Cooperative Genetic Algorithm for Optimization Problems in Distributed Computer Systems*

R. Venkateswaran, Zoran Obradović, C.S.Raghavendra
School of EECS
Washington State University
Pullman, WA, 99164-2752
{rvenkate,zoran,raghu}@eecs.wsu.edu

Abstract

In the proposed algorithm, several single population genetic algorithms with different cross-over and mutation parameters are run as a set of processes that cooperate periodically and exchange information to solve the problem efficiently. The algorithm is less stochastic than the standard genetic algorithm and a distributed implementation is appropriate for application to large scale problems. In particular, we apply it to the static task assignment problem and suggest modifications to solve other optimization problems in distributed computer systems. Preliminary experiments with fairly large-sized problems of allocating 50 tasks among 16 processors indicate that the cooperative algorithm implemented on a network of workstations quickly finds better solutions than those obtained by a standard genetic algorithm. To conclusively show that better solutions are obtained, extensive experiments have to be performed. A distributed implementation of the algorithm is highly suited for such experimentation.

I. Introduction

Current literature discusses three levels in the parallelization of genetic algorithms[2], *cellular*, *global* and *inland*. Cellular level is a subclass of cellular automata. The global level approach uses several processors to efficiently create the next generation and compute the fitness values of the strings in the population. This speeds up the computation, but does not improve the quality of the solution obtained. Our technique, *cooperative genetic algorithm*, belongs to an inland level parallelization. In this algorithm category, where several single population genetic algorithms are run in parallel, the processors explore different areas in the solution space, which enhances the chances of finding better solutions. In addition, the processors cooperate with other processors periodically to solve the problem efficiently. Our algorithm can also incorporate the global level approach to exploit the advantages of both levels of parallelization. Another related inland level parallelization called the Parallel Genetic Algorithm (PGA) is proposed in [4]. In PGA, each processor cooperates with others, maintaining only a part of the population. All processors use identical values for the cross-over and mutation probability. On the other hand, in our approach, each processor maintains the entire population and operates on it using its own cross-over and mutation probabilities (different from other slaves). This makes the algorithm less sensitive to selection of these parameters.

The cooperative algorithm is implemented on a network of work-stations. For a large class of problems, a distributed implementation is much more efficient than a sequential implementation. We have successfully used this technique for the static task assignment to processors in a distributed computer system, which is a well-known NP-complete optimization problem. The problem assumes that the execution time for a given task varies for different processors. Also, the tasks communicate with each other and this contributes to the communication cost. Given some tasks to be assigned to a set of processors, the problem is to find an optimal assignment such that the total cost, which is a function of the total execution time and the total communication cost, is minimized.

To find suboptimal task assignment, Stone formulated this problem using a network flow model and used maximal flow techniques to find optimal solutions for the two-processor problem[5]. He also proposed extensions to find suboptimal solutions for an n-processor problem. Various heuristic solutions using different cost functions were proposed by Lo[3].

In our approach, we assume that the tasks to be assigned are *independent*, that is, they can be executed in any order. The execution times for a particular task are different on different processors. It is assumed that the communication times between tasks assigned to the same processor is negligible, while communication times between tasks assigned to different processors can be significant. It is also assumed that the communication time between any two tasks is much smaller than the execution times of each of the tasks. Here, we address the *static task assignment* problem, that is, the task execution times on different processors and communication costs between tasks are assumed to be available before the algorithm is run. It is easy to extend our approach to solve the *dynamic task assignment* problem.

II. Cooperative Genetic Algorithm

Reliable results using standard genetic algorithm necessitates repeating the experiment with different cross-over and mutation probabilities because the algorithm depends significantly on these parameters. To enhance the performance, we propose a new approach called the *Cooperative Genetic Algorithm*. One of the main advantages of this algorithm is that it is relatively insensitive to the cross-over and mutation probabilities. So, reliable results can be obtained with just one experiment. Another advantage

of this algorithm is that it is appropriate for distributed systems implementation. Our implementation is on a network of work-stations having a master-slave scenario with a single master and several slave processes. The slaves can communicate with each other and also with the master.

In the cooperative genetic algorithm, the slave processes communicate only with the master process. Each slave process is assigned an initial population and uses the basic genetic operations of selection, cross-over and mutation to produce the next generation. Each slave uses a different value for the cross-over and mutation probability and consequently each creates a different new population. After each generation, the slaves pass on their created population to the master. The master then identifies the globally best solutions from all the slaves and the previous generation strings. This globally best set, which is of the same size as the previous population, forms the new population for the subsequent generation. For example, if there are 5 slaves with a population size of 10 each, the master picks the 10 best solutions from the set of 50 solutions to create the population for the next generation. All the slaves use this new population to produce the next generation. The process repeats for the specified number of generations. In this algorithm, the best solution of the previous generation is always present in a given population. Thus, the solution produced can never deteriorate from one generation to the next. Observe also that in this algorithm, every slave benefits from the solutions produced by other slaves. The slaves, thus, cooperate with the master to produce highly fit strings in every generation.

To further improve the algorithm, the heuristic solution is incorporated in the initial population. This guarantees that the solution produced by the genetic algorithm is no worse than the heuristic solution. In all the examples we experimented with, the cooperative genetic algorithm solution was always better than the heuristic solution.

III. Performance Analysis

Here, we compare the performance of the cooperative algorithm using e slaves versus the performance of the sequential algorithm repeated e times. As the sequential algorithm is stochastic, we have to repeat the experiments several times with different parameters to get reliable results. Since this repetition is not necessary in the cooperative algorithm of e slaves, the comparison is appropriate if the sequential experiment is repeated e times.

Let t_{exec} be the execution time for one generation of the genetic algorithm and g the specified number of generations for one experiment. Then the total computation time for the e experiments of the sequential algorithm is

$$t_s = e * g * t_{exec}$$

In contrast, the computation time for the cooperative algorithm implemented on a sequential machine is

$$t_c = g * (e * t_{exec} + t_r)$$

where t_r is the time to recombine the populations obtained for the e parameter values at every generation. The computation time for the cooperative algorithm using e slave processors is

$$t_{P4} = t_{init} + g * (t_{exec} + (e+1) * t_{comm} + t_r)$$

where t_{init} is the time to initialize the slaves and t_{comm} is the communication time to transfer a population from one processor to another. The term $(e+1) * t_{comm}$ is the result of one communication from all the slaves to the master and e communications from the master to the slaves per generation.

We can neglect t_{init} because the second term is dominant in genetic algorithms. We can also neglect t_r in the expressions for t_c and t_{P4} since the recombination step just picks the best strings from the e sorted populations. Now,

$$\frac{t_c}{t_s} \approx 1 \quad \text{and} \quad \frac{t_{P4}}{t_c} \approx \frac{1}{e} \left(1 + \frac{(e+1)t_{comm}}{t_{exec}}\right)$$

This shows that, for problems where the communication to execution ratio is small, the cooperative algorithm using e slaves performs much better than the single node implementation.

IV. Task Assignment Heuristics

An assignment of tasks to processors can be formally described as a function $f : T \rightarrow P$ from the set of tasks T to the set of processors P. Given n tasks $T_1, T_2 ... T_n$ to be assigned to p processors $P_1, P_2 ... P_p$, it is possible to assign them in p^n ways. Each assignment has a cost associated with it. The cost of an assignment is a function of the total execution time and the total communication time. The problem is to find an optimal assignment, that is, an assignment which minimizes this cost function.

For a given assignment, the *total execution time* is defined as the maximum of the sum of the execution times of the tasks on each processor. Similarly, the *total communication time* is defined as the maximum of the sum of the communication costs for the tasks on each processor. The *total cost* of the assignment is defined as the sum of the total execution time and the total communication cost.

A. A Global Greedy Heuristic

Consider the following greedy heuristic for the task assignment problem : *Assign tasks one by one to processors, so as to minimize the total execution time computed so far.*

The heuristic gives different results based on the order in which the tasks are assigned. Here, we assign tasks according to the increasing order of the task number. Several modifications can be made to the heuristic to give different task assignments. Experiments indicate that for small examples this heuristic usually finds a solution very close to the optimal, but for larger problems, it does not work so well. However, for large problems, we can use this heuristic and its modifications to initialize the population for the genetic algorithm to a better starting population.

B. A Local Perturbation Heuristic

After executing the genetic algorithm, the final population set consists of good solutions which may cover a large

section of the solution space. To further improve the quality of the solution, a local perturbation heuristic is applied on each member of the population set. For a given task assignment, our perturbation heuristic moves tasks from a heavily loaded processor to a lightly loaded processor, provided it does not increase the cost of the assignment. The best of these solutions is the new suboptimal solution, if it is better than the suboptimal solution computed by the genetic algorithm.

V. Experimental Results

To be able to use the genetic algorithm approach, the solutions in the solution space have to be represented as strings. In our problem, for assigning n tasks to p processors, we encode each task assignment as a binary string of length $n \log p$. Thus, each of the p^n different assignments can be represented by one string. The correspondence between a string and the assignment it represents can be deduced as follows. The decimal representation of the most significant $\log p$ bits represent the processor to which task T_0 is assigned. The next significant $\log p$ bits represent the processor to which task T_1 is assigned and so on. Finally, the least significant $\log p$ bits represent the processor to which task T_n is assigned. Since there are $2^{n \log p}$, that is, p^n possible strings of length $n \log p$, there is a 1-1 correspondence between the encoded strings and the set of solutions to the task assignment problem. This encoding assumes that the number of processors is a power of 2. This is not a major limitation because most real-life problems satisfy this criterion. For problems where this is not the case, additional processors can be added to satisfy this criterion and the execution times of the tasks on these additional processors can be made very large so that any assignment of tasks to these extra processors results in exorbitant costs. The genetic algorithm would automatically weed out such assignments from the population.

For this encoding, the communication to execution ratio in the cooperative algorithm is small. In fact, the communication time to transfer a population from one node to another (t_{comm}) is of the order $\theta(n \log p)$, while the execution time for one generation on a given node (t_{exec}) is of the order $\theta(np)$. Consequently, based on the performance analysis in section III, the task assignment problem is suited for distributed implementation.

Experiments were performed on several task assignment problems, each with randomly generated task execution times for processors and communication costs between tasks such that the execution costs are much larger than the communication costs. The heuristic solution from Section IV.A is incorporated in the initial population for both, the standard and cooperative genetic algorithm. The results of the standard and cooperative genetic algorithm for the different problem sizes varying from small to medium and large are given in Figures 1, 2 and 3 respectively. In these figures, p_c and p_m are the cross-over and mutation probabilities respectively. The cooperative algorithm with k slaves uses the parameters from standard experiments 1 ... k. The minimum known cost of Figures 2 and 3 is found by running the cooperative algorithm for a large number of generations using 10 slaves. The lower bound in Figures 2 and 3 is obtained by using the algorithm described in [6].

Problem Size : 10 tasks, 4 processors
Number of Generations : 20
Greedy Heuristic Cost : 179
Optimal cost : 163

Population Size : 30

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	163	163	179	172	172	170

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	163	163	163	163	163	163	163

(a)

Population Size : 100

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	163	163	163	163	163	163

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	163	163	163	163	163	163	163

(b)

Fig. 1. Experimental results for a small-sized problem

The results indicate that the cooperative genetic algorithm finds the optimal solution for small problems, independent of the number of slaves and their probabilities. For medium sized problems, a near-optimal solution is found within a small number of generations. For large problems, there is no way of finding the optimal solution and we can judge our solutions only by the best solution known so far. The solution found by our algorithm is about 15-20% better than the heuristic solution. The solution **does not** improve even if we use the heuristics explained in section IV.B for post-processing. To substantiate the claim that the cooperative genetic algorithm indeed finds better solutions than the standard, we are performing extensive experiments on many large size problems. Distributed implementation is appropriate for such extensive experimentation.

VI. Conclusions and Future Work

In this paper, we presented a new genetic algorithm approach for optimization problems. The algorithm uses a set of slaves which cooperate to solve problems efficiently. Analysis shows that the distributed implementation is much faster than the sequential implementation if the ratio of the communication time to execution time of the algorithm for the given optimization problem is small. If the ratio is not small, we can decrease it by assigning the work of two or more slaves to a single processor. In the task assignment problem, this ratio is small and so, the distributed implementation is computationally efficient. Ex-

Problem Size	:	20 tasks, 8 processors
Number of Generations	:	20
Greedy Heuristic Cost	:	340
Minimum known cost	:	283
Lower bound	:	192

Population Size : 30

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	334	340	321	326	326	329

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	329	309	322	303	290	308	310

(a)

Population Size : 100

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	314	329	321	326	339	326

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	295	314	297	297	285	286	295

(b)

Fig. 2. Experimental results for a medium-sized problem

Problem Size	:	50 tasks, 16 processors
Number of Generations	:	20
Greedy Heuristic Cost	:	1303
Minimum known cost	:	1075
Lower bound	:	798

Population Size : 30

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	1243	1243	1303	1303	1303	1279

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	1235	1229	1229	1192	1217	1155	1209

(a)

Population Size : 100

Standard Genetic Algorithm Results						
	Expt 1	Expt 2	Expt 3	Expt 4	Expt 5	Avg.
p_c	0.95	0.85	0.75	0.90	0.80	
p_m	0.001	0.005	0.01	0.025	0.01	
Seed	0.543	0.124	0.912	0.302	0.712	
Cost	1229	1204	1243	1255	1243	1235

Cooperative Genetic Algorithm Results							
# Slaves	2	3	4	5	6	10	Avg.
Cost	1229	1192	1192	1184	1192	1184	1195

(b)

Fig. 3. Experimental results for a large-sized problem

periments indicate that the algorithm is insensitive to selection of cross-over and mutation probability. The cooperative genetic algorithm finds the optimal solution for small-sized task assignment problems. For medium and large-sized problems, the algorithm finds solutions which are about 15-20% better than the greedy heuristic solution. Further, post-processing using local perturbation heuristics does not improve the solution. The algorithm is flexible and easy to use (coding is minimal and experimentation is simple) which makes the approach applicable to other optimization problems in distributed computer systems. By changing the objective function and the encoding, the cooperative genetic algorithm can be used for different optimization problems. With slight modifications, the algorithm can be applied to the dynamic task assignment problem. It can also be modified to solve the task assignment problem with variable costs. By encoding the dependency constraints among tasks, this algorithm can be applied to dependent task assignment problem. From a given dependency graph, one can construct a number of variations of this graph and solve the multiple optimization problems to get the best assignment. For such multiple problems, the ratio of communication to computation can be made very small and so, the distributed implementation is very efficient.

Several modifications can be made to the algorithm to improve it further. A systematic selection of the initial population can greatly improve the performance. Different strategies other than Master-Slave strategy can be used by the slaves to cooperate with other slaves. Instead of globally selecting the best population, a fraction of the population can be selected and passed to the slaves. We believe that incorporating these different strategies can further enhance the proposed algorithm.

References

[1] D.E.Goldberg, "*Genetic Algorithms in Search, Optimization and Machine Learning*," Addison-Wesley, Mass., 1989.

[2] V.S.Gordon and D.Whitley, "*Serial and Parallel Genetic Algorithms as Function Optimizers*," Proc. of the Fifth Intl. Conf. on Genetic Algorithms, Morgan Kaufmann Publishers, pp. 177-183, 1993.

[3] V.M.Lo, "*Heuristic Algorithms for Task Assignment in Distributed Systems*," IEEE Trans. on Computers, vol. 37, pp. 1384-1397, Nov. 1988.

[4] C.S.Pettey *et al*, "*A Parallel Genetic Algorithm*," Proc. of the Second Intl. Conf. for Genetic Algorithms, Morgan Kaufmann Publishers, pp. 155-161, 1987.

[5] H.S.Stone, "*Multiprocessor Scheduling with the Aid of Network Flow Algorithms*," IEEE Trans. on Software Engg., vol. 8, pp. 85-93, Jan. 1977.

[6] R.Venkateswaran, Z.Obradović and C.S.Raghavendra, "*Cooperative Genetic Algorithm for Optimization Problems in Distributed Computer Systems*," Technical Report TR-EECS-93-018, School of EECS, Washington State University, 1993.