A Static Mapping Heuristic for Mapping Parallel Applications to Heterogeneous Computing Systems

Ranieri Baraglia(1), Renato Ferrini(1), Pierluigi Ritrovato(2)
(Ranieri.Baraglia, Renato.Ferrini)@cnuce.cnr.it
ritrovato@crmpa.unisa.it

(1) ISTI - Istituto di Scienza e Tecnologie
dell'Informazione A. Faedo, Consiglio Nazionale delle Ricerche
(2) Centro di Ricerca in Matematica Pura ed Applicata,
Universitá degli Studi di Salerno

Abstract

To minimize the execution time of a parallel application running on a heterogeneous computing distributed system, an appropriate mapping scheme to allocate the application tasks to the processors is needed. The general problem of mapping tasks to machines is a well known NP-hard problem and several heuristics have been proposed to approximate its optimal solution. In this paper we propose a static graph-based mapping algorithm, called Heterogeneous Multi-phase Mapping (HMM), that permits a suboptimal mapping of a parallel application onto a heterogeneous computing distributed system by using a local search technique together with a tabu search meta-heuristic. HMM allocates parallel tasks by exploiting the information embedded in the parallelism forms used to implement an application. We compare HMM with three different leading techniques and with an exhaustive mapping algorithm. We also give an example of mapping of a pratical application where HMM verified its usefulness. Experimental results show that HMM performs well demonstrating the applicability of our approach.

Categories and Subject descriptors: D.4.1 [Operating System]: Process Management; D.4.1 [Operating System]: Process Management- Metacomputing/Grid computing/Heterogeneous computing.

Key Words: Parallel processing, static mapping, task mapping, optimization, performance evaluation.

1 Introduction

Heterogeneous Computing (HC) is a set of techniques enabling the use of different interconnected machines to provide a variety of computational capabilities to execute

collections of application tasks that have different requirements [1, 2]. There are many types of heterogeneous systems [3]. In this paper we refer to a HC system as a heterogeneous suite of independent machines connected by high-speed networks, and used as a single HC distributed system [1]. This kind of HC system is also known as *Metacomputer* [4] or as *Grid* [5]. To run on a metacomputing environment an application has to be decomposed into computationally homogeneous tasks that can either be sequential or parallel. Applications that require the use of different computational types (Sequential, SIMD, MIMD, etc.) are the best candidates to exploit the potential of a metacomputer.

An important goal of HC is to find an assignment schema to determine the machine on which every task is to be executed and the task execution scheduling order to minimize the execution time of an application. The assignment-scheduling process is called *mapping*. Factors such as machine and network power and loading, task computational requirements, data dependences between tasks, matching between task computational-machine type, have be considered to optimize the mapping process [6].

The general problem of mapping tasks to machines has been shown to be NP-complete. Mapping algorithms are usually classified as static or dynamic [26]. In the former the mapping decisions are taken before executing an application and are not changed until the end of the execution. In the latter the mapping decisions are taken while the program is running. Since the static mapping does not imply overheads on the execution time of the mapped application, more complex mapping solutions than the dynamic ones can be adopted. Static mapping heuristics can be used to plan the work on a HC system, and, in any case, an efficient static mapping constitutes a good first step in implementing a dynamic mapping heuristic.

This paper deals with static graph-based mapping algorithms. In general, these kinds of algorithms use a directed acyclic graph to model a parallel application, and an undirected graph to model the system. Moreover, they assume that the task computational cost and the communication patterns among tasks are known a priori.

In this paper we propose Heterogeneous Multi-phase Mapping (HMM), a static graph-based mapping algorithm, that allows a suboptimal mapping of a parallel application onto a metacomputer to minimize its execution time. HMM allocates parallel tasks by exploiting the information embedded in the parallelism forms used to implement an application. Moreover, it uses a local search technique together with the tabu search meta-heuristic [27].

Most of the parallel programs are designed using parallelism forms ($farm$, $pipeline$, $geometric$, etc.) [15, 16] which can be used by adopting models that allow arbitrary computation structures (unrestricted programming model) [17, 28] and also models that restrict the forms in which computation may be expressed (restricted programming model) [18, 20].

Various coordination languages have been proposed [21, 22, **?**, 19, **?**] in which a set of parallel constructs (parallelism forms) are used as program composition forms.

Parallelism forms allow the implementation of parallel computations in which the communication patterns are structured and well defined, and, in general, a significant amount of communication is concentrated within a parallelism form. Therefore, in order to reduce the communication cost, it is usually convenient to allocate the tasks inside a parallelism form on the same parallel machine.

A parallel application, implemented using both restricted or unrestricted programming models, can be modeled by using a DAG.

HMM takes in input a parallel program modeled by a DAG. If the parallel program has been designed using a restricted parallel programming model, the DAG's nodes included in parallelism forms are managed as a hierarchical nodes (clusters of tasks) that can be handled recursively. It has the advantage of driving the mapping of a parallelism form to an individual parallel machine.

This paper is organized as follows. Section 2 describes some related works. Section 3 gives a description of the problem, and Section 4 describes our algorithm. In Section 5 the algorithm time complexity is evaluated. Section 6 outlines and evaluates the experimental tests. Finally, we summarize our work in Section 7.

2 Related Work

A number of static graph-based mapping algorithms for heterogeneous computing environments have been proposed [8, 23, 3, 24, 25, 30, 31, 7], and evaluated [13], to approximate the optimal mapping solution.

In [3] M. Eshaghian et al. proposed the *Cluster-M* and the *Augmented Cluster-M* mapping algorithms that are based on *Cluster-M* programming model. Cluster-M uses a direct graph and undirected graph to represent a meta-application and a meta-computer, respectively. The direct graph is horizontal and vertical partitioned. The horizontal partitioning builds layers (clusters) of tasks that can be executed in parallel. Vertical partitioning allows consecutive layers to be merged or embedded in order to minimize communication costs. The undirected graph is represented in a multilayered format such that each layer includes a set (cluster) of completely connected processing units. Clusters of communication-intensive tasks are then mapped onto clusters of communication-efficient processing units by running Cluster-M. The latter presents a time complexity equal to $O(MP)$, where M is the number of tasks belonging to an application graph, and $P = max(M, N)$, with N numbers of processors belonging to a metacomputer graph.

Augmented Cluster-M is an extension of Cluster-M. Since it treats the task computational type-machine type affinity, that drives the mapping of a task onto a machine with the same computation type, it is more suitable for heterogeneous computing. The major drawback of this solution is the need of to re-computing the clustering of the system graph when the metacomputer configuration changes. Metacomputer are dynamic computing environment on which the availability of computational units cannot be established a priori. Moreover, the machines and networks workload can lead to different mapping for different time requests of the same application execution.

In [8] V. M. Lo proposes a max-flow min cut algorithm to find a mapping of task modules on heterogeneous processors that is an extention of the Stone's model [9]. The algorithm combines recursive invocations of Max Flow/Min Cut Algorithm with a greedy algorithm to find suboptimal assignment of task modules on heterogeneous processors. It has a complexity equal to $O(M^2N|E_p|logM)$ where M, N and E_p are the number of parallel tasks, processors and communication links between processors, respectively. This algorithm presents two main drawbacks, a high time complexity and

Figure 1: Directed Acyclic Graph representing a parallel application.

it does not treat the task data dependences.

In [10] C. Shen and W. Tsai propose the A^* Searching algorithm that exploits a graph matching model called weak homomorphism to map a task graph to a system graph. The search of the optimal weak homomorphism corresponds to the task assignment with minimum task turnaround time. It is formulated as a state-space search problem, solved using the A^* algorithm [11]. The latter is a Branch & Bound algorithm, and, therefore, due to its time complexity it is suitable for small problems.

3 Problem Description

A parallel application, is represented by a Directed Acyclic Graph (DAG) denoted as $TG = (N, E)$ where N is a set of nodes and E is a set of directd edges. A node represents a computational task (homogeneous computation) that can be sequential or parallel, called *atomic* and *hierarchical* respectively.

Atomic nodes are indivisible units of execution, and have associated a weight representing the computation amount, that is the number of clock cycles required to execute the associated task on a baseline machine [3]. Hierarchical nodes are sets of atomic and/or hierarchical nodes structured according to the parallelism forms used to implement a parallel program. Parallelism forms can be nested each other.

Hierarchical nodes are represented with a DAG as well, and its structure respects the order in which the parallelism forms are used in a parallel task. From the task mapping point of view a hierarchical node is considered to be atomic.

In Figura 1 an example of DAG representing a parallel application is shown.

We assume that loops and conditional branches are contained inside an atomic node.

The application DAG is considered structured in S *layers* that represent the depth of nested levels. In the first layer it consists of only a hierarchical node which is considered to be atomic, and in the next layers it could consist of atomic and/or hierarchical nodes represented with a DAG denoted as TG_s with $1 \leq s \leq S$. In the last layer all nodes are atomic.

Figure 2: Layers inside a TG graph representing a parallel application.

A generic node, atomic or hierarchical, belonging to a graph TG_s is identified as n_i^s with $1 \leq i \leq N_s$ with N_s number of nodes at the layer s. In Figure 2 an example of a three-stage expansion of a hierarchical node is given. The first layer consists of a hierarchical node; the second layer is represented by a task graph of four nodes, one atomic and three hierarchical (a pipeline and two task-farms), and in the last layer the task graph includes fourteen atomic nodes.

An edge $e_{i,j}$ represents a data dependence between the nodes n_i^s and n_j^s with $i \neq j$, and it has an associated weight specifying the amount of data exchanged between the nodes.

We assume that the TG graph and the node and edge weights are produced at compile-time.

A metacomputer is represented by a fully connected indirect graph denoted as MG. Each node of MG represents a metacomputer machine which is denoted as m_i with $1 \leq i \leq M$, where M is the number of machines in the metacomputer. To each node is associated a number that specifies the machine computational speed and each edge denotes a link between two machines, and has an associated value which specifies its communication bandwidth.

To find out at any time the processors available on a machine m_i, a vector V_{m_i}, with a size equal to the number of processors of m_i, is associated with each machine. The u-th entry of this vector contains the time value at which the u-th processor of m_i will be available to execute new work. HMM updates the vector V_m on the basis of the workload due to the application processes allocated on each processor of the machine m_i.

To enable a more realistic mapping it is useful to characterize both the application and the machine by relating them to some real-world parameters such as the amount of memory required, the computational model adopted, software required, architectural class, number of processors and network bandwidth.

This enables us to compute a value called *affinity* that identifies which machine in

the metacomputer is most suitable for executing a component of a parallel application [3, 12, 1].

In HMM the affinity parameter can assume values from 0 to 1. It assumes 0 when the computational resources of a machine do not allow the execution of a task. Otherwise, it assumes a value > 0 and ≤ 1 with respect to the best matching between the task computational requirements and the machine's computational features.

To reduce the mapping complexity a TG_s is horizontally divided into P $phases$, which are executed sequentially (top-down) one at a time. The nodes in the first phase do not have parents, and the nodes in phase p, with $2 \leq p \leq P$, are those which have at least one parent among the nodes in phase k with $1 \leq k \leq p-1$. By structuring TG_s into phases the precedence relations among its nodes reflect the execution sequence of the phases in the application graph, and the mapping problem is simplified because HMM finds the initial suboptimal mapping solution by operating on each phase separately. The initial problem is thus decomposed into more manageable subproblems. Since the phases are linked by the edges to map the h-th phase HMM considers the data exchanged between the nodes belonging to the phase h and the nodes of the phase k with $k < h$. The nodes in a phase can be executed in parallel, and a node in a phase is executed when all its predecessors have been completed and it has received all the data needed for its execution.

The introduction of the $layer$ concept allows us to carry out the layer whose mapping guarantees the shortest execution times of the nodes in it contained on the metacomputer used. This is achieved by adopting the following strategy: starting from node n_1^1 (node at layer one) HMM finds the most suitable metacomputer machine for its execution on the basis of the machine processor power, the machine workload, the tasks exchanged data, and the node-machine affinity value.

HMM estimates the execution time of the task graph TG_1, that consists of the node n_1^1, on the selected machine, and then TG_1 is expanded in the next layer producing the task graph TG_2. To carry out a better mapping of TG_2, it is divided into phases, and starting from the first one HMM finds the most suitable machine to execute each node contained in a phase by using the machine parameters seen before and the communication between the node and its parents. Then, the estimated execution time of TG_2 is compared with the one found for TG_1. The shortest one leads to the current initial partial mapping solution. This process is repeated for all the remaining $S-2$ layers of the application graph to find an initial mapping solution by selecting the TG_s to which a shorter execution time corresponds to. Then, to refine the initial mapping solution a local search algorithm together with the tabu search meta-heuristic [27] is adopted.

The suboptimal mapping solution is found by keeping the tasks communication cost as low as possible, and by trying to make a good trade-off between the communication costs and the workload by balancing among the metacomputer machines.

4 Algorithm Description

HMM takes in input the TG and MG graphs, which represent a parallel application and a metacomputer, respectively. The algorithm is structured according two phases.

In the first one an initial mapping solution is carried out. In the second one, to improve the intial solution quality a local search function is applied to the atomic nodes in the critical path of TG_s to which the initial solution corresponds to.

Figure 3 shows the pseudo-code of the HMM first phase which operates according to the following steps:

Figure 3: Pseudo-code of the first phase of HMM.

Step 1 computes the affinity value of all n_i^s. The affinity value of a hierarchical node is computed by considering both the required computational resources and the adopted computational model by all the hierarchical and atomic nodes in it contained. The affinity value of a hierarchical node n_i^s is 0 if n_i^s contains at least an atomic node which has no affinity with all metacomputer machines. HMM ends if at least an atomic node has affinity equal to 0 with all metacomputer machines.

Step 2 divides all nodes of a TG_s into P phases.

Step 3 arranges in descending order the nodes in a phase p with respect to their computational workload.

Step 4 selects, according to the ordering given by the previous step, a node n_i^s which has not yet been analyzed and checks whether there is at least one machine m_i that can perform it. This selecting order improves the probability of heavier hierarchi-

cal nodes to be allocated.

The selection of a machine is based on the value:

$$\max(Aff(n_i^s, m_k) \times S_{m_k} \times (1 - \frac{W_{m_k}}{100}) - \frac{D_c(n_i^s, n_{\bar{i}}^s)}{100}) \quad (1)$$
$$k = 1, .., M \quad and \quad (n_{\bar{i}}^s) \in BS(n_i^s)$$

in which $Aff(n_i^s, m_k)$ is the affinity of n_i^s with respect to m_k; S_{m_k} is the computational power of m_k and W_{m_k} is the workload of m_k. D_c is the communication cost of the node n_i^s with the node $n_{\bar{i}}^s$ belonging to its backward star $BS(n_i^s)$ (i.e. its parents). D_c is zero if n_i^s and its parent with which it has the highest communication cost can be allocated on the same machine. Otherwise it is equal to the amount of data exchanged with the parent with which it has the highest communication cost.

The machine which the maximum value computed by (1) corresponds to is selected to allocate the node n_i^s. When the maximum value corresponds to several machines, HMM selects the machine m_k which the smallest value of the ratio $\frac{W_{m_k}}{S_{m_k}}$ corresponds to.

If any machine is found, the hierarchical node n_i^s is flagged "not allocated", and the next node is analyzed. Otherwise, step 5 is executed in order to estimate the execution time of n_i^s on the candidate machine.

Step 5 estimates the overall completion time T_e of a node n_i^s on m_k. T_e is computed according to its execution precedence relations and is given by:

$$T_e(n_i^s, m_k) = T_w(n_i^s, m_k) + T_x(n_i^s, m_k) \quad (2)$$

where $T_w(n_i^s, m_k)$ is the time that the node n_i^s has to wait before executing on the machine m_k, and $T_x(n_i^s, m_k)$ is the execution time of n_i^s on m_k.

The $T_w(n_i^s, m_k)$ is given by:

$$T_w(n_i^s, m_k) = \max\{T_c(BS(n_i^s)), T_a(n_i^s, \bar{p}_{m_k})\} \quad (3)$$

where T_c is the waiting time due to the communications of n_i^s with the nodes belonging to $BS(n_i^s)$ and $T_a(n_i^s, \bar{p}_{m_k})$ is the time spent waiting to get the $\bar{p}$ processors of the machine m_k needed for the n_i^s execution. T_a is obtained from the vector V_{m_k} in which are stored the processors in increasing orders with respect to the completion time of the task being executed on them. The u-th entry of V_{m_k} stores the time at which the u-th processor becomes available.

The waiting time $T_c(BS(n_i^s))$ is given by:

$$T_c(BS(n_i^s)) = \max\left\{Tend(n_j^s) + \frac{D_c(n_i^s, n_j^s)}{B_{m_k,m_n}}\right\} \quad (4)$$
$$\forall\ j\ with\ (n_j^s) \in BS(n_i^s)$$

where B_{m_k,m_n} is the communication bandwidth between machine m_k on which n_i^s will be allocated and the machine m_n on which the n_j^s has been allocated. $Tend$ is the time at which the node n_j^s ended its execution. If both n_i^s and n_j^s are allocated on the same multiprocessors B_{m_k,m_n} is the processors interconnection network bandwidth.

$T_x(n_i^s, m_k)$ is equal to $\frac{C_{n_i^s}}{W_m}$, where $C_{n_i^s}$ is the computational workload of n_i^s. If n_i^s is a hierarchical node, to compute its execution time the equation (2) is recoursively applied to all the atomic nodes in it contained.

When the list of nodes to analyze is empty, step 6 is performed.

Step 6. HMM checks whether there are some nodes flagged "not allocated". If there are, by adopting the procedure described in steps 4-5, these nodes are allocated in the order established in step 3. The search for the most suitable machine to allocate a not yet allocated node starts from the first layer greater than s-not allocated layer for which there is a machine that can execute each node belonging to it. If there are no nodes flagged "not allocated", this process ends by carrying out the initial partial solution of the phase p analyzed, and the execution continues from step 3 to elaborate the next phase.

Step 7 analyzes all initial mapping partial solutions to find the layer completion time (S_s) which corresponds to the TG_s node which ends its execution last. S_s is then compared with the best initial mapping solution S_{init} carried out in a previous layer. If S_s is better than S_{init} it becomes S_{init}, and the execution continues from Step 2 to analyze the next layer. When all the layers has been evaluated, the algorithm ends returning S_{init}.

Figure 4 shows the pseudo-code of the HMM second phase. It considers all the nodes in the current solution as atomic, and analyzes the neighborhood of S_{init} to search a better solution. We say that a neighborhood of a solution is the set of solutions obtainable by moving a node of the critical path from the processor $p_{m_k}^i$ (current allocation) to a processor $p_{m_h}^j$ with $i \neq j$. The choices carried out by the local search algorithm for mapping a node consider the data exchanged between nodes n_i^s belonging to the phase h and n_j^s of phase k with $k < h$.

The new processor $p_{m_k}^j$ of the metacomputer machine m_k is selected according to the following criterion:

$$\max \left\{ Aff(n_i^s, m_k) \times S_{p_{m_k}^j} \right\} \tag{5}$$

where $S_{p_{m_k}^j}$ is the computational power of the j-th processor of the machine k.

If several processors are selected, the one with the smallest value of the fraction $\frac{W_{p_{m_k}^j}}{S_{p_{m_k}^j}}$ ($W_{p_{m_k}^j}$ is the processor workload) is selected. If there is not a processor able to allocate n_j^s, the node is not moved, and the process continues by selecting the next node in the critical path. Otherwise, the selected node is moved, and HMM estimates the cost of the new solution by adding the computing and communication times of the nodes belonging to critical path of the new solution. This cost represents the application execution time.

Figure 4: Pseudo-Code of the second phase of HMM algorithm.

When all the nodes on the critical path have been analyzed the local search algorithm carries out a new solution. If it is better than the initial solution, it becomes the current solution.

The search for a suboptimal solution is an iterative process. HMM ends by returning the final mapping solution, when one of the following conditions is verified: a) the total number of iterations performed reaches a user-defined threshold, b) the number of iterations performed without improving the solution reaches a user-defined threshold.

4.1 Mapping Example

To show the choices taken by HMM an example of mapping of a real quantum chemistry application on three different metacomputer configurations (see Figures 5(b), 8(b), and 9(b))is given. The chemistry application deals with the integration of a two-dimensional Schrodinger equation using a sector-diabatic coupled-channel approach [32].

In the following figures the application tasks are represented in a task graph by nodes labeled t_i. The number inside each circle representing an atomic node specifies the computationl cost of the task, and the communication cost is specified by the number associated to each edge. In a system graph the nodes labeled p_i represent the processors inside a machine. The number inside each circle specifies the processor's computational power, and the communication bandwidth of the channel linking two processors or two machines is specified by the number associated to each link between them.

Figure 5(b) shows the metacomputer configuration in which all machines have processors with equal computational power, and equal communication bandwidth. In Figures 8(b) and 9(b) are represented the two other configurations in which the machines m_1, and the m_1, m_2, m_3, m_4, respectively, have processors with half computation speed and half communication bandwidth with respect to the same machine in the metacomputer graph of Figure 5(b). This situation can happen, for example, when a machine is workloaded at 50%, and therefore we can suppose that its computational power is cut down of a half.

In Figures 5, 8, and 9, the arrows together with the grey levels indicate the layer at which the initial mapping solution (S_{init}) was carried out, and the machines onto which the tasks were mapped. The affinity parameter was computed as function of the number of processors own by each metacomputer machine.

Figure 5 shows the case in which the metacomputer has the machine m_1 large enough to carry out the application mapping at layer 1 at which S_{init} is equal to 69.

In Tables 6 and 7 are shown the values of the affinity parameter (Aff), the machine computational power (S_m), the nodes data exchanged (D_c), and the machine workload (W_m) used to compute the expression (1). Table 6 shows the values computed to allocate at layer 1 the task graph of the example of Figure 5, and Table 7 shows the same values computed to allocate at layer 2 the same task graph on the metacomputer of Figure 8. In the same way the expression (1) was computed to allocate at layer 3 the task graph of the example shown in Figure 9. In all the examples the mapping at which the smallest value of S_{init} corresponds to was chosen.

Figure 5: Mapping carried out at layer 1.

In Figures 5 (c), 8 (c), and 9 (c) the final suboptimal mapping solutions obtained after the local search process are shown.

5 Algorithm Complexity

To evaluate the algorithm time complexity we consider the number of operations performed to carry out the initial mapping solution and to run the local search algorithm.

5.1 Initial Solution

The cost $C(S_{init})$ of the initial solution is computed according to the number of operations executed to allocate the atomic and/or hierarchical nodes within all layers of TG. The cost $C(TG_s)$ to allocate the N_s nodes of a TG_s (a graph expanded at layer s) is given by:

$$C(TG_s) = \frac{N_s^3}{3} + P \times N_s log_2 N_s + N_s \times M \tag{6}$$

Figure 6: Values computed by HMM to evaluate the mapping at layer 1 of the application graph on the metacomputer shown in Figure 5.

Figure 7: Values computed by HMM to evaluate the mapping at layer 2 of the application graph on the metacomputer shown in Figure 8.

where $\frac{N_s^3}{3}$ is the number of operations performed to divided TG_s into P phases, $N_s log_2 N_s$ is the number of operations performed to sort the nodes within a phase (we suppose that in each phase there are all TG_s's nodes), and $N_s \times M$ is the number of operations executed to choose a machine on witch to allocate the N_s nodes.

Since the worst case on which there are as many layers as the number of the atomic nodes is rare to take place, to compute $C(S_{init})$ we consider the average case on which TG_s is structured in $log_2 N + 1$ layers and each layer includes $2^{(i-1)}$ nodes with $1 \leq i \leq log_2 N + 1$. In this case $C(S_{init})$ is given by:

$$C(S_{init}) = \sum_{i=1}^{log_2 N+1} C(TG_i) \tag{7}$$

Since every TG_i includes 2^{i-1} nodes the expression (7) can be written as

$$C(S_{init}) = \sum_{i=1}^{log_2 N+1} \frac{(2^{(i-1)})^3}{3} + P \times 2^{(i-1)} log_2 2^{(i-1)} + 2^{(i-1)} \times M \tag{8}$$

Figure 8: Mapping carried out at layer 2.

which is equal to

$$C(S_{init}) = \sum_{i=1}^{log_2 N+1} \frac{2^{3(i-1)}}{3} + 2^{(i-1)} \times log_2 2^{(i-1)} + 2^{(i-1)} \times M \tag{9}$$

which is equal to

$$C(S_{init}) = N^3 + N \times lnN + N \times M \tag{10}$$

5.2 Local Search

The cost $C(LC)$ of the local search is computed according to the number of operations executed at each iteration to re-allocate the atomic nodes in the critical path of a new solution. It is computed by:

$$C(LC) = \frac{N^3}{3} + Nlog_2 N + K \times (\frac{N(N-1)}{2}) + N_c \times PE + N \tag{11}$$

Figure 9: Mapping carried out at layer 3.

where $\frac{N^3}{3}$ is the number of operations executed to divide into phases the atomic nodes of TG_{init}, $Nlog_2N$ is the number of operations performed to sort the nodes within a phase (we suppose that in each phase there are N nodes), $K \times (\frac{N(N-1)}{2})$ is the number of operations to find the critical path (K is the number of iterations), $N_c \times PE$ are the operations executed to re-allocate the N_c nodes in the critical path onto PE metacomputer processors, and N is the cost payed to compute the ended time of all atomic nodes belonging to a new solution.

Since the expressions (10) and (11) can be approximate to $O(N^3)$, the algorithm time complexity, in the average case, results to be $O(N^3)$.

6 Performance Evaluation

Comparing mapping heuristics is generally a difficult task because of the different underlying assumptions in the original studies of each heuristic [13].

To evaluate HMM it was compared with three other graph-based mapping algorithms, the Cluster-M Mapping Algorithm (CM), Augmented Cluster-M Mapping (ACM), Lo's Max-Flow/Min-Cut ($MFMC$), Shen and Tsai's A^* Searching (AS), and with an exhaustive mapping algorithms (EM). EM carries out all possible M^N mappings, where N is the number of atomic tasks of an application, and M is the

Figure 10: Application DAG and metacomputer graphs used in test 1, 2, and 3.

number of machines in a metacomputer. Moreover, the mapping of a quantum chemistry application carried out by HMM was compared with that found by simulating the execution of the application on the same HC system.

The first four tests used as input the application DAGs and the metacomputer graphs from [3], and the results are shown in Gantt charts. For each task t_i the machine m_j on which it was allocated and the unit of time required for its execution (the numbers at the top of the chart) are shown.

Figure 10 shows the application DAG and the metacompurer graphs used in the first three tests conduced to compare HMM with CM, and EM.

Figures 11(a), (b), and (c) show the mapping of the application DAG of Figure 10(a) on the metacomputers 1, 2, and 3 of Figure 10(b), respectively.

In the first test (see Figure 11(a)) HMM converges to the optimal solution by analyzing only 25 mappings of the 2187 possible mappings analyzed by EM. It was obtained allocating the graph expanded at last layer. In order to reduce the task communications, t_0 and t_1 were allocate onto the same processor p_0.

In the second test (see Figure 11(b)) HMM found a suboptimal mapping to which corresponds a total application execution time of 15.1 units, which is worse than the optimal mapping by about 0.7%. It was carried out by allocating onto the same processor the more communicating tasks. HMM converges to the suboptimal solution after 27 possible mappings have been analyzed.

In the third test (see Figure 11(c)) HMM converged to the optimal solution after 16 possible mappings by allocating all the application tasks onto the highest speed

Figure 11: Gantt charts describing the mapping carried out in test 1, 2, and 3.

processor. In all the three tests the HMM execution required about 10 ms.

Figure 12 shows the application and the metacomputer graphs used in test four to compare HMM with CM, ACM, $MFMC$, and EM. The application consists of an MIMD code and a vector code (Gaussian elimination). The metacomputer used in the test models an MIMD and a vector machines. As in [3] we define the HMM affinity parameter so that the MIMD code was allocated on the MIMD machine and the vector code on the vector machine. To obtain a better view on the total application execution time we put together in the same Gantt the execution time of both the code types, even if $MFMC$, AS, EM do not treat heterogeneity in computation and machine types. Their mapping results were obtained by mapping separately the MIMD and Vector code onto the correspondent type of processor.

As shown in Figure 13, HMM and ACM carried out the same mapping to which corresponds to a total application execution time of 25.67 units with a worsening of 1.6% with respect to the optimal mapping.

To map the vector code EM analyzed 16,384 possible mappings to seek the optimal solution corresponding to a total application execution time of 25.17 units. HMM converges to the suboptimal solution by analyzing only 19 of the possible mappings. The HMM execution required 40 ms, the EMA execution required 1350 ms.

Mapping the MIMD code HMM converged to the suboptimal solution by analyzing only 21 of the 19,683 possible mappings analyzed by EM. The HMM's suboptimal solution has a worsening of 3.6$\%$ with respect to the optimal mapping. The HMM execution required 10 ms, the EM execution required 600 ms.

The longer execution time required by the exhaustive algorithm to map the vector code with respect to the MIMD code is due to the longer execution time required to compute the overall application execution time of the vector task.

The last test was conduced to evaluated HMM when used to map the real parallel

Figure 12: Application DAG and metacomputer graph used in test four.

quantum chemistry application cited in Section 4.1 (but implemented using a different parallel programming model). The application execution times and the machines utilization obtained by mapping the application using HMM were compared with those obtained by simulating the execution of the application on the same target metacomputer. The chemical simulation is composed by a set of independent processes each of them computes the propagation of particles in environments with different physical and chemical conditions. In Figure 14 the DAG represented the parallel program used to perform a simulated process is shown. A detailed description of this parallel application can be found in [33].

In the DAG the computational cost associated to each task was deduced analyzing previous works [33, 34], and the computational and communication costs of the tasks labeled $t_7 \div t_{11}$ and the related edges, respectively, vary of $\pm 20\%$ with respect to the value of physical entities (i.e. energy) used in a simulation.

The metacomputer used for conducing our experiments consisted of an 8-node machine with processor power of 35 MFlop/s fully connected at 8 Mbit/s, a 64-nodes hypercube multicomputer with processor power of 6 MFlop/s, and a network bandwidth of 2.4 Mbit/s, and a cluster of four workstations with processor power of 25 MFlop/s. All the machines were connected by a 0.25 Mbit/s LAN.

Previous works [33, 34] demonstrated that, for this application, good execution times were obtained performing the application structured according to a task farm programming model on clusters of processors. Therefore, for our tests we used the following configuration: (a) 8-node machine, partitioned into 2 clusters of 4 nodes each (machine 0), (b) 64-nodes hypercube multicomputer, partitioned into 4 clusters of 16 nodes each (machine 1), (c) 1 cluster of 4 workstations (machine 2).

Figure 13: Gantt charts describing the mapping carried out in test 4.

To simulate the application execution on this metacomputer configuration a program (called Task-Farm) which implements a task farm to dynamically assign the parallel tasks to the clusters, and estimates the application execution time has been used.

Figure 15 shows the execution times obtained by both HMM and the Task-Farm program elaborating problems of dimension from 1 to 32 chemical processes. Since the values assigned to the tasks $t_7 \div t_{11}$, and the related edges were randomly generated according to a normal distribution in the interval $\pm 20\%$ each run was performed 10 times, and the average execution time related to each problem dimension was plotted. It can be seen (Figure 15) that the application execution time noticeably smooths when the problem dimension increases, and with dimension equal to 32 (18.8) using HMM we have an application execution time equal to 255 units with respect to 660 units obtained by using the Task-Farm program.

In Figures 16 and 17 the workload distribution on the metacomputer machines is shown. When the number of processes is greater than 16 the use of HMM leads to a smaller utilization of the more power machine 0 and a higher utilization of the other machines. This better workload distribution leads to a smaller execution time with respect to that obtained simulating the application execution.

All tests were conduced on a dual processors Pentium III 800 Mhz with 512 MBytes of RAM.

Figure 14: DAG representing the chemical parallel application.

Figure 15: Application execution times obtained by using HMM and Task-Farm.

7 Conclusions

In this work we have presented a new static graph-based mapping algorithm, which allows us to find a suboptimal mapping of a parallel application onto a metacomputer in order to minimize the total application execution time. The algorithm exploits the information embedded in the parallelism forms used to implement an application, and uses the local search technique together with the tabu search meta-heuristic.

To evaluate the quality of the mapping carried out, several tests were conducted by using: (a) case studies previously used to evaluate other static mapping algorithms which use different leading techniques, (a) a quantum chemical application, (c) an exhaustive mapping algorithm. The metric used to compare the mappings obtained was the application execution time.

The tests' results show that our algorithm performs better than or the same as the other mapping algorithms, and that it has obtained good result mapping a large real

Figure 16: Machines utilization obtained by mapping the application using HMM.

Figure 17: Machines utilization obtained by simulated the application execution.

application on an heterogeneous platform composed of many processors. The comparisons with the mapping obtained by an exhaustive mapping algorithm showed that our algorithm always carried out mapping very near to the optimal solution. This is because HMM allocates parallel tasks in a way that optimizes the execution of each form of parallelism used within a parallel application.

The mapping quality carried out by HMM can be enhanced by introducing several improvements such as the exploitation of the native mapping mechanisms available on each metacomputer machine, and a better estimation of the code-machine affinity values, by using data from specific benchmarks. In addition, the introduction of dynamic mapping functionalities to permit the migration of parallel tasks among the metacomputer machines can enhance the metacomputer's throughput.

8 Acknowledgements

The authors would like to thank Domenico Laforenza for his contribution to design the first version of this algorithm, and Fabrizio Silvestri for his suggestions about the algorithm complexty. This work was funded by the Italian Ministry of Education, University and Research (MIUR) as part of the National Project MURST 5% 1999 Grid Computing: Enabling Technology for eScience.

References

[1] R. F. Freund, H. J. Siegel, *Heterogeneous Processing*,IEEE Computer, 26(6):13-17,1993.

[2] A. Khokhar, V. K. Prasanna, M. Shaaban, C. L. Wang, *Heterogeneous Processing: Challenges and Opportunities*, Computer, vol. 26, no. 6, pp. 18-27, June 1993.

[3] Mary Mehrnoosh Eshaghian, *Heterogeneous Computing*, Artech House Publishers,1996.

[4] C. E. Cattlet, L. Smarr, *Metacomputing*, Communication of the ACM, 53(6):45, June 1992.

[5] I. Foster, C. Kesselman, *The Grid: blueprint for new computing infrastructure*, Morgan Kaufmann Publishers,1998.

[6] D. S. Conwell, R. F. Freud, *Superconcurrency*, Supercomputing Review, pp. 47-51, October 1990.

[7] C. Leangsuksun, J. Potter, *Designs and experiments on heterogeneous mapping heuristics*, Proceeding of Workshop on Heterogeneous Computing, pp. 17-22, 1994.

[8] V. M. Lo, *Heuristic Algorithms for Task Assignment in Distributed Systems*, IEEE Transaction on Computers, 37(11), 1384-1397, November, 1988.

[9] H. S. Stone, *Multiprocessor scheduling with the aid of network flow algorithms*, IEEE Transactions on Software Engineering, 3(1):85-93, January 1997.

[10] C. Shen, W. Tsai, *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minmax Criterion*, IEEE Transaction on Computers, C-34(3):197-203, March 1985.

[11] N. J. Nilson, *Problem Solving Methods in Artificial Intelligence*, McGrow-Hill, New York, 1971.

[12] R. F. Freud, S. Conwell, *Superconcurrency: A form of distributed heterogeneous supercomputing*,Supercomputing Review, pages 47-51, October, 1990.

[13] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Masheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R. F. Freud. *A Comparison Study of Static Mapping Heuristic for a Class of Meta-task on Heterogeneous Computing System*, 8th Heterogeneous Computing Workshop, April 1999.

[14] Y. T. Wang, R. J. T. Morris, *Load Sharing in Distributed Systems*, IEEE Transaction on Computers, C-34(3), 204-217, March, 1985.

[15] G. C. Fox, R. D. Williams, P. C. Messina, *Parallel Computing: Works!*, Morgan Kaufmann Publishers, Inc., 1994.

[16] A. J. G. Hey, *Experiments in MIMD Parallelism*, Proceedings of Int. Conf. PARLE 89, June 1989.

[17] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, V. Sunderam, *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Network Parallel Computing*, MIT Press, 1994.

[18] B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti, M. Vanneschi, *P^3L: a structured parallel programming language and its structured support*, Concurrency: Pratice and Expirence, 7(3):225-255, May, 1995.

[19] B. Bacci, M. Danelutto, S. Pelagatti, M. Vanneschi, *$SkIE$: A heterogeneous environment for HPC applications*, Parallel Computing 25(13-14):1827-1852, 1999.

[20] D. B. Skillicorn, *Models for Practical Parallel Computation*, International Journal of Parallel Programming, 20(2):133-158, April, 1991.

[21] M. Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation*, Mit Press, 1989.

[22] J. Darlington, M. Ghanem, and H. W. To, *Structured parallel programming*,In Programming Models for Massively Parallel Computers, IEEE Computer Society Press, September 1993.

[23] Nicholas S. Bowen, Cristos N. Nikolaou, Arif Ghafoor, *On the assignment problem of arbitrary process system to heterogeneous distributed computer systems*, IEEE Transactions on Computers, 41(3):257-273, March 1992.

[24] Ammar H. Alhusaini, Viktor K Prasanna, *A Unified Resource Scheduling Framework for Heterogeneous Computing Environment*, 8th Heterogeneous Computing, April 1999.

[25] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhry, Antony A. Maciejewski, *Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach*, Journal of Parallel and Distributed Computing, 47, 8-22, 1997.

[26] H. H. ALI, H. El-Rewini, T. G. Lewis, *Task Scheduling in Parallel and Distributed Systems*, PTR Prentice Hall, June 1994.

[27] F. Glover, M. Laguna, *Tabu Search*, Kluver Academic Publishers, Boston, MA, June 1997.

[28] William Gropp, Ewing Lusk, Anthony Skjellum, *Using MPI Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, 1999.

[29] S. Chen, M. Eshaghian, A. Khokhar, M. Shaaban, *A Selection Theory and Methodology for Heterogeneous Supercomputing*, Proceeding of Workshop on Heterogeneous Processing, pp. 15-22, 1993.

[30] R. Baraglia, D. Laforenza, A. Panciatici, F. Ravaglia *A Suboptimal Mapping of Parallel Applications on Metacomputers*, Proc. of the IASTED Intern. Conference Parallel and Distributed Computing and System (PDCS' 99), November 3-6, 1999, Cambridge, Massachusetts, USA.

[31] M. Tan, J. K. Antonio, H. J. Siegel, YA Li, *Scheduling and Data Relocation for Sequentially Executed Subtasks*, Proceeding of Workshop on Heterogeneous Computing, pp. 109-120, 1993.

[32] R. Baraglia, R. Ferrini, D. Laforenza, A. Laganá, *On the optimization of a pipeline model to integrate a reduced-dimensionality Schrodinger equation for distributed memory architectures*, The International Journal of High Performance Computing Applications, Volume 13, No. 1, Spring 1999, pp. 49-62.

[33] R. Baraglia, R. Ferrini, D. Laforenza, A. Laganá, *An optimized task-farm model to integrate reduced dimensionality Schrodinger equation on distributed memory architectures*, Future Generations Computer Systems, 15(1999) 497-512.

[34] R. Baraglia, R. Ferrini, D. Laforenza, A. Laganá, *Quantum Reactive Scattering Calculations*, High Performance Cluster Computing Volume 2, ed. Rajkumar Buyya, Prentice Hall, 1999.