Clustering and Reassignment-based Mapping Strategy for Message-Passing Architectures

M.A. Senar, A. Ripoll, A. Cortés and E. Luque
Departament d'Informàtica
Unitat d'Arquitectura d'Ordinadors i Sistemes Operatius
Universitat Autònoma de Barcelona
08193 - Bellaterra (Barcelona), SPAIN
e-mail: iinfd@cc.uab.es

Abstract[1]

A fundamental issue affecting the performance of a parallel program is the assignment of tasks to processors in order to get the minimum completion time. In this paper, we present a compilation-time two-stage mapping strategy (denoted as CREMA) used for mapping arbitrary programs (modeled as TIG graphs) onto message-passing parallel systems with any architecture. In contrast to most of the other approaches found in the literature, CREMA is not tied to any particular architecture or any specific algorithm. The first stage is based on task clustering and task reassignment algorithms that contract the original task graph. The second stage takes the contracted graph and tries to successfully match the physical properties of the target system. It has been evaluated for a wide range of both regular and irregular graphs that correspond to some well-known real applications. The results show that CREMA provides a good trade-off between mapping quality and computational complexity.

Keywords: Mapping problem; Task allocation; Task assignment; Graph partitioning; Clustering Heuristics.

1. Introduction

The problem of assigning each task of a parallel program to processors of a parallel system has a major impact on the resulting performance. This problem arises in all areas of parallel and distributed computation, where programs are decomposed into tasks or processes which must be assigned to processors for execution. In many cases where persistent applications are used (applications that will be used without modification for long periods of time like those in embedded systems), a suitable amount of information on the program behavior can be deduced *a priori* and, therefore, the user or the system (compiler) can exert explicit control over the assignment of each task. This static assignment needs only to be performed once and it does not introduce any run-time overheads.

In this paper we present a compilation-time two-stage mapping strategy called CREMA (Clustering-and-REassignment based Mapping Algorithm). CREMA can be used for mapping arbitrary parallel programs onto a message-passing parallel system with any architecture. In our approach, parallel programs are represented as a collection of tasks that can be modeled by an undirected graph (called Task Interaction Graph: TIG). In contrast to most of the other strategies found in the literature [1][2], the proposed strategy is not tied to any particular architecture or any specific algorithm.

The paper is organized as follows. Section 2 describes the models adopted in our work and formulates the mapping problem considered in CREMA. Section 3 presents a detailed description of the proposed mapping strategy. Section 4 presents the experimental results. Finally, section 5 summarizes the main contributions of this work.

2. Problem formulation

In the TIG model, $G_p = \{N_p, E_p\}$ is an undirected graph where nodes, $t_i \in N_p$, correspond to parallel tasks and edges, $v_{ij} \in E_p$, correspond to intertask communication actions [3]. In this model, temporal dependencies in the execution of tasks are not explicitly addressed: all the tasks are considered simultaneously executable and communications can take place at any time during the computation.

Weights may be associated with nodes and edges. The weight w_i of each node t_i describes the computational cost of task t_i and the weight $c_{i,j}$ of each edge $v_{i,j}$ describes the communication cost between the tasks t_i and t_j. For this

[1] *This work has been supported by CICYT under contract TIC 95-0868*

study we assume that the information about the computation and communication costs are somehow available and that these costs are expressible in some common unit of measurement.

Parallel architectures with a direct interconnection network are usually described by an undirected connected graph $G_a = \{N_a, E_a\}$, where each node $p_i \in N_a$ denotes a processor and each edge $e_{i,j} \in E_a$ denotes a bi-directional physical communication link between two processors. For such architectures, a distance function $d(p_i,p_j)$ can be defined for all pairs of processors that denotes the communication cost between processor p_i and p_j. This cost is considered to be the number of links that are traversed by messages exchanged between p_i and p_j. Additionally, we assume that the communication cost between two tasks assigned to the same processor is negligible.

The objective in mapping a TIG graph to a G_a graph is the minimization of the expected execution time of the parallel program on the target architecture. Thus the mapping problem can be seen as an optimization problem by defining a cost function, *cost_map*, that is directly related to the execution time and must be minimized.

We use a *minimax cost* function, where the cost incurred by each processor (computation cost + communication cost) for a certain mapping *f* is estimated, and the maximum cost between all processors is to be minimized. The cost of each processor p_t (*cost* (p_t)) is the total cost due to computation and communication of all tasks mapped onto it, and is defined as:

$$cost(p_t) = \sum_{t_i \mid f(t_i) = p_t} w_i + \sum_{\substack{t_i \mid f(t_i) = p_t, \\ t_j \mid f(t_j) \neq p_t}} c_{i,j} * d\big(f(t_i), f(t_j)\big)$$

$f(t_i)$ being the processor to which t_i is mapped and $d(f(t_i), f(t_j))$ denotes the distance between processors to which t_i and t_j are mapped respectively. This cost function assumes that processors lack special hardware required for a maximum overlap between communication and computation. The minimax cost function used to evaluate the quality of a mapping instance f is:

$$cost\_minimax(f) = \max_{\forall i}\big(cost(p_i)\big)$$

According to the above definitions, the mapping problem is defined as follows. Given a TIG graph $G_p = \{N_p, E_p\}$ and a parallel architecture $G_a = \{N_a, E_a\}$, the question is to find a mapping function f: $G_p \rightarrow G_a$ which assigns each node of the graph G_p to a unique node of the graph G_a, and minimizes *cost_minimax(f)*.

3. Clustering and REassignment based Mapping Algorithm (CREMA)

Our approach mapping is achieved by using a two-stage method (see figure 1):

Contraction of the task graph to a smaller graph (when the number of tasks exceeds the number of processors),

Physical mapping of the contracted graph to K physical processors.

Contraction reduces the original task graph by means of a clustering step followed by a reassignment step. As a result of the contraction stage, the task graph is reduced to a set of Q clusters ($Q \leq K$, K being the number of processors). Physical mapping assigns each cluster on one processor in three steps: first the cluster graph is embedded on the processor graph; then, a cluster reassignment step follows in order to map intensively communicating clusters closely to each other, i.e. trying to match intensive communications (thick lines in figure 1) with architecture links; finally, individual tasks in each cluster are also reassigned to additionally improve the cost of the final mapping (task reassignment step). We describe our algorithms for each stage below.

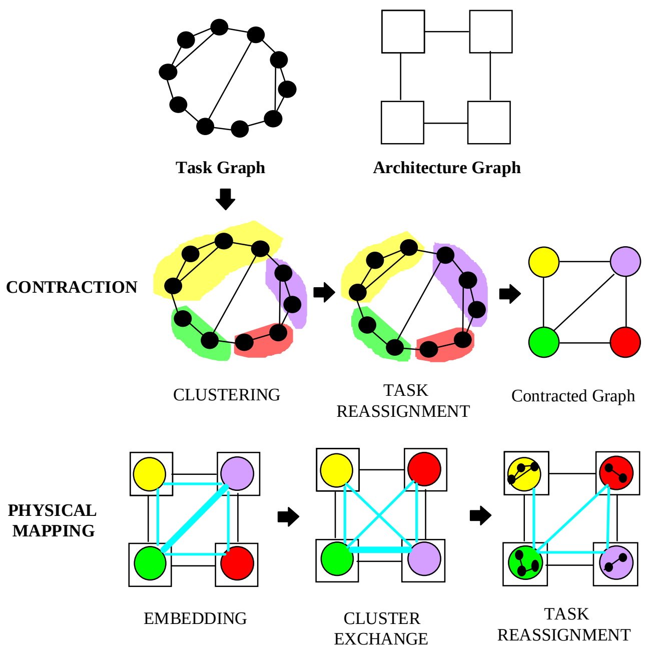

Figure 1. CREMA mapping strategy overview

3.1. Contraction

Contraction is defined as a mapping of tasks onto clusters, i. e., all tasks assigned in a cluster will execute in the same processor in the target architecture. Our contraction stage has been designed as a mixed strategy, therefore, it consists of two steps: a greedy algorithm, based in clustering, followed by an iterative algorithm, based on task reassignment.

In general, mixed heuristics that appear in the literature use either a greedy strategy that is too simple or that do not always take into account the graph connectivity in their decisions. These characteristics are detrimental to the final assignments found by these strategies because their results are far from optimum or they have to spend a lot of time in the iterative phase. Our clustering algorithm (CA) was designed to allow a gradual reduction of those edges with the biggest communication cost while, according to the minimax cost function, keeping the obtained clusters balanced.

Step 1: Clustering

This initial step assumes that each task is its own cluster. At each step, the algorithm reduces by one the number of clusters by merging two of them. This greedy process could end either when the number of clusters obtained is equal to the number of processors or when the number of tasks is less than the number of processors. A merging operation implies zeroing an edge cost connecting two clusters, and a zeroing is accepted as long as the cost of the most loaded cluster is reduced. When the most loaded cluster does not have such an edge, then the edge chosen to be zeroed is the one that generates a new cluster with minimum cost. The algorithm is able to reduce the original graph in such a way that useless parallelism exhibited by the application is removed. In that case, the number of final clusters could be less than the number of processors. Therefore, clustering steps are performed while the cost function (*cost_minimax(f)*) is further reduced at every step.

Step 2: Reassignment

After the clustering step, a reassignment algorithm will refine the allocation already found. This algorithm looks for the most loaded cluster and tries to shift individual tasks from it to another cluster, in such a way that its work-load is reduced and, as a consequence, the cost of mapping is also reduced. This process is repeated until no improvement is achieved in minimizing the load of the most loaded cluster. When shifting tasks, two different strategies have been followed: the first one (denoted as Clustering and Reassignment by Movements: CRM) only performs individual movements of tasks between clusters, and the second one (denoted as Clustering and Reassignment by Movements and Exchanges: CRME**)** also exchanges pairs of tasks between different clusters. The CRME algorithm works by iterating a sequence of task movements followed by an iteration of task exchanges. This sequence of movements and exchanges is performed until no improvement is obtained in the cost function.

Complexity of the clustering and the reassignment algorithms are respectively O(NlogM) and O(N^2), N being the number of nodes in the graph and M being the number of edges. A detailed description of the algorithms and their complexity can be found in [4].

3. 2. Physical mapping

This stage is required when processors are not completely connected (for instance, in architectures such as hypercubes or meshes) because an increase of the cost appears due to communications across some kind of direct network. At this stage, we have Q clusters (or virtual processors) and K physical processors. Three steps are followed to achieve the physical mapping.

Step 1: Cluster Embedding (CE)

First, there is an embedding step that uses a greedy algorithm to place highly communicating clusters on adjacent neighbors in the architecture graph. Figure 2 shows a high-level description of the embedding algorithm. Given a contracted graph, the embedding algorithm first maps the cluster with the biggest communication volume. Then it constructs a list of all the edges of that cluster sorted by weight, it takes the first edge in the list and assigns the cluster connected to the other end point. The cluster is assigned to the closest free processor. Once the cluster is assigned, its edges are included in the sorted list of edges and, again, the first edge in the list is picked up. The algorithm ends when the list of edges is empty, i. e., all the clusters have been assigned. Every cluster is mapped with a complexity O(K), K being the number of processors. As the number of clusters is equal to K, the overall complexity of the embedding algorithm is O(K^2).

The OREGAMI mapping tool [5] is an example of where physical mapping is achieved by means only of an embedding algorithm. However, OREGAMIS’s embedding algorithm may assign nodes randomly at some steps. This situation never occurs in our strategy because our algorithm always tries to assign a node at each step as closely as possible to the rest of nodes already assigned.

```
Edge_list = ∅
Let $Q_0$ be the cluster with the biggest
        communication volume
Map $Q_0$ to $P_0$
For all edges $v_{i,j}$ of $Q_0$ do
   Edge_list = Edge_list + { $v_{i,j}$}
While Edge_list ≠ ∅ do {
    Let $v_{j,k}$ be the biggest edge in Edge_list
    If $Q_k$ is not mapped then c_candidate = $Q_k$
      else If $Q_j$ is not mapped then
              c_candidate = $Q_j$
    Edge_list = Edge_list - {$v_{j,k}$}
    If c_candidate ≠ NULL then {
         min_cost = ∞
         For all procesors $p_q$ not used {
             cost = evaluate communication cost
                    of mapping c_candidate onto $p_q$
             If (cost < min_cost) then {
                   p_candidate = $p_q$
                   min_cost = cost
             }
         }
         Map c_candidate onto p_candidate
         For all edges $v_{i,j}$ of c_candidate do
              Edge_list = edge_list + {$v_{i,j}$}
  }
```

Figure 2. Cluster Embedding (CE) algorithm

Step 2: Cluster Reassignment (CR)

Once all the clusters are assigned, a second step is followed. This step is based on an iterative reassignment algorithm that is a modified version of the Bokhari’s heuristic algorithm [6]. In contrast to Bokhari’s original formulation, our algorithm takes into account the communication of each edge and does not use probabilistic jumps to skip local minima. Therefore, the number of iterations done by our algorithm is significantly reduced.

This step first sorts all the clusters according to their communication volume and then it tries to reduce the global communication cost (see figure 3). At each step, it picks up the first cluster in the list as a candidate cluster. Then, it searches among all the other clusters to find the one (called *max_cluster*) that will provide a maximum reduction of the global communication cost when the candidate cluster and *max_cluster* are exchanged. When no improvement is achieved by exchanging the first cluster in the list, then the algorithm scans the list until a candidate cluster is found that produces a secondary exchange which also reduces the global communication cost. The algorithm ends when there is no exchange that reduces the global communication cost.

One cluster exchange is found in the inner loop with a worst case complexity of $O(K^2)$. Therefore, if we assume that the total number of cluster exchanges performed is bound by a constant number of times, K, (which is generally the case from experimental observations), the time complexity of the CR algorithm is $O(K^3)$.

```
Let f be the original mapping
Sort all the clusters in increasing order of
         communication volume
Let global_comm be the sum of all communication
         volumes
Let c_candidate be the first cluster in the list
End = FALSE
Do {
    best-cost = global_comm
    For all processors $p_q$ do {
         cost-exchange = evaluate communication cost
         of mapping when exchanging c_candidate and
         the cluster of $p_q$
         If (cost_exchange < best_cost) then {
              max_cluster = $p_q$
              best_cost = cost_exchange
         }
         If (best_cost < global_cost) then {
              Modify the current mapping f by
                exchanging max_cluster and c_candidate
              global_comm = best_cost
              c_candidate = first cluster in the list
            }
            else {
                c_candidate = next cluster in the list
                If (c_candidate = = NULL) then
                     end = TRUE
             }
       }
}
while not (End)
```

Figure 3. Cluster Reassignment (CR) algorithm

PYRROS [7] is a mapping tool in which physical mapping is achieved by means of a reassignment algorithm alone, and which also uses a modified version of the Bokhari’s algorithm. The version of PYRROS reduces the complexity of the algorithm by fixing the number of iterations, but it still needs to start from a random initial allocation. In our approach, the reassignment algorithm reduces the number of iterations because it starts from a rather good initial mapping (coming from the embedding step) instead of an arbitrary initial mapping. Moreover, the final mapping is further improved by the task reassignment step (see step 3). Therefore, the quality of the mappings obtained in our approach will be better than those obtained using an embedding algorithm alone, or a reassignment algorithm alone, and the overall complexity will still be dominated by the complexity of the contraction stage.

Step 3: Task Reassignment (TR)

The last step in the physical mapping stage is a task reassignment algorithm similar to the reassignment algorithm described in the contraction stage. In contrast to the two previous steps that work with entire clusters, this algorithm works with individual tasks within each cluster and it tries to move them from the most loaded cluster to another cluster. With these movements the algorithm looks for a reduction in the cost of the most loaded cluster and, therefore, a reduction of the cost of the mapping. This algorithm is similar to that followed at CRM that can be found in [4]. As in the contraction stage, this task reassignment algorithm has a time complexity of $O(N^2)$.

4. Experimental study

A set of experiments were developed to evaluate the performance obtained by our CREMA strategy. The evaluation was performed by considering several graphs of different structures, sizes and granularities. In these experiments we used regular and irregular task graphs.

The set of regular graphs corresponds to regular structures (meshes, trees and rings) that usually appear in real parallel applications. The number of tasks of these TIG graphs ranged from 255 to 500 nodes. Moreover, we introduced a parameter CCR (Computation Communication Ratio) in order to analyze the algorithm performances under different ratios that all the nodes of a graph exhibit. Our simulations have been conducted for two CCR values: CCR = [5, 15] characterizing computation intensive applications (CR: CoaRse grain graphs) and CCR = [0.8, 1.2] characterizing balanced communication and computation applications (MD: MeDium grain graphs). The node and edge weights were uniformly distributed over [5,500].

The irregular graphs were obtained from a set of 9 Directed Acyclic Graphs (DAGs) that correspond to real applications (systolic matrix multiplication, Gauss back-substitution, Poisson's equation, etc.). The DAG sizes ranged from 365 to 3000 nodes and they were transformed to Task Interaction Graphs (TIGs) applying the DSC (Dominant Sequence Clustering) algorithm [7]. According to the previous definition, all irregular TIG graphs belong to the category of medium grain graphs. Both regular and irregular task graphs were mapped onto 8, 16 and 32 processors.

4.1. Evaluation of the contraction stage

We have evaluated the effectiveness of our contraction stage by comparing CA, CRM and CRME algorithms to other contraction strategies from the literature. The strategies used in the comparison cover the range of different complexity categories. There were two simple greedy algorithms: LPTF (Largest Processing Time First) and LGCF (Largest Global Cost First) with low complexity [8]; two iterative heuristics: SA (Simulated Annealing) and TS (Tabu Search) with the highest complexity [8]; and a mixed heuristic: EDTR (Even Distribution and Task Reassignment [9]) with an intermediate complexity between the other two.

The LPTF strategy obtained the worst allocations in all the examples (its mappings had the biggest costs). Therefore, we used the results of LPTF as a reference point to evaluate the improvement in the cost function achieved by the other strategies.

Figure 4 summarizes the experimental study by showing graphically the overall improvement achieved by each heuristic versus LPTF (T_{LPTF}/T_P, T_{LPTF} being the mapping cost for LPTF and T_P being the mapping cost for the other heuristics), when mapping both regular and irregular graphs. The average improvement ratio of CRME and CRM mappings over LPTF mappings was, respectively, 1.9 and 1.89 for regular graphs, and 1.46 and 1.41 for irregular. Compared to the best iterative heuristic (SA) and to the mixed heuristic (EDTR), SA obtained an improvement of 1.82 for regular graphs and 1.45 for irregular graphs, and EDTR obtained an improvement, respectively, of 1.44 and 1.41. A more detailed description of the algorithms of all the strategies used in this comparison and the experimental framework for performance assessment could be found in [4]. Additionally, table I shows the average running times ($\bar{t}$ column) of all these strategies. The σ_N, *Max* and *min* columns show, respectively, the standard deviation from the average, the maximum time and the minimum time measured for a given strategy. As is seen in Table I, both CRM and CRME algorithms are faster than SA, TS and EDTR, CRM being slightly slower than LPTF and LGCF.

As CRM provides the best trade-off between solution quality and time complexity, it is the contraction algorithm used in our global mapping strategy CREMA.

4.2. Evaluation of the physical mapping stage

To evaluate the effect of the physical mapping stage on the mapping cost we used the same set of regular and irregular graphs mentioned in the contraction stage. Once they were contracted by the CRM algorithm, the contracted graph passed through the three steps

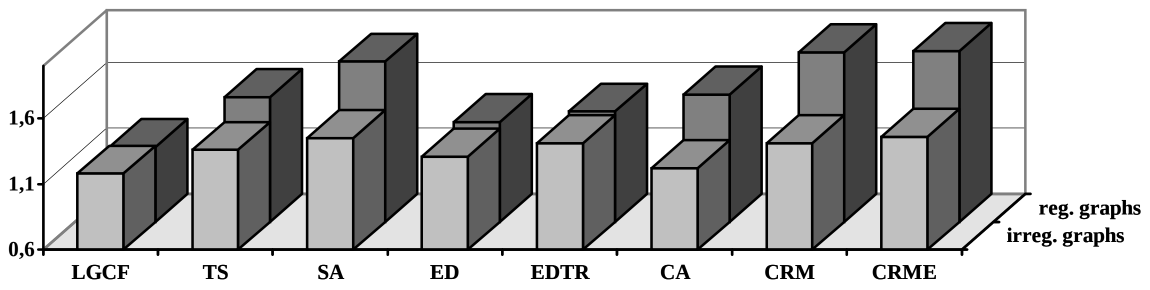

Fig. 4. Performance comparison of mapping strategies versus LPTF

	REGULAR GRAPHS				IRREGULAR GRAPHS			
	$\bar{t}$	σ_N	Max.	min.	$\bar{t}$	σ_N	Max.	min.
LPTF	0.1	0.04	0.2	0.1	0.1	0.02	0.2	0.1
LGCF	0.16	0.04	0.2	0.1	0.3	0.1	0.5	0.1
TS	46.2	78.4	517	4	749.3	861.2	3691	75
SA	2682.4	699.9	4784	1778	2513.9	864	4136	1237
ED	1.3	0.33	1.5	0.6	2.18	1.04	3.9	0.4
EDTR	23.6	39.9	285	0.9	18.5	39.6	188	1.9
CA	0.8	0.39	1.3	0.4	1.4	0.7	2.2	0.3
CRM	1.5	0.5	2.2	0.6	2.1	0.3	2.6	1.2
CRME	4.3	3.4	13.9	0.6	13.6	23.7	72.4	1.7

Table I. Average time (in seconds) spent by the mapping strategies

mentioned above. Graphs were mapped onto three different topologies: hypercube, wrap-around mesh and ring. The number of processors was 8, 16 and 32.

For each step of the physical mapping we evaluated the mapping cost achieved at that step and we compared it with the cost obtained by the CRME algorithm. The contraction result of the CRME algorithm was taken as a reference cost and we evaluated the cost increase due to the physical mapping stage. In that sense, the mapping cost of the contracted graph is the cost for mapping the task graph onto an "ideal" physical architecture. When we have an actual architecture the mapping cost will be increased once the contracted graph is mapped onto it.

Table II summarizes the results and shows the average increase of the cost function obtained by the three steps. These values have been computed using T_P/T_{CRME}, where T_P is the mapping cost found by each heuristic and T_{CRME} is the mapping cost for the CRME heuristic. The acronyms CE, CR and TR denote respectively the three steps followed at the physical mapping stage, namely Cluster Embedding, Cluster Reassignment and Task Reassignment. The AVG row gives the average increase for all the regular graphs of a given category.

It can be observed that the increase of the mapping cost is not very significant for regular graphs mapped to either hypercubes or meshes. Additionally, for coarse grain graphs, results tend to be slightly better than for medium grain graphs because communication costs do not have much influence on the cost function. On the one hand, when medium grain graphs are contracted, the computation and the communication cost terms in the cost function are similar. On the other hand, when coarse grain graphs are contracted, the computation cost term for each cluster is relatively larger than its communication cost term. As a consequence, when the contracted graph is mapped physically, the increase of the mapping cost will be relatively larger in medium grain graphs than in coarse grain graphs because computation costs remain the same but communication costs are eventually multiplied by distances bigger than one. The increase for irregular graphs is also bigger than for regular graphs because the number of edges of the contracted graph of the former is significantly bigger than the number of edges of the contracted graph of the latter. Table III shows the average number of edges of both regular and irregular contracted graphs (column *N. edg.*). It also shows the average number of edges of the cluster that has the biggest number of edges because we derive from our experiments that the final cost depends directly on this value (column *Max.*). Finally, ring architectures obtain the worst results because they have a small number of links, and distances between processors are large.

Additionally, table IV shows the average execution times (*AVG* column) as well as the maximum execution

Arquit.	Graph	Granul.	CE	CR	TR
HYPER.	Reg.	MD	1.29	1.22	1.15
		CR	1.96	1.06	1.05
		AVG	1.18	1.14	1.1
	Irreg.	MD	1.49	1.48	1.4
MESH	Reg.	MD	1.32	1.21	1.16
		CR	1.08	1.06	1.06
		AVG	1.2	1.13	1.11
	Irreg.	MD	1.57	1.45	1.44
RING	Reg.	MD	2.24	1.83	1.61
		CR	1.29	1.27	1.18
		AVG	1.76	1.55	1.39
	Irreg.	MD	3.00	2.78	2.5

Table II. Average increase of mapping cost in the physical mapping stage (T_p/T_{CRME})

	Number of clusters					
	8		16		32	
TIG	N.edg.	Max.	N.edg.	Max.	N.edg.	Max
Reg.	9,6	3,8	19,3	4,4	45,4	5,3
Irre.	16,9	6	46,4	10,6	124,6	12,9

Table III. Connectivity characteristics of contracted graphs

time (*MAX* column) for all the steps at the physical mapping stage when graphs were mapped onto all the 32 processor architectures. In general, the time complexity of all the steps is lower than the total time spent at the contraction stage (always less than 1 second), the overall complexity of CREMA being moderate.

Mapping heuristics for TIGs have been evaluated traditionally by means of cost functions. However, they are a simplification of the real situation because data dependencies, message latencies and network contentions are neglected. An additional set of experiments was conducted in order to assess the existence of a reasonable correlation between mapping costs and actual execution times. Our experiments were based on the set of DAGs mentioned above, which were simulated in the following architectures: fully-interconnected, wrap-around mesh, hypercube and ring, with 8, 16 and 32 processors.

A linear regression was evaluated between the set of cost function values for each graph and the set of execution times obtained in the simulation of the execution of the corresponding graph. Strong correlations were obtained for 8 of the 9 examples; only for one example the correlation was moderate. These results suggest that the TIG model without, however, considering dynamic parameters, provides an acceptable estimation of the execution time.

	Regular Graphs		Irregular Graphs	
	AVG	MAX	AVG	MAX
CE	0.01	0.02	0.01	0.02
CR	0.29	1.41	0.34	0.77
TR	0.42	1.45	0.57	2.64

Table IV. Computation time (in seconds) for physical mapping steps

5. Conclusions

We have presented a new task assignment strategy (CREMA) for solving the mapping problem in message-passing parallel computers when parallel applications are modeled by a Task Interaction Graph (TIG).

The experiment results confirm that the quality of solutions obtained by our atrategy was similar or even superior to those obtained by the other heuristics for both regular and irregular task graphs. Moreover, a correlation study between the actual execution times of parallel applications and the values of their corresponding mapping costs was also performed. In most of the examples used, a strong correlation was obtained. If a new model combining dependences and the TIG is used, the applicability of the strategy can thereby be extended.

References

[1] B. Robic and B. Vilfan, “Improved schemes for mapping arbitrary algorithms onto processor meshes”, Paralle Computing, 22, (1996), pp. 701-724.

[2] S. Lor, H. Shen and P. Maheshwari, “Divide-and-conquer mapping of parallel programs onto hypercube computers”, J. of Systems Architecture, 43 (1997), pp. 373-390.

[3] M. G. Norman & P. Thanish, “Models of machines and computation for mapping in multicomputers”. ACM Computer Surveys, 25 (3) (1993), pp. 263-302.

[4] M. A. Senar et alter, “Performance comparison of strategies for static mapping of parallel programs”, Lecture Notes in Comp. Sci. (1225), Springer Verlag, (1997), pp. 575-587.

[5] V. M. Lo et alter, “OREGAMI: tools for mapping parallel computations to architectures”, Int’l J. of Parallel Programming, Vol. 20, no. 3, pp. 237-270, 1991.

[6] S. H. Bokhari, “On the mapping problem”, IEEE Trans on Computers, Vol. C-30 No. 3, (1981), pp. 207-214.

[7] T. Yang and A. Gerasoulis, “PYRROS: Static task scheduling and code generation for message pasing multiprocessors”, Proc. 6th ACM Int’l Conf. Supercomputing (ICS92), ACM Press, New York, N. Y., pp. 428-437, 1992.

[8] J.P. Kitajima et alter, “ANDES: Evaluating Mapping Strategies with Synthetic Programs”, J. of Systems Architecture, 42 (1996), pp. 351-365.

[9] Shen Shen Wu & D. Sweeting, “Heuristic algorithms for task assignment and scheduling in a processor network”. Parallel Computing 20 (1994), pp. 1-14.