (ravi.manumachu@ucd.ie)

This work was supported in part by the Sustainable Energy Authority of Ireland (SEAI) under Grant 21/RDD/664, and in part by the Science Foundation Ireland (SFI) through the SFI Frontiers for the Future Programme under Grant 20/FFP-P/8683.

ABSTRACT Accelerating the bi-objective optimization of applications for performance and energy is crucial to achieving energy efficiency objectives and meeting quality-of-service requirements in modern high-performance computing platforms and cloud computing infrastructures. In this work, we highlight the crucial challenges to accelerate model-based methods proposed for the bi-objective optimization of data-parallel applications for performance and energy that employ workload distribution between the executing processors as the decision variable. The methods solve unconstrained bi-objective optimization problems and take input, the processors' performance and energy profiles in the form of discrete functions of workload size, and output Pareto-optimal solutions (workload distributions), minimizing the execution time and the total energy consumption of computations during the parallel execution of the application. One of the challenges is the fast computation of Pareto-optimal solutions. We then formulate the bi-objective optimization problem of data-parallel applications for performance and energy through workload distribution on a cluster of p identical hybrid nodes, each containing h heterogeneous processors. The state-of-the-art algorithm for solving the problem is sequential and takes exorbitant execution times to find Pareto-optimal solutions for even moderate numbers of processors. We propose two algorithms that address this shortcoming. The first algorithm is an exact sequential algorithm that is more efficient and amenable to parallelization and achieves a complexity reduction of $\mathcal{O}(m \times h)$ over the state-of-the-art sequential algorithm where m is the cardinality of the input discrete execution time and dynamic energy functions. The second algorithm is a parallel algorithm executed by q identical parallel processes that reduces the complexity of our proposed sequential algorithm by $\mathcal{O}(q)$. It, therefore, achieves a complexity reduction of $\mathcal{O}(m \times h \times q)$ over the state-of-the-art sequential algorithm. Finally, we experimentally analyze the practical efficacy of our proposed algorithms for two data-parallel applications, matrix multiplication and fast Fourier transform, on a heterogeneous hybrid node containing an Intel Haswell multicore CPU, an Nvidia k40c GPU, and an Nvidia P100 GPU and simulations of clusters of such hybrid nodes. The experiments demonstrate that our proposed algorithms provide tremendous speedups over state-of-the-art solutions.

INDEX TERMS High-performance heterogeneous computing, energy-efficient computing, bi-objective optimization, performance optimization, energy optimization, data-parallel applications, workload distribution.

I. INTRODUCTION

Performance and energy are the two most important objectives for optimization in modern high performance computing (HPC) platforms, computational grids, data centers,

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

and cloud computing infrastructures ([1], [2], [3], [4]). Furthermore, achieving the energy efficiency objectives and meeting the performance or response time quality-of-service (QoS) requirements is crucial to the commercial viability of data centers and cloud computing infrastructures. Therefore, fast solution methods for system-level and application-level bi-objective optimization for performance and energy are vital to addressing energy efficiency, environmental and commercial concerns in these computing settings.

We start with a brief review of system-level and application-level solution methods for bi-objective optimization on HPC platforms for performance and energy. Then, we present the state-of-the-art application-level solution methods for bi-objective optimization of data-parallel applications on modern HPC platforms for performance and energy, employing workload distribution as a decision variable. Finally, we summarize the challenges specific to the acceleration of the state-of-the-art application-level solution methods before presenting our solution addressing the challenges.

System-level solution methods aim to optimize the performance and energy of the environment executing the applications [2], [5], [6], [7], [8], [9]. The methods employ application-agnostic models and hardware parameters as decision variables. Dynamic voltage and frequency scaling (DVFS) is a dominant decision variable in this category. The application-level solution methods [10], [11], [12], [13], [14], [15] utilize application-level decision variables and models. The decision variables include the number of threads, number of processors, loop tile size, and workload distribution.

We will now review the state-of-the-art application-level solution methods for bi-objective optimization of data-parallel applications on modern HPC platforms for performance and energy, employing workload distribution as a decision variable. First, briefly, some terminology on energy consumption. The total energy consumption during an application execution is the sum of dynamic and static energy consumption. We define static energy consumption as the energy consumed by the platform without the application execution. Dynamic energy consumption is calculated by subtracting this static energy consumption from the total energy consumed by the platform during the application execution.

Lastovetsky and Reddy [11] propose data partitioning algorithms that solve single-objective optimization problems of data-parallel applications for performance or dynamic energy on homogeneous clusters of multicore CPUs. The algorithms take as input application-specific performance and dynamic energy profiles of the multicore CPU processor. The profiles are discrete functions of workload size with no shape assumptions that accurately and realistically account for resource contention and non-uniform memory access (NUMA) inherent in modern multicore CPU platforms. The algorithms output performance-optimal and energy-optimal workload distributions. Furthermore, Reddy and Lastovetsky [12] propose an exact global optimization algorithm to solve the bi-objective optimization problem of data-parallel applications for performance and energy on homogeneous clusters of multicore CPUs. The algorithm takes discrete performance and dynamic energy functions against workload size as input and outputs the globally Pareto-optimal set of solutions.

The algorithms ([11], [12]) target homogeneous HPC platforms and exhibit time complexity of $\mathcal{O}(m^2 \times p^2)$ where m is the cardinality of the discrete sets representing the speed and energy functions, and p is the number of available processors. However, the algorithms are sequential and exhibit impractical runtime and memory costs for large values of p (several hundred). To address this problem, Manumachu and Lastovetsky [16] propose parallel versions of the sequential data partitioning algorithms presented in [11] that achieve a complexity reduction of $\mathcal{O}(p)$. However, the parallel algorithms return only the performance-optimal and energy-optimal workload distributions.

Khaleghzadeh et al. [15] study the bi-objective optimization problem of data-parallel applications for performance and energy on heterogeneous processors. They propose a solution method (HEPOPTA) comprising an efficient and exact global optimization algorithm. The algorithm takes input performance and dynamic energy profiles of the processors as arbitrary discrete functions of workload size and returns the Pareto-optimal solutions (generally, load imbalanced). Furthermore, HEPOPTA employs a methodology [17] that accurately models the energy consumption by a hybrid data-parallel application executing on a heterogeneous HPC platform containing different computing devices using system-level power measurements provided by power meters, which is considered the *ground-truth*.

We now summarize the main steps in the application-level method ([15]) to solve the bi-objective optimization problem of data-parallel applications for performance and energy on heterogeneous hybrid platforms.

The first step involves modelling a data-parallel application executing on a heterogeneous hybrid platform. Such a hybrid application consists of several kernels (generally speaking, multithreaded) running in parallel on different computing devices of the platform. Due to the inherent tight integration and severe resource contention, the load of one computational kernel in the application may significantly impact others' performance to prevent the ability to model the performance and energy consumption of each kernel in hybrid applications individually [18]. Therefore, configurations of the hybrid application with no more than one CPU kernel or accelerator kernel running on the corresponding device are only considered. Each group of cores executing an individual kernel of the application is modelled as an abstract processor to represent the executing platform as a set of heterogeneous abstract processors. Such a grouping ensures that the sharing of system resources is maximized within groups of computational cores representing the abstract processors and minimized between the groups. This way,

TABLE 1. Specifications of the hybrid node containing an intel haswell multicore CPU, an nvidia K40c, and an nvidia P100 PCI-E GPU.

Intel Haswell E5-2670V3	
No. of cores per socket	12
Socket(s)	2
CPU MHz	1200.402
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
Total main memory	64 GB DDR4
Memory bandwidth	68 GB/sec
NVIDIA K40c	
No. of processor cores	2880
Total board memory	12 GB GDDR5
L2 cache size	1536 KB
Memory bandwidth	288 GB/sec
NVIDIA P100 PCIe	
No. of processor cores	3584
Total board memory	12 GB CoWoS HBM2
Memory bandwidth	549 GB/sec

the contention and mutual dependence between abstract processors are minimized.

Therefore, a hybrid data-parallel application is represented by a set of computational kernels executing on groups of cores, which we term *heterogeneous abstract processors*. For example, consider the hybrid node employed in our experiments in this work and whose specification is shown in Table 1. It consists of a dual-socket Intel Haswell multicore CPU involving 24 physical cores with 64 GB main memory, which hosts two accelerators, an Nvidia K40c GPU and an Nvidia P100 GPU. So, we model a data-parallel application executing on this node by three heterogeneous abstract processors, CPU_1, GPU_1, and GPU_2. CPU_1 comprises 22 (out of a total of 24) CPU cores. GPU_1 symbolizes the Nvidia K40c GPU and a host CPU core connected to this GPU via a dedicated PCI-E link. The Nvidia P100 PCI-E GPU and a host CPU core connected to this GPU via a dedicated PCI-E link are denoted by GPU_2.

Second, the computational kernels' performance and dynamic energy profiles are built offline using a methodology based on processor clocks and system-level power measurements provided by external power meters (*ground-truth method*). In this work, when we say the performance and dynamic energy profiles of the heterogeneous processors, we mean the profiles of the three computational kernels executing on the three heterogeneous abstract processors.

Finally, given the performance and dynamic energy profiles, a data-partitioning algorithm solves the bi-objective optimization problem to determine the Pareto-optimal solutions (workload distributions), minimizing the execution time and the total energy consumption of computations during the parallel execution of the application. We term the bi-objective optimization problem HEPOPT that we formulate in the following section.

However, there are two issues with the application-level solution method [15]. First, the data-partitioning algorithm is *sequential* and takes exorbitant execution times for even moderate values of p. For example, consider its execution times for HEPOPTA [15] solving the bi-objective optimization problem for two scientific data-parallel applications, matrix multiplication (DGEMM) and 2D fast Fourier transform (2D-FFT), executed on a platform comprising two connected heterogeneous multi-accelerator NUMA nodes. For the DGEMM application, the data-partitioning algorithm's execution time ranges from 4 seconds to 6 hours for values of p varying from 12 to 192. For the 2D-FFT application, the execution time increases from 16 seconds to 16 hours for values of p, going from 12 to 192.

Second, the procedure to construct the performance and dynamic energy profiles employing system-level power measurements provided by external power meters (*ground-truth method*) is also sequential and expensive. The execution times of constructing the discrete performance and dynamic energy profiles comprising 210 and 256 workload sizes for the two applications, DGEMM and 2D-FFT, are 8 hours and 14 hours, respectively. Finally, the dynamic energy profiles are constructed offline using the ground-truth method. Briefly, while the ground-truth method exhibits the highest accuracy, it is also the most expensive [17]. In addition, it cannot be employed in dynamic environments (HPC platforms and data centers) containing nodes that are not equipped with power meters.

Therefore, there are two crucial challenges to accelerating the bi-objective optimization of data-parallel applications for performance and energy on modern heterogeneous HPC platforms: (a) Fast computation of Pareto-optimal solutions optimizing the application for performance and energy, and (b) Fast construction of performance and dynamic energy profiles that are discrete functions of workload size by employing an energy measurement method that caters to environments where nodes are not equipped with power meters.

In this work, we propose algorithms that address the first challenge, the fast computation of Pareto-optimal solutions optimizing the application for performance and energy. The second challenge is an open problem. Therefore, we present just an overview of the issues involved.

To construct the performance and dynamic energy profiles of the heterogeneous processors (components), we need accurate and reliable methods for component-level measurement of the execution time and energy. Since all processing units are equipped with sufficiently precise clocks, obtaining accurate performance profiles is achievable (even in the case of tightly coupled components). However, methods for component-level measurement of energy consumption present a real challenge.

There are three component-level energy measurement approaches. The first approach employs system-level physical measurements using external power meters, which is the most accurate but very expensive. In our experience, obtaining a single experimental point with sufficient statistical confidence can take hours and even days. Nevertheless, the measurements obtained this way are considered ground truth [19]. The second approach is to use on-chip power sensors such as Intel RAPL (Running Average Power Limit), Intel Xeon Phi SMC (System Management Controller), AMD APM (Application Power Management), Nvidia NVML (Nvidia Management Library). Unfortunately, while cheap and efficient, on-chip power sensors have been found to be inaccurate and poorly documented [19]. Furthermore, extensive experiments with highly optimized scientific kernels on several mainstream CPUs and accelerators have shown that energy profiles constructed using state-of-the-art on-chip power sensors are qualitatively inaccurate [19].

The third approach uses software energy predictive models employing various measurable runtime performance-related predictor variables. This approach is the only realistic alternative to the ground-truth method. There has been good progress in energy modelling for multicore CPUs built on the selection of model variables that satisfy basic laws of energy conservation [20]. For example, best models employing performance monitoring counters (PMCs) and utilization variables have 10-20% accuracy on popular scientific kernels [21]. However, accelerators are poorly instrumented for runtime energy modelling. As a result, runtime modeling of energy consumption by components running on accelerators needs to be developed more.

We first formulate the bi-objective optimization problem (HEPOPT) of data-parallel applications for performance and energy through workload distribution on a cluster of p identical hybrid nodes, each containing h heterogeneous processors. The inputs to HEPOPT are the workload size, n; the number of identical nodes, p; the number of heterogeneous processors in each node, h; the sets T and E containing the discrete functions of execution time and dynamic energy against the workload size for the h heterogeneous processors; the static power consumption of a node, P_s. Finally, the output is the Pareto-optimal solutions for performance and total energy.

The problem formulation has some additional constraints. First, the workload size is assumed to be multiple of a basic computation unit that does not differ during the application execution. For example, a basic computation unit in matrix multiplication is a matrix update, $a + b \times c$, where a, b, and c are $u \times u$ matrices of fixed size u. Therefore, the workload sizes are given in multiples of $2 \times u^3$, which is the number of arithmetic operations in the basic computation unit. Second, the discrete sets representing the execution time and dynamic energy functions have cardinality m. The set of workload sizes in the functions is $\{1, \cdots, m\}$. Therefore, the workload size n has an upper bound, $p \times m \times h$.

The state-of-the-art sequential algorithm, HEPOPTA [15], solves HEPOPT with time complexity of $\mathcal{O}(m^3 \times p^3 \times h^3 \times \log_2(m \times p \times h))$ where m is the maximum cardinality of the discrete sets representing the performance and energy profiles of the h heterogeneous processors. However, HEPOPTA can handle input discrete sets of different cardinalities, and arbitrary sizes can separate the workload sizes in the sets. Furthermore, the workload sizes can be positive reals.

We propose a two-level hierarchical sequential algorithm called HEPOPTADP that is more efficient and amenable to parallelization. It uses the dynamic programming technique and employs HEPOPTA as the fundamental building block at the second level. The inputs to HEPOPTADP are the workload size, n; the number of identical nodes, p; the number of heterogeneous processors in each node, h; the sets T and E containing the discrete execution time and dynamic energy functions of the h heterogeneous processors; the static power consumption of a node, P_s. HEPOPTADP returns the Pareto-optimal solutions for performance and total energy. Each solution, generally not load balanced, is a workload distribution between the $p \times h$ processors where the workloads in the distribution can be different.

The time complexity of HEPOPTADP is $\mathcal{O}(n \times p^2 \times m \times h \times \log_2(p \times m \times h))$. Therefore, it achieves a complexity reduction of $\mathcal{O}(\frac{m^2 \times h^2 \times p}{n})$ over HEPOPTA. Therefore, the speedup is constant for a constant amount of work per processor, $\frac{n}{p \times h}$, keeping the other application input parameters (execution time and dynamic energy functions) the same. The reduction is due to the hierarchical design and memoization employed in dynamic programming technique. Since the workload size n has an upper bound, $p \times m \times h$, the lower bound on the complexity reduction will be $\mathcal{O}(m \times h)$.

We then propose a parallel version of HEPOPTADP called PARHEPOPTA executed by q identical parallel processes and time complexity $\mathcal{O}(n \times \frac{p^2}{q} \times m \times h \times \log_2(p \times m \times h))$. Therefore, it reduces the time complexity of HEPOPTADP by $\mathcal{O}(q)$ and HEPOPTA by $\mathcal{O}(\frac{m^2 \times h^2 \times p \times q}{n})$). Since $n \leq p \times m \times h$, the lower bound on the complexity reduction will be $\mathcal{O}(q \times m \times h)$. Therefore, the speedup increases with q for a constant $\frac{n}{p \times h}$ keeping the other application input parameters the same.

We developed two implementations of PARHEPOPTA. The first implementation employs q MPI processes. The second implementation uses hybrid parallelism and is executed by q MPI processes, each employing t threads. Experimentally, we find that the first implementation performs better. Therefore, we describe this implementation here. The implementation using hybrid parallelism is given in the supplemental.

We study the practical efficacy of HEPOPTADP and PARHEPOPTA for two data-parallel applications, matrix multiplication (DGEMM) and fast Fourier transform (2D-FFT), using the hybrid node (Table 1). The DGEMM application computes the matrix product of two square matrices of size $N \times N$. The 2D-FFT application computes the 2D discrete Fourier Transform of a complex signal matrix of size $N \times N$.

First, we present the speedups of PARHEPOPTA and HEPOPTADP over HEPOPTA. Since we do not have their GPU implementations, all three algorithms are executed on the hybrid node employing only the Intel multicore CPU. HEPOPTADP and HEPOPTA are serial algorithms executed by one process. q processes execute PARHEPOPTA. Experimentally, the best value of q on the experimental compute node is 24, equal to the total number of CPU cores.

Our experiments show that the proposed algorithms PARHEPOPTA and HEPOPTADP exhibit good speedups over HEPOPTA. For example, for DGEMM application with configuration $(N = 70656, p = 256, h = 3, m = 211)$, the speedup of HEPOPTADP over HEPOPTA is 4.6, and the speedup of PARHEPOPTA over HEPOPTADP is 6.57. On the other hand, for 2D-FFT application with $(N = 96000, p = 256, h = 3, m = 256)$, the speedups are 2.12 and 10, respectively. Thus, the total speedups of PARHEPOPTA over HEPOPTA are 30.25 and 21.05 for DGEMM and 2D-FFT, respectively.

Second, we provide examples of the usage of PARHEPOPTA. Finally, we present scenarios where PARHEPOPTA can accelerate the development of energy-efficient parallel applications and long-running simulations like numerical weather prediction due to its tremendous speedups.

In summary, the main original contributions of this work are:

- The formulation of the bi-objective optimization problem (HEPOPT) of data-parallel applications for performance and energy on a cluster of p identical hybrid nodes, each containing h heterogeneous processors;
- A model-based dynamic programming data partitioning algorithm, HEPOPTADP, solving HEPOPT that is more efficient and amenable to parallelization than the state-of-the-art algorithm. HEPOPTADP takes input discrete execution time and dynamic energy functions of cardinality m with any arbitrary shape and returns the Pareto front of load imbalanced solutions and best load balanced solutions. It achieves a complexity reduction over the state-of-the-art sequential algorithm that has the lower bound of $\mathcal{O}(m \times h)$;
- A parallel version of HEPOPTADP executed by q identical parallel processes called PARHEPOPTA solving HEPOPT. It reduces the time complexity of HEPOPTADP by $\mathcal{O}(q)$. It achieves a complexity reduction over the state-of-the-art sequential algorithm that has the lower bound of $\mathcal{O}(m \times h \times q)$;
- Two novel algorithms, which HEPOPTADP and PARHEPOPTA invoke. The first algorithm, PoP, combines k input Pareto fronts, each containing n points with a complexity of $\mathcal{O}(k \times n)$. The second algorithm, ConPar, determines the dominating Pareto front given k input Pareto fronts, each containing n points with a complexity of $\mathcal{O}(k \times n \times \log_2(k \times n))$;
- Experiments on a heterogeneous hybrid node and simulations of clusters of such hybrid nodes demonstrating that our proposed algorithms exhibit good speedups over the state-of-the-art solutions.

The rest of the paper is organized as follows. Section II contains the formulation of the bi-objective optimization problem for performance and energy on heterogeneous hybrid platforms. Our proposed sequential and parallel algorithms, HEPOPTADP and PARHEPOPTA, are described in Sections III and IV. Section V contains the experimental results. Section VI presents the related work. Finally, we conclude the paper in Section VII.

II. BI-OBJECTIVE OPTIMIZATION PROBLEM ON HYBRID NODES FOR PERFORMANCE AND TOTAL ENERGY: PROBLEM FORMULATION

This section formulates HEPOPT, the bi-objective optimization problem for data-parallel applications for performance and total energy on a heterogeneous platform of p identical hybrid nodes, each containing h heterogeneous processors. The problem employs workload distribution as the decision variable.

Consider a workload size n executed using p identical nodes, each containing h heterogeneous processors. Let the sets, $T = \{t_1(x), \ldots, t_h(x)\}$ and $E = \{e_1(x), \ldots, e_h(x)\}$, contain the discrete execution time and dynamic energy functions of the h heterogeneous processors. The function $e_i(x)$ represents the amount of dynamic energy consumed by P_i to execute the workload size x, and $t_i(x)$ is the execution time of the workload size on this processor.

The workload size n is assumed to be multiple of a basic computation unit that does not differ during the application execution. The discrete sets representing the execution time and dynamic energy functions are assumed to have cardinality m. The set of workload sizes in the functions is $\{1, \cdots, m\}$. Therefore, the maximum workload size n that can be solved is $p \times m \times h$.

HEPOPT is then formulated as follows:

HEPOPT$(n, p, h, T, E, P_s, \mathcal{D})$:

$$f_T(D) = \max_{i=1}^{p} \max_{j=1}^{h} t_j(d_{ij})$$
$$f_E(D) = \sum_{i=1}^{p} (P_s \times \max_{j=1}^{h} t_j(d_{ij}) + \sum_{j=1}^{h} e_j(d_{ij}))$$
$$\underset{D}{\text{minimize}} \quad (f_T(D), f_E(D))$$
$$\text{subject to:} \quad \sum_{i=1}^{p} \sum_{j=1}^{h} d_{ij} = n,$$
$$0 \leq d_{ij} \leq m, i \in \{1, \cdots, p\}, j \in \{1, \cdots, h\} \tag{1}$$

The two objective functions are $f_T(D)$ and $f_E(D)$. The objective function $f_T(D)$ gives the parallel execution time of the workload employing the workload distribution, $D = [d_{ij}]_{p \times h}, i = 1, 2, \ldots, p, j = 1, 2, \ldots, h$. The objective function $f_E(D)$ gives the total energy consumption during the execution of the workload. We use $T \times E : \mathbb{R}^k_{\geq 0} \times \mathbb{R}^k_{\geq 0}$ to denote the objective space of this problem.

HEPOPT returns Pareto-optimal solutions (workload distributions), minimizing the two objective functions. Our solution finds a set of triplets, $\mathcal{P} = \{(f_T(D), f_E(D), D)\}$, such that D is a Pareto-optimal decision vector and the projection of $\mathcal{P}$ onto the objective space is the Pareto front.

The state-of-the-art sequential algorithm, HEPOPTA [15], solves HEPOPT with time complexity of $\mathcal{O}(m^3 \times p^3 \times h^3 \times \log_2(m \times p \times h))$. We first propose an algorithm, HEPOPTADP employing the dynamic programming approach to solve HEPOPT, which is more efficient and amenable to parallelization. Then, we describe our parallel algorithm, PARHEPOPTA, based on HEPOPTADP.

III. HEPOPTADP: DYNAMIC PROGRAMMING BI-OBJECTIVE OPTIMIZATION ALGORITHM

We propose an algorithm HEPOPTADP employing the dynamic programming approach to solve HEPOPT, which is amenable to parallelization. HEPOPTADP is a two-level hierarchical algorithm employing HEPOPTA as the fundamental building block at the second level.

The state-of-the-art sequential algorithm, HEPOPTA [15], solves HEPOPT with time complexity of $\mathcal{O}(m^3 \times p^3 \times h^3 \times \log_2(m \times p \times h))$. HEPOPTA considers the cluster of p identical nodes, each containing h processors, as a 1D array of $p \times h$ heterogeneous processors. HEPOPTA is a branch-and-bound algorithm that organizes workload distributions, $(x_1, \cdots, x_{p\times h})$, efficiently as a tree while considering only distributions that sum to n to determine the Pareto front.

For example, let us consider HEPOPTA solving a workload size $n = 8$ on $p = 2$ identical nodes containing $h = 2$ processors each. The other inputs are two discrete execution time and dynamic energy functions of the heterogeneous abstract processors. Let us assume the set of workload sizes in the functions is $\{1, \cdots, 4\}$. First, HEPOPTA converts the platform configuration (p, h) into a 1D array of 4 heterogeneous processors (P_0, P_1, P_2, P_3) with four corresponding discrete execution time and dynamic energy discrete functions. Then it organizes the workload distributions between the four processors as a tree containing four levels. The first level contains allocations to processor P_0, the second level with allocations to processors P_0 and P_1, and the last level contains the workload distribution for all four processors. Therefore, the first level contains five branches with workload size allocations $\{0, 1, 2, 3, 4\}$ to P_0. The first branch is then extended to five other branches with workload size allocations $\{0, 0\}, \{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 4\}\}$ to P_0 and P_1 and so on. Then, it applies constraints to traverse only branches where the processor allocations sum to 8 and other optimizations to determine the Pareto front. HEPOPTA is detailed in the supplemental, Section III.

The inputs to HEPOPTADP are the workload size, n; the number of identical nodes, p; the number of heterogeneous processors in each node, h; the sets T and E containing the discrete execution time and dynamic energy functions; the static power consumption of a node, P_s.

The output of HEPOPTADP is a Pareto front represented by a set of tuples, $\mathcal{P} = \{(f_E(D), f_T(D), D)_i\}, i \in [1\ ..\ K]$, where $D = [d_{ij}]_{p\times h}, i = 1, 2, \ldots, p, j = 1, 2, \ldots, h$ is the optimal workload distribution, $f_E(D)$ is the total energy, and $f_T(D)$ is the execution time corresponding to the optimal workload distribution. The solutions in the set $\mathcal{P}$ are output in increasing order of total energy.

HEPOPTADP has three core components: a). Recurrence relation, b). Tabular computation, and c). Traceback. HEPOPTADP uses the recurrence relations to compute the cells in the dynamic programming table, dpt, of size $n \times p$, as shown in Figure 1(a). The parameter h is not shown in the figure since it is employed in the HEPOPTA invocations to obtain the Pareto fronts for the red and orange cells.

HEPOPTADP divides the partitioning of workload size r between c processors into the following sub-problems:

- Allocating i to one processor and the remaining $r - i$ to $c - 1$ processors.
- Allocating r to i processors, $i = 1, \ldots, c - 1$. Since $dpt(r, c - 1)$ is composed of the solutions to sub-problems, $dpt(r, 1), \ldots, dpt(r, c - 2)$, the only sub-program required to be solved is allocating r to $c - 1$ processors.

The recurrence relations are given below:

$$\begin{aligned}
dpt(1, c) &= \text{HEPOPTA}(1, h, T, E, P_s), \quad c = 1, \ldots, p\\
dpt(r, 1) &= \text{HEPOPTA}(r, h, T, E, P_s),\\
&\quad r = 2, \ldots, \min(n, m * h)\\
dpt(r, 1) &= \Phi, \quad r = \min(n, m * h) + 1, \ldots, n\\
dpt(r, c) &= \text{ConPar}(\text{PoP}(dpt(1, 1),\ dpt(r - 1, c - 1)), \ldots,\\
&\qquad \text{PoP}(dpt(rmax, 1),\ dpt(r - rmax, c - 1)),\\
&\qquad dpt(r, c - 1)),\\
rmax &= \min(r - 1, m \times h),\\
&\quad r = 2, \ldots, n, c = 2, \ldots, p
\end{aligned} \tag{2}$$

The row and column dimensions are given by r and c. The cell, $dpt(r, c), r = 1, 2, \ldots, n, c = 1, 2, \ldots, p$, contains the Pareto front to solve the workload size r using c nodes. We denote the Pareto front by $\mathcal{P}_{rc}$. At the end of the execution of HEPOPTADP, the cell $dpt(n, p)$ contains the Pareto front to solve the workload size n using the p identical nodes.

The top three recurrence relations (base conditions) represent the base initialization step, which computes the red cells in the first row and orange cells in the first column, as shown in Figure 1(b). The cell, $dpt(1, c)$, $c = 1, \ldots, p$, contains the Pareto front to solve the workload size one using c nodes. The workload distribution is trivial in this case, one node gets the workload, and the rest get zero. Since each node contains h heterogeneous processors, HEPOPTADP invokes HEPOPTA to determine the Pareto front, $\mathcal{P}_{1c}$, to solve the workload using the heterogeneous processors. The cell, $dpt(r, 1), r = 1, \ldots, m * h$, contains the Pareto front $\mathcal{P}_{r1}$ to solve the workload size r using one node. HEPOPTADP invokes HEPOPTA to determine the Pareto front solving the workload using h heterogeneous processors. The cell, $dpt(r, 1), r = m * h + 1, \ldots, n$, will contain a null Pareto front since $m * h$ is the maximum workload size that can be solved using h heterogeneous processors.

The fourth recurrence relation represents the computation of the cells in an anti-diagonal, $L = 3, \ldots, n + p - 1$. The

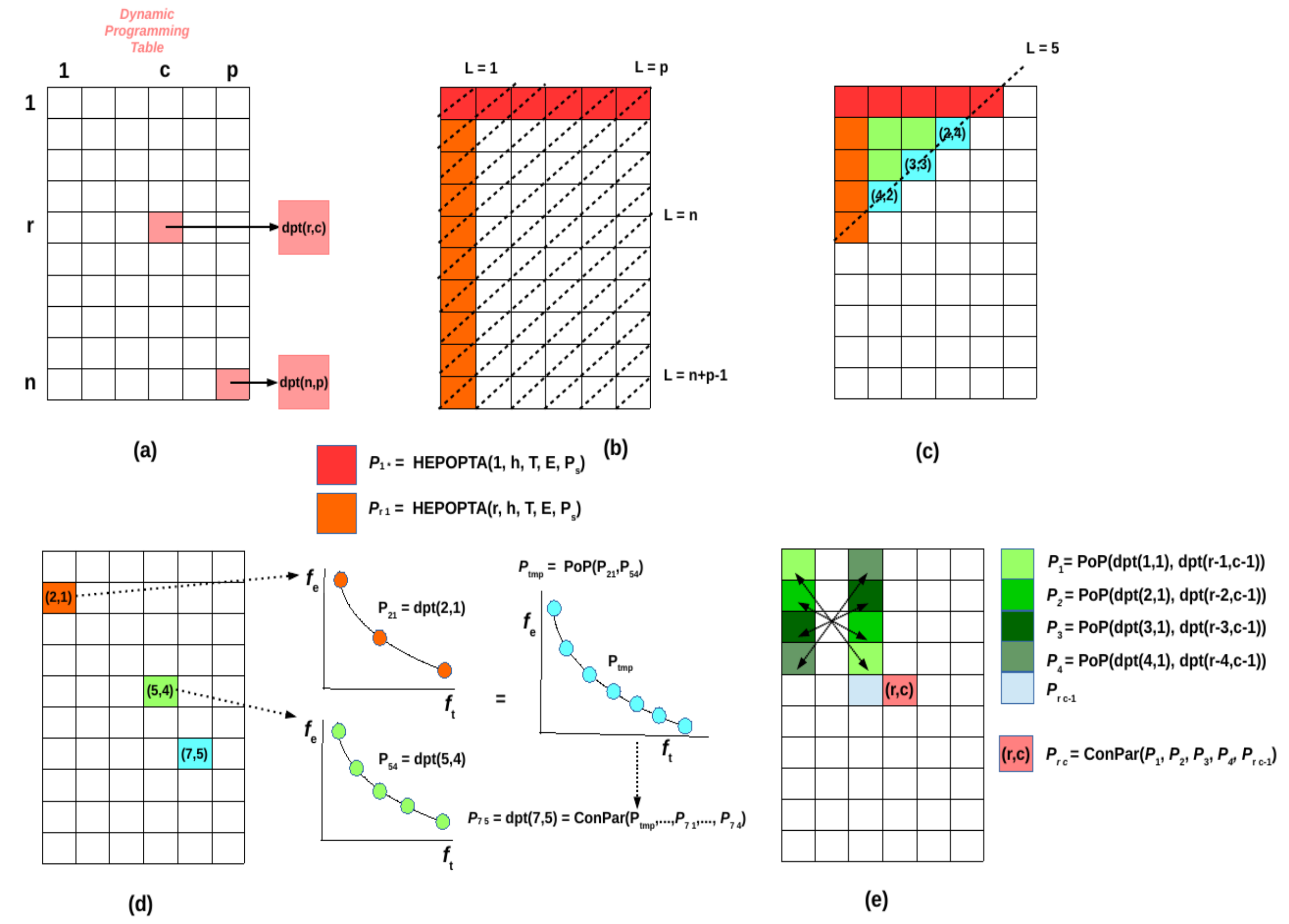

Dynamic Programming Table
1
c
p
1
r
n
dpt(r,c)
dpt(n,p)
(a)
L = 1
L = p
L = n
L = n+p-1
(b)
L = 5
(2,4)
(3,3)
(4,2)
(c)
P1* = HEPOPTA(1, h, T, E, Ps)
Pr1 = HEPOPTA(r, h, T, E, Ps)
(2,1)
(5,4)
(7,5)
fe
ft
P21 = dpt(2,1)
P54 = dpt(5,4)
Ptmp = PoP(P21,P54)
Ptmp
=
P75 = dpt(7,5) = ConPar(Ptmp,...,P71,..., P74)
(d)
(r,c)
P1= PoP(dpt(1,1), dpt(r-1,c-1))
P2= PoP(dpt(2,1), dpt(r-2,c-1))
P3= PoP(dpt(3,1), dpt(r-3,c-1))
P4= PoP(dpt(4,1), dpt(r-4,c-1))
Pr c-1
Prc = ConPar(P1, P2, P3, P4, Pr c-1)
(e)

FIGURE 1. The data structures and the main steps in the execution of HEPOPTADP. (a). The dynamic programming table is of size $n \times p$. The parameter h is not shown since it is employed in the HEPOPTA invocations to obtain the Pareto fronts for the red and orange cells. The discrete functions of execution time (T) and dynamic energy (E) are assumed to have cardinality m. The workload sizes in the functions and n are multiples of a basic computation unit. Without any loss of generality, the set of workload sizes in the functions is assumed to be $\{1, \cdots, m\}$. Therefore, the workload size n has an upper bound, $p \times m \times h$. (b). The base initialization step (top three recurrence relations) computes the red and the orange cells. The red cells contain the Pareto front obtained by invoking HEPOPTA($1, h, T, E, P_S$), solving workload size one using h processors. The orange cells contain the Pareto front output by HEPOPTA(r, h, T, E, P_S) solving workload size r using h processors. The dashed lines show the anti-diagonals, $L = 1, \ldots, n+p-1$. (c). The Pareto fronts for the blue cells, {(4,2),(3,3),(2,4)}, in the anti-diagonal, L=5, are obtained using the Pareto fronts in the cells in the anti-diagonals preceding L. There are no dependencies between computations of blue cells. (d). The routine PoP is used to combine two Pareto fronts. The computation of the Pareto front for cell (7,5) involves a PoP invocation to combine the Pareto fronts in the orange and green cells (2,1) and (5,4). (e). The computation of the Pareto front for the cell, $dpt(r, c) = dpt(5, 4) = \mathcal{P}_{rc} = \mathcal{P}_{54}$, will involve PoP invocations to combine the Pareto fronts in the cell pairs, $\{(1, 1), (4, 3)\}, \ldots, \{(4, 1), (1, 3)\}$. The routine ConPar determines the Pareto front in $dpt(5, 4) = \mathcal{P}_{54}$ from the input Pareto fronts, $\{\mathcal{P}_1, \ldots, \mathcal{P}_4, \mathcal{P}_{r1}, \mathcal{P}_{53}\}$. The cell, $dpt(n, p)$, will contain the optimal solution to the optimization problem, HEPOPT.

size of an anti-diagonal L gives the number of cells in it. All the cells in an anti-diagonal are computed sequentially. The anti-diagonals L are visited serially in the order $3, \ldots, n+p-1$. The Pareto front for a table cell in an anti-diagonal L depends only on the Pareto fronts in the cells in the anti-diagonals preceding L as shown in Figure 1(c). Consider the anti-diagonal, L=5. It has five cells, {(5,1),(4,2),(3,3),(2,4),(1,5)}. The Pareto fronts for the cells, {(5,1),(1,5)}, are computed in the base initialization step. The values for the blue cells depend only on the Pareto fronts in the green cells in the anti-diagonals preceding L=5. For the cell (4,2), the possible workload distributions for the workload size 4 are $\{(1, 1), (3, 1), (0, p-2)\}$, $\{(2, 1), (2, 1), (0, p-2)\}$, $\{(4, 1), (0, p-1)\}$. The first workload distribution allocates workload size one to one node, three to one node, and zero to the rest. The second workload distribution allocates workload size two to one node, two to one node, and zero to the rest. Finally, the last workload distribution allocates workload size four to one node and zero to the rest. The computation of the Pareto front for the cell (4,2) will involve two invocations of the Pareto front combination routine, PoP. First, two Pareto fronts, $\mathcal{P}_{11}$ and $\mathcal{P}_{31}$, are combined using PoP to get a temporary Pareto front, $\mathcal{P}_{tmp1}$. Then, two Pareto fronts, $\mathcal{P}_{21}$ and $\mathcal{P}_{21}$, are combined using PoP to get a temporary Pareto front, $\mathcal{P}_{tmp2}$. The routine ConPar builds the optimal Pareto front, $\mathcal{P}_{42}$, for the cell (4,2), from the three input Pareto fronts, $\{\mathcal{P}_{tmp1}, \mathcal{P}_{tmp2}, \mathcal{P}_{41}\}$.

The routine PoP combining two or more Pareto fronts is described in the supplemental. Figure 1(d) illustrates how PoP combines two Pareto fronts to compute the Pareto front for the cell, (7, 5). For the anti-diagonal L=6, there are six cells, {(6,1),(5,2),(4,3),(3,4),(2,5),(1,6)}. The cells {(6,1),(1,6)} are computed in the base initialization step. The Pareto fronts for the rest of the cells are determined as described above by invoking the ConPar routine that takes as input the Pareto fronts in the cells in the anti-diagonals preceding L=6.

Figure 1(e) illustrates all the PoP invocations involved in determining the Pareto front for the cell, (r, c). The last recurrence relation illustrates how the Pareto front is computed for this cell using the routine, ConPar. There are *rmax* PoP invocations, PoP($\mathcal{P}_{11}$,$\mathcal{P}_{r-1\,c-1}$), PoP($\mathcal{P}_{21}$,$\mathcal{P}_{r-2\,c-1}$),..., PoP($\mathcal{P}_{rmax\,1}$,$\mathcal{P}_{r-rmax\,c-1}$) resulting in *rmax* Pareto fronts. These Pareto fronts, along with the Pareto front ($\mathcal{P}_{r\,c-1}$), are input to the ConPar invocation.

The last anti-diagonal visited, $L = n + p - 1$, contains the cell, $dpt(n, p)$. The Pareto front for this cell is determined similarly using the ConPar routine. The ConPar invocations take as inputs Pareto fronts resulting from *rmax* PoP invocations, PoP($\mathcal{P}_{11}$,$\mathcal{P}_{n-1\,p-1}$), ..., PoP($\mathcal{P}_{rmax\,1}$,$\mathcal{P}_{n-rmax\,p-1}$), and the Pareto front, ($\mathcal{P}_{n\,p-1}$). The cell $dpt(n, p)$ contains the output of HEPOPTADP, the Pareto front needed to solve the workload size n using the p identical nodes.

HEPOPTADP provides not only the solution for (n, p) but also for any $(I, J), \forall I = 1, \ldots, n, J = 1, \ldots, p$, through its *dpt* table. The time complexity of this algorithm is $\mathcal{O}(n \times p^2 \times m \times h \times \log_2(p \times m \times h))$. Therefore, it achieves a complexity reduction of $\mathcal{O}(\frac{m^2 \times h^2 \times p}{n})$ over HEPOPTA. Since $n \leq p \times m \times h$, the lower bound on the reduction in complexity will be $\mathcal{O}(m \times h)$. The reduction is due to the hierarchical design of HEPOPTADP and memoization in dynamic programming technique. The correctness and time complexity proofs of HEPOPTADP are given in the supplemental.

IV. PARHEPOPTA: PARALLEL DYNAMIC PROGRAMMING BI-OBJECTIVE OPTIMIZATION ALGORITHM

We describe here the parallel version of HEPOPTADP called PARHEPOPTA executed by q identical parallel MPI processes.

The inputs to PARHEPOPTA are the workload size, n; the number of identical nodes, p; the number of heterogeneous processors in each node, h; the number of identical MPI processes executing PARHEPOPTA, q; the sets T and E containing the discrete execution time and dynamic energy functions; the static power consumption of a node, P_s. All the inputs and outputs are assumed to be available at process 0 only.

The output is a Pareto front represented by a set of tuples, $\mathcal{P} = \{(f_E(D), f_T(D), D)_i\}, i \in [1 \,..\, K]$, where $D = [d_{ij}]_{p \times h}, i = 1, 2, \ldots, p, j = 1, 2, \ldots, h$ is the optimal workload distribution, $f_E(D)$ is the total energy, and $f_T(D)$ is the execution time corresponding to the optimal workload distribution. The solutions in the set $\mathcal{P}$ are output in increasing order of total energy.

The pseudocode of PARHEPOPTA is illustrated in the Algorithm 1.

The dynamic programming table *dpt* of dimensions $n \times p$ is partitioned vertically between q processes. We assume

Algorithm 1 Parallel Algorithm Solving the Bi-Objective Optimization of Workload Size n for Performance and Energy Employing q Parallel Processes

```
1: procedure PARHEPOPTA($n, p, h, q, T, E, m, P_s, \mathcal{P}$)
Input:
    Workload size, $n$
    Number of identical nodes, $p$
    Number of heterogeneous processors per node, $h$
    Number of identical MPI processes executing PARHEPOPTA, $q$
    Set of discrete execution time functions, $T = \{t_1(x), \ldots, t_h(x)\}$
    Set of discrete dynamic energy functions, $E = \{e_1(x), \ldots, e_h(x)\}$
    The cardinality of the discrete sets, $m$
    Static power consumption of a node, $P_s$
Output:
    Pareto front, $\mathcal{P} = \{(f_E(D), f_T(D), X)_k\}, k \in [1 \,..\, K]$
                $D = [d_{ij}]_{p \times h}, i \in [1 \,..\, p], j \in [1 \,..\, h]$

2:      $iam \leftarrow$ MPI_Comm_rank($MPI\_COMM\_WORLD$)
3:      if $p = 1$ and $q = 1$ then
4:          return HEPOPTA($n, h, T, E, P_s$)
5:      end if
6:      for $r \leftarrow 1, \frac{\min(n, m*h)}{q}$ do
7:          $w \leftarrow r + iam * \frac{\min(n, m*h)}{q}$
8:          $dpt(w, 1) \leftarrow$ HEPOPTA($w, h, T, E, P_s$)
9:      end for
10:     for $r \leftarrow 1, (\min(n, m*h)) \bmod q$ do
11:         if ($iam = (r - 1)$) then
12:             $w \leftarrow r + q * \frac{\min(n, m*h)}{q}$
13:             $dpt(w, 1) \leftarrow$ HEPOPTA($w, h, T, E, P_s$)
14:             break
15:         end if
16:     end for
17:     MPI_Allgatherv($dpt(*, 1)$)
18:     for $L \leftarrow 3, n + p - 1$ do
19:         $dpt \leftarrow$ computeL($iam, n, m, p, h, q, L, dpt$)
20:     end for
21:     if ($iam = (q - 1)$) then
22:         $MPI\_Send(dpt(n, \frac{p}{q}), \ldots, 0, \ldots)$
23:     end if
24:     if ($iam = 0$) then
25:         $MPI\_Recv(\mathcal{P}_{np}, \ldots, q - 1, \ldots)$
26:     end if
27:     return $\mathcal{P}_{np}$
28: end procedure
```

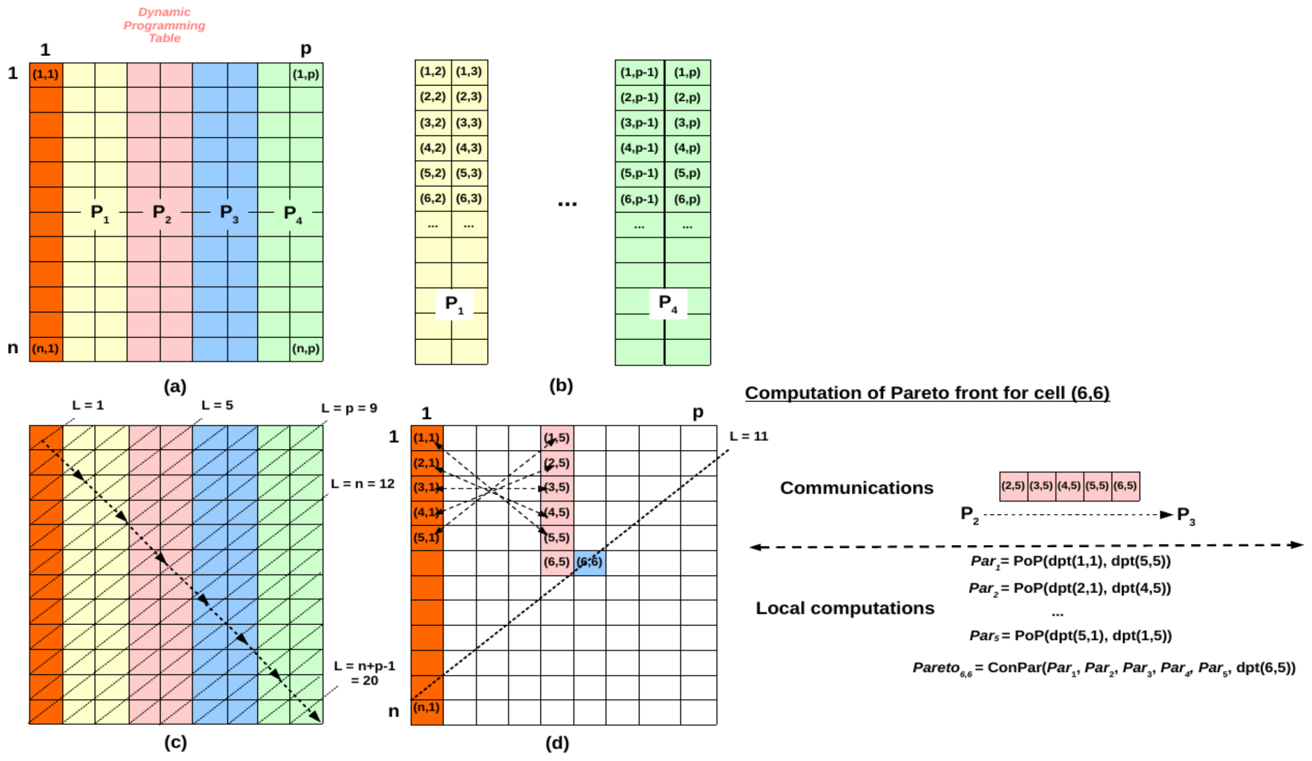

Dynamic Programming Table
(a)
(b)
(c)
(d)
P1
P2
P3
P4
L = 1
L = 5
L = p = 9
L = n = 12
L = n+p-1 = 20
L = 11
Computation of Pareto front for cell (6,6)
Communications
(2,5) (3,5) (4,5) (5,5) (6,5)
P2 → P3
Local computations
Par1 = PoP(dpt(1,1), dpt(5,5))
Par2 = PoP(dpt(2,1), dpt(4,5))
...
Par5 = PoP(dpt(5,1), dpt(1,5))
Pareto6,6 = ConPar(Par1, Par2, Par3, Par4, Par5, dpt(6,5))

FIGURE 2. The main steps in the execution of PARHEPOPTA employing q processes (4, in this case) to determine the Pareto front solving the workload size $n = 12$ using $p = 9$ identical nodes, each containing h heterogeneous processors. (a). The dynamic programming table's columns $(2, \ldots, p)$ are divided evenly between the q processes. (b). In the first main step, the Pareto fronts in the red cells, $\{(1,1),(2,1),\ldots,(m * h,1)\}$, are computed in parallel and replicated at all the q processes since they will be needed at all the main steps of PARHEPOPTA. (c). PARHEPOPTA then visits the anti-diagonals L serially in the order $3, \ldots, n+p-1$. For each anti-diagonal L, the Pareto fronts in its cells are computed in parallel by the q processes. There are no dependencies between the cells in the anti-diagonal. (d). Consider the computations for the cell (6, 6) in the anti-diagonal $L = n + 1 = 11$. The cell belongs to P_3. To compute the Pareto front in this cell, P_3 will need the Pareto fronts in the cells, $\{(1,1),\ldots,(6,1),(6,2),\ldots,(6,4),(1,5),\ldots,(6,5)\}$. P_3 contains locally the Pareto fronts in the cells, $\{(1,1),\ldots,(6,1)\}$. Note the cells in the first column $\{(1,1),\ldots,(6,1),\ldots,(m \times h,1)\}$ are replicated at all the processes. P_1 communicates the Pareto fronts in the cells, $\{(6,2),(6,3)\}$, to P_3. P_2 communicates the Pareto fronts in the cells, $\{(2,5),(3,5),(4,5),(5,5),(6,4),(6,5)\}$, to P_3. Then, P_3 performs local computations involving PoP invocations and a ConPar invocation to compute the Pareto front for cell (6,6).

p is divisible by q for aiding exposition. The first column is replicated at all the processes since its cell values are needed at each step of PARHEPOPTA. The rest of the columns $(1, \ldots, p-1)$ are distributed evenly between the q processes.

A vertical partitioning scheme is better for load balance than a horizontal one, where the n rows are divided equally between the q processes. Since the computation progresses diagonally in the *dpt* table, horizontal partitioning results in idling several processes and not doing any work. For example, $q-1$ processes (excluding process 0) will idle at the beginning of the computation, and $q-1$ processes (excluding process $q-1$) will idle at the end of the computation.

A process finds its rank, *iam*, on Line 2. Lines 3-5 cover the simple case when only one process executes PARHEPOPTA, $q = 1$, to solve the workload n using one node ($p = 1$) with h heterogeneous processors. PARHEPOPTA invokes HEPOPTA to obtain the Pareto front solving the workload using h heterogeneous processors. Lines 6-16 execute the base conditions (the top three recurrence relations). The table cells, $dpt(1, c), c = 1, \ldots, p$, store the Pareto front, $\mathcal{P}_{11}$, to solve the workload size one using one node.

The table cell, $dpt(r, 1), r \in [2 \,..\, m * h]$, is filled with the Pareto front, $\mathcal{P}_{r1}$, to solve the workload size r using one node. PARHEPOPTA invokes HEPOPTA to obtain this Pareto front solving the workload r using h heterogeneous processors. First, the Pareto fronts in the cells in the first column are determined in parallel by the q processes. Then, the column is replicated at all the processes using MPI_Allgatherv routine.

The core loop starts at Line 18. The loop variable, $L = 3, \ldots, n + p - 1$, represents the anti-diagonal. All the cells in an anti-diagonal are computed using the routine, *computeL*.

Algorithm 2 illustrates the routine, *computeL*. All the q processes take part in this routine. Since the table cells in an anti-diagonal $L = 3, \ldots, n + p - 1$ depend only on the cells in the anti-diagonals preceding L and have no inter-dependencies, they can be computed in parallel. The anti-diagonals are, however, visited sequentially, as shown in Figure 2(c). The number of cells in an anti-diagonal, *ncells*, is obtained using the routine, *getncells* (explained in the supplemental). There can only be a maximum of p cells in any anti-diagonal. The routine, *getmyncells*, returns *myncells* and *ncellspre*, which are the number of cells in an anti-diagonal

Algorithm 2 Routine Where the Processes Compute in Parallel All the Cells in the Anti-Diagonal L

```
1:  function computeL($iam, n, m, p, h, q, L, dpt$)
2:      $ncells \leftarrow$ getncells($L, n, p$)
3:      $(myncells, ncellspre) \leftarrow$ getmyncells(
            $iam, ncells, L, n, q, aggDptPart$)
4:      $myP \leftarrow aggDptPart[iam+1] - aggDptPart[iam]$
5:      $sRow \leftarrow (L < n)?(L-1) : n$
6:      $sCol \leftarrow (L \leq n)?2 : (L-n+1)$
7:      $lCol \leftarrow sCol + ncellsPre - 2 - aggDptPart[iam]$
8:      if $(iam \neq 0)$ and $(myncells! = 0)$
                and $(lCol = 0)$ then
9:          $lRow \leftarrow sRow - ncellsPre - 2$
10:         MPI_Recv($dptL(lRow), \ldots, iam-1, \ldots$)
11:     end if
12:     if $(iam \neq q-1)$ and
            $(ncellsPre + myncells < ncells)$ then
13:         $lRow \leftarrow sRow - (ncellsPre + myncells) - 2$
14:         $lCol \leftarrow myP - 1$
15:         MPI_Send($dptL(lRow, lCol)$,
                $iam+1$)
16:     end if
17:     for $v \leftarrow 1, myncells$ do
18:         $gRow \leftarrow sRow - ncellsPre - v$
19:         $gCol \leftarrow sCol + ncellsPre + v$
20:         $lRow \leftarrow gRow - 2$
21:         $lCol \leftarrow gCol - 2 - aggDptPart[iam]$
22:         $dpt(lRow, lCol) \leftarrow$
23:             computecell($iam, m, gRow, gCol$,
                    $lRow, lCol, dptL, dpt$)
24:     end for
25:     return dpt
26: end function
```

owned by the process *iam* and the number of cells that precede the cells belonging to the process *iam*. Both *getncells* and *getmyncells* are local routines.

Lines 7-16 in Algorithm 2 present the communications in PARHEPOPTA illustrated in Figure 2(d). Only one cell value is communicated between neighbouring processes *iam* and $iam + 1$ in each iteration of L. If the process $iam \neq 0$ is computing a cell in its first table column given by the condition $(lCol = 0)$, then it needs the values of the cells in the column preceding it from the neighbouring process, $iam - 1$. The buffer, *dptL*, stores these values with every increment of L. The neighbouring cell value is stored at the index, *lRow*. If the process $iam \neq (q-1)$ is not the last process owning a cell in the anti-diagonal L given by the condition $((ncellspre + myncells) < ncells)$, then it sends the cell value, $dpt(lRow, lCol)$, to the neighbouring process, $iam + 1$.

After completing the communications, the loop (Lines 17-24) contains the core computations. The local routine, *computecell* (Algorithm 3), computes the table cell value $dpt(lRow, lCol)$, which depends only on the cells,

Algorithm 3 Local Routine Computing $dpt(r, c)$ in the Anti-Diagonal L

```
1:  function computecell($iam, m, gRow, gCol$,
                $lRow, lCol, dptL, dpt$)
2:      for $row \leftarrow 1, gRow - 1$ do
3:          if $(row > m)$ and $(gCol = 2)$ then
4:              break
5:          end if
6:          if $(gRow - row) = 1$ then
7:              $dpt(lRow, lCol) \leftarrow$ PoP($dpt(row-1, 0)$,
                    $dpt(0, 0), dpt(lRow, lCol)$)
8:          else
9:              $dpt(lRow, lCol) \leftarrow$ PoP($dpt(row-1, 0)$,
                    $dptL(gRow - row - 1)$,
                    $dpt(lRow, lCol)$)
10:         end if
11:     end for
12:     if !$((gRow > m)$ and $(gCol = 2))$ then
13:         if !$((lCol = 0)$ and $(dptL \neq \phi))$ then
14:             $dpttmp \leftarrow dptL(gRow - 2)$
15:         else
16:             $dpttmp \leftarrow dpt(row - 2, lCol - 1)$
17:         end if
18:         $dpt(lRow, lCol) \leftarrow$ ConPar($dpttmp$,
                    $dpt(lRow, lCol)$)
19:     end if
20:     return $dpt(lRow, lCol)$
21: end function
```

$\{(row, 1), (gRow - row, lCol - 1)\}, \forall I \in [1\,..\,gRow - 1]$ and $(lRow, lCol - 1)$, as shown in Figure 1(e).

Lines 2-11 in Algorithm 3 contain the invocation of the PoP routine to combine the Pareto fronts in the cells, $(row, 1)$ and $(gRow - row, lCol - 1)$, as shown in Figure 2(c). Lines 12-19 invoke the ConPar routine to build the Pareto front, $dpt(lRow, lCol)$, from the input Pareto fronts, $\{dpt(lRow, lCol - 1), dpt(lRow, lCol)\}$.

Finally, in Lines 16-21 of the main algorithm 1, process $q - 1$ sends the cell value, $dpt(n, \frac{p}{q})$, to process 0. It contains the Pareto front, $\mathcal{P}_{np}$, having the optimal performance-energy application configurations (workload distributions) to solve the workload size n using the p identical nodes. $\mathcal{P}_{np}$ is the output of PARHEPOPTA.

PARHEPOPTA, through its *dpt* table, provides not only the solution for (n, p) but also for any $(I, J), \forall I = 1, \ldots, n, J = 1, \ldots, p$. It reduces the time complexity of HEPOPTADP by $\mathcal{O}(q)$. Therefore, it offers a potential speedup of $\mathcal{O}(\frac{m^2 \times h^2 \times p \times q}{n})$ over HEPOPTA. Since $n \leq p \times m \times h$, the lower bound for the reduction in complexity is $m \times h \times q$. The time and memory complexity proofs of PARHEPOPTA are given in the supplemental.

In the supplemental, we present the implementation of PARHEPOPTA employing hybrid parallelism employing q identical parallel MPI processes $(q \leq p)$, each executing t threads. There is no change in the theoretical time complexity between the hybrid and multi-process implementations. However, we experimentally found that the multi-process implementation performs better.

V. EXPERIMENTAL RESULTS AND DISCUSSION

This section first presents the experimental setup, the heterogeneous hybrid node and the data-parallel applications employed in the experiments. Then it describes the experimental methodology to construct the discrete execution time and the ground-truth dynamic energy profiles based on system-level physical power measurements using power meters for the processors involved in executing the two data-parallel applications. We then present the speedups of PARHEPOPTA and HEPOPTADP over HEPOPTA. Finally, we provide examples of usage of PARHEPOPTA and discuss scenarios where PARHEPOPTA can accelerate the development and runtime of energy-efficient parallel applications.

To ensure the reliability of our results, we follow a statistical methodology where a sample average for a response variable (energy, time, PMC, utilization variables) is obtained from multiple experimental runs. The sample average is calculated by executing the application repeatedly until it lies in the 95% confidence interval and a precision of 0.05 (5%) is 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. The methodology is described in the supplemental.

A. EXPERIMENTAL SETUP

The hybrid node used in our experiments is presented in the Introduction section. It consists of a dual-socket Intel Haswell multicore CPU containing 24 physical cores with 64 GB main memory, which hosts two accelerators, an Nvidia K40c GPU and an Nvidia P100 GPU (specifications in Table 1). Each accelerator connects to a dedicated host core via a separate PCI-E link. The static power consumption of the node is 253 W (P_s).

The hybrid node has one WattsUp Pro power meter between the wall A/C outlets and the node's input power sockets. The power meter captures the total power consumption of the node. It has a data cable connected to one USB port of the node. A Perl script collects the data from the power meter using the serial USB interface. The execution of these scripts is non-intrusive and consumes insignificant power.

The power meters are periodically calibrated using an ANSI C12.20 revenue-grade power meter, Yokogawa WT210. The maximum sampling speed of the power meters is one sample every second. The accuracy specified in the data sheets is $\pm 3\%$. The minimum measurable power is 0.5 watts. The accuracy at 0.5 watts is ± 0.3 watts.

We experiment with two popular and highly optimized scientific applications employing matrix multiplication (DGEMM) and 2D fast Fourier transform (2D-FFT) routines, respectively.

The DGEMM application computes $C+ = A \times B$, where A, B, and C are square matrices of size $N \times N$. The application employs Intel MKL DGEMM for the CPU and ZZGEMMOOC out-of-card package [22] for Nvidia GPUs. ZZGEMMOOC packages reuse CUBLAS and MKL BLAS for in-card DGEMM calls. The Intel MKL and CUDA versions are 2017.0.2 and 9.2.148. The workload size is equal to $2 \times N^3$.

The 2D-FFT application computes the 2D discrete Fourier Transform of a complex signal matrix of size $N \times N$. The workload size is $5 \times N^2 \times \log_2 N$. It employs Intel MKL FFT routines for the CPU and CUFFT routines for Nvidia GPUs. For workload sizes that cannot be factored into primes less than or equal to 127, CUFFT gives failures. Therefore, we employ out-of-card computations for these workload sizes where the workload size is divided into small sizes that can be solved on the GPU.

Only the Intel multicore CPU is employed for executing PARHEPOPTA because we do not have an implementation of PARHEPOPTA for Nvidia GPUs. The MPI used is Open MPI version 4.0.1.

B. METHODOLOGY TO CONSTRUCT DISCRETE EXECUTION TIME AND GROUND-TRUTH DYNAMIC ENERGY PROFILES

This section describes the methodology to construct the discrete execution time and ground-truth dynamic energy profiles for the processors executing the two data-parallel applications.

The data-parallel application on the hybrid node is modeled by three heterogeneous abstract processors, CPU_1, GPU_1, and GPU_2. The details are presented in the Introduction section. The execution time profiles of the abstract processors are experimentally built separately using an automated build procedure using three OpenMP threads where one thread is mapped to one abstract processor. The execution times of all the abstract processors executing the same workload are measured simultaneously, thereby considering the influence of resource contention. The execution time for abstract processors involving the accelerators includes the time to transfer data between the host and the accelerator. For example, the execution time and dynamic energy of the execution of a workload on GPU_1 include the time and energy to transfer data from the host multicore CPU core to the accelerator and back and the computations on the accelerator.

The ground-truth dynamic energy profiles of the abstract processors are constructed using the additive approach [15]. In the additive approach, the dynamic energy profiles of the three processors are constructed serially. The combined profile where the individual dynamic energy consumptions are totalled for each data point is then obtained. Then, the dynamic energy profile employing all the processors in parallel is built. The difference between the parallel and combined dynamic energy profiles is observed. We find that the average difference between parallel and combined dynamic energy profiles is around 5% for both applications and within the statistical accuracy threshold set in our experiments. Both the parallel and combined profiles also follow the same pattern. Therefore, we conclude that the processors in our experiments satisfy the additive hypothesis: the abstract processors are loosely coupled and therefore do not interfere with each other during the application. Thus, the ground-truth dynamic energy profiles of the three processors can be constructed serially or in parallel for our experimental platform and applications.

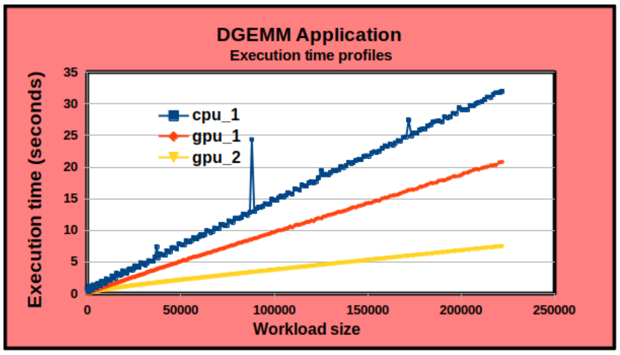
Figures 3a and 3b show the execution time and ground-truth dynamic energy profiles of the three processors for the DGEMM application obtained using the ground-truth method. The numbers of points (m) in the profiles for DGEMM and 2D-FFT applications are 210 and 256, respectively. Figures 3c and 3d display the execution time and ground-truth dynamic energy profiles of the three processors for the 2D-FFT application obtained using the ground-truth method.

In the figures, all the processors solve the same workload size for each data point in the discrete performance and energy profiles. For a given workload, PARHEPOPTA takes the discrete profiles as input and determines the optimal workload distribution to solve the workload. The workload distribution typically would contain different workload sizes assigned to the processors, but they are still members of the input discrete sets. The total execution time and energy of parallel execution of the workload is calculated to be the maximum of the execution times of the assigned workload sizes and summation of their respective energies. We can call them the model execution time and model energy. We confirm through exhaustive experimentation that the actual total execution time and total energy of parallel execution of the workload do not differ significantly from the model time and model energy.

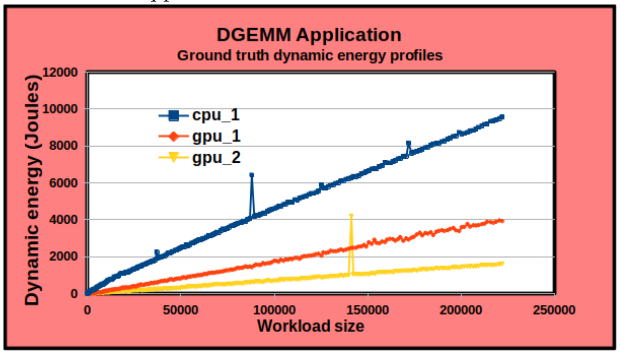
1) PRECAUTIONS TO PREVENT NOISE IN MEASURING ENERGY CONSUMPTION

The server is fully reserved and dedicated to the experiments during their execution. We also ensure that there are no drastic fluctuations in the load due to abnormal events in the server by monitoring its load continuously for a week using the tool, *sar*. Insignificant variation in the load is observed during this monitoring period suggesting normal and clean behaviour of the server.

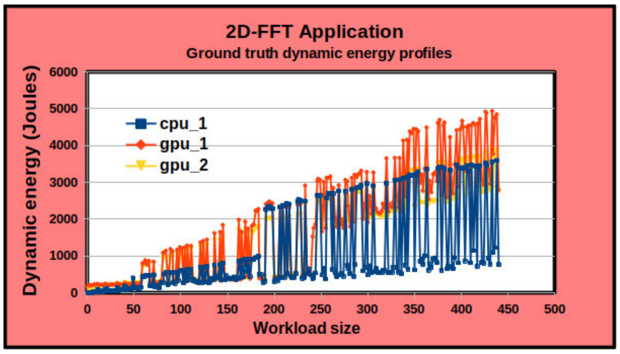
Several precautions are taken in computing energy measurements to eliminate any potential interference of the computing elements not part of the given abstract processor running the application kernel. First, we group abstract processors so that a given abstract processor constitutes solely the computing elements involved in running a given application kernel. The application kernel will, in this way, only use the computing elements of the abstract processor executing it and not use any other component for its execution. Hence, the dynamic energy consumption will solely reflect the work done by the computing elements



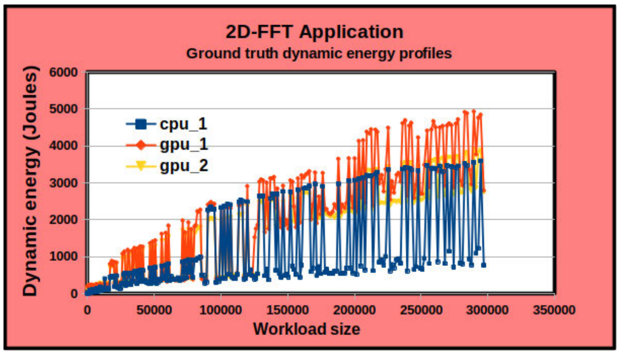
(a) Execution time profiles of the three processors employed in the DGEMM application.



(b) Dynamic energy profiles of the three processors employed in the DGEMM application.



(c) Execution time profiles of the three processors employed in the 2D FFT application.



(d) Dynamic energy profiles of the three processors employed in the 2D FFT application.

FIGURE 3. The execution time and ground-truth dynamic energy profiles of the three processors for the DGEMM and 2D-FFT applications.

of the given abstract processor executing the application kernel.

Consider the DGEMM application kernel executing on the abstract processor CPU_1 (comprised of CPU and DRAM). The *HCLWattsUp* API function gives the server's total energy consumption during an application's execution. Energy consumption includes the contribution from all components, such as NIC, SSDs, and fans. Therefore, to rule out their contribution to dynamic energy consumption, we ensure all the components other than CPUs and DRAM are not used during the execution of an application. In this way, the dynamic energy consumption we obtain using *HCLWattsUp* API reflects only the contribution of CPUs and DRAM. The following steps are employed for this purpose:

- The disk consumption is monitored before and during the application run and ensures no I/O is performed by the application using tools such as *sar* and *iotop*.
- The workload used in the execution of an application does not exceed the main memory, and swapping (paging) does not occur.
- The application does not use the network by monitoring using tools such as *sar* and *atop*.
- The application kernel's CPU affinity mask is set using SCHED API's system call, SCHED_SETAFFINITY(). To bind the DGEMM application kernel, we set its CPU affinity mask to 11 physical CPU cores of Socket 1 and 11 physical CPU cores of Socket 2.

Fans are also a great contributor to energy consumption. On our platform, fans are controlled in zones: a) zone 0: CPU or System fans, b) zone 1: Peripheral zone fans. There are four levels to control the speed of fans:

- Standard: BMC control of both fan zones, with the CPU zone based on CPU temp (target speed 50%) and Peripheral zone based on PCH temp (target speed 50%);
- Optimal: BMC control of the CPU zone (target speed 30%), with Peripheral zone fixed at low speed (fixed 30%);
- Heavy IO: BMC control of CPU zone (target speed 50%), Peripheral zone fixed at 75%;
- Full: all fans are running at 100%.

To rule out fans' contribution to dynamic energy consumption, we set the fans at full speed before launching the experiments. When set at full speed, the fans run consistently at $\sim$13400 rpm. In this way, fans consume the same amount of power, which is included in the static power of the server. Furthermore, we monitor the server's temperatures and the fans' speeds with the help of Intelligent Platform Management Interface (IPMI) sensors, both with and without the application run. We find no significant differences in temperature, and the speeds of fans are the same in both scenarios.

Thus, we ensure that the dynamic energy consumption measured reflects the contribution solely by the abstract processor executing the given application kernel.

C. SPEEDUPS OF PARHEPOPTA AND HEPOPTADP OVER HEPOPTA

This section compares the speedup of PARHEPOPTA and HEPOPTADP over HEPOPTA. All three algorithms are executed only on the Intel multicore CPU since we do not have their implementations for Nvidia GPUs.

HEPOPTADP and HEPOPTA are executed by only one process using one thread. PARHEPOPTA is executed by q MPI processes.

For DGEMM, the workload size n input to the algorithms is a multiple of the number of arithmetic operations in a basic computation unit, a matrix update, $a + b \times c$, where a, b, and c are $u \times u$ matrices of fixed size u. Therefore, the workload size n is a multiple of $2 \times u^3$. The value of u used in our experiments is 1536. For 2D FFT, the workload size n is a multiple of a basic computation unit, a 2D FFT of a signal matrix of size $v \times v$. The value of v used in our experiments is 512.

The input discrete dynamic energy profiles to all three algorithms are the ground-truth profiles constructed offline. The numbers of points (m) in the profiles for DGEMM and 2D-FFT applications are 210 and 256, respectively. The optimal workload distributions determined by all three algorithms contain workload sizes that are members of the input set X. Finally, we verify that the Pareto-optimal solutions returned by all algorithms match.

The speedup of PARHEPOPTA over HEPOPTA is calculated using the formula, $\frac{t_{hepopta}}{t_{parhepopta}}$, where $t_{hepopta}$ and $t_{parhepopta}$ are the execution times of HEPOPTA and PARHEPOPTA solving the same problem. The speedup of HEPOPTADP over HEPOPTA is calculated using the formula, $\frac{t_{hepopta}}{t_{hepoptadp}}$, where $t_{hepopta}$ and $t_{hepoptadp}$ are the execution times of HEPOPTA and HEPOPTADP solving the same problem.

The speedups for the DGEMM and 2D-FFT applications are shown in Tables 2 and 3, respectively. The upper bound on n that can be solved by the algorithms is equal to $m \times h \times p$. Therefore, the maximum $\frac{n}{p \times h}$ that can be solved is m, which is 211 and 256 for DGEMM and 2D-FFT applications, respectively.

This section compares the speedup of PARHEPOPTA and HEPOPTADP over HEPOPTA. All three algorithms are executed only on the Intel multicore CPU since we do not have their implementations for Nvidia GPUs.

HEPOPTADP and HEPOPTA are executed by only one process using one thread. q MPI processes execute PARHEPOPTA.

For DGEMM, the workload size n input to the algorithms is a multiple of the number of arithmetic operations in a basic computation unit, a matrix update, $a + b \times c$, where a, b, and c are $u \times u$ matrices of fixed size u. Therefore, the workload size n is a multiple of $2 \times u^3$. The value of u used in our experiments is 1536. For 2D FFT, the workload size n is a multiple of a basic computation unit, a 2D FFT of a signal matrix of size $v \times v$. The value of v used in our experiments is 512.

TABLE 2. Speedups of PARHEPOPTA and HEPOPTADP over HEPOPTA for the DGEMM application. n is a multiple of a basic computation unit, a matrix update of two 1536 × 1536 blocks. The other parameters are $h = 3$, $m = 211$, $P_s = 253$. PARHEPOPTA is executed by 24 parallel processes (one process per core of the multicore CPU).

$\frac{n}{p \times h}$	HEPOPTADP speedup	PARHEPOPTA speedup
64	4.65	30.69
128	4.60	30.25
184	4.77	31.24
210	4.55	29.58

TABLE 3. Speedup of PARHEPOPTA and HEPOPTADP over HEPOPTA for the 2D-FFT application. n is a multiple of a basic computation unit, a 2D FFT computation of 512 × 512 matrix. The other parameters are $h = 3$, $m = 256$, and $P_s = 253$. PARHEPOPTA is executed by 24 parallel processes (one process per core of the multicore CPU).

$\frac{n}{p \times h}$	HEPOPTADP speedup	PARHEPOPTA speedup
64	2.05	20.30
128	2.07	19.60
216	2.12	21.05
256	2.13	20.77

All three algorithms' input discrete dynamic energy profiles are the ground-truth profiles constructed offline. The numbers of points (m) in the profiles for DGEMM and 2D-FFT applications are 210 and 256, respectively. Furthermore, the optimal workload distributions determined by all three algorithms contain workload sizes that are members of the input set X. Finally, we verify that the Pareto-optimal solutions returned by all algorithms match.

The speedup of PARHEPOPTA over HEPOPTA is calculated using the formula, $\frac{t_{hepopta}}{t_{parhepopta}}$, where $t_{hepopta}$ and $t_{parhepopta}$ are the execution times of HEPOPTA and PARHEPOPTA solving the same problem. The speedup of HEPOPTADP over HEPOPTA is calculated using the formula, $\frac{t_{hepopta}}{t_{hepoptadp}}$, where $t_{hepopta}$ and $t_{hepoptadp}$ are the execution times of HEPOPTA and HEPOPTADP solving the same problem.

The speedups for the DGEMM and 2D-FFT applications are shown in Tables 2 and 3, respectively. The upper bound on n that the algorithms can solve is equal to $m \times h \times p$. Therefore, the maximum $\frac{n}{p \times h}$ that can be solved is m, 211 and 256 for DGEMM and 2D-FFT applications, respectively.

The quantity, $\frac{n}{p \times h}$, refers to the amount of work per processor or the granularity. The tables show that HEPOPTADP and PARHEPOPTA provide constant speedups over HEPOPTA, keeping the granularity constant, thereby confirming the theoretical results. The speedups of PARHEPOPTA over HEPOPTADP are about 7x and 10x for the two applications. However, the ideal speedup of 24x is unlikely owing to resource contention and (Non-uniform Memory Access) NUMA effects.

Table 4 shows the increase in the speedup of PARHEPOPTA over HEPOPTA as q is increased, keeping all the other application parameters constant, thereby confirming the theoretical result of $\mathcal{O}(q)$.

TABLE 4. Speedup of PARHEPOPTA over HEPOPTA for the DGEMM and 2D-FFT applications as q varies from 1 to 24 while keeping all the application parameters constant.

q	DGEMM	2D-FFT
1	4.60	2.13
2	7.76	4.08
4	14.74	8.17
8	16.16	15.9
16	26.9	18.11
24	30.25	20.77

D. APPLICATIONS OF PARHEPOPTA

We present examples of usage of PARHEPOPTA demonstrating how it can accelerate the development and runtime of energy-efficient parallel applications.

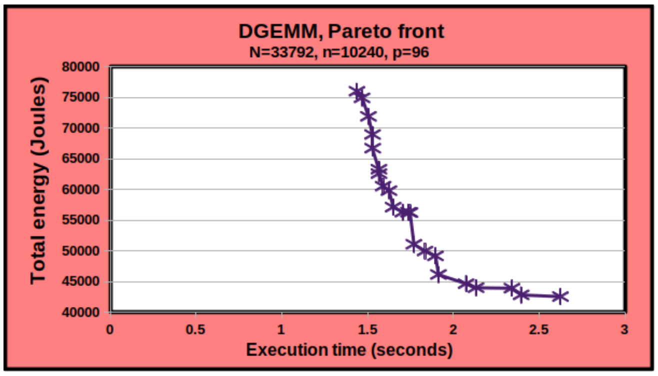
1) USAGE OF PARHEPOPTA

For a given application, the user provides the application configuration parameters, n, p, h, the number of MPI processes (q) executing PARHEPOPTA, the discrete performance and dynamic energy profiles (T, E) of cardinality m, the static energy consumption, P_s, and a criterion for selecting a Pareto-optimal solution (workload distribution) to be employed during the application execution. Some example criteria are a). select the performance-optimal solution, b). select the energy-optimal solution, and c). select a solution whose performance degradation is not more than 5% and has the lowest total energy. During the application execution, PARHEPOPTA is invoked using the parameters $(n, p, h, q, T, E, m, P_s)$ to determine the Pareto front $(\mathcal{P})$. A Pareto-optimal solution is then selected from the Pareto front using the user-specified criterion and employed for the computations.

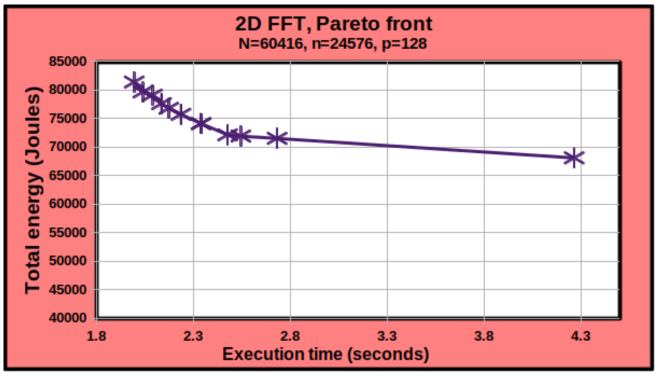
We illustrate two representative cases involving DGEMM and 2D-FFT applications. Consider, for example, the DGEMM application execution with input parameters $N = 33792, n = 10240, p = 96, h = 3, m = 211, q = 24, P_s = 253$, and the discrete performance and dynamic energy profiles (Figures 3a and 3b built offline). Figure 4a shows the Pareto front output by PARHEPOPTA. The execution time of PARHEPOPTA using one hybrid node is 19 seconds. Assuming all the 96 hybrid nodes are employed to execute PARHEPOPTA, the estimated execution time is 0.2 seconds due to potential $\mathcal{O}(p)$ speedup.

The endpoints of a Pareto front represent the optimal solutions for single-objective optimization for performance and energy. A steep slope close to the time-optimal endpoint means that allowing a slight degradation in performance can result in significant energy savings. Similarly, a steep slope close to the energy-optimal endpoint means that significant performance improvement is possible with a slight increase in energy consumption. In this case, a performance degradation of 0.34 seconds gives energy savings of 24918 J.

Due to the steepness of the Pareto front slope, the user would specify criteria that explore points further from the time-optimal endpoint down the Pareto front since the energy



(a) Pareto front for the DGEMM application for $(N = 33792, p = 96, h = 3, m = 211)$.



(b) Pareto front for the 2D-FFT application for $(N = 60416, p = 128, h = 3, m = 256)$.

FIGURE 4. The Pareto fronts for the DGEMM and 2D-FFT applications output by PARHEPOPTA using the ground-truth dynamic energy profiles of cardinality *m*. The matrix sizes employed in the applications is *N* × *N*. For DGEMM, the workload size, *n*, input to PARHEPOPTA is a multiple of a basic computation unit, a matrix update of 1536 × 1536 matrices and is equal to 10240. For 2D FFT, the workload size, *n*, input to PARHEPOPTA is 24576. The number of MPI processes (*q*) executing PARHEPOPTA is 24. The static power consumption P_s of the node is 253 W.

savings are pretty significant. Supposing the user specifies the selection of the performance-optimal solution. Then the workload distribution associated with the Pareto front point (1.44 [s], 75985 [J]) is employed to compute the DGEMM matrix multiplication. On the other hand, if the user wants a solution whose performance degradation is not more than 5% and has the lowest total energy, then the workload distribution associated with the Pareto front point (1.50 [s], 71890 [J]) is used to compute the DGEMM matrix multiplication. The selected point is the third point from the time-optimal endpoint. Allowing 1% further performance degradation can provide 5180 J energy savings due to the steepness of the Pareto front slope.

Similarly, consider the 2D-FFT application execution with input parameters, $N = 60416, n = 24576, p = 128, h = 3, m = 256, q = 24, P_s = 253$, and the discrete performance and dynamic energy profiles (Figures 3c and 3d built offline). Figure 4b shows the Pareto front output by PARHEPOPTA. The execution time of PARHEPOPTA using one hybrid node is 61 seconds. Assuming all the 128 hybrid nodes are employed to execute PARHEPOPTA, the estimated execution time is 0.47 seconds due to potential $\mathcal{O}(p)$ speedup.

The 2D-FFT Pareto front is less steep than the DGEMM Pareto front and more or less horizontal. While allowing a performance degradation of 0.34 seconds gives energy savings of 24918 J for DGEMM, the energy savings for the same performance degradation is 7286 J for 2D-FFT. Therefore, the user-specific criteria for 2D-FFT will differ from those for DGEMM.

Since the 2D-FFT slope is almost horizontal, the user would specify criteria that focus on points closer to the time-optimal endpoint since venturing further down the Pareto front will not provide substantial energy savings. Supposing the user wants a solution whose performance degradation is not more than 8% and has the lowest total energy. Then the workload distribution associated with the Pareto front point (2.17 [s], 76794 [J]) is used to compute the 2D fast Fourier transform. The selected point is the fifth point from the time-optimal endpoint. Allowing 6% further performance degradation can only yield 2685 J energy savings.

2) ACCELERATION OF DEVELOPMENT AND RUNTIME OF ENERGY-EFFICIENT APPLICATIONS

The development of an energy-efficient parallel application is typically done on a small subset of nodes in a cluster before the application is deployed in production. Therefore, the critical building blocks of such applications, such as the data-partitioning algorithm, must not slow down the development effort.

Consider, for example, the development of DGEMM and 2D-FFT applications described previously using two test configurations, $(N = 33792, n = 10240, p = 96, h = 3, m = 211, P_s = 253)$ for DGEMM and $(N = 60416, n = 24576, p = 128, h = 3, m = 256, P_s = 253)$ for 2D-FFT. The discrete performance and dynamic energy profiles employed for DGEMM are shown in Figures 3a and 3b and for 2D-FFT in Figures 3c and 3d. For these application configurations, the execution times of HEPOPTA are 3077 seconds and 11616 seconds for DGEMM and 2D-FFT, respectively. Therefore, the long execution times of HEPOPTA make it unfit for developing such energy-efficient parallel applications. However, the good speedups offered by PARHEPOPTA render it practicable.

PARHEPOPTA is ideal for accelerating long-running simulations like numerical weather prediction. For example, consider the real-life scientific application, Multidimensional Positive Definite Advection Transport Algorithm (MPDATA), which is a core component of the EULAG (Eulerian/semi-Lagrangian fluid solver) geophysical model developed for numerical weather prediction [10]. The simulation runs for over 16000 time steps for a 2-day prediction where each time step takes 0.125 seconds on an Intel Xeon Phi coprocessor. Therefore, the total simulation time is 2000 seconds.

The authors improve the performance of the simulation by 15% using a model-based data partitioning algorithm. In the improved version, the simulation comprises initialization and solution phases. In the initialization phase, the performance profiles are built and input to the data partitioning algorithm, which determines the performance-optimal heterogeneous workload distribution. The solution phase employs the same workload distribution to perform the simulation in all the time steps. The initialization phase takes less than 20 time steps. Since MPDATA computation in each time step is similar to matrix multiplication, we consider accelerating such simulations on a moderate-sized cluster of hybrid nodes, for example, $p = 96$. Using HEPOPTA can be prohibitively expensive in this case since its execution time of 3077 [s] exceeds the simulation time. However, the overhead of PARHEPOPTA is less than two time steps, and its execution time of 0.2 [s] is only 0.01% of the simulation time.

VI. RELATED WORK

We divide our related literature study into five categories: a). System-level optimization methods, b). Application-level optimization methods, c). Static and dynamic optimization methods, d). Performance models of computation, and e). Energy models of computation.

A. SYSTEM-LEVEL METHODS

System-level 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). DVFS reduces the dynamic power consumed by a processor by throttling its clock frequency.

Freeh et al. [23] analyze the performance-energy tradeoffs of serial and parallel applications on a cluster of DVFS-capable AMD nodes. The decision variable is the DVFS frequency. Rountree et al. [24] propose a runtime system that employs DVS during predicted slack periods to achieve significant energy savings while incurring negligible performance degradation. Lee and Zomaya [25] propose energy-conscious scheduling heuristics that employ DVS for bi-objective optimization of parallel applications on HPC platforms.

Mezmaz et al. [1] propose a parallel genetic algorithm for bi-objective optimization on cloud computing infrastructures for performance and energy. The decision variable is the supply voltage of the processor. Fard et al. [26] present a four-objective case study comprising performance, economic cost, energy consumption, and reliability for optimization of scientific workflows in heterogeneous computing environments. The decision variable is the task assignment or mapping. Beloglazov et al. [27] 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 virtual machines and clock frequencies.

Kessaci et al. [5] present a multi-objective genetic algorithm that minimizes energy consumption, CO2 emissions and maximizes the generated profit of a cloud computing infrastructure. The decision variable is the arrival rate of applications. Durillo et al. [6] propose a multi-objective workflow scheduling algorithm for bi-objective optimization on heterogeneous high-performance parallel and distributed computing systems for performance and energy. They study the impact of several decision variables: number of tasks, number of machines, DVFS levels, static energy, and types of tasks. Kolodziej et al. [2] propose multi-objective genetic algorithms for bi-objective optimization on green grid clusters and clouds for performance and energy. The performance is modeled using the computation speed of a processor. The decision variable is the DVFS level.

Solution methods can also be differentiated based on whether they output a partial or a full Pareto front of performance-energy optimal solutions. Some solution methods optimize HPC platforms for performance under an energy budget or optimize for energy under an execution time constraint [7], [8], [9]. They determine a partial Pareto front by applying the power cap or an execution time constraint and then select the best configuration to fulfil a user-specific criterion. Some methods solve unconstrained bi-objective optimization for performance and energy (with no time or energy constraints) [5], [6], [2]. They build the full Pareto front. Research works [28], [29], [30] are analytical studies of bi-objective optimization for performance and energy.

B. APPLICATION-LEVEL METHODS

The second approach consists of solution methods that optimize applications rather than the operating environment. The methods use application-level decision variables and predictive models for the performance and energy consumption of applications. The dominant decision variables include the number of threads, loop tile size, and workload distribution. This approach is understudied compared to the mainstream system-level approach.

Lastovetsky and Reddy [11] propose data partitioning algorithms that solve single-objective optimization problems of data-parallel applications for performance or energy on homogeneous clusters of multicore CPUs. They take as an input discrete performance and dynamic energy functions with no shape assumptions that accurately and realistically account for resource contention and NUMA inherent in modern multicore CPU platforms. Reddy and Lastovetsky [12] propose a solution method to solve the bi-objective optimization problem of an application for performance and energy on homogeneous clusters of modern multicore CPUs. They demonstrate that the method gives a diverse set of Pareto-optimal solutions and can be combined with DVFS-based multi-objective optimization methods to give a better set of (Pareto-optimal) solutions. The methods target homogeneous high-performance computing (HPC) platforms.

Modern HPC platforms, cloud computing systems, and data centers are highly heterogeneous, comprising nodes where a multicore processor is tightly integrated with one or more accelerators to address the twin critical concerns of performance and energy efficiency. A crucial requirement to determine the energy-optimal configuration of a parallel/hybrid application executing on such platforms is to determine component-level dynamic energy profiles of the application executing on multiple independent devices of the platform. Fahad et al. [17] present the first methodology to measure the component-level energy consumption of a hybrid application on a heterogeneous computing platform based on system-level power measurements using power meters. Chakraborti et al. [13] consider the effect of heterogeneous workload distribution on bi-objective optimization of data analytics applications by simulating heterogeneity on homogeneous clusters. A linear function of workload size represents the performance, and the energy is predicted using historical data tables. Khaleghzadeh et al. [15] propose a solution for solving the bi-objective optimization problem on heterogeneous processors comprising two principal components. The solution method employs the methodology [17] to determine the fine-grained component-level decomposition of an application's energy consumption.

Architects of modern multicore processors follow a key design goal called energy proportionality (EP) (Barroso and Hölzle [31]), which means designing microprocessors that consume energy proportional to the amount of work performed. Khokhriakhov et al. [14] demonstrate that EP does not hold for modern multicore processors using a novel application-level bi-objective optimization method for energy and performance on a single multicore processor. They experiment with four popular and highly optimized multithreaded data-parallel applications on four modern multicore processors. They show that optimizing for performance alone may result in a significant increase in dynamic energy consumption, and optimizing for dynamic energy alone – in considerable performance degradation. Furthermore, their optimization method determined a good number of Pareto-optimal solutions.

C. STATIC AND DYNAMIC OPTIMIZATION METHODS

Static optimization solution methods use *a priori* information about the application and platform. We will focus mainly on workload partitioning methods. The solution methods can afford to construct full and comprehensive performance and energy profiles offline, allowing them to explore a large space of solutions. They are typically used in applications where data locality is important because they do not require data redistribution. The methods [32], [33], [34] solve the single-objective optimization problem for performance on heterogeneous platforms. The methods [11], [12], [15] solve the bi-objective optimization problem for performance and energy for homogeneous and heterogeneous platforms. They take input performance and energy profiles that are built offline. The energy profiles are constructed using the system-level physical power measurements using external power meters. However, the methods are not suitable for two scenarios: a). Dynamic environments where the number of available processors and their performance and energy profiles can differ for different application runs, and b). Environments with no provision for node-level physical power measurements using external power meters, for example, supercomputing facilities, computational grids, and cloud computing infrastructures.

Dynamic optimization solution methods adapt at runtime to dynamic changes in the environment. Runtime schedulers such as KAAPI [35], StarPU [36], and DAGuE [37] schedule an application described as a Direct Acyclic Graph (DAG) or task graph onto parallel platforms. Task scheduling and work-stealing algorithms [38], [39], [40], balance the load by moving fine-grained tasks between processors during the execution. Dynamic load balancers based on graph partitioners are proposed by [41], [42] for adaptive scientific computations where two objectives, interprocessor communication and data migration costs, are considered. Lastovetsky et al. [43] propose a data partitioning algorithm for employment in self-adaptable applications due to its low runtime cost. The methods above solve the single-objective optimization problem for performance. Reddy et al. [44] propose a dynamic data partitioning algorithm which solves the bi-objective optimization problem for performance and energy on homogeneous clusters of multicore CPUs. The algorithm constructs partial performance and energy profiles at runtime and assumes that the nodes are equipped with power meters providing power measurements. Therefore, it is unsuitable for employment in dynamic environments where nodes lack power meters.

D. PERFORMANCE MODELS OF COMPUTATION

Performance models of computations can be classified into analytical and non-analytical categories.

Analytical models use techniques such as linear regression, analyzing patterns of computation and memory accesses, and static code analysis to estimate performance for CPUs and accelerators [45], [46]. In the non-analytical category, the most simple model is a constant performance model (CPM), where different notions such as normalized cycle time, normalized processor speed, average execution time, and task computation time. characterize the speed of an application [47], [48]. In CPMs, no dependence is assumed between the performance of a processor and the workload size.

CPMs are too simplistic to accurately model the performance of data-parallel applications executing on modern heterogeneous platforms. The most advanced load balancing algorithms employ functional performance models (FPMs) that are application-specific and represent the speed of a processor by a continuous function of workload size [34], [49]. The FPMs capture realistically and accurately the real-life behaviour of applications executing on nodes consisting of uniprocessors (single-core CPUs).

The complex nodal architecture of modern HPC systems, consisting of tightly integrated processors with inherent severe resource contention and NUMA, pose serious challenges to load balancing algorithms based on the FPMs. These inherent traits result in significant variations (drops) in the performance profiles of parallel applications executing on these platforms, thereby violating the assumptions on the shapes of the performance profiles considered by the FPM-based load balancing algorithms. For example, in [10], [11], [12], [15], the performance model of computations is represented by a complex (non-smooth and non-linear) function of workload size.

E. ENERGY MODELS OF COMPUTATION

A linear energy model based on the utilization of CPU, disk, and network is proposed by Heath et al. [50]. A more complex power model proposed in [51] employs utilization metrics of CPU, disk, and network components and hardware performance counters for memory as predictor variables. Fan et al. [52] propose a simple linear model that correlates the power consumption of a single-core processor with its utilization. Bertran et al. [53] present a power model that provides a per-component power breakdown of a multicore CPU. Their model is based on activity factors obtained from PMCs for various components in a multicore CPU. Basmadjian et al. [54] construct a power model of a server using the summation of power models of its components: the processor (CPU), memory (RAM), fans, and disk (HDD). Lastovetsky and Reddy [11] propose a model representing a multicore CPU's energy consumption by a non-linear workload size function.

Hong and Kim [55] present an energy model for an Nvidia GPU based on a PMC-based power prediction approach similar to [56]. Nagasaka et al. [57] propose a PMC-based statistical power consumption modelling technique for GPUs that run CUDA applications. Song and Brooks [58] present power and energy prediction models based on machine learning algorithms such as backpropagation in artificial neural networks (ANNs). Shao et al. [59] develop an instruction-level energy consumption model for a Xeon Phi processor.

Shahid et al. [20] propose a novel theory of energy predictive models of computing and unify its practical implications to increase the prediction accuracy of linear energy predictive models in a *consistency* test. The test contains a suite of properties that include determinism, reproducibility, and additivity to select model variables and constraints for model coefficients. By applying the consistency test, the authors improve the average prediction accuracy of state-of-the-art linear regression models from 31% to 18%. Shahid et al. [21] analyze the prediction accuracy of models employing utilization variables only, PMCs only, and a combination of both utilization variables and PMCs, through the lens of the theory of energy predictive models of computing for modern multicore CPU platforms. They demonstrate that application-specific and platform-level models using both utilization variables and PMCs exhibit up to 3.6x and 2.6x better average prediction accuracy, respectively, when compared with models employing utilization variables only and highly additive PMCs only.

Khokhriakhov et al. [14] propose a qualitative linear dynamic energy model employing CPU utilization and PMCs to explain the discovered energy nonproportionality on their multicore CPU platforms.

VII. CONCLUSION

Achieving energy efficiency objectives and satisfying quality-of-service requirements related to response time are critical concerns in modern high performance computing platforms, computational grids, and cloud computing infrastructures. Therefore, accelerating the bi-objective optimization of applications for performance and energy on such platforms is necessary to address these concerns.

In this work, we presented an overview of the model-based methods proposed recently for bi-objective optimization of data-parallel applications on modern HPC platforms for performance and energy. The methods use workload distribution as the decision variable. However, the methods are sequential and exhibit exorbitant execution times for even moderate input values of the number of available processors. Based on our overview, we highlighted two fundamental challenges to accelerating the methods: (a). Fast computation of Pareto-optimal solutions optimizing the application for performance and energy, and (b). Fast construction of performance and energy profiles that are discrete functions of workload size.

We then formulated the bi-objective optimization problem of data-parallel applications for performance and energy on a cluster of p identical hybrid nodes, each containing h heterogeneous processors. The problem employs workload distribution as the decision variable. We proposed two algorithms solving the problem. They address the first challenge, the fast computation of Pareto-optimal solutions. The first algorithm is an exact sequential algorithm that is more efficient and amenable to parallelization. It achieves a complexity reduction over the state-of-the-art sequential algorithm with the lower bound of $\mathcal{O}(m \times h)$ where m is the cardinality of the input discrete execution time and dynamic energy functions. Furthermore, it returns the Pareto-optimal set of workload distributions, minimizing the execution time and the total energy consumption of computations during the parallel execution of the application.

The second algorithm is a parallel algorithm executed by q identical parallel processes that reduces the complexity of our proposed sequential algorithm by $\mathcal{O}(q)$. It, therefore, achieves a complexity reduction with the lower bound of $\mathcal{O}(m \times h \times q)$ over the state-of-the-art sequential algorithm.

We experimentally studied the practical efficacy of our algorithms for two data-parallel applications, matrix multiplication and fast Fourier transform, on a state-of-the-art heterogeneous hybrid node containing an Intel Haswell multicore CPU, an Nvidia k40c GPU, and an Nvidia P100 GPU and simulations of clusters of such hybrid nodes. The experiments demonstrated that our proposed algorithms provide tremendous speedups over the state-of-the-art sequential algorithm.

The software implementations of the algorithms proposed in this paper can be downloaded from the URL [60].

REFERENCES

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

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

[3] R. Lucas. (Feb. 2014). *DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges*. [Online]. Available: https://www.osti.gov/biblio/1222713

[4] F. D. Rossi, M. G. Xavier, C. A. F. De Rose, R. N. Calheiros, and R. Buyya, "E-eco: Performance-aware energy-efficient cloud data center orchestration," *J. Netw. Comput. Appl.*, vol. 78, pp. 83–96, Jan. 2017.

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

[6] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Gener. Comput. Syst.*, vol. 36, pp. 221–236, Jul. 2014.

[7] L. Yu, Z. Zhou, S. Wallace, M. E. Papka, and Z. Lan, "Quantitative modeling of power performance tradeoffs on extreme scale systems," *J. Parallel Distrib. Comput.*, vol. 84, pp. 1–14, Oct. 2015.

[8] N. Gholkar, F. Mueller, and B. Rountree, "Power tuning HPC jobs on power-constrained systems," in *Proc. Int. Conf. Parallel Architectures Compilation*, Sep. 2016, pp. 179–191.

[9] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 2007, pp. 1–9.

[10] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 787–797, Mar. 2017.

[11] A. Lastovetsky and R. R. Manumachu, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1119–1133, Apr. 2017.

[12] R. R. Manumachu and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 160–177, Feb. 2018.

[13] A. Chakrabarti, S. Parthasarathy, and C. Stewart, "A Pareto framework for data analytics on heterogeneous systems: Implications for green energy usage and performance," in *Proc. 46th Int. Conf. Parallel Process. (ICPP)*, Aug. 2017, pp. 533–542.

[14] S. Khokhriakov, R. R. Manumachu, and A. Lastovetsky, "Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and performance," *Appl. Energy*, vol. 268, Jun. 2020, Art. no. 114957.

[15] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 543–560, Mar. 2021.

[16] R. R. Manumachu and A. Lastovetsky, "Parallel data partitioning algorithms for optimization of data-parallel applications on modern extreme-scale multicore platforms for performance and energy," *IEEE Access*, vol. 6, pp. 69075–69106, 2018.

[17] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Accurate energy modelling of hybrid parallel applications on modern heterogeneous computing platforms using system-level measurements," *IEEE Access*, vol. 8, pp. 93793–93829, 2020.

[18] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on multicore and multi-GPU platforms using functional performance models," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2506–2518, Sep. 2015.

[19] M. Fahad, A. Shahid, R. Manumachu, and A. Lastovetsky, "A comparative study of methods for measurement of energy of computing," *Sci. Found. Ireland*, vol. 12, p. 2204, Jan. 2019.

[20] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Energy predictive models of computing: Theory, practical implications and experimental analysis on multicore processors," *IEEE Access*, vol. 9, pp. 63149–63172, 2021.

[21] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Improving the accuracy of energy predictive models for multicore CPUs by combining utilization and performance events model variables," *J. Parallel Distrib. Comput.*, vol. 151, pp. 38–51, May 2021.

[22] H. Khaleghzadeh, Z. Zhong, R. Reddy, and A. Lastovetsky, "Out-of-core implementation for accelerator kernels on heterogeneous clouds," *J. Supercomput.*, vol. 74, no. 2, pp. 551–568, 2018.

[23] 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, pp. 835–848, Jun. 2007.

[24] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS practical for complex HPC applications," in *Proc. 23rd Int. Conf. Supercomput.*, Jun. 2009, pp. 460–469.

[25] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[26] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2012, pp. 300–309.

[27] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generat. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[28] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz, "Perfect strong scaling using no additional energy," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, May 2013, pp. 649–660.

[29] M. Drozdowski, J. M. Marszałkowski, and J. Marszałkowski, "Energy trade-offs analysis using equal-energy maps," *Future Gener. Comput. Syst.*, vol. 36, pp. 311–321, Jul. 2014.

[30] J. M. Marszałkowski, M. Drozdowski, and J. Marszałkowski, "Time and energy performance of parallel systems with hierarchical memory," *J. Grid Comput.*, vol. 14, no. 1, pp. 153–170, Mar. 2016.

[31] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[32] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka, "An efficient, model-based CPU-GPU heterogeneous FFT library," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 1–10.

[33] C. Yang, F. Wang, Y. Du, J. Chen, J. Liu, H. Yi, and K. Lu, "Adaptive optimization for petascale heterogeneous CPU/GPU computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2010, pp. 19–28.

[34] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 1, pp. 76–90, 2007.

[35] T. Gautier, X. Besseron, and L. Pigeon, "KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors," in *Proc. Int. Workshop Parallel Symbolic Comput.*, Jul. 2007, pp. 15–23.

[36] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Comput., Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011.

[37] G. Bosilca, A. Bouteiller, A. Danalis, T. Herault, P. Lemarinier, and J. Dongarra, "DAGuE: A generic distributed DAG engine for high performance computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 1151–1158.

[38] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: A programming model for heterogeneous multi-core systems," in *Proc. 13th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2008, pp. 287–296.
[39] G. Quintana-Ortí, F. D. Igual, E. S. Quintana-Ortí, and R. A. van de Geijn, "Solving dense linear systems on platforms with multiple hardware accelerators," *ACM SIGPLAN Notices*, vol. 44, no. 4, pp. 121–130, Feb. 2009.
[40] C. Augonnet, S. Thibault, and R. Namyst, "Automatic calibration of performance models on heterogeneous multicore architectures," in *Proc. 3rd Workshop Highly Parallel Process. Chip (HPPC)*, Aug. 2009, pp. 56–65.
[41] K. Schloegel, G. Karypis, and V. Kumar, "A unified algorithm for load-balancing adaptive scientific simulations," in *Proc. ACM/IEEE SC Conf. (SC)*, Nov. 2000, p. 59.
[42] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. Heaphy, and L. A. Riesen, "Hypergraph-based dynamic load balancing for adaptive scientific computations," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, pp. 1–11.
[43] D. Clarke, A. Lastovetsky, and V. Rychkov, "Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms," *Parallel Process. Lett.*, vol. 21, no. 2, pp. 195–217, 2011.
[44] R. Reddy Manumachu and A. L. Lastovetsky, "Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 4, Feb. 2019, Art. no. e4958.
[45] K.-H. Kim, K. Kim, and Q.-H. Park, "Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model," *Comput. Phys. Commun.*, vol. 182, no. 6, pp. 1201–1207, Jun. 2011.
[46] J. Shen, A. L. Varbanescu, Y. Lu, P. Zou, and H. Sips, "Workload partitioning for accelerating applications on heterogeneous platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2766–2780, Sep. 2016.
[47] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *J. Parallel Distrib. Comput.*, vol. 61, no. 4, pp. 520–535, Apr. 2001.
[48] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1033–1051, Oct. 2001.
[49] A. Lastovetsky and R. Reddy, "Data partitioning for multiprocessors with memory heterogeneity and memory constraints," *Sci. Program.*, vol. 13, no. 2, pp. 93–112, 2005.
[50] T. Heath, B. Diniz, E. V. Carrera, W. Meira, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proc. 10th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Jun. 2005, pp. 186–195.
[51] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Workshop Modeling, Benchmarking, Simulation*, 2006, pp. 70–77.
[52] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, Jun. 2007, pp. 13–23.
[53] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proc. 24th ACM Int. Conf. Supercomput.*, Jun. 2010, pp. 147–158.
[54] R. Basmadjian, N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani, "A methodology to predict the power consumption of servers in data centres," in *Proc. 2nd Int. Conf. Energy-Efficient Comput. Netw.*, May 2011, pp. 1–10.
[55] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010, pp. 280–289.
[56] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. 22nd Digit. Avionics Syst. Conf.*, Dec. 2003, p. 93.
[57] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 115–122.
[58] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, May 2013, pp. 673–686.
[59] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of Intel's Xeon phi processor," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Sep. 2013, pp. 389–394.
[60] R. R. Manumachu and A. Lastovetsky. (2022). *PAREPOPT: Dynamic Performance-Energy Optimization of Data-Parallel Applications on Hybrid Nodes Through Workload Distribution*. [Online]. Available: https://csgitlab.ucd.ie/manumachu/parepopt

RAVI REDDY MANUMACHU (Member, IEEE) received the B.Tech. degree from IIT Madras, in 1997, and the Ph.D. degree from the School of Computer Science, University College Dublin, in 2005. His main research interests include high-performance heterogeneous computing and energy-efficient computing.

HAMIDREZA KHALEGHZADEH received the B.Sc. and M.Sc. degrees in computer engineering (software) and the Ph.D. degree from the School of Computer Science, University College Dublin, in 2007, 2011, and 2019, respectively. He is currently a Lecturer with the School of Computing, University of Portsmouth. His research interests include performance and energy consumption optimization in massively heterogeneous systems, high-performance heterogeneous systems, energy efficiency, and parallel/distributed computing.

ALEXEY LASTOVETSKY (Member, IEEE) received the Ph.D. degree from the Moscow Aviation Institute, in 1986, and the D.Sc. degree from the Russian Academy of Sciences, in 1997. He is currently an Associate Professor with the School of Computer Science, University College Dublin (UCD). At UCD, he is also the founding Director of the Heterogeneous Computing Laboratory. He has authored the monographs of *Parallel Computing on Heterogeneous Networks* (Wiley, 2003) and *High Performance Heterogeneous Computing* (Wiley, 2009). His main research interests include high-performance heterogeneous computing and energy-efficient computing.