Received: 14 December 2017 | Revised: 23 July 2018 | Accepted: 28 July 2018
DOI: 10.1002/cpe.4958

WILEY

RESEARCH ARTICLE

Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution

Ravi Reddy Manumachu | **Alexey L. Lastovetsky**

School of Computer Science, University College Dublin, Dublin, Ireland

Correspondence
Ravi Reddy Manumachu, School of Computer Science, University College Dublin, Dublin D04 V1W8, Ireland.
Email: ravi.manumachu@ucd.ie

Funding information
Science Foundation Ireland, Grant/Award Number: 14/IA/2474

Summary

Self-adaptability is a highly preferred feature in HPC applications. A crucial building block of a self-adaptable application is a data partitioning algorithm that must possess several essential qualities apart from low runtime and memory costs. On modern platforms composed of multicore CPU processors, data partitioning algorithms striving to solve the bi-objective optimization problem for performance and energy (*BOPPE*) face a formidable challenge. They must take into account the new complexities inherent in these platforms such as severe resource contention and non-uniform memory access (NUMA). Novel model-based methods and data partitioning algorithms have been proposed that address the challenge. However, these methods take as input full functional performance and energy models (FPM and FEM), which have prohibitively high model construction costs. Therefore, they are not suitable for employment in self-adaptable applications. In this paper, we present a self-adaptable data partitioning algorithm called *ADAPTALEPH*, which solves *BOPPE* on homogeneous clusters of multicore CPUs. Unlike the state-of-the-art solving *BOPPE* that take as inputs full FPM and FEM, it constructs partial FPM and FEM during its execution using all the available processors. It returns a locally Pareto-optimal set of solutions, which are the heterogeneous workload distributions that achieve inter-node optimization of data-parallel applications for performance and energy. We experimentally study the efficiency of *ADAPTALEPH* for three data-parallel applications, ie, matrix-vector multiplication, matrix-matrix multiplication, and fast Fourier transform, on a modern multicore CPU and simulations for homogeneous clusters of such CPUs. We demonstrate that the locally Pareto-optimal front approaches the globally Pareto-optimal front as the number of points in the partial discrete FPM and FEM functions are increased. The number of points in the partial FPM/FEM when the locally Pareto-optimal front becomes the globally Pareto-optimal front is considerably less than the number of points in the full FPM/FEM thereby suggesting development of methods that can leverage this finding to drastically reduce the model construction times.

KEYWORDS

bi-objective optimization, data parallelism, energy, multicore, performance, self-adaptable

1 | INTRODUCTION

Self-adaptability is a highly preferred feature in HPC and must-have in application domains such as adaptive mesh refinement, particle simulations, transient dynamics calculations, etc. We define self-adaptable applications in the HPC context as applications that automatically adapt at runtime to any set of heterogeneous processors with a priori unknown performance characteristics.[1,2] They are ideally suited for execution in dynamic environments where the number of available processors and their performance characteristics can be different for different runs of the same application. They must adapt at runtime to dynamic changes in the environment even during a single run.

We briefly describe few real-life cases where self-adaptability is essential. They are covered in substantial detail in Section 1 of the supplemental.

- *Self-adaptability of the solver* is vital in adaptive mesh refinement on clusters for solving large computational fluid dynamics (CFD) and computational mechanics (CM) problems where the computational load varies throughout the evolution of the solution. For example, solving for flow or stress in different parts of the domain in a multiphysics casting simulation. The works of Williams,[3] Walshaw et al,[4] and Arulananthan et al[5] are notable works describing methods for dynamic partitioning of unstructured meshes.
- *Autotuning parallel softwares* typically perform an empirical search by generating numerous versions of a program at runtime, which are then executed to find the best configuration of a program. A key building block that enables them to prune and accomplish this search in reasonable runtime is a fast data partitioning algorithm that is based on realistic computation and communication performance models. Reddy et al[6] proposed a linear algebra package for heterogeneous clusters that determines the optimal number and arrangement of processors to be used during the execution of a linear algebra kernel. One important reason how the mapping runtime module in this software accomplishes this task in a reasonable time is the invocation of fast data partitioning algorithms that are based on realistic computation and communication performance models, which are efficiently constructed at runtime.
- *Supercomputer administrators* routinely report that nodes closer to the hotter regions (hotspots) execute codes slower than the nodes closer to the cooler regions in the supercomputing centers due to variations in the airflow caused by how the cooling systems are laid out.[7] Thermal-aware workload scheduling techniques[8-10] have been proposed to take into account these temperature variations to optimize for performance and energy.
- *Shared environments* such as cloud computing systems today are placing great emphasis in facilitating easier migration and execution of HPC workloads by striving to remove daunting impediments to this process. The leading objectives for optimization for the cloud service providers are performance, energy consumption, cost, and reliability. Self-adaptable applications employing fast data partitioning algorithms for optimization of their performance and energy evidently and directly address the first two concerns.

A crucial building block of a self-adaptable application executing on modern parallel platforms composed of multicore CPU processors is a data partitioning algorithm that must strive to solve the bi-objective optimization problem for performance and energy (*BOPPE*). However, it faces a formidable challenge. Modern multicore CPU platforms are composed of tightly integrated multicore CPUs with highly hierarchical arrangement of cores. This tight integration has resulted in the cores contending for various shared on-chip resources such as Last Level Cache (LLC) and interconnect (eg, Intel's Quick Path Interconnect, AMD's Hyper Transport), leading to severe resource contention and non-uniform memory access (NUMA). Due to these newly introduced complexities, the performance and energy profiles of real-life scientific applications executing on these platforms are not smooth and may deviate significantly from the shapes observed before. These behaviors limit the applicability of state-of-the-art load balancing algorithms (based on functional performance models (FPMs)). Therefore, the data partitioning algorithm must take into account the new real-life behavior of applications by employing realistic computation and communication models of performance and energy. Along with addressing the challenge, it must also possess the following essential qualities, ie, (a) it must have low practical runtime and memory costs compared to that of the application, and (b) it must minimize the cost of data redistribution arising from dynamic partitioning. We do not consider the cost of data redistribution in our proposed solution since it is quite straightforward to integrate it.

We cover briefly the new behaviors of the data-parallel applications executing on modern homogeneous clusters of multicore CPUs. They are described in sufficient detail in the following background section. The state-of-the-art load balancing algorithms designed for optimization of the computational performance of data-parallel applications assume that their performance profiles (FPMs) satisfy properties of continuity and certain assumptions on shape such as smoothness. The smooth FPMs accurately capture the shapes of real-life scientific applications on platforms consisting of uniprocessors (single-core CPUs). We illustrate this using the execution of the OpenBLAS DGEMM application on a single core of an Intel Haswell server (Table 1). Figures 1A and 1B, respectively, show the shapes of the experimentally built speed and dynamic energy functions. The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2). In these experiments, the *numactl* tool is used to bind the application to one core. The dynamic energy consumptions are obtained using Watts Up Pro power meter. To make sure the experimental results are reliable, we follow a statistical methodology described in Section 4 of the supplemental. Briefly, for every data point in the functions, the automation software executes the application repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, 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 speed and energy values shown in the graphical plots throughout this work are the sample means.

However, if we run the same application on all 24 cores of the multicore CPU of the Haswell server executing 24 threads, the picture will drastically change, as shown in Figure 3. The performance and energy profiles are no longer smooth and deviate significantly from the shapes observed before. Even more spectacular variations in speed and energy can be seen in Figure 2 for the FFTW application[11] performing a 2D FFT of size $n \times n$ (the problem size being n^2). The variations in energy reach a maximum of 400%, and the average variation in speed is 300%. It is important to note that these variations are not noise but an inherent trait of applications executing on multicore servers with resource contention and NUMA. It is evident that equal distribution of the workload between identical processors with such performance and energy profiles will no longer guarantee minimization of execution time or energy consumption. More generally, traditional methods and algorithms used for optimization of performance and/or energy of parallel applications will not work for modern multicore-based platforms.

We now present an overview of the state-of-the-art data partitioning algorithms that have addressed the first two challenges. We then explain why they are not suitable for employment in self-adaptable applications before presenting our solution.

TABLE 1 Specification of the Intel Haswell server used to build the functional performance and energy models for multithreaded *Intel MKL FFT* and *OpenBLAS DGEMM* applications

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

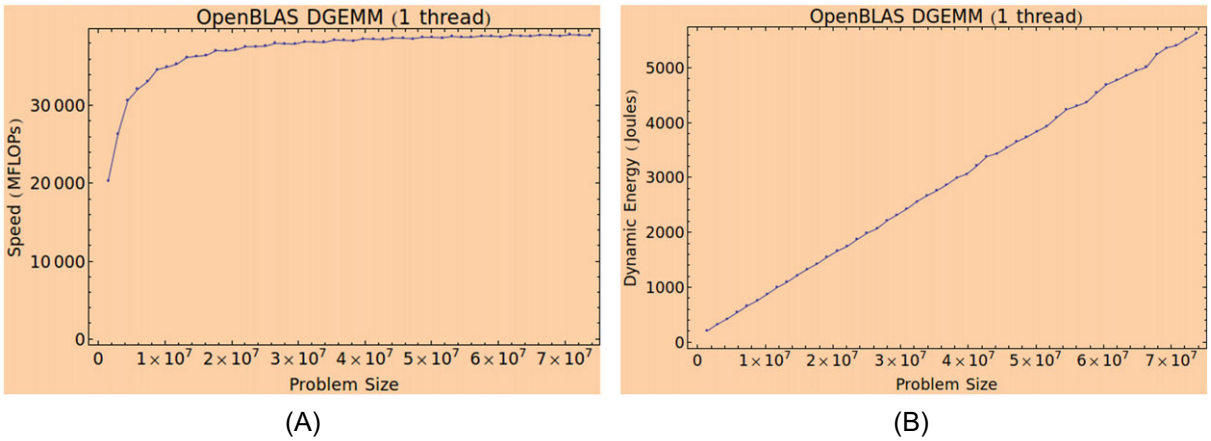

FIGURE 1 A, Speed function of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server; B, Dynamic energy consumption of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server. The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2)

Lastovetsky and Reddy[12] illustrated in depth the drastic variations in performance and energy profiles of two widely known and highly optimized scientific routines, ie, OpenBLAS DGEMM[13] and FFTW,[11,14] on a modern multicore Intel Haswell CPU platform. They propose novel model-based methods and algorithms for minimization of time and energy of computations (called *POPTA* and *EOPTA*, respectively) for the most general performance and energy profiles of data-parallel applications executing on homogeneous clusters of modern multicore CPUs. Unlike load balancing algorithms, optimal solutions found by these algorithms may not load-balance an application. Manumachu and Lastovetsky[15] studied *BOPPE* for data-parallel applications on homogeneous clusters of modern multicore CPUs, which is based on only one decision variable, ie, workload distribution. They present an efficient and exact global optimization algorithm called *ALEPH* that solved the *BOPPE*. It takes as inputs, functions of performance and dynamic energy consumption against problem size, and outputs the globally Pareto-optimal set of solutions. These solutions are the workload distributions, which achieve inter-node optimization of data-parallel applications for performance and energy.

The algorithms of the works of Lastovesky and Reddy[12] and Manumachu et al[15] have time complexity of $O(m^2 \times p^2)$, where m is the cardinality of the discrete sets representing the speed/energy functions and p is the number of available processors. The memory complexity of the algorithms is $O(n \times p^2)$. However, they are sequential, recursive, and have high practical runtime, and memory costs for large values of p (several hundreds). To address this problem, Reddy and Lastovetsky[16] proposed parallel versions of the sequential data partitioning algorithms. They have low time complexity of $O(m^2 \times p)$, where m is the number of points in the discrete speed/energy function and p is the number of available processors. The memory complexity of the algorithms is just $O(m \times p)$.

The state-of-the-art discussed above take as inputs, full functional models of performance (FPM) and dynamic energy consumption (FEM) against problem size. However, the cost of experimentally constructing the full FPM and FEM is prohibitively high. We illuminate this point by considering the execution times of building them for two data-parallel applications, *Intel MKL FFT* and *OpenBLAS DGEMM*,[13] on an Intel multicore server CPU (Table 1).

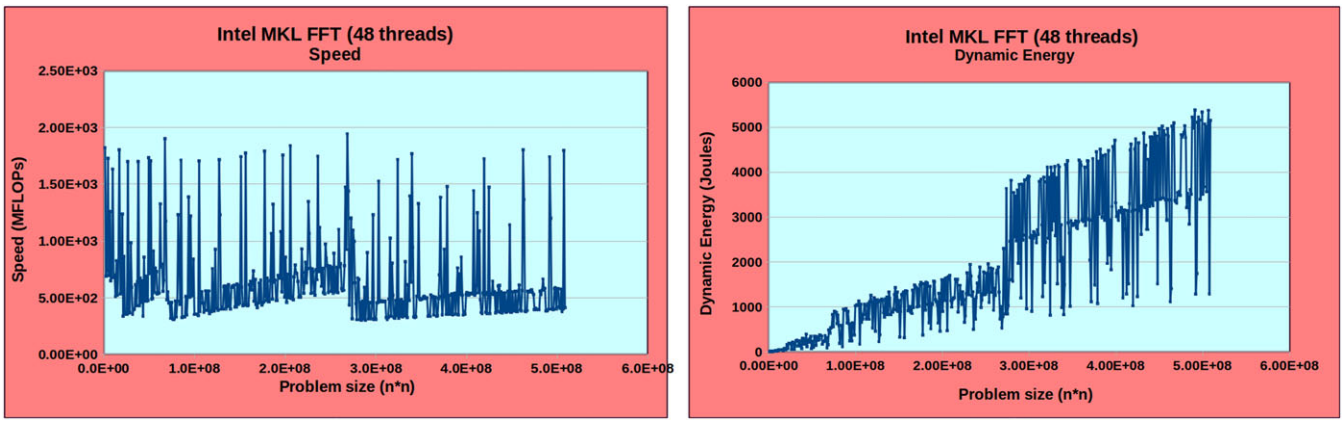

FIGURE 2 A, Speed function of multithreaded *Intel MKL FFT* executed using 48 threads on the Intel Haswell server; B, Dynamic energy function of multithreaded *Intel MKL FFT* executed using 48 threads on the Intel Haswell server. The application computes a DFT (2D, double precision, real-to-complex) of problem size n^2

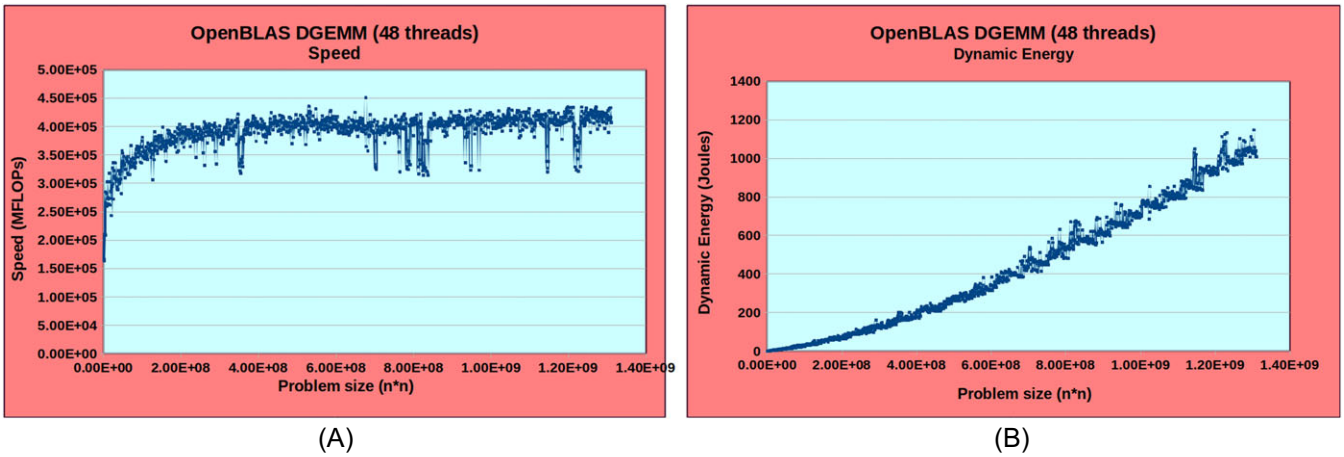

FIGURE 3 A, Speed function of multithreaded *OpenBLAS DGEMM* executed using 48 threads on the Intel Haswell server; B, Dynamic energy function of multithreaded *OpenBLAS DGEMM* executed using 48 threads on the Intel Haswell server. The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2)

Before that, we define what we mean by a full FPM/FEM. Functional performance models (FPMs) were defined in the works of Lastovetsky and Reddy.[17,18] Functional energy models were defined in the work of Manumahcu et al.[15] A full functional model contains points, $(x, f(x))$, where x is a multiple of minimum granularity. $f(x)$ is a speed function in the case of FPM and dynamic energy consumption function in the case of FEM. Minimum granularity is defined to be the minimum allocation unit to a processor during the execution of a data-parallel application. Therefore, the data is allocated to the processors in multiples of minimum granularity. Consider, eg, the parallel matrix-matrix application based on *SUMMA*.[19] The minimum allocation unit is a square block of size $b \times b$, where b is determined experimentally and the unit of computation in this application is a local DGEMM update of two $b \times b$ blocks. The last point in these functions corresponds to the problem size that fills the virtual memory. Since swapping is disabled in our Intel Haswell server, the last problem size experimentally built in the full FPM/FEM functions fills the main memory, and therefore, there are no points with problem sizes that exceed it.

The performance and energy models are built simultaneously using an automated build procedure. The dynamic energy consumption during the application execution is obtained using *Watts Up Pro* power meter. For each data point, the execution time and dynamic energy consumption (sample means) are measured together by invoking the HCLWattsUp interface functions,[20] which employ the statistical methodology detailed in Section 4 of the supplemental. The full FPM and FEM for *Intel MKL FFT* are shown in Figures 2A and 2B and for *OpenBLAS DGEMM*, in Figures 3A and 3B, respectively. Both the functions are constructed together since obtaining energy values using physical measurements from power meters adds negligible overhead to determination of execution times using processor clocks.

The number of points in the functions for *Intel MKL FFT* and *OpenBLAS DGEMM* are 485 and 1250, respectively, and their model construction execution times are 25920 seconds and 11540 seconds, respectively. The reason for the high model construction time of *Intel MKL FFT* (as compared to *OpenBLAS DGEMM*) is because it is optimized only for specific problem sizes and exhibits severe drops in performance for other problem sizes.

Consider the execution times of the parallel FFT and parallel matrix-matrix multiplication applications employing the data partitioning algorithms. Since we could not find a cluster containing multicore servers similar to our server, just to give you an idea of the application execution times involved, we perform experiments in the Grid'5000 platform hosted in France (http://www.grid5000.fr). The application execution times represent the upper bounds in the sense that they would be less on a cluster of multicore servers with the specification shown in Table 1. The platform contains 24 clusters distributed over 10 sites (nine in France and one in Luxembourg), which includes 1006 nodes and 8014 cores. We used the Graphene cluster in Nancy site for our experiments. We used a total of 576 cores from 144 nodes. Each node has a disk of 298 GB storage, 16 GB of memory, and a quad-core Intel Xeon X3440 CPU. The nodes in the cluster are interconnected via 20 Gb/s Infiniband. For the MPI communications, OpenMPI-1.6.5 is used. *gcc* compiler version used for compilation is 4.9.2.

The parallel FFT application (*PFFT*) computes a DFT (2D, double precision, real-to-complex) of size $n \times n$. For local computations, it employs the *Intel MKL FFT* routine, which computes a 2D DFT of size $n_l \times n$ (the 2D array $n \times n$ is partitioned so that each process gets a subset n_l of the rows). For homogeneous workload distribution, $n_l = \frac{n}{p}$. The execution times of *PFFT* range from 1.5 seconds to 37 seconds, respectively, to compute 2D DFT of domains whose problem sizes lie between 4096×4096 and 16384×16384 using 2 processes. Using 144 processes, the execution times range from 39 seconds to 565 seconds respectively to compute 2D DFT of domains whose problem sizes lie between 32768×32768 and 139264×139264. Using 576 processes, the execution times range from 154 seconds to 1560 seconds respectively to compute 2D DFT of domains whose problem sizes lie between 69362×69362 and 277504×277504.

The parallel matrix-matrix application (*PDGEMM*) is based on SUMMA[19] and employs heterogeneous two-dimensional block-cyclic distribution of matrices.[21] In this application, the square matrices A, B, and C of size $(n \times n)$ are distributed over a two-dimensional arrangement of processors, $p_1 \times p_2, p_1 = \sqrt{p}, p_2 = \frac{p}{p_1}$. The local computations are performed by a *OpenBLAS DGEMM* routine, which updates two matrices of sizes $(n_l \times b)$ and $(b \times n_l)$, where b is the block size (experimentally determined to be 256). For homogeneous workload distribution, $n_l = \frac{n}{p}$. The execution times of *PDGEMM* range from 10 seconds to 590 seconds, respectively, to multiply matrices with problem sizes lying between 4096×4096 and 16384×16384 using 2 processes ($p_1 = 2, p_2 = 1$). Using 144 processes ($p_1 = 12, p_2 = 12$), the execution times range from 648 seconds to 8461 seconds respectively to multiply matrices with problem sizes lying between 32768×32768 and 139264×139264. Using 576 processes ($p_1 = 24, p_2 = 24$), the execution times range from 2975 seconds to 25392 seconds, respectively, to multiply matrices with problem sizes lying between 69362×69362 and 277504×277504.

One can see that the full model construction time can be greater than the execution time of the data-parallel application. Therefore, the high model construction costs hinder the applicability of these data partitioning algorithms in self-adaptable applications. They are more suited for situations where the full FPM and FEM are built once and used for several application runs so that the model construction costs become insignificant when compared to the total performance gains from the executions of the optimized application. However, this approach does not apply to self-adaptable applications, which are executed in dynamic environments where the number of available processors and their performance characteristics can be different for different runs of the same application.

In this paper, we address this problem by proposing a self-adaptable data partitioning algorithm called *ADAPTALEPH*, which solves *BOPPE* on homogeneous clusters of multicore CPUs. Unlike the algorithms that take as inputs full FPM and FEM, it constructs partial FPM and FEM during its execution using all the available processors, which makes it a distributed algorithm. It accepts as input the size of neighborhood $\mathcal{T}$, which is the number of points in the neighborhood of (ie, on either side of load-balanced point) $x = \frac{n}{p}$ to be constructed in the FPM/FEM. Unlike the state-of-the-art data partitioning algorithms for self-adaptable applications that return a solution, which tries to maximize performance, *ADAPTALEPH* returns a locally Pareto-optimal set of solutions. These solutions are the heterogeneous workload distributions, which achieve inter-node optimization of data-parallel applications for performance and energy. The user can then select a solution that satisfies her preference, which expresses a trade-off between performance and energy, during the execution of the data-parallel application.

Our work extends the research presented in the works of Clarke et al[1] and Lastovetsky et al[2] where a data partitioning algorithm was proposed that tries to optimize the data-parallel application for performance only. It builds a partial estimate of the speed function sufficient for optimal distribution of computations and returns a solution not perfectly balancing the load of the processors but rather a solution balancing their load with a given accuracy, which acts as the termination criterion for the algorithm. It is targeted for clusters of heterogeneous processors. However, *ADAPTALEPH* tries to optimize a data-parallel application on homogeneous clusters of multicore CPUs for both performance and energy. It returns a locally Pareto-optimal front of solutions, which are workload distributions that may not load-balance the application. To determine the globally Pareto-optimal front of solutions, the algorithm must necessarily build the full FPM and FEM, which is prohibitively expensive. Therefore, it takes as input the number of points to build in the partial FPM and FEM, which allows the user to limit the execution time taken by it. It also takes as input a tolerance (ε), which expresses the tolerable variation between the speeds and the dynamic energies (using their sample means) of the processors executing a problem size. If the tolerance is exceeded, *ADAPTALEPH* indicates that the speed and energy functions of the processors are not homogeneous by quitting and returning the load-balanced workload distribution.

We present experimental analysis of the practical efficiency of our algorithm using three data-parallel applications, ie, parallel matrix-vector multiplication based on *Intel MKL DGEMV* routine, parallel matrix-matrix multiplication based on SUMMA[19] and employing *OpenBLAS DGEMM* for local computations, and parallel fast Fourier transform employing *Intel MKL FFT* for local computations, on a modern multicore CPU and simulations on clusters of such CPUs.

We demonstrate that while the partial FPM and FEM model construction times for *OpenBLAS DGEMM* application and the total execution time of *ADAPTALEPH* are quite reasonable compared to the execution times of the parallel application, it is not the case for *Intel MKL FFT*. This is because

the *Intel MKL FFT* is optimized for only specific problem sizes (power-of-two, prime number). We propose two approaches to reduce the partial FPM and FEM construction times. One is to increase the granularity between the problem sizes in the FPM/FEM. However, we show that the locally Pareto-optimal fronts of solutions in this case are actually quite inferior. The other approach is to a priori determine if the variation between the speeds and the dynamic energies is less than the input tolerance ϵ. In this case, all the available p processors can build p data points in the FPM and FEM in parallel thereby reducing the FPM and FEM construction times by a factor of p. However, there is no straight-forward method to find out *a priori* if the variation constraint is satisfied since the underlying execution environment is dynamic. One approach is to use past records of variations and use them as a guide.

While using partial speed and energy functions, one would expect to obtain a locally Pareto-optimal front, which would lie between the load-balanced solution and the globally Pareto-optimal front. In addition, as the number of points in the partial speed and energy functions are increased, the locally Pareto-optimal front would approach the globally Pareto-optimal front. We show this to be true from our experiments.

We do not consider communications in this work. While execution times of communications (or a communication performance model) can be easily integrated in our algorithm, energy model of communications is still an open research problem.

To summarize, our main contributions in this paper are the following.

- The data partitioning algorithm *ADAPTALEPH* for solving *BOPPE* that is ideally suited for self-adaptable data-parallel applications on homogeneous clusters of multicore CPUs. Unlike the state-of-the-art for self-adaptable applications that return a single solution, which tries to maximize performance, *ADAPTALEPH* returns a locally Pareto-optimal set of solutions. The user can then select a solution that satisfies her preference, which expresses a trade-off between performance and energy, during the execution of the data-parallel application.
- We describe a procedure detailing how DVFS-based bi-objective optimization methods can be combined with *ADAPTALEPH* to determine a better Pareto-optimal front of solutions.

The rest of this paper is structured as follows. Section 2 presents the challenges posed to solve *BOPPE* by the inherent complexities in modern multicore CPUs. Section 3 contains theory and notation of multi-objective optimization and the concept of optimality. We then formulate the bi-objective optimization of data-parallel applications on homogeneous clusters of multicore CPU processors for performance and energy. Section 4 presents the self-adaptable data partitioning algorithm solving the *BOPPE*. Section 5 contains experimental analysis of the algorithm. Section 6 presents related work on data partitioning techniques targeted for self-adaptable applications. Section 7 concludes this paper.

2 | BACKGROUND: PERFORMANCE AND ENERGY OF DATA-PARALLEL APPLICATIONS ON HOMOGENEOUS CLUSTERS OF MULTICORE CPUS

We will now describe in detail the new behaviors of the data-parallel applications executing on modern homogeneous clusters of multicore CPUs. To elucidate the new behaviors, we compare the typical shapes of scientific data-parallel applications on platforms consisting of uniprocessors and modern multicore CPUs. For this purpose, we select two widely used and highly optimized scientific routines, dense matrix-matrix multiplication using *OpenBLAS DGEMM*[13] and fast Fourier transform using *Intel MKL FFT*. The *OpenBLAS DGEMM* application multiplies two dense square matrices of size $n \times n$ (problem size is equal to n^2). The *Intel MKL FFT* routine computes a discrete Fourier transform (DFT, 2D, double precision, real-to-complex) of size $n \times n$.

Consider the shapes of the speed and dynamic energy consumption functions of the OpenBLAS DGEMM application built experimentally by executing it on a single core of an Intel Haswell workstation (specification shown in Table 2). In these experiments, the *numactl* tool is used to bind the application to one core. The dynamic energy consumptions are obtained using *Watts Up Pro* power meter (refer to Section 3 of the supplemental). Figures 4A and 4B show the shapes of the speed and energy functions, whose properties are summarized below.

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

For such shapes, Lastovetsky et al[12,22,23] proved that the solutions determined by the traditional (constant performance model) and the state-of-the-art load-balancing algorithms (based on functional performance models (FPMs))[17,18,24-26] simultaneously minimize the execution time and dynamic energy consumption of computations in the parallel execution of the application. Figures 1A and 1B show the shapes of the speed and dynamic energy consumption functions, respectively, for the same application built experimentally by executing it on a single core of an Intel Haswell server (specification shown in Table 1). It can be seen that, while the shape of the speed function is the same as before, the shape of the dynamic energy consumption is linear. This implies that all workload distributions will result in same dynamic energy consumption and therefore parallelization has no effect on the dynamic energy consumption of computations in the parallel execution of the application.

TABLE 2 Specification of the Intel Haswell workstation used to build the uniprocessor speed and energy models

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

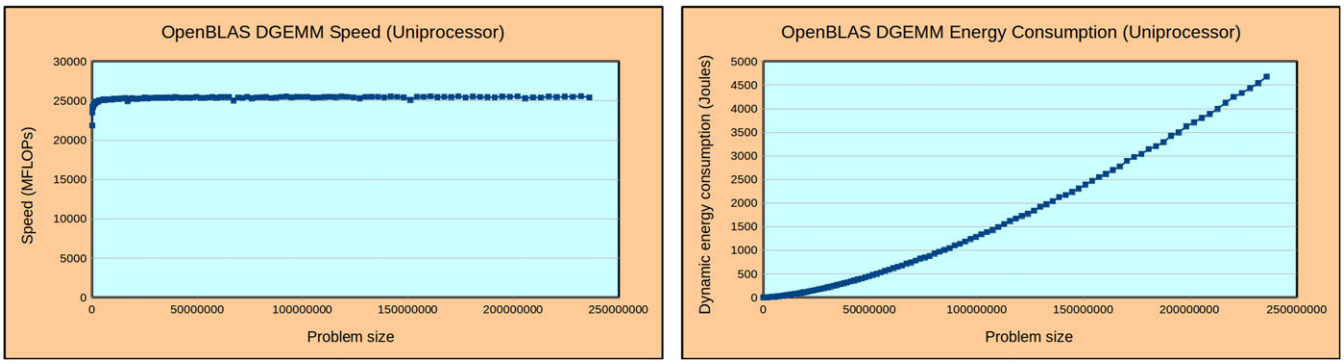

FIGURE 4 A, Speed function of OpenBLAS DGEMM application executed on a single core on the Intel Haswell workstation; B, Dynamic energy consumption of OpenBLAS DGEMM application executed on a single core on the Intel Haswell workstation. The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2)

To summarize, on platforms composed of uniprocessors, one can observe that the shapes of the performance and energy functions are smooth with minimal variations. The performance functions comfortably satisfy the conditions imposed by the FPMs that are crucial for the correct operation of the load balancing algorithms.

On modern homogeneous clusters composed of multicore CPUs, the performance and energy profiles of real-life scientific applications executing on these platforms are not smooth and may deviate significantly from the shapes observed before. This is due to the newly introduced inherent complexities such as resource contention and NUMA. This is illustrated in Figures 2 and 3, which show the speed and dynamic energy consumption graphs for multi-threaded *OpenBLAS DGEMM* and *Intel MKL FFT* applications executed with 48 threads on the Intel Haswell multicore server CPU (specification is shown in Table 1). The data points in the graphs are connected by solid lines to highlight the variations.

The variation observed is not noise but is an inherent trait of applications executing on multicore servers with resource contention and NUMA. In a function (speed or energy *f*), it is defined as the difference of function values between two subsequent local minima (f_1) and maxima (f_2), ie, $variation(\%) = \frac{|f_1 - f_2|}{\min(f_1, f_2)} \times 100$.

Some interesting properties about the variations are summarized below.

- The variations can be quite large. This is evident from the speed and energy functions shown in Figure 2A and 2B, respectively. From the speed function plot, one can observe performance drops of around 300% for many problem sizes. From the energy function plot, there are energy increases of about 400% for many problem sizes.
- The variations cannot be explained by the constant and stochastic fluctuations due to OS activity or a workload executing in a node in common networks of computers. In such networks, a node is persistently performing minor routine computations and communications by being an integral part of the network. Examples of such routine activities include OS-level daemon tasks, bookkeeping, etc. As a result, the node will experience constant and stochastic fluctuations in the workload. This changing transient load will cause a fluctuation in the speed of the node in the sense that the speed will vary for different runs of the same workload. One way to represent these inherent fluctuations in the speed is to use a speed band rather than a speed function. The width of the band characterizes the level of fluctuation in the speed due to changes in load over time.[17,18,24] For a node with uniprocessors, the width of the band has been shown to decrease as the problem size increases. For a node with a very high level

of network integration, typical widths of the speed bands were observed to be around 40% for small problem sizes and narrowing down to 3% for large problem sizes. Therefore, as the problem size increases, the width of the speed band is observed to decrease. Therefore, for long running applications, one would observe the width to become quite narrow (3%). However, this is not the case for variations in the presented graphs. The dynamic energy consumption in Figures 2B and 3B show the widths of the variations increasing as problem size increases. These widths reach a maximum of 400% and 25%, respectively, for large problem sizes. The speed functions in Figures 2A and 3A demonstrate that the widths are bounded with the averages around 300% and 20% respectively. This suggests therefore that the variation is largely due to the newly introduced complexities and not due to the fluctuations caused by changing transient load.

Although we use two standard scientific kernels to illustrate the drastic variations in performance and energy profiles, these variations have been the central research focus in the works of Lastovetsky et al[22,23] where the authors studied them for a real-life scientific application, Multidimensional Positive Definite Advection Transport Algorithm (MPDATA). MPDATA is a core component of the EULAG (Eulerian/semi-Lagrangian fluid solver) geophysical model,[27] which is an established computational model developed for simulating thermo-fluid flows across a wide range of scales and physical scenarios.

Therefore, these variations are not singular and will become inherent because chip manufacturers are increasingly favoring and featuring tighter integration of processor cores, memory, and interconnect in their products.

To summarize, the new inherent complexities introduced in modern multicore CPUs have posed formidable challenges to solving optimization problems of minimization of time and energy of computations for the most general shapes of performance and energy profiles of data-parallel applications observed on such platforms. These profiles limit the applicability of state-of-the-art load balancing algorithms (based on FPMs) thereby necessitating either a thorough redesign or development of novel models and algorithms.

3 | MULTI-OBJECTIVE OPTIMIZATION (MOP): BACKGROUND

In this section, we briefly describe the theory of multi-objective optimization and the concept of optimality.

A multi-objective optimization (MOP) problem may be defined as follows:[28,29]

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

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

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

If there is no conflict between the objective functions, then a solution x^* can be found where every objective function attains its optimum[29]

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

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

Definition 1. A decision vector $x^* \in \mathcal{S}$ is *Pareto-optimal* if there does not exist another decision vector $x \in \mathcal{S}$ such that $f_i(x) \leq f_i(x^*), \forall i = 1, \ldots, k$ and $f_j(x) < f_j(x)^*$ for at least one index j. [28]

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

Mathematically speaking, every Pareto-optimal point is an equally acceptable solution of the multi-objective optimization problem. Therefore, user preference relations (or preferences of decision maker) are provided as input to the solution process to select one or more points from the set of Pareto-optimal solutions.[28]

In Figure 5, a feasible region $\mathcal{S} \subset \mathbb{R}^3$ and its image, a feasible objective region $\mathcal{Z} \subset \mathbb{R}^2$, are shown. The thick blue line in the figure showing the objective space contains all the Pareto-optimal objective vectors. The vector z^* is one of them.

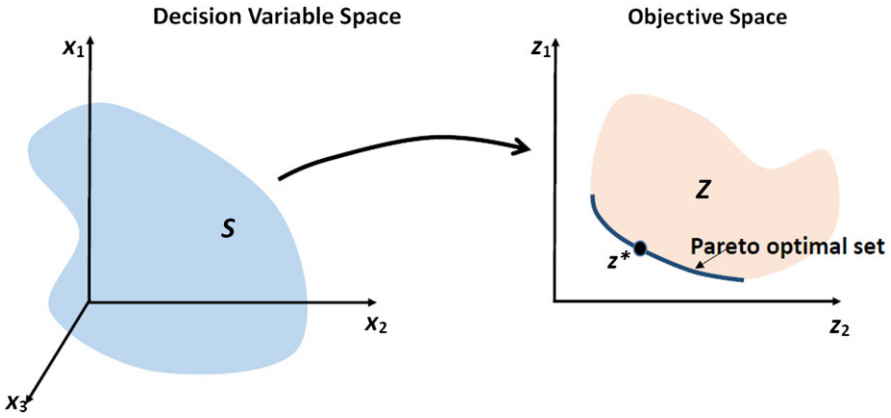

FIGURE 5 An example showing the set $\mathcal{S}$ of decision variable vectors, the set $\mathcal{Z}$ of objective vectors, and Pareto-optimal objective vectors

3.1 | Bi-objective optimization for performance and energy on homogeneous multicore clusters *BOPPE*: problem formulation

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

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

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

4 | *ADAPTALEPH*: SELF-ADAPTABLE DATA PARTITIONING ALGORITHM SOLVING *BOPPE*

In this section, we present the self-adaptable data partitioning algorithm solving *BOPPE* called *ADAPTALEPH* (Algorithm 1).

The inputs to the algorithm are data-parallel application workload size n, the number of available processors p, the size of neighborhood, $\mathcal{T}$, which is the number of points in the neighborhood of ($x = \frac{n}{p}$) to be constructed in the speed/energy function, the step size between the points in the speed/energy function, Δx, the tolerance, ε, which represents the maximum variation in the speeds and the dynamic energies that can be tolerated, and finally, the data-parallel kernel, *DPKernel*. The data-parallel kernel contains the core computations of the data-parallel application.

We define the neighborhood $\mathcal{N}$ to be the following set of points in the vicinity of the load-balanced point, $\frac{n}{p}$: $\mathcal{N} = \{\frac{n}{p}, \frac{n}{p} - \Delta x, \frac{n}{p} + \Delta x, \ldots, \frac{n}{p} - \mathcal{T} \times \Delta x, \frac{n}{p} + \mathcal{T} \times \Delta x\}$. The size of the neighborhood is given by $2 \times \mathcal{T} + 1$. The points are separated by the granularity of computation, Δx. Briefly, if the granularity is small, there is ample scope for finding globally Pareto-optimal solutions. However, the cost of building the FPM and FEM functions increases.

All the processors take part in the execution of *ADAPTALEPH* and are identified by $id \in \{1, 2, \ldots, p\}$. The size of the neighborhood, $\mathcal{T}$, and the granularity or the step size, Δx, determine how the points in the partial FPM/FEM are constructed, That is, all the processors take part in the construction of each point in the partial FPM/FEM and the order of construction is following: $\{\frac{n}{p}, \frac{n}{p} - \Delta x, \frac{n}{p} + \Delta x, \ldots, \frac{n}{p} - \mathcal{T} \times \Delta x, \frac{n}{p} + \mathcal{T} \times \Delta x\}$.

The outputs from the algorithm are stored at processor 1. They are the set of locally Pareto-optimal solutions for performance and energy, $\mathcal{P}_p$, and the corresponding workload distributions, $\mathcal{D}_p$.

Algorithm 1 Self-adaptable algorithm determining locally Pareto-optimal solutions for performance and energy of computations in the parallel execution of workload of size n

1: **procedure** ADAPTALEPH($n, p, \mathcal{D}_p, \mathcal{P}_p$)

Input:

Data-parallel application workload size, $n \in \mathbb{Z}_{>0}$

Number of processors, $p \in \mathbb{Z}_{>0}$

Size of neighbourhood, $\mathcal{T} \in \mathbb{Z}_{>0}$

Step size between points, $\Delta x \in \mathbb{Z}_{>0}$

Tolerance, $\epsilon \in \mathbb{R}_{>0}$

Data-parallel kernel, *DPKernel*

Output:

Set of trade-off solutions optimizing performance and energy,

$\mathcal{D}_p = \{(x[1][1], \ldots, x[1][p]), \ldots, (x[|\mathcal{D}_p|][1], \ldots, x_{|}|\mathcal{D}_p|][p]), x[i][j] \in \mathbb{Z}_{>0}, \forall i \in [1, |\mathcal{D}_p|], \forall j \in [1, p]$

$\mathcal{P}_p = \{((t_1, e_1), \ldots, (t_{|\mathcal{P}_p|}, e_{|\mathcal{P}_p|})\}, t_i, e_i \in \mathbb{R}_{>0}$

2: $(rc, \mathcal{X}, \mathcal{S}, \mathcal{E}) \leftarrow$ BUILDPARTIALMODELS($n, p, \mathcal{T}, \Delta x, \epsilon$, *DPKernel*)

3: **if** $(rc \neq 0)$ **and** $(id = 1)$ **then**

4: $\mathcal{D}_p[1][i] \leftarrow \frac{n}{p}, \forall i \in [1, p]$; $\mathcal{D}_p[1][i] \leftarrow \mathcal{D}_p[1][i] + 1, \forall i \in [1, n \mod p]$

5: $\mathcal{P}_p[1][0] \leftarrow \max_{1 \le i \le p}(\frac{\mathcal{D}_p[1][i]}{\mathcal{S}[\mathcal{D}_p[1][i]]})$; $\mathcal{P}_p[1][1] \leftarrow \sum_{i=1}^{p} \mathcal{E}[\mathcal{D}_p[1][i]]$

6: **return** $(\mathcal{D}_p, \mathcal{P}_p)$

7: **end if**

8: **if** $(id = 1)$ **then**

9: $memoized[l][J][K] \leftarrow (l, 0, 0), \forall l \in [\frac{n}{p}, |\mathcal{X}|], J \in [1, p], K \in [1, p]$

10: $(\mathcal{D}_p, \mathcal{P}_p) \leftarrow$ ALEPHCORE(*TIME*, $n, p, |\mathcal{X}|, \mathcal{X}, \mathcal{S}, \mathcal{E}$, *memoized*, 1)

11: $(\mathcal{D}_p, \mathcal{P}_p) \leftarrow$ ALEPHCORE(*ENERGY*, $n, p, |\mathcal{X}|, \mathcal{X}, \mathcal{S}, \mathcal{E}$, *memoized*, 1)

12: **end if**

13: **return** $(\mathcal{D}_p, \mathcal{P}_p)$

14: **end procedure**

The first step is to build the partial FPM and FEM by invoking the function, *BuildPartialModels* (Algorithm 2, Section 6 of the supplemental). All the processors take part in the execution of this function. The processors start by executing the data-parallel kernel *DPKernel* for the problem size, $\frac{n}{p}$. The speeds and dynamic energies for this problem size are broadcast to processor 1, which determines the variation in the speeds and the dynamic energies using the functions, *GetSpeedVariation* and *GetEnergyVariation* (Algorithm 1, Section 5 in the supplemental), respectively. The variation is then broadcast by processor 1 to all the other processors. If the variation exceeds the tolerance, ϵ, then it indicates that the speed and energy functions are not homogeneous, which means that we can not use the average for the speeds and the dynamic energies.[22,23] It should be noted that the speeds and the energies that we refer to here are actually sample means output by a statistical method. In this case, *ADAPTALEPH* terminates by returning one solution, which is the execution time and dynamic energy consumption for the load balanced distribution ($x_i = \frac{n}{p}, \forall i \in [1, p], x_i += 1, \forall i \in [1, n \bmod p]$). If the variation does not exceed the tolerance, then the speed and energy functions for the problem size, $x = \frac{n}{p}$, are updated at processor 1. The index in the arrays representing the speed and energy functions is given by $\mathcal{T}$. The speed and energy set for a problem size in the output speed and energy functions ($(\mathcal{S}, \mathcal{E})$) are calculated to be the average of all the speeds and all the energies, respectively.

Then, the processors execute the problem size $\frac{n}{p} - \Delta x$ in the neighborhood of $\frac{n}{p}$ followed by $\frac{n}{p} + \Delta x$ and so on until the number of points experimentally obtained equals $2 \times \mathcal{T} + 1$.

Once the partial FPM and FEM are constructed, processor 1 executes the remainder of *ADAPTALEPH* (Algorithm 1, lines 9 to 11). The algorithm is an extension to the algorithm presented in the work of Manumachu and Lastovetsky,[15] which takes as input the full FPM and FEM instead of partial FPM and FEM. The core of *ALEPH* contains two invocations to the algorithm, *ALEPHCORE* (Algorithm 2). *ALEPHCORE* determines the optimal workload distribution solving single objective optimization problem of performance/energy based on the input partial FPM and FEM functions. The input *ObjType* signifies the type of optimization (performance or energy).

The *ALEPHCORE* algorithms for performance and energy, respectively, are variants of the *POPTA* and *EOPTA* algorithms, which are presented in detail in the work of Lastovetsky and Reddy.[12] Unlike *POPTA*, which examines a subset of points in the full FPM ($\mathcal{X}, \mathcal{S}$), and *EOPTA*, which examines only the convex points in the full FEM ($\mathcal{X}, \mathcal{E}$), *ALEPHCORE* examines all the points in the partial FPM and FEM, which is equal to $(2 \times \mathcal{T} + 1)$ (the cardinality of the sets ($\mathcal{X}, \mathcal{S}, \mathcal{E}$)).

A key optimization in *ALEPHCORE* is the 3D array, *memoized*, of size $O(\mathcal{T} \times p^2)$, which memorizes the points that have already been visited during the recursive invocations of *ALEPHCORE* in the invocation of *ADAPTALEPH*. Briefly speaking, for the execution of the problem size n using p processors, the array value $memoized(\frac{n}{p}, p, n \bmod p)$ contains the ending index of the range of points examined during the previous invocation, the optimal

workload distribution, and the optimal value of the objective function. The array entry $memoized(\frac{n}{p}, p)$ is of size p, where the index $nmodp$ represents a problem size $(\frac{n}{p} + n \bmod p)$ in the range $[\frac{n}{p}, \frac{n}{p} + p]$. This memorization ensures that there are only $O(\mathcal{T} \times p^2)$ recursive invocations of the core kernel (Algorithm 2) to solve a problem size of n using p processors.

Algorithm 2 Core algorithm of *ADAPTALEPH* determining feasible partitions of workload of size n

```
1: function ALEPHCORE(ObjType, n, p, F, 𝒳, 𝒮, ℰ, memoized, rLevel, 𝒟_p, 𝒫_p)
2:     if (p = 1) then return ({n}, {n/𝒮[n], ℰ[n]}) end if
3:     d_opt[i] ← n/p, ∀i ∈ [1, p]; d_opt[i] ← d_opt[i] + 1, ∀i ∈ [1, n mod p]
4:     if ObjType = TIME then f_opt ← max_{1≤i≤p}(d_opt[i] / 𝒮[d_opt[i]]) else f_opt ← Σ_{i=1}^{p} ℰ[d_opt[i]]
5:     for L ← memoized(n/p, p, n mod p, 1), F do
6:         for r ← 1, p − 1 do
7:             n_r ← 𝒳[L]; n_l ← n − r × n_r
8:             if n_l < 0 then
9:                 if n > |𝒳| then break end if
10:                if ObjType = TIME then f_r ← n/𝒮[n] else f_r ← ℰ[n]
11:                if (rLevel = 1) then UpdateParetoSet(ObjType, (n, 0, ..., 0), f_tmp, 𝒟_p, 𝒫_p) end if
12:                if f_r < f_opt then
13:                    d_opt[i] ← 0, ∀i ∈ [2, p]; d_opt[1] ← n; f_opt ← f_r
14:                    break
15:                end if
16:            end if
17:            if ObjType = TIME then f_r ← n_r/𝒮[n_r] else f_r ← r × ℰ[n_r]
18:            if n_l = 0 then
19:                if (rLevel = 1) then UpdateParetoSet(ObjType, (n_1, ..., n_r, 0, ..., 0), f_tmp, 𝒟_p, 𝒫_p) end if
20:                if f_r < f_opt then
21:                    d_opt[i] ← n_r, ∀i ∈ [1, r]; d_opt[i] ← 0, ∀i ∈ [r + 1, p]; f_opt ← f_r
22:                    break
23:                end if
24:            end if
25:            if memoized(n_l/(p−r), p − r, n mod p, 1) > L then
26:                f_l ← memoized(n_l/(p−r), p − r, n mod p, 3)
27:                if ObjType = TIME then f_tmp ← max(f_l, f_r) else f_tmp ← e_l + e_r
28:                if f_tmp < f_opt then
29:                    x_i ← memoized(n_l/(p−r), p − r, extra, 2), ∀i ∈ [r + 1, p]
30:                end if
31:            else ((x_{r+1}, ..., x_p), f_tmp, 𝒟_p, 𝒫_p) ← ALEPHCORE(ObjType, n_l, p − r, L, 𝒳, 𝒮, ℰ, memoized, rLevel)
32:            end if
33:            if (rLevel = 1) then UpdateParetoSet(ObjType, (n_r, ..., n_r, x_{r+1}, ..., x_p), f_tmp, 𝒟_p, 𝒫_p) end if
34:            if f_tmp < f_opt then
35:                d_opt[i] ← n_r, ∀i ∈ [1, r]; d_opt[i] ← x_i, ∀i ∈ [r + 1, p]; f_opt ← f_tmp
36:            end if
37:        end for
38:    end for
39:    memoized(n/p, p, n mod p) ← (L, d_opt, f_opt)
40:    return (𝒟_p, 𝒫_p)
41: end function
```

Line 2 deals with the simple case of solving the problem size n using one processor. The execution time to solve the problem is $\frac{n}{\mathcal{S}[n]}$ and the energy consumption is $\mathcal{E}[n]$ using the partial FPM and FEM.

Then, it starts from a balanced workload distribution, $x_i = \frac{n}{p}, \forall i \in [1, p]$. Lines 3 and 4 initialize the optimal workload distribution, d_{opt}, and the optimal value of the objective function, f_{opt}.

Lines 5 to 40 contain the kernel of *ALEPHCORE*. The points between $B = \frac{n}{p}$ and $F = |\mathcal{X}|$ are sequentially examined (Line 5). For each point A, there are $(p - 1)$ main execution steps in the nested *for* loop (Line 6). In a main step, each of the r processors is allocated the problem size n_r to the right of B. If the remaining problem size n_l is less than 0, that means there is excessive allocation to the right of B. In this situation, we allocate all the problem size n to the processor 1 and save this distribution if the objective function value is lesser (ie, $f_r < f_{opt}$) (lines 8 to 16). We then break from the loop

because subsequent allocations to the right of B will always result in negative remaining problem size to the left of B to be solved using a recursive invocation of *ALEPHCORE*. If the remaining problem size n_l is equal to 0, then we save this distribution if the objective function value is lesser (ie, $f_r < f_{opt}$) (lines 18 to 24).

To solve now the problem size n_l to the left of B using $p - r$ processors, we check if the range of points ($[\frac{n_l}{p-r}, L]$) have already been examined (Line 25). If yes, then they will not be re-examined due to the memorization and recursive invocation of *ALEPHCORE* is avoided. The memorized optimal objective function value solving this problem size is retrieved in f_l. If the combined objective function value (f_{tmp}) is less than f_{opt}, then we save the memorized workload distribution and avoid recursion (lines 26 to 30).

For the main step, if the objective function value of the parallel execution (f_{tmp}) is lesser than the previously saved value, f_{opt}, then we save the improved solution (lines 34 to 36). For each problem size n_l solved using $p - r$ processors, the ending index L, which contains the range of points already examined, is saved (Line 39). Therefore, if an invocation for solving this problem size recurs, then recursion is avoided using the memorized arrays (lines 26 to 30). Therefore, memorization ensures that the total number of points (including those in the recursive invocations) for a point in the interval $[\frac{n}{p}, |\mathcal{X}|]$ is not more than $O(\mathcal{T} \times p^2)$.

The Pareto-optimal set of solutions is updated using the call, *UpdateParetoSet* (lines 11, 19, and 33). This call is invoked only when the recursion level is 1. This level essentially represents the different workload distributions for workload of size n using p possible combinations of number of processors.

ADAPTALEPH can be easily extended for the case where the user can specify input tolerances (δ_1, δ_2) for performance and energy, respectively, and it would determine the set, which has solutions where the execution times and dynamic energy consumptions do not exceed $(1 + \delta_1) \times t_{opt}$ and $(1 + \delta_2) \times e_{opt}$, respectively.

4.1 | DVFS-based MOP methods plus *ADAPTALEPH*

Dynamic voltage and frequency scaling (DVFS) is a dominant decision variable employed in several notable bi-objective optimization methods,[30-33] etc.

Our method *ADAPTALEPH*, which employs workload distribution as the only decision variable, can be combined with *DVFS*-based methods to determine a better set of (locally and globally Pareto-optimal) solutions. The essential steps are the following.

1. For a given workload size n and number of available processors p, employ load-balanced workload distribution (where all the processors are assigned $x = \frac{n}{p}$) and obtain the Pareto-optimal front using the DVFS-based bi-objective optimization method. Each point in the front is a DVFS combination for the cores in the multicore CPU processor.
2. For each point in the Pareto-optimal front (determined in Step 1)
   (a) Set all the cores of the multicore CPU processors to the frequencies in the DVFS combination.
   (b) Given the number of points to construct in the neighborhood ($\mathcal{N}$) of $\frac{n}{p}$, execute *ADAPTALEPH* to determine the locally Pareto-optimal front of solutions based on the partial FPM and FEM.
3. Construct the final Pareto-optimal front of solutions from the locally Pareto-optimal fronts obtained for the key DVFS combinations. If the total number of points from all the fronts are n, then there is an efficient algorithm of complexity $O(n \log n)$[34] to determine the final frontier.

The reader is advised to read the work of Manumachu and Lastovetsky[15] for a demonstration of this approach for two applications, matrix-matrix multiplication and fast Fourier transform.

5 | EXPERIMENTAL ANALYSIS AND DISCUSSION

In this section, we study the efficiency of *ADAPTALEPH* using three data-parallel applications that employ *Intel MKL DGEMV*, *OpenBLAS DGEMM*,[13] and *Intel MKL FFT*, respectively, for local computations. To make sure the experimental results are reliable, we follow an experimental methodology, which is described in detail in Section 4 of the supplemental. It employs automated software, which takes as inputs the application and application parameters (problem size, number of threads, etc) and statistical confidence interval. To obtain a data point in the partial FPM/FEM function, the software executes the application repeatedly until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, 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. Hereafter, when we refer to speed or energy values, we signify the sample means.

5.1 | Intel MKL DGEMV Application

For our first application, we consider the execution of a matrix-vector multiplication application executing the highly optimized multi-threaded *Intel MKL DGEMV* routine in a homogeneous cluster of six nodes where each node contains two Intel Xeon Phi accelerators. Therefore, altogether, there are twelve identical Xeon Phi co-processors. The specification of the accelerator is shown in Table 3. The application multiplies a dense matrix of size

TABLE 3 Specification of the Intel Xeon Phi coprocessor SE10/7120 series

Technical Specifications	Intel Xeon Phi SE10/7120 series
No. of processor cores	61
Base frequency	1333 MHz
Total main memory	15 GB GDDR5
L2 cache size	30.5 MB
Memory bandwidth	352 GB/sec
Memory clock	2750000 kHz
TDP	300 W
Idle Power	98 W

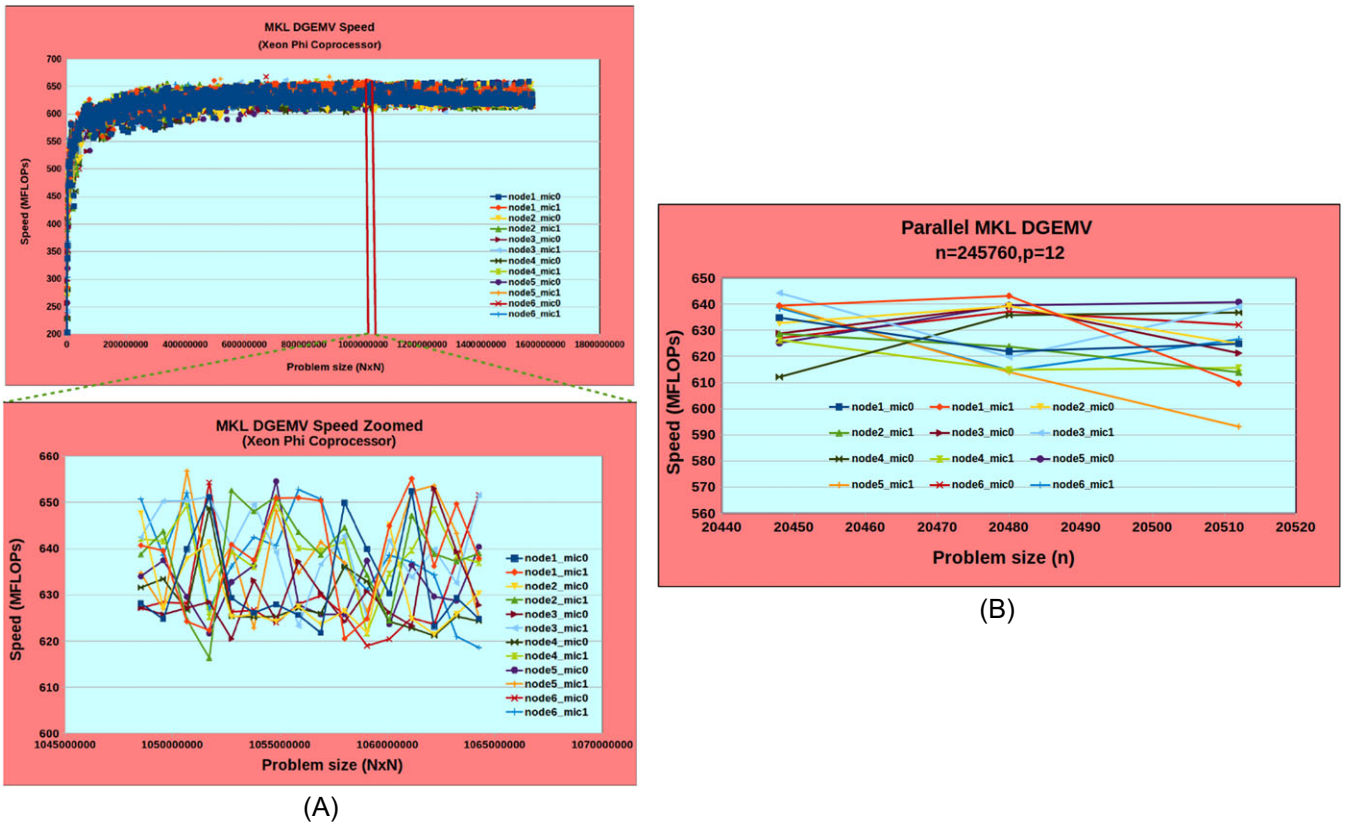

FIGURE 6 A, Full FPMs of Intel MKL DGEMV application for twelve Intel Xeon Phi SE10/7120 series coprocessors. The application multiplies a square matrix of size $n \times n$ and a vector of size n (problem size is equal to n^2); B Speeds of Intel MKL DGEMV application for twelve Intel Xeon Phi SE10/7120 series coprocessors during the execution of *ADAPTALEPH* for $(n = 245760, p = 12)$. The relative difference (variation) in speeds exceeds the tolerance, $\varepsilon = 0.05$

$n \times n$ with a vector of size n. The inputs, Δx and ε, to *ADAPTALEPH* are fixed to be 1048576 and 0.05 (5%). Therefore, for a data point in the speed function, if the variation between the speeds (using their sample means) is greater than 5%, then we consider this scenario to be heterogeneous. In this case, *ADAPTALEPH* returns the load-balanced solution.

Consider the execution of *ADAPTALEPH* for parallel DGEMV application for $(n = 245760, p = 12)$ shown in Figure 6B. For the first data point $\frac{n}{p} = 20480$, the variation is 4.7%. For the neighboring points $(\frac{n}{p} - \Delta x = 20448)$ and $(\frac{n}{p} + \Delta x = 20512)$, the variations are 5.2% and 7.5%, respectively. They exceed the input tolerance suggesting that the speed functions are not homogeneous, and therefore, *ADAPTALEPH* returns the load-balanced solution.

Figure 6A shows the full FPMs (each containing 1500 data points) of the application for all the twelve accelerators. The model construction execution time is 18226 seconds. These profiles were built simultaneously to take into account resource contention. The average variation in speeds is around 7% with maximum about 12%. To deal with such speed functions, we need to extend *ADAPTALEPH* to deal with the case of heterogeneous speed functions, which we will consider in our future work.

TABLE 4 Solutions for *PFFT* for increasing sizes of neighborhood. $n = 3974, p = 32$. Each cell shows a Pareto-optimal set containing tuples, (Execution time(sec), dynamic energy consumption (Joules)). The granularity is $\Delta x = 1048576$

$\mathcal{T} = 0$	$\mathcal{T} = 65$	$\mathcal{T} = 75$	$\mathcal{T} = 100$	$\mathcal{T} = 120$	$\mathcal{T} = 128$	*Full FPM,FEM*
					(5.89,7780.01)	(5.89,7780.01)
				(5.89,7780.01)	(5.99,7549.49)	(5.99,7549.49)
	(14.86,20362.30)		(5.89,7780.01)	(5.99,7549.49)	(6.78,7515.21)	(6.78,7515.21)
(22.10,37405.80)	(14.89,16598.16)	(5.89,7780.01)	(5.99,7549.49)	(6.78,7515.21)	(7.72,7513.10)	(7.72,7513.10)
	(15.05,13539.09)	(5.99,7549.49)	(6.78,7515.21)	(7.72,7513.10)	(9.40,7475.91)	(9.40,7475.91)
			(7.72,7513.10)	(9.40,7475.91)	(9.67,7143.50)	(9.67,7143.50)

5.2 | Parallel FFT and matrix-multiplication applications

In this section, we consider the other two data-parallel applications, parallel matrix-matrix multiplication based on *OpenBLAS DGEMM*,[13] and parallel FFT application based on *Intel MKL FFT*. The experiments are a combination of actual measurements conducted on the server (specification shown in 1) and simulations for clusters containing 256 such identical servers.

The local computations in the applications are executed on the multicores in a Intel Haswell server whose specification is shown in Table 1. The inputs, Δx and ϵ, to *ADAPTALEPH* are fixed to be 1048576 and 0.05.

5.2.1 | Full FPM and FEM

The full speed and the energy functions (FPM and FEM, respectively) are experimentally built simultaneously on the Intel Haswell server. Figures 2A and 2B, respectively, show the full FPM and FEM functions of the *FFT* application. Figures 3A and 3B, respectively, show the full FPM and FEM functions of *OpenBLAS DGEMM* application. The total dynamic energy consumption during the application execution is obtained using Watts Up Pro power meter.

The cardinality of the discrete sets representing the speed and dynamic energy functions of *OpenBLAS DGEMM* application is 1250. The cardinality of the discrete sets representing the speed and dynamic energy functions of *FFT* application is 485. The step size (Δx) selected for both the applications is the same, ie, 1048576. For the FFT application, this represents a 2D Discrete Fourier Transform of size 1024×1024. For the *OpenBLAS DGEMM* application, this represents a local DGEMM update of two 1024×256 and 256×1024 matrices.

The full FPM and FEM construction execution times for *FFT* and *OpenBLAS DGEMM* are 25920 seconds and 11540 seconds, respectively.

5.2.2 | Analysis of *ADAPTALEPH* for small *n* and *p*

We present two experiments, one with small workload size executed using small number of processors and the other using large workload size executed using large number of processors. For each experiment, we illustrate two examples demonstrating the Pareto-optimal solutions determined by *ADAPTALEPH* for the *FFT* and *OpenBLAS DGEMM* applications as the size of the neighborhood ($\mathcal{T}$) is increased. For the value of neighborhood $\mathcal{T} = 0$, the solution is the execution time and energy for the load-balanced distribution.

We consider now the first experiment for small workload sizes executed using small number of processors ($p \leq 100$).

Table 4 shows the Pareto-optimal fronts for *Intel MKL FFT* determined for $n = 3974, p = 32$ for increasing values of ($\mathcal{T}$). The load-balanced solution is 22.10 seconds and 37405 Joules, respectively. The execution time of the parallel FFT application employing *ADAPTALEPH* is 153 seconds. There are several noteworthy observations.

- The optimal execution time and the optimal dynamic energy consumptions are 5.9 seconds and 7143 Joules, respectively. The performance improvement and dynamic reduction percentages are therefore, respectively, 275% and 424%.
- *ADAPTALEPH* finds the solution containing the optimal execution time by constructing 75 points in the neighborhood of the load-balanced point, $x = \frac{n}{p}$, ($\mathcal{T} = 75$). This results in partial FPM and FEM construction times of 3712 seconds. However, it must construct 257 points (128 points on either side of the load-balanced point) to determine the optimal dynamic energy consumption, which means partial FPM and FEM construction times of 5835 seconds.
- When the size of neighborhood $\mathcal{T}$ is equal to 128, the Pareto-optimal front determined by *ADAPTALEPH* becomes the Pareto-optimal front determined given the full FPM and FEM functions (we call it the globally Pareto-optimal front). Therefore, one need not build/use the full FPM and FEM (containing 485 points) to determine the globally Pareto-optimal front. Partial FPM and FEM containing just 257 points (128 points on either side of the load-balanced point, $x = \frac{n}{p}$) are sufficient. This reduces the FPM and FEM construction time from 25920 seconds to 5835 seconds, a reduction of 350%.

There are two approaches to reduce the partial FPM and FEM construction time. One is to increase the granularity (or the step size), Δx, between the points. Consider the case where we double the granularity, $\Delta x = 2097152$. That is, we double the step size between the points, which essentially reduces the partial FPM and FEM model construction time by 2 times. Table 5 shows the Pareto-optimal fronts for increasing values of ($\mathcal{T}$). Table 6 shows the Pareto-optimal fronts when the granularity is tripled. One can see that the locally Pareto-optimal fronts are quite inferior.

TABLE 5 Solutions for *PFFT* for increasing sizes of neighborhood. $n = 3974, p = 32$. The granularity is doubled, $\Delta x = 2097152$

$\mathcal{T} = 0$	$\mathcal{T} = 65$	$\mathcal{T} = 100$	$\mathcal{T} = 200$	*Full FPM,FEM*
				(5.89,7780.01)
				(5.99,7549.49)
	(14.73,17059.41)	(12.31,17217.31)		(6.78,7515.21)
(22.10,37405.80)	(15.93,16959.31)	(13.62,16565.53)	(9.67,16497.32)	(7.72,7513.10)
				(9.40,7475.91)
				(9.67,7143.50)

TABLE 6 Solutions for *PFFT* for increasing sizes of neighborhood. $n = 3974, p = 32$. The granularity is tripled, $\Delta x = 3145728$

$\mathcal{T} = 0$	$\mathcal{T} = 65$	$\mathcal{T} = 70$	$\mathcal{T} = 100$	$\mathcal{T} = 200$	*Full FPM,FEM*
					(5.89,7780.01)
					(5.99,7549.49)
					(6.78,7515.21)
(22.10,37405.80)	(22.10,37405.80)	(20.66,33258.91)	(18.45,28441.07)	(18.45,28441.07)	(7.72,7513.10)
					(9.40,7475.91)
					(9.67,7143.50)

The other approach is to determine *a priori* that the variation between the speeds and the energies is less than the tolerance ε. Assuming this to be the case, the partial FPM and FEM construction time when the locally Pareto-optimal front becomes the globally Pareto-optimal front becomes 182 seconds. This represents a reduction of 14140% over construction times of full FPM and FEM.

Table 7 shows the Pareto-optimal fronts for *OpenBLAS DGEMM* determined for $n = 2766, p = 64$ for increasing values of ($\mathcal{T}$). The load-balanced solution is 0.84 seconds and 753 Joules respectively. The execution time of the parallel matrix-matrix multiplication application employing *ADAPTALEPH* is 3166 seconds. Some interesting observations follow.

- The optimal execution time and the optimal dynamic energy consumptions are 0.07 seconds and 597 Joules, respectively. The performance improvement and dynamic reduction percentages are therefore, respectively, 21% and 26%.
- *ADAPTALEPH* finds the solution containing the optimal execution time by constructing 22 points in the neighborhood of the load-balanced point, $x = \frac{n}{p}, (\mathcal{T} = 22)$ resulting in construction of a total of 45 points. This results in partial FPM and FEM construction times of 385 seconds. However, it must construct 93 points (43 points to the right and 50 points to the left of the load-balanced point) to determine the optimal dynamic energy consumption, which means partial FPM and FEM construction times of 804 seconds.
- When the size of neighborhood $\mathcal{T}$ becomes equal to 50, the Pareto-optimal front determined by *ADAPTALEPH* becomes the globally Pareto-optimal front determined given the full FPM and FEM functions. Therefore, one need not build/use the full FPM and FEM (containing 1250 points) to determine the globally Pareto-optimal front. Partial FPM and FEM containing just 93 points are sufficient. This reduces the FPM and FEM construction time from 11540 seconds to 804 seconds, a reduction of 1330%.
- The execution time of *ADAPTALEPH* for $\mathcal{T} = 50$ is 25% of the execution time of the parallel application.

Tables 8 and 9 shows the Pareto-optimal fronts when granularity is doubled and tripled respectively. One can observe that, in this case, there are no Pareto-optimal solutions better than the load-balanced solution. Assuming that the variation between the speeds and the energies is less than the tolerance ε, the partial FPM and FEM construction time when the locally Pareto-optimal front becomes the globally Pareto-optimal front becomes 13 seconds. This represents a reduction of 88700% over construction times of full FPM and FEM. The execution time of *ADAPTALEPH* becomes just 0.4% of the execution time of the parallel application.

As expected, the locally Pareto-optimal front approaches the globally Pareto-optimal front as one increases the number of points in the partial speed/energy functions.

5.2.3 | Analysis of *ADAPTALEPH* for large *n* and *p*

We consider now the second experiment for large workload sizes executed using large number of processors ($p > 100$).

Table 10 shows the Pareto-optimal fronts for *Intel MKL FFT* determined for $n = 20544, p = 144$ for increasing values of ($\mathcal{T}$). The load-balanced solution is 22.4 seconds and 152970 Joules, respectively. The execution time of the parallel FFT application employing *ADAPTALEPH* is 700 seconds.

Following are some interesting observations.

- The optimal execution time and the optimal dynamic energy consumptions are 5.9 seconds and 38082 Joules, respectively. The performance improvement and dynamic reduction percentages are therefore, respectively, 280% and 300%.

TABLE 7 Solutions for *OpenBLAS DGEMM* for increasing sizes of neighborhood, $\mathcal{T}$. $n = 2766, p = 64$. The granularity is $\Delta x = 1048576$

$\mathcal{T} = 0$	$\mathcal{T} = 22$	$\mathcal{T} = 23$	$\mathcal{T} = 24$	$\mathcal{T} = 30$	$\mathcal{T} = 35$	$\mathcal{T} = 40$	$\mathcal{T} = 45$	$\mathcal{T} = 50$	*Full FPM,FEM*
								(0.07,737.03)	(0.07,737.03)
							(0.07,737.03)	(0.07,681.98)	(0.07,681.98)
						(0.07,737.03)	(0.07,681.98)	(0.07,680.38)	(0.07,680.38)
					(0.07,737.03)	(0.07,681.98)	(0.07,680.38)	(0.07,676.16)	(0.07,676.16)
				(0.07,737.03)	(0.07,681.98)	(0.07,680.38)	(0.07,676.16)	(0.08,624.67)	(0.08,624.67)
			(0.07,737.03)	(0.07,681.98)	(0.07,680.38)	(0.07,676.16)	(0.08,624.67)	(0.08,612.63)	(0.08,612.63)
		(0.07,737.03)	(0.07,681.98)	(0.07,680.38)	(0.07,676.16)	(0.08,624.67)	(0.08,612.63)	(0.08,612.28)	(0.08,612.28)
	(0.07,680.96)	(0.07,681.98)	(0.07,680.38)	(0.07,676.16)	(0.08,624.67)	(0.08,612.63)	(0.08,612.28)	(0.09,601.15)	(0.09,601.15)
(0.084,906.02)	(0.07,679.69)	(0.07,680.38)	(0.07,676.16)	(0.08,624.67)	(0.08,612.63)	(0.08,612.28)	(0.09,601.15)	(0.10,600.82)	(0.10,600.82)
		(0.07,676.16)	(0.08,624.67)	(0.08,612.63)	(0.08,612.28)	(0.09,601.15)	(0.10,600.82)	(0.10,599.93)	(0.10,599.93)
			(0.08,612.63)	(0.09,601.15)	(0.09,601.15)	(0.10,600.82)	(0.10,599.93)	(0.11,598.99)	(0.11,598.99)
					(0.10,600.82)	(0.10,599.93)	(0.11,598.99)	(0.13,597.89)	(0.13,597.89)
					(0.10,599.93)	(0.11,598.99)	(0.13,597.89)	(0.14,596.70)	(0.14,596.70)

TABLE 8 Solutions for *OpenBLAS DGEMM* for increasing sizes of neighborhood, $\mathcal{T}$. $n = 2766, p = 64$. The granularity is doubled, $\Delta x = 2097152$

$\mathcal{T} = 0$	$\mathcal{T} = 50$	$\mathcal{T} = 500$	*Full FPM,FEM*
			(0.07,737.03)
			(0.07,681.98)
			(0.07,680.38)
			(0.07,676.16)
			(0.08,624.67)
			(0.08,612.63)
(0.084,906.02)	(0.084,906.02)	(0.084,906.02)	(0.08,612.28)
			(0.09,601.15)
			(0.10,600.82)
			(0.10,599.93)
			(0.11,598.99)
			(0.13,597.89)
			(0.14,596.70)

TABLE 9 Solutions for *OpenBLAS DGEMM* for increasing sizes of neighborhood, $\mathcal{T}$. $n = 2766, p = 64$. The granularity is tripled, $\Delta x = 3145728$

$\mathcal{T} = 0$	$\mathcal{T} = 50$	$\mathcal{T} = 300$	*Full FPM,FEM*
			(0.07,737.03)
			(0.07,681.98)
			(0.07,680.38)
			(0.07,676.16)
			(0.08,624.67)
			(0.08,612.63)
(0.084,906.02)	(0.084,906.02)	(0.084,906.02)	(0.08,612.28)
			(0.09,601.15)
			(0.10,600.82)
			(0.10,599.93)
			(0.11,598.99)
			(0.13,597.89)
			(0.14,596.70)

TABLE 10 Solutions for *PFFT* for increasing sizes of neighborhood, $\mathcal{T}$. $n = 20544, p = 144$. The granularity is $\Delta x = 1048576$

$\mathcal{T} = 0$	$\mathcal{T} = 80$	$\mathcal{T} = 90$	$\mathcal{T} = 100$	$\mathcal{T} = 120$	$\mathcal{T} = 150$	*Full FPM,FEM*
					(5.89,46415.33)	(5.89,46415.33)
				(5.89,46415.33)	(5.99,42414.93)	(5.99,42414.93)
		(5.89,46415.33)	(5.89,46415.33)	(5.99,42414.93)	(6.78,42049.17)	(6.78,42049.17)
(22.37,152970.4)	(5.89,46415.33)	(5.99,42436.85)	(5.99,42414.93)	(6.78,42049.17)	(9.40,40594.82)	(9.40,40594.82)
	(5.99,42436.85)	(6.78,42049.17)	(6.78,42049.17)	(9.40,40594.82)	(9.67,38082.86)	(9.67,38082.86)

- *ADAPTALEPH* finds the solution containing the optimal execution time by constructing 80 points in the neighborhood of the load-balanced point, $x = \frac{n}{p}, (\mathcal{T} = 80)$. This results in partial FPM and FEM construction times of 4385 seconds. However, it must construct 292 points to determine the optimal dynamic energy consumption, which means partial FPM and FEM construction times of 8320 seconds.
- When the size of neighborhood $\mathcal{T}$ is equal to 150, the Pareto-optimal front determined by *ADAPTALEPH* becomes the Pareto-optimal front determined given the full FPM and FEM functions (we call it the globally Pareto-optimal front). Therefore, one need not build/use the full FPM and FEM (containing 485 points) to determine the globally Pareto-optimal front. Partial FPM and FEM containing just 292 points are sufficient. This reduces the FPM and FEM construction time from 25920 seconds to 8320 seconds, a reduction of 211%.
- The execution time of *ADAPTALEPH* for $\mathcal{T} = 150$ is quite large compared to the execution time of the parallel application. Assuming that the variation between the speeds and the energies is less than the tolerance ϵ, the partial FPM and FEM construction time when the locally Pareto-optimal

TABLE 11 Solutions for *OpenBLAS DGEMM* for increasing sizes of neighborhood, $\mathcal{T}$. $n = 16896, p = 256$. The granularity is $\Delta x = 1048576$

$\mathcal{T} = 0$	$\mathcal{T} = 33$	$\mathcal{T} = 35$	$\mathcal{T} = 50$	$\mathcal{T} = 55$	*Full FPM,FEM*
				(0.10,4535.25)	(0.10,4535.25)
		(0.10,4535.25)	(0.10,4535.25)	(0.10,4464.88)	(0.10,4464.88)
	(0.11,6362.97)	(0.10,4464.88)	(0.10,4464.88)	(0.11,4010.73)	(0.11,4010.73)
(0.11,6362.96)	(0.11,5015.08)	(0.11,4464.53)	(0.11,4010.73)	(0.13,3966.88)	(0.13,3966.88)
				(0.14,3933.87)	(0.14,3933.87)

front becomes the globally Pareto-optimal front becomes 58 seconds, which is a huge order of magnitude less than the construction times of full FPM and FEM. The execution time of *ADAPTALEPH* is then 8.25% of the execution time of the parallel application.

Table 11 shows the Pareto-optimal fronts for *OpenBLAS DGEMM* determined for $n = 16896, p = 256$ for increasing values of ($\mathcal{T}$). The load-balanced solution is 0.11 seconds and 6362 Joules, respectively. The execution time of the parallel matrix-matrix multiplication application employing *ADAPTALEPH* is 10450 seconds. Some observations follow.

Some observations follow:

- The optimal execution time and the optimal dynamic energy consumptions are 0.10 seconds and 3933 Joules, respectively. The performance improvement and dynamic reduction percentages are therefore, respectively, 7% and 61%.
- *ADAPTALEPH* finds the solution containing the optimal execution time by constructing 35 points in the neighborhood of the load-balanced point, $x = \frac{n}{p}, (\mathcal{T} = 35)$ resulting in construction of a total of 71 points. This results in partial FPM and FEM construction times of 608 seconds. However, it must construct 101 points (55 points to the right and 55 poins to the left of the load-balanced point) to determine the optimal dynamic energy consumption, which means partial FPM and FEM construction times of 970 seconds.
- When the size of neighborhood $\mathcal{T}$ becomes equal to 55, the Pareto-optimal front determined by *ADAPTALEPH* becomes the globally Pareto-optimal front determined given the full FPM and FEM functions. Therefore, one need not build/use the full FPM and FEM (containing 1250 points) to determine the globally Pareto-optimal front. Partial FPM and FEM containing just 101 points are sufficient. This reduces the FPM and FEM construction time from 11540 seconds to 970 seconds, a reduction of 1090%.
- The execution time of *ADAPTALEPH* for $\mathcal{T} = 55$ is 9% of the execution time of the parallel application. Assuming that the variation between the speeds and the energies is less than the tolerance ε, the partial FPM and FEM construction time, when the locally Pareto-optimal front becomes the globally Pareto-optimal front, becomes 4 seconds. The execution time of *ADAPTALEPH* becomes quite insignificant compared to the execution time of the parallel application.

It can be seen that the execution time of *ADAPTALEPH* compared to the execution time of the parallel application becomes less and less when the problem size *n* and *p* become large.

Again, one can observe that the locally Pareto-optimal front approaches the globally Pareto-optimal front as one increases the number of points in the partial speed/energy functions.

5.3 | Discussion

The most important findings are summarized below.

- The experiments confirm our expectations that as the number of points in the partial FPM/FEM functions is increased; the locally Pareto-optimal front of solutions approaches the globally Pareto-optimal front (determined using the full FPM and FEM). In several cases, the globally Pareto-optimal front is built incrementally as one increases the number of points to be built in the partial FPM/FEM.
- The number of points in the partial FPM/FEM when the output locally Pareto-optimal front of solutions becomes the globally Pareto-optimal front is quite less compared to the number of points in the full FPM/FEM. This suggests that one can considerably reduce the model construction times if one can predict the juncture analytically when this happens. From our experiments, we show reductions of about 1330%. However, it is a difficult research problem to tackle and we would look at it in our future work.
- A common observation is that one must explore a larger neighbourhood to determine the optimal dynamic energy consumption compared to that for optimal execution time. This leads to high partial FPM and FEM construction times. Therefore, if the goal is to construct a locally Pareto-optimal front close to the energy-optimal point, we must design and implement a different algorithm to *ADAPTALEPH* that starts exploring from the low-energy workload distributions. However, this goal is not legitimate in HPC since there would be unacceptable performance degradation associated with such locally Pareto-optimal front of solutions.
- While the partial FPM and FEM model construction times for *OpenBLAS DGEMM* application and the total execution time of *ADAPTALEPH* are quite reasonable compared to the execution times of the parallel application, it is not the case for *FFT*. This is because the *Intel MKL FFT* is optimized for only specific problem sizes (power-of-two, prime number). We would explore softwares that provide optimized FFT for all problem sizes.

- There are two approaches to reduce the partial FPM and FEM construction time. One is to increase the granularity/step size between the points in the partial FPM/FEM. However, this may result in inferior locally Pareto-optimal front or in the worse-case situation, no solutions better than load-balanced solution. The other approach is to *a priori* determine if the variation between the speeds and the energies is less than the input tolerance ϵ. In this case, all the available p processors can build p data points in the FPM and FEM in parallel thereby reducing the FPM and FEM construction times by a factor of p. This is another interesting piece of research that we would pursue.

6 | RELATED WORK

In this section, we study briefly research works in three relevant categories. The first category presents dynamic load balancers. The second category deals specifically with works that have proposed data partitioners for self-adaptable applications on heterogeneous platforms. The third category surveys research that present methods solving multi-objective optimization problems (eg, performance, energy, reliability, etc) of scientific applications.

6.1 | Dynamic load balancers

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. The DAG expresses different types of tasks and the data dependencies between them and is created either statically or dynamically. Little information exists on the computational performance and memory utilization of DAG schedulers. They cater to particular classes of applications (sparse, irregular, etc) that are not the target of our work in his paper.

Legrand et al[38] studied mapping of iterative computations onto heterogeneous clusters. The processors employed in the application is assumed to be arranged in a virtual ring. At each iteration, local computations are performed in parallel and some communications (boundary information) take place between consecutive processors in the ring. Several processor pairs share communication links. The authors consider the problem of optimal partitioning the workload in each iteration taking into account the computations and communications so that the total execution time is minimized. They prove the NP-completeness of the problem and design an efficient heuristic. Mahanti and Eager[39] studied different data redistribution policies when processors (or nodes) are added or removed during the execution of a data parallel application in a dynamic heterogeneous environment. To be precise, they study how to change the shape of the partitions when nodes are added or removed without adversely affecting the quality of partitioning (measured by the cost of migration of the redistributed data). In this paper, we do not consider the cost of data migration arising from dynamic partitioning.

Dynamic algorithms, such as task scheduling and work stealing,[40-42] balance the load by moving fine-grained tasks between processors during the execution. They do not require a priori information about execution but may incur large communication overhead due to data migration. They can use static partitioning for the initial step due to its provably near-optimal communication cost, bounded tiny load imbalance, and lesser scheduling overhead.

Dynamic load balancers based on graph partitioners were proposed by Schloegel et al[43] and Catalyurek et al[44] for adaptive scientific computations where two objectives, interprocessor communication and data migration costs, are considered.

6.1.1 | Dynamic data partitioners for heterogeneous platforms

Galindo et al[45] proposed a dynamic load balancing approach to balance the workload of iterative algorithms in dedicated heterogeneous platforms. Before the start of execution of the iterative algorithm, homogeneous distribution of the workload is used. The speeds of the processors is determined after the execution of one iteration. These speeds are used to determine new workload distribution for the next iteration. Martínez et al[46,47] proposed a dynamic load balancing approach to balance the workload of iterative algorithms in heterogeneous dedicated and non-dedicated platforms composed of multiprocessor nodes. Sanjuan-Estrada et al[48] proposed a dynamic load balancing strategy, which determines the number of threads at runtime (at various stages of an application execution) based on two decisions. These are the completed work and the existence of a sleeping thread in the application. The execution of an application starts with one thread. The strategy uses these decisions to determine if a thread needs to be created to maintain load balance at various (predetermined or equidistant) stages of the application. Wang et al[49] present a self-adaptive and parallelized maximum likelihood evaluation (MLE) framework. It consists of a master process and a set of worker processes in a distributed environment where the master is responsible for re-distributing the computing tasks to workers; the workers compute tasks. The goal of the framework is to achieve load balance of workload between the workers. The workload distribution is based on the execution times of the workers. Acosta et al[50] proposed a dynamic load balancing approach to balance the workload of iterative algorithms in homogeneous and heterogeneous multi-GPU platforms. The approach is similar to the efforts presented earlier.

Clarke et al[1] and Lastovetsky et al[2] proposed a data partitioning algorithm for employment in self-adaptable applications due to its low runtime cost. This algorithm does not require as input the full functional performance model (FPM). Unlike algorithms that require construction of full FPMs as a prerequisite, it builds a partial estimate of the FPM and uses it to determine optimal data partitioning satisfying a user-defined accuracy.

However, the proposed algorithm does not take into account the new behaviors arising from the inherent complexities of resource contention and NUMA on modern multicore platforms.

The data partitioning algorithm that we propose in this paper has several noteworthy differences. First, it takes as input a functional performance model and not a constant performance model such as an execution time to determine the workload distribution. Second, the workload distribution output by it may not load-balance the application. Third, it takes into account the new behaviors caused by resource contention and NUMA on modern multicore platforms. Finally, it solves *BOPPE* (and not single objective performance optimization problem) and returns a locally Pareto-optimal front of solutions instead of a single solution.

6.2 | Multi-objective optimization for scientific applications

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

6.2.1 | System-level methods

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

Ge et al[51] presented a runtime system (CPU MISER) based on DVFS that provides energy savings with minimal performance degradation by using a performance model. Huang and Feng[52] proposed an eco-friendly daemon that employs workload characterization as a guide to DVFS to reduce power and energy consumption with little impact on application performance. Mezmaz et al[53] proposed a parallel bi-objective genetic algorithm to maximize the performance and minimize the energy consumption in cloud computing infrastructures. The parameters used in their method are the computation cost of a task (w) and the communication costs between two tasks. The decision variable is the supply voltage (V) of the processor. Energy consumption of computations is modeled as a function of $V^2 \times w$. Fard et al[54] presented a four-objective case study comprising performance, economic cost, energy consumption, and reliability for optimization of scientific workflows in heterogeneous computing environments. The parameters are the computation speeds of the processors and the bandwidths of the communication links connecting a pair of processors. The decision variable is the task assignment or mapping. The energy consumption of computations is modeled as cube-root of clock frequency. Beloglazov et al[55] proposed heuristics that consider twin objectives of energy efficiency and Quality of Service (QoS) for provisioning data center resources. The decision variables are the number of VMs and clock frequencies. The energy consumption is modeled as a linear function of CPU utilization.

Kessaci et al[56] presented a multi-objective genetic algorithm that minimizes the energy consumption, CO2 emissions, and maximizes the generated profit of a cloud computing infrastructure. The parameters are the execution time of an application, the number of processors used in the execution of an application, and the deadline for completion of the application. The decision variable is the arrival rate. The energy consumption is calculated as a product of execution time and power consumption, which is modeled using the formula $\alpha \times f^3 + \beta$, where f is the clock frequency. Durillo et al[57] proposed a multi-objective workflow scheduling algorithm that maximizes performance and minimizes energy consumption of applications executing in heterogeneous high-performance parallel and distributed computing systems. A machine is characterized using nine parameters (from technology(nm) to TDP). They study the impact of different decision variables, ie, number of tasks, number of machines, DVFS levels, static energy, and types of tasks. The execution time and energy consumption are predicted using neural networks.

Zhang and Chang[58] presented a DVFS scheduler that makes sure the multiple user applications executing on multicores in clouds meet their SLA requirement, which is the specific allowed performance loss. Kołodziej et al[59] proposed multi-objective genetic algorithms that aim to maximize performance and energy consumption of applications executing in green grid clusters and clouds. The performance is modeled using computation speed of a processor. The decision variable is the DVFS level. Energy consumption is modeled using the equation, $\gamma \times V^2 \times f \times t_e$, where γ is a constant for a processor, V is the supply voltage, f is the clock frequency, and t_e is the estimated completion time. Sundriyal and Sosonkina[60] presented a runtime system that performs both processor and DRAM frequency scaling and demonstrate total energy savings with minimal performance loss.

Inadomi et al[61] and Gholkar et al[62] considered the fluctuations in performance arising from manufacturing and thermal variations and propose approaches that take into account these variations to assign jobs to machines, which have a specified power budget. In our work, we consider the variations in performance caused by severe resource contention and NUMA inherent in modern multicore platforms during the execution of highly multithreaded scientific data-parallel applications.

6.2.2 | Application-level methods

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

Intra-node Methods: The work of Freeh et al[30] is an intra-node optimization method that analyzes the performance-energy trade-offs of serial and parallel applications on a cluster of DVFS-capable AMD nodes. They use three parameters in their study, ie, β, which compares the

application slowdown to the CPU slowdown; *memory pressure*, which is determined using hardware performance counters such as memory of operations retired and L2 cache misses; and *slack*, which predicts communication bottlenecks. Ahmad et al[31] formulated a bi-objective optimization problem of power-aware scheduling of tasks onto heterogeneous and homogeneous multicore processor architectures. The twin objectives in their problem formulation are minimization of energy consumption and the makespan of computationally intensive scientific problems. Their approach aims to achieve intra-node optimization by considering node-level parameters such as DVFS, computational cycles, and core architecture type in their optimization problem. They mention a solution that combines a classical game-theoretic approach and Karush-Kuhn-Tucker (KKT) conditions to simultaneously optimize both the objectives. Choi et al[63] presented an energy roofline model based on the time-based roofline model.[64] Choi et al[65] extended the roofline model by adding an extra parameter, *power caps*, to their execution time model. These two works[63,65] presented an intra-node optimization approach to study the performance-energy trade-offs. The work of Balaprakash et al[33] is an intra-node optimization approach that explores trade-offs among power, energy, and performance using various application-level tuning parameters such as number of threads and hardware parameters such as DVFS.

Intra-node and Inter-node Methods: Subramaniam and Feng[32] used multi-variable regression to study the performance-energy trade-offs of the high-performance LINPACK (HPL) benchmark. Their model contains four parameters, ie, N, the problem size, NB; the block size, P, Q; the rows and columns, respectively, of the process grid. They study performance-energy tradeoffs using the following decision variables separately: (a) threads, (b) number of nodes, and (c) DVFS levels. Song et al[66] proposed an iso-energy-efficiency model to quantify the improvements in energy consumption of parallel applications. It is based on lines similar to performance iso-efficiency function. The energy improvement (of parallel over sequential application) is studied using pairs of decision variables, ie, level of parallelism, clock frequency, and problem size. Demmel et al[67] presented an intra-node and inter-node optimization approach that studies energy savings at the algorithmic level. The performance is modeled as a linear function of parameters representing costs of computations and communications. The energy consumption is modeled as a linear function of parameters representing costs of computations, communications, and leakage (static power). Drozdowski et al[68] proposed a concept called an iso-energy map, which represent points of equal energy consumption in a multi-dimensional space of system and application parameters. They use iso-energy maps to study performance-energy trade-offs. Marszalkowski et al[69] analyzed the impact of memory hierarchies on time-energy trade-off in parallel computations, which are represented as divisible loads. They represent execution time and energy by two linear functions on problem size, one for in-core computations and the other for out-of-core computations.

It should be noted the surveyed works do not consider workload distribution as a decision variable. In this work, we propose an inter-node optimization method for self-adaptable applications that is based on this single decision variable. We show using experiments on a modern multicore CPU (see introduction) that the speed and dynamic energy functions of problem size are highly non-linear and non-convex.

7 | CONCLUSION

Self-adaptability is a highly preferred feature in HPC due to several reasons, two of the important ones being the variation of the computational load during the evolution of a solution and the other the dynamic underlying execution environment. A crucial building block of a self-adaptable application is a data partitioning algorithm, which must possess several essential qualities. It must take into account the real-life behavior of applications executing on the platform by employing realistic computation and communication models of performance and energy, must exhibit low practical runtime and memory costs compared to that of the data-parallel application in which it is applied, and must minimize the cost of data redistribution arising from dynamic partitioning.

On modern platforms composed of multicore CPUs, the data partitioning algorithms must address a formidable challenge posed by the new inherent complexities that have been introduced due to severe resource contention and NUMA. Innovative model-based methods and data partitioning algorithms have been proposed that address the challenge. However, these algorithms take as input full functional performance and energy models (FPM and FEM), which have prohibitively high model construction costs. Therefore, they are not suitable for employment in self-adaptable applications, which are executed in dynamic environments where the number of available processors and their performance characteristics can be different for different runs of the same application.

In this paper, we have presented a self-adaptable data partitioning algorithm called *ADAPTALEPH*, which solves the bi-objective optimization problem for performance and energy (BOPPE) on homogeneous clusters of multicore CPUs. Unlike the state-of-the-art solving BOPPE that take as inputs full FPM and FEM, it constructs partial FPM and FEM during its execution using all the available processors. It returns a locally Pareto-optimal set of solutions, which are the heterogeneous workload distributions that achieve inter-node optimization of data-parallel applications for performance and energy.

We experimentally study the efficiency of *ADAPTALEPH* for three data-parallel applications, ie, matrix-vector multiplication, matrix-matrix multiplication, and fast Fourier transform, on a modern multicore CPU and homogeneous clusters of such CPUs. We demonstrated that the locally Pareto-optimal front lies between the load-balanced solution and the globally Pareto-optimal front determined using the full FPM and FEM and that, as the number of points in the partial FPM and FEM functions are increased, it approaches the globally Pareto-optimal front.

We showed that the number of points in the partial FPM/FEM when the output locally Pareto-optimal front of solutions becomes the globally Pareto-optimal front is quite less compared to the number of points in the full FPM/FEM. This suggests that one can considerably reduce the model construction times if one can predict the juncture analytically. However, this is a difficult research problem that we intend to tackle in our future work.

While the partial FPM and FEM model construction times for *OpenBLAS DGEMM* application and the total execution time of *ADAPTALEPH* are quite reasonable compared to the execution times of the parallel application, it is not the case for *FFT*. This is because the *Intel MKL FFT* is optimized for only specific problem sizes (power-of-two, prime number). We would explore softwares that provide optimized FFT for all problem sizes.

However, the partial FPM and FEM construction times can be reduced by a large order of magnitude if all the available processors build the data points in these functions in parallel based on a priori knowledge that the variation between the speeds and energies of execution of a problem size between the processors is within user-specified tolerance. This is also one line of research that we would pursue.

We also observed that one must explore a larger neighborhood (in the vicinity of load-balanced workload distribution) to determine the optimal dynamic energy consumption compared to that for optimal execution time. This leads to high partial FPM and FEM construction times. Therefore, if the goal is to construct a locally Pareto-optimal front close to the energy-optimal point, we must design and implement an algorithm different from *ADAPTALEPH* that starts exploring from the low-energy workload distribution points. However, this goal is not legitimate in HPC since there would be unacceptable performance degradation associated with such locally Pareto-optimal front of solutions.

ACKNOWLEDGMENT

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

ORCID

Ravi Reddy Manumachu http://orcid.org/0000-0001-9181-3290

REFERENCES

1. Clarke D, Lastovetsky A, Rychkov V. Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms. *Parallel Process Lett*. 2011;21(2):195-217.
2. Lastovetsky A, Reddy R, Rychkov V, Clarke D. Design and implementation of self-adaptable parallel algorithms for scientific computing on highly heterogeneous HPC platforms. CoRR. 2011. https://arxiv.org/abs/1109.3074
3. Williams RD. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurr Pract Exp*. 1991;3(5):457-481.
4. Walshaw C, Cross M, Everett MG. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J Parallel Distributed Comput*. 1997;47(2):102-108.
5. Arulananthan A, Johnson SP, McManus K, Walshaw C, Cross M. A generic strategy for dynamic load balancing of distributed memory parallel computational mechanics using unstructured meshed. In: *Parallel Computational Fluid Dynamics: Recent Developments and Advances Using Parallel Computers*. Amsterdam, The Netherlands: Elsevier Science BV; 1998:43-50.
6. Reddy R, Lastovetsky A, Alonso P. HeteroPBLAS: a set of parallel basic linear algebra subprograms optimized for heterogeneous computational clusters. *Scalable Comput Pract Exp*. 2009;10(2):201-216.
7. Patel CD, Bash CE, Sharma R, Beitelmal M, Friedrich R. Smart cooling of data centers. Paper presented at: ASME 2003 International Electronic Packaging Technical Conference and Exhibition; 2003; Maui, HI.
8. Bash C, Forman G. Cool job allocation: measuring the power savings of placing jobs at cooling-efficient locations in the data center. Paper presented at: USENIX Annual Technical Conference; 2007; Santa Clara, CA.
9. Wang L, Von Laszewski G, Dayal J, He X, Younge AJ, Furlani TR. Towards thermal aware workload scheduling in a data center. In: 10th International Symposium on Pervasive Systems, Algorithms, and Networks (I-SPAN); 2009; Kaoshiung, Taiwan.
10. Sarood O, Gupta A, Kale LV. Temperature aware load balancing for parallel applications: preliminary work. Paper presented at: IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW); 2011; Anchorage, AK.
11. FFTW. FFTW: A fast, free C FFT library. 2016. http://www.fftw.org/
12. Lastovetsky A, Reddy R. New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. *IEEE Trans Parallel Distributed Syst*. 2017;28(4):1119-1133.
13. OpenBLAS. OpenBLAS: an optimized BLAS library. 2016. http://www.openblas.net/
14. PFFTW. Parallel FFTW. 2016. http://www.fftw.org/fftw2_doc/fftw_4.html
15. Manumachu RR, Lastovetsky A. Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. *IEEE Trans Comput*. 2018;67(2):160-177.
16. Reddy R, Lastovetsky A. Parallel data partitioning algorithms for optimization of data-parallel applications on modern extreme-scale multicore platforms for performance and energy. *J Parallel Distributed Comput*. 2017.
17. Lastovetsky AL, Reddy R. Data partitioning with a realistic performance model of networks of heterogeneous computers. Paper presented at: 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2004; Santa Fe, NM.
18. Lastovetsky A, Reddy R. Data partitioning with a functional performance model of heterogeneous processors. *Int J High Perform Comput Appl*. 2007;21(1):76-90.
19. Van De Geijn RA, Watts J. SUMMA: scalable universal matrix multiplication algorithm. *Concurr Pract Exp*. 1997;9(4):255-274.
20. HCL. HCLWattsUp: API for power and energy measurements usingWattsUp Pro Meter. 2016. http://git.ucd.ie/hcl/hclwattsup
21. Lastovetsky A, Reddy R. On performance analysis of heterogeneous parallel algorithms. *Parallel Comput*. 2004;30(11):1195-1216.
22. Lastovetsky A, Szustak L, Wyrzykowski R. Model-based optimization of MPDATA on Intel Xeon Phi through load imbalancing. CoRR. 2015. https://arxiv.org/abs/1507.01265

23. Lastovetsky A, Szustak L, Wyrzykowski R. Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. *IEEE Trans Parallel Distributed Syst*. 2017;28(3):787-797.
24. Lastovetsky A, Twamley J. Towards a realistic performance model for networks of heterogeneous computers. In: *High Performance Computational Science and Engineering*. New York, NY: Springer Science+Business Media; 2005.
25. Lastovetsky A, Reddy R. Data distribution for dense factorization on computers with memory heterogeneity. *Parallel Comput*. December 2007;33(12):757-779.
26. Clarke D, Lastovetsky A, Rychkov V. Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models. In: *Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 - September 2, 2011, Revised Selected Papers, Part I*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012.
27. Smolarkiewicz PK, Grabowski WW. The multidimensional positive definite advection transport algorithm: nonoscillatory option. *J Comput Phys*. 1990;86(2)355-375.
28. Miettinen KM. *Nonlinear Multiobjective Optimization*. Dordrecht, The Netherlands: Kluwer Academic Publisher; 1998.
29. Talbi E-G. *Metaheuristics: From Design to Implementation*. vol 74. Hoboken, NJ: John Wiley & Sons; 2009.
30. Freeh VW, Lowenthal DK, Pan F, et al. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans Parallel Distributed Syst*. 2007;18(6).
31. Ahmad I, Ranka S, Khan SU. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. Paper presented at: 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2008; Miami, FL.
32. Subramaniam B, Feng WC. Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing; 2010; Hangzhou, China.
33. Balaprakash P, Tiwari A, Wild SM. Multi objective optimization of HPC Kernels for performance, power, and energy. In: *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation: 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013. Revised Selected Papers*. Cham, Switzerland: Springer International Publishing; 2014;239-260.
34. Ding L, Zeng S, Kang L. A fast algorithm on finding the non-dominated set in multi-objective optimization. In: The 2003 Congress on Evolutionary Computation; 2003; Piscataway, NJ.
35. Gautier T, Besseron X, Pigeon L. KAAPI: a thread scheduling runtime system for data flow computations on cluster of multi-processors. In: Proceedings of the 2007 International Workshop on Parallel Symbolic Computation; 2007; London, Canada.
36. Augonnet C, Thibault S, Namyst R, Wacrenier P-A. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurr Pract Exp*. 2011;23(2):187-198.
37. Bosilca G, Bouteiller A, Danalis A, Herault T, Lemarinier P, Dongarra J. DAGUe: a generic distributed DAG engine for high performance computing. Paper presented at: IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW); 2011; Anchorage, AK.
38. Legrand A, Renard H, Robert Y, Vivien F. Mapping and load-balancing iterative computations. *IEEE Trans Parallel Distributed Syst*. 2004;15(6):546-558.
39. Mahanti A, Eager DL. Adaptive data parallel computing on workstation clusters. *J Parallel Distributed Comput (JPDC)*. 2004;64(11):1241-1255.
40. Linderman MD, Collins JD, Wang H, Meng TH. Merge: a programming model for heterogeneous multi-core systems. *ACMSIGOPS Oper Syst Rev*. 2008;42(2):287-296.
41. Quintana-Ortí G, Igual FD, Quintana-Ortí ES, Van de Geijn RA. Solving dense linear systems on platforms with multiple hardware accelerators. *SIGPLAN Notices*. 2009;44(4):121-130.
42. Augonnet C, Thibault S, Namyst R. Automatic calibration of performance models on heterogeneous multicore architectures. In: *Euro-Par 2009 - Parallel Processing Workshops: HPPC, HeteroPar, PROPER, ROIA, UNICORE, VHPC, Delft, The Netherlands, August 25-28, 2009, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2009.
43. Schloegel K, Karypis G, Kumar V. A unified algorithm for load-balancing adaptive scientific simulations. In: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing ; 2000; Dallas, TX.
44. Catalyurek UV, Boman EG, Devine KD, Bozdag D, Heaphy R, Riesen LA. Hypergraph-based dynamic load balancing for adaptive scientific computations. Paper presented at: 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2007; Long Beach, CA.
45. Galindo I, Almeida F, Badía-Contelles JM. Dynamic load balancing on dedicated heterogeneous systems. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 15th European PVM/MPI Users' Group Meeting, Dublin, Ireland, September 7-10, 2008. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:64-74.
46. Martínez JA, Garzón EM, Plaza A, García I. Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. *J Supercomput*. 2011;58(2):151-159.
47. Martínez JA, Almeida F, Garzón EM, Acosta A, Blanco V. Adaptive load balancing of iterative computation on heterogeneous nondedicated systems. *J Supercomput*. 2011;58(3):385-393.
48. Sanjuan-Estrada JF, Casado LG, García I. Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures. *J Supercomput*. 2011;58(3):376-384.
49. Wang W-J, Chang Y-S, Wu C-H, Kang W-X. A self-adaptive computing framework for parallel maximum likelihood evaluation. *J Supercomput*. 2012;61(1):67-83.
50. Acosta A, Blanco V, Almeida F. Towards the dynamic load balancing on heterogeneous multi-GPU systems. Paper presented at: 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA); 2012; Leganés, Spain.
51. Ge R, Feng X, Feng W-C, Cameron KW. CPU MISER: a performance-directed, run-time system for power-aware clusters. In: International Conference on Parallel Processing (ICPP); 2007; Xi'an, China.
52. Huang S, Feng W. Energy-efficient cluster computing via accurate workload characterization. Paper presented at: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid; 2009; Shanghai, China.
53. Mezmaz M, Melab N, Kessaci Y, et al. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *J Parallel Distributed Comput*. 2011;71(11):1497-1508.

54. Fard HM, Prodan R, Barrionuevo JJD, Fahringer T. A multi-objective approach for workflow scheduling in heterogeneous environments. Paper presented at: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2012; Ottawa, Canada.
55. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener Comput Syst*. 2012;28(5):755-768. Special Section: Energy efficiency in large-scale distributed systems.
56. Kessaci Y, Melab N, Talbi E-G. A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. *Clust Comput*. 2013;16(3):451-468.
57. Durillo JJ, Nae V, Prodan R. Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Gener Comput Syst*. 2014;36:221-236.
58. Zhang Z, Chang JM. A cool scheduler for multi-core systems exploiting program phases. *IEEE Trans Comput*. 2014;63(5).
59. Kołodziej J, Khan SU, Wang L, Zomaya AY. Energy efficient genetic-based schedulers in computational grids. *Concurr Pract Exp*. 2015;27(4):809-829.
60. Sundriyal V, Sosonkina M. Joint frequency scaling of processor and DRAM. *J Supercomput*. 2016;72(4):1549-1569.
61. Inadomi Y, Patki T, Inoue K, et al. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC); 2015; Austin, TX.
62. Gholkar N, Mueller F, Rountree B. Power tuning HPC jobs on power-constrained systems. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT); 2016; Haifa, Israel.
63. Choi JW, Bedard D, Fowler R, Vuduc R. A roofline model of energy. Paper presented at: 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2013; Boston, MA.
64. Williams S, Waterman A, Patterson D. Roofline: an insightful vmisual performance model for multicore architectures. *Commun ACM*. 2009;52(4):65-76.
65. Choi J, Dukhan M, Liu X, Vuduc R. Algorithmic time, energy, and power on candidate HPC compute building blocks. Paper presented at: 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2014; Phoenix, AZ.
66. Song S, Su CY, Ge R, Vishnu A, Cameron KW. Iso-energy-efficiency: an approach to power-constrained parallel computation. Paper presented at: 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2011; Anchorage, AK.
67. Demmel J, Gearhart A, Lipshitz B, Schwartz O. Perfect strong scaling using no additional energy. Paper presented at: 27th IEEE International Parallel Distributed Processing Symposium (IPDPS); 2013; Boston, MA.
68. Drozdowski M, Marszalkowski JM, Marszalkowski J. Energy trade-offs analysis using equal-energy maps. *Future Gener Comput Syst*. 2014;36:311-321.
69. Marszalkowski JM, Drozdowski M, Marszalkowski J. Time and energy performance of parallel systems with hierarchical memory. *J Grid Comput*. 2016;14(1):153-170.

How to cite this article: Reddy Manumachu R, Lastovetsky AL. Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution. *Concurrency Computat Pract Exper*. 2018;e4958. https://doi.org/10.1002/cpe.4958