A Heuristic Algorithm for Mapping Parallel Applications on Computational Grids

Panu Phinjaroenphan[1], Savitri Bevinakoppa[1], and Panlop Zeephongsekul[2]

[1] School of Computer Science and Information Technology,
[2] School of Mathematical and Geospatial Sciences,
RMIT University, GPO Box 2476V, Melbourne Australia
{pphinjar, savitri}@cs.rmit.edu.au, panlopz@rmit.edu.au

Abstract. The mapping problem has been studied extensively. However, algorithms which were designed to map a parallel application on a computational grid, such as MiniMax, FastMap and genetic algorithms have shortcomings. In this paper, a new algorithm, Quick-quality Map (QM), is presented. Experimental results show that QM performs better than the other algorithms. For instance, QM can map a 10000-task parallel application on a testbed of 2992 nodes in 6.35 seconds, and gives the lowest execution time whereas MiniMax and a genetic algorithm, respectively, take approximately 1700 and 660 seconds, but produce 1.34 and 6.60 times greater execution times than QM's.

1 Introduction

Computational grid has been introduced as a distributed computing paradigm that is able to interconnect heterogeneous networks and a large number of nodes regardless of their geographical locations [1]. This paradigm provides an access to tremendous computational power that can be harnessed for various applications. Parallel applications are developed to solve implementations of computational intensive engineering or scientific problems that require such power.

The main aim of solving such problems with a parallel application is to reduce the execution time. As a computational grid involves a large number of nodes, one of the challenging problems that needs to be addressed is to decide the destination nodes where the tasks of the application are to be executed. This process is formally known as the *mapping problem* [2].

Unfortunately, the mapping problem is known to be a non-deterministic polynomially bounded (NP) complete problem [3], which means that the problem is intractable and very time consuming. Hence, heuristic algorithms have been employed to solve the mapping problem. Two of these algorithms are MiniMax [4] and FastMap [5] which have the same scope as the heuristic algorithm adopted in this paper. Genetic algorithms (GAs) are another approach that can be applied to this problem. However, those algorithms have shortcomings, as these will be discussed in the next section.

In this paper, a new mapping algorithm, Quick-quality Map (QM), is presented. Experimental results from the evaluation of QM compared with MiniMax

[4] and a genetic algorithm show that QM performs better than the other algorithms. For example, when mapping a 10000-task parallel application on a testbed of 2992 nodes, QM gives the best solution. The mapping time of QM is 6.35 seconds whereas MiniMax and GA take about 1700 and 660 seconds, respectively.

2 Background and Related Work

In the literature, the mapping problem has been studied extensively. Researchers often focus on the specific models of parallel applications and parallel systems, and concentrate on optimising a particular metric. These three features then are used to differentiate between the studies of the mapping problem.

A parallel application is usually modelled by a graph. Task Interaction Graph (TIG) and Task Precedence Graph (TPG) – also known as Directed Acyclic Graph (DAG), are the traditionally tools used. A DAG is used to model a parallel application that the tasks have order of executions whereas a TIG a parallel application that the tasks are simultaneously executable [6]. In both models, there can be computational costs associated with the tasks (vertices), and communication costs with the communications (edges) between the tasks.

In this paper, the parallel systems are broadly categorised into *modern* and *legacy* systems. An example of a legacy system is the Massively Parallel Processors (MPP) node. Such node often consists of many processors. It is not uncommon to assume that users can assign a particular task to a particular processor. The network topology that links the processors together is static, such as torus and hypercube. Examples of modern systems are a cluster and a computational grid. Nodes and networks are two major resources. In general, users can only assign a task to a node, and the operating system takes care of which processor will execute the task if that node has more than one processor. The specific network topology is not assumed. A graph is also used to model a parallel system with computational costs associated with the processors/nodes (vertices), and communication costs with the links (edges) between the processors/nodes.

Two major optimised metrics are *communication* and *execution times* (or *costs*). The choice of which metric to optimise depends on the assumptions of the application and system models. The model of the application studied in this paper is a TIG and its associated costs are heterogeneous. The parallel system is a computational grid, and the costs associated with the nodes and networks are also heterogeneous. The metric to be optimised is the execution time of the application. In the literature, MiniMax [4] and FastMap [5] are the two algorithms which have the same scope as the process considered in this paper.

MiniMax is a suit of heuristic algorithms [4] consisting of three steps: *graph coarsening*, *initial partition* and *refinement*.

In the first step, the application graph is coarsened until the number of tasks falls below a predefined threshold. Coarsening is an approach of producing a new application graph with less number of tasks by merging a task with one of its neighbours to form a new task (see Fig.1 for an example).

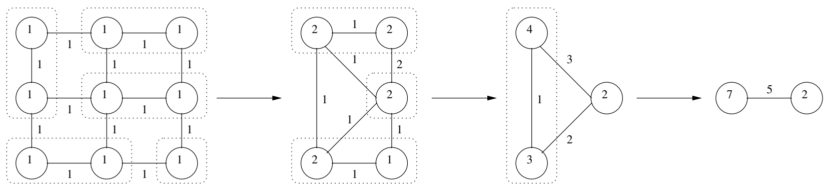

Fig. 1. An example of graph coarsening. After coarsening the application graph three times the number of tasks is reduced from 9 (finest) to 2 (coarsest)

In the second step, the coarsest application graph is mapped on the system with the *graph growing algorithm*. The algorithm maps high cost tasks to low cost nodes. The limitation is the algorithm can function only when the number of nodes is less than the number of tasks. Otherwise, a form of node selection is needed. This raises the issue of how to select the nodes since it is necessary to map tasks on nodes before judging whether the nodes should be chosen.

In the final step, the application graph is un-coarsened to produce the finer graph. During each un-coarsening to the finest graph, the execution time of the application is iteratively optimised with the *vertex migration algorithm*. The performance of the algorithm is the issue in this step as the complexity grows polynomially with both the numbers of tasks and nodes [7].

FastMap is also a suit of heuristic algorithms [5]. The optimisation is a genetic algorithm. A serious problem with FastMap is the assumption that all clusters have the same number of nodes. This is often not the case in real environments.

Genetic Algorithms (GAs) are a well-known optimisation technique. Braun et at. [8] studied the efficiencies of eleven algorithms by mapping independent tasks (zero on all communication costs) on heterogeneous parallel systems. A GA is among those algorithms and has shown to be one of the most efficient. Nevertheless, the high numbers of tasks and nodes can result in a massive search space. Thus, the computational cost of applying GAs could be prohibitively expensive, which significantly reduces their merit.

3 The Mapping Models

This section explains the models used to formulate the mapping problem.

3.1 The Parallel Application Model

An application is modelled as a weighted undirected graph $G = (V, E, W_V, W_E)$, where V is a finite set of vertices, and E a finite set of edges. An edge $e \in E$ is an unordered pair (v_x, v_y), where $v_x, v_y \in V$. V represents the tasks of application G, $|V|$ is the number of tasks, $W_V(v)$ the computational cost of task v, $e_{v_x v_y}$ represents the communication between tasks v_x and v_y (i.e. v_x and v_y

are neighbours), and $W_E(e_{v_x v_y})$ is the communication cost between tasks v_x and v_y. This model is the same as the ones used in MiniMax [4] and FastMap [5].

3.2 The Computational Grid Model

A computational grid is modelled by a *three-level-tree*. The levels are grid (g), cluster (c) and node (n) levels. Let $G' = (V', E', W'_V, W'_E)$ be a three-level-tree representing a computational grid. V' represents the nodes in G', $|C'|$ and $|V'|$ are the numbers of clusters and nodes in G', respectively, and $W'_V(v')$ is the computational cost of node v'. E' is a finite set of undirected edges. An edge $e' \in E'$ is an unordered pair $(v'_x, v'_y) \in V'$, $e_{v'_x v'_y}$ represents the communication between nodes v'_x and v'_y, and $W'_E(e_{v'_x v'_y})$ is the communication cost between nodes v'_x and v'_y.

A computational grid with a three-level-tree is specified according to the following rules. All nodes are in the same node level. When nodes can communicate to one another with the same communication cost, they can be grouped into the same cluster. A real cluster that has its nodes linked with the same network technology and medium is an example of a cluster in this model. An individual node is considered as a cluster of one node. All participating clusters (also the nodes) are in the same grid level and communicate through the grid network.

3.3 The Mapping Functions

When a parallel application, G, is mapped on a computational grid, G', the execution time of the application, $ET(G)$, is the execution time of the slowest node in G'; i.e.,

$$ET(G) = ET(v') \tag{1}$$

where $ET(v')$ is the execution time of the slowest node v'. $ET(v')$ is the sum of the computational and communication times of the tasks mapped on v'; i.e.,

$$ET(v') = \sum_{i=1}^{N} W_V(v_i) W'_V(v') + \sum_{i=1}^{N} \sum_{j=1}^{M_{v_i}} W_E(e_{v_i v_j}) W'_E(e'_{v' v'_j}) \tag{2}$$

where v_i is the i^{th} task mapped on node v', N the number of tasks mapped on node v', v_j the j^{th} neighbour of task v_i, M_{v_i} the number of neighbours of task v_i, and v'_j the node on which task v_j is mapped (see Fig.2 for an example).

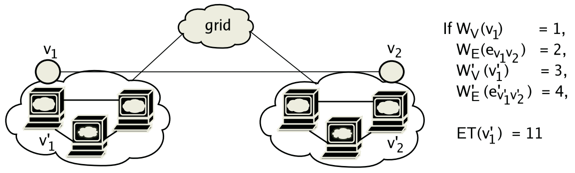

Fig. 2. A 2-task parallel application mapped on a grid modelled by a three-level-tree

4 QM Algorithm

Unlike the other algorithms that optimise the execution time of the slowest node, QM instead optimises the execution time of each task. The core idea is that each task is iteratively mapped on a new node such that the execution time of the task is lower than its current execution time. Given v as a task mapped on node v', the execution time of task v, $ET(v)$, is equal to the execution time of node v', $ET(v')$; that is,

$$ET(v) = ET(v') \tag{3}$$

The flows of QM algorithm are shown in Fig.3. The first step is to coarsen the application graph until the number of tasks is less than a threshold. Each task in the coarsest graph is mapped on a randomly chosen node, which becomes the *current node* of the task. The execution times of all chosen nodes are then calculated using (2). In this step, each task has its own execution time, which is equal to the execution time of its current node.

Iteratively, a better node for each task is searched. In the case that more than one such node exist, the task is mapped on the *best node*, which is the node that gives the task the lowest execution time. However, not all nodes in the environment need to be considered.

Let c' be a cluster in G', v'_x the node in cluster c' such that its execution time, $ET(v'_x)$, is lowest. If more than one such node exist, the node with the lowest computational cost, $W'_V(\cdot)$, is considered to be v'_x. Let V'_y be a set of nodes in cluster c' that their computational costs, $W'_V(V'_y)$, are less than the computational cost of v'_x, $W'_V(v'_x)$. Let v be a task to be mapped, and V'_z a set of nodes in cluster c', which the neighbours of task v are mapped on (see Fig. 4 for an example).

```
00. QM ($G$, $G'$)
01.     while ($|V| \geq threshold$)
02.         $G$ = coarsen $G$;
03.     for ($i = 0$; $i < |V|$; $i = i + 1$)
04.         $v'_r$ = randomly choose a node in $G'$;
05.         map $v_i$ on $v'_r$;
06.     update execution times of all nodes in $G'$;
07.     do
08.         for ($i = 0$; $i < \lfloor \frac{max}{\log_{10}|V|+1} \rfloor$; $i = i + 1$)
09.             for ($j = 0$; $j < |V|$; $j = j + 1$)
10.                 $v'_b$ = find the best node in $G'$;
11.                 if ($v'_b$ is found)
12.                     map $v_j$ on $v'_b$;
13.                     update execution times of $P(v_j)$, $v'_b$, $N(v_j)$;
14.                     update $v'_x$ and $V'_y$ of relevant clusters;
15.     while ($G$ = un-coarsen $G$ is applicable)
```

Fig. 3. QM algorithm

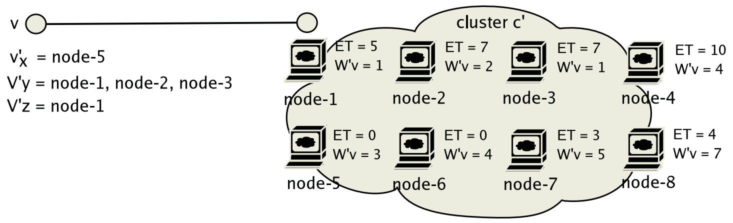

Fig. 4. Nodes in the cluster according to proposition 1

Proposition 1. *If the best node v'_b for task v exists in cluster c', then either $v'_b = v'_x$ or $v'_b \in (V'_y \cup V'_z)$ is true.*

Proof. Given the condition $v'_b \neq v'_x$ and $v'_b \notin (V'_y \cup V'_z)$ is true. Thus, before mapping task v, the execution times and computational costs of nodes v'_b and v'_x in (4) and (5) are true.

$$ET(v'_b) \geq ET(v'_x) \tag{4}$$

$$W'_V(v'_b) > W'_V(v'_x) \tag{5}$$

If the given condition is true, then the comparison of the execution time between nodes v'_b and v'_x after mapping task v in (6) must hold.

$$ET'(v'_b) < ET'(v'_x) \tag{6}$$

where ET' denotes the execution time after the mapping. (6) is equal to

$$ET(v'_b) + W_V(v)W'_V(v'_b) < ET(v'_x) + W_V(v)W'_V(v'_x) \tag{7}$$

However, (6) is false since $ET(v'_b) - ET(v'_x) \geq 0$ and $W'_V(v'_x) - W'_V(v'_b) < 0$. Hence, the given condition $v'_b \neq v'_x$ and $v'_b \notin (V'_y \cup V'_z)$ is false.

It can be seen that the number of nodes that each task needs to consider is reduced significantly without any impact on the mapping solution.

If the best node v'_b for task v_j is found, v_j is mapped on the best node, and the execution times of all relevant nodes are updated. These nodes are the previous node of v_j, $P(v_j)$, the best node, v'_b, and all the nodes on which the neighbours of task v_j are mapped, $N(v_j)$. v'_x and V'_y of the relevant clusters also need to be updated. The relevant clusters are the ones that the relevant nodes belong to. If the best node is not found, the task is remained on its current node.

Then, the application graph is un-coarsened, and the same optimisation is applied. This procedure is repeated until the application graph cannot be un-coarsened. The current nodes of the tasks are then the mapping solution.

Two parameters that have effect on the performance of the algorithm are *threshold* and the number of iterations for a task to find the best node. To avoid the degrade in performance, the number of iterations is reversely proportional to the number of tasks (i.e. $\lfloor \frac{max}{\log_{10}|V|+1} \rfloor$). Preliminary experiments showed that $threshold = \sqrt{|V|}$ and $max = 13$ gave promising results.

5 Experiments

In the experiments, app-1, app-2 and app-3 consists of 100, 2500 and 10000 tasks representing small, medium and large scale parallel applications, respectively, are mapped on ten grid testbeds. The topology of the applications is the two-dimensional circular Cartesian. The number of neighbours of each task, and the computational and communication costs are randomly varied from 1 to 4. The testbeds are generated with different numbers of clusters and nodes (as shown in Table 1). In each cluster, the number of nodes is randomly varied from 1 to 64. The computational and communication costs are randomly varied from 1 to 10. All the costs in the application graphs and testbeds are the same as the ones used in MiniMax [4] and FastMap [5].

Table 1. The specifications of the grid testbeds

G'	grid-10	grid-20	grid-30	grid-40	grid-50	grid-60	grid-70	grid-80	grid-90	grid-100
$\|C'\|$	10	20	30	40	50	60	70	80	90	100
$\|V'\|$	359	615	914	1228	1679	1842	2535	2684	2915	2992

QM, MiniMax and a genetic algorithm (GA) are the experimented algorithms. It is preferable to compare FastMap with these algorithms. However, we are unable to do so due to incomplete information of the algorithm on how to calculate the execution time of the tasks during a mapping step.

There are two versions of QM: QM-1 and QM-2. QM-1 searches all nodes to find the best node, but does not coarsen the application graph. QM-2 searches the nodes according to proposition 1, and coarsens the application graph.

Since MiniMax cannot function if the number of tasks is less than the number of nodes (i.e. $|V| < |V'|$), in such situation, $|V|$ nodes are selected from the testbed. Three selecting algorithms are employed, and hence there are four versions of MiniMax: MiniMax-1, MiniMax-2, MiniMax-3 and MiniMax-4.

MiniMax-1 selects $|V|$ nodes as to minimise their total communication cost. This selection algorithm is used to select the nodes for Cactus (an astrophysics application) [7]. MiniMax-2 selects $|V|$ nodes that have the lowest computational costs while MiniMax-3 randomly selects $|V|$ nodes. MiniMax-4 functions in the situation that $|V| \geq |V'|$, and no node selection is required. The GA implemented is the same as the one used in [8].

The experiments are conducted on a 2.8 GHz Pentium-4 computer, and the presented results are an average of 10 runs. Fig.5, Fig.6 and Fig.7 show the quality (the lower the execution time the higher the quality), and performance (the lower the mapping time the higher the performance) of the experimented algorithms when mapping app-1, app-2 and app-3 on the testbeds, respectively.

In Fig.5, QMs are better than the other algorithms in terms of quality. QM-2, which coarsens the application graph, gives the better solutions than QM-1 while GA is the worst. Note that MiniMax-4 is not applicable since the number of tasks (i.e. $|V| = 100$) is less than the number of nodes in all testbeds. Also notice that selecting algorithms have effects on the quality of the solutions. Random selection (MiniMax-3) is the worst; however, it is not conclusive between MiniMax-1 and MiniMax-2. In terms of performance, GA takes much longer time than the other algorithms. MiniMax-3 is the fastest while QM-2 has only little overhead over MiniMax-3's. Notice that QM-1 is slower than QM-2 since QM-1 searches for the best node from all nodes in the environment.

In Fig.6, MiniMax-4 is applicable when mapping the application on grid-10 to grid-60 since the number of tasks (i.e. $|V| = 2500$) is more than the numbers of nodes in these testbeds. In terms of quality, QM-2 outperforms the other algorithms, and GA is the worst. In terms of performance, GA appears to be the worst. However, when mapping the application on grid-50 and grid-60, MiniMax-4 takes the longest time. This is due the large number of tasks (i.e. $|V| = 2500$) and the sharp increase in the number of nodes (from 1228 in grid-40 to 1679 and 1842 in grid-50 and grid-60, respectively). QM-2 is the fastest, and both QMs are faster than all MiniMax algorithms in all mapping cases.

In Fig.7, QM-2 still outperforms the other algorithms. In terms of quality, MiniMax-4 can produce the solutions with the quality close to QM-2's while GA

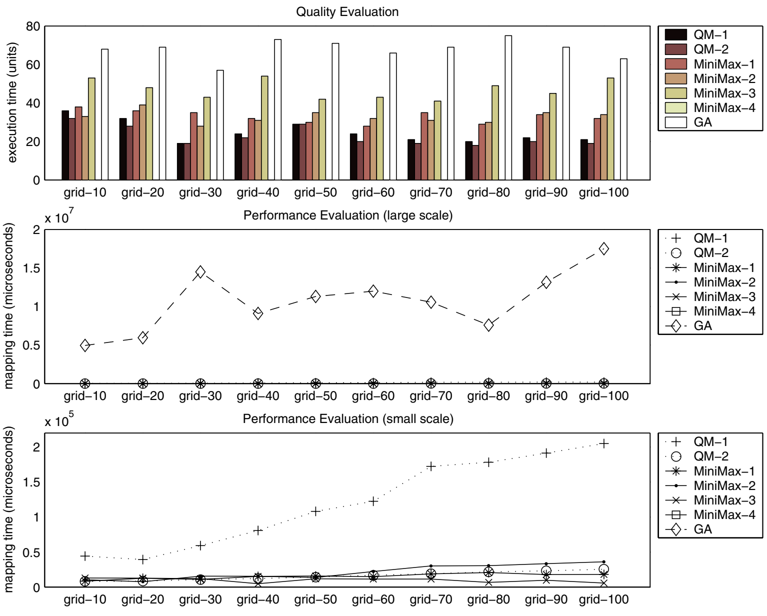

Fig. 5. The quality and performance of QM, MiniMax and GA when mapping app-1

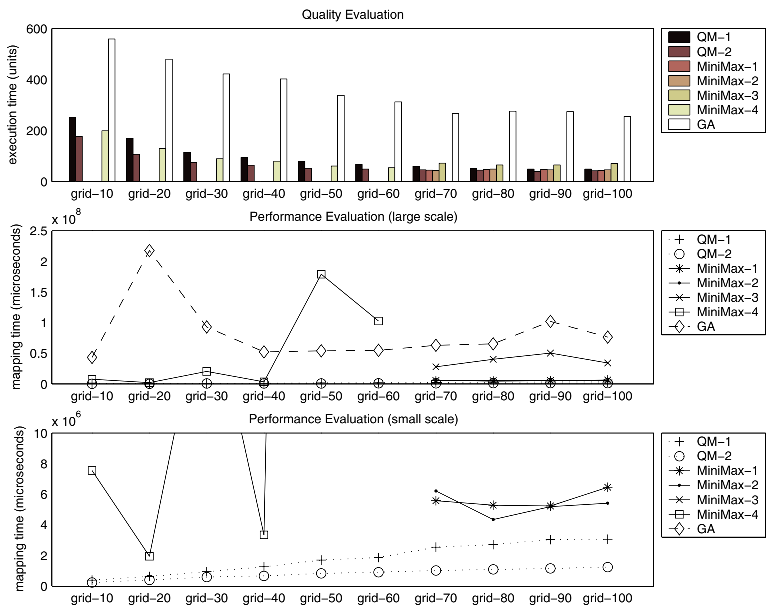

Fig. 6. The quality and performance of QM, MiniMax and GA when mapping app-2

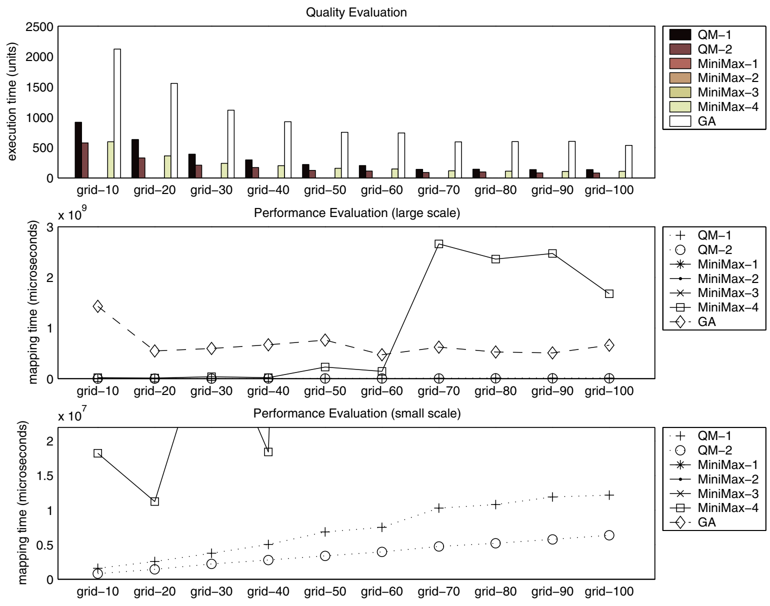

Fig. 7. The quality and performance of QM, MiniMax and GA when mapping app-3

is still the worst. In terms of performance, GA takes the longest time to map the application on grid-10 to grid-60 whereas MiniMax-4 is the slowest algorithm when mapping the application on grid-70 to grid-100. This is also due to the large number of tasks (i.e. $|V| = 10000$), and the sharp increase in the number of nodes (from 1843 in grid-60 to 2535 in grid-70). QM-2 is still the fastest.

When considering both the quality and the performance, QM-2 performs better than the other algorithms. For example, QM-2 can map app-3 (10000 tasks) on grid-100 (2992 nodes) in 6.35 seconds, and gives the lowest execution time while GA and MiniMax-4, respectively, take approximately 660 and 1700 seconds, but produce 6.60 and 1.34 times greater execution times than QM's. From the results of QM-1 and QM-2, it is conclusive that considering the best node according to proposition 1 improves the performance significantly while graph coarsening is a major key to improve the quality of the mapping solutions.

6 Conclusions

The existing algorithms to map parallel applications on computational grids, such as MiniMax, FastMap and genetic algorithms have shortcomings. In this paper, a new mapping algorithm is presented. The core idea is to map each task to a new node that gives a lower execution time to the task than its current node. The technique to coarsen the application graph is also employed. Experimental results show that QM performs better than the other algorithms, and graph coarsening is a major key to improve the quality of the mapping solutions.

Future work aims at deploying the algorithm in real environments. However, unrealistic assumptions hinder the deployment. These problems are currently being investigated [9].

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organisations. International Journal of Supercomputer Applications **15** (2001)
2. Bokhari, S.: On the mapping problem. IEEE Transaction on Computers **C-30** (1981) 207–214
3. Ullman, J.: Np-complete scheduling problems. Computer and System Sciences **10** (1975) 434–439
4. Kumar, S., Das, S., Biswas, R.: Graph partitioning for parallel applications in heterogeneous grid environments. In: International Parallel and Distributed Processing Symposium (IPDPS 2002). (2002) 66–72
5. Sanyal, S., Jain, A., Das, S., Biswas, R.: A hierarchical and distributed approach for mapping large applications onto heterogeneous grids using genetic algorithms. In: IEEE International Conference on Cluster Computing. (2003) 496–499
6. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys **31** (1999)

7. Liu, C., Yang, L., Foster, I., Angulo, D.: Design and evaluation of a resource selection framework for grid applications. In: 11th IEEE International Symposium on High Performance and Distributed Computing (HPDC 2002), Edinburgh, Scotland (2002)
8. Braun, T., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **61** (2001) 810–837
9. Phinjaroenphan, P., Bevinakoppa, S., Zeephongsekul, P.: A method for estimating the execution time of a parallel task on a grid node. In: European Grid Conference. (2005)