IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 4, 2021, accepted May 25, 2021, date of publication June 3, 2021, date of current version June 11, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3085774*

# A Proposed Data Partitioning Approach on Heterogeneous HPC Platforms: Data Locality Perspective

**HIND TAHA AL-HASHIMI AND ABDULLAH AHMAD BASUHAIL**
Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
Corresponding author: Hind Taha Al-Hashimi (hthashimi@gmail.com)

**ABSTRACT** We propose a new data partitioning approach to improve the performance of heterogeneous parallel applications in modern high-performance computing (HPC) systems. Existing approaches do not consider an important aspect that has a critical impact on the performance of parallel applications: the method of assigning partitions to each processor so as to minimize the communication cost and hence minimize data movement, which dominates energy and performance cost. Such an aspect for managing data locality is important for a large range of applications. Therefore, to achieve efficient data partitioning, we propose a method for distribution considering this aspect. Our algorithm seeks to minimize execution time by using two models. The first is a fine-grained computational model of heterogeneous processors, which is sufficiently adequate and accurate to guarantee efficient partitioning results that maximize utilization. The second is a communication model of heterogeneous processors to minimize data motion and hide communication overheads. The correctness of our algorithm was analyzed and validated. The complexity of our algorithm is approximately of order $\mathbf{O(p \times \log s + p \times s^2)}$, where $s$ is problem size/steps (where steps is the step size between data points in the computational model of each processor), and $p$ is the number of heterogeneous processors. The experiments were performed on AZIZ supercomputer using two types of applications: an application with no dependency between its partitions, i.e., matrix multiplication, and another one with high dependency between its partitions, i.e., the Jacobi method. The results show the efficiency of our algorithm in improving performance.

**INDEX TERMS** Data partitioning, data distribution, heterogeneous system, performance optimization, load balance, performance model, HPC, data locality, communication cost.

## I. INTRODUCTION

Going deeper into the characteristics of modern heterogeneous systems [1], [2], researchers have found many factors that contribute to making the task of applications development to be more complicated. Some of the factors are issues related to the integration of dissimilar processors on one chip, such as contention for shared resources, e.g., highest-level cache, interconnection network, limited bandwidth of the PCI-E bus, and limited memory of accelerators [3]. All such factors introduce new challenges to the process of designing and optimizing parallel applications for these systems.

Indeed, in parallelization, the design of data partitioning and distribution is one of the most time- and effort-consuming tasks in runtime optimization [4], [5]. Data partitioning is an important composition unit of data access patterns and significantly affects the performance of an application [6]. It refers to the process of partitioning the data into smaller chunks for assignment to different memory spaces for parallelism or optimizing for data locality [7]. There are many methods of data partitioning, and selection of the appropriate scheme depends on the application's requirements as well as the hardware used [5].

In applications where data locality is critical for performance, managing data locality is a fundamental aspect that must be considered in the process of designing a partitioning algorithm, namely the methods of allocating partitions to each processor in order to minimize data transformation. Minimizing data motion means minimization of execution time and energy consumption [8], [9].

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

Traditionally, data partitioning algorithms have been designed based on a realistic performance model that captures the application behavior on modern heterogeneous platforms. The designed performance models have evolved to integrate the most important features of heterogeneous systems, e.g., heterogeneity of architectures and memory structure and the effect of paging [10]. Using such advanced performance models enables partitioning algorithms to find an optimal partition size for each processor, which maximizes the performance of parallel application. State-of-the-art algorithms have shown significant performance improvements in this regard. However, most of them do not consider the communication cost between the processing units. It is well known that in the process of workload partitioning, calculating the communication cost is a critical factor for minimizing data motion and hence improving performance and energy consumption.

In this paper, we propose a data partitioning algorithm that considers the two aspects of managing data locality, contributing to managing locality in the memory hierarchy and between processors by assigning consecutive suitably sized blocks of data structure to successive processors with a minimum communication cost between them. Using two models, specifically a fine-grained computation model of heterogeneous processors and the communication model between the heterogeneous processors, the algorithm works towards a more optimal solution.

The rest of the paper is organized as follows. In Section II we review the concept and evolution of performance models, the classical method used by partitioning algorithms for finding the optimal distribution in heterogeneous platforms. Then, in Section III, we provide an overview of the partitioning methods classified regarding the existing state-of-the-art algorithms. After that, we discuss the existing limitations in Section IV. In Section V, we present our proposed data partitioning algorithm, the main contribution of this paper. In Section VI, we provide theoretical and experimental analyses and validation of our approach. We conclude in Section VII.

## II. PERFORMANCE MODEL FOR OPTIMAL PARTITIONING IN HETEROGENEOUS PLATFORMS

In modern HPC platforms, performance models of processors are very common and are efficient for the optimization of parallel applications [11]. These models enable designers and developers to gain insight into the optimal means of mapping their applications to parallel architectures with high accuracy [12].

Performance-model-based data partitioning is where offline profiling information on the performance of processors is used as an input to data partitioning algorithms to find optimal distribution on HPC systems. In the performance model of a processor, the speed is represented by a positive number, and hence the performance model is called the constant performance model (CPM) [3]. In modern HPC platforms where the systems are highly heterogeneous, a performance model where the processor speed is represented by a function of the problem size is usually used since it is more realistic. It combines important features of heterogeneous systems, such as heterogeneity of architecture and memory structure and paging effects, therefore leading to an accurate partitioning result. The CPM could be used for medium-sized applications running on single-core systems [13]. In this section, we briefly review the evolution of a performance model used by data partitioning algorithms for performance optimization in heterogeneous HPC platforms.

We start with the CPM, which is the simplest model for data partitioning. It uses a constant number to represent the speed of each processor, e.g., normalized processor speed, normalized cycle time, task computation time, and average execution time. A number of methods [14]–[16] use this model for partitioning and distribution across processors. The partitioning is performed such that the volume of each partition is proportional to the speed of the assigned processor. All parallel algorithms use the CPM for their data distribution assuming the following [11]: first, the processors speed does not depend on the size of the problem, and second, the processors are independent of each other, so their performance can be measured individually.

Unfortunately, such a model does not work well with modern systems that have heterogeneous architecture unless each partition is fit into the memory of the assigned processor [17]. Therefore, to achieve accurate partitioning and distribution on these systems, a suitable model that can capture different aspects of heterogeneous platforms, for example memory heterogeneity, is required.

Lastovetsky and Reddy [18] introduced the concept of a functional performance model (FPM) and used it as an input to their data partitioning algorithm for a heterogeneous network of uniprocessors. The FPM is used as a key input for load-balance-based data partitioning algorithms. In this model, the speed of processors depends on the problem size, and it is represented by a continuous function versus problem size. The problem size means ''the amount of data stored and processed by the algorithm, instead of the number of basic computations, since the former one does influence the speed of the processor'' [19]. In addition to the continuity, the shape of the speed function is assumed to have specific characteristics. It must be monotonically decreasing or monotonically increasing, concave, then monotonically decreasing. The performance profile of applications must satisfy these conditions so the partitioning algorithm can return the best result to load-balance application. The continuity and the shape of the graph guarantee that any straight line passing through the origin will intersect the graph, and there will be only one intersection point per line respectively. The introduced FPM encapsulates many features of both the architecture (for example heterogeneity of processors, memory hierarchy, and the effect of paging) and the application.

The FPM of [19] is not accurate in a situation where heterogeneous processing elements exist together in one node, and thus, contend for shared resources, so Z. Zhong *et al.* [11] extended it to heterogeneous multicore and multi-GPU

platforms, e.g., the hybrid node. They modeled multi-GPU multi-core platforms as a set of abstract processing units. Each processing elements executing the same kernel of the application is represented as one abstract processing unit, e.g., if CPU core (or a group of cores) executing a kernel then it (or they) represented as one abstract processing unit. The GPU and its host CPU core are represented as one abstract processing unit.

Lastovetsky *et al.* studied the behavior of data parallel applications in homogenous HPC [10], [20], e.g., multicore, and in modern heterogeneous HPC systems [3]. They found that the function shape in the performance profile violate the assumptions of the FPM, due to complexities related to the tight integration of asymmetry processing elements such as contention for shared resources. Due to this deviation, load-balancing algorithms based on the FPM of [18] are not able to return an optimal solution. Accordingly, Lastovetsky *et al.* proposed a new model to tackle the limitation of the FPM. In [3], the authors modified the performance model of [11] by representing the performance model of abstract processors using a discrete function of speed versus problem size. Their algorithm for data partitioning on heterogeneous platforms, which uses this new performance model, is discussed in Section III. B.

## III. DATA PARTITIONING ALGORITHMS FOR HETEROGENEOUS SYSTEM

Data partitioning is unavoidable due to the parallelism inherent in scientific computing platforms. There are load-balance-based techniques and load-imbalance-based techniques. The following section discusses each class independently.

### A. LOAD-BALANCE-BASED TECHNIQUES

On heterogeneous platforms, load-balance-based data partitioning algorithms usually seek to tackle the challenges accompanying the achievement of load balance across heterogeneous architecture, e.g., resource contention or limited memory of accelerators [19].

Load-balancing data partitioning in heterogeneous systems refers to the situation where all processors incorporated in the execution complete their work at the same time:

$$t_1(x_1) \approx t_2(x_2) \approx \ldots\ldots t_p(x_p) \tag{1}$$

where $t_i(x_i)$ is the execution time of processor $p_i$, assigned the partition size $x_i$, i $\in \{1, 2, \ldots p\}$, and $p$ is the number of heterogenous processors. Equation (1) can be expressed as:

$$\frac{x_1}{s_1(x_1)} \approx \frac{x_2}{s_2(x_2)} \approx \ldots\ldots\ldots\ldots \frac{x_p}{s_p(x_p)}. \tag{2}$$

$x_1 + x_2 + \ldots\ldots x_{\mathrm{p}} = n$, where $n$ is the total size of the problem and $s_{\mathrm{i}}$ is the speed of processor $p_{\mathrm{i}}$.

The plot of this distribution, using the number of elements and processor speed, gives a straight line passing the coordinate system through its origin and intersecting the speed functions. This line represents an optimal load-balanced solution for a specific problem size.

Lastovetsky and Reddy [19] proposed functional performance model (FPM)-based data partitioning for applications executing in nodes consisting of uniprocessors (single-core CPUs). The proposed technique attempts to find an approximate solution that is close enough to the exact optimal one. Briefly, their algorithm uses two initial boundary lines, upper line U and lower line L, of the solution space. These lines represent an optimal distribution for the problem size: $\mathrm{n_U} < \mathrm{n}$ and $\mathrm{n_L} > \mathrm{n}$. The algorithm iteratively bisects the angle between lines U and L by the line M and then decides whether M is the new U or L line based on M's coordinates. This procedure continues until an approximation is achieved.

Several partitioning algorithms have been proposed to improve the performance of matrix multiplication in heterogeneous clusters. Some algorithms utilize column-based partitioning to reduce the range of possible solutions [13]. In column-based partitioning, the algorithm distributes the partitions between p processors arranged in columns, each of which consists of a specific number of processors. In [21], the algorithm uses a hybrid model of the CPM and FPM to partition the matrix. It uses the CPM to indicate the width of each column and the FPM to indicate the partition size for processors within each column. However, using the CPM reduces the accuracy of the result since this model is not suitable for partitioning in modern heterogeneous system. The algorithm of [13] uses the FPM-based algorithm of [19] to find the partition size, i.e., rectangle area, for each processor. The algorithm then adopts an approach from [16] to minimize communication volume by calculating the optimum shape and ordering of partitions. S. Tabik *et al.* [5] proposed a workload distribution approach based on a light offline performance model.

DeFlumere *et al.* [22] proposed a solution for partitioning of the matrix for executing matrix–matrix multiplication on two heterogeneous processors that tends to minimize execution time and communication cost. They focused on the shape of the partition to achieve this goal; hence, they adopted a non-rectangular partitioning approach where one of the partitions was a small square (in the corner of the matrix) assigned to the slower processor, and the other large partition was assigned to the faster processor. This approach extended to include three processors, and its optimality was proved [22].

The partitioning algorithm of [19] was used by the authors of [11] for load balancing the distribution across heterogeneous processing units in a hybrid node as well. They adopted a different performance model to accurately measure the performance of heterogeneous processors in the hybrid node, as discussed in Section 2.2.

Other algorithms proposed by [23] and [24] are based on different performance models as an input to predict the execution of an application. Y. Ogata *et al.* [23] investigated load-balancing partitioning by finding an optimal distribution ratio between the CPU and GPUs. Their approach relies on the performance model that captures the behavior of the

CPU–GPU combination in terms of their contribution and predicts the total execution time. C. Yang *et al.* [24] proposed an adaptive optimization framework to find the optimal distribution between the CPU and GPU.

### B. LOAD-IMBALANCE-BASED TECHNIQUES

In the load-imbalance approach, optimal partitioning and distribution may not balance the application across available processors [3], [10], and [20].

Khaleghzadeh *et al.* [3] proposed the first performance-model-based data partitioning algorithm for heterogeneous systems that may not load-balance an application. The following briefly summarizes their approach. Given a set of $p$ discrete speed functions of a problem size, of specific cardinality, where $p$ is the number of available heterogeneous processors, the algorithm is designed to examine all combinations of processors and then select the distribution of workload that leads to minimum execution time. Their algorithm implements several optimizations to avoid examining all possibilities.

## IV. DISCUSSION

In this paper we discuss data partitioning algorithms in heterogeneous systems for performance optimization: load-balance-based partitioning and load-imbalance-based partitioning. The experimental results provided in previous works show significant performance improvement. However, the research in relation to this subject is limited, and hence it is still an open research issue. Most of the previous works do not address the problem of minimizing communication cost, which is an important related concept.

Modern HPC systems have become more parallel and heterogeneous to accommodate the growing demands of performance and energy efficiency [25]. The programming community is facing a great challenge in respect of how to minimize data movement, which is the most dominant factor in performance and energy consumption [6], [8], [26]. The obvious answer is managing data locality. This means managing vertical locality in the memory hierarchy as well as the horizontal locality between processors [27]. In terms of data partitioning algorithms, for a large range of applications, both concepts are important to find an optimal solution. In our literature review, we found that all proposed methods focus on finding the optimal partition size for each processor in heterogeneous systems to avoid paging and thus contribute to managing vertical locality. However, researchers did not consider minimizing the communication cost between processing units, which is critical for minimizing data motion horizontally in the system. In [13] and [28], the partitioning algorithms utilized a method for minimizing communication cost, mainly to improve the performance of matrix multiplication partitioning across a cluster of heterogeneous processing units. However, these approaches are limited and designed for a specific problem.

In this paper we propose a solution that considers the communication cost. Our algorithm works based on an accurate performance model of the heterogeneous system and communication model to find a more optimal solution, which may or may not balance the application. We explain our algorithm in detail in the following section.

## V. PROPOSED RESEARCH SOLUTION

In this section, we propose a solution to tackle data partitioning across heterogeneous architectures, such that overall execution time is minimized by allocating blocks of a suitable size to successive processors with minimum communication cost between them. To achieve this goal the algorithm uses two models: the performance model of each processor in the system and the communication model between these processors. The following sections discuss our approach in detail.

### A. FINE-GRAINED PERFORMANCE MODEL

The performance model (PM) of the processing elements is an important input to our data partitioning algorithm, and its adequateness and accuracy are important to guarantee obtaining optimal results. We utilize the PM proposed by [3], with a modification. In [3], the PM considers the configuration of parallel applications and integrates many important features of heterogeneous architecture. Each processing unit i.e., one or more cores executing one computational kernel is modeled by abstract processor, and accelerator together with its host CPU core is considered as one abstract processing unit. The model is characterized by the following: a) the speed of abstract processors is represented by a discrete function of the problem size, b) the PM is composed of a series of data points stored in a file, c) each point is generated by timing the execution of the application for a given problem size, and d) the performance of processors is measured simultaneously to consider contending for shared resources.

To ensure model adequacy and accuracy, we built it to be fine-grained. For example, to build the performance profile of matrix multiplication in a heterogeneous system, we measured the execution time for problem sizes $x_1 \times n_1, x_2 \times n_1, \ldots, n_1 \times n_1, x_1 \times n_2, x_2 \times n_2, \ldots, n_2 \times n_2, \ldots etc$, where $x_i < n_{\mathrm{j}}$. We chose the step size between data points to be 64 (it is a selective parameter).

### B. THE COMMUNICATION MODEL

In [29], the interconnected network components of the heterogeneous system consisting of P processors are modeled as a completely connected virtual network, where a path between any pair of processors $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ is represented by a single link. The communication model is a matrix that represents the communication cost between any two processors using two parameters: start-up time (latency) $\boldsymbol{T_{ij}}$ and data transmission rate (bandwidth) $\boldsymbol{B_{ij}}$. The time taken to transmit a message of size $m$ bytes between $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ can be represented by:

$$Cmm_{pi-pj} = \boldsymbol{T_{ij}} + \frac{m}{\boldsymbol{B_{ij}}} \tag{3}$$

$\boldsymbol{T_{ij}}$ latency is estimated to be half of the time of a ping-pong operation with a data of size zero.

$\boldsymbol{B_{ij}}$, the bandwidth can be estimated using the ping-pong and ring programs.

If the abstract processor represents an accelerator, then we add additional term $t_{AccH}$ to Equation (3) to include the time of transferring data between the accelerator and the host core, when halo exchange occurs between this abstract processor and others:

$$Cmm_{pi-pj} = \boldsymbol{T_{ij}} + \frac{m}{\boldsymbol{B_{ij}}} + t_{AccH} \tag{4}$$

We use the communication model as an input to the partitioning algorithm to make it consider the communication cost when implementing the partitioning process.

### C. THE PROPOSED DATA PARTITIONING ALGORITHM

Assume we have data size $n$ to be distributed across available heterogeneous processors $p$, and time function of size $m$ for each processor $t_i(x)$, where $i \in \{1, 2, \ldots, p\}$, and $x \in \{1, 2, \ldots, m\}$, and suppose $m$ is equal to $n$ for simplicity. Our algorithm works toward finding suitable partition size $\boldsymbol{d_i}$ for each processor. It follows the fact that ''During execution of parallel application, each processor is computing, communicating, or idling'', so when our algorithm starts assigning a data size to more than one processor it considers the following generalized form for describing the total execution time, $\boldsymbol{t_i}$, of each processor:

$$\boldsymbol{t_i} = \boldsymbol{t_{comp}} + \boldsymbol{t_{comm}} + \boldsymbol{t_{idl}} \tag{5}$$

where $t_{comp}$ is the computation time of the processor, $t_{comm}$ is the communication time with another processor, and $t_{idl}$ is idle time—we assume that there is no idle time, or it is negligible.

The algorithm uses the greedy technique at some points and works iteratively to find the suitable partition size for each processor to minimize the total execution time. It implements 1D partitioning and can be applied to 2D and 3D data structures since the dimensionality can be decreased to 1D.

The algorithm first indicates the processor $\boldsymbol{p_{highest}}$, which is the processor with highest speed at problem size $\frac{n}{p}$.

Then, the algorithm determines whether there is a point in the range $\left\{\frac{n}{p}, \frac{n}{p}+1 \ldots, m\right\}$ at which this processor or any other processor has a faster speed. The data size at the point where the faster processor is indicated is assigned to the partition size of that processor. The partition is labeled as $\boldsymbol{d_{highest}}$ and the processor as $\boldsymbol{p_{highest}}$. The algorithm then initializes each partition $\boldsymbol{d_i}$ of processor $\boldsymbol{p_i}$ with data size equal to 0, i.e., $\boldsymbol{d_i} = 0$.

The algorithm starts tuning the data size for processor $\boldsymbol{p_{highest}}$. It iteratively increases $\boldsymbol{d_{highest}}$ by one as long as $\boldsymbol{p_{highest}}$ continues to represent higher performance than the others.

It continues to increase partition size $\boldsymbol{d_{highest}}$ until another processor $\boldsymbol{p_i}$ with higher performance at any point x $<$ $\boldsymbol{d_{highest}}$ is found. That is, if $\boldsymbol{p_i}$ is found, such that its computation time at x plus communication time with $\boldsymbol{p_{highest}}$ is less than the computation time of $\boldsymbol{p_{highest}}$ at $\boldsymbol{d_{highest}} + 1$, where x + $\boldsymbol{d_{highest}}$ <= n, then the algorithm indicates $\boldsymbol{p_i}$ to be the second higher-performance processor, and begins tuning its partition size $\boldsymbol{d_i}$, starting from $\boldsymbol{d_i}$ = x. The metric used in this step, which is based on Equation (5), to decide which next processor to be assigned a partition, enables the algorithm to find the distribution where the communication overhead is hidden by the computations.

The algorithm continues iteratively in this way until $\sum_{i=1}^{p} \boldsymbol{d_i} = \mathrm{n}$.

The output of the algorithm is the distribution that minimizes the total parallel execution time by assigning partitions of suitable sizes to successive higher-performance processors with lower communication cost between them. The algorithm may load-imbalance the application.

## VI. ANALYSIS OF THE ALGORITHM

### A. THEORETICAL ANALYSIS

In this section we show that our algorithm usually returns the distribution that minimizes the execution time by preserving data locality, minimizing data transfer, and hiding communication overheads. We also calculate the complexity of our algorithm. Suppose there are $P = \{p_1, p_2, p_3 \ldots, p_p\}$ abstract processors and a problem of size $n$ to be distributed across them. The set of time functions $T = \{t_1, t_2, \ldots, t_p\} \in R^+$ represents the time functions of the processors. The cardinality of $t_i$ is $m$. For each data point in the time function, $t_i(n)$, there is a subset $\{t_{i,1}(x_{i,1}), t_{i,2}(x_{i,2}), \ldots, t_{i,s}(n)\}$, where $x_{i,j} \in \mathrm{Z}$, $0 < x_{i,j} < \mathrm{n}$, and $s$ is the cardinality of the subset equal to $\frac{\mathrm{n}}{64}$. The number 64 is selective, and it represents the step size between data points in the subset.

In parallel execution, if the total execution time of n is measured by the longest elapsed time taken by one of the processors, then what is the optimal point in the performance models that can lead to the minimum execution time? How to minimize data motion and overcome with the communication overhead, and finally, how should the total shape of the distribution look?

At load equal balance $\frac{\mathrm{n}}{p}$ distribution, the processor with the highest performance i.e., faster processor, is assigned the smallest possible size it can take, which makes the sum of all chunks $x_{\mathrm{highest},\frac{s}{p}} + x_{\mathrm{i,j}} \ldots + x_{p,j} = n$, where $x_{\mathrm{i,j}}$ represent the chunk size of $p_{\mathrm{i}}$. At this point the processor with the highest performance $p_{highest}$ has the smallest execution time $t_{highest,\frac{s}{p}}(\frac{\mathrm{n}}{p})$ compared to the other contributing processors, namely, $t_{highest,\frac{s}{p}}(\frac{\mathrm{n}}{p}) < t_{i,\frac{s}{p}}\left(\frac{\mathrm{n}}{p}\right)$. This means that the probability that the execution time of the optimal solution is less than $t_{highest,\frac{s}{p}}\left(\frac{\mathrm{n}}{p}\right)$ is zero.

(If n and s are not divisible by p we consider the ceiling value. Sometimes, it requires to adjust the results of $\left\lceil \frac{n}{p} \right\rceil$ to match the position $\frac{\mathrm{s}}{p}$ due to the steps considered between data points).

From the previous, the minimum time $t_{opt,j}(x_{opt,j})$ that represents the execution time of the longest running processor to complete the parallel execution should be the smallest point

in the range $\bigcup_{i=1}^{P} \{t_{i,j=\frac{s}{p}}(x_{i,j=\frac{s}{p}}), t_{i,j=s}(x_{i,j=s})\}$ under three conditions.

First, the sum of all other chunks plus $x_{opt,j}$ is equal to the total problem size, $x_{opt,j} + x_{i,j} + \ldots + x_{u,j} = n$, where $u \in Z$, and $u \leq p$. Second, $t_{opt,j}(x_{opt,j})$ represents the maximum execution time in the current distribution, that is: $t_{i,j}(x_{i,j}) \leq t_{opt,j}(x_{opt,j})$. Third, the size of the other chunks $x_{i,j}, x_{i+1,j}, \ldots, x_{u,j}$ is specified based on their execution time plus communication time with their predecessors.

These three facts guarantee that the indicated parallel execution time is not exceeded by the slower processor, and that the communication overhead is hidden by computations.

We calculated the complexity of our algorithm. Apparently, from Section V. B., Algorithm 1, and the previous analysis, the complexity sum of the main steps is $\cong \mathbf{O}(\mathbf{P} \times \mathbf{log}\, \boldsymbol{s} + \mathbf{P} \times \mathbf{s^2})$. First, chunk of size $\frac{n}{p}$ is assigned to each processor, and the processor with the highest performance at that point is found. This step has a complexity $O(P)$. Finding The smallest execution time in the range $\bigcup_{i=1}^{P} \{t_{i,j=\frac{s}{p}}(x_{i,j=\frac{s}{p}}), t_{i,j=s}(x_{i,j=s})\}$ has a complexity $O(P \times \log \mathbf{s})$. Then, we re-initialize workload size for each processor except the one with highest performance. This step has a complexity $O(P)$. Finally, regarding tuning chunk size, we have nested *for* loops inside *while* loop, therefore the nested *for* loops has a complexity $O(P \times \mathbf{s})$, and the *while* loop has a complexity of $O(\mathbf{s})$. Accordingly, this step has a total complexity equal to $O(P \times \mathbf{s}^2)$.

### B. EXPERIMENTAL ANALYSIS

#### 1) EXPERIMENT PLATFORM

We worked on two nodes of AZIZ supercomputer. The first node was equipped with an NVIDIA Tesla K20® GPU, and the second node was equipped with an Intel® Xeon Phi accelerator. The characteristics of the GPU and the CPU of the first node are illustrated in Table 1 and Table 2, respectively. We composed two platforms from these nodes. Platform 1 consists of four processing units: (1) 23 cores of CPU from the first node (called CPUK20), (2) GPU with its host core, (3) 23 cores of CPU from the second node (called CPUPHI), and (4) Xeon Phi accelerator with its host core.

Platform 2 consists of three processing units: (1) CPUK20, (2) CPUPHI, and (3) Xeon Phi accelerators with its host core.

#### 2) HETEROGENEOUS APPLICATION

We experimented with two types of applications: one with no dependency between its partitions (i.e., matrix multiplication) and the other with high dependency between its partitions (i.e., the Jacobi method). Both applications were configured to run on previously defined platforms.

The time functions composing the performance profile of each application were built simultaneously on the specified platform. For matrix multiplication, we worked on Platform 1. We used heterogeneous matrix multiplication with

**Algorithm 1** The Proposed Data Partitioning Algorithm for More Optimal Distribution for Performance Maximization

*HeterogeneousDataPartitioning(n, ProcNum, s, T, Comm)*
**INPUT:**
$\boldsymbol{n}$: *Problem size* $\in Z > 0$
***ProcNum***: *Number of processors* $\in Z > 0$
**'s**: *is* $\frac{n}{stepsize}$.
$\boldsymbol{T} = \{\boldsymbol{T_0}, \ldots, \boldsymbol{T_{ProcNum-1}}\}$: *time function, where* $T_i = \{(x_{ij}, t_{ij}) | i \in [0, ProcNum\text{-}1), j \in [0,s), x_{ij} \in Z > 0, t_{ij} \in R^+ > 0\}$
$\boldsymbol{Comm} = \{\boldsymbol{Comm_0}, \ldots, \boldsymbol{Comm_{ProcNum-1}}\}$ : *Communication model, where* $Comm_i = \{(pid, \boldsymbol{t_{ij}}) | i \in [0, ProcNum\text{-}1), j \in [0, ProcNum\text{-}2), pid \in Z > 0, t_{ij} \in R^+ > 0\}$
**OUTPUT:**
$\boldsymbol{D} = \{\boldsymbol{d_1}, \ldots, \boldsymbol{d_u}\}$ *partition sizes* $d_i$ *for* $p_i$, *where* $i \in [0, u]$, $u \leq P$
**START**
*FIND THE HIGHEST PERFORMANCE PROCESSOR AT* $\frac{n}{p}$.
*1. for* $i = 1$ *to ProcNum*
*2.* $d_i = \frac{n}{p}$
*3.* $HighestPoint = min_{i=0}^{P-1} t_{i,\frac{s}{p}}(\frac{n}{p})$ *// minimum execution time at* $(\frac{n}{p})$
*4.* $highest = i$,
*5.* $d_{highest} = \frac{n}{p}$
*FIND THE SMALLEST EXECUTION TIME IN THE RANGE* $\{\frac{s}{p}, s\}$ *IN THE TIME FUNCTIONS*
*6. for* $i = 1$ *to ProcNum*
*7.* *for* $j = \frac{s}{p} + 1$ *to s*
*8.* *If* $t_{ij}(x_{ij}) < HighestPoint$
*9.* $HighestPoint = t_{ij}(x_{ij})$
*10.* $highest = i$,
*11.* $d_i = x_{ij}$
*RE-INITIALIZE PARTITIONS SIZES* $d_i$ *with* 0 *EXCEPT* $d_{highest}$
*12. for* $i = 0$ *to ProcNum-1*
*13.* $d_i = 0, \forall i \in \{1, \ldots, ProcNum\}$- *highest*
*14.* $sum = d_{highest}$
*TUNING PARTITIONS SIZE UNTIL SUM BECOMES EQUAL TO n*
*15.* $high = HighestPoint$
*16. while* $sum \neq n$
*17.* *for* $i = 1$ *to ProcNum*
*18.* *for* $j = 1$ *to s*
*19.* *if* $t_{ij}(x_{ij}) + comm_i(highest) < high$
*20.* $d_i = x_{i,j}$.
*21* $high = t_{ij}(x_{ij})$
*22.* $highest = i$
*23.* *else*
*24.* $d_{highest} = d_{highest} + 1$
*25.* $high = \boldsymbol{t}_{highest,\boldsymbol{j}}(d_{highest} + 1)$
*26. End-while*
***END***

**TABLE 1.** Specifications of NVIDIA K20m GPU.

| TECHNICAL SPECIFICATIONS | |
|---|---|
| Number of cores | 2496 |
| Clock rate | 705 MHz |
| Memory clock rate: | 2600 MHz |
| Memory bus width: | 320 bits |
| L2 cache size: | 1310720 bytes |
| TDP | 225 W |
| Memory size per board (GDDR5) | 5 GB |
| Bandwidth | 208 GB/s |

**TABLE 2.** Specifications of Intel Xeon CPU E5-2695 v2 2.40 GHz.

| TECHNICAL SPECIFICATIONS | |
|---|---|
| CPUs | 24 |
| Cores per socket | 12 |
| Sockets | 2 |
| NUMA nodes | 2 |
| CPU MHz | 2399.842 |
| L1d cache | 32 K |
| L1i cache | 32 K |
| L2 cache | 256 K |
| L3 cache | 30720 K |

four high-performance kernels, each using Intel Math Kernel Library MKL DGEMM to work on CPUs, GPU, and Xeon Phi accelerator. As mentioned previously, we considered the two CPUs in all platforms as two dissimilar processors.

In the performance profile, each time function consists of a discrete set of fine-grained data points. Namely, for each problem size $n^2$, there is a subset $\{x_1 \times n, x_2 \times n, \ldots, n \times n\}$, the execution time of which is also measured. We repeated the measurements multiple times to ensure the reliability of the results.

To prove the optimality of our partitioning results, we used the matrix multiplication application, as there is no dependency between partitioned data, to compare the results returned by our algorithm with those of the algorithm in [3] (the algorithm of [3] is available in [30]). We used the fine-grained performance models as inputs to both algorithms. We performed the comparison using four arbitrary problem sizes. Figure 1 shows the partitioning results of our algorithm, and Figure 2 shows those of the algorithm in [3]. The results returned by both algorithms are identical. The exhaustive search adopted by the algorithm of [3] returns an optimal solution that minimizes the total execution time of parallel execution. This comparison proves that our algorithm returns the optimal solution. Figure 3 illustrates that the results of partitioning returned by both algorithms give identical estimated total execution time. However, the actual execution times of the application when distributed using the results obtained were similar as illustrated in Figure 4 (we executed the application multiple times with the same distribution). Because of the nature of modern HPC represented in the tight integration of its processors, the performance profile of the application has different shapes of fluctuations.

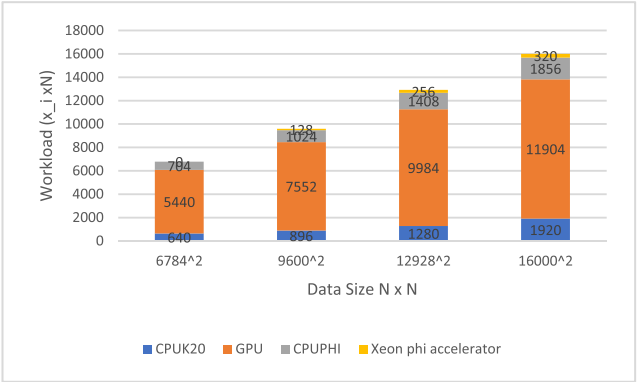


**FIGURE 1.** Result of data partitioning returned by our proposed algorithm using fine-grained performance models. The performance models were built for heterogeneous matrix multiplication on Platform 1.

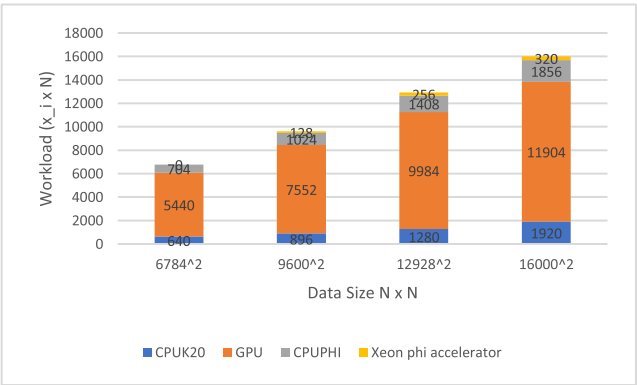


**FIGURE 2.** Result of data partitioning returned by algorithm of [3] using fine-grained performance models. The performance models were built for heterogeneous matrix multiplication on Platform1.

Each time the application is executed, the variations in these fluctuations change within specific range. This explains the reason behind the similarity in actual execution time even if the estimated execution time identical or differed slightly. Moreover, this indicates that the optimal partitioning result for a problem size can be represented by a specific range of partitions sizes between heterogenous processors and not only by a single solution. Our experiment results support this observation.

For the Jacobi method, we worked on Platform 2. We used the benchmark in [31] with some modifications to implement heterogeneous distribution on Platform 2. In the application, the square matrix is sliced vertically, since the benchmark is built with Fortran (i.e., the matrix is stored in column-major order). In the communication model, the communication time for halo exchange between the Xeon Phi processor and other processors includes the transfer time between the accelerator and the host core plus the transfer time between the host core and the target processor. Figure 5 shows a comparison of the partitioning when the communication cost is considered and when it is not considered. The experimental results illustrate that the transfer time between Xeon phi accelerator and the

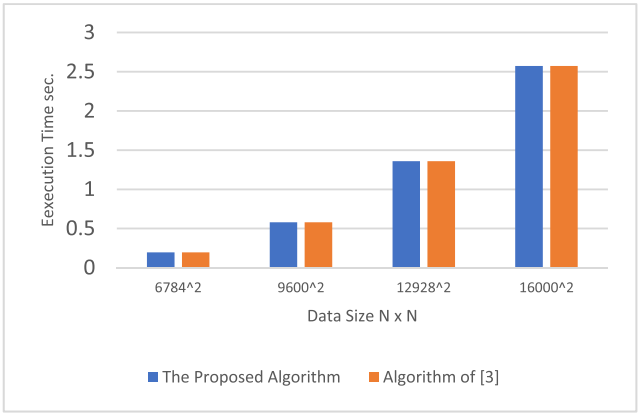


**FIGURE 3.** Estimated execution time of heterogeneous matrix multiplication for Platform1 using the distribution returned by our proposed algorithm and that returned by the algorithm of [3].

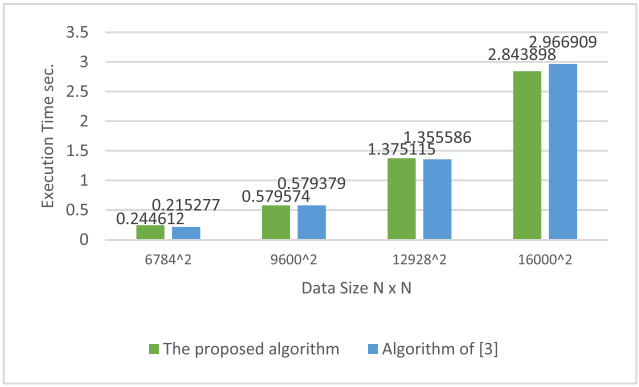


**FIGURE 4.** Total execution time of heterogeneous matrix multiplication for Platform1. The application is executed using the distribution returned by our proposed algorithm and that returned by the algorithm of [3].

core host is the dominant time. This problem occurs especially for Xeon phi accelerators. GPU uses two engines for optimizing data transfer between GPU and the host core [3]. Figure 6 shows a comparison between the execution time when distributing based on computation and communication models and when distributing using the computation model only. The influence of data transfer latency between the accelerators and the core host when halo exchange occurs is obvious in the execution when we partition with no consideration to the communication cost. It is increases as the data size increases, which may lead to losing the performance benefits of the accelerators and reducing the overall performance of the application. Our algorithm considers the communication cost by using the metric discussed in section V.C., which based on Equation 5, to calculate suitable partition size for each processor. It prevents assigning a partition to a processor whose total execution time (i.e., computation time and communication time) exceeds the execution time of its predecessors. This way it can hide the communication overheads effectively, and hence, improve the application performance. The percentage improvement becomes more obvious. In fact, it increases as the data size increases.

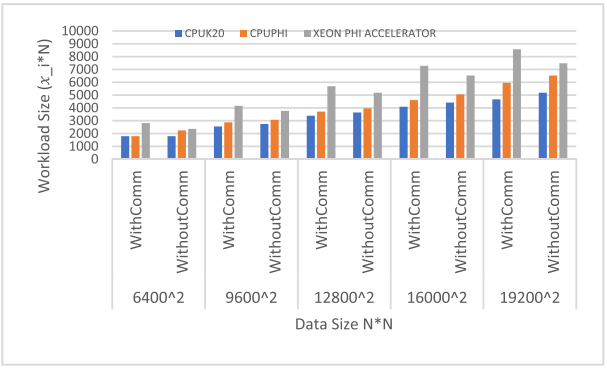


**FIGURE 5.** Comparison of the distribution of the Jacobi method, on Platform 2 when using both computation and communication models of processing elements and when using the computation models only.

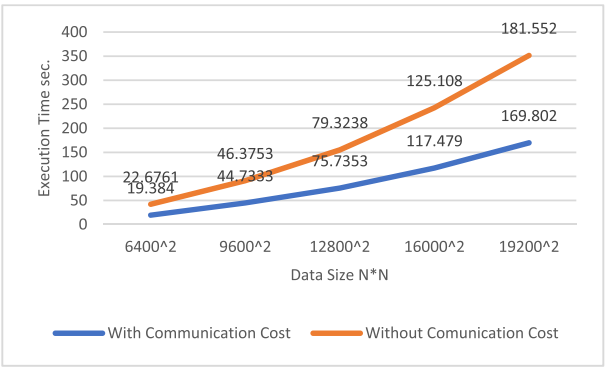


**FIGURE 6.** Total execution time of the Jacobi method, on Platform 2 when our partitioning algorithm considers computation models and communication models of processors and when it considers computation models only.

### 3) FFT

The authors of [3] provide Fast Fourier Transform (FFT) performance functions of CPU, GPU, and Xeon phi accelerators in [30] of the platform on which they performed their experiments. The performance functions are coarse-grained. We used those performance functions as inputs to our algorithm. We utilized four arbitrary problem sizes and compared the partitioning results of our algorithm with those returned by the algorithm of [3]. Figures 7 and 8 illustrate that the distribution results returned by our algorithm and that returned by the algorithm of [3] were similar. The estimated execution time of the application after partitioning is illustrated in Figure 9. Based on our previous observations and experiment results, we expect that when the application is executed using the results obtained by both algorithms, the total execution time will be similar even if the distribution results were not identical. This comparison with the exhaustive-search based algorithm of [3], which guarantees returning the optimal solution, proves that our algorithm also returns the optimal solution that minimizes the parallel execution time.

### 4) FINE-GRAINED PERFORMANCE MODEL VS. COARSE-GRAINED PERFORMANCE MODEL

To illustrate the importance of using fine-grained performance model in finding the optimal partitioning results,

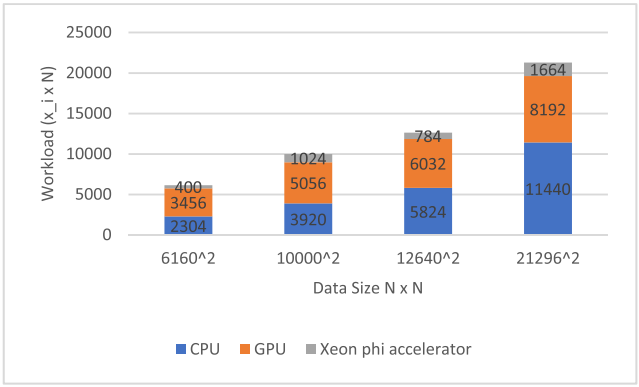


**FIGURE 7.** Result of the data partitioning returned by our proposed algorithm using FFT performance functions of the platform used by [3].

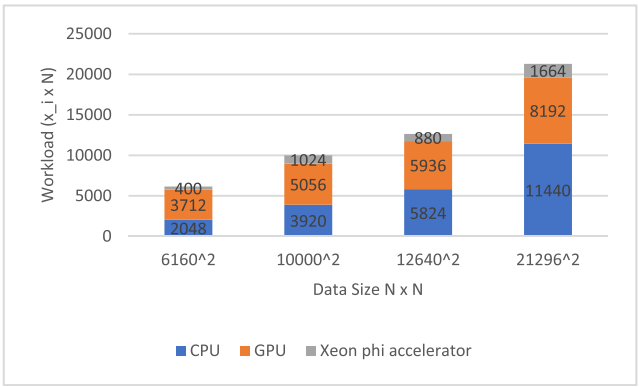


**FIGURE 8.** Result of the data partitioning returned by the algorithm of [3] using FFT performance functions of the platform used by [3].

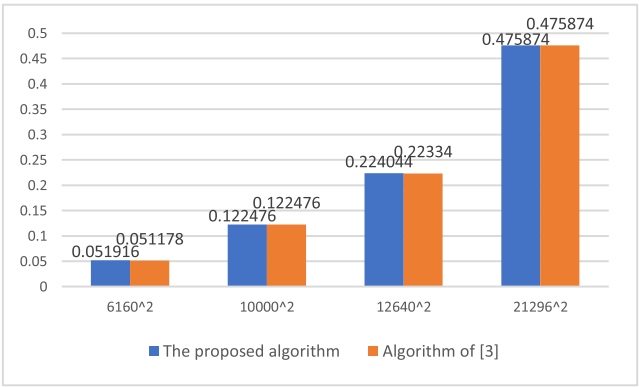


**FIGURE 9.** Estimated execution time of heterogeneous FFT for the platform used by [3]. The estimated execution time using the distribution returned by our proposed algorithm and that returned by the algorithm of [3].

we compared the results obtained by our algorithm when using fine-grained performance model and when using coarse-grained performance model. We utilized the coarse-grained performance profile of the matrix multiplication application. The coarse-grained models are built on Platform 1 using data sizes $N^2$ ranging from $(64 \times 100)^2$ to $(64 \times 300)^2$.

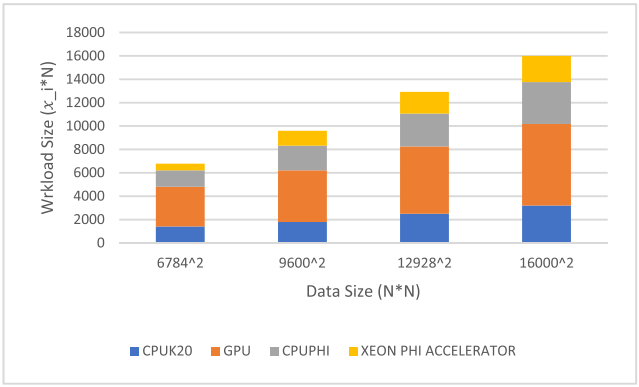


**FIGURE 10.** Workload distribution (by our algorithm) of matrix multiplication based on coarse-grained performance models. The performance models are built on Platform1.

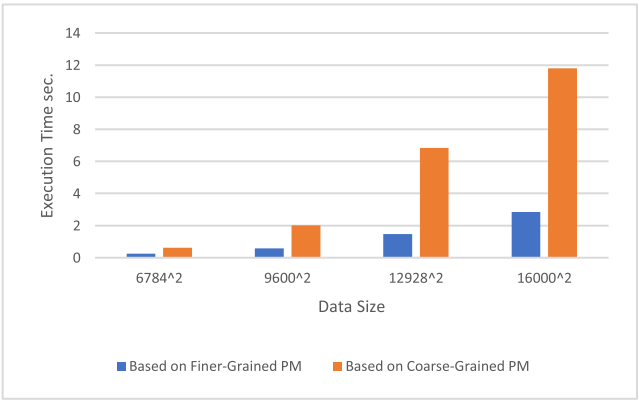


**FIGURE 11.** Comparison of the total execution time on Platform 1 when the partitioning of matrix multiplication is done based on fine-grained performance models versus coarse-grained performance models.

Figure 10 shows the results of distributing matrix multiplication on Platform 1 using our proposed algorithm with the coarse-grained performance model. The results in Figures 1 and 10 illustrate that the distribution shapes vary. This difference can greatly affect the result of performance optimization. Figure 11 shows the comparison between the execution times when using the distribution obtained in Figure 1, which is based on the fine-grained performance model, and when using the coarse-grained performance-models-based distribution. It illustrates that both accuracy and adequateness of the performance models are required to guarantee an optimal solution that effectively optimizes performance.

## VII. CONCLUSION

HPC is commonly used and widely available. It is utilized to predict the weather and climate in environmental studies, optimize fuel efficiency (e.g., for automobiles in manufacturing), and interact with robots and smartphones for artificial intelligence applications, among other functions. The common factor across these uses is the need for high performance. In this paper we have discussed the concept

of data partitioning in heterogeneous HPC platforms from the perspective of data locality, which is an important aspect contributing to performance optimization.

Traditionally, data partitioning algorithms are based on an accurate performance model; we have briefly reviewed the evolution of the performance models used by data partitioning algorithms for performance optimization in modern HPC platforms. Regarding existing studies, we discussed approaches to find the optimal data partitioning, namely, load-balance-based and load-imbalance-based algorithms.

Despite the existing techniques, the problem of data partitioning in heterogeneous HPC platforms remains an open research issue. Most existing approaches do not consider the communication cost between processing elements, which is important because of its significant impact on finding the optimal partitioning. Accordingly, we have proposed a solution that considers this important aspect of managing data locality. Our algorithm uses two models and works toward the optimal partitioning results. The first model is the fine-grained performance model of each processor in the system which is sufficiently adequate and accurate to guarantee efficient partitioning results. The second model is the communication model, which represents the communication cost between each pair processors.

We have provided theoretical and experimental analyses to analyze and validate our approach. The time complexity of our algorithm is $\mathbf{O}\left(\mathbf{P} \times \mathbf{log}\, \mathbf{s} + \mathbf{P} \times \mathbf{s^2}\right)$, where s is $\frac{problem\ size}{steps}$ (where steps is the step size between data points in the discrete time function of each processor), and P is the number of heterogeneous processors. We experimented with two types of applications: matrix multiplication, in which there is no dependency between partitions, and the Jacobi method, in which there is high dependency between partitions. The results show the optimality of the results returned by our algorithm and its efficiency in improving performance. We have also demonstrated the importance of using fine-grained performance model over coarse-grained performance model in finding the distribution shape that effectively minimizes the overall execution time of application.

## ACKNOWLEDGMENT

Experiments presented in this paper were supported by KAU's High Performance Computing Center (AZIZ Supercomputer) (http://hpc.kau.edu.sa).

## REFERENCES

[1] *TOP500 Supercomputer Sites*. Accessed: 2019. [Online]. Available: http://top500.org/

[2] Y. Gao and P. Zhang, "A survey of homogeneous and heterogeneous system architectures in high performance computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2016, pp. 170–175.

[3] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2176–2190, Oct. 2018.

[4] A. Meade, D. K. Deeptimahanti, J. Buckley, and J. J. Collins, "An empirical study of data decomposition for software parallelization," *J. Syst. Softw.*, vol. 125, pp. 401–416, Mar. 2017.

[5] S. Tabik, G. Ortega, E. M. Garzón, and D. Suárez, "A data partitioning model for highly heterogeneous systems," in *Proc. Euro-Par Parallel Process. Workshops Euro-Par*, vol. 10104, F. Desprez, Eds. Cham, Switzerland: Springer, 2017, pp. 468–479.

[6] D. Unat *et al.*, "Trends in data locality abstractions for HPC systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 3007–3020, Oct. 2017.

[7] B. Bastem, D. Unat, W. Zhang, A. Almgren, and J. Shalf, "Overlapping data transfers with computation on GPU with tiles," in *Proc. Int. Conf. Parallel Process.*, 2017, pp. 171–180.

[8] D. Unat, T. Schulthess, T. Hoefle, A. Dubey, and J. Shalf, "Programming abstractions for data locality," in *Proc. PADAL Workshop Swiss Nat. Supercomput. Center (CSCS)*, Lugano, Switzerland, Apr. 2014.

[9] D. Unat, T. Nguyen, W. Zhang, and M. N. Farooqi, "TiDA: High-level programming abstractions," in *Proc. Int. Conf. High Perform. Comput.*, vol. 1. Cham, Switzerland: Springer, 2016, pp. 116–135.

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

[11] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2012, pp. 191–199.

[12] A. Snavely, X. Gao, C. Lee, L. Carrington, N. Wolter, J. Labarta, J. Gimenez, and P. Jones, "Performance modeling of HPC applications," in *Advances in Parallel Computing*, vol. 13. Amsterdam, The Netherlands: North Holland, 2004, pp. 777–784.

[13] D. Clarke, A. Lastovetsky, and V. Rychkov, *Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models* (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7155, no. 1. Berlin, Germany: Springer, 2012, pp. 450–459.

[14] M. Cierniak, "Compile-time scheduling algorithms for a heterogeneous network of workstations," *Comput. J.*, vol. 40, no. 6, pp. 356–372, Jun. 1997.

[15] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *J. Parallel Distrib. Comput.*, vol. 61, no. 4, pp. 520–535, Apr. 2001.

[16] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-matrix multiplication on heterogeneous platforms," in *Proc. Int. Conf. Parallel Process.*, 2000, pp. 289–298.

[17] A. Lastovetsky and R. Reddy, "Data distribution for dense factorization on computers with memory heterogeneity," *Parallel Comput.*, vol. 33, no. 12, pp. 757–779, Dec. 2007.

[18] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers with task size limits," in *Proc. 3rd Int. Symp. Parallel Distrib. Comput./3rd Int. Workshop Algorithms, Models Tools Parallel Comput. Heterogeneous Netw.*, 2004, pp. 133–140.

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

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

[21] A. Lastovetsky and R. Reddy, *Two-Dimensional Matrix Partitioning for Parallel Computing on Heterogeneous Processors Based on Their Functional Performance Models* (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6043. Berlin, Germany: Springer, 2010, pp. 112–121.

[22] A. DeFlumere, A. Lastovetsky, and B. A. Becker, "Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum*, May 2012, pp. 125–139.

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

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

[25] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A hierarchical data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous multi-accelerator NUMA nodes," *IEEE Access*, vol. 8, pp. 7861–7876, 2020.
[26] T. Hoefler, E. Jeannot, and G. Mercier, "An overview of topology mapping algorithms and techniques in high-performance computing," *High-Performance Computing on Complex Environments*, vol. 9781118712. Hoboken, NJ, USA: Wiley, 2014, pp. 73–94.
[27] N. Vijaykumar, E. Ebrahimi, K. Hsieh, P. B. Gibbons, and O. Mutlu, "The locality descriptor: A holistic cross-layer abstraction to express data locality in GPUs," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 829–842.
[28] B. A. Becker, "High-level data partitioning for parallel computing on heterogeneous hierarchical computational platforms by," Ph.D. dissertation, Dept. Comput. Sci. Inform., College Eng. Math. Phys. Sci., Univ. College Dublin, Dublin, Ireland, 2010.
[29] P. Bhat, V. Prasanna, and C. S. Raghavendra, "Block-cyclic redistribution over heterogeneous networks," *Cluster Comput.*, vol. 3, no. 1, pp. 25–34, 2000.
[30] *Hamidreza Khaleghzadeh_HPOPT GitLab*. Accessed: 2020. [Online]. Available: https://csgitlab.ucd.ie/HKhaleghzadeh/hpopt
[31] *GitHub—DC-Fukuoka_Jacobi_Jacobi—A Benchmark by Solving 2D Laplace Equation With Jacobi Iterative Method*. Accessed: 2019. [Online]. Available: https://github.com/dc-fukuoka/jacobi

**HIND TAHA AL-HASHIMI** received the B.Sc. degree in computer science from Umm Al Qura University, Makkah, Saudi Arabia, and the M.S. degree in computer science from King Saud University, Riyadh, Saudi Arabia, in 2012, and she is currently pursuing the Ph.D. degree in computer science. She is a Lecturer with Umm Al Qura University. Her research interests include high performance computing, parallel computing, exascale computing, big data, and software engineering.

**ABDULLAH AHMAD BASUHAIL** is currently a Professor of computer engineering and sciences with the Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include digital image processing and computer vision, cyber security, e-learning, resilient systems, and HPC systems. He was a member of IEEE Computer Society and Saudi Computer Society. He is currently a member of Saudi OER Network (SHMS).