588224

Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments*

Shailendra Kumar and Sajal K. Das
Dept. of Computer Science & Engineering
The University of Texas at Arlington
Arlington, TX 76019-0015
E-mail: {skumar,das}@cse.uta.edu

Rupak Biswas
NAS Division
NASA Ames Research Center
Moffet Field, CA 94035-1000
E-mail: rbiswas@nas.nasa.gov

Abstract

The problem of partitioning irregular graphs and meshes for parallel computations on homogeneous systems has been extensively studied. However, these partitioning schemes fail when the target system architecture exhibits heterogeneity in resource characteristics. With the emergence of technologies such as the Grid, it is imperative to study the partitioning problem taking into consideration the differing capabilities of such distributed heterogeneous systems. In our model, the heterogeneous system consists of processors with varying processing power and an underlying non-uniform communication network. We present in this paper a novel multilevel partitioning scheme for irregular graphs and meshes, that takes into account issues pertinent to Grid computing environments. Our partitioning algorithm, called *MiniMax*, generates and maps partitions onto a heterogeneous system with the objective of minimizing the maximum execution time of the parallel distributed application. For experimental performance study, we have considered both a realistic mesh problem from NASA as well as synthetic workloads. Simulation results demonstrate that MiniMax generates high quality partitions for various classes of applications targeted for parallel execution in a distributed heterogeneous environment.

Keywords: Distributed heterogeneous systems, Grid computing, mesh applications, multilevel graph partitioning, load balancing, performance study.

I. Introduction

Exploiting existing heterogeneous systems to solve problems faster is a cost-effective approach as opposed to replacing these systems with yet powerful machines. The popularity of Grid computing infrastructures [5] has stirred active research in the arena of harnessing computational capacity from distributed heterogeneous systems. The Grid couples geographically distributed heterogeneous resources to provide a metacomputing platform promising computational power of magnitude never anticipated before at affordable cost. However, these promises come with challenges for the research community to successfully map varied applications on such heterogeneous collections of high performance systems.

The Information Power Grid (IPG) [14], NASA's successful venture into the Grid computing arena, aims to provide scientific and engineering communities orders of magnitude increase in their ability to solve problems that depend on the use of large-scale dispersed resources: aggregated computing, diverse data archives, laboratory instruments, engineering test facilities, and human collaborators. However, the scope of this work is limited only to the parallel and distributed solution of large-scale computational problems. Mesh applications are a class of problems that require such high-end computational power since performance must be scalable to hundreds of processors under the current technology [1], [4]. In the literature, these are termed as distributed supercomputing applications [5]. The IPG provides an infrastructure for parallel distributed computing across heterogeneous systems, thus satiating the computational demands of such applications. A parallel application could be initiated at any node (a uniprocessor workstation or a tightly-coupled supercomputer) on the Grid. The application would then have to be partitioned among available processors on the Grid, and the computation carried out in a parallel distributed fashion to generate results that are finally collected at the application initiator node.

* This work was supported by NASA Ames Research Center under Cooperative Agreement Number NCC 2-5395.

The successful deployment of compute-intensive applications in a Grid environment such as the IPG involves efficient partitioning on a truly heterogeneous distributed metasystem which makes no assumptions about the resources that comprise it. As more diverse resources are added, the Grid scales to larger capability and hence greater heterogeneity. The interconnects could range from a very high speed optical link between the IPG nodes to relatively slower intra-node communication (for a loosely-coupled network of workstations). There are also cases where the inter-node communication is slower than the intra-node links. Apart from these variations, there are no constraints on the network topology connecting these resources. In other words, we impose no uniformity on the processor architecture or the links that connect them. This leads to a purely heterogeneous system model with varying system characteristics such as processor computational speed, network communication speed, and network topology. In this paper, we demonstrate how diversity in system characteristics affects the partitioning of an application in such environments. Workload heterogeneity is reflected by the differing (random) weights of the vertices and edges of the graph representation of the application. Unlike previous models of mesh applications, we do not assume the graph to be sparse or uniformly distributed in terms of weights. Thus, a partitioning scheme for such general purpose graphs encompasses a wide variety of parallel distributed computing applications on the Grid.

Multilevel algorithms have been a universal approach to solving the graph partitioning problem for homogeneous system models. Partitioning schemes such as Metis [16], Chaco [12], and Jostle [23] employ multilevel strategies but fail to address the limitations imposed by heterogeneity in the underlying system model. They all assume the processing speeds of the computing resources to be uniform and the communication stratum connecting any two resources to be of equal capacity. They also assume a uniform network topology. Furthermore, they do not delve into the mapping problem that addresses how partitions are assigned to processors in a system to reduce application execution time, which is the ultimate objective. Above all, these schemes optimize the wrong metrics. Traditionally, the standard approach has been to divide the vertices of a graph into approximately equal-weight partitions (balance computations) and minimize the number of cut edges between partitions (minimize total runtime communication). We first show that minimizing edgecut may not necessarily reduce the parallel execution time of an application on a distributed suite of heterogeneous resources. Previous work reported in [10], [11] have also pointed out similar flaws of standard partitioning schemes. Our objective is to relax the above assumptions on system models, thereby overcoming the inherent shortcomings, and propose a novel partitioner for distributed heterogeneous environments.

Little prior work has been done on partitioning applications for heterogeneous Grid computing environments. Das et al. [3] studied heterogeneity in inter-cluster communication speeds to determine the feasibility of mesh applications in the Grid. The work did not consider heterogeneity within a cluster and diversity in system parameters, such as processing speeds and network connectivity, that are characteristic of any Grid. However, the authors attempted to optimize the correct metric, namely the execution time of the application, when partitioning a mesh application for a distributed environment. The optimization was based on a *throttle* value that depend on the input mesh application. As a result, this partitioning scheme will not work well for all possible types of applications. Furthermore, their optimization scheme suffers from the danger of falling into a local minima. Walshaw and Cross [24] modeled a heterogeneous communication network and considered the mapping problem, but the work was limited with respect to the extent of heterogeneity experienced in Grid computing environments. In particular, they optimize on edgecuts, a flawed metric for parallel computation as argued above. Schnekenburger and Huber [20] partitioned a data parallel program on a network of workstations with varying processor weights, but did not take into account the network heterogeneity.

We propose a more general purpose heterogeneous partitioner, called *MiniMax*, in the sense that it allows full heterogeneity in both the system and workload characteristics, and attempts to directly minimize the application runtime. MiniMax is tested against diverse systems and workloads to establish its feasibility for different types of metasystem architectures and a wide variety of parallel applications. MiniMax generates and maps high quality partitions and a near perfect balance is achieved in the execution times of individual processors comprising the metasystem. Our major contributions are optimizations based on the application execution time as opposed to indirect objectives targeting edgecuts and subsequent assignment of partitions to processors. We have also considered complete heterogeneity in the system as well as in the workload graph, thereby modeling a variety of metasystems and applications.

The remainder of this paper is organized as follows. In Section II, we describe the workload model, followed by a model of distributed heterogeneous metasystems in Section III. In Section IV, we formally define the partitioning problem. Section V discusses MiniMax, the proposed multilevel heterogeneous partitioner and its complexity analysis. In Section VI, we experimentally study the performance of MiniMax for realistic and synthetic workloads. Finally, Section VII concludes the paper with directions for future research.

II. Workload Model

Many compute-intensive and complex problems are modeled using unstructured (irregular) graphs and meshes [2], [18], [21]. These constitute a significant portion of high performance supercomputing applications that are ideal for a Grid environment. Such applications emerge from numerous scientific and engineering domains, namely computational fluid dynamics (CFD) [22], numerical methods for solving differential equations and matrix problems [8], VLSI testing [15], and data parallel computations for geographic and extra terrestrial information processing [17]. Finite element and finite volume methods generate unstructured meshes which are better represented as a dual graph [19] for the purpose of efficient partitioning and load balancing. Parallel implementations of these applications involve a graph representation of the problem and its efficient partitioning onto a target system.

Discretely, the problem domain can be modeled as a weighted undirected graph $G = (V, E)$, which we refer to as the *workload graph*. Each vertex v has a *computational weight* W^v, which reflects the amount of computation at v. An edge between vertices u and v has a *communication weight* $C^{u,v}$, which reflects the data dependency between them. However, this model does not impose any precedence constraint between u and v. In this paper, we have considered non-uniform random weight distribution of vertices and edges, simulating the workload model as in [9], [13].

When mapped to a distributed heterogeneous environment, the costs associated with these weights depend on system characteristics, such as the processing speed and the communication latency of network resources. For example, if vertices u and v are assigned to the same processor, the communication cost will be zero. On the other hand, if they are assigned to different processors, the communication cost will depend on the edge weight and the communication latency between the two processors involved.

III. Heterogeneous System Model

Partitioning applications for a Grid environment involves a heterogeneous system model of the Grid. Each Grid computational resource is referred to as a node, which could be a uniprocessor system, a symmetric multiprocessor (SMP) cluster, a distributed memory multicomputer, a massively parallel supercomputer, or a network of workstations. The Grid imposes no limitations on a system that could be coupled to the resource pool. Each node thus has different resource characteristics. The communication stratum coupling these diverse resources is also non-homogeneous. Again, the Grid imposes no constraint on the network topology and the communication latency between the nodes. Figure 1 shows a heterogeneous system model, with machines of different makes and a non-homogeneous communication network. This is a scaled-down model of the Grid, whereas actual Grid implementations like the IPG proposes an infrastructure coupling thousands of diverse resources. Each node in Fig. 1 is shown within a solid rectangle, and comprises of one or more processors communicating with other processors within the node via intra-node links, and with other nodes via inter-node links. Processors and links within a node could also be heterogeneous. With respect to the partitioning problem, the Grid can be viewed as a pool of processors dedicated to solving a parallel application. Each dark oval in Fig. 1 denotes a processor. We distinguish processors in the conglomerate pool by virtue

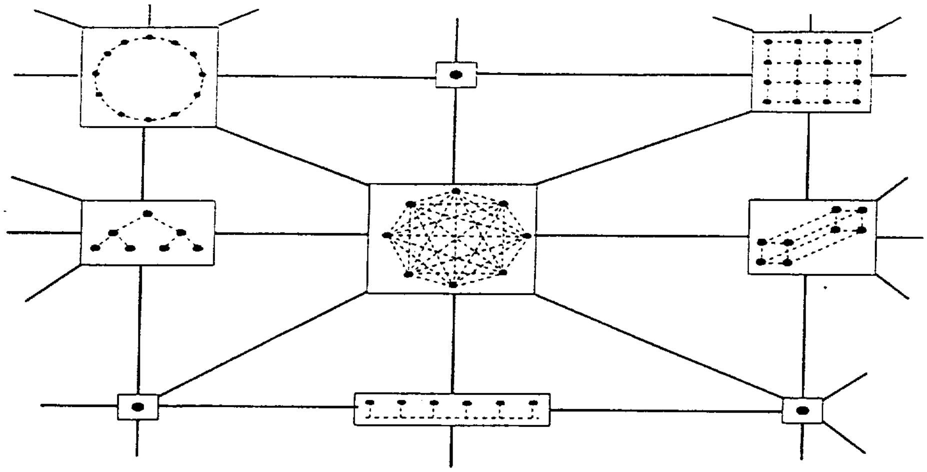
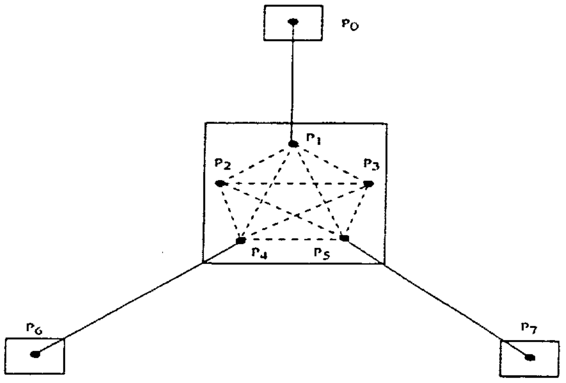


Fig. 1. System model of a heterogeneous Grid environment.

of their processing speed and the communication latency between them.

This heterogeneous processor pool can be represented as a weighted undirected graph $S = (P, L)$, which we refer to as the *system graph*. It consists of a set of vertices $P = \{p_1, p_2, \ldots, p_n\}$, denoting processors, and a set of edges $L = \{(p_i, p_j) \mid p_i, p_j \in P\}$, representing communication links between processors. Each processor (vertex) p has a *processing weight* s_p, modeling its processing cost per unit of computation. Each link (edge) has a *link weight* $c_{p,q}$, that denotes the communication latency (cost) per unit of communication between processors p and q. We assume full duplex communication for our experiments; however, this constraint could easily be relaxed with minor modifications to the implementation.

When two processors are not directly connected, their link weight is the sum of the link weights on the shortest path between them. For example, in Fig. 2, assuming all intra-node links of weight 1, and all inter-node links of weight 5, the weight of a link between processors p_0 and p_7 is 11, along the path p_0, p_1, p_5, p_7. This approach for link weight calculation gives us a complete weighted graph for the system from which we can derive a *communication cost matrix* that represents the communication cost between any two processors in the system. This matrix is symmetric for our case, as we assume full duplex links between nodes. The communication cost matrix in Fig. 2 corresponds to the adjacent system graph. Incorporating varying latency based on the direction of communication between two processors will lead to an asymmetric communication cost matrix.



$$\begin{bmatrix}
0 & 5 & 6 & 6 & 6 & 6 & 11 & 11 \\
5 & 0 & 1 & 1 & 1 & 1 & 6 & 6 \\
6 & 1 & 0 & 1 & 1 & 1 & 6 & 6 \\
6 & 1 & 1 & 0 & 1 & 1 & 6 & 6 \\
6 & 1 & 1 & 1 & 0 & 1 & 5 & 6 \\
6 & 1 & 1 & 1 & 1 & 0 & 6 & 5 \\
11 & 6 & 6 & 6 & 6 & 6 & 0 & 11 \\
11 & 6 & 6 & 6 & 6 & 6 & 11 & 0
\end{bmatrix}$$

Fig. 2. The communication cost matrix for a heterogeneous network.

In order to reflect full heterogeneity in the system model, vertex and edge weights in the system graph should take random values. The network topology (connectivity) should also be random. The motivation behind these requirements is the desire to model a true Grid computing environment. We generate system graphs having random network topology with random vertex and edge weights. We next compute the communication cost matrix based on the shortest path between processors, as discussed above. We generate several instances of the system graph in this way, to evaluate our partitioning scheme.

A. *Processing Cost*

Given an assignment of vertices in the workload graph to processors in the system graph, vertices incur a processing cost. When a vertex v is assigned to a processor p, the *processing cost* is given by $W_p^v = W^v \times s_p$, the product of the computational weight of v and the processing weight of p. Processing cost reflects time units required by p to process v.

B. *Communication Cost*

If there exists an edge between vertices u and v of the workload graph, the data communication from u to v incurs a *communication cost*. When u and v are assigned to the same processor p, the data set for v is already present in the local memory of p. Thus the communication cost is zero, assuming that local memory access delays are incorporated within the processing weight of a processor. However, when u and v are assigned to different processors, the communication latency between the processors comes into play. Suppose u is assigned to processor p and v is assigned to processor q. Hence, data is transferred from the local memory of p to the local memory of q via an interprocessor message. In this case, the communication cost is given by $C_{p,q}^{u,v} = C^{u,v} \times c_{p,q}$, the product of the communication weight of edge (u, v) and the link weight between p and q. The communication cost reflects time units required for data transfer between the vertices u and v.

This communication model directly reflects distributed memory communication via message passing, but the same mechanism could also be valid for shared memory machines, with the only difference being communication via a global memory. In order to correctly reflect the type of communication (shared or distributed memory), the link and processing weights must be chosen appropriately.

C. *System Potential*

Given a system graph, it is necessary to derive a composite metric to denote the overall potential of the system (Grid). By *system potential*, we mean the computational power of the Grid taken as a whole, a conglomerate of diverse resources solving a single distributed supercomputing application. This metric is required to evaluate the scalability of any partitioning scheme with increases in the potential of the Grid. Given a heterogeneous setting, it is not necessarily true that a system with more processors has a higher system potential than one with fewer processors. System potential is a function of several factors: the number of processors, the individual processor's computational speeds, the connectivity between nodes, and the communication cost matrix. A system with high processor speeds could have a lower system potential than a system with relatively lower processor speeds because the communication cost matrix of the former may be weaker than that of the latter. This means that the communication network plays an important role in determining the system potential of a Grid.

We model system potential relative to a given workload. Given a workload graph $G = (V, E)$, and a system graph $S = (P, L)$, we compute the system potential Φ_{sys} as the sum of individual processor potentials. Thus,

$$\Phi_{sys} = \sum_{p \in P} \phi_p, \tag{1}$$

where the potential ϕ_p of a processor p is defined as

$$\phi_p = 1/(e_p + \delta c_p). \tag{2}$$

Here, e_p is the mean time required by p to compute a vertex. averaged over all vertices of the workload; c_p is the mean time required by p for an inter-vertex communication. averaged over all edges of the workload; and δ is the average degree of the workload graph:

$$e_p = (\sum_{v \in V} W^v/|V|) \times s_p \tag{3}$$

$$c_p = (\sum_{(u,v) \in E} C^{u,v}/|E|) \times \bar{c}_p \tag{4}$$

$$\delta = 2 \times |E|/|V|. \tag{5}$$

The term $\bar{c}_p$ is the mean link weight at p, averaged over all links incident on p, and is given by

$$\bar{c}_p = \sum_{q \in N,\ q \neq p} c_{p,q}/\delta_p, \tag{6}$$

where δ_p is the degree of processor p in the system graph. Thus, ϕ_p is the reciprocal of the average time required by p to execute a vertex of the workload. (It is analogous to the clock rate of a processor which is defined as the inverse of the cycle time.) Therefore, it is reasonable to compute the overall system potential as a sum of individual processor potentials. It is important to mention that we use average values of the parameters for both the workload graph and the system graph to calculate Φ_{sys}. Thus, Φ_{sys} is an approximate metric and applies only in the mean; however, we shall observe its merit in Section VI.

We assume, for modeling purposes, that all inter-vertex communications incur a communication cost that is dependent on the link weights (see Section III-B). This assumption does not weaken our model as it is consistent for all processors in the system and over all system graphs that we have generated. Succinctly, our formulation of Φ_{sys} is an estimate of the absolute power of a heterogeneous system relative to a given workload; it is needed to evaluate the scalability of a partitioning scheme.

As mentioned earlier, system potential is a function of the number of processors ($|P|$), their processing weights (s_p), the link weights ($c_{p,q}$), and network connectivity (λ). We adjust network connectivity by varying λ, the ratio of the number of links in the system graph to the maximum possible number of links: $\lambda = 2|L|/(|P|(|P|-1))$. It also represents the probability of a link between any two processors in the system. For example, $\lambda = 1$ represents a fully connected system graph. Smaller values of λ model a system with weaker connectivity. Figure 3 shows how Φ_{sys} varies with changes in these four parameters. For all experiments, Φ_{sys} is calculated with respect to the dual graph of a real life mesh application.

Figure 3(a) plots Φ_{sys} against each of the four parameters independently. For each case, we vary one of the parameters, keeping the others constant. For example, when varying processing weights, a set of nine graphs are generated with $\lambda = 1$, constant link weights, and a fixed number of processors. Here we allow processing weights to vary randomly in the range 10–100 for each graph in the set. Similarly, when generating system graphs for link weights, we randomly change the link weights and keep the other parameters constant. For the case of processor count, we alter the number of processors from 10 to 100, and keep other parameters constant. However, for the case of network connectivity, we vary λ from 0.1 to 1. In Fig. 3(a), λ is expressed as a percentage ranging from 10 to 100. It is clear from the plots that Φ_{sys} has a positive correlation with the number of processors in the system, provided other parameters are held constant. For network connectivity, we also see a gradual increase in Φ_{sys} with an increase in λ. However, for the other two cases, we see Φ_{sys} decreasing with an increase in the respective weights. It is worth noting that the steepest change in Φ_{sys} occurs with a change in the link weights whereas processing weights affect it the least. This demonstrates the important role of communication cost associated with parallel execution in a distributed environment.

Figure 3(b) shows scenarios where an increase in the number of processors in the system does not guarantee an increase in Φ_{sys}. Here, we generate a set of 20 system graphs with the number of processors varying from

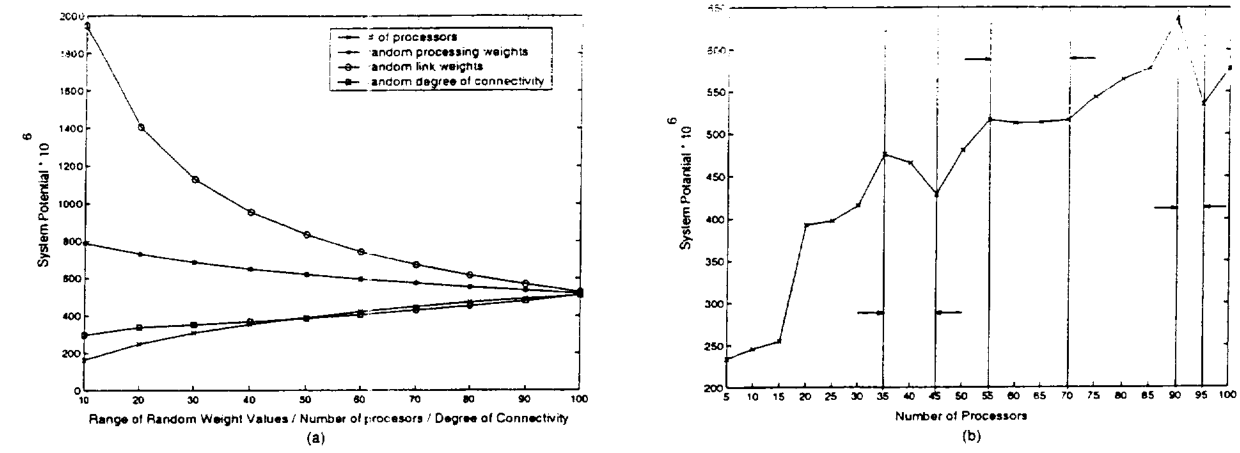


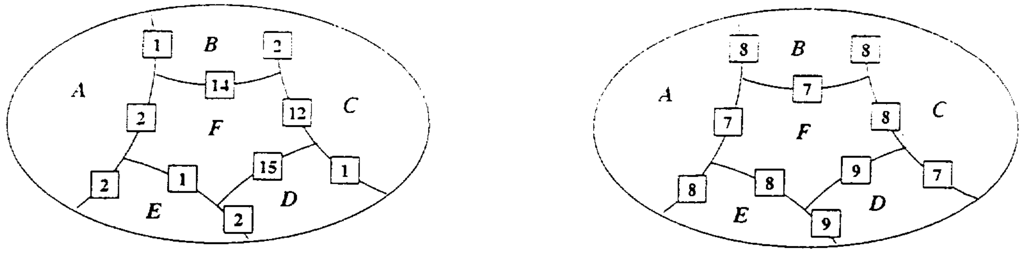
Fig. 3. System potential (function of diverse system characteristics) (a) versus individual parameters, and (b) versus the number of processors while varying all other parameters.

5 to 100. We allow random values for processing and link weights, and a random degree of connectivity for each graph. Observe that Φ_{sys} is not directly related to the number of processors, and that there are instances when it decreases with an increase in the number of processors. These instances are highlighted within pairs of arrows in the figure. For example, an increase in number of processors from 35 to 45 reduces Φ_{sys}. The justification is that several other parameters, as discussed above, collectively determine the potential of a system.

IV. Partitioning Problem

Given a weighted workload graph representation of an application, the standard partitioning approach till date has been to group the vertices of the graph into almost equally weighted sets such that the total *edgecut* across partitions (sets) is minimized. Edgecut is defined as the total weight of the edges in the workload graph whose incident vertices belong to different partitions. A number of software packages such as Metis [16], Chaco [12], and Jostle [23] solve the partitioning problem with this objective. However, when the aim of partitioning is parallel execution of an application on a Grid, these schemes fail to meet the ultimate objective of minimizing the application execution time mainly due to the following three reasons:

- First, standard partitioning schemes do not delve into the mapping problem which determines the assignment of partitions to processors. The drawback of a separate mapping algorithm is that assignment of vertices to partitions cannot be changed in spite of the possibility of reducing the execution time on reassignment. Moreover, there is an additional time complexity associated with the mapping process. Thus, we shall refer to our partitioning problem as the cumulative process of determining load shares by partitioning and the assignment of partitions to processors.
- Second, the standard approach evenly balances workloads across partitions and minimizes the total communication volume by minimizing the edgecut across partitions. Unfortunately, the edgecut metric is flawed [11], because it does not minimize the parallel execution time of the application. The overall performance of a parallel application is governed by the busiest processor. Thus, in order to minimize the parallel execution time of an application, the objective should be to minimize the maximum execution time among the processors constituting the computing environment. So, rather than minimizing the total communication volume, the objective should be to minimize the communication volume of the busiest processor. Figure 4 shows how the execution time of an application can be lower even if edgecut increases. Figure 4(a) shows a 6-way partition of an arbitrary graph among processors A to F. The edgecut representing the communication volume across



Egdecut = 52
Exec. Time (Parallel comm.) = t + 15
Exec. Time (Sequential comm.) = t + 44

Egdecut = 80
Exec. Time (Parallel comm.) = t + 9
Exec. Time (Sequential comm.) = t + 39

Fig. 4. Total edgecuts versus application execution time.

partitions is denoted within a small box at the partition boundaries. The overall edgecut for this partition is 52. Assume all processors have an equal distribution of the computational load denoted by computation time t, and that link weights between processors denote time required for communication. Therefore, the execution time of the application for this partition is $t + 15$, where 15 time units are required for communication by processor F (all communication taking place in parallel). Figure 4(b) shows another partition for the same graph on the same number of processors where the execution time is lowered to $t + 9$ while edgecut has risen to 79. This validates our claim that edgecut is a flawed metric when the objective of partitioning is the efficient parallel execution of an application. We therefore attempt to directly minimize the application execution time in MiniMax. Parallel communication may not be a valid assumption in a truly distributed heterogeneous environment owing to the lack of I/O ports or explicit links between communicating processors. Even for the case of sequential communication, the execution time in Fig. 4(a) is $t + 44$, which is higher than the $t + 39$ time for the partition in Fig. 4(b). In our experiments, we have assumed sequential communication at each processor to calculate the execution time. However, this assumption can easily be relaxed by associating a parameter with each link denoting the possibility of parallel communication. Then, the communication cost of a processor would be the maximum among all communications at that processor. Approaches based on sequential communications to compute the parallel execution time have been considered in [3]. Our contribution is that we attempt to optimize the correct metric, i.e., the execution time of the application.

- Third, existing partitioning schemes do not take into account any heterogeneity in the system graph. They produce partitions with the assumption that all processors are of equal processing capability, the system graph is fully or uniformly connected, and links between processors are of equal weight. With the aim of parallel distributed processing on the Grid, our partitioning scheme eliminates all such assumptions. Edgecut in itself is not a correct measure for the communication cost owing to the heterogeneity in the link weights; each edgecut and the communication link to which it is assigned together determine the cost of communication. However, previous schemes have been successful despite these shortcomings by imposing constraints on the workload graph: a sparse graph representation, uniform distribution of vertex and/or edge weights, and geometric locality [11]. Results of our scheme show success with more general problems where the workload graph is also non-uniform.

Given a workload graph $G = (V, E)$ representation of the application, and a system graph $S = (P, L)$ representation of the Grid, the objective is to assign vertices of the workload graph to partitions, where a partition is mapped to a vertex of the system graph, such that the execution time of the application is minimized. Clearly, the problem is extensible to the classical graph partitioning and task assignment problem [6], and is thus NP-complete. The generalization of this problem by considering heterogeneous workload (task) and system graphs makes the problem more complex. Given a partitioning of G over S, the execution time of the application is the maximum over the execution time of individual processors:

$$ET = \max_{p}\{ET_p\}, \quad p \in P \tag{7}$$

where ET_p is the execution time of processor p and is given by

$$ET_p = \sum_{v\mathcal{A}p} W^v \times s_p + \sum_{v\mathcal{A}p} \sum_{u\mathcal{A}q,\ p\neq q} C^{u,v} \times c_{p,q}. \tag{8}$$

Here, $v\mathcal{A}p$ denotes an assignment of vertex v to processor p. Note that ET_p is the sum of the processing and communication costs as discussed in Sections III-A and III-B, respectively. The first term in Eq. 8 denotes the processing cost associated with all v assigned to p, and the second term denotes the communication cost associated with all neighbors of v not assigned to p.

V. MiniMax: A Multilevel Heterogeneous Partitioner

MiniMax is a graph/mesh partitioning scheme developed for distributed heterogeneous systems such as the Grid. It differs from existing partitioning schemes in that it takes into consideration heterogeneity in both the system and workload graphs, and produces high quality partitions for all types of parallel Grid applications. We employ a multilevel strategy where the given graph is first coarsened through several levels to a smaller number of vertices. Next, the coarsest graph is partitioned among processors with the objective of minimizing the application execution time. However, the best partition for a coarsened graph may not be optimal for the original graph. Hence, while the coarse graph is refined back to the original, an optimization scheme is used at each level to reduce application execution time. At each refinement level, we consider the possible migration of vertices to other processors which could minimize the maximum execution time among all processors. In the following, we describe each step of our scheme in more detail.

A. Graph Coarsening

Given a graph $G_i = (V_i, E_i)$ at level i, we generate a coarsened graph $G_{i+1} = (V_{i+1}, E_{i+1})$ for level $i+1$ by finding a maximal matching of G_i and collapsing edges in this matching. Maximal matching of a graph is the set of edges $E_m \subset E_i$ such that no two edges in E_m are incident on a common vertex, and that any edge not in the maximal matching has at least one of its endpoints matched [16]. Vertices u_1 and u_2 at the endpoints of matched edges in E_m are merged to form a new vertex $v \in V_{i+1}$ with weight $W^v = W^{u_1} + W^{u_2}$. Vertices in V_i that are not endpoints of the matching are copied over to V_{i+1}. If merged vertices are adjacent to a common vertex, a new edge replaces the merged edges with weight equal to the sum of the edge weights it replaces. A number of procedures to determine the maximal matching has been discussed in [16], e.g., random edge, heavy edge, light edge, and heavy clique. After exploring these procedures for developing a good matching scheme for MiniMax, we found that a variant of the heavy clique matching worked the best. We call this *low degree matching*. We generate a random list of vertices in G_i and visit each vertex on this list, matching an unmatched vertex with an unmatched neighbor. To arbitrate between several unmatched neighbors, we choose the neighbor with the minimum degree. When a vertex is matched, it is marked to eliminate it from further consideration during subsequent matchings for other vertices. This coarsening procedure requires time proportional to the number of edges in the finer graph, $O(|E_i|)$. The motivation behind low degree matching is that the coarsest graph will have a high degree of connectivity and hence give more options for our initial partitioning to obtain a low execution time. Also, since we use a graph growing algorithm for the initial partition, a higher degree of connectivity ensures more directions for growth. In addition, our refinement/optimization step at each level also benefits from a graph with a large number of edges.

B. Initial Partition

The graph coarsening process is terminated when the number of vertices in the coarsest graph falls below a given threshold. An initial partition of the coarsest graph is then made among the processors of the system graph with the objective of minimizing the application execution time. With $|P|$ vertices in the system graph, we employ a $|P|$-way graph growing algorithm to assign each vertex in V to one of the vertices in P. To begin, each processor p is assigned a vertex from V and a region is grown around it for all processors with the objective of minimizing the maximum execution time. Each region continues to grow until all vertices in V have been assigned to one of the processors in P. The choice of initial vertices to start the growing procedure affects the quality of the partitioning. We choose a list L of $|P|$ vertices from V with the highest degrees as the one to begin the growing procedure. The motivation behind this approach is that higher the degree, the greater will be the directions for partition growth at a processor. This approach for choosing initial vertices produces fairly good partitions. A vertex from L is assigned to a processor in accordance with the processor's processing weight. We employ a linear time bucket sort to order processors with non-increasing processing weight. Similarly, we order vertices in L with non-decreasing computational weight, and assign the i-th element of L to the i-th processor. Once each processor has a vertex assigned to it, it has an associated execution time (see Eq. 8). A region could be grown around each assigned vertex in a breadth first fashion as in [7]. However, we employ a greedy approach whereby the region corresponding to a processor is allowed to grow only along a neighboring vertex w that leads to a minimum increase in the execution time of the processor. The gain in the execution time of processor p on assignment of an unassigned vertex v is given by

$$Gain_p^v = W^v \times s_p + \sum_{u\mathcal{A}q,\ \forall q \in P,\ q \neq p} C^{u,v} \times c_{p,q}. \tag{9}$$

Thus, the increased execution time on assignment of v to p is $ET_p^v = ET_p{}' + Gain_p^v$, where $ET_p{}'$ is the execution time prior to the assignment. Each region has a candidate vertex w among all its neighbors, the growth along which will lead to the minimum increase in execution time at the processor. Hence, $w = \{v \mid \min_v\{ET_p^v\},\ v \in N(R_p)\}$, where $N(R_p)$ represents the set of vertices that are neighbors to the region corresponding to processor p. Over all w's associated with each region, only one region is allowed to grow in each iteration. A region is selected for growth if its candidate vertex w on inclusion into the region results in the lowest execution time over all processors. In total, $|V_C| - |P|$ iterations of individual growing are required, where $|V_C|$ is the number of vertices in the coarsest graph. To speed up the graph growing procedure, we use a min-heap to store vertices on the frontier of each region, defined to be the set of vertices on the boundary. Each element of the heap has three fields, namely the vertex on the frontier, the region to which it could be assigned, and the corresponding execution time of the application if the vertex is assigned to that region. The min-heap is keyed on the application execution time. Thus, the vertex with the minimum application execution time is inserted first into its corresponding region. When a vertex is inserted, the adjacent vertices already in the heap are updated to reflect the new application execution time and those vertices not in the heap are inserted. A table of pointers to heap elements is also maintained to allow constant time access. The motivation behind this graph growing strategy is that it attempts to minimize the application execution time as well as the variance of individual processor execution times. The graph growing algorithm only requires $O(|V_C|)$ time since it is performed on the coarsest graph.

C. Refinement

The coarsened graph is projected back to the original through several levels of refinement. At each refinement level, the partition for a coarsened graph is mapped to a partition for the finer graph, by assigning vertices u and v that were merged to produce vertex w to the processor to which w was assigned at the coarser level. However, a good initial partition does not guarantee a high quality partition for the original graph. Hence, at each refinement level, the application execution time is optimized by checking for fruitful vertex migrations across partitions. Finer granularity and a high degree of connectivity (which was our objective in the coarsening step) gives greater scope for optimization. Given a partition, with processor m having the maximum execution time $MaxET$, we attempt to reduce it using three types of vertex migration as discussed below. It is important to note that no other vertex migrations apart from those discussed below can reduce the execution time of m. A vertex migration that reduces the execution time of m should not simultaneously increase the execution time of other processors in the system beyond $MaxET$.

- *Migrate vertex from m:* In this migration class, we consider only the border vertices in the partition for m because empirical results show that almost all fruitful vertex migrations are along a partition border. Each border vertex of m is checked for migration to every other processor in the system. We compute $TentMaxET_p^v$, the tentative maximum execution time that would result if a border vertex v is migrated to processor p. If $TentMaxET < MaxET$, the migration is fruitful. Note that several border vertices could be candidates for a fruitful migration. We choose the vertex with the lowest value of $TentMaxET$ as the successful candidate for migration.
- *Migrate vertex to m:* Owing to the diversity in weights of the edges in both the system and workload graphs, migration of a vertex to the processor with $MaxET$ could also reduce the execution time. This situation arises when the processing cost associated with a vertex being migrated is very low compared to its communication cost. However, in this class, migration of only the neighbors of the border vertices of m that are assigned to other processors can lead to a reduction in $MaxET$. We compute $TentMaxET$ for each possible vertex migration in this fashion and choose the one causing the largest decrease in execution time.
- *Migrate vertex from p to q ($p \neq q \neq m$):* This class deals with vertex migration among processors other than m. It is possible to reduce $MaxET$ by migrating vertices among processors not involving m because of the heterogeneity in link and processing weights of a system graph. However, the vertex being migrated from p to q must have an edge into m for the migration be to fruitful. Hence we consider only a subset of P for possible choices of p and q. As in the other two cases above, we compute $TentMaxET$ for each possible migration and select the one with the lowest value.

At each refinement level, the above-mentioned migrations are carried out iteratively. During each iteration, each of the three classes produce their best candidate vertices for fruitful migration. Among the three, the one for which $TentMaxET$ is minimum is ultimately committed for migration. On commitment, we update the processor assignment for vertices, and the values for $MaxET$ and m. Our iterative refinement scheme continues in this fashion until none of the three classes produces a candidate vertex.

Unfortunately, this could denote a situation where the partitioning process has reached a local minima and a sub-optimal solution could be obtained by stopping the refinement step. We therefore have a mechanism for climbing out of local minima. Basically, when a minima is achieved, a non-fruitful vertex is still allowed to make a false migration with the expectation that future fruitful migrations will compensate positively for the incurred rise in execution time. We record a sequence of false migrations and if we encounter a cumulative overall decrease in execution time, we commit these migrations. Experimental results with our test cases have shown that a sequence of 10 false migrations helps MiniMax avoid local minima in most cases and produces good results. With false migrations, the vertex-processor combination is recorded so that the same vertex cannot be re-migrated to the same processor in two consecutive migrations. This avoids toggling vertices between processors during the false sequence of migrations.

VI. Experimental Study

Experiments to evaluate our MiniMax partitioner were performed on a wide range of workloads ranging from realistic size computational meshes used for aerodynamics simulations at NASA to synthetic workloads. The NASA test case is a dual graph representation of a finite element mesh whereas the synthetic workload graphs are more general in the sense that vertex and edge weights as well as the degree of connectivity can be chosen arbitrarily. Partitioning quality is measured using the following three metrics.

- *Execution Time of the application (ET):* This is a direct measure of any partitioning or assignment solution and reflects the total time taken for the application to complete. Given a partition, the execution time of individual processors is calculated according to Eq. 8, and the maximum value gives the execution time (*ET*) of the application. Our objective is to minimize this metric.

- *Standard Deviation (σ):* An alternative way to evaluate a partitioning scheme is to determine the deviation of each processor's execution time from the average execution time among all processors. This reflects the variance among each processor's load assignment:

$$\sigma = \sqrt{\sum_{p \in P} (ET_p - ET_{av})^2/|P|} \tag{10}$$

where the average execution time ET_{av} is given by

$$ET_{av} = \sum_{p \in P} ET_p/|P|. \tag{11}$$

A low value for σ is desired.

- *Imbalance Factor (Imb):* Load imbalance is defined as the ratio of the maximum execution time to the average execution time among processors. It is defined as

$$Imb = MaxET \ / \ ET_{av}. \tag{12}$$

Clearly, the best possible value for *Imb* is unity.

A. *Performance Results*

Table I shows results when MiniMax was run for the sparse dual graph of a mesh application from NASA. Vertex and edge weights for this graph range uniformly between 1 and 4; the degree of vertices also lie in the range 1–4. Here, edges are highly localized to a vertex. However, we have scaled the vertex and edge weights by constant values 2500 and 10, respectively, giving a larger computation-to-communication ratio. The uniformity of the weights and the sparseness of the graph still exists. Furthermore, we have generated five sets of system graphs with ten graphs in each set, varying number of processors $|P|$ from 10 to 100. Each set of graphs represents a different level of heterogeneity in the system. In the table, *Homo* represents a homogeneous Grid with equal processing weights and link weights, and a fully connected topology. In *PHetero*, we vary the processing weights randomly in the range 10–100, reflecting processor heterogeneity. In *LHetero*, only the link weights are varied randomly in the range 1–10. In *CHetero*, we allow a random connectivity between vertices while keeping other parameters constant. Finally, in *FullHetero*, we randomly vary all four parameters of the system graph.

Tables II and III show results for synthetic workloads. During the generation of these workload graphs, we considered the computation-to-communication ratio at a vertex. Generally, the computational requirements of an application targeted for parallel distributed execution on the Grid are significantly larger than the communication requirements, otherwise the benefits of Grid computing would be limited. Also, in order to incorporate heterogeneity in the workload graph, the vertex and edge weights vary randomly. Therefore, we have kept the computation-to-communication ratio diverse, ranging from 1000:1 to 2:1. We have chosen the range of random values for vertex and edge weights, and the degree of connectivity of the workload graph in such a way that this ratio falls within the desired range. Table II shows results for a workload graph where vertex and edge weights are random, but with a uniform distribution. Table III shows results when weights are non-uniformly distributed. We have run our partitioner for a number of non-uniform distributions, but the results in Table III are for exponential random distribution. Good partitioning results in all cases proves the usefulness of our partitioner for a wide variety of application types.

All tables record application execution time (ET) and standard deviation (σ), when MiniMax is run for each of the five set of system graphs. The imbalance factor (Imb) is not shown as it is consistently equal to unity for all runs of MiniMax, imbalance being recorded up to two decimal places of precision. We also

TABLE I

RESULTS FOR NASA TEST MESH (50000 VERTICES AND 191600 EDGES)

System	Metric	10	20	30	40	50	60	70	80	90	100
Homo	Φ_{sys}	177.83	352.80	525.29	695.29	862.87	1028.08	1191.01	1351.71	1510.23	1666.65
	ET	279.34	139.70	93.13	69.86	55.88	46.58	39.93	34.95	31.05	27.95
	σ	7719	9222	2708	9271	8635	11155	11035	7322	1491	6856
PHetero	Φ_{sys}	46.06	103.94	110.15	154.48	245.51	293.16	348.40	387.50	396.54	413.95
	ET	714.21	477.91	450.95	320.50	199.63	167.54	139.94	126.35	122.79	117.87
	σ	54235	58645	68891	39588	51322	47854	50852	56372	60010	55734
LHetero	Φ_{sys}	171.33	331.04	481.82	617.80	748.29	872.98	985.88	1093.80	1197.83	1298.46
	ET	297.77	139.90	93.27	69.98	55.98	46.66	40.00	35.01	31.12	28.01
	σ	5894	7370	7362	7591	7241	7691	7125	6837	7284	7854
CHetero	Φ_{sys}	177.20	349.75	519.84	685.35	847.14	1006.29	1163.37	1315.50	1464.71	1614.95
	ET	279.36	139.72	93.15	69.87	55.90	46.59	39.93	34.95	31.06	27.95
	σ	7094	7239	5668	8113	7900	10241	10283	6924	2235	6494
FullHetero	Φ_{sys}	70.97	79.22	133.50	206.17	222.18	227.13	303.91	345.81	373.78	400.34
	ET	687.77	613.81	361.85	230.00	208.66	205.61	150.12	131.06	119.46	110.50
	σ	55084	50030	45976	55155	56603	58015	55618	57645	54783	49256

TABLE II

RESULTS FOR UNIFORM SYNTHETIC WORKLOAD (5000 VERTICES AND 199892 EDGES)

System	Metric	10	20	30	40	50	60	70	80	90	100
Homo	Φ_{sys}	113.20	223.64	331.46	436.78	539.72	640.37	738.85	835.24	929.63	1022.10
	ET	43.73	21.87	14.58	10.94	8.75	7.29	6.26	5.47	4.87	4.38
	σ	4513	5001	4729	4267	5219	5328	7039	5460	6224	4960
PHetero	Φ_{sys}	29.41	66.23	70.19	98.24	155.12	185.46	219.45	244.67	249.58	260.83
	ET	169.16	74.74	70.51	50.12	31.23	26.21	21.89	19.77	19.22	18.45
	σ	11409	23514	28440	32285	27334	26283	27939	28494	40186	40131
LHetero	Φ_{sys}	106.69	202.20	289.84	364.75	435.65	502.51	560.19	614.85	667.13	717.24
	ET	44.19	22.08	14.72	11.05	8.84	7.37	6.32	5.53	4.91	4.42
	σ	5784	3938	4403	3106	3774	4322	4260	3948	4312	4023
CHetero	Φ_{sys}	112.65	220.51	325.95	426.88	524.26	619.24	712.36	800.98	887.02	974.20
	ET	43.76	21.90	14.60	10.95	8.76	7.30	6.26	5.48	4.87	4.38
	σ	4402	4136	3559	4754	4107	5008	3530	4098	4856	5191
FullHetero	Φ_{sys}	44.70	49.81	83.58	127.88	136.27	139.94	184.65	209.67	224.72	240.13
	ET	108.19	96.23	56.77	36.08	32.73	32.23	23.54	20.55	18.73	17.33
	σ	9668	28107	31516	19443	28829	24952	22852	25752	24547	27786

TABLE III

RESULTS FOR NON-UNIFORM SYNTHETIC WORKLOAD (5000 VERTICES AND 199892 EDGES)

System	Metric	10	20	30	40	50	60	70	80	90	100
Homo	Φ_{sys}	360.53	723.44	1059.08	1397.75	1729.72	2055.24	2374.58	2687.95	2995.57	3297.66
	ET	13.75	6.88	4.59	3.44	2.75	2.30	1.97	1.73	1.53	1.38
	σ	5030	4420	4699	4718	3858	4742	4981	4950	4651	4855
PHetero	Φ_{sys}	93.58	210.89	223.50	313.05	495.41	592.05	710.60	781.58	798.41	833.80
	ET	53.21	23.51	22.19	15.77	9.83	8.25	6.89	6.22	6.05	5.81
	σ	30340	24754	27052	23347	26995	22553	21329	27541	24756	29909
LHetero	Φ_{sys}	342.46	653.27	941.13	1191.64	1429.68	1654.99	1852.51	2040.08	2219.82	2392.58
	ET	13.87	6.93	4.62	3.47	2.78	2.31	1.98	1.74	1.55	1.39
	σ	4588	3811	2473	3580	3180	2768	4204	3441	3776	3047
CHetero	Φ_{sys}	359.01	704.78	1043.74	1370.05	1686.28	1995.59	2291.51	2590.44	2873.86	3160.40
	ET	13.76	6.39	4.59	3.44	2.76	2.30	1.97	1.72	1.53	1.38
	σ	3754	3779	3572	3187	4269	3670	3823	3590	3384	3710
FullHetero	Φ_{sys}	142.93	159.36	267.78	410.96	439.56	450.84	597.51	678.88	729.70	780.23
	ET	33.99	30.27	17.85	11.35	10.29	10.14	7.41	6.46	5.89	5.45
	σ	9123	29408	17231	24606	20934	22376	20981	20910	18034	22528

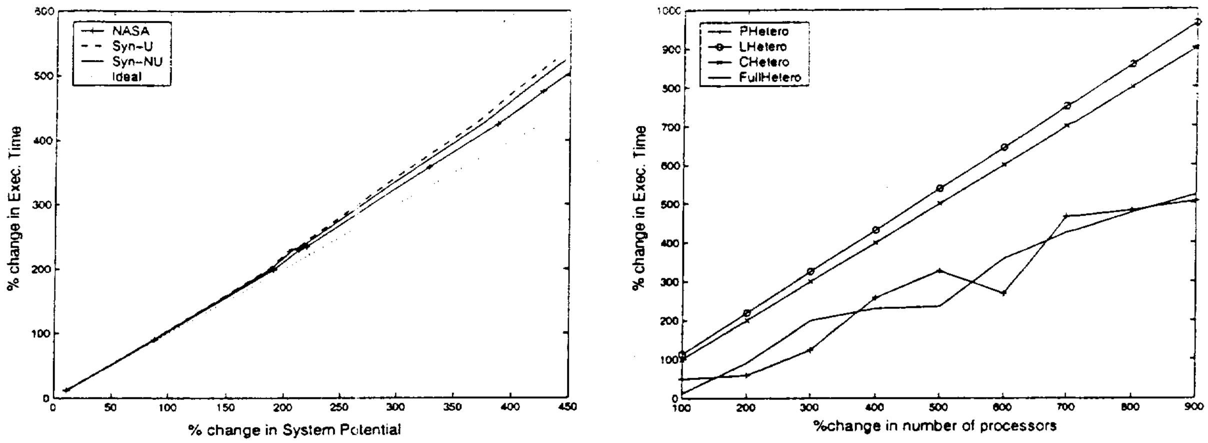
record the system potential (Φ_{sys}) corresponding to each system graph. The choice of vertex and edge weights for system and workload graphs determines the execution time of individual processors. Therefore, the results generated by our experiments are dependent on the range of random values assigned to the vertex and edge weights as discussed above. We need to assume a unit of time that is consistent throughout our experiments. The results from our experiments yield very large values and thus, without the loss of generality, we have assumed the smallest unit of time to be microseconds. The execution times are reported in seconds, whereas the standard deviation is shown in microseconds. The imbalance factor is a dimensionless quantity, while system potential is expressed as the inverse of time in seconds. It is apparent from the results that MiniMax produces very high quality partitions. We observe that σ is orders of magnitude lower than ET for all the experiments we have performed. These imply that all processors have approximately the same ET and hence a near perfect Imb is obtained.

We also observe that ET is a function of Φ_{sys}, as described in Section III-C, and not strictly dependent on the number of processors in the system. The reduction in ET more closely follows an increase in Φ_{sys} rather than an increase in the number of processors. This is clear from the ET entries in the tables for systems having full heterogeneity. Figure 5(a) shows the percentage change in ET against percentage changes in Φ_{sys} for three different workload graphs and a fully heterogeneous system graph. Notice that there is a linear relationship between changes in ET and changes in Φ_{sys}. For all three cases shown, the percentage change in ET is nearly equal to the percentage change in Φ_{sys}. The dotted line represents the ideal situation where the two changes are equal. These plots have been generated from data in Tables I–III. The changes in ET and Φ_{sys} are calculated with respect to the first entry of each of the two parameters.

Figure 5(b) shows percentage changes in ET against percentage changes in the number of processors for the NASA test mesh for each type of system graph. Here, we observe that there is no direct correlation between the two changes. For the case of fully-heterogeneous and processor-heterogeneous systems, this observation is obvious. The same claim can be made for link-heterogeneous and connectivity-heterogeneous systems, but with a lesser emphasis for our test cases. The reason is that our workload graphs are compute-intensive owing to our choice of keeping the computational weights much higher than the communication weights. This causes the change in Φ_{sys} to be directly proportional to the change in number of processors if processing weights are kept constant. As Φ_{sys} is measured with respect to a workload graph (see Section III-C), the low values of communication weights in our test cases have their effect subdued by much larger computational weights. When communication weights were given larger values, similar lack of correlation has been observed. These experimental results validate our claim in Section III-C that Φ_{sys} is a more powerful metric along which a partitioner for heterogeneous systems can be measured for scalability.

Our experimental results are dependent on the random weights assigned to the workload and system graphs. However, we can draw insightful conclusions from these results. For a compute intensive workload, ET is largely dependent on processor heterogeneity and to a lesser extent on link heterogeneity. For example, in Table III, when 20 processors are used, the execution times for each of the five types of system graphs (*Homo, PHetero, LHetero, CHetero, and FullHetero*) are 6.88, 23.51, 6.93, 6.89, and 30.27 respectively. Thus, each additional degree of heterogeneity causes an increase in ET over that of *Homo* by factors of 3.42, 1.01, 1.00, and 4.40, respectively. Processor heterogeneity and full heterogeneity cause the largest change in ET. The same pattern holds for both the other tables and for other processor counts. If the workload graph is communication intensive, link and connectivity heterogeneity would show greater changes in execution times.

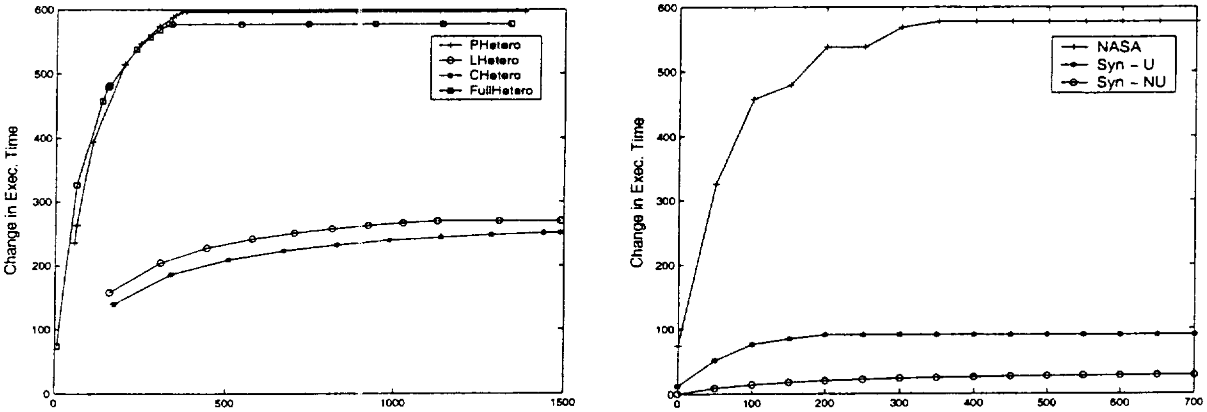
We can also compare the improvement in ET against changes in Φ_{sys}. For our test applications, there is a dramatic improvement in ET for an initial increase in Φ_{sys} as seen in Figure 6. Figure 6(a) plots absolute change in ET for a corresponding change in Φ_{sys} for the NASA test application for each of the four types of system graphs. Gradually, a state of saturation is reached where further increases in Φ_{sys} causes little improvement in ET. This indicates that for a given application, there would be very little gain by employing



(a) FullyHetero System Graph

(b) NASA Benchmark Mesh

Fig. 5. Execution time is closely dependent on changes in the system potential.



(a) NASA Benchmark Mesh

(b) FullyHetero System Graph

Fig. 6. Change in execution time versus change in system potential.

more processors once a saturation point is achieved. The steep initial increase for all types of system graphs confirms that our partitioner attains this saturation point very quickly. Figure 6(b) shows changes in ET against changes in Φ_{sys} for three different workload graphs when the system graph is fully heterogeneous. Similar observations can be made here. Again, the data for these figures were obtained from Tables I–III.

VII. Conclusion

In this paper, we have presented a novel graph partitioner, called *MiniMax*, for partitioning heterogeneous workload graphs onto heterogeneous system graphs. The major contributions are optimizations based on the application execution time, which is a direct measure of the quality of the partitioning and processor assignment, as opposed to indirect objectives used by traditional graph partitioners based on edgecuts and subsequent assignment of partitions to processors. We have considered complete heterogeneity in the system by allowing randomly varying weights for processors and links, and assumed non-uniform connectivity between processors. To encompass a variety of applications that could be modeled as a graph for parallel distributed computing on a Grid of resources, we have also allowed heterogeneity in the workload graph. A composite metric, called system potential, that reflects the overall power of a Grid is developed to measure the scalability of our partitioning scheme. Direct comparisons of MiniMax with previous partitioning schemes is not justified as our work is innovative with respect to the generality of the problem it covers. However, future work would consist of pruning down the capability of MiniMax to the limited domain of problems that standard partitioning schemes cover and do a comparative study.

REFERENCES

[1] S. Barnard, R. Biswas, S. Saini, R. Van der Wijngaart, M. Yarrow, L. Zechtzer, I. Foster, and O. Larsson, "Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus," *7th Symp. on the Frontiers of Massively Parallel Computation*, Annapolis, MD, Feb 1999, pp. 60–67.

[2] K. L. Bibb, J. Peraire, and C. J. Riley, "Hypersonic Flow Computations on Unstructured Meshes," *35th AIAA Aerospace Sciences Meeting & Exhibit*, Reno, NV, Jan 1997, AIAA Paper 97-0625.

[3] S. K. Das, D. J. Harvey, and R. Biswas, "A Latency-Tolerant Partitioner for Distributed Computing on the Information Power Grid," *Intl. Parallel and Distributed Processing Symp.*, San Francisco, CA, April 2001.

[4] M. J. Djomehri, R. Biswas, R. F. Van der Wijngaart, and M. Yarrow, "Parallel and Distributed Computational Fluid Dynamics: Experimental Results and Challenges," *HiPC 2000 High Performance Computing: Proc. 7th Intl. Conf.*, Bangalore, India, Dec 2000, Springer-Verlag LNCS Vol. 1970, pp. 183–193.

[5] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufman, San Francisco, CA, 1999.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractibility*, Freeman and Company, New York, NY, 1979.

[7] T. Geohring and Y. Saad, "Heuristic Algorithms for Automatic Graph Partitioning," Technical Report, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN, 1994.

[8] G. H. Golub and C. F. Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, MD, 1996.

[9] E. Haddad, "Optimal Distribution of Random Workloads over Heterogeneous Processors with Loading Constraints," *Intl. Conf. on Parallel Processing*, St. Charles, IL, Aug 1992.

[10] B. Hendrickson, "Graph Partitioning and Parallel Solvers: Has the Emperor no Clothes?" *Solving Irregularly Structured Problems in Parallel: Proc. 5th Intl. Symp.*, Berkeley, CA, Aug 1998, Springer Verlag LNCS Vol. 1457, pp. 218–225.

[11] B. Hendrickson and T. G. Kolda, "Graph Partitioning Models for Parallel Computing," *Parallel Computing*, Vol. 26, Nov 2000, pp. 1519–1534.

[12] B. Hendrickson and R. Leland, "The Chaco User's Guide, version 2.0," Technical Report SAND95-2344, Sandia National Laboratory, Albuquerque, NM, 1995.

[13] B. Indukhya, H. S. Stone, and L. Xi-Cheng, "Optimal Partitioning of Randomly Generated Distributed Programs," *IEEE Trans. Software Engineering* Vol. SE-12, March 1986.

[14] W. E. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid," *8th Intl. Symp. on High Performance Distributed Computing*, Redondo Beach, CA, Aug 1999.

[15] W. Jone and C. A. Papachristou, "A Coordinated Circuit Partitioning and Test Generation Method for Pseudo-Exhaustive Testing of VLSI Circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, March 1995, pp. 374–384.

[16] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," Technical Report 95-035, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN, 1995.

[17] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home: Massively Distributed Computing for SETI," *Computing in Science and Engineering*, Vol. 3, Feb 2001.

[18] D. J. Mavriplis and S. Pirzadeh, "Large-Scale Parallel Unstructured Mesh Computations for 3D High-Lift Analysis," *37th AIAA Aerospace Sciences Meeting & Exhibit*, Reno, NV, Jan 1999, AIAA Paper 99-0537.

[19] L. Oliker and R. Biswas, "PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes," *Journal of Parallel and Distributed Computing*, Vol. 52, Aug 1998, pp. 150–177.

[20] T. Schnekenburger and M. Huber, "Heterogeneous Partitioning in a Workstation Network," *Workshop on Heterogeneous Computing*, Mexico, April 1994, pp. 72–77.

[21] R. C. Strawn, R. Biswas, and M. Garceau, "Unstructured Adaptive Mesh Computations of Rotorcraft High-Speed Impulsive Noise," *J. Aircraft*, Vol. 32, July/Aug 1995, pp. 754–760.

[22] J. F. Thompson, B. Soni, and N. P. Weatherill, *Handbook of Grid Generation*, CRC Press, Boca Raton, FL, 1999.

[23] C. Walshaw and M. Cross, "Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm," *SIAM J. Scientific Computing*, Vol. 22, Jan 2000, pp. 63–80.

[24] C. Walshaw and M. Cross, "Multilevel Mesh Partitioning for Heterogeneous Communication Networks," Technical Report 00/IM/57, School of Computing and Mathematical Sciences, Univ. of Greenwich, London, UK, 2000.