Hill-Climbing, Simulated Annealing and Genetic Algorithms: A Comparative Study and Application to the Mapping Problem

E-G.TALBI & T.MUNTEAN
Institut IMAG - Laboratoire de Génie Informatique (*)
University of Grenoble

Abstract

Hill-climbing, simulated annealing and genetic algorithms are search techniques that can be applied to most combinatorial optimization problems. In this paper, the three algorithms are used to solve the mapping problem: optimal static allocation of communicating processes (tasks, objects, agents) on distributed memory architectures.

Each algorithm is independently evaluated and optimized according to its parameters. The parallelization of the algorithms is also considered. As an example, a massively parallel genetic algorithm is proposed for the problem, and results of its implementation on a 128-processor Supernode (reconfigurable network of transputers) are given.

A comparative study of the algorithms is then carried out. The criteria of performances considered are the quality of the solutions obtained and the amount of search time used for several benchmarks. A hybrid approach consisting in a combination of genetic algorithms and hill-climbing is also proposed and evaluated.

Key Words

Mapping problem, Distributed memory parallel architectures, Genetic algorithms, Hill-climbing, Simulated annealing.

(*) Authors adress: IMAG/LGI Laboratory, BP53X F-38041; Grenoble, France.
Phone: (33)76.51.48.64 ; Fax: (33)76.44.66.75
Email: ghazali@imag.fr & traian@imag.fr

I. INTRODUCTION

In this paper, we are interested to the mapping problem: optimal static placement of communicating processes on the processors of a distributed memory parallel machine. The problem is known to be NP-complete [Garey79]. Consequently, heuristic methods shall be used. They may find only approximations of the optimum, but they will do it in a "reasonable" amount of time.

Heuristic algorithms may be divided in two main classes. First, the general purpose optimization algorithms independent of the given optimization problem and, on the other hand, the heuristic approaches especially designed for the mapping problem. As we want to avoid the intrinsic disadvantages of the algorithms of this second class (their limited applicability), this paper is only concerned with the first class of algorithms.

Two widely used optimization techniques are the hill-climbing algorithm [Johnson85] and simulated annealing [Kirkpatrick83]. Hill-climbing finds the global minimum only in convex spaces. Otherwise, most often it is rather a local instead of a global minimum which is found. Simulated annealing offers a way to overcome this major drawback of hill-climbing but the price to pay is a huge computation time. Worst, simulated annealing algorithm is rather of a sequential nature, its parallelization is quite a difficult task [Greening90].

More distributed optimization techniques, inherently parallel, may also be considered. Some of them are closely related to neural networks algorithms. Others, namely, genetic algorithms are considered in this paper.

Genetic algorithms are stochastic search techniques, introduced by Holland twenty years ago [Holland75]. They are inspired by adaptation in evolving natural systems. They have recently been applied to combinatorial optimization problems in various fields [Neuhaus90][Startweather90], such as, for instance, the traveling salesman problem, the optimization of connections and connectivity of neural networks, and classifier systems. In this paper we propose and compare a parallel genetic algorithm with the techniques above.

0-8186-1060-3425/93 $03.00 © 1993 IEEE

The remaining sections of the paper are as follows. In the next section, we define the mapping problem and classify heuristics that have been proposed. The sections 3, 4 and 5 are devoted respectively to the description and evaluation of hill-climbing, simulated annealing and genetic algorithms. In the sixth section, a comparison of the three algorithms is carried out. Finally, in section 7 a hybrid algorithm which consists in a combination of hill-climbing and genetic algorithms is proposed and evaluated.

II. The mapping problem

A parallel program can be modeled by a graph $G_p=(V_p,E_p)$ where the vertices represent processes and the weights associated to the vertices represent known or estimated computation costs for these processes. The edges represent communication exchanges between processes and their weights estimate the communication costs. We assume here a static graph of processes ; no dynamic process creation is done. Otherwise, a dynamic allocation strategy must be used.

A parallel architecture is also modeled by an undirected connected graph $G_t=(V_t,E_t)$, where vertices represent processors and edges represent communication links between processors. It is assumed that the architecture is static; the configuration of the physical network will not be changed dynamically during run time.

The following terminology is used:

M : the number of processes to be mapped, $M=|V_p|$.

N : the number of processors of the target architecture, $N=|V_t|$.

e_i : the computation cost of process p_i.

c_{ij} : the communication cost between processes p_i and p_j.

d_{kl} : the distance between processors t_k and t_l. The distance is defined as the minimum number of links of a path between the processors.

The mapping problem can be defined by a function $\Pi: V_p \rightarrow V_t$, assigning each process to a processor. A cost function F: $\Pi \rightarrow \Re$, which associates a value to each mapping must be defined to compare the different possible solutions.

Two contradictory mapping criteria have been considered:

- minimize the sum of the total communication costs between processors. This cost may be measured by the product of the communication cost between all pairs of processes and the cost of exchanges between the processors where processes are assigned.

$$MIN(C) = MIN\left(\sum_{i,j \in V_p} c_{ij} d_{\Pi(i)\Pi(j)}\right)$$

- minimize the load imbalance across the system. The quantitative measure used to deal with this criterion is the variance of the loads of the different processors.

$$MIN(V) = MIN\left(\frac{1}{N}\sum_{k=1}^{N} L_k^2 - L^2\right) \qquad L = \frac{\sum_{i=1}^{M} e_i}{N}$$

$$L_k = \sum_{\Pi(i)=k}^{M} e_i$$

The cost function F chosed is a weighted sum of the two functions C and V.

$$F = C + w.V$$

w is the weight of the contribution of the communication cost relative to the computational load balance across the system. Choosing a suitable value for w depends on the knowledge about characteristics of the parallel architecture. Very small values of w would suggest a uniprocessor solution, and very large values would reduce the problem to load balancing without communication costs. The parallel architecture used in our experiments was a Supernode of T800 transputers and w=2 has been estimated by empirical experiment.

The different mapping strategies that have been proposed in the literature are based on one of the following approaches: mathematical programming [Ma82], graph theory [Shen85], and queuing theory [Bryant81] (fig.1). They give optimal solutions but are time consuming. To speed up the search, approximate algorithms have been used; they are based on one of the above optimal approaches but are limited by the search time used [Kasahara84]. Another solution to the problem is the utilization of heuristics (process clustering [Lo88], routing limitation [Bokhari81]). They may be divided in two categories: greedy and iterative. The greedy algorithms are initialized by a partial solution and search to extend this solution until a complete mapping is achieved. At each step, one process assignment is done and we can't change this decision in the remaining steps. Iterative algorithms are initialized by a complete mapping and search to improve it.

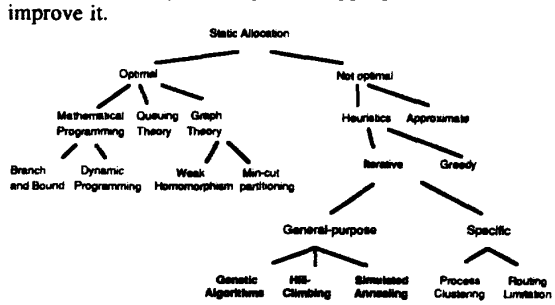

Figure 1 A taxonomy of mapping strategies.

III. Hill-climbing

The hill-climbing algorithm starts with a complete configuration, and tries to improve it by local transformations. A move between neighbouring processors is selected, the cost change of the move is evaluated, and if

the change is positive the move is accepted and a new configuration is generated. Otherwise, the old configuration is kept. This process is repeated until there are no changes to the configuration that will reduce the cost function further. When this occurs a local minimum has usually been found, rather than the required global minimum. Hill-climbing algorithm can be pictured as follows.

- Generate an initial configuration S_0 ($S:=S_0$).
- Repeat
  - compute a neighbouring configuration S' by local transformation.
  - if cost(S') < cost(S) then S:=S'

  Until there is no better move.

In our comparative study, we have considered many versions of the algorithm. They differ by the strategies used in the generation of the initial configuration, in the local transformation, and in the replacement strategy.

We have used two strategies for the generation of the initial configuration. The first one is spreading. It consists in mapping each process on a processor randomly chosen. The second one is *cyclic*. It consists in mapping each process p_i on the processor $t_{(i \bmod N)}$. Thus, the load balancing criterion is taken into account in the initial configuration.

For local transformations, two strategies have been used. The first one moves a given process on another neighbour processor (*movement strategy*). The second one exchanges the mapping location of two processes (*exchange strategy*). The main caracteristic of this strategy is that it conserves the load balancing propriety of a given configuration.

Two replacement strategies have been considered. The first consists in replacing the current configuration by the first neigbouring configuration with a smaller cost. The second one replace the current configuration by the best neighbouring configuration.

We have evaluated the performances of the algorithm with each combination of these strategies. When the movement strategy is used in the local transformation, the initial configuration does not influence very much the solution; the final solutions obtained are very similar. However, when the exchange strategy is used, the process distribution (number of processes per processor) does not change during run time. The load balancing criterion must then be taken into account in the initial configuration with the use of a cyclic strategy.

Therefore, we have used in our experiments four hill-climbing algorithms :

AIRL1 :
- Generate a random initial configuration S_0 ($S:=S_0$).

Repeat
- Generate a neighbouring configuration S' using the movement strategy.
- If cost(S') < cost(S) Then S:=S'.

Until there is no better neighbour.

AIRL2 :
- Generate a random initial configuration S_0 ($S:=S_0$).

Repeat
- Generate all neighbours of S using the movement strategy.
- If cost(S') < cost(S) Then S:=S' (S' is the best neighbour of S).

Until there is no better neighbour.

AIRL3 :
- Generate a cyclic initial configuration S_0 ($S:=S_0$).

Repeat
- Generate a neighbouring configuration S' using the exchange strategy.
- If cost(S') < cost(S) Then S:=S'.

Until there is no better neighbour.

AIRL4 :
- Generate a cyclic initial configuration S_0 ($S:=S_0$).

Repeat
- Generate all neighbours of S using the exchange strategy.
- If cost(S') < cost(S) Then S:=S' (S' is the best neighbour of S).

Until there is no better neighbour.

Figures 2 and 3 show respectively the quality of the solutions obtained and the search time function of the algorithm used. The search time is represented by the number of configurations generated. The benchmarks used are the following:

- benchmark 1 : pipeline of 32 processes and a complete network of 8 processors,

- benchmark 2 : grid of 32 processes and a complete network of 8 processors,

- benchmark 3 : binary tree of 31 processes and a biprocessor.

The communication cost and the execution cost of processes are set to one.

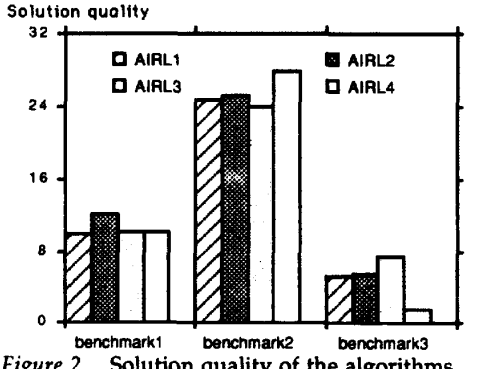

Figure 2 Solution quality of the algorithms. (function F is used)

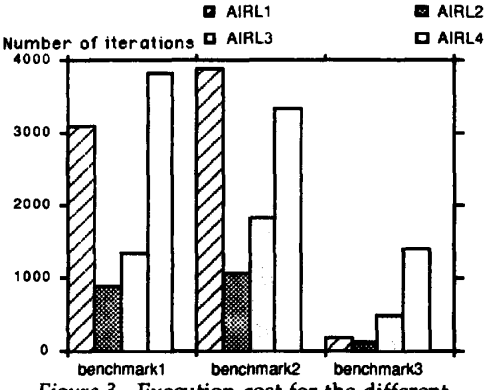

Figure 3 Execution cost for the different algorithms.

For the benchmark 2, the AIRL1 algorithm is better than AIRL4 algorithm concerning the solution quality. For the benchmark 3, the AIRL4 algorithm gives better performances. The performances are quite dependant on the benchmark used, AIRL1 algorithm give a good compromise between the solution quality obtained and the search time used. we chose therfore AIRL1 for representing this class of algorithms.

IV. Simulated annealing

The principle of the simulated annealing algorithm is the following: the system is put in a high temperature environment. At this temperature is applied a sequence of random local transformations (markov chain) to reach the equilibrium at this temperature. Then, the temperature is slightly decreased and a new sequence of random moves is applied. At each temperature the permitted energy states are governed by the metropolis criterion, which allows the configuration to be accepted with a probability P(ΔE,T). The search terminates when the system stabilizes.

The literature on this topic, and the basic algorithm allows considerable variation and tuning of parameters. The number of available changes to the configuration, denoted by L, when moving one process to another processor, is given by L=M*(N-1). This value gives a measure of the size of the problem and is used as a parameter in the annealing schedule. Below the simulated annealing algorithm used in our experiments is pictured.

1. (Initialization step)
   - start with a random initial configuration S_0 ($S:=S_0$);
   - T := Tmax; /* Tmax: initial temperature */
2. (Stochastic hill-climb)
   - generate and compute a random neighbouring configuration S'; ΔE:=cost(S')-cost(S)
   - select the new configuration (S:=S') with probability P(ΔE,T)=min(1,exp(-ΔE/T));
   - repeat this step X*M*(N-1) times; /* length of the markov chain spent at each T */
3. (Anneal/Convergence test)
   - set T:=a.T; /* a: temperature decrease rate */
   - if T≥Tmin goto step2. /* Tmin: minimal temperature */

Figures 4 and 5 show the effect of the temperature decreasing rate on the solution quality and the execution cost of the algorithm. A rapid decrease gives local optimum and a slow decrease gives better results but it is time consuming.

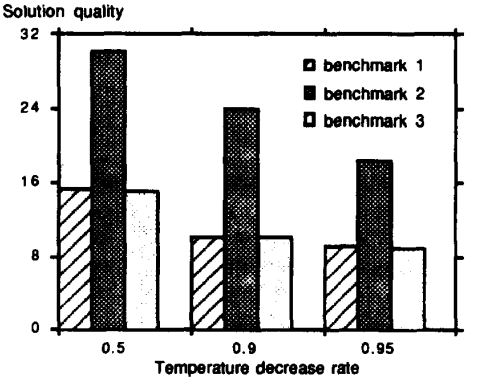

Figure 4 Solution quality.

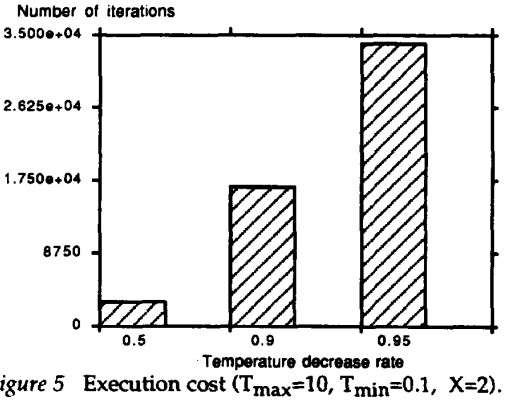

Figure 5 Execution cost (T_{max}=10, T_{min}=0.1, X=2).

Figures 6 and 7 show the effect of the parameter X (length of the markov chain) on the solution quality and the execution cost of the algorithm. As expected, the solution quality is better when X increases, however, the execution cost increases linearly with X.

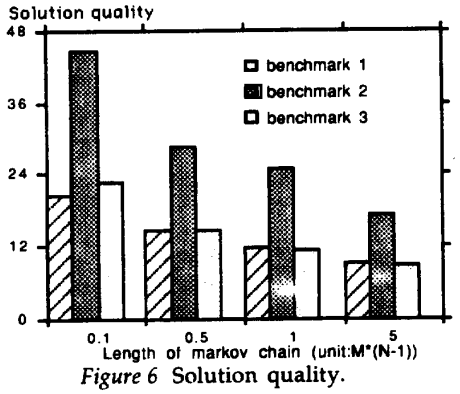

Figure 6 Solution quality.

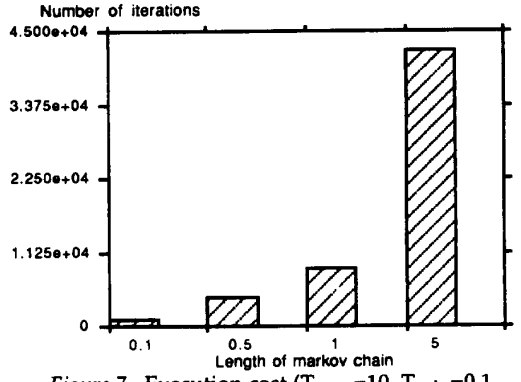

Figure 7 Execution cost (T_{max}=10, T_{min}=0.1, a=0.9).

Results obtained by simulated annealing are good, but the execution time is quite huge. Such onerous run times have driven researchers to implement the algorithm on multiprocessors. To improve performances several techniques have been proposed [Greening90]. *Serial-like algorithms* maintain the properties of sequential algorithms. *Altered generation algorithms* modify state generation to reduce communication, but retain accurate cost calculations. *Asynchronous algorithms* reduce communication further by calculating cost with outdated information to get a better speedup.

V. Genetic algorithms

5.1. Basic principle

Genetic algorithms compose a very interesting family of optimization algorithms. Their basic principle is quite simple.

Given a search space Σ of size N^M and N symbols: any point of this space may be represented by a string (individual) of M of these N symbols.

Given a fitness function f from Σ into R associating a real value to any point of Σ.

Given an initial set of strings, called the initial population.

Some genetic operators are used to generate new points of Σ given some old ones in a phase of the process called "reproduction". The fundamental principle of GAs is: "the fitter a string, the most probable its reproduction".

Given that the size of the population is constant, we will inevitably have a competition for survival of the individuals in the next generation. We have a Darwinian "survival of the fittest" situation. A "replacement" phase is then performed; it consists in replacing the worse individuals of the population by the best individuals produced. The genetic process is iterated on the new population until a given number of generations.

The standard genetic algorithm is:
Generate a population of random individuals.
Evaluation - assign a fitness value to each individual.
While number_of_generation ≤ max Do
Selection - make a list of pairs of individuals likely to mate, with fitter individuals listed more frequently.
Reproduction - apply genetic operators to the selected pairs.
Evaluation - assign a fitness value to each offspring.
Replacement - form a new population by replacing worst individuals by best ones.

The genetic operators used during reproduction are crossover and mutation. Crossover, is defined by given two strings, cut them both at the same random point and exchange the two portions (fig.8a). Crossover is synonymous with sexual reproduction. Mutation is simply flitting a bit (fig.8b). In biological systems, mutation is vital for species survival when the environment is changing. Two parameters need to be defined: P_c and P_m. They represent respectively the probability of application of the crossover and mutations operators.

Parents	Offspring
01100011101	0110**000001**
→	
1101**010001**	1101**011101**

(a) Crossover

Individual 0110001101 → 0100001100
(b) Mutation

Figure 8 Genetic operators.

Considering virtual massively parallel architectures, we chose a parallel fine-grained model, where the population is mapped on a connected processor graph, one individual per processor [Muntean91]. We have a bijection between the individual set and the processor set. The neighbourhoods of

different individuals overlap. The selection is done locally in a neighbourhood of each individual. Selection depends only on local information.

The choice of the neighbourhood is the adjustable parameter. To avoid overhead and complexity of routing algorithms in parallel distributed machines, a good choice may be to restrict neighbourhood to only directly connected individuals. Another motivation behind local selection is biological. In nature there is no global selection. Instead, natural selection is a local phenomenon, taking place in an individual's local environment.

The following is a pseudo-Occam description of the parallel genetic algorithm (PGA) used in our experiments. The PAR and SEQ constructors in Occam stand respectively for simultaneous and sequential execution of processes at the same level of indentation.

```
PAR j=0 FOR number_of_processors
   -- process executed in parallel by each processor
   SEQ
      Generate (local)
      Evaluate (local)
      While number_of_generation ≤ max  Do
         SEQ
            -- communication phase (selection)
            PAR  i=0  FOR nber_of_neighbors
               PAR
                  neighbor_in[i] ? neighbor[i]
                  neighbor_out[i] ! local
            -- computation phase
            PAR i=0 FOR nber_of_neighbors
               SEQ
                  -- crossover and mutation
                  Reproduction(local,neighbor[i])
                  Evaluate(offspring[i])
            Replacement
```

Each reproduction produces two offsprings. Our strategy is to choose randomly one of the offsprings. The replacement phase is deterministic. It consists in replacing the current local individual with the best local offspring produced in the reproduction phase.

Algorithm complexity

In this section we give the complexity of the PGA. The following notations are used:

n : population size,
s : neighbourhood size,
t : individual length.

We begin by calculating the complexity of the standard GA. This requires the complexities of the different steps of the algorithm (selection, crossover, mutation, replacement). The evaluation step is not considered because it depends on the optimization problem treated.

The complexity of the selection step is $o(n^2)$. The crossover operator needs o(t), which give a complexity of o(n.t) for the whole population. We have the same complexity for the mutation operator o(n.t). The complexity of the replacement step is o(n.log(n)). For the complete GA we have then a complexity of $o(n^2+n.t)$.

For the PGA the complexity of the selection step is o(s). The reproduction step has a complexity of o(s.t) as well as for the crossover and the mutation. The complexity of the replacement step is o(s). Table 1 summarizes the GA and the PGA complexities.

	GA	PGA
Selection	o(n.n)	o(s)
Crossover	o(n.t)	o(s.t)
Mutation	o(n.t)	o(s.t)
Replacement	o(n.log(n))	o(s)
Total	o(n.n+n.t)	o(s.t)

Table 1 GA and PGA complexities.

SuperNode Implementation

The Supernode is a loosely coupled, highly parallel machine based on transputers. One of its most important characteristics is its ability to dynamically reconfigure the network topology by using a programmable VLSI switch device. This architecture offers a range of 16 to 1024 processors, delivering from 24 to 1500 Mflops performance. To achieve these performance, a hierarchical structure has been adopted. The basic component is a T800 transputer. It is a 32-bit microprocessor, with on-chip memory and F.P.U. (Floating Point Unit), delivering 10Mips and 1.5Mflops peak performance. Communication between transputers is supported by 4 bidirectional, serial, asynchronous, point-to-point connection links. An Unix workstation is used as a host to provide the connection between the root processor and the external world.

The programming environment used in our experiments is on a Parallel C language. A configurator of the physical network has been used to obtain the desired topology of the architecture.

The population is placed on a torus. Given the four links of the transputer, each individual has four neighbours. No routing is needed in the processor network because only directly connected processors have to exchange information.

We do not consider the best solution found globally since the communication involved to find out this solution would considerably increase. We only pick up the best solution routing through a "spy process" placed on the "root processor" (fig.9).

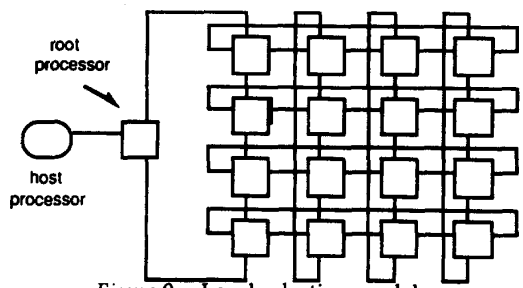

Figure 9 Local selection model.

To use genetic algorithms for the mapping problem, the following formalism is used: let us suppose that we have M communicating processes to map on a parallel architecture of N processors. Each of these processors is labelled by a symbol (for instance an integer between 0 and N-1). A given mapping is represented by a M vector of those symbols; where symbol p in position q means that process q has been placed on processor p.

5.2 Performance evaluation of the PGA

The purpose of the first evaluation is to measure the speed-up when running the parallel genetic algorithm (for a given population size) on different sizes of a torus .

We use the speed-up ratio as a metric for the performance of the parallel genetic algorithm. The speed-up ratio S is defined as S=Ts/Tp where Ts is the execution time on a single processor and Tp corresponds to execution time for a p processors implementation. Figure 10 shows the results obtained.

The algorithm has a near-linear speed-up. This is due to the fact that the communication cost between processes is relatively small compared with the computation cost, and is independant of the size of the architecture.

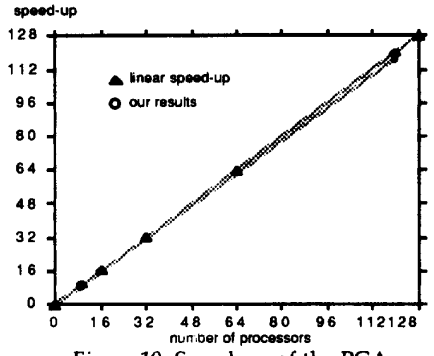

Figure 10 Speed-up of the PGA .

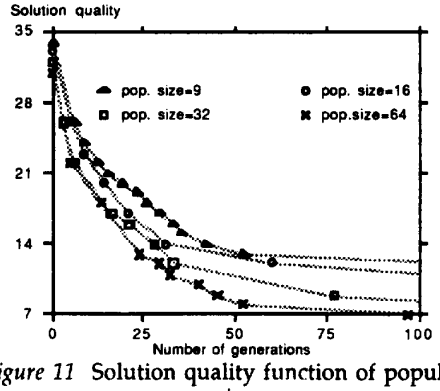

Figure 11 Solution quality function of population size.

The purpose of the second evaluation is to measure the evolution of solution's quality when running the parallel genetic algorithm with different sizes of population.

Figure 11 shows the obtained results. Notice that given the benchmark used (a pipeline of 32 processes to be mapped on a complete network of 8 processors), the best possible solution scores 7. As expected, for a given number of generations, the solution quality improves with an increase of population's size. It may even happen that for a too small population a premature convergence occurs and that the optimal solution will not be ever reached.

VI. Algorithms' comparison

In this section, the performances of the three algorithms are compared. Each algorithm was run 10 times to obtain an average performance estimate. The annealing schedule and the genetic algorithm parameter's that have been used during our experiments are given in table 2. The genetic algorithm was run until no significant improvement was obtained. A significant number of experiments were performed which are not described here due to space limitations.

Symbol	Value
Tmax	10
Tmin	0.05
a	0.9
X	10

(a) Simulated annealing

Symbol	Value	Description
Pops	120	Population size
Pc	0.3	Probability of crossover
Pm	0.04	probability of mutation

(b) Genetic algorithm

Table 2 Parameters of the algorithms.

The tables below show the minimum, maximum, average value and the variance of the obtained solutions for different benchmarks. The results for the hill-climbing and the simulated annealing algorithms are based on an implementation on a single T800 transputer.

Algorithm	Solution				CPUtime (seconds)
	min	max	mean	variance	
H.climbing	8.5	12	9.75	1.3	13
S. annealing	7	7	7	0	398
Genetic	7	7	7	0	10

Table 3 Benchmarking with a pipeline of 32 processes and a ring of 8 processors.

Algorithm	Solution				CPUtime (seconds)
	min	max	mean	variance	
H.climbing	12.37	21.37	17.47	6.89	60
S. annealing	4.37	10.37	7.57	3.16	1651
Genetic	6.37	9.37	7.47	1.29	62

Table 4 Benchmarking with a binary tree of 63 processes and a grid of 4 processors.

Algorithm	Solution				CPUtime (seconds)
	min	max	mean	variance	
H.climbing	28	40	34.19	10.36	61
S. annealing	16	25	18.89	10.09	1313
Genetic	16	21	18.1	4.48	57

Table 5 Benchmarking with a grid of 64 processes and a complete network of 4 processors.

It can be observed from tables 3, 4 and 5 that the results obtained by simulated annealing are better than those of hill-climbing, but they are slower. The tables also indicate that a mapping comparable in quality can be obtained by simulated annealing and genetic algorithm, but genetic algorithm is less time consuming than simulated annealing, which illustrates the efficiency of the genetic search process.

Figure 12 gives the evolution in time of the solution obtained by simulated annealing and genetic algorithms. They are both executed on a uniprocessor (T800 transputer). We show that for genetic algorithms the greatest reduction in the cost of the mapping occurs at the beginning. Thus a moderate quality mapping can be obtained very quickly.

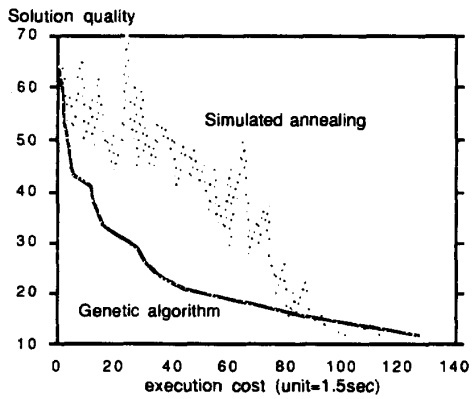

Figure 12 Evolution in time of the quality of the solution .
(benchmark : grid of 32 processes and a ring of 4 processors, σ=12)
(Pops=64, Pc=0.3, Pm=O.04, Tmax=10, Tmin=0.05, X=10, a=0.9)

VII. A hybrid genetic algorithm

The initial configuration has a great effect on the search time used and the solution quality obtained. In the present version of the genetic algorithm, the initial population is generated randomly. An interesting experiment is to generate the population by using heuristics. We have evaluated the genetic algorithm where the initial population is generated by a hill-climbing algorithm.

Figure 13 and table 6 show that the solution quality obtained by the hybrid algorithm are better but the price to pay is a more important execution cost.

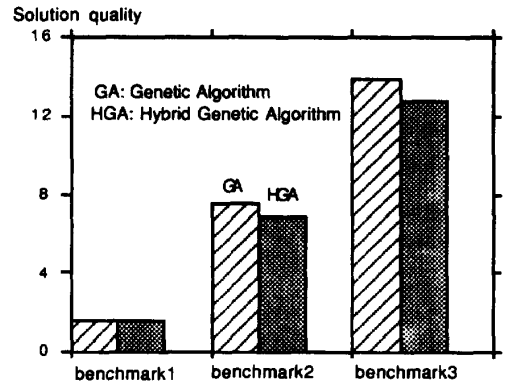

Figure 13 Solution quality function of the algorithm used.
(benchmark : a binary tree of 31, 63 and 127 processes and a grid of 4 processors).

Algorithm	Genetic algorithm	Hybrid algorithm
benchmark 1	2	4
benchmark 2	63	172
benchmark 3	1390	3407

Table 6 Execution time of the algorithms (in seconds).

VIII. Summary and Conclusions

Three general purpose optimization algorithms have been used to solve the mapping problem: hill-climbing, simulated annealing, and genetic algorithms. Each algorithm has been independently evaluated and optimized according to its parameters. A massively parallel genetic algorithm has been proposed as an example for solving such a problem, and results of its implementation on a processor reconfigurable network are given. It will be interesting to emphasize that this parallel algorithm achieves near-linear speed-up.

A comparative study of the algorithms has been carried out. The criteria of performances considered are the quality of the solutions obtained and the amount of search time used for several benchmarks. The results obtained show that

hill-climbing gives worse quality solutions compared to simulated annealing but they are faster. Genetic algorithms give comparable solutions as simulated annealing, but with a search time which is of the same order of magnitude than hill-climbing. The main advantage of genetic algorithms is that they are intrinsically parallel.

A hybrid approach which consists in a combination of genetic algorithms and hill-climbing has been evaluated. The quality of the solutions obtained is better but the price to pay is a greater search time. In order to avoid hill-climbing local minima we are currently investigating a tabu search algorithm [Glover86], another general purpose optimization method.

We are using this parallel genetic algorithm to solve other optimization problems in the field of robotics, medicine and neural networks. Encouraging results have been obtained.

BIBLIOGRAPHY

[Bokhari 81] S.H.Bokhari, "On the mapping problem", IEEE Trans. on Comp., Vol.C-30, No.3, pp.207-214, Mar 1981.

[Bryant 81] R.M.Bryant, J.R.Agre, "A queueing network approach to the module allocation problem in distributed systems", Performance Evaluation Review, Vol.10, No.3, pp.191-204, 1981.

[Garey 79] M.R.Garey, D.S.Johnson, "Computers and intractability: A guide to the theory of NP-completeness", Freeman, San Francisco, 1979.

[Glover 86] F.Glover, "Future paths for integer programming and links to artificial intelligence", Comput. & Ops. Res., Vol.13, No.5, pp.533-549, 1986.

[Greening 90] D.R.Greening, "Parallel simulated annealing techniques", Physica D: Nonlinear phenomena, Vol.42, pp.293-306, 1990.

[Holland 75] J.H.Holland, "Adaptation in natural and artificial systems", Ann Arbor: Univ. of Michigan Press, 1975.

[Johnson 85] D.S.Johnson, C.H.Papadimitriou, M.Yannakakis, "How easy is local search ?", Proc. Annual Symp. of Foundation of Computer Science, pp.39-42, 1985.

[Kasahara 84]] H.Kasahara, S.Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing", IEEE Trans. on Comp., Vol.C-33, No.11, pp.1023-1029, Nov 1984.

[Kirkpatrick 83] S.Kirkpatrick, C.D.Gelatt, M.P. Vecchi, "Optimization by simulated annealing", Science, Vol.220, No.4598, pp.671-680, Mai 1983.

[Lo 88] V.M.Lo, "Algorithms for static task assignment and symetric contraction in distributed systems", Proc. of the 11th Int. Conf. on Parallel Processing, the Penn. State Univ. Press, pp.239-244, Aou 1988.

[Ma 82] P.R.Ma, E.Y.Lee, M.Tsuchiya, "A task allocation model for distributed computing systems", IEEE Trans. on Comp., Vol.C-31, No.1, pp.41-47, Jan 1982.

[Muntean 91] T.Muntean, E-G.Talbi, "A parallel genetic algorithm for process-processors mapping", Int. Conf. on High Speed Computing II, Montpellier, M.Durand and F.El Dabaghi (Editors), Elsevier Science Pub., North-Holland, pp.71-82, Oct 1991.

[Neuhaus 90] P.Neuhaus, "Solving the mapping problem - Experience with a genetic algorithm", in PPSN I, LNCS No.496, Oct 1990.

[Shen 85] C-C.Shen, W-H.Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minmax criterion", IEEE Trans. on Comp., Vol.C-34, No.3, pp.197-203, Mar 1985.

[Starkweather90] T.Starkweather & al., "Optimization using distributed genetic algorithms", in PPSN I, LNCS No.496, Oct 1990.