DOI 10.12694/scpe.v17i4.1205
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 4, pp. 331–349. http://www.scpe.org

TILING AND SCHEDULING OF THREE-LEVEL PERFECTLY NESTED LOOPS WITH DEPENDENCIES ON HETEROGENEOUS SYSTEMS

EBRAHIM ZAREI ZEFREH*, SHAHRIAR LOTFI†, LEYLI MOHAMMAD KHANLI‡, AND JABER KARIMPOUR§

Abstract. Nested loops are one of the most time-consuming parts and the largest sources of parallelism in many scientific applications. In this paper, we address the problem of 3-dimensional tiling and scheduling of three-level perfectly nested loops with dependencies on heterogeneous systems. To exploit the parallelism, we tile and schedule nested loops with dependencies by awareness of computational power of the processing nodes and execute them in pipeline mode. The tile size plays an important role to improve the parallel execution time of nested loops. We develop and evaluate a theoretical model to estimate the parallel execution time of tilled nested loops. Also, we propose a tiling genetic algorithm that used the proposed model to find the near-optimal tile size, minimizing the parallel execution time of dependence nested loops. We demonstrate the accuracy of theoretical model and effectiveness of the proposed tiling genetic algorithm by several experiments on heterogeneous systems. The 3D tiling reduces the parallel execution time by a factor of 1.2× to 2× over the 2D tiling, while parallelizing 3D heat equation as a benchmark.

Key words: Dependence loop, tiling, load balancing, communication, heterogeneous system

AMS subject classifications. 65Y05, 68M14

1. Introduction. Today, there are so many scientific applications in various fields such as meteorology, biology, medical research, signal and image processing, military industry, etc. that need high performance computing to be solved. These problems are either computationally intensive, or working on large-scale multi-dimensional data or both [1, 2]. Nested loops are one of the most time-consuming parts and the largest sources of parallelism in these problems [3, 4]. In order to meet the ever-increasing computing requirement of scientific applications, it is necessary to use high-level computational capacity and optimization techniques.

A heterogeneous computing system is a set of multiple computing nodes connected via a high-speed network interconnection, used for executing parallel and distributed scientific applications [5, 6, 7]. A homogeneous computing system is a special case of a heterogeneous computing system, in which all computing nodes have the same computing capabilities [5]. There are several ways to enhance the computational capacity of parallel computing systems such as (1) scaling up by adding more processing nodes, (2) replacement of all processing nodes with newer, faster ones, (3) upgrading computing systems by adding newer, faster nodes, (4) combing multiple clusters into a bigger computational system, known as multi-cluster systems, (5) using hybrid CPU-GPU architectures, etc. [8, 9, 10, 11]. In cases 1 and 2, the computing system remains homogeneous, but it can be very costly. In other cases, the computing system becomes heterogeneous. According to the Top500 list (http://www.top500.org), we could witness an increasing trend to heterogeneous computing systems from a 3.4% to 18.0% between June 2010 and June 2015 [12]. Hence, heterogeneity is one of the most important and challenging issues in parallel computing systems [13].

Loop optimization and parallelization have always been an important role to achieve higher performance [4]. A lot of loop optimization techniques have been developed to decrease the execution time of the nested loops and improve the performance. Loop tiling is an important loop optimization technique in scientific applications, used to improve data locality, expose fine-grained and coarse-grained parallelism, enhance cache reuse, etc. [14, 15, 16, 17, 18, 19].

In this paper, we consider the parallelization problem of perfectly nested loops with dependencies on heterogeneous systems. In order to achieve the maximum performance of nested loops with dependencies on heterogeneous systems, two issues must be adequately addressed:

- Load balancing: it is a technique that tries to distribute the data and computation across the computing resources of the parallel machine so that all tasks terminate at approximately the same time. In fact,

*Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran (zarei@tabrizu.ac.ir).
†Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran (shahriar_lotfi@tabrizu.ac.ir).
‡Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran (l-khanli@tabrizu.ac.ir).
§Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran (karimpour@tabrizu.ac.ir).

the goals of load balancing are optimization of resource utilization, maximization of throughput and minimization of response time [17, 20]. Processing nodes of heterogeneous systems may have different computational powers that depend on CPU speed, cache size, RAM size etc. So, load balancing is an important concept in heterogeneous systems that guarantees the amount of data and computation of any processing node correspond to their computational power [21].

- Communication: In a distributed-memory parallel architectures, each processing node has its own memory and nodes communicate together to exchanging data during program execution. Since accessing to the local memory is much faster than the remote memory, the cost of intra-node communication is much less than inter-node communication. Due to network latency of inter-node communication, data should place as close as possible to computation, referred to as the data locality [20, 22, 23, 24]. So, data and computation can be partitioned into blocks and distributed across the processing node to improve data locality and reduce communications during program execution.

In order to parallelize perfectly nested loops with dependencies on heterogeneous systems, the loop's iteration space partitioned into a series of small chunks of given tile size, executed one after another in pipeline mode. At the runtime, processing nodes communicate each other to exchange data while executing tiles. The number of inter-node communication (or tiles communication) is corresponding to the inter-tile dependency. Since communication is one of the most important reasons for performance degradation of the parallelized loops with data dependencies on heterogeneous systems, inter-tile dependency should be minimize as much as possible. To overcome communication overhead and improve the pipeline parallelism, we should determine the optimal tile size. The problem of determining the optimal tile size is NP-Hard [25]. There are many approaches that attempt to determine the near-optimal tile size in homogeneous platforms such as analytical, auto-tuning and evolutionary approaches [25, 26]. In heterogeneous platforms, tile size determined by the computational power awareness of the processing nodes [21, 27, 28]. The 3D tiling of the nested loop with dependencies for heterogeneous systems has not been given enough attention so far.

We believe that the use of 3D tiling and scheduling of perfectly nested loops with dependencies and taking into account the characteristics of heterogeneity in heterogeneous systems can enhance the execution time of the scientific applications. For this purpose, we first calculate the computational power of the processing nodes by running 3D benchmarks. Then, with computational power awareness of the processing nodes, we tile and schedule perfectly nested loops with data dependencies. Therefore, loop tiling combined with heterogeneity feature and a pipeline-like execution could help to decrees the execution time and improve efficiency of computation on heterogeneous computing systems.

In this paper, we propose a 3D tiling and scheduling approach for three-level perfectly nested loops with data dependencies on heterogeneous systems using the computational power awareness of the processing nodes. Our idea is to exploit the computational power of the processing nodes of heterogeneous platforms in order to achieve higher computing power for executing nested loops. In addition, we use loop tiling to partition the iteration space into chunks and subchunks with equal and unequal size such that the load balancing between the computational nodes increases and the internode communication is minimized as much as possible. Then, we use pipeline approaches to achieve the maximum degree of potential parallelism and consequently, the improved execution time of programs. We provide a theoretical model to estimate the parallel execution time of nested loop with dependencies and propose a tiling genetic algorithm to determine the near-optimal tile size.

The main contributions of our paper are as follows:

- We propose a 3D tiling and scheduling approach for three-level perfectly nested loops with dependencies on heterogeneous systems.
- We develop a theoretical model to estimate the parallel execution time.
- We propose a tiling genetic algorithm to determine the near-optimal tile size.

The rest of the paper is organized as follows. Section 2 describes the program model and notation and discusses an overview of related works. Section 3 describes the proposed method. Section 4 is concerned with simulations and experimental results. Finally, Section 5 is conclusions and future works.

2. Background and related work.

2.1. Program model and notation. An n-nested loop, a nested loop of depth n, is defined as a set of n loops where each loop is contained in its previous loops. If all statements are nested inside the innermost loop,

then it is called perfectly nested loop. Each iteration of n-nested loop is represented as $J = (j_1, j_2, \cdots, j_n) \in Z^n$. When a data dependency exists in a nested loop, the result of one loop iteration affects the results of other loop iterations. In fact, dependencies impose precedence constraints in the execution order of loops iterations [29]. In an n-nested loop, data dependencies are denoted by a distance dependence vector. Suppose that the matrix $D = [d_{ij}]_{n \times m}$ shows the m dependency vectors of the n-nested loop. Intra-iteration dependence occurs in the same iteration between the statements of nested loop while inter-iteration dependence occurs in different iterations [30]. Figure 2.1(a) illustrates a three-level perfectly nested loop and its iteration space denoted by $J = \{(j_1, j_2, j_3) | 1 \leq j_1 \leq N_1, 1 \leq j_2 \leq N_2, 1 \leq j_3 \leq N_3\}$. Figure 2.1 (b), (c) and (d) illustrate the iteration space, the intra-iteration and inter-iteration dependencies and the dependency matrix for the following nested loop. Nested loops categories in parallel and dependence loops. If there are no inter-iteration dependence among their loops iterations, the nested loop is called a parallel loop otherwise the nested loop is called a dependence loop [21, 31].

One of the most important of loop optimization techniques is loop tiling that could improve data locality and expose parallelism. Loop tiling decomposes an n-nested loop into a 2n-nested loop where the outer n loops move between tiles and the inner n loops traverse iteration within a tile. Suppose that the $H \in Q^{n \times n}$ be the tiling matrix that each row is a normal vector and shows the edges of the tile. V_{comp} expresses the number of iterations within a tile and V_{comm} expresses the number of iterations that need to send data to the neighboring tiles (the number of dependences exit from the tile). V_{comp} and V_{comm} are calculated by the following formulas [25, 27, 32]:

$$V_{comp}(H) = \frac{1}{|det(H)|} \tag{2.1}$$

$$V_{comm}(H) = \frac{1}{|det(H)|} \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{m} h_{i,k} d_{k,j} \tag{2.2}$$

Figure 2.1(e) shows the code after loop tiling transform. Figure 2.1(f) shows a $2 \times 2 \times 2$ parallelepiped tiling of the 3-nested loop and Fig. 2.1(g) illustrates the tiling matrix H for the parallelepiped tiling in Fig. 2.1(f). So, $V_{comp}(H) = 8$ and $V_{comm}(H) = 12$.

Pipeline parallelism can improve the efficiency of the nested loop with dependencies. In pipeline parallelism, each node performs its tasks, then passes its set of data along to the next node and receives the next set of data from the previous node [17]. In distributed-memory parallel systems, communication and synchronization overhead between the nodes are the important reasons of the performance degradation when running dependence loops. So, we use coarse-grain pipeline parallelism to balance trade-offs between parallelization, communication and synchronization overhead [20, 33].

2.2. Related work. There are a lot of research efforts on determining the optimal partitioning (tiling) of nested loops without dependencies on heterogeneous systems ([34, 35, 36] and references therein). However, there are a few research efforts targeting tiling problem for nested loops with dependencies on heterogeneous systems. Most of these works are bounded into 2D tiling.

Boulet et al. [37, 38] used loop tiling on heterogeneous systems for the first time. Iteration space is divided into tiles with same size and assigned column blocks with more tiles to the faster node. Then nodes execute the tiles in a row-wise order within each block to minimize latency between starting of blocks. The authors target fully permutable 2-nested loops with horizontal and vertical dependencies.

Chen and Xue [27] proposed the 2D partitioning and scheduling loops for a network of heterogeneous workstations (NOWs). As shown in Fig. 2.2, to consider heterogeneously of NOWs, the iteration space is partitioned into 2D tiles of the same shape and different sizes according to computational powers of theirs workstations. The same colored tiles can be executed simultaneously. The authors consider the doubly nested loop or two adjacent loops of nested loop with constant data dependency.

Ciorba et al. [31, 39] proposed enhancing self-scheduling algorithm for loops with dependencies on heterogeneous systems. The self-scheduling algorithms such as chunk self-scheduling, guided self-scheduling, trapezoid self-scheduling and factoring self-scheduling are dynamic scheduling algorithms that are used to schedule nested

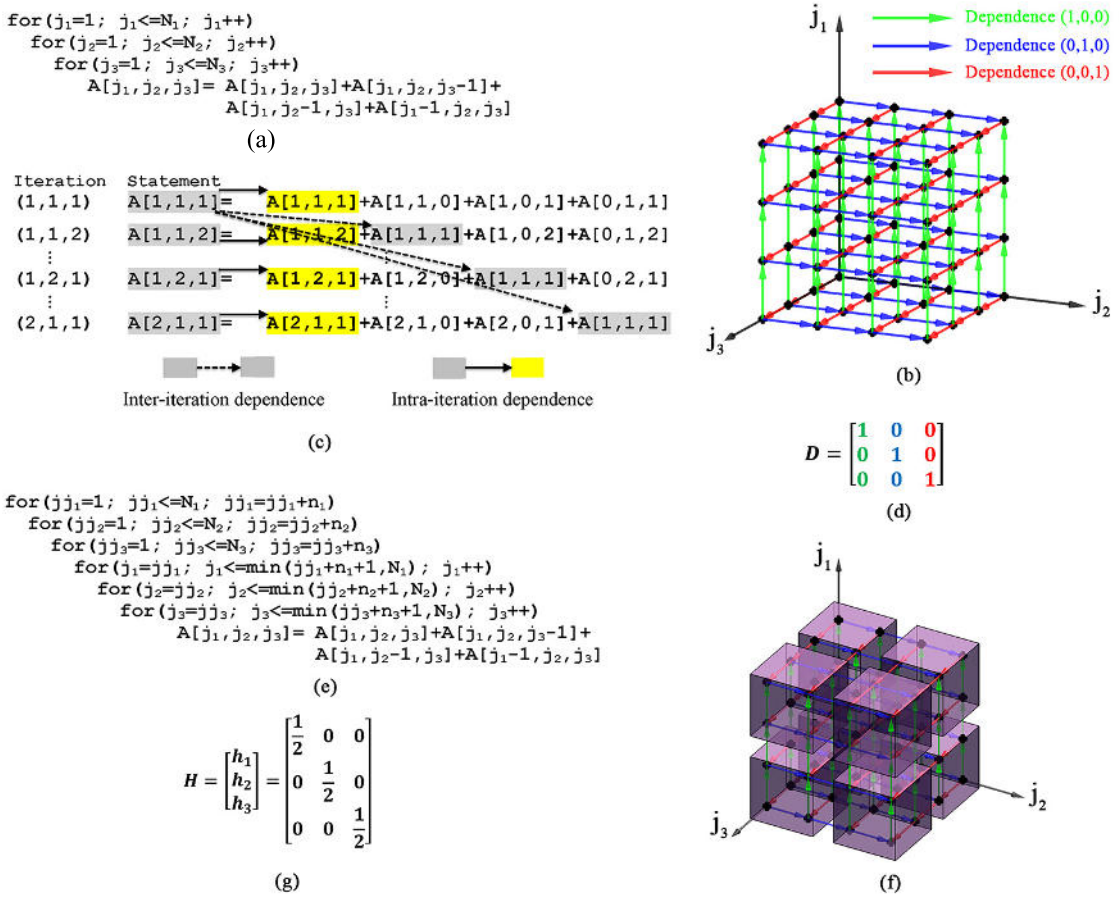

FIG. 2.1. *(a) Three-level perfectly nested loop, (b) iteration space and dependencies between iterations, (c) the intra-iteration and inter-iteration dependencies, (d) dependency matrix, (e) the tiled perfectly nested loop, (f) 3D tiling and (g) the tiling matrix*

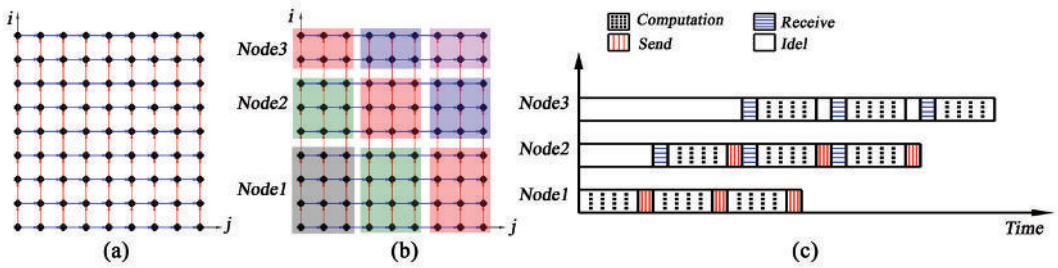

FIG. 2.2. *(a) Iteration space of nested loop with two constant data dependency, (b) 2D heterogeneous tiling and (c) parallelization strategy [27]*

loops without dependencies on homogeneous systems. They enhance self-scheduling algorithm to handle nested loops with dependencies by inserting synchronization points to enable inter-node communication. They also consider a weighted mechanism for self-scheduling algorithms to improve the performance and make them suitable for heterogeneous systems. Therefore, the iteration space is divided into chunks according to the computational power of nodes.

Andronikos et al. [21, 33, 40] claimed that the problem of finding the optimal partitioning of nested loops

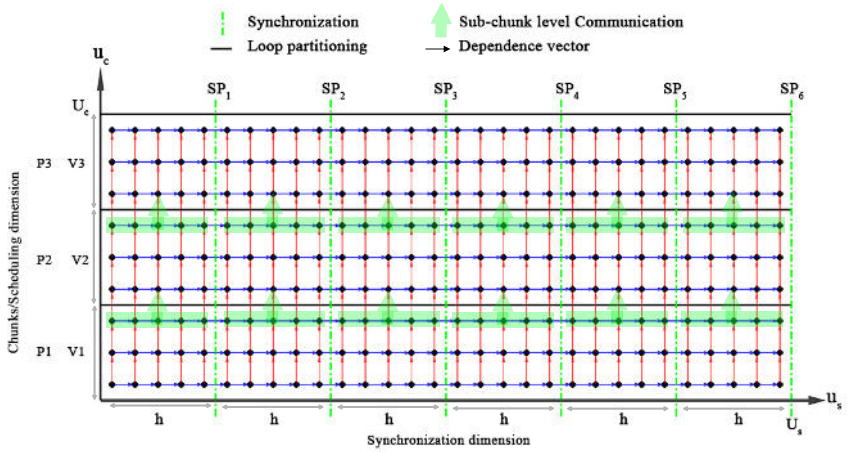

FIG. 2.3. *Tiling of a 2-nested loop with dependencies on a homogeneous system [21]*

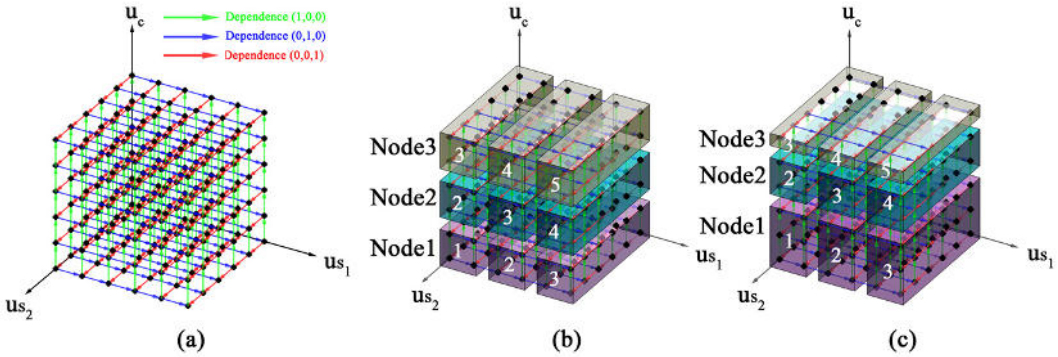

FIG. 2.4. *(a) Iteration space and data dependencies of three-level nested loop, (b) and (c) partitioning and scheduling of nested loop with proposed methods in [21, 33, 40] on homogeneous and heterogeneous systems, respectively. The tiles with the same number can be executed simultaneously.*

with dependencies for heterogeneous systems has not been given enough attention. Therefore, they proposed a theoretical model to estimate parallel execution time as a function of the synchronization frequency for nested loops with dependencies on heterogeneous systems. As shown in Fig. 2.3, the iteration space partitioned into chunks along chunk/scheduling dimension based on the computational powers of nodes using self-scheduling schema. The chunks are divided into subchunks along synchronization dimension by inserting synchronization points. They find the optimal subchunk size based on the theoretical model. This paper targets n-nested loops ($n >= 2$) with dependencies where the outer loop is considered as synchronization dimension and another loop as scheduling dimension. Figure 2.4 shows how to use this method for three-level nested loop with dependencies on homogeneous and heterogeneous systems. In this case, the u_c-dimension partitioned into chunks corresponding to the computational power of the processing nodes, the u_{s_1}-dimension partitioned into subchunks and the u_{s_2}-dimension executed as serial. Then subchunks are executed in a wavefront fashion to exploit the potential parallelism.

As mentioned above, these works are generally focused on 2D tiling of the nested loop with dependencies on heterogeneous systems. The 3D tiling of the nested loop with dependencies for heterogeneous systems has not been given enough attention so far. In this paper, we address this issue.

3. Proposed methods. In this section, we propose an approach to 3D tiling and scheduling of three-level perfectly nested loops with dependencies on heterogeneous systems. In the paper, we use the notation in [21, 33], indicated in Table 3.1. Algorithm 1 outlines the main steps of proposed method.

TABLE 3.1
Notations used within the proposed method

Parameter	Description
P	The number of processing nodes
p_i	The i^{th} processing node
$N = U_c \times U_{s_1} \times U_{s_2}$	The size of iteration space
U_c	The upper bound of u_c dimension
U_{s_1}	The upper bound of u_{s_1} dimension
U_{s_2}	The upper bound of u_{s_2} dimension
vp_i	The computational power of i^{th} processing node
V_i	The size of chunk i in the u_c dimension
h_1 and h_2	The size of tile in the u_{s_1} and u_{s_2} dimensions
c_{p_i}	The execution cost per iteration of i^{th} processing node
t_{p_i}	The computation time of a tile in node i
c_d	The start-up latency cost
c_c	The transfer cost per unit of data
t_s	The send time of message between a pair of nodes
t_r	The receive time of message between a pair of nodes

Algorithm1: 3D tiling and scheduling

Input:

A heterogeneous system consist of P nodes $p_1, \ldots, p_P$ with computational powers $vp_1, \ldots, vp_P$
c_p: The execution cost per iteration of nodes
c_c and c_d: Communication parameter
$U_c \times U_{s_1} \times U_{s_2}$: The size of iteration space

- Sorting computational power of nodes such that $vp_1 \geq vp_2 \geq \ldots \geq vp_P$
- Partitioning the u_c scheduling dimension into chunks of given size V_i by $V_i = U_c \times vp_i$ (Figs. 3.1(b) and 3.2(b))
- Partitioning each chunk into subchunks with unknown sizes h_1 and h_2 along the u_{s_1} and u_{s_2} synchronization dimensions (Figs. 3.1(c) and 3.2(c))
- Calculating the computation time of a tile by $t_p = V_i h_1 h_2 c_p$
- Calculating the communication time of a tile by $t_s = t_r = c_d + h_1 h_2 c_c$
- Estimation the parallel execution time, $T_P(h_1, h_2)$, based on parallel execution flow of subchunks with unknown sizes h_1 and h_2 (Fig. 3.3)

$$T_P(h_1, h_2) = (t_p + t_s) + \left(\sum_{i=2}^{P-1}(t_r + t_p + t_s)\right) + \frac{U_{s_1}U_{s_2}}{h_1 h_2}(t_r + t_p) + \left(\frac{U_{s_1}U_{s_2}}{h_1 h_2} - 1\right) t_{idle} + T_{wa}$$

- Formulate the problem of finding the optimal tiling as follows (Eq. 3.5):

$$\begin{aligned} &Minimize \;\; T_P(h_1, h_2) \\ &Subject\ to \;\; V_i h_1 h_2 \leq CacheSize_i, \;\; i = 1, 2, \ldots, P \\ &\qquad\qquad h_1 \text{ and } h_2 \text{ are integer} \\ &\qquad\qquad 1 \leq h_1 \leq U_{s_1} \\ &\qquad\qquad 1 \leq h_2 \leq U_{s_2} \end{aligned}$$

- Solving the above optimization problem using:
  1- NOMAD (Nonlinear Optimization using the MADS Algorithm)
  2- The proposed tiling genetic algorithm

Output: optimal tile sizes

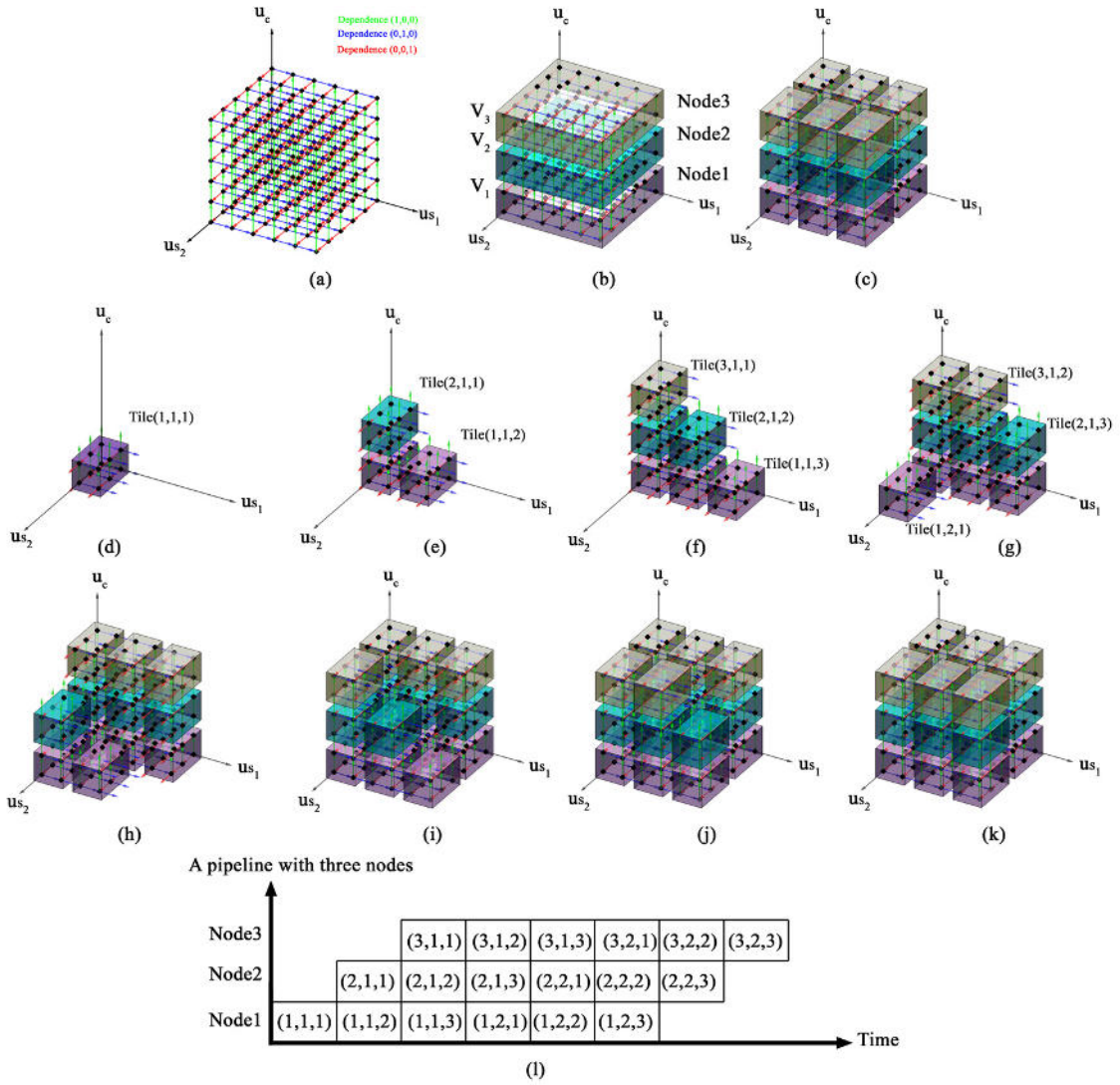

FIG. 3.1. *(a) Iteration space and dependencies vectors, (b) partitioning iteration space into three equal chunks in a homogeneous system (c) partitioning each chunk into subchunks, (d) to (k) the execution process of tiles in node 1, 2 and 3, and (l) pipelined execution of tiles in time*

Suppose, there exists P processing nodes $p_1, p_2, \cdots, p_P$ of the computational powers $vp_1, vp_2, \cdots, vp_P$ in the heterogeneous system such that $\sum_{i=1}^{P} vp_i = 1$ and $vp_1 \geq vp_2 \geq \cdots \geq vp_P$. In this paper, we consider the three-level perfectly nested loops with uniform dependencies in three dimensions. We partition the iteration space into chunks along one dimension by using self-scheduling algorithms. This dimension is called the scheduling dimension and is denoted by u_c. Let V_i be the size of the chunk i in the u_c dimension assigned to i^{th} processing node of the heterogeneous system. It should be noted that the size of each chucks is corresponding to the computational power of the processing nodes. If the distributed system has homogeneous nodes, then the sizes of chunks are equal (see Fig. 3.1(b)), otherwise the sizes of chunks are unequal (see Fig. 3.2(b)). The two other dimensions are denoted by u_{s_1} and u_{s_2} consider as synchronization dimensions. Each chunk is partitioned into subchunks with setting synchronization points along u_{s_1} and u_{s_2} dimensions (see Fig. 3.1(c) and Fig. 3.2(c)). Each of 3D boxes of points in the iteration space is considered as a tile.

In the execution flow, the tile first receives the needed data from other tiles, then does computation and

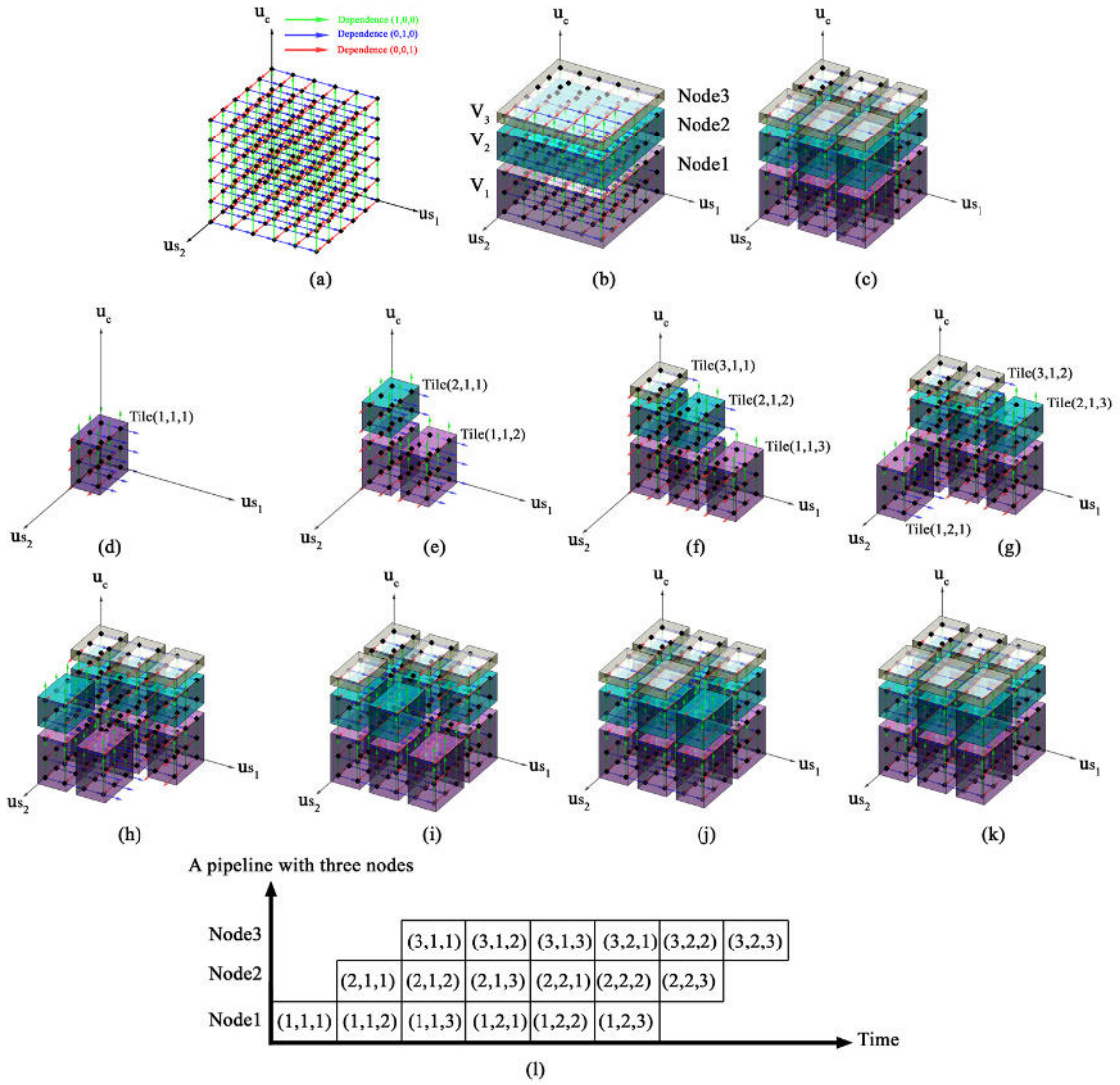

FIG. 3.2. *(a) Iteration space and dependencies vectors, (b) partitioning iteration space into three unequal chunks in a heterogeneous system, (c) partitioning each chunk to subchunks, (d) to (k) the execution process of tiles in node 1, 2 and 3, and (l) pipelined execution of tiles in time*

finally sends data to other tiles that needed it. Due to the presence of dependencies, no nodes can start the execution at the same time and we should consider a precedence order. Notice that according to the partitioning of the iteration space, dependencies $(0,1,0)^T$ and $(0,0,1)^T$ occur in each node and the dependency $(1,0,0)^T$ occurs between two neighboring nodes. As shown in Fig. 3.1(d), node 1 runs tile (1,1,1) and then sends necessary data to tile (2,1,1) that schedule on node 2. Then node 1 and node 2 simultaneously run tiles (1,1,2) and (2,1,1) respectively, as shown in Fig. 3.1(e). After that node 1, node 2 and node 3 simultaneously runs tiles (1,1,3), (2,1,2) and (3,1,1) respectively, as shown in Fig. 3.1(f). This process continues until the node 3 runs tile (3,2,3). Actually, the tiles establish a communication and synchronization mechanism between the processing nodes.

Idle time is an important factor that affects the execution time in the tiled loop. At any time during the execution of the tiled nested loops, some nodes are active and some are idle. Idle time represents the time when the node is in idle mode during the execution of the tiled iteration space. The idle time can arise due to two reasons: (1) because of the presence of dependence, a node may have to wait for the necessary data from other

nodes; (2) some nodes may have completed their works and are waiting for the last node to finish its work [41]. In homogeneous platforms, the size of tiles is the same, so choosing the shape of tiles is very important to reduce the idle time in the parallel execution. However, in heterogeneous platforms, both size and shape of the tiles have significant effect to reduce the idle time. Load balancing can reduce the idle time and guarantee that the amount of workloads of any processing node corresponds to its computational power. Therefore, in heterogeneous platforms, we use tiles with the same shape and different sizes such that nodes complete the execution of their tiles at the same time. Figure 3.2 shows the heterogeneous tiling for a heterogeneous platform with normalized computational powers $VP = \{0.5, 0.33, 0.17\}$.

To estimate the parallel execution time of nested loops with dependencies on heterogeneous systems, we need a communication and computation cost model. We use the notations in [21] and extend them.

3.1. Computation cost model. The computation time of a tile in node i is defined as a function of the number of iterations within a tile, $V_{comp}(H)$, multiplied by the execution cost per iteration, c_{p_i}. We can calculate it as follows:

$$t_{p_i} = V_{comp}(H) \times c_{p_i} \tag{3.1}$$

3.2. Communication cost model. We consider heterogeneous computing systems of P processing nodes p_1, p_2, $\cdots$, p_P that is connected with homogeneous communication links. In this work, we use the one-port model as the communication cost model to quantify the communication overhead between the processing nodes. In one-port model, a node can either send or receive a message at each time step and distinct node pairs communicate simultaneously. There are two different costs to transfer a message from one node to another: (1) the start-up latency cost between a pair of nodes, c_d; (2) the transfer cost per unit of data between a pair of nodes, c_c [17, 21]. We suppose that the send (t_s) and receive (t_r) times of a message between each pairs of nodes are equal since the number of message elements are the same in the process of sending and receiving. The communication time of a tile is defined as a function of the start-up latency cost and the number of iterations that need to send data to the neighboring tiles and the transfer cost per unit of data as follows:

$$t_s = t_r = c_d + V_{comm}(H) \times c_c \tag{3.2}$$

3.3. The proposed theoretical model. In this paper, we consider parallelizing the three-level perfectly nested loops with dependencies in three dimensions on heterogeneous computing systems. We tile and schedule these loops with the computational power awareness of the processing nodes and execute them in pipeline mode. To estimate the parallel execution time of nested loops, we build a theoretical model as a function of tile sizes.

As shown in Figs. 3.1 and 3.2, we partition the iteration space of nested loop into 3D tiles. Let V_i be the size of one side of the tile (i, j, k) along the u_c dimension assigned to i^{th} processing node. To satisfy load balancing, we calculate V_i as a function of the computational power of processing node by $V_i = U_c \times vp_i$. Suppose that h_1 and h_2 are the size of other sides of the tile along the synchronization dimensions u_{s_1} and u_{s_2}. h_1 and h_2 are the same for all tiles. So, the computation time of a tile in node i is calculated by $t_{p_i} = V_i h_1 h_2 c_{p_i}$ and the communication time of a tile in node i is calculated by $t_s = t_r = c_d + h_1 h_2 c_c$ because only dependency $(1, 0, 0)^T$ occurs between two neighboring nodes.

Figure 3.3 shows the parallel execution flow of tiled nested loops on homogeneous and heterogeneous platforms for Figs. 3.1 and 3.2. In the following, we consider the parallel execution flow and construct a formula to estimate the parallel execution time. Here, we use the master-worker model. The processing nodes (or workers) send a request message for assigning the work to the master. Master, that has all the information about the nodes, receives the requests, calculates the chunk sizes and assigns them to the processing nodes. The duration between sending a request and assigning a chunk to nodes is considered as the work assignment time and is denoted by T_{wa}. The nodes are responsible for executing the assigned chunk. Node 1 starts the execution of the tile $(1, 1, 1)$. Due to the presence of dependence, node 2 should be expected to receive data from node 1. This is the idle time and is shown by white strip in Fig. 3.3. Node 1 completes the execution of the tile $(1, 1, 1)$, sends the necessary data to node 2 and starts the execution of tile $(1, 1, 2)$. Node 2 after receiving the necessary data from node 1 executes tile $(2, 1, 1)$ and sends the necessary data to node 3. Node 3 is the last node and does not need to send data. Node 3 should be expected to receive data from node 2, so there is an idle time between the operations of execution and receiving.

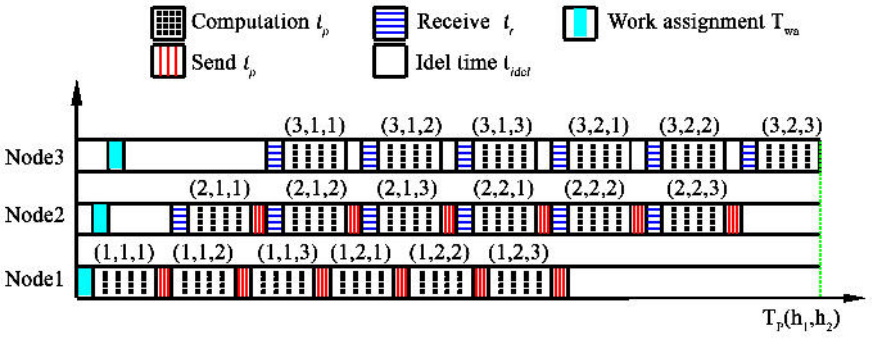

FIG. 3.3. *Parallel execution flow for three nodes*

Suppose that the theoretical parallel time, $T_P(h_1, h_2)$, is the parallel execution time of the last tile that is carry out by node P. All nodes have to *receive*, *compute* and *send* except for the first and last nodes. Node 1 only computes and sends data. So, the time required to compute each tile in node 1 and send the necessary data to node 2 is $t_{p_1} + t_s$. The time needed to receive data, compute and send data of the first tile in node $2, 3, \cdots, P-1$ is $\sum_{i=2}^{P-1}(t_r + t_{p_i} + t_s)$. The last node, node P, only receives data and computes. So, the time needed to receive the necessary data from node $P-1$ and compute all tiles in node P is $\frac{U_{s_1}U_{s_2}}{h_1h_2}(t_r + t_{p_P})$. Node P also spent $(\frac{U_{s_1}U_{s_2}}{h_1h_2} - 1)t_{idle}$ idle time for receiving data from node $P-1$. t_{idle} approximately equals to t_s. Therefore, the total parallel execution time is

$$T_P(h_1,h_2) = (t_{p_1} + t_s) + \left(\sum_{i=2}^{P-1}(t_r + t_{p_i} + t_s)\right) + \frac{U_{s_1}U_{s_2}}{h_1h_2}(t_r + t_{p_P}) + \left(\frac{U_{s_1}U_{s_2}}{h_1h_2} - 1\right)t_{idle} + T_{wa} \tag{3.3}$$

Since the processing nodes of homogeneous computing systems have the same computational power, V_i and c_{p_i} are the equivalent for all nodes. In the heterogeneous computing systems, processing nodes have different computational power. Therefore, the execution costs per iteration c_{p_i} of nodes are different. According to load balancing, the best state is when all nodes execute their assigned tiles at the same time, in the other words $t_{p_1} = t_{p_2} = \cdots = t_{p_i} = \cdots = t_{p_P}$. It is noticed that a perfect load balancing is not always possible. In this case, we want that all nodes execute their tiles at approximately the same time as much as possible $t_{p_1} \cong t_{p_2} \cong \cdots \cong tp_i \cong \cdots \cong t_{p_P}$. When running multiple tiles in parallel, maybe a node, which finishes the execution of its tile, has to wait for the other one to complete its execution before they could exchange data. To control the situation in heterogeneous systems, we consider $t_p = max\,(t_{p_1}, t_{p_2}, \ldots, t_{p_i}, \ldots, t_{p_P})$. Therefore, we have

$$T_P(h_1,h_2) = (t_p + t_s) + \left(\sum_{i=2}^{P-1}(t_r + t_p + t_s)\right) + \frac{U_{s_1}U_{s_2}}{h_1h_2}(t_r + t_p) + \left(\frac{U_{s_1}U_{s_2}}{h_1h_2} - 1\right)t_{idle} + T_{wa} \tag{3.4}$$

h_1 and h_2 require fine-tuning so that nodes can start their computation as soon as possible and achieve minimum parallel execution time. We also consider two constrains:

1. If we want to improve data locality in each node, then data items should stay in the cache between successive uses. In order to get a good performance, tile sizes are better to fit in the cache of nodes.
2. Since the sides of the tile are positive integer, we need integer solutions for h_1 and h_2.

Considering these observations, we have a nonlinear pure integer-programming problem (NLIP) as follow:

$$\begin{aligned} &\textit{Minimize} && T_P(h_1,h_2) && \\ &\textit{Subject to} && V_ih_1h_2 \le CacheSize_i, && i = 1,2,\ldots,P \\ &&& 1 \le h_1 \le U_{s_1}, 1 \le h_2 \le U_{s_2}, h_1 \text{ and } h_2 \text{ are integer} && \end{aligned} \tag{3.5}$$

Nonlinear integer programming problems are NP-complete. These problems can solve using nonlinear integer programming solvers or evolutionary approaches. In this paper, we use the NOMAD (Nonlinear Optimization using the MADS Algorithm) [42] as a nonlinear integer programming solver and proposed an evolutionary approach based on the genetic algorithm to find a near-optimal solution that minimize $T_P(h_1, h_2)$.

3.4. The proposed tiling genetic algorithm. In this section, we use Genetic Algorithm (GA) to solve the nonlinear integer-programming problem, Eq. 3.5, derived from the 3D tiling of nested loops with dependencies on heterogeneous systems.

The GA is a population-based heuristic search that follows an iterative process toward better solutions. The GA begins with an initial random population of the problem solution, called chromosomes. In each iteration, the fitness of every chromosome in the population is evaluated by using objective function. The fitter chromosomes are stochastically selected and then evolutionary operators such as crossover and mutation are used to generate new population. The GA is terminated for a maximum number of generations [25, 43].

Problem encoding. Each problem solution is represented by a chromosome. Here, chromosome is specified as a pair of integer number $\langle h_1, h_2 \rangle$ where $1 \leq h_1 \leq U_{s_1}$ and $1 \leq h_2 \leq U_{s_2}$.

Initial population. We use a random integer number generator to create the initial population of chromosomes. To generate a chromosome, the h_1 and h_2 are defined randomly by using formulas $h_1 = Round\,(1 + U_{s_1} \times Rand())$ and $h_2 = Round\,(1 + U_{s_2} \times Rand())$ where U_{s_1} and U_{s_2} are the upper bound of h_1 and h_2, respectively. The function $Rand()$ returns standard uniform distribution on the interval $(0, 1)$ and the function $Round(x)$ returns rounding of the elements of x to the nearest integer. So, $h_i = Round(1 + U_{s_i} \times Rand())$ generates integer values from the uniform distribution on the interval $[1, U_{s_i}]$ for $i = 1, 2$.

Fitness function. The main objective is to find integer values h_1 and h_2 such that the parallel execution time $T_P(h_1, h_2)$ of the heterogeneous system with P processing nodes is minimized. In addition, we have a constraint to fit the tiles into the cache memory of the processing nodes of the heterogeneous system. When the requiring space for the iteration points within tiles is not exceed the cache size of the processing nodes, the tiles are feasible (on the other hand, the chromosomes are feasible). According to Eq. 3.6, we consider the objective function as a summation of two positive numbers, the parallel execution time and the penalty value computed for the chromosomes. We use a constant value for penalty which is zero for feasible chromosomes and $c > 0$ for an infeasible one.

$$
\begin{aligned}
Objective(\langle h_1, h_2 \rangle) &= T_P(\langle h_1, h_2 \rangle) + Penalty(\langle h_1, h_2 \rangle, M) \\
M &= min_{i=1,\ldots,P} \frac{Cache\ size\ of\ node\ i\ in\ byte}{V_i \times (\#Byte\ of\ data\ type)} \\
Penalty(\langle h_1, h_2 \rangle, M) &= \{ \begin{array}{ll} 0 & if h_1 \times h_2 \leq M \\ c & if h_1 \times h_2 > M \end{array}
\end{aligned} \tag{3.6}
$$

We assign a fitness value to each chromosome in the population, calculated by Eq. 3.7. The better chromosome, the bigger fitness value.

$$
Fitness(\langle h_1, h_2 \rangle) = \frac{1}{Objective(\langle h_1, h_2 \rangle) + 1} \tag{3.7}
$$

Selection, crossover and mutation operators. After assigning the fitness value to each chromosome in the current population, the roulette wheel selection method is used to choose a couple of parent chromosomes for the crossing over operation. The bigger the fitness value of chromosomes are, the more chances to be chosen they have. Crossover and mutation are two important genetic operators. Crossover is an exploitation operator that is used to create new population by combining a couple of parent chromosomes. Mutation is an exploration operator that is used to maintain diversity in the new population [43]. Here, the crossover operator is applied to the selected parent chromosomes using an arithmetic crossover. The crossover operator is done with the combined probability, $P_{Crossover}$, as follows:

$0 \leq \lambda \leq 1$ is chosen randomly
If $Rand() \leq P_{Crossover}$

$\text{ChildChromosome}_1 = \lambda\times \text{ ParentChromosome}_1 + (1-\lambda)\times \text{ ParentChromosome}_2$
$\text{ChildChromosome}_2 = (1-\lambda)\times \text{ ParentChromosome}_1 + \lambda\times \text{ ParentChromosome}_2$
else
$\text{ChildChromosome}_1 = \text{ParentChromosome}_1$
$\text{ChildChromosome}_2 = \text{ParentChromosome}_2$
end

After applying the crossover operator, the mutation operator with the probability, $P_{Mutation}$, is applied to newly generated chromosomes. It replaced the value of the chosen chromosomes, $\langle h_1, h_2\rangle$, with integer values from the uniform distribution between the upper and lower bounds of h_1 and h_2.

Replacement Scheme. After generating the new population using selection, crossover and mutation operations, the GA replaces the current population with the new one. We use elitism in the replacement scheme. If the fittest chromosome in the current population is better than the fittest chromosome in the new population, then it is moved to the next population directly. Elitism is important since it allows preserving the fittest chromosome over the time.

4. Experiments and results. In this section, our simulation and experimental results are presented. We evaluate the performance of the proposed theoretical model and tiling genetic algorithm by using the 3D heat equation, three-level perfectly nested loops with dependencies, as a benchmark. Table 4.1 shows the specifications of nine classes of processing nodes used in experiments. They are multi-core processors. A 100 Mbits/s fast Ethernet network is used to interconnect processing nodes. The benchmark is implemented in C using OpenMP for intra-node communication and MPI for inter-node communication.

TABLE 4.1
Specifications of processing nodes

	Processing nodes								
	1	2	3	4	5	6	7	8	9
Name of processors	Intel Core 2 Duo T5870	Intel Pentium E5300	Intel Core 2 Duo E7500	Intel Core i3 2350M	Intel Pentium G620	Intel Pentium G2020	Intel Core i5 2410M	Intel Pentium G2030	Intel Core i7 4710HQ
#Processors	1	1	1	1	1	1	1	1	1
CPU Speed (GHz)	2.00	2.60	2.93	2.30	2.60	2.90	2.30	3.00	2.50
#Cores	2	2	2	2	2	2	2	2	4
L1 Cache (KB)	2 x 32	2 x 32	2 x 32	2 x 32	2 x 32	2 x 32	2 x 32	2 x 32	4 x 32
L2 Cache (KB)	2048	2048	3072	2 x 256	2 x 256	2 x 256	2 x 256	2 x 256	4 x 256
L3 Cache (MB)	-	-	-	3	3	3	3	3	6
Memory type	DDR2	DDR2	DDR3	DDR3	DDR3	DDR3	DDR3	DDR3	DDR3
RAM (GB)	4	2	2	4	2	4	4	4	8
Normalized Computational Power for 3D Heat Equation	0.0529	0.0531	0.0920	0.1029	0.1197	0.1383	0.1405	0.1438	0.1568

We use hierarchical tiling to exploit the computational power of all cores in multi-core nodes. For this purpose, we first partition the iteration space of nested loops with dependencies into chunks and assign each chunk to each node. Due to the dependence, each assigned chunk is partitioned to subchunks and run in pipeline mode to achieve the maximum degree of parallelism between nodes of a heterogeneous system. In multi-core node, the subchunk is tiled again and assign to their cores. Figures 4.1(a) and (b) show the pseudo code of a subchunk of size $n_i \times n_j \times n_k$ of the 3D heat equation and the wavefront-parallel 3D heat equation for a subchunk of size $n_i \times n_j \times n_k$, respectively [44].

We execute the 3D heat equation on each node several times, measure the average execution time and calculate the computational power of the processing nodes. These values, which are used as weights that scale

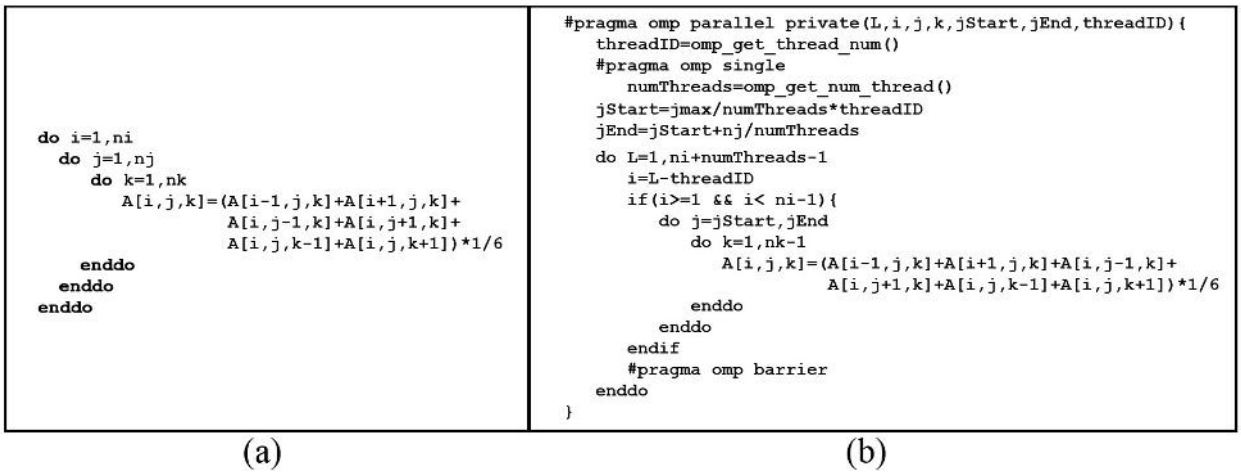

FIG. 4.1. *(a) Pseudo code of 3D heat equation and (b) the wavefront-parallel 3D heat equation [44]*

TABLE 4.2
Specifications of experiments

Experiment	Node type											
#1	1	1	1	1	1	1	1	1	-	-	-	-
#2	3	3	3	3	1	1	1	1	-	-	-	-
#3	8	8	8	8	1	1	1	1	-	-	-	-
#4	7	7	7	7	4	4	4	4	1	1	1	1
#5	9	8	7	6	5	4	3	2	1	-	-	-

the size of each chunks assigned to each processing node, are normalized and showed in the last row of Table 4.1.

Simulations and experimental results are presented for one homogeneous and several heterogeneous computing systems to evaluate the performance of the proposed theoretical model for estimating the parallel execution time and the tiling genetic algorithm for finding the near-optimal tiling. Table 4.2 describes the specification of experiments.

All nodes of computing systems connected together with homogeneous communication links. An MPI program in C used to exchange data with different sizes between every pair of processing nodes. We measured the average time to send and receive messages. The estimated value of the start-up latency, c_d, and the transfer cost per unit of data, c_c, between each pairs of nodes are 300e-06 and 0.80e-06, respectively.

We approximate the execution cost per iteration of each node as a function of tile size (namely, the constant value V_i and variable integer values h_1 and h_2) to consider processor heterogeneity, the heterogeneity in memory structure, and the effect of paging [45]. To do so, we run the benchmark for several integer values h_1 and h_2, then the execution cost for all integer values of $1 \leq h_1 \leq U_{s_1}$ and $1 \leq h_2 \leq U_{s_2}$ was predicted using bilinear interpolation methods. The execution time of each tile is measured once and is used several times in practice. So, the cost of calculating the execution time of each tile will be amortized on the total execution time. Since intra-node communication cost is negligible compared to inter-node communication cost, we did not directly consider intra-node communication cost in Eq. 3.4. In fact, intra-node communication cost indirectly have regarded in c_{p_i} parameter.

4.1. Evaluation of the theoretical model. In this section, we evaluate the proposed theoretical model for estimating the parallel execution time and genetic tiling algorithm for finding near-optimal tiling. In experiment 1, we consider a homogeneous computing system consists of eight same processing nodes of type 1 as mentioned in Table 4.2. First, the computational powers of these nodes are normalized such that the summation of them equals one. The size of the iteration space is $U_c \times U_{s_1} \times U_{s_2} = 1024 \times 1024 \times 1024$. The size of the assigned

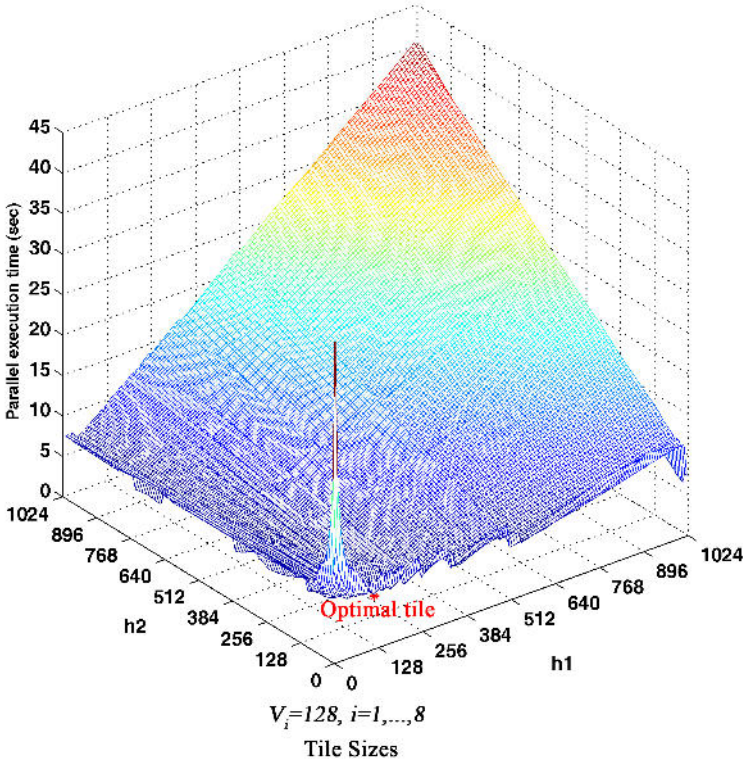

FIG. 4.2. *Parallel execution time for different tile sizes in experiment 1*

chunk to the respective nodes in the u_c dimension is $V_i = U_c \times vp_i = 1024 \times 0.125 = 128$ for $i = 1, \ldots, 8$. Now, we can determine the optimal size of other sides of 3D tile, h_1 and h_2, along the synchronization dimensions u_{s_1} and u_{s_2}. Figure 4.2 shows the parallel execution time for various tile sizes (the V_i, h_1 and h_2). By searching the entire space of solutions of h_1 and h_2, the optimal value of $\langle h_1, h_2 \rangle$ are $\langle 128, 16 \rangle$. As theoretically expected, when the tile sizes fit into the cache of nodes, the cache utilization and data locality maximize and it would lead to improvement in the parallel execution time.

It is to be noted that searching the entire solution space of tile sizes can be very time consuming, especially in the large solution space. So, we use proposed tiling genetic algorithm and the nonlinear integer programming solver, NOMAD, to find the near-optimal value of h_1 and h_2 from the theoretical model, Eq. 3.5. Table 4.3 shows the results of 30 runs of the tiling genetic algorithm and NOMAD to solve Eq. 3.5 in experiment 1. The comparison of the average and standard deviation of $\frac{|AT-OPT|}{OPT}$ indicate that the reliability of the proposed tiling genetic algorithm to find the near-optimal value of h_1 and h_2 is better than NOMAD algorithm. On the other hand, the accuracy of value $\langle h_1, h_2 \rangle$ of the tiling genetic algorithm and NOMAD is achieved by the error less than 0.0055 and 0.0248 in 30 runs, respectively.

4.2. Comparison of 3D and 2D Tiling. As mentioned in the related work, the proposed methods in [21, 33, 40] could find the near-optimal partitioning of 3-nested loop with dependencies for homogeneous/heterogeneous computing systems. It targets two loops of the nested loop and considers the outer loop as synchronization dimension and another loop as scheduling dimension. We refer to this work as the 2D tiling. In the following, we compare the proposed 3D tiling with the 2D tiling for the 3-nested loop with dependencies on homogeneous/heterogeneous computing systems. We find the near-optimal 3D tiling and 2D tiling for one homogeneous and several heterogeneous computing systems. Table 4.4 shows the near-optimal tile sizes of 2D and 3D tiling with/without considering heterogeneity feature and Fig. 4.3 plots their corresponding execution time. The results presented in Tables 4.5 and 4.6 show the speedup of execution time for the 3D tiling versus

TABLE 4.3
The results of 30 runs of the tiling genetic algorithm and NOMAD Algorithm in experiment 1

Run	NOMAD Algorithm					Genetic Tiling Algorithm					
	h_1	h_2	TT	AT	$\frac{\lvert AT-OPT\rvert}{OPT}$	Generation	h_1	h_2	TT	AT	$\frac{\lvert AT-OPT\rvert}{OPT}$
1	128	16	6.0560	6.0643	0.0000	68	128	16	6.0560	6.0643	0.0000
2	227	9	6.3812	6.3904	0.0538	97	128	16	6.0560	6.0643	0.0000
3	136	15	6.2110	6.2199	0.0257	500	128	15	6.2429	6.2518	0.0309
4	128	16	6.0560	6.0643	0.0000	225	128	16	6.0560	6.0643	0.0000
5	256	8	6.2871	6.2954	0.0381	522	128	16	6.0560	6.0643	0.0000
6	127	16	6.1178	6.1271	0.0104	494	128	16	6.0560	6.0643	0.0000
7	255	8	6.3724	6.3828	0.0525	109	128	16	6.0560	6.0643	0.0000
8	135	15	6.2147	6.2237	0.0263	173	128	16	6.0560	6.0643	0.0000
9	128	16	6.0560	6.0643	0.0000	422	128	16	6.0560	6.0643	0.0000
10	256	8	6.2871	6.2954	0.0381	500	127	16	6.1178	6.1271	0.0104
11	127	16	6.1178	6.1271	0.0104	389	128	16	6.0560	6.0643	0.0000
12	128	16	6.0560	6.0643	0.0000	398	128	16	6.0560	6.0643	0.0000
13	64	32	6.3656	6.3739	0.0511	239	128	16	6.0560	6.0643	0.0000
14	146	14	6.3077	6.3172	0.0417	67	128	16	6.0560	6.0643	0.0000
15	128	16	6.0560	6.0643	0.0000	288	128	16	6.0560	6.0643	0.0000
16	128	16	6.0560	6.0643	0.0000	473	128	16	6.0560	6.0643	0.0000
17	156	13	6.3014	6.3104	0.0406	500	128	15	6.2429	6.2518	0.0309
18	227	9	6.3812	6.3904	0.0538	500	128	16	6.0560	6.0643	0.0000
19	136	15	6.2110	6.2199	0.0257	500	129	15	6.2387	6.2476	0.0302
20	128	16	6.0560	6.0643	0.0000	79	128	16	6.0560	6.0643	0.0000
21	136	15	6.2110	6.2199	0.0257	112	128	16	6.0560	6.0643	0.0000
22	128	16	6.0560	6.0643	0.0000	500	128	15	6.2429	6.2518	0.0309
23	128	16	6.0560	6.0643	0.0000	500	129	15	6.2387	6.2476	0.3022
24	128	15	6.2429	6.2518	0.0309	121	128	16	6.0560	6.0643	0.0000
25	227	9	6.3812	6.3904	0.0538	152	128	16	6.0560	6.0643	0.0000
26	128	16	6.0560	6.0643	0.0000	168	128	16	6.0560	6.0643	0.0000
27	63	32	6.4037	6.4125	0.0574	206	128	16	6.0560	6.0643	0.0000
28	119	16	6.2917	6.3011	0.0390	224	128	16	6.0560	6.0643	0.0000
29	136	15	6.2110	6.2199	0.0257	178	128	16	6.0560	6.0643	0.0000
30	145	14	6.3114	6.3210	0.0423	453	128	16	6.0560	6.0643	0.0000
	Average of $\frac{\lvert AT-OPT\rvert}{OPT}$=0.0248 Standard Deviation of $\frac{\lvert AT-OPT\rvert}{OPT}$=0.0212					Average of $\frac{\lvert AT-OPT\rvert}{OPT}$=0.0055 Standard Deviation of $\frac{\lvert AT-OPT\rvert}{OPT}$=0.0114					

Comment:
TT is the Theoretical Time for $\langle h_1, h_2\rangle$.
AT is the Actual Time for $\langle h_1, h_2\rangle$.
OPT is the Optimal Time for $\langle h_1, h_2\rangle$.
Optimal values for $\langle h_1, h_2\rangle$ via searching the entire space of feasible solutions is $\langle 128, 16\rangle$ with the actual time 6.0643

2D tiling with/without considering heterogeneity feature.

In experiment 1, the homogeneous computing system consists of eight similar nodes of type 1. So, the parallel execution time in 2D tiling with and without considering heterogeneity feature is the same and similarly for 3D tiling. In this case, the 3D tiling achieves 1.65× speedup of execution time compared to the 2D tiling.

In experiment 2, the heterogeneous computing system consists of eight processing nodes, four nodes of type 1 and four nodes of type 3, as mentioned in Table 4.2. Since nodes 1 and 3 have the computational power close to each other, the resulting speedup of execution time is almost close to experiment 1.

In experiment 3, the heterogeneous computing system consists of eight processing nodes, four nodes of type 1 and four nodes of type 8, as mentioned in Table 4.2. The nodes 1 and 8 have the computational power very different from each other. In this case, the 3D tiling achieves 1.74× speedup of execution time compared to the

TABLE 4.4
Near-optimal tile sizes of 2D and 3D tiling with/without considering heterogeneity feature

Exp	The sides of the tile	Without considering heterogeneity		With considering heterogeneity	
		2D Tiling	3D Tiling	2D Tiling	3D Tiling
Exp.1	V_i	{ 128,128,128,128,128,128,128,128 }		{ 128,128,128,128,128,128,128,128}	
	h_1	135	128	135	128
	h_2	1024	16	1024	16
Exp.2	V_i	{ 128,128,128,128,128,128,128,128 }		{ 163,163,163,163,93,93,93,93}	
	h_1	135	128	150	74
	h_2	1024	16	1024	32
Exp.3	V_i	{ 128, 128, 128, 128, 128, 128, 128, 128 }		{ 188, 188, 188, 188, 68, 68, 68, 68}	
	h_1	135	128	165	82
	h_2	1024	16	1024	25
Exp.4	V_i	{ 86,86,86,86,85,85,85,85,85,85,85,85 }		{ 122,122,122,122,89,89,89,89,45,45,45,45}	
	h_1	140	192	144	84
	h_2	1024	16	1024	38
Exp.5	V_i	{ 114,114,114,114,114,114,114,113,113 }		{ 161,148,144,142,122,105,94,54,54}	
	h_1	132	32	152	147
	h_2	1024	64	1024	18
Exp.6	V_i	{ 114,114,114,114,114,114,114,113,113 }		{ 152,152,152,152,152,66,66,66,66 }	
	h_1	132	32	156	128
	h_2	1024	64	1024	20
Exp.7	V_i	{ 114,114,114,114,114,114,114,113,113 }		{ 152,152,152,152,101,101,100,57,57 }	
	h_1	132	32	162	80
	h_2	1024	64	1024	32

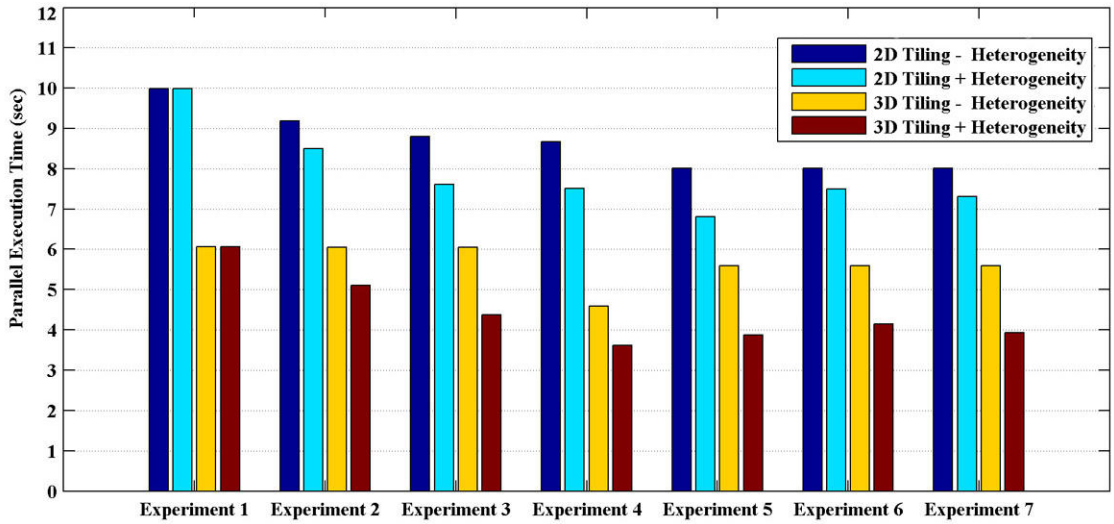

FIG. 4.3. *Comparison of 3D tiling and 2D tiling*

TABLE 4.5
The speedup of execution time of 3D tiling vs 2D tiling without considering heterogeneity feature

	2D Tiling-Heterogeneity						
	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6	Exp.7
3D Tiling-Heterogeneity	1.65	1.52	1.45	1.89	1.43	1.43	1.43

TABLE 4.6
The speedup of execution time of 3D tiling vs 2D tiling with considering heterogeneity feature

	2D Tiling+Heterogeneity						
	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6	Exp.7
3D Tiling+Heterogeneity	1.65	1.67	1.74	2.08	1.76	1.81	1.86

TABLE 4.7
Partitioning nodes in experiment 5 into two or three groups of similar performance

Computational power of nodes	Group 1	Group 2	Group 3
0.1568, 0.1438, 0.1405, 0.1383, 0.1197, 0.1029, 0.0920, 0.0531, 0.0529	0.1568, 0.1438, 0.1405, 0.1383, **0.1197**	0.1029, 0.0920, 0.0531, **0.0529**	-
	0.1568, 0.1438, 0.1405 **0.1383**	0.1197, 0.1029, **0.0920**	0.0531, **0.0529**

2D tiling with considering heterogeneity feature.

In experiment 4, the heterogeneous computing system consists of 12 processing nodes of three types 1, 4 and 7 as mentioned in Table 4.2. In this case, the 3D tiling can achieve 1.89× and 2.08× speedup of execution time compared to the 2D tiling without and with considering heterogeneity feature, respectively.

In experiment 5, the heterogeneous computing system consists of nine nodes of fully different computational powers as mentioned in Table 4.2. In this case, the 3D tiling can achieve 1.43× and 1.76× speedup of execution time compared to the 2D tiling without and with considering heterogeneity feature, respectively.

The heterogeneity is an important feature in parallel and distributed computing systems but considering fully heterogeneity in practice is very difficult. Therefore, we partition nodes of experiment 5 into two or three groups of almost similar performance in terms of their computational power and consider the weakest node in each group as the representative. Table 4.7 show the results of the grouping that was done with *fastclus* procedure on SAS software. The weakest node in each group is bold. The parallel execution times in experiment 6 and 7 are very close to experiment 5.

According to the experimental results, the parallel execution time of the 2D tiling and 3D tiling with considering heterogeneity feature is less than the 2D tiling and 3D tiling without considering heterogeneity feature. Therefore, loop tiling combined with the heterogeneity feature could help to improve the efficiency of computation on heterogeneous systems. Overall, the results show the minimum parallel execution time for the 3D tiling with considering heterogeneity feature in all experiments.

As already mentioned, Fig. 4.3 shows the cost to implement the obtained solution for 2D and 3D tiling. The proposed genetic tiling algorithm takes, on average, less than one second to find a solution. Therefore, the cost to obtain the solution for 3D tiling using the genetic algorithm is higher than 2D tiling, because it involves the cost of the evolutionary process. However, the results presented in Fig. 4.3 shows that the 3D tiling might lead to a more parsimonious solution in terms of implementation cost.

5. Conclusions and future work. This paper addresses the problem of 3D tiling and scheduling when parallelizing three-level perfectly nested loop with dependencies on heterogeneous systems. The tile size plays an important role to improve the parallel execution time of nested loops. Searching the entire feasible solution space of tile size can be very time consuming, especially in cases where the solution space is large. We build a theoretical model to estimate the parallel execution time with the computational power awareness of the nodes of computing systems. We use the proposed tiling genetic algorithm and nonlinear integer programming solvers, NOMAD, to find the near-optimal value of tile size from the theoretical model. Experiment results by 3D heat equation on heterogeneous systems show the accuracy and efficiency of the proposed theoretical model and the tiling genetic algorithm in estimating the parallel execution time and finding the near-optimal 3D tiling. Furthermore, we show that the 3D tiling combined with heterogeneity feature and a pipeline-like execution could exploit the potential parallelism and improve the parallel execution time of perfectly nested loop with dependencies on heterogeneous systems.

The plans for future work include: (i) extend the 3D tiling algorithm for the imperfectly nested loops with

dependencies on heterogeneous computing systems; and (ii) extend the 3D tiling algorithm to handle partially connected network.

Acknowledgments. The authors would like to thanks the editor and the reviewers for their helpful and constructive suggestions, which considerably improved the quality of the paper. They would also like to thanks Nasrin Nasrabadi and Fateme Karimi, PhD Students, for all very valuable comments.

REFERENCES

[1] S. Fide and S. Jenks, *A middleware approach for pipelining communications in clusters*, Cluster Computing, 10 (2007), pp. 409-424.
[2] I. Riakiotakis and P. Tsanakas, *Dynamic scheduling of nested loops with uniform dependencies in heterogeneous networks of workstations*, 8th International Symposium on Parallel Architectures, Algorithms and Network, ISPAN 2005, 2005.
[3] R. L. Cariño and I. Banicescu, *A load balancing tool for distributed parallel loops*, Cluster Computing, 8 (2005), pp. 313-321.
[4] X. Zhou, M. J. Garzarán, and D. A. Padua , *Optimal parallelogram selection for hierarchical tiling*, ACM Transactions on Architecture and Code Optimization, 11 (2015), pp. 1-23.
[5] M. I. Daoud and N. Kharma, *An efficient genetic algorithm for task scheduling in heterogeneous distributed computing systems*, IEEE Congress on Evolutionary Computation, CEC, pp. 3258-3265, 2006.
[6] G. Wang, Y. Wang, H. Liu, and H. Guo, *HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing*, Scientific Programming, 2016 (2016), pp. 1-11.
[7] K. Qinma and H. He, *Honeybee mating optimization algorithm for task assignment in heterogeneous computing systems*, Intelligent Automation & Soft Computing, 19 (2013), pp. 69-84.
[8] C.-T. Yang and L.-H. Cheng, *Implementation of a performance-based loop scheduling on heterogeneous clusters*, Algorithms and Architectures for Parallel Processing, Springer, pp. 44-54, 2009.
[9] J. Dongarra and A. L. Lastovetsky, *High performance heterogeneous computing*, John Wiley & Sons, 2009.
[10] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of thing*, Morgan Kaufmann, 2013.
[11] R. Bleuse, S. KedadSidhoum, F. Monna, G. Mounié, and D. Trystram, *Scheduling independent tasks on multicores with GPU accelerators*, Concurrency and Computation: Practice and Experience, 27 (2015), pp. 1625-1638.
[12] M. G. Lopez, J. Young, J. S. Meredith, P. C. Roth, M. Horton, and J. S. Vetter, *Examining recent many-core architectures and programming models using SHOC*, Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems, 2015.
[13] P. A. La Fratta and P. M. Kogge, *Heterogeneity in parallel and distributed computing*, Journal of Parallel and Distributed Computing, 73 (2013), pp. 1523-1524.
[14] M. E. Wolf and M. S. Lam, *A data locality optimizing algorithm*, ACM Sigplan Notices, pp. 30-44, 1991.
[15] G. Rivera and C.-W. Tseng, *Tiling optimizations for 3D scientific computations*, ACM/IEEE Conference in Supercomputing, pp. 32-32, 2000.
[16] M. E. Wolf and M. S. Lam, *A loop transformation theory and an algorithm to maximize parallelism*, IEEE Transactions on Parallel and Distributed Systems, 2(1991), pp. 452-471.
[17] D. Padua, *Encyclopedia of parallel computing* , Springer Science & Business Media, 2011.
[18] M. Kowarschik and C. Weib, *An overview of cache optimization techniques and cache-aware numerical algorithms*, Algorithms for Memory Hierarchies, LNCS 2625, Springer, pp. 213-232, 2003.
[19] S. Parsa and M. Hamzei, *Locality-Conscious Nested-Loops Parallelization*, ETRI Journal, 36 (2014), pp. 124-133.
[20] I. Riakiotakis, F. M. Ciorba, T. Andronikos, and G. Papakonstantinou, *Distributed dynamic load balancing for pipelined computations on heterogeneous systems*, Parallel Computing, 37 (2011), pp. 713-729.
[21] T. Andronikos, F. M. Ciorba, I. Riakiotakis, G. Papakonstantinou, and A. T. Chronopoulos, *Studying the impact of synchronization frequency on scheduling tasks with dependencies in heterogeneous systems*, Performance Evaluation, 67 (2010), pp. 1324-1339.
[22] U. Bondhugula, *Compiling affine loop nests for distributed-memory parallel architectures*, International Conference in High Performance Computing, Networking, Storage and Analysis (SC), pp. 1-12, 2013.
[23] H. El-Rewini and M. Abd-El-Barr, *Advanced computer architecture and parallel processing*, John Wiley & Sons, 2005.
[24] C. L. Abad, Y. Lu, and R. H. Campbell, *DARE: Adaptive data replication for efficient cluster scheduling*, International Conference on Cluster Computing (CLUSTER), IEEE, pp. 159-168, 2011.
[25] S. Parsa and S. Lotfi, *A new genetic algorithm for loop tiling*, The Journal of Supercomputing, 37 (2006), pp. 249-269.
[26] S. Mehta, G. Beeraka, and P.-C. Yew, *Tile size selection revisited*, ACM Transactions on Architecture and Code Optimization, 10 (2013).
[27] S. Chen and J. Xue, *Partitioning and scheduling loops on NOWs*, Computer Communications, 22 (1999), pp. 1017-1033.
[28] F. M. Ciorba, I. Riakiotakis, G. K. Papakonstantinou, T. Andronikos, and A. T. Chronopoulos, *Studying the impact of synchronization frequency on scheduling tasks with dependencies in heterogeneous systems*, PACT, 2007.
[29] J. Ramanujam and P. Sadayappan, *Nested loop tiling for distributed memory machines*, Proceedings of the Fifth Conference in Distributed Memory Computing, pp. 1088-1096, 1990.
[30] D. Liu, Y. Wang, Z. Shao, M. Guo, and J. Xue, *Optimally maximizing iteration-level loop parallelism*, IEEE Transactions on Parallel and Distributed Systems, 23(2012), pp. 564-572.

[31] F. M. Ciorba, I. Riakiotakis, T. Andronikos, G. Papakonstantinou, and A. T. Chronopoulos, *Enhancing self-scheduling algorithms via synchronization and weighting*, Journal of Parallel and Distributed Computing, 68 (2008), pp. 246-264.
[32] J. Xue, *Communication-minimal tiling of uniform dependence loops*, Journal of Parallel and Distributed Computing, 42 (1997), pp. 42-59.
[33] I. Riakiotakis, F. M. Ciorba, T. Andronikos, G. Papakonstantinou, and A. T. Chronopoulos, *Towards the optimal synchronization granularity for dynamic scheduling of pipelined computations on heterogeneous computing systems*, Concurrency and Computation: Practice and Experience, 24 (2012), pp. 2302-2327.
[34] P. Crandall , M. J. Quinn, *Three-Dimensional Grid Partitioning for Network Parallel Processing*, ACM Conference on Computer Science. Citeseer, pp. 210-217, 1994.
[35] O. Beaumont, V. Boudet, F. Rastello and Y. Robert, *Matrix multiplication on heterogeneous platforms*, IEEE Transactions on Parallel and Distributed Systems, 12 (2001), 1033-1051.
[36] E. Z. Zefreh, S. Lotfi, L. M. Khanli, and J. Karimpour, *3D data partitioning for three-level perfectly nested loops on heterogeneous distributed systems*, Concurrency and Computation: Practice and Experience, accepted, 2016.
[37] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, *Algorithmic issues on heterogeneous computing platforms*, Parallel processing letters, 9 (1999), pp. 197-213.
[38] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien, *Static tiling for heterogeneous computing platforms*, Parallel Computing, 25 (1999), pp. 547-568.
[39] F. M. Ciorba, T. Andronikos, I. Riakiotakis, A. T. Chronopoulos, and G. Papakonstantinou, *Dynamic multi phase scheduling for heterogeneous clusters*, 20th International in Parallel and Distributed Processing Symposium, IPDPS, 2006.
[40] F. M. Ciorba, I. Riakiotakis, T. Andronikos, A. T. Chronopoulos, and G. Papakonstantinou, *Optimal synchronization frequency for dynamic pipelined computations on heterogeneous systems*, International Conference on Cluster Computing, IEEE, pp. 410-415, 2007.
[41] F. Desprez, J. Dongarra, F. Rastello, and Y. Robert, *Determining the idle time of a tiling: new results*, International Conference on Parallel Architectures and Compilation Techniques, pp. 307-317, 1997.
[42] S. Le Digabel, *NOMAD: Nonlinear optimization with the MADS algorithm*, ACM Transactions on Mathematical Software (TOMS), 37 (2011).
[43] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*, John Wiley & Sons, 2000.
[44] G. Hager and G. Wellein, *Introduction to high performance computing for scientists and engineers*, CRC Press, 2010.
[45] A. Lastovetsky and R. Reddy, *Data partitioning with a functional performance model of heterogeneous processors*, International Journal of High Performance Computing Applications, 21 (2007), pp. 76-90.

Edited by: Dana Petcu
Received: May 28, 2016
Accepted: August 2, 2016