Efficient Solutions for Mapping Parallel Programs*

P. Bouvry[1] and J. Chassin de Kergommeaux[2] and D. Trystram[2]

[1] CWI - Center for Mathematics and Computer Science,
Kruislaan 413, PP.O. Box 94079, 1090 GB Amsterdam - The Netherlands
[2] LMC-IMAG, 46, avenue Félix Viallet, 38031 Grenoble cedex - France

Abstract. This paper describes a mapping toolbox, whose aim is to optimize the execution time of parallel programs described as task graphs on distributed memory parallel systems. The toolbox includes several classical mapping algorithms. It was assessed by computing the mapping of randomly generated task graphs and by mapping and executing on a parallel system synthetic programs representing some classical numerical algorithms. A large number of experiments were used to validate the cost functions used in the toolbox and to compare the algorithms.

Keywords: Parallel environment, Load-balancing, Mapping.

1 Introduction

Efficient use of distributed-memory parallel systems requires the use of specific tools, whose mapping is one of the most important one. The goal of the mapping tools is to minimize the execution time of parallel programs on distributed-memory machines by controlling the use of computation and communication resources. This article describes a toolbox aiming at providing an assignment of the tasks of a parallel program to the available processors to obtain the shortest possible execution time for the entire program. Therefore, mapping algorithms aim at maximizing the (useful) occupation of processors without increasing too much communication costs.

The tasks execution times and inter-tasks communication costs of some regular programs can be entirely determined at compile time. In this case, it is possible to perform static task allocation in advance. This is known as the *mapping* operation whose complexity is exponential in the general case. Thus, it is difficult to obtain an optimal mapping and numerous heuristic solutions have been proposed, representing different tradeoffs between computation cost and quality of mapping [9].

This paper presents a *mapping toolbox*, implementing several "classical" mapping algorithms. This toolbox was used to assess different cost functions by computing the relation between the value of these functions, optimized by the

* This work was partially sponsored by the EU's Copernicus programme under contract number CIPA-C193-0251

mapping algorithms, and the actual execution times of parallel programs. The implemented mapping algorithms were also evaluated, by comparing the execution times of a set of representative parallel programs, mapped using the algorithms of the toolbox.

This work is part of the APACHE research project whose aim is the design and development of a general programming environment to balance automatically the load of parallel applications, resulting in reduced development time and increased portability of parallel applications [10].

2 The mapping problem

2.1 Models of machines and programs

A distributed-memory parallel computer is composed of a set of nodes connected via an interconnection network. Each node includes some computation facilities and a local memory. A communication between two processors is much more time consuming than a local memory access. The MIMD model intends to map different executable codes, called tasks, onto processors. Designing a program such that only one task will be allocated on one processor of the target machine would lead to an architecture-dependent and non-scalable code. On the contrary, a too large number of tasks is difficult to manage efficiently. The granularity (size of tasks) is one important parameter for the efficiency of a parallel program.

Most parallel programs can be described using a graph formalism. In most representations, each vertex represents a task and each edge a communication link. We consider that a task can be allocated to a single processor. Any processor can make some communications and computations. We add to this basic model the computation costs of the tasks (execution time) and the amount of information communicated on the links. Often, the user cannot determine the exact values of the program parameters but can only approximate them. The model used in this paper is based, as a large number of related works [9], on tasks graphs without precedence. It is closely related to the programming model used by the transputer based parallel system that was used for experiments, where undirected task graphs are often explicitly described in a separate configuration file.

In the following, we will denote: T, the set of tasks and n their number, P, the set of processors and m their number, $ex(t)$, the computation time of task t, $comm(t, t')$, the total communication time between t and t'.

2.2 Description of the Problem

The parallelization process requires first to distribute data among the different processors. The objective considered here is to minimize the execution time of the whole program. Formally, a mapping is an application (called *alloc*) from T to P which associates to each task t an unique processor $q = alloc(t)$. The number of all possible solutions is n^m.

Mapping tools are part of programming environments. Ideally, the user of a parallel machine would use a parallel compiler which would distribute data among the processors and organize automatically (implicitly) the communications induced by local computations. Practically, parallelization directives can be included in the source code and used to generate a task graph by determining computation and communication costs and analyzing data dependences. This phase is usually followed by a clustering operation, parameterized by the granularity of the target machine. Then, mapping is performed.

2.3 Quality of the solution

Most solutions of the mapping problem are based on the optimization of cost functions, denoted z. There exist in the literature many choices for z. Norman and Thanish propose a classification of the parameters which influence the cost of a mapping [9]. Two opposite criteria have to be taken into account: minimization of inter-processors communications and load-balancing of computations between processors. We chose to minimize the most loaded processor, which is a trade-off between these two criteria:

$$z = \max_{p \in P}\Big(\sum_{t | alloc(t) = p} ex(t) + \sum_{t' | alloc(t') \neq p} comm(t, t')\Big)$$

This basic function does not consider that communications can be overlapped by computations, but it can be adapted by considering the maximum between computation and communication times in place of the second sum. The above cost function does not take into account the length of the exchanged messages nor the topology. Two refinements were introduced to take into account distances between processors.

1. measures on the parallel target system can give the costs to transfer bytes between two processors depending on the number of bytes communicated.
2. communications can be expressed as a linear function of the distance between two processors.

3 Description of the Mapping Toolbox (ALTO)

Many solutions can be found in the literature for solving the mapping problem [4]. Exact algorithms give the optimal solutions but in practical cases they can not be used because of the combinatorial explosion of the number of solutions. The goal of heuristic algorithms is to give good solutions in relatively reasonable time. Two sub-classes of heuristic algorithms were explored: greedy algorithms which construct partially the solution and iterative algorithms whose principle is to improve an existing solution. Obviously, the cost of the mapping algorithm itself must be related to the use of the solution. The more used a given mapping, the greater the time that ought to be invested computing it.

3.1 Basic hypotheses

The mapping toolbox "ALTO" (for ALlocation TOol box) was originally developed to map parallel programs written in a parallel dialect of C, where tasks are source code files, on a transputer based (Supernode architecture) Meganode [7], including 128 T-800 transputers. In parallel C programs, a description of the different tasks and a configuration description must be supplied by the user. The configuration file describes the task interconnection graph, the processor network and the mapping of the different tasks on the processors. To use ALTO, the configuration file must be extended to include cost values (computing and communication costs). In addition, we assume that all processors have the same processing capabilities, that processors may have different memory sizes, and that communication costs between tasks allocated on the same processor are negligible.

3.2 Greedy algorithms

In a greedy algorithm, the mapping is done without backtracking (a choice already done can never be reconsidered). The allocation of the i^{th} task is based on a criterion depending on the mapping of the $(i-1)^{th}$ first tasks. Two kinds of greedy algorithms can be envisaged: either they are based on empirical methods or they come from the relaxation of classical graph theory algorithms which are optimal for some restricted cases. They are easy to implement and have a polynomial complexity. *List algorithms* are the most used greedy algorithms. Tasks are first sorted on a given criterion and then are mapped in this order on the processors. In ALTO, the following greedy algorithms were implemented:

Modulo: the modulo algorithm consists in allocating the i^{th} task onto the i^{th} *modulo m* processor. Theoretically, this algorithm has the same behavior as a random mapping algorithm with a great number of tasks. It was mainly implemented to serve as a reference. The only modification made to the basic algorithm was to skip processors that have not enough memory for a task.

Largest Processing Time First (LPTF): LPTF is a heuristic whose criterion is restricted to load balancing. Tasks are first sorted by decreasing computation cost order, then allocated on the less loaded processor having enough memory.

Largest Global Cost First (LGCF): this greedy algorithm aims at balancing the global load. Tasks are first sorted according to this order, then allocated on the less globally loaded (communication and computation costs taken into account) processor having enough memory.

Struc-quanti: this algorithm uses first a mixed criterion, i.e. qualitative (the number of links of each task) and quantitative (communication and computation costs), to sort tasks. Then, tasks are allocated on the less globally loaded (communication and computation costs taken into account) processor having enough memory.

3.3 Iterative algorithms

All iterative algorithms try to improve an initial solution usually obtained by a greedy algorithm. Most iterative algorithms exchange tasks between processors to improve locally a solution. Most of such algorithms use random perturbations to leave local minima of the cost function and to obtain better solutions. In ALTO two kinds of neighborhood were used: transfering a task from the most loaded processor to another one and exchanging a task from the most loaded processor with another task communicating with it.

Simulated annealing. It is based on an analogy with statistical physics: the annealing technique is used to obtain a metal with the most regular structure possible. It consists of heating the metal and reducing slowly the heat so that it keeps its equilibrium. When the temperature is low enough, the metal is in an equilibrium state corresponding to the minimal energy. At high temperature, there is a lot of thermic agitation which can locally increase the energy of the system. This phenomenon occurs with a given probability decreasing with the temperature. It corresponds mathematically to give the possibility of leaving a local minimum of the function to optimize.

In ALTO a mapping is improved by elementary operations involving task exchanges. The percentage of bad exchanges (leading to a worse solution) is high at the beginning and decreases during the execution of the algorithm. Theoretical studies proved the convergence to the optimal solution of the continuous version if some properties are verified such as a very slow decreasing of the heat, that it is not practical for real problems. It is very hard to tune: finding the starting temperature or the heat decreasing steps have to be done after many experiments. All the parameters were determined according to the literature [2, 6] and a large number of experiments. An estimated value of the average differential value of the cost function between moves denoted δ_f is first computed. Next, using a starting percentage of acceptance of bad solutions of 0.8 (denoted τ) the starting heat is determined ($k = \frac{\delta_f}{log\left(\frac{1}{\tau}\right)}$). In this implementation the function $k^{0.98}$ is used for practical heat descent. A bad solution is accepted if $e^{\frac{-\delta_f}{k}}$ is greater than a random number uniformly chosen in $(0, 1)$ (see figure 1).

Tabu search. It is a deterministic meta-heuristic [5]. As for the simulated annealing, a lot of parameters have to be tuned or defined [1]. The tabu search starts from a given solution and improves it by local pair-wise exchanges. Accessible solutions using local moves are called neighbors. At a given step, the best unexplored neighbor is chosen. This implies that the last explored moves must be recorded in a tabu list. Only the last moves can be recorded in order to limit the memory and time costs. Possibilities of cycling are also reduced by this recording. The tabu list length is fixed empirically for each implementation of a tabu search. Aspiration criteria can be used to override a tabu list (e.g. if the proposed move leads to the best ever found value of the cost fonction). If

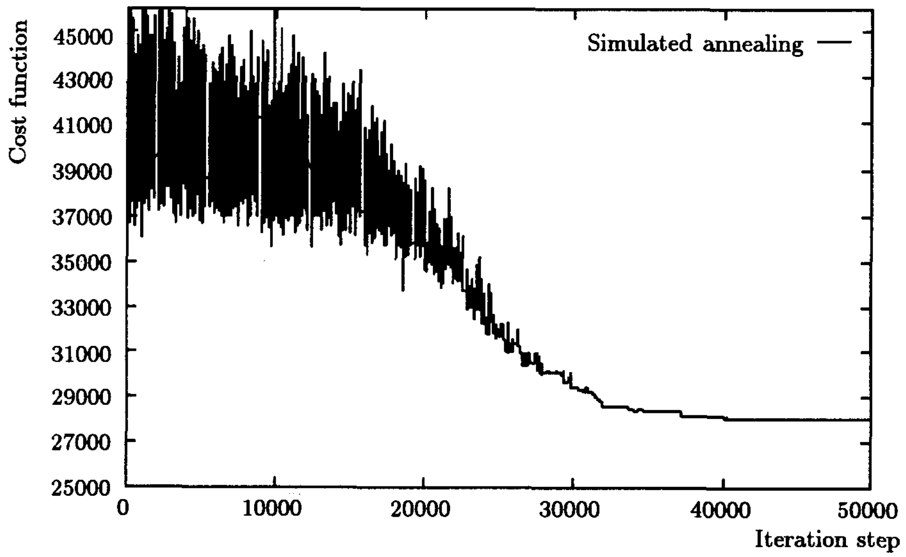

Fig. 1. Typical execution of Simulated annealing

too much time is spent without significative improvements of the solution, diversification factors can be used in order to move to other areas. Intensification factors can be used if some areas seem very promising. Implementation details can be found in [1].

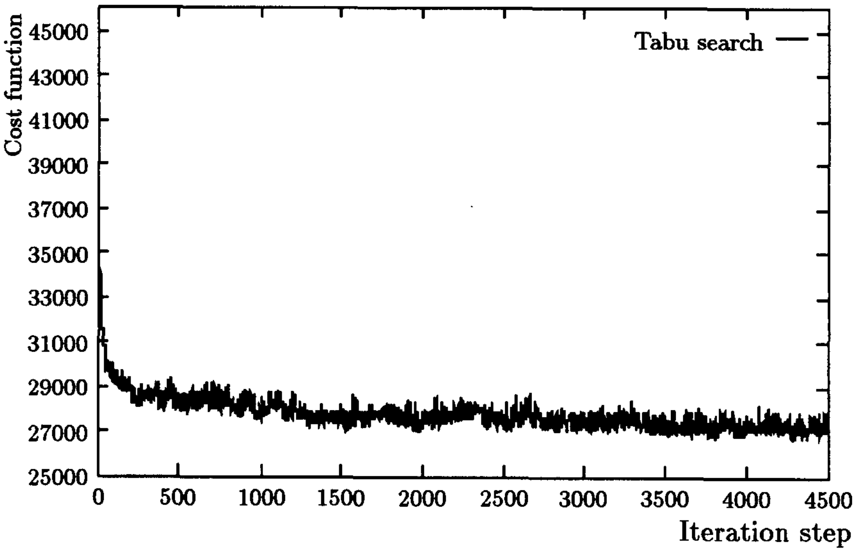

Fig. 2. Typical execution of Tabu search

Figures 1 and 2 examplify the simulated annealing and tabu search methods, for the same example of mapping a randomly generated task graph of 100 tasks on 10 processors. They indicate that the former improves the cost function after a long latence time and is very unstable at the beginning, while on the contrary, the latter is more efficient at the beginning but converges slower. The basic iteration of the tabu is greater than simulated annealing since at each step, it explores all the neighbor configurations to find the best.

4 Validation and experiments

Two kinds of validation were done: first, we generated "artificial" parallel programs using a random task graph generator; then, the mapping algorithms of ALTO were tested on a set of benchmarks, representative of many classical parallel numerical algorithms.

4.1 Task graphs generated randomly

To tune iterative algorithms and to show the behavior of the different mapping algorithms, many task graphs were randomly generated, with the following parameters: number of tasks, maximal computing cost of a task, maximal communication cost, and maximal degree of a task (number of neighbors).

Different task graphs were generated uniformly using these parameters. Figure 3 presents the average improvement (in %) given by the different algorithms versus the behavior of the modulo algorithm. Each value of the table is based on 100 random task graphs. The parameters used in the random task graph generations are: 100 tasks, each task communicating to a maximum of 4 other tasks, 16 processors, a maximal computing cost of 1000 seconds. The cost function used takes into account the sum between communication and computation costs (all the processors being at the same distance). These tests reflect mostly the expectations:

- The most sophisticated mapping algorithms (simulated annealing and tabu) are the most efficient ones.
- LPTF performs well for parallel programs with low communication costs.
- The higher the ratio communication/computation, the more interesting are the iterative algorithms.
- The results of *LGCF* are better than *struct_quanti* which takes into account a qualitative criterion while the quality of results is estimated in terms of quantitative criteria (i.e. the cost function).

The previous results are encouraging but not sufficient since we are not sure that random task graphs are representative of "real" parallel programs.

4.2 Experiments with real programs executed on a real machine

Extensive tests were run on a 128 nodes transputer-based MegaNode, with the VCR software router[3], using the ALPES performance evaluation environment of

[3] VCR: Virtual Channel Router, developed by Southampton University.

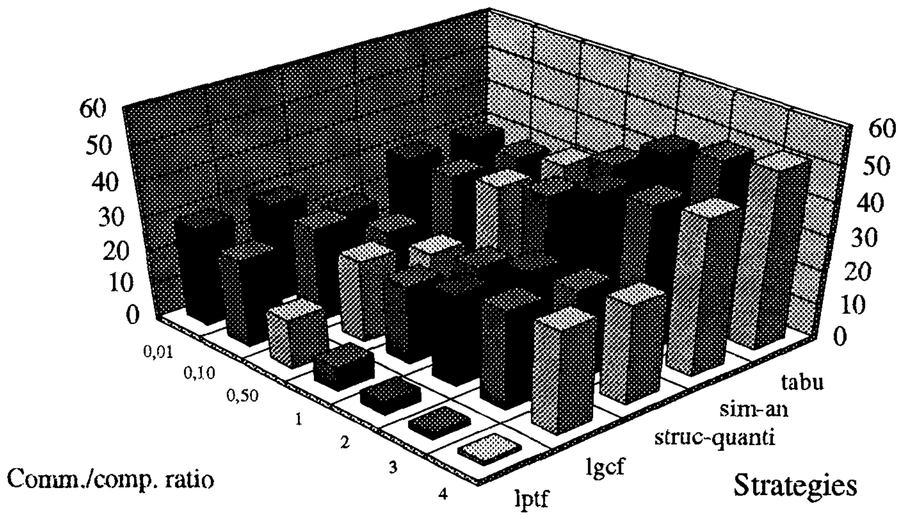

Fig. 3. Percentage of improvements of mapping algorithms relative to modulo

the APACHE project [8]. ALPES provides a modeling language called ANDES, allowing the generation of synthetic programs from the description of an application. Synthetic programs consume resources, as their actual models, although they do not produce any results. Performance measurements were obtained by executing these synthetic programs on an actual parallel computer. Parameters of the synthetic programs (mainly computation/communication ratios) can be easily changed to *emulate* various parallel target architectures. ALPES includes also monitoring tools.

A benchmark for mapping tools was designed using ANDES: it includes the description of classical parallel programs (FFT, matrix-vector multiplication, Gaussian elimination, matrix product using the Strassen algorithm, Divide and conquer, PDE solver, etc.) [3]. Several problem sizes were used for each problem to generate a total of 17 different benchmarks. Each one was run 100 times in order to eliminate the effect of execution indeterminism.

ALTO was coupled with PYRROS which is a complete scheduling platform designed at Rutgers by Gerasoulis and Yang [11]. PYRROS takes as inputs the precedence graphs generated by ANDES to group the tasks. The output of PYRROS, where the orientation of the arcs is not taken into account, is then passed to ALTO.

Adequation of the cost functions. Several mapping experiments were done using various cost functions, in order to determine the best one, whose costs are the closest to the execution times of the benchmarks. The four cost functions defined in section 2.3 were tested on the Meganode, configured as a torus.

Figure 4 presents the cumulated results of the whole set of benchmarks, run on 16 processors for the different cost functions. We observe a linear correlation between all cost functions and execution time. In theory, it is possible to communicate over several transputer links simultaneously and to overlap communications by computations. However, because of the software overhead induced by VCR, this is not the case in practice and the first cost function is not applicable. In addition, since VCR uses packetisation and pipelining mechanisms, the first refinement is not usable. The best cost function is therefore the sum of computation and communication costs (second cost function above), coupled with the second refinement (torus topology, using VCR routing tables).

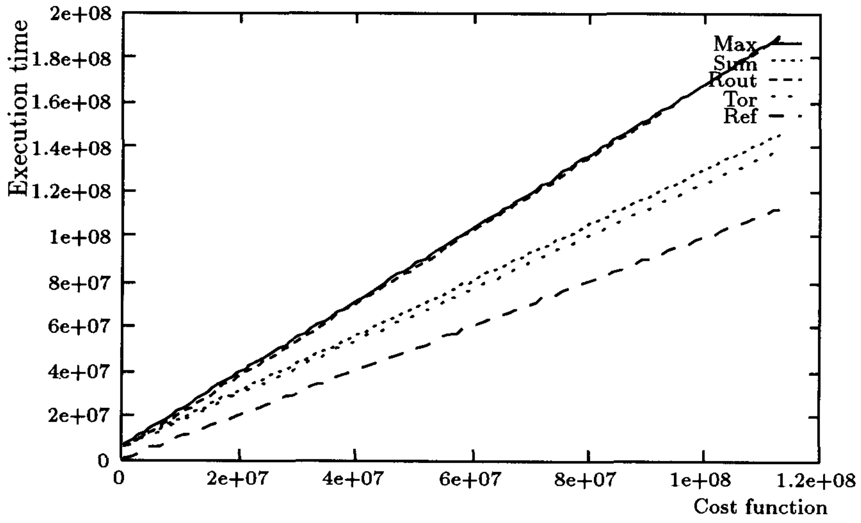

Fig. 4. Regression between cost functions and execution times

In this figure, *Ref* represents the experimental execution time. The best cost function is *Tor*: torus topology with VCR routing tables.

Experimental results. A very large number of experiments were done using synthetic programs. The greedies always delivered their results in less than one second on a workstation, while iterative algorithms could spend up to one hour. To summarize:

- LPTF is better than modulo in most of the tests and may be sufficient when communication costs are low.
- Taking into account communication times is important and LGCF is for this reason the best greedy algorithm.
- Iterative algorithms did not result in important improvements relatively to LGCF. Figure 5 summarizes experimental mapping results by giving the per-

centage of improvement resulting from the use of tabu instead of LGCF, for the cost function which takes into account the torus architecture. The improvement remains low for the majority of the tests (about 15% in the worst case). This may be due to several reasons: the graphs of these benchmarks are well-structured and the ratio communication/computation was not chosen high. Tests using random task graphs demonstrated that iterative algorithms perform better for programs having a high communication/computation ratio.

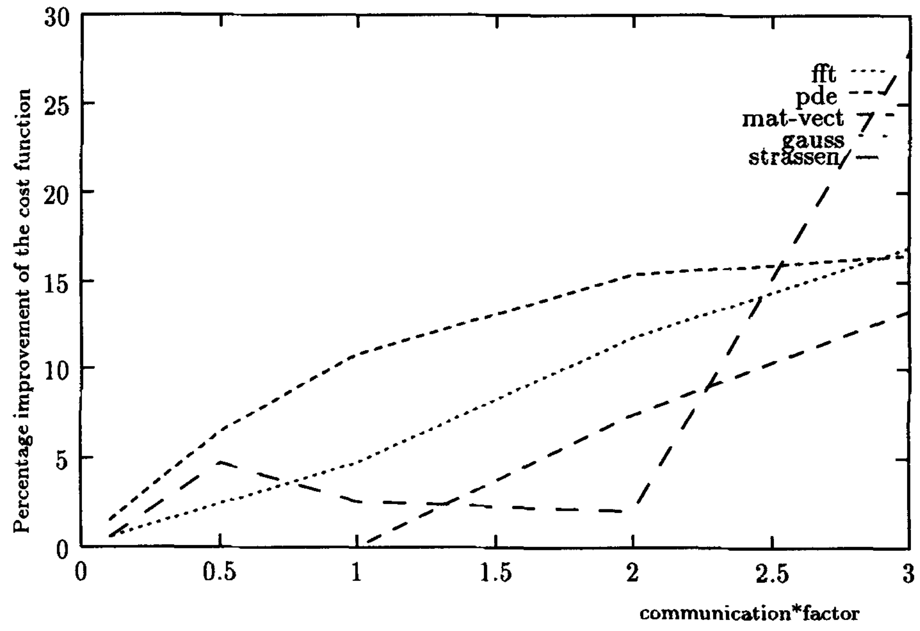

Fig. 5. Comparison between tabu and LGCF

4.3 Use of ALTO

We propose a stepwise process where a first mapping is done with a greedy algorithm. If some parameters are unknown, the greedy uses default parameters. Next, the parallel program is executed using monitoring tools. Monitoring results are used to run an iterative mapping algorithm. If the mapping is not good enough, these steps can be repeated as long as needed and the statistical analysis will try to improve the mapping. Usually, one step is sufficient.

Monitoring is used to determine the mapping parameters, computation and communication costs, and thus improve the quality. Monitoring gives the total execution time of a program, the number of bytes communicated between each pair of tasks, the computing time of each task, the idle time of each task (the time wasted by waiting communications), the total idle time of each processor, and the number of bytes communicated between each pair of processors.

In our benchmarks, the knowledge of communication and computation costs has proved to be very important. To simulate the behavior of an user having only an approximate knowledge of these values and to use the possibility of a feedback mechanism (to use the post-mortem trace analysis), we modified the information given to the tabu for the graph corresponding to the matrix-vector product. In figure 6, the communication costs were multiplied by x for the mapping. The results stress the importance of having a correct estimate of communication times to generate an efficient mapping. They also indicate that it is better to overestimate than to underestimate communication costs.

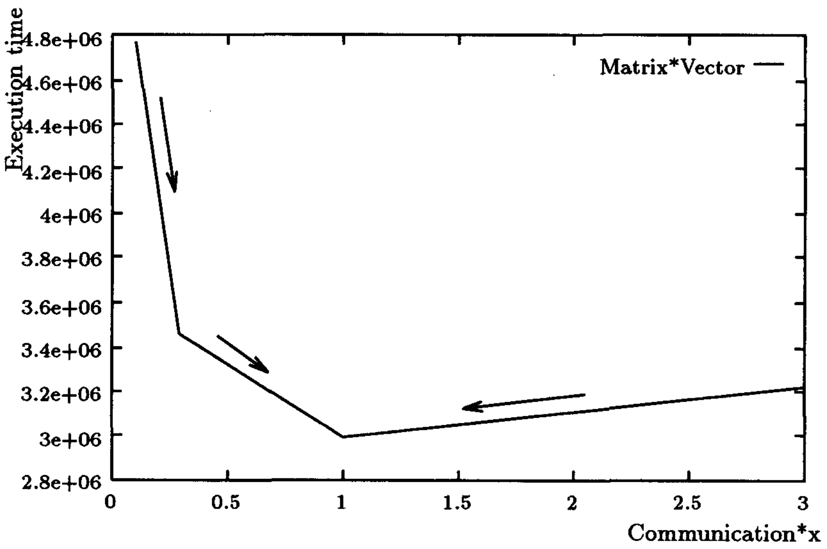

Fig. 6. Improvement of a mapping using feedbacks

5 Conclusion

The ALTO toolbox includes greedy algorithms, *modulo, LPTF, LGCF and struct-quanti* as well as iterative ones, *simulated annealing* and *tabu search*. ALTO was thoroughly assessed to study both the efficiency of the mapping algorithms, that is the time taken to deliver a solution, and the quality of the mapping, that is the value of the cost function optimized by the mapping algorithm.

The first experiments were performed using randomly generated task graphs. A large number of more realistic experiments were performed using synthetic programs modeling the most classical numerical algorithms, executed on a transputer based architecture including 128 processors. These were used to assess the cost functions which can be used in ALTO and which proved to be linearly related to the program execution times. Therefore any of them can be used in the mapping algorithms since decreasing its value will result in improved execution time.

Another experiment studied the sensitivity of the mapping result to the cost estimates done for the programmer for the computation of tasks and inter-tasks communication volumes. It indicated the interest of combining the mapping tools with monitoring tools. The main result of the large number of experiments done with ALTO is that a mapping tool should include several mapping algorithms and a method to use these algorithms.

References

1. J. Błażewicz, P. Bouvry, D. Trystram, and R. Walkowiak. A tabu search algorithm for solving the mapping problem. 1995. European Conference on Combinatorial Optimization, ECCO'95.
2. S. W. Bollinger and S. F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *ICPP*, 1988.
3. P. Bouvry, J.-P. Kitajima, B. Plateau, and D. Trystram. Andes: A performance evaluation tool, application to the mapping problem. *submitted for publication.*
4. T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 1988.
5. F. Glover and M. Laguna. *Tabu Search, a chapter in Modern Heuristic Techniques for Combinatorial Problems.* W.H. Freeman, N-Y, 1992.
6. P. Haden and F. Berman. A comparative study of mapping algorithms for an automated parallel programming environment. Technical Report CS-088, UC San Diego, 1988.
7. J.G. Harp, C.R. Jesshope, T. Muntean, and C. Whitby-Stevens. The development and application of a low cost high performance multiprocessor machine. In *ESPRIT'86: results and achievements*, Amsterdam, 1986. North Holland.
8. J. Kitajima. *Modèles Quantitatifs d'Algorithmes parallèles.* PhD thesis, Institut National Polytechnique de Grenoble, Grenoble - France, November 1994. in french.
9. M. Norman and P. Thanish. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, September 1993.
10. B. Plateau. Présentation d'APACHE. Rapport APACHE 1, IMAG, Grenoble, October 1994. Available at *ftp.imag.fr:/imag/APACHE/RAPPORTS.*
11. T. Yang and A. Gerasoulis. PYRROS: static scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th ACM International Conference on Supercomputing*, pages 428–437. ACM, July 1992.