A Parallel Genetic Algorithm for task mapping on parallel machines

S. Mounir Alaoui[1], O. Frieder[2], and T. El-Ghazawi[3]

[1] Florida Institute of Technology, Melbourne, Florida, USA
salim@ee.fit.edu,
[2] Illinois Institute of Technology, Chicago, Illinois, USA
ophir@csam.iit.edu,
[3] George Mason University, Washington D.C, USA
tarek@gmu.edu

Abstract. In parallel processing systems, a fundamental consideration is the maximization of system performance through task mapping. A good allocation strategy may improve resource utilization and increase significantly the throughput of the system. We demonstrate how to map the tasks among the processors to meet performance criteria, such as minimizing execution time or communication delays. We review the Local Neighborhhod Search (LNS) strategy for the mapping problem.We base our approach on LNS since it was shown that this method outperforms a large number of heuristic-based algorithms. We call our mapping algorithm, that is based on LNS, Genetic Local Neighborhood Search (GLNS), and its parallel version, Parallel Genetic Local Neighborhood Search (P-GLNS). We implemented and compared all three of these mapping strategies. The experimental results demonstrate that 1) GLNS algorithm has better performance than LNS and, 2) The P-GLNS algorithm achieves near linear speedup.

1 Introduction

Parallel processor systems offer a promising and powerful alternative for high performance computing. Inefficient mapping or scheduling of parallel programs on architectures, however, limit our success. A parallel program is a collection of separate cooperating and communicating modules called tasks or processes. Tasks can execute in sequence or at the same time on two or more processors. Task mapping distributes the load of the system among its processors so that the overall processing objective according to given criteria is maximized. An efficient allocation strategy avoids the situation where some processors are idle while others have multiple jobs queued up.
A study in 1994 [8] demonstrated the power of the ***Local Neighborhood Search*** (LNS) algorithm for task allocation over a wide range of methods. We develop two new load balancing algorithms based on LNS. The first, called ***Genetic Local Neighborhood Search*** (GLNS), is a genetic algorithm based on LNS. The second is a coarse grain parallel implementation of GLNS, we call

it ***P-GLNS***. In the reminder of this paper, we present our algorithm GLNS and a parallelization of this algorithm called P-GLNS. Our experimental results demonstrate that GLNS and P-GLNS achieve better mappings than LNS and P-GLNS yields near linear speedup.

2 Background

A popular classification scheme for task scheduling algorithms was introduced by Casavant [2]. This classification scheme, at the highest level, distinguishes between local and global scheduling. Local scheduling schedules the execution of processes within one processor . Global scheduling relates to parallel and distributed machines and determines on which processor to execute a job.
The next level in the hierarchy differentiates between dynamic and static scheduling. In other words, it defines the time where the scheduling decisions are made. In dynamic scheduling, the decisions are made during the execution of the program. The scheduler dynamically balances the workload each time there is unbalance. In this case, the process of load balancing can be made by a single processor (nondistributed), or it can be distributed physically among processors (distributed).
In the case of distributed dynamic global scheduling, we distinguish between cooperative and noncooperative scheduling. In cooperative scheduling, a processor balances the load cooperatively with other processors. In a noncooperative scheduling, a processor balances the load based only on the information it has locally. Each processor is independent of the actions of other processors. The major disadvantages of dynamic distributed load balancing algorithms is the runtime overhead for load information distribution, making placement decisions, and transferring a job to a target host continually during program execution.
In static scheduling, the decisions are made at the compilation stage, before the program execution. The resource needs of processes and the information regarding the state of the system are known. An "optimal" assignment can be made based on selected criteria [16], [20]. This problem, however, is generally NP-complete [15]. Sub-optimal solutions can be divided into two categories. Algorithms in the first category (approximate) of the "suboptimal" class consist of searching in a sub-part of the solution space, and stopping when a "good" solution is obtained. The second category is made up of heuristic methods, which assume a priori knowledge concerning processes, communication, and system characteristics.
Another classification scheme is Sender- or Receiver-initiated [5]. A load balancing algorithm is said to be sender-initiated if the load balancing process is initiated by the overloaded nodes; whereas for receiver-initiated, the underloaded nodes initiate the algorithm. A combination of both sender and receiver initiated yields symmetrically-initiated algorithms. The load balancing can be initiated either by the overloaded node if the load exceeds a pre-determined threshold, or by the underloaded node if the load index drops below a given threshold.
In 1994, Benmohammed-Mahieddine, et al [1] developed a variant of symmetri-

cally initiated algorithm: the PSI (Periodic Symmetrically-Initiated) that outperformed its predecessor.
We focus on global, static and sub-optimal heuristic scheduling algorithms. Talbi and Bessiere developed a parallel genetic algorithm for load balancing[6]. Their implementation is based on the assumption that all the communication latencies between processes are known. They also compared their results with those found by the simulated annealing method. They demonstrated that genetic approach gives better results. A paper by Watts and Taylor [11] exposes a serious deficiency in current load balancing strategies, motivating further work in this area.

3 A Genetic Local Neighborhood Search

As described in Golberg [4], in general terms, a genetic algorithm consists of four parts.

1) Generate an initial population
2) Select pair of individuals based on the fitness function.
3) Produce next generation from the selected pairs by applying pre-selected genetic operators.
4) If the termination condition is satisfied stop, else go to step 2.

The termination condition can be either:
1) No improvement in the solution after a certain number of generations.
2) The solution converges to a pre-determined threshold.

3.1 Initial Population

The representation of an individual (mapping) is similar to the one in [6]. This representation satisfies two constraints: 1) We must be able to represent all the possible mapping solutions, and 2) We must be able to apply genetic operators on those individuals. A mapping is represented by a vector where number p in position q means that process q has been placed on processor p. Based on the three cost functions of LNS, we define our general cost function (GCF). GCF is the fitness function in our genetic approach. The goal is to minimize this general cost function. GCF is defined as a weighted sum of the three LNS cost functions.

$$GCF = w1 \times H + w2 \times L + w3 \times D$$

The weights w1, w2, and w3 depend on the mapping problem. In our experiment, these weights are chosen equal to those used in the LNS algorithm [19] to be able to compare the two algorithms.

3.2 Individual representation

3.3 selection scheme

Two common Genetic Algorithms selection schemes were tested:
rank selection and roulette wheel selection [4]. Rank selection selects the fittest individuals to take part in the reproductive steps. This method opts for more exploitation at the expense of exploration. The second scheme is the roulette wheel selection. In this selection the search is more randomized but we still give to the fittest individuals more probability to be selected. These two different schemes are used in Genetic Algorithms depending on the problem. In our genetic algorithm we chose rank selection. We found experimentaly that this selection converges faster than the roulette wheel selection for the mapping problem[19].

3.4 crossover

The best individuals resulting from the selection step are paired up. The two elements of each give birth to two offsprings. The crossover operator combines two elements of a pair to produce two new individuals. This operator picks randomly a crossover point number (between 0 and the size of an individual). It combines the pair, depending on this number, as shown in Figure 3. This new pair constitutes a pair replacing the parent pair in the new population. The crossover operator combine all the pairs to form the new population.

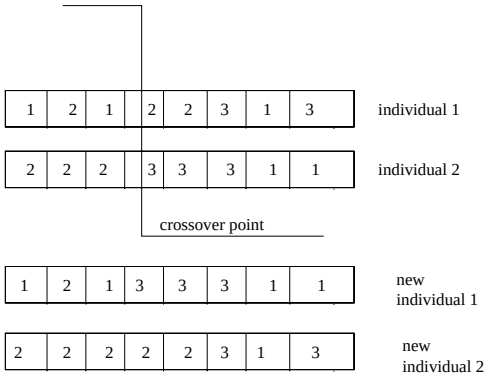

Fig. 1. Crossover

3.5 mutation

A mutation is to applied to a random transformation on the individuals after a certain number of generations. This mutation potentially explores new solutions and introduces random behavior into the search. The use of mutation avoids potentially limiting the search space due to local minimum values.
Mutation occurs after a certain number of generations. Its objective is to change randomly one bit or gene. In addition to traditional mutation, we also evaluate our adaptive mutation. That is, in adaptive mutation we mutade only if the fitness of the best individual is not increased after N generations. The value of N is determined experimentally. We ran our genetic algorithm 50 times starting with different initial populations each time. Six different mutation rates were taken into consideration: 0.1, 0.3, 0.5, 0.7, 0.9 and the adaptive method, we found experimentally that our adaptive mutation yields a better general cost function value.

3.6 Experimental Results

We experiment with five different data sets (DATA1, DATA2, DATA3, DATA4, DATA5). Each data set corresponds to a different communication graph. LNS and GLNS algorithms are runned on these five data sets. We plot the results after 100,200...800 generations for each algorithm (Table 2).

	DATA1		DATA2		DATA3		DATA4		DATA5	
Generations	LNS	GA	LNS	GA	LNS	GA	LNS	GA	LNS	GA
0	7.28	7.25	10.93	10.93	6.88	6.72	6.44	6.44	7.28	7.03
100	6.87	6.88	10	8.90	4.71	3.11	5.60	3.14	6.87	6.88
200	6.87	6.88	9.83	8.82	4.65	2.87	4.82	3.08	6.87	6.88
300	6.66	6.32	9.83	8.73	4.20	2.60	4.28	3.08	6.66	6.72
400	6.11	5.89	9.44	8.55	3.88	2.60	4.04	3.02	6.11	6.02
500	5.95	5.69	9.23	8.55	3.88	2.43	3.88	3.02	5.95	5.69
600	5.22	4.89	9.36	8.48	3.27	2.40	3.64	2.97	5.95	5.64
700	5.22	4.75	8.85	8.48	3.20	2.40	3.64	2.97	5.22	5.18
800	5.22	4.75	8.85	8.33	3.20	2.33	3.64	2.86	5.22	5.18

Table 1. performance of GLNS VS LNS

Table 2 summarizes the performance of both LNS and GLNS. We run the two algorithms on five different data sets. The weights used for the genetic algorithm are the same than for the LNS algorithm: W1=1, W2=0.1, W3=0.01 The GA is the GCF value of the best member of the population at termination.

For all the data sets the Genetic Algorithm converge to a better solution than LNS. For the data set DATA1 for example the GA converges to a value of 4.75 after 800 generations. The LNS algorithm converges to 5.22 which is much greater than 4.75. The worst case is with the last data set. The GA converge

to a value of 5.22 after 800 generations not much better than the LNS value of 5.18, but before 400 generations LNS found better results than GA. The Genetic Algorithm outperforms the LNS algorithm which is known to be better than a wide range of methods including simulated annealing.

4 Parallel Genetic Local Neighborhood Search

4.1 Parallelization strategies

Evolution is a highly parallel process. Each individual is selected according to its fitness to survive and reproduce. Genetic algorithms are an abstraction of the evolutionary process and are indeed very easy to parallelize. Nowadays, the coarse grain parallelization is the most applied model for parallel genetic algorithms [13], [12] [7] [18]. This model divides the population into few sub-populations and an individual can migrate from one sub-population to another one. It is characterized by three parameters, namely, the ***sub-population size***, the ***migration rate*** wich defines how many individuals migrate, and the ***migration interval*** that represents the interval between two migrations. The main problem is to define the migration rate and the migration interval correctly [7]. We use this model in our computation. We have implemented both GLNS and P-GLNS algorithms. P-GLNS is implemented on a network of workstations using Parallel Virtual Machine (PVM) language.

For this parallelization we define three parameters. The first one is the sub-populations sizes. The second is the migration rate which define how many individuals migrate. The third is the migration interval that represents the interval between two migrations.

4.2 strategy

We use a master-slave model. The processors that are executing the genetic algorithm on a subpopulation send their results to a master after a fixed number of generations. This master will select the best individuals and broadcast them to slaves. Those slaves will work on those new individuals combined with the previous ones. This method is quite similar to the one used in [14].

4.3 migration rate

The migration rate corresponds to the number of individuals that migrate each time there is a migration. The migration rate is taken as half of the population size. In other words at each migration the processors send to the master half of their population. The individuals sent are the best of the corresponding subpopulation. A larger migration rate will increase the communication cost between processes. It will be shown that a migration rate of 1/2 is the best for the problem parameters considered.

4.4 migration interval

The migration interval represents the interval between two migrations. This migration interval is determined experimentally. After a certain number of generations the processors send their best individuals to the master. It will be shown that a migration interval of 110 is the best for the problem parameters considered.

5 Experimental Results

5.1 Experiments

The experiments were conducted on a network of workstations. For our parallel implementation we used the Parallel Virtual Machine message passing library (PVM). We run P-GLNS to define the best migration rate and the best migration interval for the problem parameters used. To compute the speedup we runned P-GLNS and GLNS on three different data sets.

We found experimentally that the minimum cost is achieved with a migration rate of 0.5. The more individuals we send, the bigger is the migration rate. This results in more interaction between the nodes. This explains the decreasing cost between rates 0 and 0.5. However, when the number of individuals sent becomes larger, the communication cost becomes higher. This explains the increasing cost between 0.5 and 0.8 migration rates. For the problem parameters used, 0.5 is the perfect trade off point between communication cost and interaction between nodes. The position of such trade off point should be determined experimentally for each case. In the same way, the best migration interval found is 110.

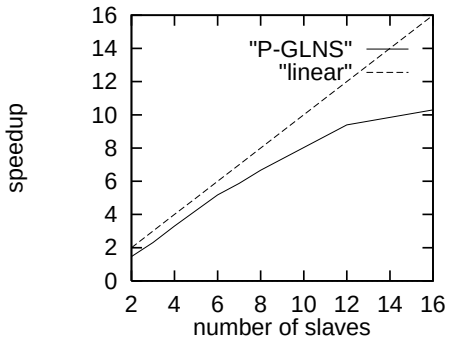

Fig. 2. speedup

For the parallel implementation three different mapping problems are taken into consideration. For each one we perform 100 runs and compute the sequential and the parallel time to find an acceptable solution (fitness greater than some

threshold). The speedup is the ratio between these two time measurements. Our algorithm was run on a 10 Base-T ethernet network. We can see in (Figure 5) that for 2,4 and 8 slaves we have a near linear speedup with our experiments on a 10 Base-T network. Assuming, we are using a 100 Base-T network the speedup will be much better when larger number of slaves are used. Thus, near linear speedup can be obtained in any parallel processing environement with a reasonable interprocessor communication network. Each slave works independently of other slaves. They search during a certain number of generations in their search space until they reached a sub-solution. The best sub-solution will be the start of new searchs. In the sequential genetic algorithm the best individuals mate together. There is no possibility with a ranking selection that two locally-optimal solutions mate together. On the other hand, in a parallel implementation each individual has its own sub-optimal solutions, and they are mixed without other sub-populations interferences. This fact makes the parallel search powerfull and efficient. For a large number of stations the communication cost become higher, and this explain the decreasing speedup with 12 and 16 processors in a 10 Base-T network.

6 Conclusion

Based on the Local Neighborhood Search algorithm we developed a coarse graine parallel genetic algorithm called Parallel Genetic Local Neighborhood Search (P-GLNS). The fitness function of P-GLNS is a weighted sum of the three LNS cost functions[19]. Our algorithm balances communication requests and takes into consideration the case where the communication latencies are unknown. Both GLNS and P-GLNS were implemented for different data sets. Simulation results show that GLNS outperforms the LNS algorithm wich is known to be better than a wide range of methods. The parallelization of G-LNS, P-GLNS achieves a near linear speedup and thus is scalable. We have shown that by increasing the migration rate and migration interval, the performance of the parallel algorithm improves. However, increasing these parameters above certain operation points can result in larger overhead that may decrease the benefit from using parallel genetic algorithms. A trade off can be easily obtained experimentally as demonstrated in this work.

References

1. Benmohammed-Mahieddine, K., P. M. Dew and M. Kara. 1994. " A Periodic Symmetrically-initiated Load Balancing Algorithm for Distributed Systems.
2. Casavant, T. L., and J.G. Kuhl. 1988. " A taxonomy of scheduling in General-purpose Distributed Computing Systems" IEEE Trans. Software Eng., vol. 14, No. 2,
3. Cybenko, G. 1989 "Load balancing for distributed memory multiprocessors" Journal of Parallel and distributed computing. 7: 279-301, 1989.
4. D.E.Goldberg. Genetic Algorithms in search Optimization and Machine learning. Addison Wesley, New York, 1989.

5. Eager, D. L., E. D. Lazowska, and J. Zahorjan. 1986. " A comparison of receiver initiated and sender initiated adaptive load sharing" Performance evaluation, Vol. 6, 1986, pp. 53-68.
6. E-G.Talbi, P.Bessiere : Parallel genetic algorithm for the graph partitioning problem. ACM international Conference on SuperComputing, Cologne, Germany, June 1991
7. Erik Cantu-Paz. A summary of Research on Parallel Genetic Algorithms. Illinois Genetic Algorithms Laboratory http://www-illigal.ge.uiuc.edu/ cantu-paz/publications.html, 01/20/98.
8. F.Berman and Bernd Stramm. Mapping Function-Parallel Programs with the Prep-P Automatic Mapping Preprocessor. University of California, San Diego. January 7,1994.
9. Grosso, P. (1985). Computer simulations of genetic adaptation: Parallel Subcomponent Interaction in a Multilocus Model. PhD thesis, University of Michigan.
10. Horton, G. 1993. "A multi-level diffusion method for dynamic load balancing." Par. Comp. 19, pp. 209-218 (1993). In proc 9th International.
11. J. Watts and S. Taylor. A Practical Approach to Dynamic Load Balancing. Submitted to IEEE Transactions on Parallel and Distributed Systems. 1998.
12. Levine, D. (1994). A parallel genetic algorithm for the set partitionning problem. PhD thesis, Illinois Institute of technology.
13. Li T. and Mashford J. (1990) A parallel genetic algorithm for quadratic assignement. In R.A Ammar, editor, Proceedings of the ISMM International Conference. Parallel and Distributed Computing and Systems, pages 391-394, New York, NY: Acta Press, Anaheim, CA.
14. Marin, F., Trelles-Salazar, O., and Sandoval, F. (1994). Genetic algorithms on lan-message passing architectures using pvm: Application to the routing problem. In Y.Davidor, H-P. Shwefel, and R.Manner, editors, Parallel Problem Solving from Nature -PPSN III, volume 866 of Lecture Notes in Computer Science, pages 534-543, berlin, Germany: Springer-Verlag.
15. M. R. Garey and D.S. Johnson. Computers and Intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, New York 1979.
16. Ni, L. M. and K. Hwang. 1981. "Optimal load balancing strategies for a multiple processors system" in Proc. Int. Conf. Parallel Proc., 1981, pp. 352-357
17. P.J.M van Laarhoven and E.H.L Aarts. Simulated Annealing: Theory and Applications. D.Reidel Publishing Company, Boston, 1987.
18. S. Mounir Alaoui, A. Bellaachia, A.Bensaid, O. Frieder "A Parallel Genetic Load Balancing Strategy" Cluster Computing Conference - CCC '97 March 9-11, 1997 Emory University, Atlanta, Georgia.
19. S. Mounir Alaoui, Master Thesis, Alakhawayn University, Rabat Morocco. June 1997
20. Shen, C. and W. Tsai. 1985. "A graph matching approach to optimal tasks assignment in distributed computing systems using a minimax criterion" IEEE Trans.