AN EFFICIENT MAPPING STRATEGY FOR PARALLEL PROGRAMMING

J.L. ORTEGA-ARJONA† and H. BENITEZ-PEREZ‡

† *Departamento de Matemáticas, Facultad de Ciencias, UNAM, México. jloa@ciencias.unam.mx*
‡ *Departamento de Ingeniería en Sistemas Computacionales y Automatización, IIMAS, UNAM, México. hector@uxdea4.iimas.unam.mx*

***Abstract*— Obtaining an effective execution of a parallel system requires that the mapping of the processes (of the parallel software) on the processors (of the parallel hardware) is efficiently performed. Hence, this paper presents an efficient mapping strategy based on optimizing communications between processes as well as load balancing process distribution onto an arbitrary processor network. Such a mapping strategy is developed as a parallel program, based on the simultaneous execution of local, independent processes. This fact contrasts with many other approaches for solving the mapping problem, like simulated annealing, heuristic search, and others, which require a centralized control for the mapping. In this paper, it is shown that the present mapping strategy is efficient enough when applied to two different mapping problems. Based upon an experimental setup, it is possible to review this mapping strategy following the related impact.**

***Keywords* — Parallel programming, mapping problem, mapping strategy.**

I. INTRODUCTION – MAPPING AND THE MAPPING PROBLEM

Mapping and the mapping problem are commonly defined as follows (Bokhari, 1981): "Suppose a problem made up of several modules that execute in parallel is to be solved on an incompletely connected array. When assigning modules to processors, pairs of modules that communicate with each other should be placed, as far as possible, on processors that are directly connected. We call the assignment of modules to processors a mapping and the problem of maximizing the number of pairs of communicating modules that fall on pairs of directly connected processors the mapping problem."

In topological terms, the mapping problem refers to find whether a graph that represents the system of communicating processes can be mapped onto a graph representing the processor network, so that neighboring processes are allocated on neighboring processors. Nevertheless, even though this seems simple enough, the mapping problem is known to be NP-complete (Bokhari, 1981). Therefore, it seems useless trying to propose exact algorithms for solving the mapping problem. Instead, only mapping strategies that are able to efficiently obtain suboptimal solutions have been proposed.

Let us consider that a parallel program is defined as the specification of a set of processes executing simultaneously, and communicating among themselves to achieve a common objective (Hoare, 1978). Based on this definition, a parallel software program can be represented in the form of a graph, in which each process is a vertex, and each communication between any two processes is an edge. In a similar way, a parallel hardware or processor network can be represented as a graph, in which now each processor is a vertex and each interconnection between any two processors is an edge. The mapping problem, hence, reduces to embedding the software graph into the hardware graph.

Nevertheless, since mapping should provide a certain distribution of the processes onto the processors so the most efficient execution is obtained; two optimization issues have to be considered:

- Load balancing. The processes have to be mapped onto the processors so the processing load caused by all processes is fairly distributed over all processors. In such a situation, the parallel system is considered to be balanced.
- Communication optimization. The communications between any two processes should be distributed as evenly as possible over all connections between processors. When this is the case, it is said that communications are optimal.

Regarding the second issue above, optimizing communications means that neighboring processes should be mapped onto neighboring processors. Otherwise, for any communication between any two processes, more than one connection has to be used. This produces a communication overhead due to communication re-emission, increasing overall execution time of the parallel program. Moreover, communication re-emission also causes load on the intermediate processors, since processing in needed for routing a communication until it reaches its final destination, and also tends to increase overall execution time of the parallel system. Nevertheless, it is commonly not possible to map every neighboring process onto neighboring processors. Thus, the processes should be allocated so that the overall communication costs are kept as minimal as possible.

A. The Mapping Problem: A Formal Description

For the current purposes, and for obtaining a more formal definition of the mapping problem, let us assume the following features of the parallel system:

- The parallel hardware platform has a distributed memory organization. Each processor has its own local, private memory.

- The only means for communicating two processors are connections of a network. This network has a fixed topology or, at most, it is statically reconfigurable.

Definition 1: Parallel Hardware Platform. Let us define a parallel hardware platform with distributed memory as a pair:

$$HW=(<P, A_{HW}>, C_{HW}) \quad (1)$$

where $<P,A_{HW}>$ is an undirected graph representing the network in which $P=\{P_1, P_2, \ldots, P_N\}$ is a set N processors of the parallel hardware platform, and A_{HW} is an adjacency matrix representing such a graph; and C_{HW} is a communication cost matrix associated with the parallel hardware platform, in which an element $C_{HW}[m,n]$ represents the costs of transmitting a single data unit between processors P_m and P_n.

Definition 2: Parallel Software System. Let us define a parallel software system as a static network of communicating processes as a triple:

$$SW=(<Q, A_{SW}>, C_{SW}, t) \quad (2)$$

where $<Q, A_{SW}>$ is a directed graph representing the structure of the parallel software system, in which $Q=\{Q_1, Q_2, \ldots, Q_M\}$ is the set M processes in the parallel software system, and A_{SW} is an adjacency matrix representing such a graph; C_{SW} is a communication cost matrix associated with the parallel software system, in which an element $C_{SW}[i,j]$ represents the number of data units sent from process Q_i to process Q_j. Three types of communications should be considered: i) Internal communications, ii) Short communications and iii) Long communication; and t:$Q \rightarrow Z$ is a function where $t(Q_i)$ represents the processing time of a process Q_i in terms of a number Z of atomic computational steps.

Definition 3: Restrictions. When mapping processes onto processors, a restriction implies that some processes must be placed on a particular processor, due to such a processor offers unique capabilities needed for the parallel software system. For example, certain processes may require access to peripheral resources and I/O devices that belong to specific processors. Hence, this fact should be taken into consideration for the formal description as follows. Let us define a restriction as a subset $R \subseteq Q \times P$, where $(i,m) \in R$ means that process Q_i must be mapped on processor P_m.

Definition 4: Load. The load of a processor refers to the amount of time that such a processor spends processing a process or a group of specific processes. Obviously, the load is relative to a mapping π. Hence, in terms of time per process on a processor, load l can be thus defined as:

$$l(P_m) = \sum_{\pi(Q_i)=P_m} t(Q_i) \quad (3)$$

The function t here, which returns the amount of time caused by a process, tends to be problematic since it is not easy to obtain the amount of time or load that a process will require from the processor. In general, determining the load can be achieved by measuring the wall-clock time that a process actually takes during real execution. Another way can be calculating the theoretical time complexity of such a process. However, such a calculation tends to be not very realistic, since theoretical complexity is a notion that cannot be automatically obtained. Measuring tends to be accurate enough, although it involves the problem of process monitoring within the parallel software system. Besides, in general, the load produced by a process tends to dynamically vary during execution. However, for a static mapping strategy, the load of a process can be considered as the statistical mean of its measured load.

Definition 5: Optimal Mapping. An optimal process mapping is defined in terms of the function:

$$\pi: Q \rightarrow P \quad (4)$$

where $\pi(Q_i)= P_m \ \forall (i,j) \in R$, which means that the mapping function takes into account the restrictions above, $l(P_m) \approx \frac{\sum_{n=1}^{N} l(P_n)}{N}$, $\forall\ m=1, \ldots. N$, which means that the processors are (mostly) load balanced, and

$$\min\left(\sum_{i,j} c_{SW}[i,j] \cdot c_{HW}[t(P_m), t(P_n)]\right),$$

which means that the communication costs are minimized.

B. Other Mapping Strategies

There are many different mapping strategies that attempt to solve the mapping problem. Among the most known ones are simulated annealing (Hart and Chen, 1994; Robic *et al.*, 1995), cardinality based algorithms (Bokhari, 1981), heuristic search (Martínez-Gallar *et al.*, 2010; Soriano and Orduña, 2009), and evolutionary algorithms (Erbas *et al.*, 2006).

All these methods share a commonality: the mapping optimization is controlled by a centralized decision-making mechanism, like for example, the temperature decrease in the case of simulated annealing. Strategies based on these methods report adequate results, although when both hardware and software graphs present a considerable size, they normally require too much time to provide a satisfactory solution. Moreover, some of these strategies are reported to be applied with reasonable results for a larger number of processes which meet certain restriction. For instance, the cardinality based algorithm (Bokhari 1981) provides results in a reasonable time when both hardware and software graphs present an identical size. Furthermore, some approaches propose that, in order to speed up the mapping operation using, the strategies can be parallelized. This is mostly right, although this parallelization has certain limit due to the need of continuous communication with the centralized decision-making mechanism. This is, the implementations of these strategies, whether sequential or parallel, are always a centralized.

II. AN EFFICIENT MAPPING STRATEGY

The mapping strategy here is distributed, and allows in a short time for optimal or suboptimal mapping solutions. It is implemented for mapping a static parallel software system of communicating processes onto a cluster, this

is, a network of processor with distributed memory organization. It has been implemented in the Java programming language, and can be executed on a cluster with an arbitrary number of processors. Nevertheless, this does not imply a restriction to the mapping strategy: this mapping strategy can be programmed with a different parallel programming language, and executed in parallel on other target hardware. This is due to the mapping strategy has the following characteristics:

- it is distributed
- it makes use of local, nearest neighbor optimizations, which considered together, lead to a global optimization
- it is based on the information exchange of load and communication costs between neighboring processors
- it allows processes migration according to the optimization demands
- it makes use of the parallel hardware itself to increase the speed of the mapping

In summary, the strategy is implemented as identical communicating processes, executing on each processor of the parallel platform. These processes execute in parallel, iterating through a number of steps, looking for an optimal or suboptimal solution to a current mapping problem. At each iteration step, processes on neighboring processors exchange their current information about load and position. To reduce local communication costs, processes with high local costs are sent to neighboring processors. Since the location of neighboring processes of the process to be moved is known, this process can be sent to the adequate processor.

The general steps of the mapping strategy, when executing in parallel on each processor, is as follows.

1. Information exchange: Exchange information with process neighbors and allow processes migration between direct processor neighbors.
2. Communication optimization: Obtain processes with high local communication costs, and decide whether to send them to neighboring processor
3. Load balancing: Locally load balance

Initially, all processes are mapped on any arbitrary processor (except those associated with a restriction). At each iteration, processes are moved in order to meet optimization demands, considering first the most optimal load balance. Local communication costs are reduced by selecting processes that cause high costs, and moving them to another processor. Since load balancing is considered a more important feature than communication minimization, the strategy may only obtain a suboptimal mapping solution in which load balancing is achieved, but its communications demands are not completely met. Thus the decision that, in order to reduce communication costs, the strategy selects (with a decreasing probability) a process to be moved to another processor if such a process causes a high local communication costs.

The distributed mapping strategy carries out a mapping π: $Q \rightarrow P$ considering the following:

- The communication costs C_{HW} $[m,n]$ between any two processors P_m and P_n is:

$$c_{HW}[m,n] = \begin{cases} 1 & if\ A_{HW}[m,n] = 1 \\ 2 \times d_{mn} & else \end{cases}$$

where d_{mn} denotes the shortest distance between P_m and P_n.

- The communication costs between any two connected processes is 1.

Regarding the formal definition of mapping, both these considerations do not introduce any loss of generality, even though the available storage size of the processors and storage requirements of the processes are not considered. The mapping strategy is executed in parallel on each processor. Commonly, because the mapping problem is NP-complete and the optimal solution of a mapping is not known in advance, the number of iterations to execute is set by considering load balancing only, as shown in the following section.

A. Optimization of Communication Costs

The communication costs between any two processes have been defined above to be set to $c_{sw}[i,j]$=1. Nevertheless, during the execution of the mapping strategy, three types of communications should be considered:

1. Internal communications.
2. Short communications.
3. Long communications.

Internal communications are performed between any two processes currently mapped on the same processor. Its value is set to $c_{SW}[i,j]$ (always 1); short communications are carried out between any two processes currently mapped on neighboring processors, so its value is set to $c_{SW}[i,j].c_{HW}[m,n]$ (again, always 1); long communications exist between processes currently mapped on processors which are not neighbors, and thus, its value is set to $c_{SW}[i,j].c_{HW}[m,n]$. Considering these types of communication, selecting a candidate process to be moved to another processor is based on an attraction function, defined in the following section.

B. An Attraction Function

An attraction function is defined as based on the following assumptions:

1. Each processor has the information about the shortest distance to every other processor in the network.
2. Each processor P_m has the information about all communication connections $I_{HW}[m,n]$ to a neighboring processor P_n, through which a shortest distance to all the other processors can be obtained.
3. Each processor has the information of the **probable** location of all processes. Notice that, at each iteration, locations are exchanged with the neighboring processors.

Due to only the neighbors of a processor have the information about the actual location of a process which has moved to another processor, such information is not always up to date. The reason is that when a process Q_i has been moved, it normally takes a few iterations until all processors are aware that it has been allocated

somewhere else. Thus, at a certain moment of execution of the mapping strategy, the current information about the location of a process is only precise, but not exact. The number of iterations required since a new allocation is produced by a processor P_m until this information is available to a processor P_n depends on the shortest distance between them (d_{mn}). Thus, the attraction function is defined here to decide whether and which process Q_i currently allocated on processor P_m is sent to a neighboring processor. The attraction function is constructed in the following way:

Let Q_i be a process currently allocated on processor P_m. Assume that Q_i communicates with Q_j whose probable location is P_n. Let $I_{HW}[m,n]$ be a communication connection, and $\{Q_j\}_i=\{Q_j|c_{SW}[i,j]\neq 0\}$ be the set of all processes communicating with process Q_i. The attraction function $F_k(i)$ defines the communication costs for Q_i communicating with the processes in $\{Q_j\}_i$ through the communication connection k. If the two processes Q_i and Q_j are mapped on the same processor P_m, k is set to 0. The attraction value for a process Q_i through a communication connection k is defined as follows:

$$F_0(i)=\begin{cases}\sum_j c_{SW}[i,j] & \text{if } Q_i \text{ and } Q_j \text{ are on processor } P_m \\ 0 & \text{else}\end{cases} \quad (5)$$

$$F_k(i)=\begin{cases}\sum_j c_{SW}[i,j]\cdot c_{HW}[m,n] & \text{if } I_{HW}[m,k] \text{ realizes a shortest } d_{mn} \\ 0 & \text{else}\end{cases} \quad (6)$$

The mapping strategy here selects at each iteration the processes on a processor with the highest attraction values. This consideration still fulfills the load balancing requirement. If two or more processes have the same attraction value, the processes to be moved are randomly selected. Any process with the highest attraction value is sent to a neighboring processor with decreasing probability, in this case neglecting the load balancing requirement, in order to assure that the mapping strategy does not stay in communication suboptimum. Notice that the decreasing probability is a function of the number of iterations and the current iteration number, as follows:

$$1-\frac{\text{current iteration}}{\text{total number of iterations}} \quad (7)$$

III. ANALYSIS OF THE MAPPING STRATEGY

Since minimizing communications proceeds neglecting the load balance and with decreasing probability, and also most of minimizing communications is part of the load balancing, the analysis of the mapping strategy focuses on the load balance. For simplicity, let us suppose that the number of processes is a lot larger than the number of processors, this is:

$$N>>>M \quad (8)$$

Moreover, let us neglect the number of processes executing on each processor, and just consider the load on each processor represented by a real number.

Let π be a mapping. A (global) cost function $\Gamma(\pi)$ for the load distribution may be defined as:

$$\Gamma(\pi)=\sum_{m\in P}(\bar{l}-l(m))^2 \quad (9)$$

where $\bar{l}=\sum_{m\in P} l(m)/N$.

A global minimum of $\Gamma(\pi)$ can be reached by locally optimizing the function (Boillat, 1990):

$$\Gamma_i(\pi)=\sum_{(m,n)\in P}(l(m)-l(n))^2 \quad \forall m\in P \quad (10)$$

From this, the number of iterations needed to reach an uniform distribution is at most $O(N^3)$.

IV. PERFORMANCE EVALUATION OF THE MAPPING STRATEGY

The performance of the mapping strategy is evaluated depending on how fast an optimal (or sub-optimal) solution is reached on a given parallel platform (remember, the mapping strategy is developed as a parallel program). Thus, the performance is measured as the time used to achieve a percentage of success, on actual hardware platform, provided a number of iterations.

Two mapping examples are presented: *(a)* a homogeneous load, which has an optimal solution, that is, a uniform distribution is achieved for a particular number of processors, and *(b)* a non-homogeneous load, whose solution is sub-optimal, that is, there is no even distribution of the software processes to hardware processors. Both mapping examples are developed as actual parallel programs (no simulations), executing on a cluster of computers, with a fixed number of processors. Also, for each example, all tests are performed so software processes represent the same load, and all communications are considered to have the same costs.

The first mapping example is performed to shown that a homogeneous-loaded, parallel software system, a uniform distribution is obtained. For this example, Table 1 shows the results of mapping a 4 × 4 mesh of software processes onto a cluster of 4 processors. In these tests, for each number of iterations, 50 executions have been run. The load has been optimally balanced in all executions.

Table 2 shows the results of other mapping example, whose solution is planned to be suboptimal: a 5 dimensional hypercube of processes onto a 3 dimensional processor hypercube. The load balance has been obtained in all but one of 200 executions. Again, for each number of iterations, 50 executions have been run. The load balance is obtained after about 20 steps. The other steps are required in order to optimize the communication costs.

In Table 2, the results show that after 800 iterations, a suboptimal mapping is obtained, differing in the mean only 4% from the optimal solution. 96% of the mappings are optimal; and they do not require any routing optimization. Notice that all process mappings in both

Table 1: Mapping a 4 × 4 process mesh onto a cluster of 4 processors.

Number of iterations	Mean costs	Variance	Success rate	Time used
50	25.54	1.53	86%	0.32s
100	25.06	0.52	98%	0.46s
200	25.00	0.00	100%	0.73s

Table 2: Mapping a 5 dimensional process hypercube onto a 3 dimensional processor hypercube.

Number of iterations	Mean costs	Variance	Success Rate	Time used
200	98.21	23.86	55%	1.22s
400	90.53	18.09	67%	2.451s
600	84.16	13.38	86%	3.55s
800	84.82	12.49	96%	4.63s

these examples have been obtained in a time of just a few seconds.

V. CONCLUSIONS

In this paper, a mapping strategy is presented as an efficient alternative to other traditional mapping techniques. The mapping strategy here has the following advantages:

- It is efficient, and capable of producing suboptimal mapping solutions
- It is simple and easy to implement
- It is distributed over any network of processors
- It does not require global synchronization: Synchronization is restricted to the local interactions between neighboring processors.

The mapping strategy here is qualified as *efficient* since, when compared with the performance of other previous work, the times used to obtain a optimal (or sub-optimal) solution are, in average, shorter. For the examples here, the mapping strategy, as a parallel program, achieved about 1908 ms of average time used. This is, in the order of thousands of milliseconds. Other referred approaches require more time for achieving a solution. For instance, Bokhari (1981) presents the mapping of several problems using his cardinality based algorithms, whose times used is also in the order of thousands of milliseconds, but their average is about 7615 ms (between 321 ms and 29219 ms). Martínez-Gallar *et al.* (2010) present also four mapping examples using heuristic search, whose average time used is in the order of seconds, having an average of 2060 s, with a minimum of 1796 s and a maximum of 2411 s. Other approaches simply do not consider time for their performance evaluation. Robic *et al.* (1995) make use of a *Q* function for the performance evaluation, and only considers that their approach obtains results within an "acceptable time". Erbas *et al.* (2006) evaluate performance regarding the accuracy, uniformity, and extent of multimedia applications, but do not provide information about the time used for obtaining the mapping.

As future work, it is proposed to use the mapping strategy to dynamic parallel platforms. The mapping strategy seems well suited for dynamic parallel platforms since it is adaptable to changes in the communication links between processors. Moreover, the mapping strategy seems to be a candidate to be included as part of a real distributed operating system (along with a communication kernel) or part of a configuration tool for parallel software systems.

ACKNOWLEDGMENTS

The authors acknowledge the support of UNAM-PAPIIT IN103310 and ICyTDF PICCO 10-53

REFERENCES

Boillat, J.E., "Load balancing and Poisson equation in a graph," *Concurrency: Practice and Experience*, **2**, 289-313 (1990).

Bokhari, S.H., "On the Mapping Problem," *IEEE Transactions on Computers*, **30**, 207-214 (1981).

Erbas, C., S. Cerav-Erbas and A.D. Pimentel, "Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design," *IEEE Transactions on Evolutionary Computation*, **10**, 358-374 (2006).

Hart, S.M. and C.S. Chen, "Simulated annealing and the mapping problem: a computational study," *Computers and Operations Research*, **21**, 455-461 (1994).

Hoare, C.A.R., "Communicating Sequential Processes," *Communications of the ACM*, **21**, 666-677 (1978).

Martínez-Gallar, J.P., F. Almeida and D. Giménez, "Mapping in Heterogeneous Systems with Heuristic Methods. Applied Parallel Computing," *State of the Art in Scientific Computing*. Lecture Notes in Computer Science, **4699**, 1084-1093 (2010).

Robic, B., J. Silc and B. Robic, "Algorithm Mapping with Parallel Simulated Annealing," *Computers and Artificial Intelligence*, **14**, 339-351 (1995).

Soriano, R. and J.M. Orduña, "Improving the Scalability of Communication-Aware Task Mapping Techniques," *2009 International Conference on Advanced Information Networking and Applications Workshops*, Bradford, United Kingdom, 1061-1066 (2009).

Received: October 10, 2011.
Accepted: June 28, 2012.
Recommended by Subject Editor José Guivant.