IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 14, 2019, accepted December 8, 2019, date of publication December 16, 2019,
date of current version January 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2959905

A Hierarchical Data-Partitioning Algorithm for Performance Optimization of Data-Parallel Applications on Heterogeneous Multi-Accelerator NUMA Nodes

HAMIDREZA KHALEGHZADEH, RAVI REDDY MANUMACHU, AND ALEXEY LASTOVETSKY
School of Computer Science, University College Dublin, Dublin 4, D04 V1W8 Ireland

Corresponding author: Hamidreza Khaleghzadeh (hamidreza.khaleghzadeh@ucd.ie)

This work was supported by the Science Foundation Ireland (SFI) under Grant 14/IA/2474.

ABSTRACT Modern HPC platforms are highly heterogeneous with tight integration of multicore CPUs and accelerators (such as Graphics Processing Units, Intel Xeon Phis, or Field-Programmable Gate Arrays) empowering them to address the twin critical concerns of performance and energy efficiency. Due to this inherent characteristic, processing elements contend for shared on-chip resources such as Last Level Cache (LLC), interconnect, etc. and shared nodal resources such as DRAM, PCI-E links, etc., resulting in complexities such as resource contention, non-uniform memory access (NUMA), and accelerator-specific limitations such as limited main memory thereby necessitating support for efficient out-of-card execution. Due to these complexities, the performance profiles of data-parallel applications executing on these platforms are not smooth and deviate significantly from the shapes that allowed state-of-the-art load-balancing algorithms to find optimal solutions. In this paper, we propose a hierarchical two-level data partitioning algorithm minimizing the parallel execution time of data-parallel applications on clusters of h identical nodes where each node has c heterogeneous processors. This algorithm takes as input c discrete speed functions of cardinality m corresponding to the c heterogeneous processors. It does not make any assumptions about the shapes of these functions. Unlike load balancing algorithms, optimal solutions found by the algorithm may not load-balance an application in terms of execution time. The proposed algorithm has low time complexity of $O(m^2 \times h + m^3 \times c^3)$ unlike the state-of-the-art algorithm solving the same problem with the complexity of $O(m^3 \times c^3 \times h^3)$. We also propose an extension of the algorithm for clusters of h non-identical nodes where each node has c heterogeneous processors. We experimentally demonstrate the optimality of our algorithm using two well-known and highly optimized multi-threaded data-parallel applications, matrix-matrix multiplication and 2D fast Fourier transform, on a heterogeneous multi-accelerator NUMA node containing an Intel multicore Haswell CPU, an Nvidia K40c GPU, and an Intel Xeon Phi co-processor and a simulated homogeneous cluster of such nodes.

INDEX TERMS Heterogeneous platforms, multicore, Nvidia GPU, Intel Xeon Phi, workload partitioning, performance, HPC, hierarchical.

I. INTRODUCTION

Modern HPC platforms have become highly heterogeneous owing to the tight integration of multicore CPUs and accelerators (such as Graphics Processing Units (GPUs), Intel Xeon Phis, or Field-Programmable Gate Arrays) empowering them to address the twin critical concerns of performance and energy efficiency. The Top500 list [1] contains about 138 systems with accelerators; NVIDIA's Tesla P100 and Tesla V100 account for about 100 systems. All the top 25 computers in the Green500 list [2] contain accelerators.

Optimization of data-parallel applications for performance on such platforms faces several challenges due to the inherent complexities introduced by the tight integration of the compute devices. The complexities and the ensuing challenges include the following:

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

- Severe resource contention for shared resources in multicore CPUs such as Last Level Cache (LLC), on-chip interconnect, DRAM controllers, and shared nodal resources such as DRAM, and PCI-E links;
- Non-uniform Memory Access (NUMA) where the time for memory access between a core and main memory is not uniform and where main memory is distributed between locality domains or groups called NUMA nodes.
- The tight integration of accelerators with multicore CPUs via PCI-E communication links contains inherent limitations such as limited main memory of accelerators and limited bandwidth of the PCI-E communication links. These limitations pose formidable programming challenges to execution of large problem sizes on these accelerators thereby requiring support for efficient out-of-card implementations. Out-of-card implementations not only facilitate solving large workload sizes but also broaden the space of workload distributions (solutions) between the compute devices solving an input workload size.
  - Out-of-card executions, however, involve multiple data transfers of data structures (that fit inside the main memory of the accelerator) from the host CPU to the accelerator and back via the PCI-E communication link. The execution times of the out-of-card implementation is therefore impacted by the limited bandwidth of the link.
  - There is a lack of libraries providing interfaces that allow programmers to write efficient out-of-card implementations for their data-parallel kernels on accelerators. There are exceptions such as Nvidia's CUBLAS-XT package [3], which provides a set of basic linear algebra subroutines (BLAS) that utilize multiple GPUs, and Matrix Algebra on GPU and Multicore Architectures (MAGMA) [4], which provides out-of-card dense matrix factorizations. However, vendor out-of-card implementations (such as [3]) are shown not to be the best in terms of performance [5].

A visible manifestation of these complexities is a complex functional relationship between performance and workload size of multi-threaded data-parallel applications executing on these platforms where the shape of the performance profiles may be highly non-linear and non-smooth with drastic variations. To elucidate this relationship, we present two use cases.

Figure 1 illustrates the performance profiles of a matrix-vector multiplication application executing the highly optimized multi-threaded DGEMV routine in Intel math kernel library (MKL) on twelve tightly integrated identical Intel Xeon Phi coprocessor SE10/7120 accelerators. The specification of the accelerator is shown in Table 1. The application multiplies a dense matrix of size $N \times N$ with a vector of size N and runs on all cores of each Intel Xeon Phi co-processor. The figure shows the performance profiles of the application for all the twelve accelerators. The profiles

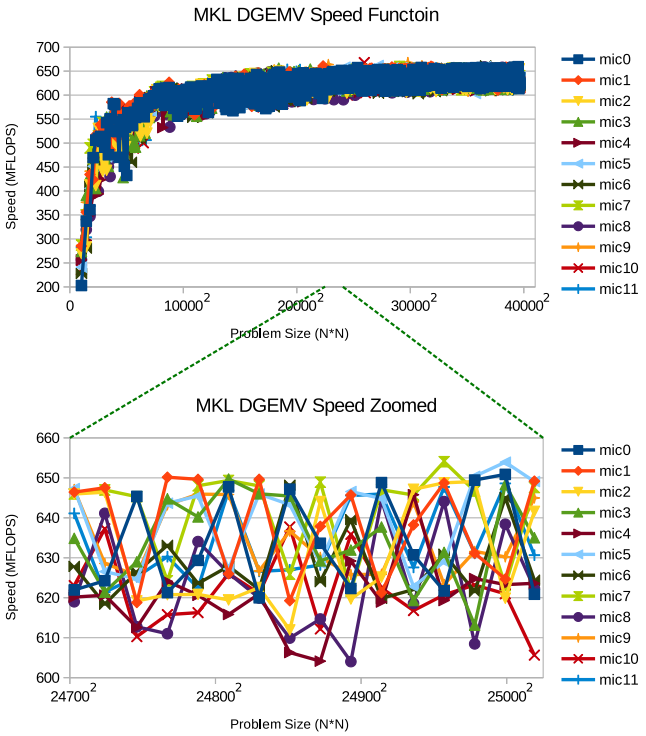

FIGURE 1. Speeds of Intel MKL DGEMV application for twelve Intel Xeon Phi SE10/7120 series coprocessors. Each function shows the speed of one Intel Xeon Phi coprocessor where mic0 shows the speed function of the first Xeon Phi and so on.

TABLE 1. 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

are built simultaneously to take into account resource contention. The construction of the profiles is automated using a strict experimental methodology (explained in Section IV) to ensure that the performance variations are not noise. The figure also shows the zoomed speed function between two arbitrarily chosen points in the speed functions.

From the figure, we can observe the following:

- Although all the Xeon Phi co-processors are identical, their speed functions (or performance profiles) demonstrate different shapes with noteworthy variations. The average of the variations is 7%. The variation is related to the difference of speed between two subsequent local minima (s_1) and maxima (s_2) and is defined as: $variation(\%) = \frac{|s_1 - s_2|}{\min(s_1, s_2)} \times 100$. Therefore, due to the inherent nature of the platforms today, even in a homogeneous environment, one must use heterogeneous workload distribution for optimal performance.

TABLE 2. HCLServer: Specifications of the Intel Haswell multicore CPU, Nvidia K40c, and Intel Xeon Phi 3120P.

Intel Haswell E5-2670V3	
Launch Date	Q3'14
No. of cores per socket	12
Socket(s)	2
CPU MHz	1200.402
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
Total main memory	64 GB DDR4
Memory bandwidth	68 GB/sec
Nvidia K40c	
Launch Date	Q4'13
No. of processor cores	2880
Total board memory	12 GB GDDR5
L2 cache size	1536 KB
Memory bandwidth	288 GB/sec
Intel Xeon Phi 3120P	
Launch Date	Q2'13
No. of processor cores	57
Total main memory	6 GB GDDR5
Memory bandwidth	240 GB/sec

- Due to the non-linear and non-smooth nature of the shapes, state-of-the-art load balancing data partitioning algorithms using such shapes as inputs to determine workload distribution [6]–[8] may not return optimal solutions.
- Using an average discrete speed function as input to data partitioning algorithm may not be optimal [9]. The model-based methods proposed in [9]–[11] cannot be used since they consider all identical processors and take a single speed function as an input.

In our second use case, we study the performance profiles of a 2D fast Fourier transform (2D-FFT) application on a multi-accelerator NUMA platform, HCLServer (the specification is shown in the Table 2). HCLServer contains an Intel Haswell multicore CPU consisting of 24 physical cores with 64 GB main memory and integrated with two accelerators, Nvidia K40c GPU and Intel Xeon Phi 3120P. Each accelerator is connected to a dedicated host core via a separate PCI-E link.

The hybrid 2D-FFT application executing in HCLServer is modelled by three abstract processors. A processing unit made of one or a group of CPU cores executing one (generally speaking, multi-threaded) computational kernel of the data-parallel application is modelled by an abstract processor [12]. To build the performance models of the abstract processors, the performance of the processing units representing these processors must be measured accurately. To ensure this, the processing units are grouped by shared system resources. Each group becomes an abstract processor. The performance of processing units in a group is measured when all the processing units in the group are executing a workload simultaneously, thereby taking into account the influence of resource contention. We represent a performance model by a discrete function of speed versus the problem size.

The first abstract processor contains 22 CPU cores executing the multi-threaded CPU kernel. The second abstract processor comprises the Nvidia K40c GPU along with its dedicated host CPU core executing the GPU kernel. Finally, the third abstract processor consists of Intel Xeon Phi 3120P co-processor along with its dedicated host CPU core executing the Xeon Phi kernel. The dedicated host CPU core is responsible for sending data from host to accelerator, kernel invocations on the accelerator and then copying results back from accelerator to host. Therefore, the pair consisting of an accelerator and its dedicated host core executing one accelerator kernel is modelled by an abstract processor. The kernel executing on the accelerator uses all the cores of the accelerator. The execution time of a kernel in the GPU and Xeon Phi abstract processors includes the times of data transfer between the accelerators and their host cores.

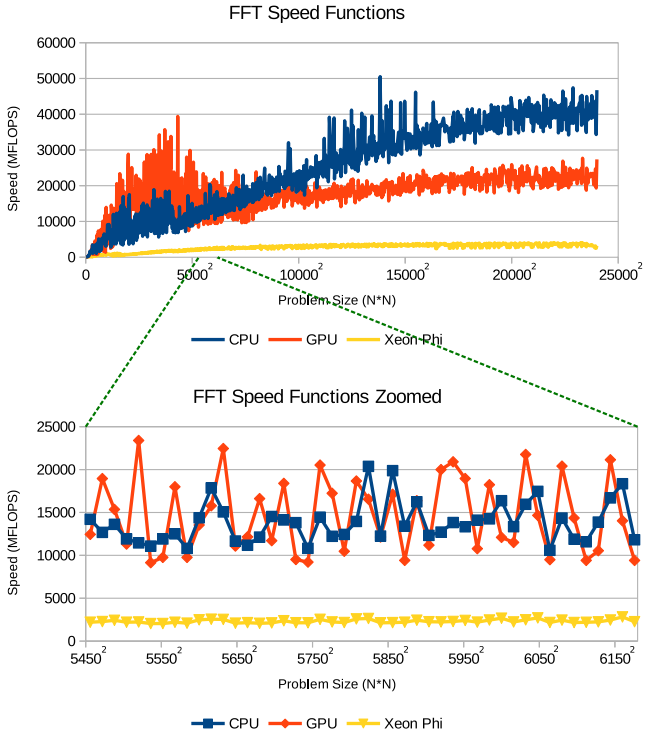

FIGURE 2. Speed functions of heterogeneous 2D DFT application for the multicore CPU, GPU, and Xeon Phi in the *HCLServer*. Also shown is the zoomed speed function between two points 5456^2 and 6176^2.

For the multicore CPU and Xeon Phi, Intel MKL FFT package is used. For the Nvidia GPU, CUFFT package is used. Unlike the matrix multiplication application, all computations for FFT are in-core (or in-card for the accelerators). Figure 2 shows the speed functions of abstract processors along with their zoomed functions between two data points 5456^2 and 6176^2. The Intel MKL FFT kernel for the multicore CPU uses 44 threads executing on 22 out of 24 physical cores.

From the figure, we can observe that Xeon Phi is markedly slower than CPU and GPU. It is because the execution time of communications between Xeon Phi and host CPU dominates the execution time of computations performed by Xeon Phi. However, GPU uses optimized data transfers by deploying two data engines (for transfers from CPU host to GPU and

from GPU to CPU host) and does not suffer from this problem. The maximum variations for CPU, GPU, and Xeon Phi are almost 350%, 560% and 200%, respectively. The maximum variations for Xeon Phi occur for problem sizes in the range of $[16^2, 800^2]$.

Due to the non-linear and non-smooth nature of the shapes, state-of-the-art load balancing data partitioning algorithms (based on functional performance models (FPMs)) using such shapes as inputs to determine workload distribution [6]–[8] may not return optimal solutions. Also the new model-based methods proposed in [9], [10] cannot be used for this case.

Khaleghzadeh *et al.* [13] address this shortcoming by proposing a novel data-partitioning algorithm (*Heterogeneous Performance OPTimization Algorithm (HPOPTA)*), which minimizes parallel execution time for the most general shapes of performance profiles for data-parallel applications executing on heterogeneous clusters of hybrid nodes. It has a time complexity of $O(m^3 \times p^3)$, where m represents the cardinality of the discrete performance/speed functions and p is the number of available heterogeneous processors. Optimal solutions found by *HPOPTA* may not be balanced in terms of execution time.

HPOPTA, however, has high theoretical and practical complexity for large p. We propose a hierarchical two-level data-partitioning algorithm, called HiPOPTA, to solve the performance optimization problem on large clusters of identical heterogeneous multi-accelerator NUMA nodes. To be specific, HiPOPTA determines workload distribution that minimizes the parallel execution time of data-parallel applications on clusters of h identical nodes where each node has c heterogeneous processors. It takes as input c discrete speed functions corresponding to the c heterogeneous processors. It does not make any assumptions about the shapes of these functions. Unlike load balancing algorithms, optimal solutions found by the algorithm may not load-balance an application in terms of execution time. HiPOPTA has low time complexity of $O(m^2 \times h + m^3 \times c^3)$ compared to HPOPTA, which solves the same problem with complexity $O(m^3 \times c^3 \times h^3)$.

We also propose an extension of HiPOPTA for a cluster of heterogeneous multi-accelerator NUMA nodes where all the nodes are non-identical. The non-identicality could be not only due to the physical composition of the platform but also due to performance differences or imbalances between compute nodes arising, for example, from employment of dynamic voltage and frequency scaling (DVFS) on one or more nodes to satisfy either a power budget or thermal envelope [14]–[16].

We experimentally analyse the optimality of HiPOPTA using two multi-threaded data-parallel applications, matrix-matrix multiplication (DGEMM) and 2D-FFT, on a heterogeneous multi-accelerator NUMA node containing an Intel multicore Haswell CPU, an Nvidia K40c GPU, and an Intel Xeon Phi co-processor and a simulated homogeneous cluster of such nodes. We show that HiPOPTA determines better solutions than load-balancing and FPM-based algorithms. We also prove that HiPOPTA finds optimal solutions with less computational complexity in comparison with HPOPTA.

The main contribution of our work is a hierarchical two-level workload partitioning algorithm that minimizes the parallel execution time of data-parallel applications executing on clusters of heterogeneous multi-accelerator NUMA nodes for two cases respectively where the nodes are identical and non-identical.

The rest of the paper is organized as follows. Section II contains the formulation of the hierarchical performance optimization problem. Section III presents our hierarchical data partitioning algorithm solving the problem. The experimental results are explained in the section IV. Section V presents the related work. Finally, we conclude the paper in section VI.

II. HETEROGENEOUS PERFORMANCE OPTIMIZATION PROBLEM: FORMULATION

Consider a workload size n executing on a cluster of h identical nodes. Suppose each node consists of c heterogeneous processors with discrete speed functions, $S = \{s_0(x), \ldots, s_{c-1}(x)\}$ where $s_i(x)$, $i \in \{0, 1, \cdots, c-1\}$, is the speed function of the processor P_i with a cardinality of m. The hierarchical heterogeneous performance optimization problem can be then formulated as follows:

HiPOPT(n, h, c, m, S, X_{opt}, t_{opt}): The problem is to find a workload distribution, $X_{opt} = \{x_{00}, x_{01}, \cdots, x_{0\,c-1}, x_{10}, \cdots, x_{h-1\,c-1}\}$, for the workload n executing on $h \times c$ heterogeneous processors so that the solution minimizes the parallel computation time during the execution of n. $x_{ij} \in X_{opt}$ represents the optimal workload distribution allocated to the j-th heterogeneous processor in the i-th node. The parameters (n, h, c, m, S) are the inputs to the problem. The outputs are the optimal workload distribution (X_{opt}), and the parallel execution time of the optimal solution (t_{opt}). The problem formulation, which is a integer non-linear programming (INLP) problem, is shown below:

$$
\begin{aligned}
t_{opt} = \min_{X} \max_{i=0}^{h-1} \max_{j=0}^{c-1} \frac{x_{ij}}{s_j(x_{ij})}, \\
\text{Subject to: } \sum_{i=0}^{h-1} \sum_{j=0}^{c-1} x_{ij} = n \\
\text{where } h, c, m, n \in \mathbb{Z}_{>0} \\
x_{ij} \in \mathbb{Z}_{\geq 0} \\
s_j(x) \in \mathbb{R}_{>0}
\end{aligned} \tag{1}
$$

The objective function in Eq. 1 is a function of workload distribution X, $X = \{x_{00}, x_{01}, \cdots, x_{h-1\,c-1}\}$, for a given workload n executing on the $h \times c$ processors. The inner-most maximum term inside Eq. 1, $\max_{j=0}^{c-1}$, finds the execution time of the workload size for a node. The next maximum term, $\max_{i=0}^{h-1}$, represents the execution time of the cluster of such identical nodes. Finally, Eq. 1 finds the distribution minimizing the execution time of n on the whole cluster. The number of active processors (processors that are assigned

non-zero workload size) in the optimal solution (X_{opt}) may be less than $h \times c$.

III. HIPOPTA: HIERARCHICAL TWO-LEVEL DATA PARTITIONING ALGORITHM SOLVING HIPOPT

The algorithm, HPOPTA [13], can be used to solve HiPOPT. It requires $c \times h$ speed functions as input, where m represents the cardinality of input speed functions, and $p = h \times c$ is the number of heterogeneous processors. We informally describe the algorithm using an example. The input to the algorithm are discrete time functions, which are derived from discrete speed functions. In the example, consider four heterogeneous processors ($p = 4$), which are available for execution of a workload of size $n = 16$. Figures 3 and 4 respectively show the sample speed functions, $S = \{s_0(x), \cdots, s_3(x)\}$, and the equivalent time functions, $T = \{t_0(x), \cdots, t_3(x)\}$, of the processors ($m = n = 16$ in our example for simplicity). The time functions are samples, which are representative of real-life data-parallel applications.

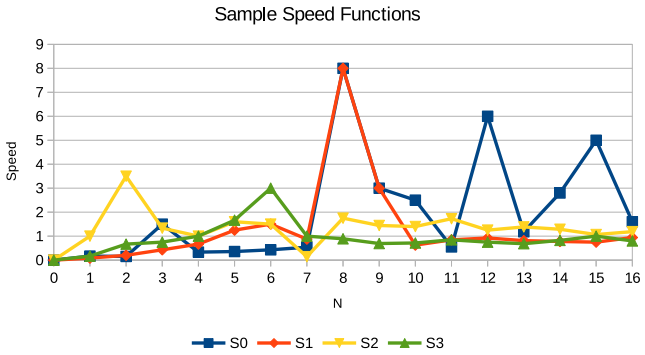

FIGURE 3. Speed functions of a sample application executing on an assumed parallel machine which consists of 4 processors.

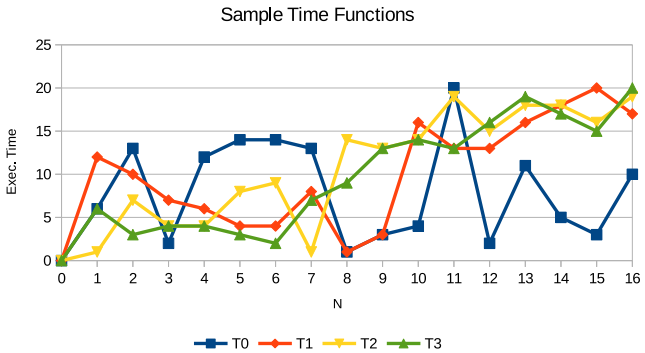

FIGURE 4. The equivalent time functions for the sample speed functions in Fig. 3.

The solution tree is constructed from the root, which is the only node at level L_0 of the tree. The value 16, which labels the root node, represents the whole workload size to be distributed between 4 processors $\{P_0, P_1, P_2, P_3\}$. Then, 17 workload sizes, including a zero workload size along with all workload sizes existing in the time function ($t_0(x)$), are assigned to the processor P_0 one by one. Although the workload sizes can be given to the processor in any order, we assign them in non-decreasing order of their execution time by the processor. As shown in Figure 5, problem sizes $\{0, 8, 3, 12, 9, 15, 10, 14, 1, 16, 13, 4, 2, 7, 5, 6, 11\}$, which have been sorted in non-decreasing order of execution times, are assigned to P_0 one-by-one at level L_0. Therefore, the root node is expanded into 17 children. The value, which labels an internal node at level L_1 (the root's child), represents the remaining workload size to be distributed between processors $\{P_1, P_2, P_3\}$. In its turn, each internal node at level L_1 becomes a root of a sub-tree, which is a solution tree for distribution of the remaining workload between three processors $\{P_1, P_2, P_3\}$. Finally, a distribution minimizing the parallel execution time will be returned as the optimal solution. In this example, the workload distribution $(8, 8, 0, 0)$, represented by the red *solution* leaf and resulting in the execution time of 1, will be returned as optimal.

To find the optimal workload distribution, an exhaustive approach will examine all combinations and select a workload distribution with the minimum computation time of parallel execution of the workload. Figure 5 shows the tree, which contains all the combinations. Due to the lack of space, we only show the tree partially. The complexity of the exhaustive algorithm is exponential.

HPOPTA is a branch-and-bound algorithm solving HiPOPT. It employs two key optimizations, memorization and smart backtracking, and two bounding criteria, time threshold and size threshold, to avoid examining all the possible solutions and to avoid exploring all the paths in the tree.

HPOPTA has time complexity of $O(m^3 \times p^3)$, which is high for large p. To address this shortcoming, we present a two-level hierarchical data partitioning algorithm, HiPOPTA, solving HiPOPT with low complexity. It uses two building blocks: a). POPTA [9]–[11], a model-based optimization algorithm used to minimize the parallel execution time of data-parallel applications executing on *homogeneous* clusters, and b). HPOPTA [13].

Figure 6 shows the schema of HiPOPTA, which is structured as a two-level hierarchical algorithm. The inputs to HiPOPTA are c performance profiles, representing the performances of c heterogeneous processors inside each node.

The top-most level (inter-node workload distribution) uses POPTA [10] to determine the workload distribution of the input workload size, n, between h identical nodes. The input to POPTA is the speed function of the whole node, which is determined during the first step of the execution of HiPOPTA. The output workload distribution is represented by $\{n_0, n_1, \cdots, n_{h-1}\}$ so that $n = \sum_{i=0}^{h-1} n_i$. HPOPTA is then invoked by all the nodes in parallel to find optimal workload distribution for c heterogeneous processors inside each node where $\{x_{i\,0}, x_{i\,1}, \cdots, x_{i\,c-1}\}$ is the intra-node optimal workload distribution for the i-th node, $n_i = \sum_{j=0}^{c-1} x_{i\,j}$, $i \in [0, h)$.

Consider a workload size of n executing on a given cluster of h identical nodes where each node contains

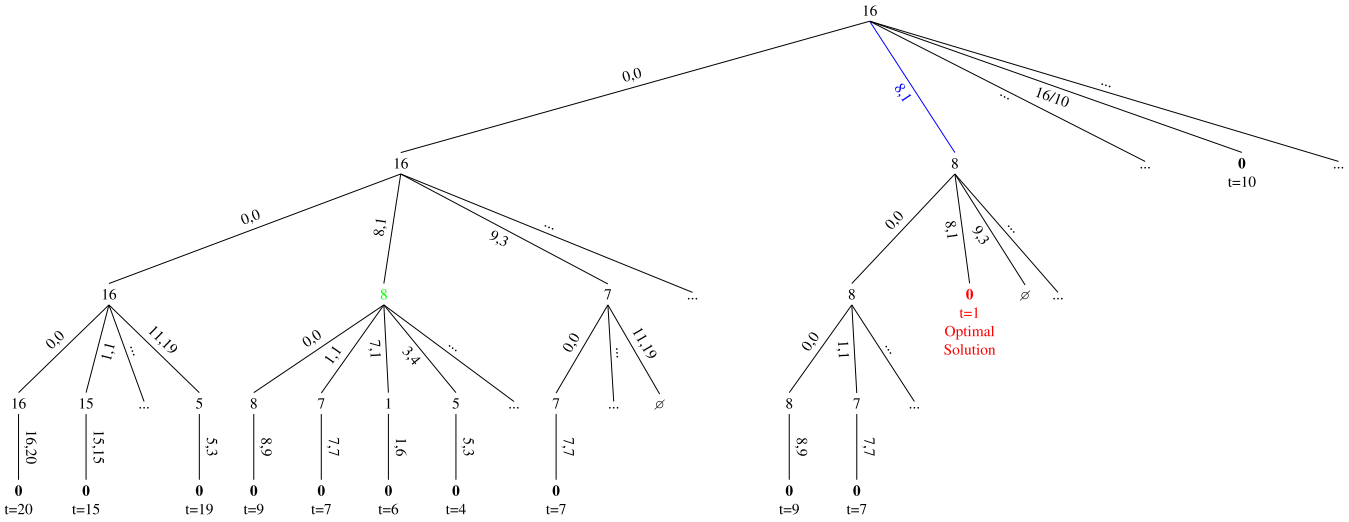

FIGURE 5. Applying naive approach to examine all combinations and select a workload distribution with the minimum computation time of parallel execution of the workload.

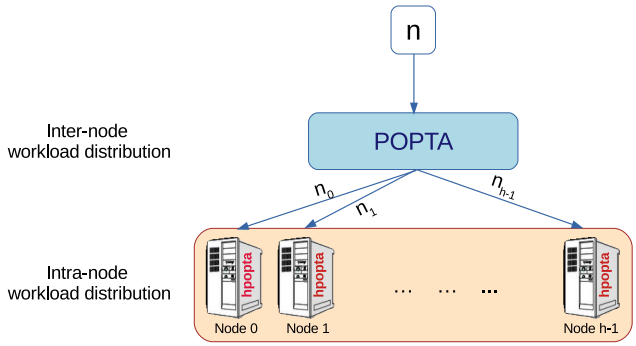

FIGURE 6. HiPOPTA schema consisting of two levels for parallel workload distribution on HPC clusters.

c heterogeneous processors. The main steps of HiPOPTA to find the optimal workload distribution are:

- **Building speed function of whole node using HPOPTA:** For each workload size in the speed function of a heterogeneous processor, we invoke HPOPTA to determine the optimal workload distribution solving the given workload size and the c performance profiles of the heterogeneous processors as inputs. We then execute the heterogeneous application on a node of the cluster using the HPOPTA workload distribution and measure its parallel execution time. The resulting discrete speed function consisting of data points obtained for all the workload sizes characterizes the performance of the node as a whole and represents its speed function. Since all the nodes in the cluster are identical, their speed functions will be the same, too.
- **Inter-node workload distribution:** We use the whole speed function to distribute the workload n between the nodes of the cluster. Because all nodes are identical, we can use parallel POPTA [11] for finding the optimal workload distribution between nodes. It has low complexity of $O(m^2 \times h)$ compared to HPOPTA solving the same problem of complexity $O(m^3 \times h^3)$.
- **Intra-node workload distribution:** HPOPTA is then applied inside each node to divide the assigned workload between the c heterogeneous processors of this node so that the execution time is minimized. The intra-node workload distributions can be determined by running HPOPTA on the nodes of the cluster in parallel. In this step, the c profiles determining the performance of heterogeneous processors are used as input to HPOPTA.

The hierarchical partitioning solution reduces the theoretical complexity for finding optimal distributions on large scale clusters. To prove this, assume a cluster containing h identical nodes where each node has c processors. The total number of heterogeneous processors available during an application execution is $p = c \times h$. Let the cardinality of performance functions be m.

We first calculate the time complexity of HiPOPTA. There are h identical nodes and therefore parallel POPTA finds the optimal inter-node distribution with the time complexity of $O(m^2 \times h)$. Optimal intra-node workload distributions are then found using parallel executions of HPOPTA on h nodes with time complexity of $O(m^3 \times c^3)$. Therefore, the total theoretical complexity will be equal to $O(m^2 \times h + m^3 \times c^3)$.

The theoretical complexity of the non-hierarchical partitioning algorithm, HPOPTA, is equal to $O(m^3 \times c^3 \times h^3)$. Therefore, HiPOPTA is $O(\frac{m^3 \times c^3 \times h^3}{m^2 \times h + m^3 \times c^3})$ times faster than the non-hierarchical one. For $h \gg c \gg m$, speedup is $O(h^2)$.

HiPOPTA always returns an optimal distribution. Indeed, according to [10], POPTA finds an optimal workload distribution between identical compute nodes represented by their speed function. Assuming that the speed function of a node reflects the fastest speed of execution of any given workload, it will find a globally optimal distribution. However,

by construction, the speed function of a node as a whole found locally by HPOPTA does give the fastest possible speed of execution for any workload given to the node.

Since there may be more than one optimal distribution, distributions returned by HiPOPTA and HPOPTA may be different. However, their execution times will always be the same.

The cost of building the speed function for a node is bounded by the cost of building the speed functions of the abstract processors. In the experimental section, we present our experimental methodology (since it is non-trivial) to construct the speed functions of the abstract processors. The cost of building the full speed functions of the abstract processors can be expensive. To reduce the cost, one approach is to build partial speed functions that are input to HiPOPTA to output optimal workload distribution for the specific input speed functions [7], [17].

A. EXTENSION OF HIPOPTA FOR A CLUSTER OF NON-IDENTICAL NODES

In this section, we consider an extension of HiPOPTA, called HiPOPTAX, for the case of a heterogeneous cluster of multi-accelerator NUMA nodes where all the nodes are non-identical. The non-identicality could be not only due to the physical composition of the platform but also due to performance differences or imbalances between compute nodes arising, for example, from employment of dynamic voltage and frequency scaling (DVFS) on one or more nodes to satisfy either a power budget or thermal envelope [14]–[16].

Consider a workload size of n executing on a given cluster of h nodes where each node contains c heterogeneous processors. The inputs to HiPOPTAX are $h \times c$ performance profiles where c profiles per node represent the performances of the c heterogeneous processors inside that node. The performance profiles of the c processors for all the nodes can be constructed in parallel.

The main steps to find the optimal workload distribution are:

- **Building speed functions of whole nodes using HPOPTA:** For each node, we build its speed function as explained in the first step of the HiPOPTA. The h speed functions corresponding to the h nodes are constructed in parallel.
- **Inter-node workload distribution:** The h speed functions are input to HPOPTA to determine the optimal workload distribution of workload size n between the nodes. The time complexity of this step is $O(m^3 \times h^3)$.
- **Intra-node workload distribution:** HPOPTA is then applied inside each node to determine the optimal workload distribution of the assigned workload between the c heterogeneous processors of this node. The intra-node workload distributions for the nodes are determined using parallel executions of HPOPTA. The time complexity of this step is $O(m^3 \times c^3)$.

To summarize, unlike HiPOPTA, HiPOPTAX invokes HPOPTA for finding both inter-node and intra-node workload distributions on a cluster of non-identical nodes. The total theoretical complexity of HiPOPTAX is equal to $O(m^3 \times h^3 + m^3 \times c^3)$. The theoretical complexity of the non-hierarchical partitioning using just HPOPTA is equal to $O(m^3 \times c^3 \times h^3)$. Therefore, HiPOPTAX is $O(\frac{m^3 \times c^3 \times h^3}{m^3 \times h^3 + m^3 \times c^3})$ times faster than the non-hierarchical one. For $h \gg c$, speedup is $O(c^3)$.

The optimality of HiPOPTAX follows on the similar lines as HiPOPTA.

IV. EXPERIMENTAL RESULTS

We experimentally study the performance of HiPOPTA compared with HPOPTA and load-balancing algorithms using two well-known multi-threaded data-parallel applications, matrix-matrix multiplication (DGEMM) and 2D fast Fourier transform (2D-FFT).

The experiments are a combination of actual measurements conducted on the HCLServer node and simulations for clusters of identical HCLServer nodes. The HCLServer node consists of one Intel Haswell CPU, one Nvidia K40c GPU and one Intel Xeon Phi 3120P. Table 2 summarizes the specification of the node. The actual measurements conducted on the HCLServer node includes the construction of the speed functions of the three abstract processors and the execution times of the HiPOPTA algorithm. The clusters contain $8, 16, \cdots, 256$ HCLServer nodes ($h \in \{8, 16, \cdots, 256\}$). Since each node has three abstract processors ($c = 3$), the total number of abstract processors ($p = h \times c$) ranges from 24 to 768. We do not consider the cost of communications.

For the sake of brevity, we will use the term *simulated cluster* to refer to simulations that are executed on a HCLServer node but that take as input parameter, the number of identical HCLServer nodes that constitute a cluster. Our experiments are a combination of actual measurements (speed functions, speedups, etc) and such simulations conducted on the HCLServer node.

A. MULTI-THREADED DATA-PARALLEL APPLICATIONS

The matrix-matrix multiplication application (DGEMM) computes $C = \alpha \times A \times B + \beta \times C$, where A, B, and C are respectively dense square matrices of size n^2, and α and β are constant floating-point numbers. The 2D fast Fourier transform (2D-FFT) computes the Fourier transform of a complex square matrix of size n^2. Each application consists of three different kernels, one for CPU, one for GPU, and one for PHI abstract processors.

For the CPU, the DGEMM application invokes the DGEMM routine provided in Intel MKL BLAS [18]. For the GPU and the Intel Xeon Phi, the application employs two packages, *ZZGemmOOC* and *XeonPhiOOC*, that perform out-of-card matrix multiplication of large dense matrices on them [5]. The *ZZGemmOOC* out-of-card package reuses CUBLAS [19] for in-card DGEMM calls, and *XeonPhiOOC* out-of-card package reuses Intel MKL BLAS [18] for in-card

DGEMM calls. The Intel MKL and CUDA versions are 2017.0.2 and 7.5.

The 2D-FFT application invokes Intel MKL FFT [20] for the multicore CPUs and the Xeon Phi, and CUFFT [21] for the Nvidia GPU. All computations are performed in-card.

B. SPEED/PERFORMANCE FUNCTIONS OF THE APPLICATIONS

HiPOPTA requires as an input the speed function of every single computational kernel executing in a hybrid parallel application. Each function contains a set of data points where each point represents the speed of the computational kernel for a given problem size.

A hybrid data-parallel application, which consists of a number of kernels (generally speaking, multi-threaded), runs in parallel on different parts of a hybrid platform. Due to tight integration and severe resource contention in heterogeneous hybrid platforms, the load of one computational kernel in a given hybrid application may significantly impact the performance of others to the extent, preventing from the ability to model the speed of each kernel in hybrid applications individually. Henceforth, computational kernels cannot be considered independent and their performance (execution times) should not be measured separately. To address this issue, in this work we restrict our study to such configurations of hybrid applications, where individual kernels are coupled loosely enough to allow us to build their individual speed functions with the accuracy sufficient for successful application of our optimization algorithm, HiPOPTA. To achieve this, we only consider configurations where no more than one CPU kernel or accelerator kernel is running on the corresponding device. Then, each group of cores executing an individual kernel of the application is modelled as an abstract processor [12] so that the executing platform is represented as a set of heterogeneous abstract processors. Each abstract processor solely constitutes the processing elements and resources which are involved in the execution of a given application kernel on it. We make sure that the sharing of system resources is maximized within groups of computational cores representing the abstract processors and minimized between the groups. This way, the contention and mutual dependence between abstract processors are minimized.

HCLServer, therefore following the design principles, is modelled by three loosely-coupled abstract processors *CPU*, *GPU* and *PHI*. The first abstract processor comprises 22 (out of total 24) CPU cores executing the multi-threaded CPU kernel. The second abstract processor contains the Nvidia K40c GPU, its dedicated host CPU core executing the GPU kernel along with the PCI-E link connecting the host to the accelerator. Finally, the Intel Xeon Phi 3120P coprocessor, its dedicated host CPU core executing the Xeon Phi kernel along with its PCI-E link make the third abstract processor. The dedicated host CPU core is responsible for sending data from host to accelerator, kernel invocations on the accelerator and then copying results back from the accelerator to host. Therefore, the pair consisting of an accelerator and its dedicated host core executing one accelerator kernel is modelled by an abstract processor. The kernel executing on an accelerator uses all its cores. The execution time of a kernel in the GPU and PHI abstract processors includes the times of data transfer between the accelerators and their host cores. Since there should be a one-to-one mapping between the abstract processors and computational kernels, any hybrid application executing on the server in parallel should consist of three kernels, one kernel per computational device.

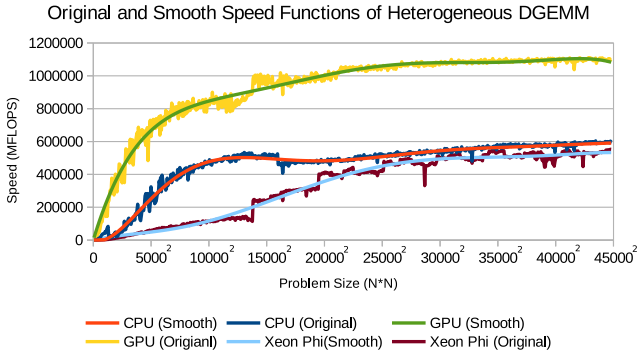

FIGURE 7. Original and smoothed speed functions of the heterogeneous hybrid DGEMM on HCLServer. The original functions are smoothed using polynomial trend line in LibreOffice Calc.

To build the speed functions for the hybrid heterogeneous DGEMM application, the same workload is simultaneously executed on all the abstract processors (CPU, GPU, PHI) of one HCLServer node, and the execution time of each kernel is precisely measured as explained in the following section. We calculate the speed of a workload of size n^2 as $\frac{2\times n^3}{t}$ where t is execution time taken to multiply two $n \times n$ square matrices. The profiles labelled as Original in Figure 7 represents the speed functions for the three processors for data points $\{64^2, 128^2, \cdots, 44800^2\}$.

We smooth the original speed function using a polynomial trend line in LibreOffice Calc. These smoothed profiles are labelled as Smooth in Figure 7. The smoothed functions are used as an input to the load-balancing algorithm based on Functional Performance Models (FPMs).

For 2D-FFT, the speed of a workload of size n^2 is calculated as $\frac{2.5\times n^2\times \log_2 n^2}{t}$ where t is execution time taken to compute 2D DFT of size n^2. Figure 8 shows original and smoothed speed functions of 2D-FFT for data points $\{16^2, 32^2, \cdots, 24000^2\}$.

The time to build the full speed functions of the abstract processors can be expensive. This is because, for each data point, statistical averaging is performed, which involves multiple runs of the application to determine the sample means for the execution times. To reduce the cost, one approach is to build partial speed functions [7], [17], which are input to HiPOPTA to output optimal workload distribution for the specific input speed functions.

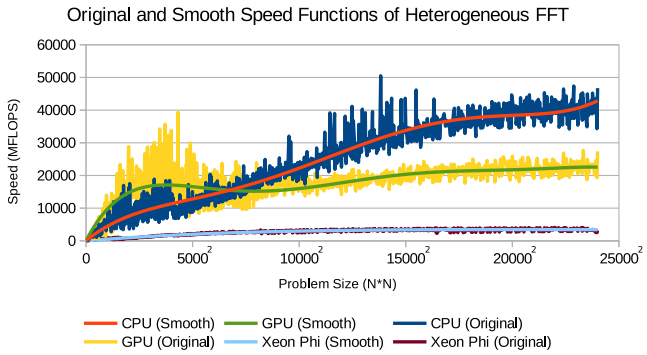

FIGURE 8. Original and smoothed speed functions of the heterogeneous hybrid 2D-FFT on HCLServer. The original functions are smoothed using polynomial trend line in LibreOffice Calc.

C. EXPERIMENTAL METHODOLOGY TO CONSTRUCT SPEED FUNCTIONS

To make sure the experimental results are reliable, we follow the methodology described below:

- The server is fully reserved and dedicated to these experiments during their execution. We also ensure that there are no drastic fluctuations in the load due to abnormal events in the server by monitoring its load continuously for a week using the tool *sar*. Insignificant variation in the load was observed during this monitoring period suggesting normal and clean behaviour of the server.
- Our hybrid application is executed simultaneously on all the three abstract processors, CPU, GPU, and Xeon Phi. To obtain a data point in the speed functions, the application is repeatedly executed until the sample mean lies in the 95% confidence interval and a precision of 0.1 (10%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations.
- We set *OMP_PLACES* and *OMP_PROC_BIND* environment variables to bind all the threads of a hybrid application to CPU cores.

Consider the hybrid application, which is named *app*, consisting of three sample kernels, *Kernel_cpu*, *Kernel_gpu* and *Kernel_phi*, which are run in parallel. The goal is to measure the execution time of the kernels in the application. To do this, we instrument the application as shown in Algorithm 1. This instrumented application returns the execution time of each kernel of all the three kernels.

We instrument each kernel in the hybrid application (*app*) by using the member function *gettimeofday()* of the Linux library *sys/time.h* to measure its execution time separately. As shown in Algorithm 1, the execution times are stored in variables te_{cpu}, te_{gpu} and te_{phi} and are returned at the end of the application execution.

We keep running the application until the sample means of the measured execution times of the application lie within a given confidence interval, and a given precision is achieved.

Algorithm 1 Instrumentation of a Sample Application (*App*) Consisting of Three Kernels, Executing on CPU, GPU and PHI Simultaneously

```
1:  #pragma parallel
2:  Begin
3:      te_cpu1 ← gettimeofday()
4:          Kernel_cpu( )
5:      te_cpu2 ← gettimeofday()
6:  End
7:  Begin
8:      te_gpu1 ← gettimeofday()
9:          Kernel_gpu( )
10:     te_gpu2 ← gettimeofday()
11: End
12: Begin
13:     te_phi1 ← gettimeofday()
14:         Kernel_phi( )
15:     te_phi2 ← gettimeofday()
16: End
17: te_cpu ← te_cpu2 − te_cpu1
18: te_gpu ← te_gpu2 − te_gpu1
19: te_phi ← te_phi2 − te_phi1
20: return (te_cpu, te_gpu, te_phi)
```

For this, we employ a script, which is named MeanUsingTtest. Algorithm 2 presents the pseudocode of this script. It executes the application *app* repeatedly until one of the following three conditions is satisfied:

1) The maximum number of repetitions (*maxReps*) has been exceeded (Line 5).
2) The sample means of all devices (kernel execution times) fall in the confidence interval (or the precision of measurement *eps* has been achieved) (Lines 13-18).
3) The elapsed time of the repetitions of application execution has exceeded the maximum time allowed (*maxT* in seconds) (Lines 20-22).

MeanUsingTtest returns the sample means of the execution times for each abstract processor (i.e. $time_{cpu}$, $time_{gpu}$, $time_{phi}$). The input parameters are minimum and maximum number of repetitions, *minReps* and *maxReps*. These parameter values differ based on the problem size solved. For small problem sizes ($32 \leq n \leq 1024$), these values are set to 10000 and 100000. For medium problem sizes ($1024 < n \leq 5120$), these values are set to 100 and 1000. For large problem sizes ($n > 5120$), these values are set to 5 and 50. The values of *maxT*, *cl*, and *eps* are set to 3600, 0.95, and 0.1. If the precision of measurement is not achieved before the maximum number of repeats have been completed, we increase the number of repetitions and also the maximum elapsed time allowed. However, we observed that condition (2) is always satisfied before the other two in our experiments.

Algorithm 3 shows the pseudocode of the helper functions CalAccuracy, which is used by MeanUsingTtest. It returns 1 if the sample mean of a given reading lies

Algorithm 2 Script Determining the Mean of an Experimental Run Using Student's t-Test

procedure MeanUsingTtest(app, $minReps$, $maxReps$, $maxT$, cl, eps, $reps\#$, $elapsedTime$, $time_{cpu}$, $time_{gpu}$, $time_{phi}$)

Input:
The application to execute, app
The minimum number of repetitions, $minReps \in \mathbb{Z}_{>0}$
The maximum number of repetitions, $maxReps \in \mathbb{Z}_{>0}$
The maximum time allowed for the application to run, $maxT \in \mathbb{R}_{>0}$
The required confidence level, $cl \in \mathbb{R}_{>0}$
The required accuracy, $eps \in \mathbb{R}_{>0}$

Output:
The number of experimental runs actually made, $reps\# \in \mathbb{Z}_{>0}$
The elapsed time, $elapsedTime \in \mathbb{R}_{>0}$
The mean execution times, $time_{cpu}, time_{gpu}, time_{phi} \in \mathbb{R}_{\geq 0}$

$reps \leftarrow 0$; $stop \leftarrow 0$; $etime \leftarrow 0$
$sum_{cpu} \leftarrow 0$; $sum_{gpu} \leftarrow 0$
$sum_{phi} \leftarrow 0$
while ($reps < maxReps$) and (!$stop$) **do**
  ($t_{cpu}[reps]$, $t_{gpu}[reps]$, $t_{phi}[reps]$) $\leftarrow$ Execute(app)
  $sum_{cpu} += t_{cpu}[reps]$
  $sum_{gpu} += t_{gpu}[reps]$
  $sum_{phi} += t_{phi}[reps]$
  if $reps > minReps$ **then**
    $stop_{cpu} \leftarrow$ CalAccuracy(cl, $reps+1$, t_{cpu}, eps)
    $stop_{gpu} \leftarrow$ CalAccuracy(cl, $reps+1$, t_{gpu}, eps)
    $stop_{phi} \leftarrow$ CalAccuracy(cl, $reps+1$, t_{phi}, eps)
    $stop \leftarrow stop_{cpu} \wedge stop_{gpu} \wedge stop_{phi}$
    $mT \leftarrow \max\{sum_{cpu}, sum_{gpu}, sum_{phi}\}$
    if $mT > maxT$ **then**
      $stop \leftarrow 1$
    end if
  end if
  $reps \leftarrow reps + 1$
end while
$reps\# \leftarrow reps$
$elapsedTime \leftarrow \max\{sum_{cpu}, sum_{gpu}, sum_{phi}\}$
$time_{cpu} \leftarrow \frac{sum_{cpu}}{reps}$; $time_{gpu} \leftarrow \frac{sum_{gpu}}{reps}$
$time_{phi} \leftarrow \frac{sum_{phi}}{reps}$
return ($reps\#$, $elapsedTime$, $time_{cpu}$, $time_{gpu}$, $time_{phi}$)
end procedure

in the 95% confidence interval (cl) and a precision of 0.1 ($eps = 10\%$) has been achieved. Otherwise, it returns 0.

If the precision of measurement is not achieved before the maximum number of repeats have been completed,

Algorithm 3 Algorithm Calculating Accuracy

function CalAccuracy(cl, $reps$, $Array$, eps)
  $clOut \leftarrow$ fabs(gsl_cdf_tdist_Pinv(cl, $reps-1$)) $\times$ gsl_stats_sd($Array$, 1, $reps$) / sqrt($reps$)
  if $clOut \times \frac{reps}{\sum_{i=0}^{reps-1} Array[i]} < eps$ **then**
    return 1
  end if
  return 0
end function

we increase the number of repetitions and also the maximum elapsed time allowed. However, we observed that condition (2) is always satisfied before the other two in our experiments.

To make sure that the experimental results are reliable, we employ an experimental methodology containing the following main steps: 1). We make sure the platform is fully reserved and dedicated to our experiments and is exhibiting clean and normal behaviour by monitoring its load continuously for a week. 2) For each data point obtained (in the graphs of speed functions, performance improvements), the sample mean is used, which is calculated by executing the application repeatedly until statistical confidence is achieved (95% confidence interval, the precision of 0.025 (2.5%)). Student's t-test is used to determine the sample mean. The test assumes 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.

D. DATA PARTITIONING ON CLUSTERS OF HETEROGENEOUS NODES USING HIPOPTA

For the first set of experiments, we study the optimality of HiPOPTA by comparing its workload distributions with the optimal solutions returned by HPOPTA.

The experimental data set for DGEMM includes workload sizes ranging from $(\frac{p}{3} \times 64 \times 100)^2$ to $(p \times 64 \times 700)^2$ with step size of 64^2, where p determines the total number of processors in the simulated cluster. The experimental dataset for 2D-FFT contains workload sizes ranging from $(\frac{p}{3} \times 16 \times 100)^2$ to $(p \times 16 \times 1500)^2$ with step size of 16^2.

HiPOPTA takes as input the three speed functions corresponding to the three abstract processors inside the HCLServer node. The main steps of HiPOPTA for a cluster of HCLServer nodes are below:

- **Building speed function of whole HCLServer using HPOPTA:** For each workload size, we run the heterogeneous application on HCLServer using the HPOPTA workload distribution and measure its parallel execution time. The resulting speed function characterizes the performance of HCLServer as a whole. Figures 9 and 10 respectively show the speed functions of DGEMM and 2D-FFT of whole HCLServer.
- **Inter-node workload distribution:** We use the whole HCLServer speed function to distribute workload

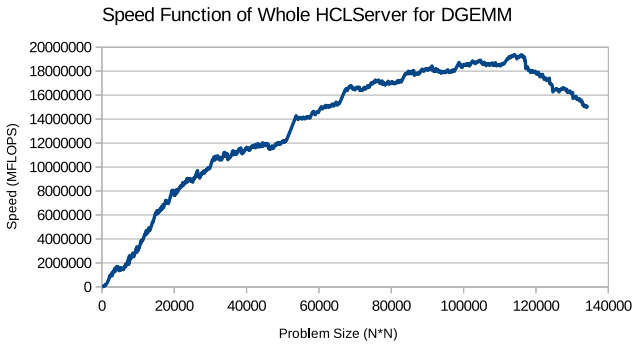

FIGURE 9. Speed functions of the DGEMM application for the whole HCLServer.

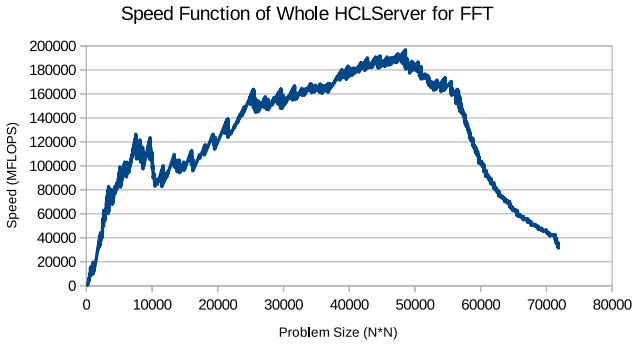

FIGURE 10. Speed functions of the 2D-FFT application for the whole HCLServer.

between the nodes of the simulated cluster using POPTA [10], since it has low complexity than HPOPTA.

- **Intra-node workload distribution:** HPOPTA is then applied inside each node to determine the optimal distribution of the assigned workload between CPU, GPU, and Xeon Phi of this node. The intra-node workload distributions can be determined by running HPOPTA on the nodes of the cluster in parallel. One must invoke HPOPTA for all the nodes since the workloads assigned to the nodes could all be different.

To compare the workload distribution obtained using HiPOPTA with HPOPTA, we use HPOPTA to obtain optimal workload distributions between processors in clusters of hybrid nodes. It takes as input, the speed function of each abstract processor in the simulated cluster. Because all nodes are identical, we use the speed functions for one node HCLServer (the original profiles in Figures 7 and 8) for all nodes in the simulated cluster. For example, for a simulated cluster consisting of 8 nodes of HCLServer ($h = 8$), the number of input functions to HPOPTA is 24 ($= h \times c = 8 \times 3$).

To evaluate the optimality of HiPOPTA, we compare the parallel execution times of HiPOPTA solutions with those obtained by applying plain HPOPTA with the experimental data sets explained earlier. Although the workload distributions generated by HiPOPTA and HPOPTA are not identical in a general case, the resulting execution times of the distributions returned by the two algorithms are the same. However, as shown in tables 3 and 4, HiPOPTA outperforms HPOPTA

TABLE 3. Percentage speed-up of HiPOPTA over HPOPTA for DGEMM.

$n^2 \setminus h$	8	16	32	64	128	256
55680^2	1771%	3031%	5664%	13153%	26170%	57100%
74240^2	2199%	3390%	6995%	15123%	29112%	62538%
111360^2	2208%	3611%	7277%	15885%	31507%	65173%

TABLE 4. Percentage speed-up of HiPOPTA over HPOPTA for 2D-FFT.

$n^2 \setminus h$	8	16	32	64	128	256
30296^2	3203%	6535%	13525%	21936%	43101%	80978%
40384^2	2515%	6037%	10512%	19113%	37364%	69794%
60592^2	2181%	4809%	9735%	17878%	40425%	73653%

by $O(h^2)$ for the applications DGEMM and 2D-FFT. Each table shows the percentage speed-up of HiPOPTA over HPOPTA for workload sizes $n^2 \times h$. For instance, regarding table 3, HiPOPTA is 6995% faster than HPOPTA to find the optimal distribution for the workload size $74240^2 \times 32$ on a cluster of 32 HCLServer nodes.

E. ANALYSING HIPOPTA OVER LOAD-BALANCING ALGORITHMS

For the second set of our experiments, we compare the performance improvement of the optimal solutions returned by HiPOPTA over the state-of-the-art workload distribution algorithms based on FPM [6]–[8] and the straightforward load-balancing algorithm using the same experimental data sets.

Since FPM algorithms suppose a smooth shape for speed functions, we use the smoothed speed functions for DGEMM and 2D-FFT applications as inputs. For comparison purposes, we label *FPM*-based algorithms by *smooth-FPM*. The straightforward load-balancing algorithm uses original functions to take into consideration variations in the performance profiles for finding load-balanced solutions. A load-balanced solution is one with the minimum difference between the execution times of processors. Like the optimal solutions returned by HiPOPTA, the number of processors with a non-zero workload (active processors) in load-balanced solutions may be less than the total number of processors. Because of its exponential time complexity, we compare HiPOPTA with the straightforward load-balancing algorithm for only one HCLServer node ($h = 1$).

The performance improvement (*PIMP*) of optimal solutions found by HiPOPTA against *smooth-FPM* is calculated as follows: $PIMP_{FPM}(\%) = \frac{t_{smooth-FPM} - t_{HiPOPTA}}{t_{HiPOPTA}} \times 100$, where $t_{smooth-FPM}$ and $t_{HiPOPTA}$ respectively are the execution times of solutions found by executing HiPOPTA using smoothed and actual time functions. $t_{smooth-FPM}$ is estimated as follows. First, the workload distribution for a given workload is found by HiPOPTA using smoothed time functions as input. Then, the execution time for this distribution is calculated using the original, not smoothed, speed functions. Thus, the smoothed speed functions are used for finding the FPM workload distribution, and its execution time is then found using the real time functions.

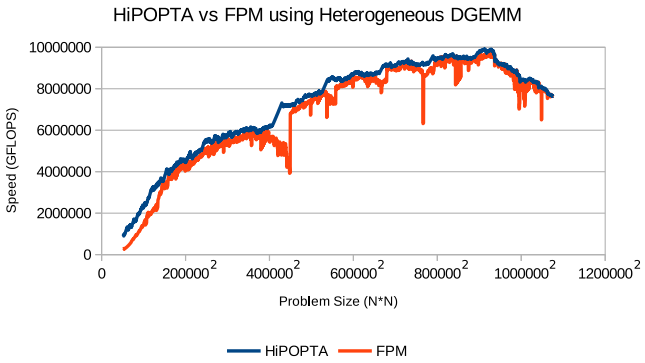

FIGURE 11. Speed functions of the heterogeneous DGEMM executing on on 8 HCLServer nodes. The application is executed for each problem size *n* using two different workload distributions, HiPOPTA and FPM.

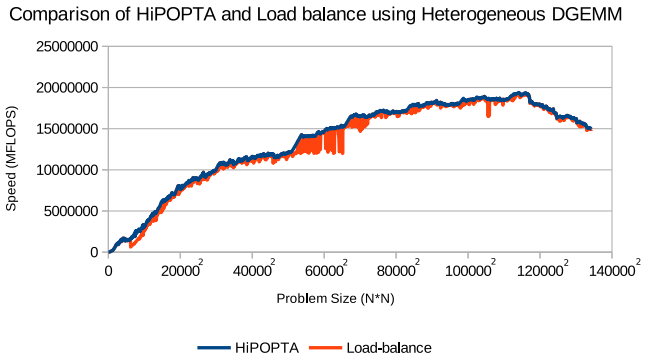

FIGURE 12. Speed functions of the DGEMM application for whole HCLServer. The application is executed for each problem size using two different workload distributions HiPOPTA and load-balancing on one HCLServer node.

The percentage performance improvement (*PIMP*) of HiPOPTA against load-balancing algorithm is calculated as follows: $PIMP_{balance}(\%) = \frac{t_{balance} - t_{HiPOPTA}}{t_{HiPOPTA}} \times 100$, where $t_{balance}$ is the execution times of solutions with minimum difference between the execution times of processors, and $t_{HiPOPTA}$ represents the execution time of solution found using HiPOPTA.

1) HETEROGENEOUS HYBRID MATRIX-MATRIX MULTIPLICATION

The minimum, average, and maximum performance improvements of HiPOPTA over smooth-FPM on a cluster of 8 HCLServer nodes are 0, 14, 261 percent respectively. Figure 11 shows the speed of heterogeneous DGEMM on HCLServer when executed using HiPOPTA in comparison with FPM workload distribution on a simulated cluster of 8 nodes.

We use the aforementioned experimental data set to compare HiPOPTA against the straightforward load-balancing approach. On a single node, HiPOPTA gives the minimum, average, and maximum performance improvements of 0, 5, 143 percent. Figure 12 shows the speed of heterogeneous DGEMM on HCLServer when executed using HiPOPTA in comparison with load-balanced workload distribution on one HCLServer node.

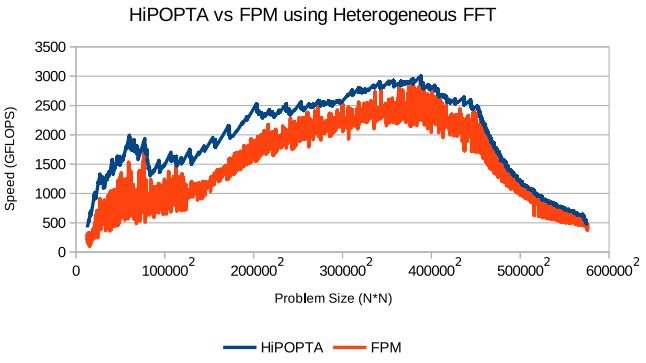

FIGURE 13. Speed functions of the heterogeneous hybrid 2D-FFT running on a cluster of 8 HCLServer nodes. The application is executed for each problem size *n* using two different workload distributions HiPOPTA and FPM.

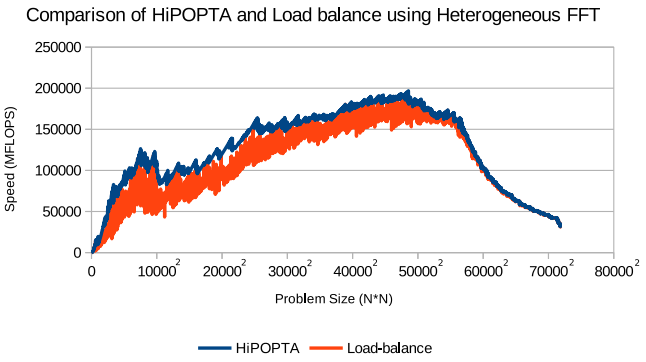

FIGURE 14. Speed functions of the heterogeneous hybrid 2D-FFT for whole HCLServer. The application is executed for each problem size using two different workload distributions HiPOPTA and load-balancing on one HCLServer node.

2) HETEROGENEOUS HYBRID 2D FAST FOURIER TRANSFORM

HiPOPTA gives the minimum, average and maximum performance improvements of 0, 43, 513 percent respectively in comparison with smooth-FPM on 8 identical nodes of HCLServer. Figure 13 compares the speed of heterogeneous FFT when executed using HiPOPTA with the speed when the workload is distributed using FPM on a cluster of 8 HCLServer nodes.

Figure 14 shows the speed of 2D-FFT on a single node of HCLServer when executed using HiPOPTA in comparison with the straightforward load-balanced workload distribution. HiPOPTA gives the minimum, average, and maximum performance improvements of 0, 19, 331 percent.

3) DISCUSSION

We observed almost the same percentage of improvement for different cluster sizes for both DGEMM and 2D-FFT. It can be concluded that the performance improvement is independent of $p = h \times c$ assuming the cost of communications is not taken into account.

There is a strong correlation between the average performance improvements and the average variations in speed functions. Furthermore, the maximum performance improvement over FPM cannot exceed the maximum variation in the

speed functions. In our experiments, all nodes in simulated clusters are identical and their speed functions consequently will be identical. Thus, average and maximum performance improvements of a simulated cluster consisting of identical nodes are not related to the number of nodes but are related to the shapes of speed functions which are identical for all nodes.

For any workload size in the experimental data sets for matrix multiplication and 2D-FFT applications, the execution time of HiPOPTA is insignificant compared to the execution time of the application.

V. RELATED WORK

We divide the related research into following categories:

- High performance heterogeneous computing platforms, where each node contains multicore CPUs and multiple types of accelerators (GPUs, Intel Xeon Phis, and FPGAs). For the sake of brevity, we represent high performance heterogeneous computing platforms by the acronym, HPC^2.
- Programming models and tools for HPC^2 platforms.
- Applications and benchmark suites for HPC^2 platforms.
- Load balancing and load imbalancing optimization algorithms for HPC platforms.
- Finally, hierarchical approaches to reducing complexity.

A. HPC^2 PLATFORMS

In this section, we survey research platforms, which contain nodes composed of multicore CPUs and multiple types of accelerators. Kelmelis et al. [22] built a desktop supercomputer composed of a CPU, GPU, and FPGA (Field-Programmable Gate Array) as a cost-effective alternative to building a cluster. They use it to reduce the simulation times in the modelling of nanophotonic devices using a compute and memory intensive electromagnetic solver. Showerman et al. [23] describe a 16-node cluster, where each node contains two dual-core AMD Opteron CPUs, four NVIDIA Quadro FX 5600 GPU cards, and one Nallatech H101-PCIX FPGA accelerator card. Three applications, NAMD, WRF, and TRACF, were implemented for this platform. NAMD and WRF used only CPUs and GPUs. However, it is not clear if the CPUs also took part in computation. TRACF is executed using either GPUs or FPGAs. No application used all the different types of available accelerators.

Tsoi et al. [24] present a heterogeneous computer cluster called Axel, which contains 16 nodes where each node has an AMD Phenom Quad-Core CPU, an Nvidia Tesla C1060 GPU, and a Xilinx Virtex-5 LX330 FPGA. They use the map-reduce model to map an application to the various computing devices in their platform. To demonstrate the use of all the computing devices in their platform, they use an N-body application. Kastl and Loimayr *et al.* [25] relate their experiences with an implementation of cryptographic and cryptanalytic algorithms on a cluster composed of multicore CPUs, GPUs, and FPGAs. They do not use all the accelerators together in their brute force algorithms.

Inta et al. [26] build a heterogeneous platform composed of multicore CPUs, GPUs, and FPGAs with the prime motivation of satisfying the insatiable computational needs of astronomy applications using a low-cost platform built from commodity-off-the-shelf components. Two applications, Monte Carlo Integration (for calculation of π) and Normalized Cross-Correlation (sliding dot product) are used to demonstrate how applications have been computationally mapped to the different devices of the platform. They partition the application into parts where each part is executed on the device, which is reportedly ideally matched for executing the part. Danczul *et al.* [27] relate their experience building a heterogeneous cluster of 16 nodes on which they implement a password bruteforcer for password-protected documents. Each node is equipped with an Nvidia GeForce GTX 680 device and a Xilinx ML605 FPGA development board. The core computation of the password cracker algorithm is the computation of the user key, which involves MD5 hashes and RC4 encryptions. The hashes are performed on the GPU and the encryptions are executed in the FPGA. That is the core computations are divided into distinct algorithms and mapped to the accelerators, which are best suited for executing these algorithms. Wu *et al.* [28] build a heterogeneous computing platform composed from multicore CPUs, GPUs, and FPGAs to understand how to map an application to the various architectures with an objective of maximizing energy efficiency.

B. PROGRAMMING MODELS AND TOOLS FOR HPC^2

This section surveys programming models and tools for HPC^2 platforms. The main challenge in the heterogeneous system for applications is portability and programmability. There is not a common application written in one language to execute it through all heterogeneous devices. This problem invokes creating special tools and models for executing applications on the platform with multiple types of accelerators (CPU-GPU-PHI or CPU-GPU-FPGA). In the present, the most popular programming model for heterogeneous systems is provided by OpenCL library.

Krommydas *et al.* [29] evaluate the portability of OpenCL to all devices such as CPU, GPU, APU, FPGA and the Xeon Phi co-processor. Kreutzer *et al.* [30] propose a tool called *GHOST*, which stands for General, Hybrid, and Optimized Sparse Toolkit. This tool provides hybrid-parallel numerical kernels, intelligent resource management, and truly heterogeneous parallelism for multicore CPUs, Nvidia GPUs, and the Intel Xeon Phi (CPU-GPU-PHI) However, the data partitioning is static and is based on single-number performances of devices. Deepika *et al.* [31] present a tool called *OpenCLGen*. OpenCLGen is a web-based software to automate OpenCL program generation for Single Kernel on Single Device as well as Multiple Kernels on Multiple Devices. The user provides the input of a kernel name, kernel code or file, the type of target device and their number. The output contains source code, binary, and header directories along with Makefile.

The outputs are then executed on individual devices in their heterogeneous cluster, which contains accelerators such as NVIDIA TeslaM2050, NVIDIA K20, Vitrex5 FPGA card mounted on the PCIe slots.

To the best of our knowledge, there are very few focused research efforts, such as OmpSs [32], [33], proposing programming models and tools for heterogeneous systems that allow writing hybrid applications that utilize all the devices (multicore CPUs and accelerators). Due to this, programmers use a mix of tools such as MPI, OpenMP, CUDA, OpenCL, etc to program hybrid applications. However, this gap affords many opportunities for researchers to build new programming models and tools for such hybrid platforms.

C. APPLICATIONS AND BENCHMARK SUITES FOR HPC[2]

OpenCL is a well-known framework that offers a common programming model for heterogeneous computing devices. As a result, many existing benchmarks (such as Rodinia [34]) were re-implemented in OpenCL, and some new ones (for example, SHOC [35]) based on OpenCL are also released. SHOC benchmark suite [35] is distinct from Rodinia in its design. It is scalable to large clusters and a large number of devices (for example, multiple GPUs). Therefore, in contrast with Rodinia, it can be run on a large cluster to test multiple GPUs in parallel. But similar to Rodinia, SHOC is also designed especially for the systems containing GPUs and multi-core processors, and therefore it is optimized to test the performance and stability of such computing systems. Krommydas *et al.* [36], have introduced OpenDwarfs: a benchmark suite which is portable across multiple devices (parallel platforms) including multi-core CPUs, discrete and integrated GPUs (GPUs and APUs), the Intel Xeon Phi coprocessor and even FPGAs. However, none of the OpenDwarfs' benchmarks supports partitioning, and so cannot be run across all the devices simultaneously.

D. LOAD BALANCING AND LOAD IMBALANCING ALGORITHMS

In this section, we will survey load balancing and load imbalancing algorithms followed by hierarchical approaches to reducing complexity.

Load-balancing algorithms developed over the years for performance optimization on parallel platforms have attempted to take into consideration the real-life behaviour of applications executing on these platforms. This can be discerned from the evolution of performance models for computation used in these algorithms. The simplest models used positive constant numbers and different notions such as normalized processor speed, normalized cycle time, task computation time, average execution time, etc. to characterize the speed of an application [37]–[43]. The performance of a processor in these works is assumed to have no dependence on the size of the workload. Advanced load balancing algorithms use FPMs, which are application-specific and represent the speed of a processor by a continuous function of problem size but satisfying some assumptions on its shape [44]–[46]. The FPMs capture accurately the real-life behaviour of applications executing on nodes consisting of uniprocessors (single-core CPUs).

Performance profiles of multi-threaded data-parallel applications executing on modern HPC platforms suffer from drastic variations due to the complexities of resource contention and NUMA inherent in these platforms. The shapes of the profiles violate the assumptions of the FPM-based algorithms proposed in [6]–[8], [47]–[52]. To deal with this challenge, novel model-based algorithms [9]–[11] have been proposed which find optimal workload distribution on homogeneous HPC platforms. The proposed approaches make no assumptions about the shapes of performance profiles. They imbalance the workload between processors to maximize performance.

Khaleghzadeh *et al.* [13] explore in-depth the deterministic and reproducible performance variations for hybrid applications executing on modern heterogeneous HPC platforms and present a novel data-partitioning algorithm for performance optimization. The proposed algorithm also imbalances the workload between processors to maximize performance.

E. HIERARCHICAL APPROACHES

Baldwin *et al.* [53] propose hierarchical partitioning algorithms for massive scientific data sets generated by large-scale simulations. Clarke *et al.* [54] present a two-level hierarchical algorithm to optimize parallel matrix-matrix multiplication on a cluster of CPU+GPU nodes. Dorronsoro *et al.* [55] present a hierarchical two-level strategy to optimize parallel applications on multicore distributed systems to maximize the quality of service while reducing energy consumption. The high level is between distributed data centres and the lower level is within a data centre. von Kistowski *et al.* [56] employ hierarchical load distribution using four-levels (Hyperthreading units, CPU cores, sockets, servers) to maximize the energy efficiency of a multi-node heterogeneous platform.

VI. CONCLUSION

Modern high-performance computing platforms have become highly heterogeneous due to tight integration of multicore CPU processors and accelerators. This tight integration causes contention on shared resources such as Last Level Cache (LLC), DRAM, PCI-E links, etc., resulting in severe resource contention and Non-Uniform Memory Access (NUMA). Moreover, the accelerators feature limited main memory compared to the multicore CPU host and are connected to it via limited bandwidth PCI-E links thereby requiring support for efficient out-of-card execution. Due to these complexities, the performance profiles of data-parallel applications executing on these platforms are not smooth and deviate significantly from the shapes that would allow state-of-the-art load-balancing algorithms to find optimal solutions.

State-of-the-art model-based data partitioning algorithm [13] solves the performance optimization problem of

data-parallel applications on heterogeneous HPC clusters without making any assumptions about the shapes of these functions. The algorithm, however, has high theoretical and practical complexity.

In this work, we proposed a hierarchical two-level data partitioning algorithm (HiPOPTA) minimizing the parallel execution time of data-parallel applications on clusters heterogeneous multi-accelerator NUMA nodes. This algorithm takes as input discrete speed functions corresponding to the heterogeneous processors contained inside each node. It does not make any assumptions about the shapes of these functions. Unlike load balancing algorithms, optimal solutions found by the algorithm may not load-balance an application in terms of execution time. Our proposed algorithm has lower time complexity compared to the state-of-the-art data partitioning algorithm. We also proposed an extension of HiPOPTA for a cluster of heterogeneous multi-accelerator NUMA nodes where all the nodes are non-identical.

We experimentally demonstrated the optimality of our algorithm using two well-known multi-threaded data-parallel applications, matrix-matrix multiplication and 2D fast Fourier transform, on a heterogeneous multi-accelerator NUMA node containing an Intel multicore Haswell CPU, an Nvidia K40c GPU, and an Intel Xeon Phi co-processor and a simulated homogeneous cluster of such nodes. We also showed that its practical complexity is low compared to the state-of-the-art data partitioning algorithm.

REFERENCES

[1] Top500. (2018). *TOP500 List for November 2018*. [Online]. Available: https://www.top500.org/lists/2018/11/

[2] Green500. (2018). *Green500 List for November 2018*. [Online]. Available: https://www.top500.org/green500/lists/2018/11/

[3] CUBLAS-XT. (2016). *CUBLAS-XT: Multi-GPU Version of CUBLAS Library Supporting Out-of-Core Routines*. [Online]. Available: https://developer.nvidia.com/cublas

[4] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Comput.*, vol. 36, nos. 5–6, pp. 232–240, Jun. 2010.

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

[6] 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.

[7] D. Clarke, A. Lastovetsky, and V. Rychkov, "Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms," *Parallel Process. Lett.*, vol. 21, no. 2, pp. 195–217, 2011.

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

[9] 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.

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

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

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

[13] 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.

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

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

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

[17] R. R. Manumachu and A. Lastovetsky, "Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 4, 2019, Art. no. e4958.

[18] I. Corporation. (2018). *Intel Math Kernel Library-Intel MKL BLAS*. [Online]. Available: https://software.intel.com/en-us/mkl/features/linear-algebra

[19] Nvidia. (2017). *CUDA Toolkit Documentation*. [Online]. Available: http://docs.nvidia.com/cuda/cublas/index.html#axzz4kRVc2o6B

[20] I. Corporation. (2018). *Intel Math Kernel Library-Intel MKL FFT*. [Online]. Available: https://software.intel.com/en-us/mkl/features/fft

[21] Nvidia. (2018). *Optimized FFT Routines for Nvidia Graphics Processors*. [Online]. Available: https://docs.nvidia.com/cuda/cufft/index.html

[22] E. J. Kelmelis, J. P. Durbano, J. R. Humphrey, F. E. Ortiz, and P. F. Curt, "Modeling and simulation of nanoscale devices with a desktop supercomputer," *Proc. SPIE*, vol. 6328, Sep. 2006.

[23] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, and W.-M. Hwu, "QP: A heterogeneous multi-accelerator cluster," in *Proc. 10th LCI Int. Conf. High-Perform. Clustered Comput.*, 2009.

[24] K. H. Tsoi and W. Luk, "Axel: A heterogeneous cluster with FPGAs and GPUs," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2010, pp. 115–124.

[25] W. Kastl and T. Loimayr, "A parallel computing system with specialized coprocessors for cryptanalytic algorithms," in *Proc. Sicherheit*, 2010, pp. 73–84.

[26] R. Inta, D. J. Bowman, and S. M. Scott, "The 'Chimera': An off-the-shelf CPU/GPGPU/FPGA hybrid computing platform," *Int. J. Reconfig. Comput.*, vol. 2012, nos. 1687–7195, p. 2, Jan. 2012.

[27] B. Danczul, J. Fuß, S. Gradinger, B. Greslehner, W. Kastl, and F. Wex, "Cuteforce analyzer: A distributed bruteforce attack on pdf encryption with gpus and fpgas," in *Proc. Int. Conf. Availability, Rel. Secur.*, Sep. 2013, pp. 720–725.

[28] Q. Wu, Y. Ha, A. Kumar, S. Luo, A. Li, and S. Mohamed, "A heterogeneous platform with GPU and FPGA for power efficient high performance computing," in *Proc. 14th Int. Symp. Integr. Circuits (ISIC)*, Dec. 2014, pp. 220–223.

[29] K. Krommydas, M. Owaida, C. D. Antonopoulos, N. Bellas, and W. C. Feng, "On the portability of the OpenCL dwarfs on fixed and reconfigurable parallel platforms," in *Proc. Int. Conf. Parallel Distrib. Syst.*, Dec. 2013, pp. 432–433.

[30] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, and G. Wellein, "GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems," *Int. J. Parallel Program.*, vol. 45, no. 5, pp. 1–27, 2016.

[31] H. V. Deepika, N. N. Mangala, and S. C. Babu, "Automatic program generation for heterogeneous architectures," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Sep. 2016, pp. 102–109.

[32] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "OmpSs: A proposal for programming heterogeneous multi-core architectures," *Parallel Process. Lett.*, vol. 21, no. 2, pp. 173–193, 2011.

[33] BSC. (2019). *The OmpSs Programming Model*. [Online]. Available: https://pm.bsc.es/ompss

[34] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.

[35] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proc. 3rd Workshop General-Purpose Comput. Graph. Process. Units (GPGPU)*, 2010, pp. 63–74.
[36] K. Krommydas, W.-C. Feng, C. D. Antonopoulos, and N. Bellas, "Opendwarfs: Characterization of dwarf-based benchmarks on fixed and reconfigurable architectures," *J. Signal Process. Syst.*, vol. 85, no. 3, pp. 373–392, Dec. 2016.
[37] M. Cierniak, M. J. Zaki, and W. Li, "Compile-time scheduling algorithms for a heterogeneous network of workstations," *Comput. J.*, vol. 40, no. 6, pp. 356–372, 1997.
[38] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1033–1051, Oct. 2001.
[39] 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.
[40] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)," *IEEE Trans. Comput.*, vol. 50, no. 10, pp. 1052–1070, Oct. 2001.
[41] A. Lastovetsky and R. Reddy, "A novel algorithm of optimal matrix partitioning for parallel dense factorization on heterogeneous processors," in *Proc. Int. Conf. Parallel Comput. Technol.* Berlin, Germany: Springer, 2007, pp. 261–275.
[42] M. Fatica, "Accelerating linpack with CUDA on heterogenous clusters," in *Proc. 2nd Workshop Gen. Purpose Process. Graph. Process. Units*, 2009, pp. 46–51.
[43] R. Wyrzykowski, L. Szustak, K. Rojek, and A. Tomas, "Towards efficient decomposition and parallelization of MPDATA on hybrid CPU-GPU cluster," in *Proc. Int. Conf. Large-Scale Sci. Comput.* Berlin, Germany: Springer, 2013, pp. 457–464.
[44] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Apr. 2004, p. 104.
[45] A. Lastovetsky and R. Reddy, "Data partitioning for multiprocessors with memory heterogeneity and memory constraints," *Sci. Program.*, vol. 13, no. 2, pp. 93–112, 2005.
[46] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 1, pp. 76–90, 2007.
[47] A. Ilić, F. Pratas, P. Trancoso, and L. Sousa, "High-performance computing on heterogeneous systems: Database queries on CPU and GPU," in *High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives*. Amsterdam, The Netherlands: IOS Press, 2010, pp. 202–222.
[48] X. Liu, Z. Zhong, and K. Xu, "A hybrid solution method for CFD applications on GPU-accelerated hybrid HPC platforms," *Future Gener. Comput. Syst.*, vol. 56, pp. 759–765, Mar. 2016.
[49] M. Radmanović, D. Gajić, and R. S. Stanković, "Efficient computation of Galois field expressions on hybrid CPU-GPU platforms," *J. Multiple-Valued Logic Soft Comput.*, vol. 26, nos. 3–5, pp. 417–438, 2016.
[50] A. Ilic and L. Sousa, "Simultaneous multi-level divisible load balancing for heterogeneous desktop systems," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl. (ISPA)*, Jul. 2012, pp. 683–690.
[51] J. Colaço, A. Matoga, A. Ilic, N. Roma, P. Tomás, and R. Chaves, "Transparent application acceleration by intelligent scheduling of shared library calls on heterogeneous systems," in *Parallel Processing and Applied Mathematics*. Berlin, Germany: Springer, 2013, pp. 693–703.
[52] V. Cardellini, A. Fanfarillo, and S. Filippone, "Heterogeneous sparse matrix computations on hybrid GPU/CPU platforms," in *Proc. PARCO*, 2013, pp. 203–212.
[53] C. Baldwin, T. Eliassi-Rad, G. Abdulla, and T. Critchlow, "The evolution of a hierarchical partitioning algorithm for large-scale scientific data: Three steps of increasing complexity," in *Proc. 15th Int. Conf. Sci. Stat. Database Manage.*, Jul. 2003, pp. 225–228.
[54] D. Clarke, A. Ilic, A. Lastovetsky, and L. Sousa, "Hierarchical partitioning algorithm for scientific computing on highly heterogeneous CPU + GPU clusters," in *Euro-Par 2012 Parallel Processing*, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Berlin, Germany: Springer, 2012, pp. 489–501.
[55] B. Dorronsoro, S. Nesmachnow, J. Taheri, A. Y. Zomaya, E.-G. Talbi, and P. Bouvry, "A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems," *Sustain. Comput., Inform. Syst.*, vol. 4, no. 4, pp. 252–261, 2014.
[56] J. von Kistowski, J. Beckett, K. Lange, H. Block, J. A. Arnold, and S. Kounev, "Energy efficiency of hierarchical server load distribution strategies," in *Proc. IEEE 23rd Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Oct. 2015, pp. 75–84.

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

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

ALEXEY LASTOVETSKY received the Ph.D. degree from the Moscow Aviation Institute, in 1986, and the Ph.D. degree from the Russian Academy of Sciences, in 1997. He has published over a hundred technical articles in refereed journals, edited books, and international conferences. He has authored the monographs *Parallel Computing on Heterogeneous Networks* (Wiley, 2003) and *High Performance Heterogeneous Computing* (Wiley, 2009). His main research interests include algorithms, models, and programming tools for high-performance heterogeneous computing.