Comput Syst Sci & Eng (2003) 4: 217–222
© 2003 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

Heuristic algorithm based on a genetic algorithm for mapping parallel programs on hypercube multiprocessors

Jose Aguilar

CEMISID. Dpto. de Computación, Facultad de Ingeniería, Universidad de los Andes, Av. Tulio Febres, Mérida, Edo. Mérida-Venezuela
Email: aguilar@ing.ula.ve

In this work, we propose a heuristic algorithm based on Genetic Algorithm for the task-to-processor mapping problem in the context of local-memory multiprocessors with a hypercube interconnection topology. Hypercube multiprocessors have offered a cost effective and feasible approach to supercomputing through parallelism at the processor level by directly connecting a large number of low-cost processors with local memory which communicate by message passing instead of shared variables. We use concepts of the graph theory (task graph precedence to represent parallel programs, graph partitioning to solve the program decomposition problem, etc.) to model the problem. This problem is NP-complete which means heuristic approaches must be adopted. We develop a heuristic algorithm based on Genetic Algorithms to solve it.

Keywords: heuristic algorithm, genetic algorithm, hypercube multiprocessors, parallel programs

1. INTRODUCTION

The advent of cost-effective VLSI components in the past few years has made feasible the commercial development of massively parallel computers with hundreds of processors. Because of appealing properties such as node and edge symmetry, logarithmic diameter, high fault resilience, scalability, and the ability to host popular interconnection networks, namely ring, torus, tree and linear array, *hypercube multiprocessors* has been the focus of many researchers over the past few years [2, 5, 6]. This topology has result in several commercial product (Origin2000, Intel iPSC, NCUBE/10, Caltech/JPL, etc.).

Conceptually, the hypercube interconnection network is a multidimensional binary cube with a processor or procesors cluster at each of its vertices. An n-dimmensional hypercube has 2^n processors or processor clusters and $n2^{n-1}$ links. Each processor or processors cluster has its own local memory and interprocessor communication is done by explicit message passing directly or through some intermediate processors. This type of architecture is more readily scaled up to very large numbers of processors than multiprocessors designs based on globally shared memory [4, 6].

The effective exploitation of the potential power of this type of parallel architecture requires efficient solutions to the task-to-procesor mapping problem. The problem is that of optimally allocating the tasks of parallel program among the processors in order to minimize the execution time of the program. The mapping problem is NP-complete [4, 7, 8, 9] which means heuristics approaches must be adopted. The mapping of the tasks to processors may either be performed statically (before program execution) or dynamically in an adaptive manner as the parallel program executes. The appropriate approach depends on the nature of the model of the parallel programs. If the characterization of the parallel program, i.e. the dependence between tasks and their execution time can be accurately estimated a priori, then a static approach is more attractive, since the mapping computation need only be performed once. We only consider static mapping scheme in this paper.

In this work, the task-to-processor static mapping problem in the context of a local-memory multiprocessors with a hypercube interconnection topology is solved using an algorithm based on Genetic Algorithms. We will model a parallel program execution as an acyclic directed graph whose nodes represent the tasks with known (or estimated) computation times, and whose arcs represent the precedence relations between tasks, that is the explicit execution dependences [7–9]. This graph is called *tasks graph*. Our approach is composed by two phases: in the first phase an initial tasks graph k-partitioning is done in a manner of minimize communication volume and load imbalance cost between the subgraphs (clusters), where *k* is the number of processors. In the second phase, task clusters are assigned among processors of the system with hypercube interconnection topology in a manner that minimize the communication distance between clusters.

This paper is organized as follows. In section 2 we formalized the mapping problem. In section 3 the theoretical basis of Genetic Algorithms are reviewed. Then, we present our algorithm. In section 4 we compare the effectiveness of our scheme with previous work [5]. Remarks concerning future work and conclusions are provided in section 5.

2. MAPPING PROBLEM

The parallel program is characterized by a task graph: $\Pi = (N, A, \Omega, t)$, where $N = \{1, \ldots, n\}$ is the set of n tasks that compose the program, and Ω, t denote the times related to task execution and to communication between tasks. Thus, each task i has a weight $\Omega(i)$ which defines its execution time, for $i = 1, \ldots, n$. t_{ij} will denote the data communication requirements between tasks j and i. $A = \{a_{ij}\}$ is the adjacency matrix representing the precedence order between the tasks. Since the graph is acyclic, we may number the tasks in a manner such that $a_{ij} = 0$ if $i > j$ [7–9].

The parallel computer is represented as a hypercube graph $G_m = (P, E)$ where m denotes the dimension. An hypercube of dimension m has 2^m nodes and $(m2^{m-1}) = k$ edges. That is, the nodes $P = (1, \ldots, k)$ represent the processors and the edges E represent the communication links. The system is assumed to be homogeneous, with identical processors. Hence, in contraste to the task graph, no weights are associated with the nodes or edges of the processors graph. If the nodes are labeled from 0 to 2^{m-1} in binary, then an edge connects two nodes if only if their binary labels differ in exactly one bit position. The hamming distance between two nodes equals the minimal length of any path connecting these nodes.

The problem is that of assigning the n tasks to k processors. This means that we have to create task clusters $(\Pi_1, \ldots, \Pi_k)$ in a way which optimizes performance. The problem is then characterized by the following objectives:

- The load of the different task clusters must be balanced.
- The communication between different task clusters must be kept to a minimum.
- Two task clusters with communication between them must be mapped onto nearest-neighbor processors.

That is, the task-to-processor mapping is a function $M{:}N \rightarrow P$. $M(i)$ gives the processor onto which task i is mapped.The tasks cluster p (TC_p) is defined as the set of tasks assign to cluster p:

$$TC_p = \{j \,/\, M(j) = p\} \qquad \text{for } p = 1, \ldots, k$$

The load of TC_p (L_TC_p) is the total execution time of all tasks assigned onto it:

$$L\_TC_p = \sum_{i \in TC_p} \check{z}(i)$$

and the idealized average load is given by:

$$L\_TC_p = \sum_{p=1}^{k} (L\_TC_p - \sum_{p=1}^{k} L\_TC_p / k)^2$$

The communication between TC_p and TC_q is equal to:

$$C\_TC_{pq} = \sum_{i,j \in D} t_{ij}$$

where D $=\{(i \in TC_p)$ and $(j \in TC_q)$ and $(p \neq q)$ and $(a_{ij} = 1$ or $a_{ji} = 1)\}$. The first and second contraint can be solved as

$$\min_M (L\_TC + - \sum_{p,q} C\_TC_{pq}) \tag{1}$$

If we suppose that each task cluster must be mapped to a different processor, the nearest-neighbor approach can be solved using the next cost function

$$CCP = \sum_{p,q=1 \& p \in q} C\_TC_{pq} PATH_{pq})$$

where, PATH_{pq} is the minimal length of any path connecting nodes p and q (hamming distance) and, the function to be minimized is

$$\min_M (CCP) \tag{2}$$

3. OUR APPROACH

In this section, we present Genetic Algorithms and formalize our strategy to solve the mapping problem.

3.1 Genetic algorithms

This is an optimization algorithm based on the principles of evolution in biology. A genetic algorithm (GA) follows an "intelligent evolution" process for individuals based on the utilization of evolution operators such as mutation, inversion, selection and crossover []. The idea is to find the best local optimum, starting from a set of initial solutions (individuals), by applying the evolution operators to successive solutions so as to generate a new and better local minimum. The procedure evolves until it remains trapped in a local minimum. We can represent individuals as string. The main program for a GA is the following:

```
Generation of individuals which represent potential
  solutions
  Repeat until system convergence
  Evaluation of every individual
  Selection of the best individual for reproduction
  Reproduction of the individual using the evolutive
  operators
  Replace the worst old individuals by the new individuals
```

In this work, we used the mutation, inversion and crossover operators. The crossover used is standard, a single cutting point chosen with uniform probability over the string length (individuals representation) and a swap of the genetic material following it. The mutation operator is the standard, which modifies each string element according to probability pm. Under inversion operator two points are chosen along the length of the individual, the individual is cut at those points, and the end points of the cut section switch places. In this method three parameters are studied: the maximum number of generations (NUMGEN), the size of the population and the probability (PM) to use the mutation operator after the crossover operator.

3.2 Our heuristic algorithm

The mapping algorithm proceeds in two phases: An initial mapping is first generated by grouping tasks of the tasks graph into clusters in a manner that improves load balancing and communication cost. Then, clusters are assigned among processors in a manner that the nearest neighbor property is satisfied.

For the first phase (*Cluster formation*), the tasks graph is partitioned into as many clusters as the number of processors. We define this problem as a graph partitioning problem, the objective is to split the tasks graph in several subgraphs, so as to minimize the cost of imbalance and the cost of connection *(communication cost)* between them (cost function 1). The GA applied in this case follows the next procedure: we define a search space of *n* vectors where everyone represents an individual, and every individual represents a possible solution (partition). Each vector has *n* elements (tasks) and every element has a value among *1...K,* according to the cluster to which it belongs. We begin with an initial population of individuals randomly defined and we choose the individuals with minimal cost for generating new individuals using the *mutation* and *crossover* operators.

Since the population is constant, we substitute the worst individuals of initial solution by the best individuals generated.

For the second phase (*Processor Allocation*), the clusters generated in the first are allocated to some processor, one cluster per processor, in a manner that minimizes the inter-cluster communication path length. That is, an optimal mapping is generated by assigning clusters to processors in a manner to minimize the cost function 2 (nearest-neighbor property). The GA applied in this case follows the next procedure: we define a search space of k vectors where everyone represents an individual, and every individual represents a possible solution (cluster assignment). Each vector has *k* elements (clusters) and every element has a value among *l...K,* according to the processor to which it is assigned. We begin with an initial population of individuals randomly defined and we choose the individuals with minimal cost for generating new individuals using the *inversion* operators. Since the population is constant, we substitute the worst individuals of initial solution by the best individuals generated.

4. RESULTS ANALYSIS

In this section, we compare the effectiveness of our algorithm (GA) with a mapping algorithm (NN) proposed in [5], using a number of sample Task Graphs. One of the sample is a finite element graph (Figure 1) and one is a random graph. The first graph is representative of the kinds of graphs that result from exploiting parallelism from computation modeling physical systems by finite elements (it is not a directed graph). The last sample is completely a random acyclic directed graph. It is defined for the number of tasks in the graph (n) and the average degrees (d) of the tasks.

We have used the parameters that give the best performances in GA according to the results of the works [8, 9]. NUMGEN allows to optimize the speed-up of the algorithm to reach an optimal solution. We remark than the quality of the solutions improves more rapidly in the first generations that in the following. Thus, a satisfactory quality can be obtained rapidly without to wait that the algorithm converges (NUMGEN = 10). If the size of the population is large, we obtain better results, but the execution time if large. For small size is possible a rapid convergence but an optimal solution can not be found. We begin with an initial population of *n* and *k* individuals in each phase. In the first phase, we used the crossover operator and then the mutation operator according to the PM probability. If PM is large we obtain good results, but it implies an execution time large. We define PM as 0,8.

The performance criteria studied are: execution time of the algorithms and cost function 2 value of the solutions. Mappings were generated for a target 16 and 32 processor hypercube system. The number of simulations per a given set of parameters is either (depending of which occurs first): 30 simulations or the number of simulation required to obtain a given standard deviation of the cost function 2 (σ). Due to

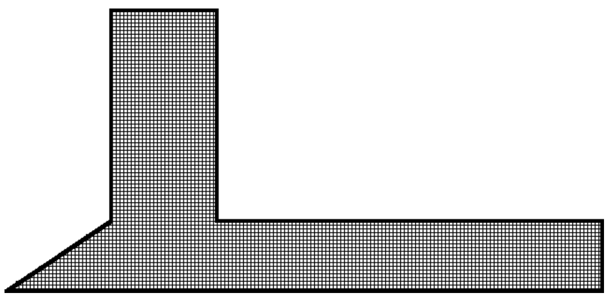
Figure 1 Sample problem graph used for performance evaluation

space limitations, the results presented in this section were chosen because they are representative of the phenomena studied. We fix $\sigma = 0.1$. We have used a Ultra I SPARCstation.

Both approaches result in mappings requiring many interprocessor communication messages (cost function 2), that is due to load balancing constraint. In the first case (Figure 2), NN gives the best results, and the execution time are similar. NN make a nearest-neighbor mapping that permits good solutions for high degree of locality and planar nature graphs (sample 1). Our approach explicitly minimizes total communication (length and volume), and NN approach minimizes total number of messages. This is the reason of the best results in the second case (Figure 3) with GA (due to a low degree of random task graphs). In the second case (Figure 3), NN execution times are smaller. That is due, GA need a large time to reach a good suboptimal solution.

5. CONCLUSIONS

In this paper, we introduced a method to solve the task-to-processor mapping problem in the context of a local-memory multiprocessors with a hypercube interconnection topology.

Our method explicitly minimizes total communication (length and volume) and load imbalance costs. The method is based on two phases: in the first phase an initial tasks graph k-partitioning is done in a manner of minimize communication (volume) and load imbalance costs between the subgraphs. In the second phase, task clusters are assigned among processors of the system with hypercube interconnection topology in a manner that minimizes communication length .

To evaluate the quality of the results obtained, we compared results obtained with a heuristic proposed in [5] for the same problem. The experiments we have run show that the results obtained by our method vary widely depending on the type and size of the graphs considered. Overall, GA appears to be preferable for random task graph with low messages between tasks. That is due to our approach uses a procedure that explicitly attempts to improve load balance. The Genetic Algorithm is easy to implement on a parallel machine, and this can considerably improve the speed obtained with our approach [9].

ACKNOWLEDGMENTS

This work was supported by CONICIT grant S1-95000884, CDCHT-ULA grant I–503–95–A05 and CeCalCULA (High Performance Computing Center of Venezuela).

REFERENCES

1. Muhlenbein, H., Schleutter, G. and Kramm, D. Evolution algorithms in combinatorial optimization, *Parallel Computing,* Vol. 7, No 2, pp 65–93, 1988.
2. Lo, V. Heuristic Algorithms for task assignment in Distributed Systems. *IEEE Transactions on Computer*, Vol. 37, pp. 1384–1397, 1988.

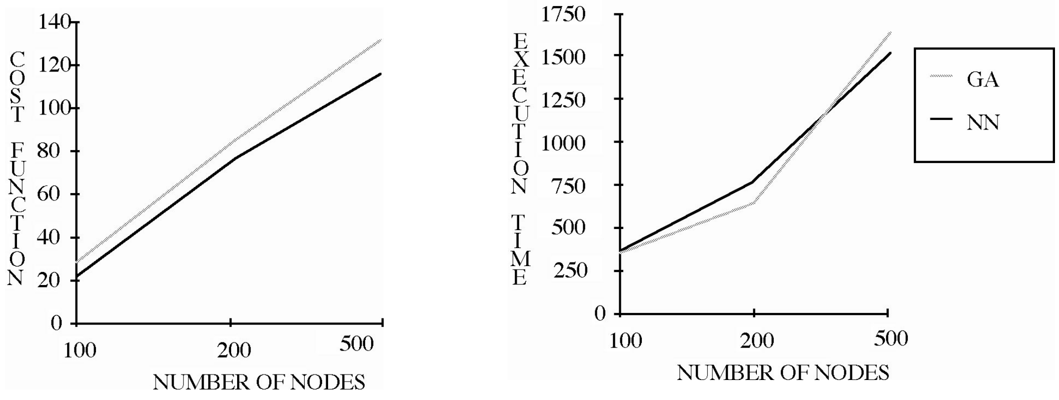

Figure 2 Results and exectution time of the simulation for 16 processors and sample 1

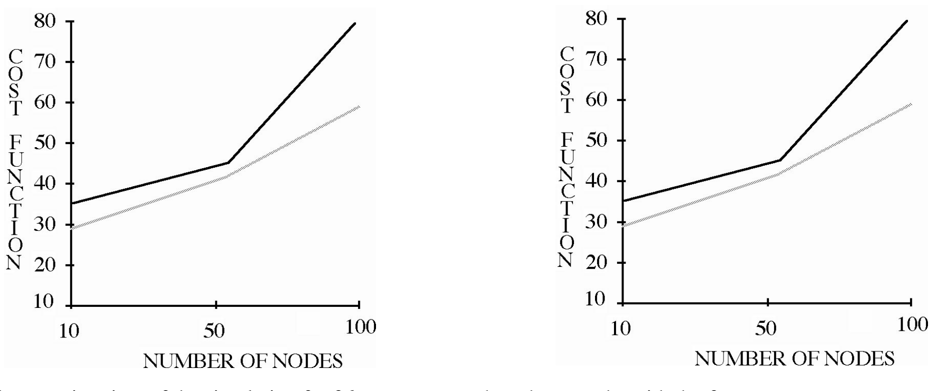

Figure 3 Results and exectution time of the simulation for 26 processors and random graphs with d = 3

3. Goldberg, D. *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.
4. Shin, K. and Chen, M. On the number of acceptable task assignment in Distributed Computer Systems. *IEEE Transactions on Computer*, Vol. 39, No 1, 1990.
5. Sadayappan, P., Ercal, F. and Ramanujam, J. Cluster partitioning approaches to mapping parallel programs onto a hypercube. *Parallel Computing,* Vol. 13, pp. 1–16, 1990.
6. Bowen, N. and Nikolau, C. On the assignment problem of arbitrary process systems to heterogeneous Distributed Computer Systems. *IEEE Transactions on Computers*, Vol. 41, No 3, March 1992.
7. Aguilar, J. Heuristics to optimize the Speed-up of Parallel Programs. *Lecture Notes in Computer Science*, Vol. 1127, pp. 174–183, 1996.
8. Aguilar, J. El Problema de Asignación de Tareas en los Sistemas Distribuidos. *Proceeding of the XXIII Conferencia Latinoamericana de Informática* (Ed. R. Monge), pp.179–188, 1997.
9. Aguilar, J. and Gelenbe, E. Task Assignment and Transaction Clustering Heuristics for Distributed Systems. Accepted to publication in *Informatics and Computer Science*.