Available online at www.sciencedirect.com

SCIENCE DIRECT®

J. Parallel Distrib. Comput. 66 (2006) 32 – 46

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Task assignment in heterogeneous computing systems☆

Bora Ucar[a], Cevdet Aykanat[a,*], Kamer Kaya[a], Murat Ikinci[b]

[a]*Department of Computer Engineering, Bilkent University, 06800, Ankara, Turkey*
[b]*STM Inc., Mecnun Sokak No 58, Beştepe, 06510, Ankara, Turkey*

Received 25 January 2005; received in revised form 14 June 2005; accepted 16 June 2005
Available online 11 August 2005

Abstract

The problem of task assignment in heterogeneous computing systems has been studied for many years with many variations. We consider the version in which communicating tasks are to be assigned to heterogeneous processors with identical communication links to minimize the sum of the total execution and communication costs. Our contributions are three fold: a task clustering method which takes the execution times of the tasks into account; two metrics to determine the order in which tasks are assigned to the processors; a refinement heuristic which improves a given assignment. We use these three methods to obtain a family of task assignment algorithms including multilevel ones that apply clustering and refinement heuristics repeatedly. We have implemented eight existing algorithms to test the proposed methods. Our refinement algorithm improves the solutions of the existing algorithms by up to 15% and the proposed algorithms obtain better solutions than these refined solutions.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Task assignment; Heterogeneous computing systems; Task interaction graph

1. Introduction

The problem of task assignment in heterogeneous systems deals with finding proper assignment of tasks to processors in order to optimize some performance metric such as the system utilization and the turnaround time. There exists a large body of literature covering many task and parallel computer models. In this paper, we consider the task assignment problem with the following characteristics. The tasks are modeled using a task interaction graph (TIG). In the TIG model, the vertices of the graph correspond to the tasks and the edges correspond to the intertask communications. There is no precedence relation among tasks. The processors are heterogeneous, i.e., the execution cost of a task depends on the processor on which it is executed. The network is homogeneous, i.e., the communication between two tasks depends only on whether or not they are assigned to the same processor. The objective is to minimize the sum of the total execution and communication costs in order to optimize system utilization.

The problem is formally defined as follows. Let P be the set of n processors in the heterogeneous computing system, T be the set of m tasks to be assigned to the processors, $ETC = \{x_{ip}\}_{m\times n}$ be the expected time to compute matrix where x_{ip} denotes the execution cost of task i on processor p, and $G = (T, E)$ be the TIG, where E is the set of edges representing the communication between tasks. Each edge $(i, j) \in E$ is associated with a communication cost c_{ij}, which incurs only when tasks i and j are assigned to different processors. The processors are heterogeneous in the sense that there is no special structure in the ETC matrix. In other words, processor p being faster than processor q on task i, e.g., $x_{ip} \leqslant x_{iq}$, does not imply anything about their speeds for another task. In general, cost models are composed from constituent cost components that reflect the application activities. In these

☆ This work is partially supported by the Scientific and Technical Research Council of Turkey under grant 103E028 and by the European Commission FP6 project SEEGRID with contract no 002356.

* Corresponding author. Fax: +90 312 266 4027.
E-mail addresses: ubora@cs.bilkent.edu.tr (B. Ucar), aykanat@cs.bilkent.edu.tr (C. Aykanat), kamer@cs.bilkent.edu.tr (K. Kaya), ikinci@stm.com.tr (M. Ikinci).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.06.014

compositional models, cost components such as local disk I/O costs are modeled separately [3]. In this work, we consider only the execution and communication costs.

Given the above definitions, the objective is to find an assignment $A : T \rightarrow P$ that minimizes the sum of execution and communication costs:

$$\text{Minimize} \quad \left(\sum_{i=1}^{m} \sum_{p=1}^{n} a_{ip} x_{ip} + \sum_{(i,j) \in E} \sum_{p=1}^{n} a_{ip}(1 - a_{jp}) c_{ij} \right)$$

$$\text{subject to} \quad \sum_{p=1}^{n} a_{ip} = 1, \quad i \in T$$

$$a_{ip} \in \{0, 1\}, \quad p \in P, \; i \in T.$$

Here, if task i is assigned to processor p, then $a_{ip} = 1$ and 0 otherwise. The constraint $\sum_{p=1}^{n} a_{ip} = 1$ ensures that the task i is assigned to only one processor. Although the problem is *NP-complete* [6], some special instances are polynomial time solvable: two-processor systems in the time complexity of a maximum flow algorithm [46], tree TIGs on heterogeneous networks in $O(mn^2)$ time [6], tree TIGs on homogeneous networks in $O(mn)$ time [4], series parallel TIGs in $O(mn^3)$ time [29,49,50], k-ary tree TIGs in $O(mn^{k+1})$ time [17].

1.1. Background

The problem defined above was first introduced by Stone [46]. Stone's original work lays down the TIG model to represent sequentially executing tasks. In other words, at any time exactly one task is being executed on one of the processors. The edges represent two-way interactions between two persistent tasks, e.g., a task passes control to another one and waits the control to be returned back again [40]. Some later works interpreted the TIG model in such a way that all tasks are simultaneously executable and communications take place either at any time or intermittently throughout the program execution (see for example [27,43,47]). These later interpretations consider the minimization of the turnaround time, e.g., minimizing the maximum load in terms of execution and communication costs per processor. We work under the original interpretation and address the minimization of the sum of the total execution and communication costs. This interpretation has been used to develop grid scheduling models [3] such as for mapping parallel pipelines [51] and phased message-passing programs [20]. CPU and communication intensive tasks when mapped to a set of computers in a common LAN are most likely to be charged in terms of the total CPU cycles they consume and the total network activity they generate under various economy models for the Grid [10]. Therefore, we believe that minimization of the sum of the total execution and communication costs will be the objective in scheduling grid applications.

There are numerous studies addressing the task assignment problem under various characterizations. A comprehensive survey discussing the models before early 1990s can be found in [40]. The books [44] and [16] cover many aspects of the task scheduling problem. Among some recent surveys covering certain variations are [32] which addresses directed task graphs, [8,45] which address independent tasks, and [21–23] which address file sharing otherwise independent tasks. For some later works on mapping TIGs to processors in order to minimize turnaround time see: [27] for exact algorithms under processor heterogeneity and network homogeneity; [48] for exact algorithms under processor and network heterogeneity; [35] for exact algorithms under processor homogeneity and network heterogeneity; [47] for heuristics under processor and network heterogeneity where each processor and communication link have computation and communication capacity, respectively; [43] for heuristics under processor homogeneity and network heterogeneity; [26] for heuristics under processor and network homogeneity. See [33] for exact algorithms that map TIGs to processors in the array networks for minimizing the sum of total execution and communication costs. The variant in [33] assumes that some processors have unique resources and hence there are restrictions in the task assignments. See [41] for heuristics that map TIGs to processors in order to minimize total communication time in a heterogeneous network.

Apart from the differences in the objectives, computing system characteristics, and computation models, the task assignment algorithms differ in the solution methods. The papers [14,27,42] categorize the solution methods into graph-theoretic, mathematical programming, state-space search, probabilistic and randomized optimization methods. The papers cited above include numerous references for these approaches. Therefore, we refer the reader to these papers for references regarding a particular method.

We have implemented eight algorithms from the literature given in Table 1 in order to build a sound experimental framework. These eight algorithms are quite different in nature. The first four are state-of-the-art meta-heuristics and hence fall into the category of randomized optimization. The next two are based on graph-theoretic concepts. Specifically, the KLZ algorithm uses matching based and agglomerative clustering techniques to reduce the problem size. The VML algorithm uses network-flows techniques to obtain a partial task assignment and then uses greedy heuristics to complete the assignment. The algorithm TOpt obtains optimal solutions for the problem instances whose TIGs are in tree structure. Specifically, this algorithm solves the recursion $A(i, p) = \sum_{j \in \text{child}(i)} \min_k\{A(j, k) + c'_{ij}(p, k)\} + x_{ip}$ from leaves to the root of the tree using a dynamic programming approach and returns $\min_k\{A(r, k)\}$. Here, r is the root of the tree and $A(i, p)$ is the optimal solution for the subtree whose root is i under the condition that the task associated with node i is assigned to processor p, and $c'_{ij}(p, k) = c_{ij}$ if $p \neq k$ and 0 otherwise. The algorithm A* is an informed-search algorithm which finds optimal solutions for very small instances of the task assignment problem.

Table 1
Existing task assignment algorithms implemented in this work

Algorithm	Reference	Approach
GA	[1]	Genetic algorithm
SA	[24]	Simulated annealing
TSN	[13]	Tabu search and noising
PSO	[42]	Particle swarm optimization
KLZ	[31]	Graph theoretic (clustering)
VML	[34]	Graph theoretic (network flows)
TOpt	[6]	Graph theoretic (dynamic programming on trees)
A*	[27,48]	State-space search algorithm (based on A*)

1.2. Contributions

Among the previous works that address Stone's original problem, those that use task clustering (for example [7,15,31,34,36,52]) are of particular interest to us because we are improving upon these works. These works use clustering approaches in which highly interacting tasks are merged to reduce the original problem into a smaller one. Some of these works [31,36] consider processors for clustering by augmenting processor vertices and processor-to-task edges and obtain task-to-processor assignments during the clustering process. This type of algorithms are called single-phase heuristics. Some other works [7,15,34,52] obtain task-to-processor assignments in an assignment phase separated from the clustering phase and hence are called two-phase heuristics.

In previous clustering approaches, the decision on clustering two tasks depends solely on the communication cost between them. However, these two tasks may be dissimilar in the sense that their total execution cost may be inferior when assigned to the favorite processor of either one, where the favorite processor of a task is the processor that has the minimum execution time for that task. Motivated by this observation, we propose a clustering heuristic which considers the communication costs between two tasks as well as their dissimilarity in §2. In general, the order in which tasks are assigned to processors affects the assignment quality. We propose two metrics in §3 to determine a favorable order in two-phase approaches. Furthermore, we develop an iterative-improvement-based heuristic to refine task assignments in §4.

We build a family of assignment heuristics by using the proposed clustering metric, assignment ordering, and refinement heuristic. In §5.1, we propose a method that starts like a two-phase heuristic and then later behaves like a single-phase heuristic. Then, we adopt the multilevel framework, which has proven to be successful in graph and hypergraph partitioning, in two different settings: multilevel task clustering and multilevel task assignment. The multilevel clustering setting presented in §5.2 reduces the given task assignment problem by forming task clusters. This method is better suited to the two-phase assignment heuristics as the clustering and assignment phases are separated. The multilevel assignment setting presented in §5.3 reduces the task assignment problem by assigning disjoint task sets to processors at each level. This method is better suited to the single-phase assignment heuristics.

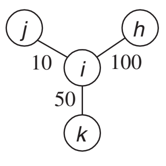
The proposed assignment algorithms are static in the sense that the assignment of tasks to processors is done before the program execution begins. Large and nondedicated computing platforms may require dynamic task assignment methods to adapt to the run-time changes such as increases in the workload, processor failures, and link failures. The proposed refinement heuristics seem to be viable to adapt the original assignments to the run-time changes. However, dynamic task assignment methods interact with other system components such as process migration mechanism whose costs should be considered in the refinement heuristics. In this paper, we do not dwell into these issues. See references [5,19,37] for dynamic task assignment and fault tolerance management.

2. A novel clustering approach

2.1. Motivation

Most of the task assignment algorithms that use clustering reduce the intercluster communication costs first, and then find a solution by assigning the task clusters to their favorite processors. Since they do not consider the difference between the execution times of tasks on the same processors, they may form clusters of tasks that are not similar to each other. For the sample TIG given in Fig. 1, traditional clustering algorithms tend to merge tasks i and h, since (i, h) is the edge with the maximum weight. The validity of this decision is investigated in the rightmost table of Fig. 1. Although clustering i and h saves 100 units of communication cost, the cluster has at least 400 units of execution cost. The other alternatives lead to smaller savings in communication costs, but also lead to smaller execution costs. Therefore, it seems that clustering tasks i and h is not preferable. This deficiency cannot be avoided without taking the execution times of tasks into consideration.

In a clustering approach, the communication cost between a task i and a cluster is equal to the sum of communication costs between task i and all tasks in that cluster. In most of the clustering approaches, clusters are formed iteratively (i.e., new clusters are formed one at a time) based on the



Task	Execution costs		
	P_1	P_2	P_3
i	2	200	400
j	1	100	200
k	200	2	400
h	400	200	2

Mate	Save comm.	Min exec.
h	100	400
j	10	3
k	50	202

Fig. 1. Task i is to be clustered.

communication costs between tasks and clusters. This approach corresponds to *agglomerative clustering* in clustering classification. In these approaches, the edges that are incident on the clusters usually have large communication costs and hence iterative clustering algorithms will most likely contract such edges incident on the currently formed cluster. This problem is known as the *polarization problem*. Kopidakis et al. [31] proposed two solutions for this problem. The first solution is to use hierarchical clustering approaches such as matching-based algorithms instead of the iterative ones. In hierarchical clustering algorithms, several new clusters may be formed simultaneously. This approach solves the polarization problem, but the experimental results given in [31] show that it generally leads to inferior assignment quality. The other solution presented by Kopidakis et al. is to set the communication cost between a task i and a cluster to the maximum of the communication costs between the task i and the tasks in that cluster. Choosing the maximum communication cost prevents polarization towards the growing cluster. However, this scheme causes unfairness and usually does not yield good clusters in terms of intercluster communication costs.

According to the first observation, a clustering scheme which considers the similarities of tasks while looking at the communication costs is expected to obtain better clusters than the traditional clustering approaches. The second observation displays the need for a clustering scheme that avoids polarization during agglomerative clustering. These observations are the key points for the motivation of the proposed clustering approach.

2.2. *Clustering metric*

Most of the previous clustering approaches, such as [7,15], are used in a two-phase setting. Clustering phase, as the first phase of those algorithms, has more flexibility than the assignment phase. Therefore, success of the overall assignment algorithm depends heavily on the success of the clustering phase. Main decisions about the solution are given in the clustering phase and assignment phase usually completes the solution by using a straightforward heuristic; such as assigning all the clusters to their favorite processors as in Lo's greedy phase [34]. An issue with the clustering approach is that an optimal solution to the reduced problem is not always an optimal solution to the original problem. This is because of the shortsighted decisions made in the clustering phase of the algorithms. Such algorithms try to maximize the total intertask communication costs within the clusters so as to minimize the total communication costs between the clusters. However, this approach may not give good clusters, especially when the processors are heterogeneous. We propose a new clustering approach which considers the differences between execution costs of tasks on the same processors.

Let i and j be two communicating tasks in G. If these tasks are assigned to different processors, then their contribution to the total cost will be at least

$$c_{ij} + \min_{p\in P}\{x_{ip}\} + \min_{p\in P}\{x_{jp}\},$$

where the last two terms are the minimum execution costs of tasks i and j. If tasks i and j are assigned to the same processor, then their contribution to the total cost will be at least

$$\min_{p\in P}\{x_{ip} + x_{jp}\}.$$

With an optimistic view, we derive an equation for the profit α_{ij} of clustering tasks i and j by subtracting the above two costs

$$\begin{aligned}\alpha_{ij} = c_{ij} + \min_{p\in P}\{x_{ip}\} + \min_{p\in P}\{x_{jp}\} \\ - \min_{p\in P}\{x_{ip} + x_{jp}\}.\end{aligned} \tag{1}$$

Eq. (1) can be rewritten as

$$\alpha_{ij} = c_{ij} - d_{ij}, \tag{2}$$

where d_{ij} effectively represents the dissimilarity between tasks i and j in terms of their execution costs. That is,

$$d_{ij} = \min_{p\in P}\{x_{ip} + x_{jp}\} - \left(\min_{p\in P}\{x_{ip}\} + \min_{p\in P}\{x_{jp}\}\right).$$

Note that since $\min_{p\in P}\{x_{ip} + x_{jp}\} \geqslant \min_{p\in P}\{x_{ip}\} + \min_{p\in P}\{x_{jp}\}$ for all i, j, p, we have $d_{ij} \geqslant 0$. Dissimilarity metric achieves its minimum value of $d_{ij} = 0$ when the tasks i and j have the same favorite processor. As seen in Eq. (2), the clustering profit decreases with the increasing dissimilarity between the respective pair of tasks. Hence, unlike the traditional clustering approaches, our clustering profit does not depend only on the intertask communication costs but also depends on the similarities of the tasks to be clustered.

The proposed profit metric for clustering two tasks can be extended to a set S of tasks by preserving the general principles. The profit of clustering the tasks in S can be computed as

$$\begin{aligned}&\alpha_S = c_S - d_S, \quad \text{where} \\ &c_S = \frac{1}{2}\sum_{i\in S}\sum_{j\in S} c_{ij} \quad \text{and} \\ &d_S = \min_{p\in P}\left\{\sum_{i\in S} x_{ip}\right\} - \sum_{i\in S}\min_{p\in P}\{x_{ip}\}.\end{aligned}$$

Here, c_S represents the savings in communication cost due to the internal edges of S, and d_S represents the dissimilarity of the tasks that constitute S.

The proposed metric inherently solves the polarization problem because it considers the difference between the execution times of the tasks being clustered. As in most

```
MERGE-CLUSTERS $(G, Q, x, i, j)$
  DELETE$(Q, j)$
  merge tasks $i$ and $j$ into a new supertask $k$
  construct $Adj[k]$ by performing weighted union of $Adj[i]$ and $Adj[j]$
  update $Adj[h]$ accordingly for each task $h \in Adj[k]$
  for each processor $p \in P$ do
    $x_{kp} \leftarrow x_{ip} + x_{jp}$
  for each $h \in Adj[k]$ do
    compute clustering profit $\alpha_{hk} = \alpha_{kh}$
    if $key[h] = \alpha_{hk}$ then
      INCREASE-KEY $(Q, h, \alpha_{hk})$ with $mate[h] = k$
    elseif $mate[h] = i$ or $mate[h] = j$ then
      recompute the best mate $\ell \in Adj[h]$ of task $h$
      DECREASE-KEY $(Q, h, \alpha_{h\ell})$
  choose the best mate $\ell \in Adj[k]$ for task $k$
  INSERT $(Q, k, \alpha_{k\ell})$ with $mate[k] = \ell$
```

Fig. 2. Clustering task clusters i and j in the agglomerative clustering algorithm.

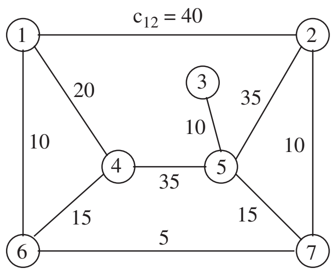
of the clustering algorithms, the communication cost between a task and a cluster is likely to be larger than the communication costs between pairs of single tasks in our clustering scheme. But the dissimilarity between the execution times of a task and a cluster is also likely to be larger than that of a pair of single tasks. Therefore, our clustering metric does not degenerate when the clusters get bigger.

2.3. An agglomerative clustering algorithm

We develop an agglomerative clustering algorithm which uses the proposed clustering metric. Initially, each task is considered to be a singleton cluster. At each step, a pair of task clusters with the maximum clustering profit are merged until the maximum profit becomes negative. We use a priority queue Q implemented as a max-heap to select the pair of tasks at each step.

When two clusters i and j are merged into a new cluster k, the edge between i and j is contracted and the adjacency list of k is set to the weighted union of the remaining edges of i and j. Creating the new cluster k requires computing the execution costs of k as $x_{kp} = \sum_{i \in k} x_{ip}$. After forming the adjacency list of k, clustering profits of the tasks that become adjacent to k are computed. If the clustering profit of such a task h with k is greater than the old key value of h, then k will be the best mate of h with a key value of α_{hk}. Otherwise, the algorithm recomputes the best clustering profit of h only if the old best mate of h is either i or j. In this case, the key value of h has to be decreased. These steps are shown in the algorithm given in Fig. 2.

Fig. 4 presents the steps of our clustering algorithm for the sample problem given in Fig. 3. The execution costs of the new clusters are also presented in Fig. 3. The clustering



Tasks	x_{i1}	x_{i2}	x_{i3}
1	65	30	**15**
2	50	**45**	100
3	100	**5**	100
4	85	45	**10**
5	**10**	95	100
6	85	**30**	95
7	35	**25**	90
{2,5}	**60**	140	200
{1,4}	150	75	**25**
{2,5,7}	**95**	165	290

Fig. 3. TIG and execution times for a sample task assignment problem.

algorithm forms two clusters; first one is formed by merging tasks 1 and 4, and second one is formed by merging tasks 2, 5, and 7. By doing so, two decisions are made in the clustering phase: tasks 1 and 4 should be assigned to the same processor; tasks 2, 5, and 7 should be assigned to the same processor. With these decisions, the original problem is reduced to a smaller one. An optimal solution, which has a cost of 270 units, is found for the problem in Fig. 3 with an exhaustive enumeration. Assigning the resulting clusters {1, 4}, {2, 5, 7}, {3}, and {6} to their favorite processors P_3, P_1, P_2, and P_2, respectively, achieves the optimal solution. This achievement shows that our clustering algorithm produces perfect clusters for the sample problem. Lo's algorithm [34] obtains a solution whose cost is 275 units while the algorithm proposed by Kopidakis et al. [31] obtains a solution whose cost is 285 units.

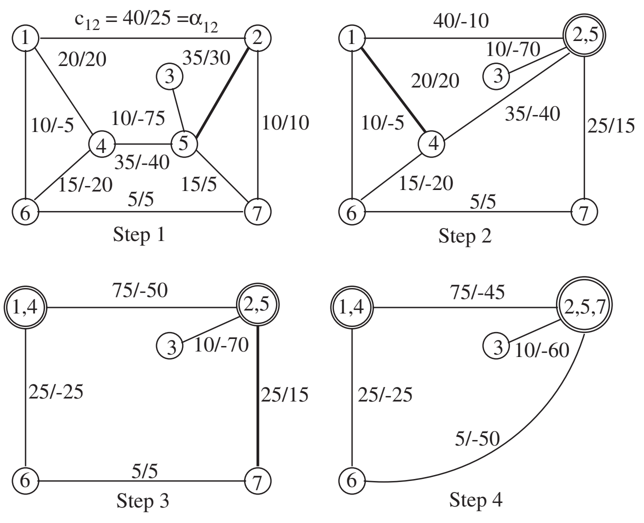


Fig. 4. Clustering steps for the sample TIG given in Fig. 3.

3. Algorithms for determining the assignment order

Numerous research works on iterative assignment algorithms show that the quality of the solution depends on the order in which the tasks are assigned to processors. There are a lot of assignment heuristics that try to find a good order for assigning tasks. See for example [52] which sorts the tasks according to their sum of communication costs and then assigns the tasks in that order to their favorite processors. Here, we propose two new heuristics to determine the assignment order. In both of the heuristics, each task cluster selected for assignment will be assigned to its favorite processor.

3.1. Assignment order according to clustering loss

In the previous section, we presented a profit metric α_S for clustering a set S of tasks into a new cluster. If $\alpha_S > 0$, then clustering the tasks in S may be a good decision. If $\alpha_S \leqslant 0$ for all S containing a task i, then forming a cluster including task i is meaningless; it is better to assign task i to its favorite processor. But if there are two or more tasks that have negative clustering profits for all their clustering alternatives, then the order in which clusters are assigned may affect the solution quality. Our ordering scheme depends on the expectation that assigning the task with the smallest clustering profit first gives better solutions. This is reasonable, because the task with the smallest clustering profits is the most independent task in general. Therefore, in case of an imperfect assignment, other tasks will not be affected very much.

3.2. Assignment order according to grab affinity

Lo [34] used the word "grab" to identify the first phase of her algorithm. In this phase, the algorithm tries to find a prefix to all optimal solutions by using a maximum flow algorithm on a commodity flow network. In each iteration of the grab phase, a number of tasks may be grabbed by an individual processor, and these tasks are then assigned to the respective processor. Assume that only task i is grabbed by a processor p in a step of the grab phase. Then, the inequality

$$\frac{X_i}{n-1} - x_{ip} \geqslant \sum_{(i,j)\in E} c_{ij} + x_{ip}$$

must hold for the task i, where $X_i = \sum_{p\in P} x_{ip}$.

By reorganizing the above inequality, we obtain the residual

$$r_i = \frac{X_i}{n-1} - 2\min_{p}\{x_{ip}\} - \sum_{(i,j)\in E} c_{ij},$$

for each task i. If $r_i > 0$, then task i should be assigned to its favorite processor in any optimal assignment. For $r_i \leqslant 0$, a greater r_i means that task i is more likely to be assigned to its favorite processor in an optimal solution. Due to this observation, selecting the task i with the greatest r_i for assigning first is more likely to give better solutions. We use this criterion to determine the order in which task clusters are assigned to processors.

After assigning task i to a processor, assigning an adjacent task j of i to the same processor will save the communication cost c_{ij}. Therefore, r_j should be updated according to this saving. We use the method proposed by Lo [34] to update r_j by modifying the execution cost of j on processor $q \neq p$ as

$$x^*_{jq} = x_{jq} + c_{ij}, \tag{3}$$

where the execution cost of j on processor p is kept intact.

4. A heuristic for refining task assignments

Kernighan and Lin (KL) [30] propose a fast refinement heuristic which is used in the refinement phase of the graph and hypergraph partitioning tools. KL algorithm, starting from an initial partition, performs a number of passes until it finds a locally optimum partition. Each pass consists of a sequence of vertex swaps. Fiduccia and Mattheyses (FM) [18] introduce a faster implementation of KL algorithm by using vertex movements instead of vertex swaps. Here, we propose an FM-based refinement heuristic for task assignments. The notion of movement in our approach is the task reassignment.

Let task i be assigned to processor p. The *reassignment gain* of task i from processor p to processor q is the decrease in the cost if task i is assigned to processor q instead of processor p. In other words, the reassignment gain for task i from processor p to processor q is

$$g_i(p\to q) = \left(x_{ip} + \sum_{j\in Adj[i], a[j]=q} c_{ij}\right) - \left(x_{iq} + \sum_{j\in Adj[i], a[j]=p} c_{ij}\right),$$

```
SLA $(G, x)$
  $Q \leftarrow \emptyset$
  for each task $i \in T$ do
    compute clustering profit $\alpha_{ik}$ for each task $k \in Adj[i]$ according to Eq. 1
    choose the best mate $j \in Adj[i]$ of task $i$ with $\alpha_{ij} = \max_{k \in Adj[i]}\{\alpha_{ik}\}$
    INSERT $(Q, i, \alpha_{ij})$
    $mate[i] \leftarrow j$
  while $Q \in \emptyset$ do
    $i \leftarrow MAX(Q)$
    if $key[i] > 0$ then
        $i \leftarrow$ EXTRACT-MAX $(Q)$
        MERGE-CLUSTERS $(G, Q, x, i, mate[i])$
    else
        select the task $i$ with maximum assignment affinity
        ASSIGN $(G, Q, x, i)$
```

Fig. 5. SLA task assignment algorithm.

where $a[j]$ denotes the current processor assignment for task j.

The proposed algorithm begins with calculating the maximum reassignment gain for each task i according to the current assignment. This initial gain computation step runs in $O(mn + |E|)$ time. The tasks are inserted into a priority queue according to their maximum reassignment gains. Initially, all tasks are unlocked, i.e., they are free to be reassigned. The algorithm selects an unlocked task with the largest reassignment gain from the priority queue and assigns it to the processor that gives the maximum reassignment gain. After reassigning task i from processor p to q, the algorithm locks i and updates the reassignment gains of each task $j \in adj[i]$ to processors p and q as

$$g_j(a[j] \to p) = g_j, (a[j] \to p) - c_{ij} \quad \text{and}$$
$$g_j(a[j] \to q) = g_j, (a[j] \to q) + c_{ij}.$$

For each $j \in adj[i]$, if $a[j] \notin \{p, q\}$, then both of these updates are realized, otherwise only one of them is realized. This constant number of updates for each $j \in adj[i]$ is possible because of the network homogeneity. In a heterogeneous network, it is necessary to update all reassignment gains for each such j, e.g., $n - 1$ updates for each $j \in adj[i]$. Gain update operation, including the key update in the priority queue, takes $O(n + \log m)$ time for each $j \in adj[i]$. The proposed algorithm does not allow the reassignment of the locked tasks in a pass since this may result in thrashing. A single pass of the algorithm ends when all tasks have been reassigned. Therefore, a single pass takes $O(m \log m + |E|(n + \log m))$ time.

At the end of a refinement pass, we have a sequence of tentative task reassignments and their respective gains. Then from this sequence, we construct the maximum prefix sum of gains which incurs the maximum decrease in the cost of the initial assignment. The permanent realization of the reassignments in this prefix is efficiently achieved by rolling back the remaining moves of the whole sequence. This assignment becomes the initial assignment for the next pass of the algorithm. Allowing tentative reassignments with negative gains provides a limited hill-climbing ability. The overall refinement process terminates if the maximum prefix sum of a pass is not positive. Similar to most of the FM-based algorithms, the proposed refinement algorithm obtains major improvements only in the first few passes. Hence, we allow only a constant number of passes for the sake of efficiency. Thus, the overall runtime of the proposed refinement algorithm is $O((m + |E|)(n + \log m))$.

5. Proposed assignment heuristics

In this section, we propose task assignment heuristics which exploit the clustering metric, assignment order and refinement heuristics proposed in the previous sections. The heuristic proposed in §5.1 is referred to as a single level approach in order to differentiate it from the multilevel ones presented in §5.2 and §5.3.

5.1. SLA: single level task assignment

The SLA algorithm uses the agglomerative clustering method described in §2.3 to reduce the problem size and assigns tasks to processors in an order imposed by either of the criteria described in §3. The task which is selected for assignment is assigned to its favorite processor according to the modified execution times of tasks. The heuristic SLA has a loose asymptotic upper bound of $O(|E|^2 n + |E| m \log m)$. The pseudocodes for the SLA and its assignment phase are given in Figs. 5 and 6, respectively.

The SLA algorithm continuously forms supertasks by merging pairs of tasks with the maximum positive cluster-

```
ASSIGN $(G, Q, x, i)$
    DELETE$(Q, i)$
    assign task $i$ to its favorite processor
    for each task $j \in Adj[i]$ do
        $Adj[j] \leftarrow Adj[j] - i$
        for each processor $q \in P - \{p\}$ do
            $x_{jq} \leftarrow x_{jq} + c_{ij}$
    for each task $j \in Adj[i]$ do
            UpdateKey$(Q, j)$
```

Fig. 6. Algorithm for assigning task i to processor p in SLA.

ing profit. If the clustering profits of all tasks/supertasks become negative, then a task/supertask is selected according to one of the proposed assignment criteria and is assigned to its favorite processor. Assigning a supertask to a processor effectively means assigning all its constituent tasks to that processor. Note that after the assignment of a task, the clustering profits of some unassigned task pairs may become positive and hence the algorithm may form intermittent clusters. After each clustering and assignment, the key values of the unassigned tasks may change. Therefore, the key values of the tasks in the priority queue are updated appropriately after these two operations. The algorithm given in Fig. 2 already handles the updates after a clustering step. When a task/supertask i is assigned to its favorite processor, the execution times of all unassigned tasks/supertasks that are adjacent to it are updated according to Eq. (3), and their clustering profits and best mates are recomputed. The SLA algorithm terminates when all tasks are assigned.

5.2. *Multilevel task clustering and refinement*

Here, we propose a multilevel approach for the two-phase assignment framework. The multilevel approach has previously proven to be successful in graph and hypergraph partitioning problems [9,11,12,25,28]. There are three phases in the multilevel approach: clustering, initial solution, and refinement. For the task assignment problem, we use the proposed clustering heuristics to reduce the original problem down to a series of smaller problems. We then adopt the assignment heuristics of §5.1 to obtain an initial solution. Then, we use the refinement heuristics proposed in §4 periodically while projecting the initial solution to the original problem. Note that in graph and hypergraph partitioning problems, the processors are assumed to be homogeneous and there is a balance constraint. Therefore, the clustering, initial solution, and refinement heuristics developed for these problems are not directly applicable to our target problem.

5.2.1. *Clustering phase*

In this phase, the given TIG $G = G_0 = (T_0, E_0)$ is coarsened into a sequence of smaller TIGs $G_1 = (T_1, E_1), \ldots, G_k = (T_k, E_k)$, where $|T_0| > |T_1| > \cdots > |T_k|$. This coarsening is achieved by coalescing disjoint subsets of tasks of G_ℓ at level ℓ into supertasks such that each supertask in G_ℓ forms a single task of $G_{\ell+1}$ at level $\ell + 1$. The execution times of each task of $G_{\ell+1}$ become equal to the sum of execution times of its constituent tasks in G_ℓ. The edge set of each supertask is set to the weighted union of the edge sets of its constituent tasks, where the internal edges are deleted. Coarsening phase terminates when the number of tasks in the coarsened TIG reduces below the number of processors or reduction on the number of tasks between successive levels is below 10% (i.e., $|T_{\ell+1}|/|T_\ell| > 0.90$). We use the clustering profit metric presented in §2.2 within the following four clustering heuristics.

A. Matching-based clustering: Matching-based clustering permits the clustering of pairs of tasks at a level and it works as follows. For each edge (i, j) in G_ℓ, the clustering profit α_{ij} for tasks i and j is calculated. Then, the edges with nonnegative clustering profits are visited in the descending order of clustering profits. If both of the incident tasks are not matched yet, then these two tasks are merged into a cluster. At the end, unmatched tasks remain as singleton clusters for the next level. Note that this heuristic does not find a maximum weighted matching in terms of the clustering profits. However, it is possible to compute one in $O(m|E|\log m)$ time [39].

B. Randomized semi-agglomerative clustering: In this scheme, each task i is assumed to constitute a singleton cluster, $C_i = \{i\}$, at the beginning of each coarsening level. We also use C_i to denote the supertask cluster that contains task i during the clustering process. The clusters are visited in a random order. If a task i has already been clustered (i.e., $|C_i| > 1$), then it is skipped, otherwise it selects a neighboring singleton or supertask cluster with the maximum nonnegative clustering profit to join. If the clustering profits of a task i are all negative, then task i remains unclustered at the current coarsening level. The clustering quality of this scheme is not predictable, because it highly depends on the order in which the task clusters are visited. That is, at each run, this clustering scheme may form different clusters. Therefore, we use this clustering scheme in a randomized assignment algorithm which we run many times to find solutions to a task assignment problem.

C. Semi-agglomerative clustering: This clustering scheme is very similar to the randomized semi-agglomerative clustering. The only difference is that, a single task to be clustered is not selected randomly, instead, a single task with the highest clustering profit is selected to be clustered. The solution quality obtained by this scheme is more predictable. In fact, it gives relatively better solution quality than the average solution quality of the randomized version. But it is also very likely to be stuck to a local optimal solution whose refinement may not be easy.

D. Agglomerative clustering: This clustering scheme, different from the semi-agglomerative one, allows two supertask clusters to be merged at a coarsening level. In a sense, it tries to overcome the limitations of the semi-agglomerative scheme. Note that this clustering approach is the application of the agglomerative clustering algorithm presented in §2.3 in a multilevel setting.

5.2.2. *Initial assignment phase*

The aim of this phase is to find an assignment for the task assignment problem at the coarsest level. Although we use the SLA algorithm described in §5.1 to find the initial assignments, any other algorithm is also viable.

5.2.3. *Uncoarsening phase*

At each level ℓ, assignment A_ℓ found on the task set T_ℓ is projected back to an assignment $A_{\ell-1}$ on the task set $T_{\ell-1}$. The constituent tasks of each supertask in $G_{\ell-1}$ are assigned to the processor to which the respective supertask is assigned in G_ℓ. Obviously, $A_{\ell-1}$ has the same cost as A_ℓ. We then refine this assignment by using the refinement algorithm given in §4. Note that even if the assignment A_ℓ is at a local minimum (i.e., reassignment of any single task does not decrease the assignment cost), the projected assignment $A_{\ell-1}$ may not be as such. Since $G_{\ell-1}$ is finer, it has more degrees of freedom that can be exploited to further improve the assignment $A_{\ell-1}$. In a multilevel framework, the refinement scheme becomes very effective, because the initial assignment available at each level is already a good assignment.

5.3. *Multilevel task assignment and refinement*

In the multilevel algorithms given in §5.2, the original problem is reduced by forming task clusters. In this section, we propose another approach to reduce the original problem under the multilevel setting by assigning some of the tasks to processors at each level. In essence, the algorithm proposed in this section is a multilevel approach for single-phase assignment framework.

Suppose a randomized assignment algorithm, e.g., the multilevel algorithm with randomized semi-agglomerative clustering approach described in §5.2.1-B, is run several times for a given task assignment problem instance. If a task i is found to be assigned to the same processor p in all or the majority of the solutions produced by these runs, then we can expect the processor p to be a "good" assignment for the task i. Based on this expectation, we find five different assignments for a given task assignment problem by using a randomized multilevel assignment algorithm. From those five assignments, we choose the best four assignments to eliminate the negative effects of significantly bad assignments. If task i is assigned to the same processor p in all of the four assignments, then it is assigned to processor p at the current level. Then, task i together with its edges are deleted from the TIG for the following coarsening levels. But in the refinement phase, task i will be free to be reassigned to any other processor at higher levels. After this assignment, we adjust the execution costs of the adjacent tasks. For each edge $(i, j) \in E$, we add c_{ij} to the execution times of task j on all processors except p. This approach promises high-quality solutions, but it has a relatively high running time. This tradeoff can be controlled by using less than five assignments, but in that case it is likely to obtain worse solutions.

6. Experiments

6.1. *Data set*

We evaluate the performance of the proposed task assignment algorithms on two sets of problem instances. The first set of problems are those whose TIGs are in tree structure. The second set of problem instances are those whose TIGs are general graphs.

The topologies of the tree TIGs are generated as follows. First, for each $m = 100, 200, 300, 1200$, and 2600, we create a complete graph with m vertices (tasks). Then, we pick edges randomly to grow a forest until a spanning tree of the complete graph is obtained.

The topologies of the general TIGs are obtained from the DWT matrices of Harwell–Boeing matrix collection via MatrixMarket [38]. The rows/columns of the matrices correspond to the vertices of the TIGs, where the off-diagonal nonzeros correspond to the edges of the TIGs. We choose the DWT set to designate task interactions, because it contains matrices that are rich in nonzero patterns and hence enables generation of TIGs that are rich in interaction forms. The properties of the TIGs are given in Table 2.

Another parameter is the number of processors n. We evaluate the performance of the proposed algorithms for $n = 4$, 8, and 16 processors.

Once the topologies of the TIGs are obtained, we assign random integers in range 1–100 to the edges to represent communication cost of the interactions. We use the methods discussed in [2] to generate expected execution time to compute (ETC) matrix for each TIG and n pair. Recall that the ETC matrix is of size $m \times n$, where the entry (i, p) is the expected execution time of task i on machine p, i.e., x_{ip}. We generate all four types of ETC matrices: low task heterogeneity and low machine heterogeneity (ETC0), low task heterogeneity and high machine heterogeneity (ETC1), high task

Table 2
Properties of TIGs obtained from DWT matrices

Topology	m	$\|E\|$	Vertex degree		
			min	max	avg
DWT59	59	104	1	5	3.53
DWT66	66	127	1	5	3.85
DWT72	72	75	1	4	2.08
DWT209	209	767	3	16	7.34
DWT221	221	704	3	11	6.37
DWT234	234	300	1	9	2.56
DWT1242	1242	4592	1	11	7.39
DWT2680	2680	1173	3	18	8.34

heterogeneity and low machine heterogeneity (ETC2), high task heterogeneity and high machine heterogeneity (ETC3). ETC matrices are further classified into two categories [2]. In the *consistent* ETC matrices, there is a special structure which implies that if a machine has a lower execution time than another machine for some task, then the same is true for any other task. The *inconsistent* ETC matrices have no such special structure. We evaluate the performance of the proposed algorithms with inconsistent ETC matrices.

Final parameter is the communication-to-computation ratio r_{com}. Let the scaling factor be $f = r_{\text{com}} \times \left(\sum_{i,j} c_{ij}\right) \Big/ \left(\sum_{i,p} x_{ip}/n\right)$, where the numerator represents the total intertask communication cost and the denominator represents the total task execution cost on the average. Then scaling each x_{ip} by f results in an average communication-to-computation ratio of r_{com}. We evaluate the performance of the proposed algorithms with $r_{\text{com}} = 0.7$, 1.0, and 1.4. These three choices characterize the problem instances in which computations have more impact than the communications, computations and communications have comparable impacts, and communications have more impact, respectively.

In order to be able to obtain reproducible performance results, we generate 10 random instances for a given quartet of TIG, n, r_{com}, and ETC type. The performance of an algorithm on a problem instance is given as the average of the 10 runs corresponding to these random instances.

6.2. Set up

We have implemented the eight algorithms given in Table 3 from the literature in order to assess the performance of the proposed algorithms. We run the meta-heuristics (the first four algorithms) on tree TIGs with 100, 200, and 300 vertices and on general TIGs with less than 234 vertices. The KLZ and VML algorithms are run on all problem instances. TOpt is run on tree TIGs with all parameters and A* is run on general TIGs with 59, 66, and 72 vertices to find optimal solutions for 4-processor systems.

We apply the refinement algorithm given in §4 to improve the solutions of all but the exact algorithms given above. Since the following tables present the quality of the refined solutions, we give the improvement ratios in Table 4. The numbers in this table are computed as follows. For each problem instance specified by a quartet of TIG, n, r_{com}, and ETC type which has 10 random samples, we divide the quality of the unrefined solution by that of the refined solution and take the average of these ratios. Hence, by multiplying the quality of the solutions given in the following tables with these average values, the average quality of the solutions obtained by the original algorithms can be found.

Table 5 summarizes the properties of the proposed algorithms whose performance results are displayed in the following tables. We use assignment order according to grab

Table 3
Existing task assignment algorithms from the literature

Algorithm	Description
GA	Genetic algorithm of Ahuja et al. [1] applied to task assignment
SA	Simulated annealing [24]
TSN	A combination of tabu search and noising [13]
PSO	Particle swarm optimization [42] modified to handle heterogeneous processors
KLZ	Kopidakis et al.'s MaxEdge algorithm which is the best of the two heuristics proposed in [31]
VML	Lo's task assignment algorithm [34]
TOpt	Bokhari's shortest tree algorithm [6]
A*	A state space search algorithm based on A* algorithms given in [27] and [48]

Table 4
The ratios of unrefined solutions' quality to refined solutions' quality

TIG	ETC	GA	SA	TSN	PSO	VML	KLZ
Tree	0	1.02	1.01	1.03	1.10	1.00	1.00
	1	1.03	1.17	1.49	2.44	1.00	1.04
	2	1.02	1.01	1.04	1.11	1.00	1.00
	3	1.10	1.01	1.08	1.40	1.30	1.05
	Average	1.04	1.05	1.16	1.51	1.08	1.02
General	0	1.01	1.04	1.08	1.11	1.00	1.08
	1	1.01	1.17	1.40	2.69	1.00	1.30
	2	1.01	1.04	1.08	1.13	1.00	1.11
	3	1.09	1.01	1.07	1.33	1.16	1.21
	Average	1.03	1.07	1.16	1.57	1.04	1.17

Table 5
Proposed task assignment algorithms

Algorithm	Description
SLA	Single level algorithm described in §5.1
MLC-M	Multilevel algorithm with matching-based clustering described in §5.2.1-A
MLC-S	Multilevel algorithm with semi-agglomerative clustering described in §5.2.1-C
MLC-A	Multilevel algorithm with agglomerative clustering described in §5.2.1-D
MLA	Multilevel algorithm with assignment-based reduction described in §5.3

affinity in the single level algorithm and also in the initial solution phase of the multilevel clustering and refinement algorithms. We run the proposed algorithms on all problem instances and compare their performance with the appropriate existing algorithms.

6.3. Experiments with tree TIGs

The performance of the existing algorithms are normalized with respect to the optimal solutions found by the TOpt algorithm and the average results are given in Table 6. We do not present the data for the proposed algorithms, because

Table 6
Averages of the normalized solution qualities of the existing algorithms on tree TIGs

n	r_{com}	ETC	GA	SA	TSN	PSO	VML	KLZ
4	0.7	0	1.02	1.08	1.12	1.02	1.01	1.02
		1	1.01	1.16	1.17	1.02	1.00	1.01
		2	1.04	1.09	1.12	1.02	1.01	1.02
		3	1.11	1.12	1.17	1.13	1.14	1.08
	1.0	0	1.03	1.06	1.09	1.02	1.02	1.02
		1	1.01	1.11	1.13	1.01	1.00	1.01
		2	1.03	1.07	1.09	1.02	1.02	1.03
		3	1.08	1.07	1.13	1.09	1.11	1.07
	1.4	0	1.04	1.05	1.06	1.03	1.02	1.03
		1	1.01	1.06	1.07	1.02	1.00	1.02
		2	1.04	1.05	1.07	1.02	1.02	1.03
		3	1.05	1.04	1.09	1.07	1.07	1.05
8	0.7	0	1.02	1.13	1.13	1.03	1.02	1.02
		1	1.03	1.27	1.24	1.08	1.00	1.02
		2	1.03	1.13	1.13	1.03	1.02	1.02
		3	1.17	1.21	1.26	1.21	1.21	1.14
	1.0	0	1.02	1.09	1.09	1.03	1.02	1.02
		1	1.03	1.20	1.16	1.04	1.00	1.03
		2	1.03	1.10	1.10	1.03	1.02	1.02
		3	1.12	1.14	1.18	1.13	1.14	1.12
	1.4	0	1.03	1.07	1.08	1.04	1.02	1.02
		1	1.01	1.10	1.08	1.03	1.00	1.02
		2	1.04	1.07	1.08	1.04	1.03	1.02
		3	1.08	1.09	1.13	1.09	1.09	1.09
16	0.7	0	1.03	1.15	1.14	1.07	1.02	1.03
		1	1.06	1.35	1.33	1.18	1.00	1.04
		2	1.03	1.16	1.15	1.08	1.03	1.02
		3	1.22	1.30	1.31	1.31	1.28	1.20
	1.0	0	1.03	1.10	1.09	1.06	1.03	1.03
		1	1.05	1.21	1.20	1.19	1.00	1.05
		2	1.03	1.12	1.12	1.06	1.03	1.02
		3	1.16	1.22	1.24	1.24	1.18	1.17
	1.4	0	1.03	1.08	1.08	1.06	1.03	1.03
		1	1.03	1.14	1.14	1.11	1.00	1.05
		2	1.04	1.08	1.08	1.06	1.04	1.03
		3	1.12	1.16	1.17	1.19	1.13	1.13
	Averages	0	1.03	1.09	1.10	1.04	1.02	1.02
		1	1.03	1.18	1.17	1.08	1.00	1.03
		2	1.03	1.10	1.10	1.04	1.02	1.02
		3	1.12	1.15	1.19	1.16	1.15	1.12

The normalization is with respect to the optimal solutions found by TOpt.

they obtain solutions whose qualities are very close to those obtained by the TOpt algorithm. All of the proposed algorithms perform almost equally well; the ratios of the solutions obtained by them are in the range 1.00–1.01. These almost equal solution qualities verify the effectiveness of the proposed clustering and refinement heuristics.

As seen in Table 6, problem instances with ETC3 are the hardest instances, because of the higher degree of heterogeneity, for all algorithms except SA and TSN. These two meta-heuristics use one-way moves which reassign one task from its current processor to another and two-way moves which exchange two tasks assigned to two different processors. The two-way moves may lead to solutions that are far from the optimal solution in which all tasks are assigned to a restricted set (even just one processor) of processors. This kind of optimal solutions exists for the problem instances with lower degree of heterogeneity (ETC0, ETC1, and ETC2).

Performances of all existing algorithms degrade with the increasing number of processors, because the search space gets larger. The degradations are higher in meta-heuristics as compared to KLZ and VML, because the meta-heuristics explicitly search this larger space.

Although the algorithms given in Table 3 are quite different in nature, they all perform better as r_{com} increases for fixed n and ETC type. Upon observing this phenomenon, we investigated the solutions of the existing algorithms prior to the refinement process. In fact, there is no such pattern in the unrefined solutions of PSO, KLZ, and VML. Such a pattern exists in the solutions of GA for ETC3, and in the solutions of SA and TSN. Therefore, we can say that our refinement algorithm gives rise to such a phenomenon. Since the communication costs increase with the increasing r_{com}, the refinement algorithm finds amplified opportunities to reduce the total cost.

VML performs well for all instances with ETC0, ETC1 and ETC2. However, KLZ outperforms VML for ETC3 instances especially for large n. This is because VML's performance decreases with the increasing number of processors due to its dependence on the success of the grab phase. For small number of processors, the grab phase works but for large number of processors, the assignments of VML are generally provided by the greedy phase.

6.4. Experiments with general TIGs

As evident from the performance results given for tree TIGs, the instances with large processor counts and ETC of type 3 assist in distinguishing among task assignment heuristics. Therefore, the performance of the proposed algorithms on general TIGs are presented only for $n = 16$ and ETC3. The solutions of the proposed algorithms are normalized with respect to the best solutions found by the existing algorithms and the average results are given in Table 7. As seen from the table, all the proposed algorithms perform almost equally well, and they perform better than the existing algorithms. Note that the performance gap between the proposed algorithms and the existing ones closes as the number of tasks increases. However, this is mostly due to the refinement algorithm that we use to improve the solutions of the existing algorithms. In the dwt1242 and dwt2680 instances, the best unrefined solutions obtained by the existing algorithms are 1.08, 1.14, and 1.20 multiples of the best refined solution, on the average, for $r_{\text{com}} = 0.7$, 1.0, and 1.4, respectively.

For small graphs (dwt59, dwt66, dwt72), we run A* on the problem instances with 4 processors to find optimal solutions. The normalized qualities of the solutions obtained

Table 7
Averages of the normalized solution qualities of the proposed algorithms on general TIGs for $n = 16$ and ETC3

TIG	r_{com}	SLA	MLC-M	MLC-S	MLC-A	MLA
dwt59	0.7	0.94	0.94	0.93	0.94	0.93
	1.0	0.91	0.91	0.91	0.91	0.91
	1.4	0.95	0.96	0.95	0.95	0.96
dwt66	0.7	0.91	0.91	0.91	0.91	0.90
	1.0	0.93	0.94	0.94	0.93	0.94
	1.4	0.96	0.96	0.96	0.96	0.97
dwt72	0.7	0.87	0.88	0.87	0.87	0.88
	1.0	0.88	0.89	0.89	0.88	0.89
	1.4	0.93	0.94	0.94	0.93	0.94
dwt209	0.7	0.98	0.97	0.99	0.97	0.96
	1.0	1.00	0.98	0.99	1.00	0.98
	1.4	0.99	0.98	0.98	0.98	0.97
dwt221	0.7	0.98	0.97	0.97	0.97	0.97
	1.0	0.95	0.95	0.95	0.95	0.95
	1.4	0.97	0.97	0.97	0.97	0.97
dwt234	0.7	0.93	0.93	0.94	0.93	0.94
	1.0	0.93	0.92	0.92	0.92	0.93
	1.4	0.93	0.93	0.93	0.93	0.94
dwt1242	0.7	1.00	0.99	1.00	1.00	0.98
	1.0	0.98	0.97	0.97	0.97	0.97
	1.4	0.98	0.97	0.98	0.98	0.98
dwt2680	0.7	1.02	1.01	1.02	1.02	1.00
	1.0	0.99	0.99	0.99	0.99	0.98
	1.4	0.98	0.98	0.98	0.98	0.98
Average	0.7	0.95	0.95	0.95	0.95	0.95
	1.0	0.95	0.94	0.94	0.95	0.94
	1.4	0.96	0.96	0.96	0.96	0.96

The normalization is with respect to the best solutions obtained by the existing algorithms.

by the proposed algorithms are again in between 1.00 and 1.01 multiples of the optimal solutions. Although the task assignment problem with these instances are quite easy, the refined solutions of the existing heuristics are not as good as the solutions of the proposed heuristics. For example, for ETC3 and $r_{\mathrm{com}} = 0.7$, the best and worst solutions found by the existing algorithms are far from the optimal by factors of 1.05 and 1.10.

6.5. Scalability studies

In order to see whether the proposed clustering metric and refinement heuristics scale well with the increasing number of processors and tasks, we test the proposed algorithms with large tree instances (whose optimal solutions are found by the TOpt algorithm proposed by Bokhari). For Grid applications, large instances are usually considered to be those in which the number of tasks is at least one order of magnitude larger than the number of available processor [21–23]. Therefore, we have created tree TIGs with 10 000 tasks to be mapped to 100 processors. Since the heuristics VML and KLZ use clustering approaches we also give their performance.

Table 8
Averages of the normalized solution qualities of the proposed algorithms, VML, and KLZ on tree TIGs with 10 000 tasks on 100 processors

r_{com}	ETC	SLA	MLC-M	MLC-S	MLC-A	MLA	VML	KLZ
0.7	0	1.00	1.00	1.00	1.00	1.01	1.06	1.05
	1	1.00	1.00	1.00	1.00	1.00	1.00	1.10
	2	1.00	1.00	1.00	1.00	1.01	1.05	1.03
	3	1.01	1.02	1.01	1.01	1.07	1.31	1.32
1.0	0	1.00	1.00	1.00	1.00	1.01	1.07	1.05
	1	1.00	1.00	1.00	1.00	1.00	1.00	1.10
	2	1.00	1.00	1.00	1.00	1.01	1.06	1.03
	3	1.01	1.01	1.01	1.01	1.07	1.24	1.27
1.4	0	1.00	1.00	1.00	1.00	1.01	1.06	1.05
	1	1.00	1.00	1.00	1.00	1.00	1.00	1.05
	2	1.00	1.00	1.00	1.00	1.01	1.07	1.04
	3	1.01	1.01	1.01	1.01	1.07	1.19	1.21
Averages	0	1.00	1.00	1.00	1.00	1.01	1.06	1.05
	1	1.00	1.00	1.00	1.00	1.00	1.00	1.09
	2	1.00	1.00	1.00	1.00	1.01	1.06	1.03
	3	1.01	1.01	1.01	1.01	1.07	1.25	1.26

The normalization is with respect to the optimal solutions found by TOpt.

The performance of VML, KLZ, and the proposed heuristics are given in Table 8. The quality of the solutions are again normalized with respect to the optimal solutions found by Bokhari's algorithm. As seen from the table, all of the proposed heuristics obtain solutions whose qualities are again very close to the optimal solutions. All results, except those of the MLA algorithm for ETC3 instances, are within the range 1.00–1.02 of the optimal solutions. However, the VML and KLZ heuristics are not found to be as scalable as the proposed heuristics. The performance of the VML and KLZ heuristics drop especially for the problem instances with ETC3. The averages of VML and KLZ given in Table 6 as 1.15 and 1.12 for the small problem instances with ETC3 become 1.25 and 1.26 for the large problem instances as seen in Table 8. Note also that in the large problem instances, the ratios of the qualities of unrefined solutions to those of the refined solutions for ETC3 are 1.99 and 1.10 for the VML and KLZ heuristics, respectively. For the small tree TIGs given in Table 6, these ratios were 1.30 and 1.05 as given in Table 4, i.e., the original heuristics do not scale as well as the proposed task assignment heuristics.

6.6. Running times

The algorithms were implemented in C/C++, compiled with gcc/g++ compilers using -O3 optimization flag, and run on a machine with 2.66 GHz P4 processor, 512 KB cache,

Table 9
The running times of the proposed task assignment heuristics on the tree TIGs with 10 000 vertices

n	ETC	SLA	MLC-M	MLC-S	MLC-A	MLA
25	0	0.58	0.16	0.42	0.48	1.65
	1	1.03	0.14	0.43	0.54	1.53
	2	0.60	0.15	0.42	0.48	1.68
	3	0.54	0.17	0.42	0.48	1.80
50	0	0.76	0.30	0.62	0.70	2.71
	1	1.43	0.24	0.63	0.81	2.43
	2	0.78	0.28	0.62	0.71	2.68
	3	0.71	0.32	0.62	0.73	2.90
100	0	1.11	0.64	1.07	1.19	5.18
	1	2.01	0.54	1.01	1.34	4.61
	2	1.18	0.60	1.05	1.20	5.03
	3	1.06	0.66	1.07	1.19	5.65

The numbers are the averages of 10 runs for each parameter set. Running times are given in seconds.

and 2 GB main memory. We do not present the time of the A* algorithm because it is not meant to be a general purpose solution.

Here we first report the running times of the proposed heuristics in comparison with the existing heuristics given in Table 3 for the fixed parameter set of $n = 16$, $r_{\text{com}} = 1.0$, and ETC3 with varying TIGs. The GA, PSO, and TSN algorithms have large execution times. For example, GA obtains a solution for the dwt1242 instance in 161 s. PSO obtains a solution for the dwt209 instance in 29 s. TSN obtains a solution for the dwt72 instance in 15 s. TOpt, KLZ, and all of the proposed algorithms except MLA obtain a solution for the tree TIG with $m = 2600$ in less than 0.05 s. MLA solves the same problem in 0.16 s, VML solves the problem in 0.85 s, and SA solves the problem in 27 s. KLZ and all of the proposed algorithms except MLA solve dwt2680 in less than 0.07 s. MLA solves the same problem in 0.26 s. VML solves the same problem in 38 s. This again shows that VML's grab phase do not work well in general graphs with large number of vertices.

Next we report the running times of the proposed heuristics to see whether their running times scale well with the increasing number of processors for a large number of tasks. We run the proposed heuristics for the parameter set $n = 25, 50, 100$, $r_{\text{com}} = 0.7, 1.0, 1.4$, all ETC types with the tree TIG having 10 000 vertices. Table 9 displays the averages of the running times in seconds.

As seen from the table, all of the proposed algorithms scale almost linearly with the increasing number of processors. The slowest algorithm is again MLA. The table also reveals that clustering-based multilevel algorithms (MLC-M, MLC-S, and MLC-A) are faster than the single level algorithm. We admit that the MLA algorithm is not as good as the other proposed algorithms in terms of both running time and solution quality. Recall from §5.3 that it is a multilevel approach for the single-phase assignment framework. Therefore, we keep it in the presentation for the sake of completeness.

7. Conclusion

We proposed a novel clustering approach which considers the task execution costs as well as the communication costs between the tasks; two metrics to determine the order in which tasks are assigned to processors; an iterative-improvement-based heuristic for refining task assignments; a family of fast heuristics, including multilevel ones, which obtain high-quality solutions.

The key point in the clustering heuristics including ours is the following. Clustering any two tasks will guarantee a saving which is equal to the communication cost that incurs if these tasks are assigned to any two different processors under the homogeneous network model. However, when the network is heterogeneous, the amount of saving will no longer be a constant and it will differ according to the processors to which the tasks are assigned. We tried incorporating minimum possible amount of saving into the clustering metric but it did not work. We think that developing clustering heuristics for the heterogeneous networks is a research issue.

Acknowledgments

All of the algorithms mentioned in §6 are available upon request. We acknowledge the participation of Melih Onuş in the development of the GA, SA, TSN, and PSO algorithms. Part of the experimental tests were carried out at the TUBITAK ULAKBIM High Performance Computing Center.

References

[1] R.K. Ahuja, J.B. Orlin, A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, Comput. Oper. Res. 27 (10) (2000) 917–934.
[2] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, S. Ali, Task execution time modeling for heterogeneous computing systems, in: C. Raghavendra (Ed.), Proceedings of the Ninth Heterogeneous Computing Workshop (HCW 2000), IEEE, Cancun, Mexico, 2000, pp. 185–199.
[3] F. Berman, High-performance schedulers, in: I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, Los Altos, CA, 1999, pp. 279–309, Chapter 12.
[4] A. Billionnet, Allocating tree structured programs in a distributed system with uniform communication costs, IEEE Trans. Parallel Distrib. Systems 5 (4) (1994) 445–448.
[5] C. Boeres, A. Lima, V.E.F. Rebello, Hybrid task scheduling: integrating static and dynamic heuristics, in: Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing, IEEE, New York, 2003, pp. 199–206.
[6] S.H. Bokhari, A shortest tree algorithm for optimal assignments across space and time in distributed processor system, IEEE Trans. Software Engrg. 7 (6) (1981) 583–589.

[7] N.S. Bowen, C.N. Nikolaou, A. Ghafoor, On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems, IEEE Trans. Comput. 41 (3) (1992) 257–273.

[8] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.

[9] T.N. Bui, C. Jones, A heuristic for reducing fill in sparse matrix factorization, in: Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Philadelphia, 1993, pp. 445–452.

[10] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in grid computing, Concurrency Comput.: Practice Exp. 14 (13–15) (2002) 1507–1542.

[11] U.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Systems 10 (7) (1999) 673–693.

[12] U.V. Çatalyürek, C. Aykanat, PaToH: A multilevel hypergraph partitioning tool, version 3.0, Technical Report BU-CE-9915, Computer Engineering Department, Bilkent University, 1999.

[13] W.-H. Chen, C.-S. Lin, A hybrid heuristic to solve a task allocation problem, Comput. Oper. Res. 27 (3) (2000) 287–303.

[14] M.K. Dhodhi, I. Ahmad, A. Yatama, I. Ahmad, An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems, J. Parallel Distrib. Comput. 62 (9) (2002) 1338–1361.

[15] K. Efe, Heuristic models of task assignment scheduling in distributed systems, IEEE Comput. 15 (6) (1982) 50–56.

[16] H. El-Rewini, T.G. Lewis, H.H. Ali, Task Scheduling in Parallel and Distributed Systems, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1994.

[17] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Engrg. 15 (11) (1989) 1427–1436.

[18] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th Design Automation Conference, IEEE Press, New York, 1982, pp. 175–181.

[19] B. Folliot, P. Sens, Load sharing and fault tolerance manager, in: R. Buyya (Ed.), High Performance Cluster Computing, Vol. 1: Architectures and Systems, Prentice-Hall, Englewood Cliffs, NJ, 1999, pp. 534–552, Chapter 22.

[20] J. Gehring, A. Reinefeld, MARS—a framework for minimizing the job execution time in a metacomputing environment, Future Generation Comput. Systems 12 (1) (1996) 97–99.

[21] A. Giersch, Y. Robert, F. Vivien, Scheduling tasks sharing files on heterogeneous clusters, Technical Report RR-2003-28, LIP, ENS Lyon, France, May 2003.

[22] A. Giersch, Y. Robert, F. Vivien, Scheduling tasks sharing files from distributed repositories, Technical Report 5124, INRIA, February 2004.

[23] A. Giersch, Y. Robert, F. Vivien, Scheduling tasks sharing files on heterogeneous master-slave platforms, in: PDP'2004, 12th Euromicro Workshop on Parallel Distributed and Network-based Processing, IEEE Computer Society Press, Silver Spring, MD, 2004.

[24] Y. Hamam, K.S. Hindi, Assignment of program modules to processors: A simulated annealing approach, European J. Oper. Res. 122 (2) (2000) 509–513.

[25] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM), ACM Press, San Diego, CA, 1995, p. 28.

[26] Z. Juhasz, S.J. Turner, A new heuristic for the process-processor mapping problem, in: G. Kotsis, P. Kacsuk (Eds.), Proceedings of the Third Austrian–Hungarian Workshop on Distributed and Parallel Systems, DAPSYS 2000, Distributed and Parallel Systems: From Concepts to Applications, Kluwer, Dordrecht, 2000, pp. 91–94.

[27] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (3) (1998) 42–51.

[28] G. Karypis, V. Kumar, MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, September 1998.

[29] F. Kaudel, Comments on ‘Allocating Programs Containing Branches and Loops Within a Multiple Processor System’ by D. Towsley, IEEE Trans. Software Engrg. 16 (4) (1990) 471.

[30] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Tech. J. 49 (2) (1970) 291–307.

[31] Y. Kopidakis, M. Lamari, V. Zissimopoulos, On the task assignment problem: two new heuristic algorithms, J. Parallel Distrib. Comput. 42 (1) (1997) 21–29.

[32] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.

[33] C.-H. Lee, K.G. Shin, Optimal task assignment in homogeneous networks, IEEE Trans. Parallel Distrib. Systems 8 (2) (1997) 119–129.

[34] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, IEEE Trans. Comput. 37 (11) (1988) 1384–1397.

[35] Y.-C. Ma, T.-F. Chen, C.-P. Chung, Branch-and-bound task allocation with task clustering-based pruning, J. Parallel Distrib. Comput. 64 (11) (2004) 1223–1240.

[36] V.F. Magirou, An improved partial solution to the task assignment and multiway cut problems, Oper. Res. Lett. 12 (1992) 3–10.

[37] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (1999) 107–131.

[38] MatrixMarket, http://math.nist.gov/MatrixMarket.

[39] K. Mehlhorn, S. Naher, Leda: A Platform for Combinatorial and Geometric Computing, Cambridge University Press, Cambridge, 1999.

[40] M.G. Norman, P. Thanisch, Models of machines and computation for mapping in multicomputers, ACM Comput. Surv. 25 (3) (1993) 263–302.

[41] J.M. Orduña, F. Silla, J. Duato, On the development of a communication-aware task mapping technique, J. Systems Archit. 50 (4) (2004) 207–220.

[42] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocessors Microsystems 26 (8) (2002) 363–371.

[43] M.A. Senar, A. Ripoll, A. Cortés, E. Luque, Clustering and reassignment-based mapping strategy for message-passing architectures, J. Systems Archit. 48 (8–10) (2003) 267–283.

[44] B.A. Shirazi, A.R. Hurson, K.M. Kavi, Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Computer Society Press, Los Altos, CA, USA, 1995.

[45] H.J. Siegel, S. Ali, Techniques for mapping tasks to machines in heterogeneous computing systems, J. Systems Archit. 46 (8) (2000) 627–639.

[46] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Software Engrg. SE-3 (1) (1977) 85–93.

[47] K. Taura, A.A. Chien, Heuristic algorithm for mapping communicating tasks on heterogeneous resources, in: C. Raghavendra (Ed.), Proceedings of the Ninth Heterogeneous Computing Workshop (HCW 2000), IEEE, Cancun, Mexico, 2000, pp. 102–115.

[48] A.P. Tom, C.S.R. Murthy, Optimal task allocation in distributed systems by graph matching and state space search, J. Systems and Software 46 (1) (1999) 59–75.

[49] D. Towsley, Allocating programs containing branches and loops within a multiple processor system, IEEE Trans. Software Engrg. 12 (10) (1986) 1018–1024.

[50] D. Towsley, Correction to ‘Allocating Programs Containing Branches and Loops Within a Multiple Processor System’, IEEE Trans. Software Engrg. 16 (4) (1990) 472.
[51] J.B. Weissman, X. Zhao, Run-time support for scheduling parallel applications in heterogeneous nows, in: HPDC ’97: Proceedings of the Sixth International Symposium on High Performance Distributed Computing, IEEE, Portland, OR, USA, 1997, pp. 347–355.
[52] E.A. Williams, Design analysis and implementation of distributed systems from a performance perspective, Ph.D. Thesis, University of Texas at Austin, 1983.

Bora Ucar received the B.S. (1997) and M.S. (1999) degrees in Computer Engineering from Bilkent University, Ankara, Turkey. He expects to be graduated with a Ph.D. degree from the same department by September 2005. His research interests are combinatorial scientific computing and high-performance computing.

Cevdet Aykanat received the B.S. and M.S. degrees from Middle East Technical University, Ankara, Turkey, both in electrical engineering, and the Ph.D. degree from Ohio State University, Columbus, in electrical and computer engineering. He was a Fulbright scholar during his Ph.D. studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects, parallel computer graphics applications, parallel data mining, graph and hypergraph-partitioning, load balancing, neural network algorithms, high-performance information retrieval systems, parallel and distributed web crawling, parallel and distributed databases, and grid computing. He has (co)authored 35 technical papers published in academic journals indexed in SCI. He is the recipient of the 1995 Young Investigator Award of The Scientific and Technical Research Council of Turkey. He is a member of the ACM and the IEEE Computer Society. He has been recently appointed as a member of IFIP Working Group 10.3 (Concurrent Systems) and INTAS Council of Scientists.

Kamer Kaya graduated from Bilkent University, Turkey in 2004 with a M.Sc. degree in Computer Engineering where he is currently a Ph.D. candidate. His research deals with cryptography, parallel computing and algorithms.

Murat Ikinci received the B.S. (1996) and M.S. (1998) degrees in Computer Engineering from Bilkent University, Ankara, Turkey. He is currently a software project manager at the STM Inc located in Ankara, Turkey. His research interests include interoperability of distributed systems and network centric operations.