DYNAMIC TASK SCHEDULING USING PARALLEL GENETIC ALGORITHMS FOR HETEROGENEOUS DISTRIBUTED COMPUTING

R.Nedunchelian, K.Koushik, N.Meiyappan, V.Raghu
Department of Computer Science and Engineering,
Sri Venkateswara College Of Engineering,
Pennalur, Sriperumbudhur 602105, India.

Abstract - *A parallel genetic algorithm has been developed to dynamically schedule heterogeneous tasks to heterogeneous processors in a distributed environment. The scheduling problem is known to be NP complete. Genetic algorithms, a meta-heuristic search technique, have been used successfully in this field. The proposed algorithm uses multiple processors with centralized control for scheduling. Tasks are taken as batches and are scheduled to minimize the execution time and balance the loads of the processors. According to our experimental results, the proposed parallel genetic algorithm (PPGA) considerably decreases the scheduling time without adversely affecting the maxspan of the resulting schedules.*

Keywords: *Genetic algorithms, Parallel Processing, Scheduling*

1 Introduction

Distributed computing is a promising approach to meet the ever-increasing computational requirements [3]. Scheduling is the most important issue in the distributed system because the effectiveness of the scheduling directly corresponds to the parallelization obtained. With inappropriate scheduling mechanisms can fail to exploit the true potential of the distributed system. The scheduler has the dual responsibility of minimizing the execution time of the resulting schedule and balancing the load among the processors. Even when the target processors are fully connected and when no communication delay is considered, scheduling is NP complete. This problem is known as strong NP-hard intractable optimization problem when it assumes arbitrary number of processors and arbitrary task processing time [4]. Scheduling is usually handled by heuristic methods which provide reasonable solutions of the problem.

Multiprocessor scheduling methods can be divided into list heuristics and Meta heuristics. In list heuristics [5], the tasks are maintained in a priority queue in decreasing order of priority. When a free processor is available, the task at the front of the queue is assigned to the free processor. Most list heuristics are not efficient for all situations.

Genetic algorithms (GAs) are a Meta heuristic searching techniques which mimics the principles of evolution and natural genetics. These are a guided random search which scans through the entire sample space and therefore provide reasonable solutions in all situations. Many researchers have investigated the use of GAs to schedule tasks in homogeneous [8, 9] and heterogeneous [3, 6, 7] multi-processor systems with notable success. Nevertheless, the main draw back of using GAs is that too much time is used for doing scheduling. Hence, we propose a parallelized genetic algorithm to speed up the scheduling.

Fitness evaluation is the most CPU intensive tasks and can be very time consuming and therefore become a bottleneck in the scheduler's performance. We propose to use, synchronous master slave parallelization by which fitness evaluation is done in parallel. This is equivalent to a sequential GA as observed by [2].

The rest of the paper is organized as follows: Section 2 discusses Parallel Genetic Algorithm. Section 3 elaborates on scheduling and parallel genetic algorithm. The proposed parallel genetic algorithm is discussed in Section 4. Section 5 discusses experimental results and Section 6 concludes with future research directions.

2 Parallel Genetic Algorithms (PGAs)

Sequential Genetic algorithms have been applied successfully in many different domains. However, there exist some problems in their utilization [2] which can be addressed by PGA. Some GAs need to have very large populations which make them impossible to run efficiently on a single machine. Some GAs get trapped in a suboptimal region of search space. The most common problem faced by a sequential GA is the CPU time. Computation time of more than 1 CPU year has been reported in the literature [10].

In most parallel algorithms, the basic idea behind the algorithm is to divide the task into subtasks and use different processors to execute each subtask. This divide-and-conquer approach can be applied to GAs in many different ways, and the literature contains many examples of successful parallel implementations. A complete classification of PGAs is given in [11].

In master slave parallelization, there is a single panmictic population, but the evaluation of the fitness function is distributed among several processors. Since, selection and crossover consider the entire population they are also called Global parallel GAs. The master always waits for the slowest slave processor before starting the next generation. Asynchronous master slave parallelization is an extension of synchronous master slave parallelization in which the master does not wait for the slowest processor. The selection operator gets affected because of the change and the resulting GA dynamics are difficult to analyze [2].One of the advantages of synchronous master slave parallelization is that the underlying GA characteristics are not affected. This model does not assume anything about the underlying architecture and therefore is most suited in distributed environment.

Fine-grained parallel GAs are most suited for massively parallel computers and consists of one spatially-structured population. Selection and mating are restricted to a small neighborhood, but neighborhoods overlap permitting some interaction among all the individuals. The selection and mating operations in these PGAs are different from those of a sequential GA.

PGAs with multipopulation are also possible. The important characteristics of multi-deme PGAs, also called as coarse grained GAs, are a few relatively large subpopulations and migration. They introduce fundamental changes in the operations of GA and are the most difficult to understand. Since the deme sizes are small, these GAs converge faster but the quality of the resulting solutions might be poorer.

Hierarchical PGAs combine the characteristics of multi-deme PGAs with fine-grained or master slave PGAs [12]. At the higher level, they are multideme algorithms with single population parallel GAs at the lower level.

3 Parallel Genetic Algorithms and Scheduling

Artificial intelligence techniques have been successfully applied to task scheduling [3, 6, 7, 8, 9, 13, 14]. Good results have resulted from the use of GAs in task scheduling algorithms [3, 6, 7, 8, 9].Parallel Genetic algorithms have been applied to homogeneous multiprocessor scheduling [15, 16]. Abramson, Mills, and Perkins [17] designed a train time-table generation algorithm using PGA in a distributed environment.

Genetic algorithms is a searching strategy in which an initial number of 'guesses' is made to get an optimal solution (the initial population). Each guess is evaluated and assigned a 'goodness' value (the fitness function). Those guesses with good values are selected and new guesses are made by combining the existing guesses in a particular fashion (crossover). The guesses are evolved to the next generation on a survival of the fittest basis (selecting), thereby the 'good' guesses are forwarded to the next generation and the 'bad' guesses are not. Random mutation is done to prevent the GA from getting struck in a local maximum.

GAs have been used in static scheduling [8, 13], where the schedule is generated before run-time and dynamic scheduling [3, 9, 14], where the schedule is generated at run-time based on the run-time characteristics which are not possibly known beforehand. Dynamic scheduling is more applicable to a real world environment.

Current Dynamic GA schedulers show near optimum solutions in simulations [3, 9, 14]. But, all of the existing approaches using sequential GAs take a long time to converge [18]. We propose to parallelize the GA so that the scheduling time is considerably reduced.

There are many ways to perform the parallelization as discussed in section 2. Synchronous master slave parallelization suits our proposed algorithm well because,

1. The population of the GA is fixed as 20, to improve the speed. These types of GAs called micro GAs are widely used for scheduling [3, 9] and have been shown to be successful. For micro GAs, multi-deme parallelization is not applicable.

2. The resulting PGA has the same characteristics of the underlying GA but for the increase in speed.

3. This type of PGA does not make any assumptions regarding the underlying processors, which makes them ideal for distributed environment as observed by [17].

Each chromosome can be processed (evaluated) individually without considering the rest of the population. Mathematical analysis of master slave parallelization of GA has been done in [20].

4 The Proposed Parallel Genetic algorithm

The proposed parallel genetic algorithm involves a master scheduler, which has the processor lists and the task queue. The processors of the distributed system are heterogeneous. The available network resources between processors in the distributed system can vary over time. The availability of each processor can vary over time (processors are not dedicated can may have other tasks that partially use their resources). Tasks are indivisible, independent of all other tasks, arrive randomly, and can be processed by any processor in the distributed system. The master scheduler runs a sequential GA in which the fitness function evaluation alone is done by slave processors.

When tasks arrive they are placed in the unscheduled task queue. They tasks are taken in batches and scheduled. Batch schedulers are shown to have higher performance than immediate schedulers in [3]. When any processor is idle, the processor asks for a task to perform and the task scheduled for that processor (if any) is given to that processor. All the task data are maintained only in the

Synchronous master slave parallelization is used to evaluate the fitness function alone in a distributed fashion. These are the steps in parallelization,

1) A **master** scheduler which is the processor in charge of scheduling **chooses the slaves**. This choice is based upon the communication overhead involved and the computational potential of the slave processor. In other words, a processor which is too slow or too remote will not be used as a slave. The number of slaves selected is almost 20, the population size.

2) The **master has the population** of chromosomes for which the fitness function is to be evaluated.

3) Each **slave evaluates the fitness** of a fraction (F_i) of the population in the master scheduler and returns the value.

    a. $\ell_j=P_j/L_j$ where L_j denotes previously assigned load in MFLOPS and P_j represents the current processing power of the slave processor j.

    b. $T=\sum \ell_j$,for all chosen slave processors.

    c. $Fi= (\ell_i/T)*100$ for all i, chosen slave processors.

    d. The fraction F_i of the total population is assigned to the ith slave processor.

    This fraction calculation ensures that a relatively slow slave processor does not become a bottleneck for the entire scheduling task. This distribution of task ensures that all slave processors are utilized efficiently. The slave processors need not be dedicated to the scheduling alone as their previous loads are also taken into account.

4) After partitioning the population into fractions, the slave processors receive their fraction of chromosomes one at a time, evaluate and return the result to the master. This approach is efficient because, it limits the data transfer. In a distributed environment, the slaves may leave the system at any time. So the chromosomes are transferred only just before the calculation is to be performed.

The above algorithm has exactly the same properties as a sequential GA, but executes faster. The Pseudo code for the underlying sequential genetic algorithm is shown in figure 1.

```
Encode the chromosome.

Initialize the population (randomize)

do {

Stochastic sampling with partial replacement selection

Cycle crossover

Mutation: randomize and rebalancing

}while(stopping conditions not met)

Return best individual
```

Figure 1. Pseudo code for genetic algorithm

This genetic algorithm is an extension of the genetic algorithm proposed by Page [3]. Each step in the GA is explained in detail.

4.1 Encoding the Chromosome

Each task in the batch has a unique identification number. The total number of tasks in the batch is N and total number of processors in M. The unique identification task number of all the tasks allocated to a processor is encoded in the chromosome with -1 being used to delimit the different processor queues.

2	3	-1	1	-1	5	4

Figure 2: A sample chromosome

The sample chromosome in figure 2 has a batch size of 5 tasks with 3 processors. This chromosome represents the following task allocation.

Processor s	Tasks
1	2,3
2	1
3	5,4

Table 1: Task Allocation

4.2 Fitness Function

A fitness function computes a single positive integer to represent how good the schedule is. We use relative error to generate the fitness values. The fitness of each individual in the population is calculated using synchronous master slave parallelization,in other words, by this function itself is computed by the slave processors. Previously assigned, but unprocessed, load for each processor is considered by calculating the finishing time of a processor *j*. δ*i* = (A*j/Pj*), where A*j* denotes the previously assigned load, measured in MFLOPs, and *Pj* is the current processing power in Mflop/s of processor *j*. The current load of each processor is calculated,

$$L_j = [(\sum_{i=1}^{n} t_i)/p_j] + \delta_j + \sum_{i=1}^{n} \Gamma_c(i,j)$$

where *ti* is the processing requirements of task *i* in the batch (in MFLOPs) and *n* is the total number of tasks assigned to the processor j. $\acute{\Gamma}_C$(i,j) is the communication time for the ith task in the jth processor.

The mean value of Lj for all the processors is given as

$$\mu = \sum_{j=1}^{M} L_j / M$$

The relative load imbalance error of individual *i* is given as

$$E_i = \sqrt{\sum_{j=1}^{M} \left|\mu - L_j\right|^2}$$

This Error value denotes how unbalanced the schedule is. For instance, if all the tasks are assigned to only one processor while the others are idle, this error value will be very large. Minimizing the error value ensures that schedules which utilize more processors at the same time (increase parallelization) will be consider fitter schedules.

The fitness value of individual *i* is $F_i = \left(\left(\max - \max span_i\right) + 0.001 \times \max\right) / E_i$

where,

max is the maximum of all finishing times.

maxspan$_i$ is the largest task completion time among all the processors.

The maxspan is nothing but the execution time of the resulting schedule which is to be reduced. This value is reduced from the maximum possible execution time.This value could become zero at some cases. To avoid this 1/1000th of the max time is added to the numerator. A larger value of F indicates a better or fitter schedule.

4.3 Communication Time

The communication time $\acute{\Gamma}_C$(i,j) is calculated as follows :

Time taken in seconds for execution of ith task on the jth processor

T (i, j) = t (i)/P (j),

Communication cost in seconds/bits for ith task on the jth processor

C (i, j) = [R (i, j) – T (i, j)] / S (i),

where R (i, j) is the round trip time for the ith task and S(i) is the size in bits of the ith task.

Predicted Communication cost in seconds/bits for ith task on the jth processor

τ_C (i, j) = [C(i-1,j) + τ_C (i-1,j) * (i-2)] / (i-1),

i>=2 and **τ_C (0,j) = 0.**

Factor **ά = [τ_C (i, j) - τ_C (i-1,j)] / τ_C (i-1,j)**

Therefore the Communication time in seconds is

$\acute{\Gamma}_C$(i,j) = ά * T(i-1,j) + (1- ά) *[τ_C (i,j)*S(i)]

4.4 Stochastic Sampling with partial replacement selection

In a standard weighted roulette wheel selection algorithm, the selection is totally based on the fitness function. The chromosomes are assigned slots in the roulette wheel based on their relative fitness function values. The roulette wheel is spun N times to select N chromosomes. Stochastic sampling with partial replacement selection is a simple extension of the roulette wheel selection in which the sector assigned for a particular chromosome is reduced if the chosen chromosome has a fitness value less than the average fitness value. This increases the survival rate of the fitter solutions when compared to standard roulette wheel selection.

4.5 Cycle Crossover

Cycle crossover [20] is a crossover operator which applies to permutation encoding schemes which need to preserve both the allele value and the allele order of the gene. This operator ensures that, the two offspring will have their gene values taken from the same value and position of either of their parents. This ensures that the properties of the parents are carried over to the children there by making fitter children possible.

Parents

2	3	-1	1	-1	5	4
4	-1	2	5	-1	1	3

Children

2	3	-1	5	-1	1	4
4	-1	2	1	-1	5	3

The above example uses the randomized locus for start of the start of the cycle as the first position. The cycle formed is 2-4-3-2.The 5 and 1 of the parents, which are not part of the cycle, are swapped to get the resulting children.

4.6 Swapping Mutation

An individual in the population is randomly selected and any two tasks in that chromosome are randomly selected and swapped. This approach ensures that all the solutions in the search space are more thoroughly examined.

4.7 Stopping Conditions

A maximum of 1000 evolutions are used. The fitness values of the chromosomes obtained after 1000 evolutions did not show considerable improvement. The GA will also stop evolving if one of the processors becomes idle, in which case it will return the best schedule found so far.

5 Experimental Results

5.1 Setup

We construct a simulation and evaluation environment to evaluate a PGA based Dynamic Scheduling Algorithm for a heterogeneous distributed system. We simulated our algorithm on java platform with one node acting as a master scheduler and one to six slave processors, which are chosen by the master scheduler based on proximity and processor efficiency. The fitness function evaluation class is installed on all the slave processors and PPGA algorithm is installed in the master scheduler. The input consists of tasks whose sizes are uniformly distributed between 10 and 1000 MFLOPS.

5.2 Tests

We compare our scheduler with two other GA based schedulers, Page [3], and Zomaya [9] and evaluate the results using two different metrics, namely, maxspan and time required to calculate the best schedule. Maxspan is the total execution time of a schedule. Time required to calculate the best schedule is required here as a result of scheduler efficiency consideration. Mathematical Analysis of the master slave parallelization of GA has been done in Cantu-Paz [20].

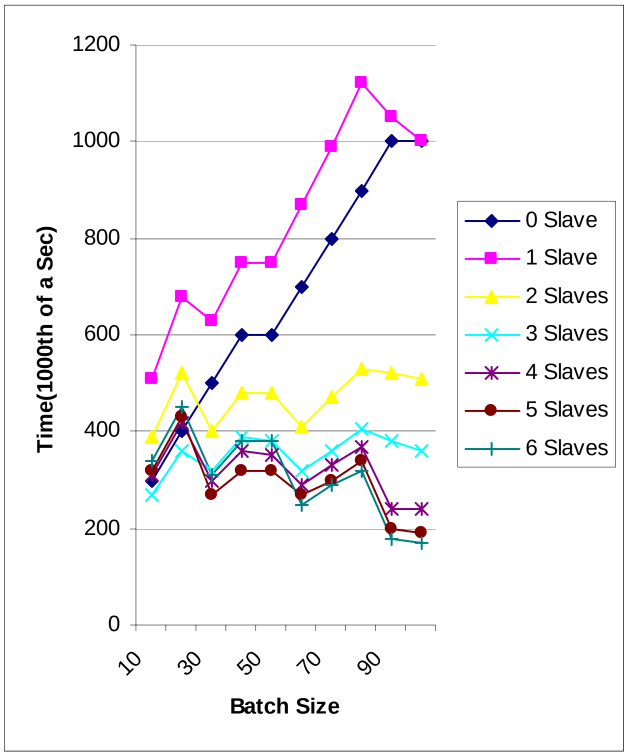

Figure 3: Comparison of Scheduling Times

The average execution time of the proposed algorithm was 350ms as opposed to 680 and 830ms for algorithms proposed in [9] and [3] respectively. The average maxspan of the best schedule from the proposed algorithm was 4200ms, similar as in [3] and considerably lesser than [9] which is 6200ms. In figure 3 we illustrate the decrease in time for finding the optimal schedule by increasing the slave processors from 1 to 6 for varying batch sizes.

6 Conclusion

A scheduling algorithm has been developed to schedule heterogeneous tasks onto heterogeneous processors on a distributed environment. It provides near-optimal schedules and adapts to varying processing resources and communication costs. The algorithm uses a dedicated master scheduler for centralized scheduling. It uses slave processors (which are not dedicated to scheduling) to parallelize the GA and thereby speed up the result. Genetic Algorithms are powerful but usually suffer from longer scheduling time which is reduced in our algorithm due to the parallelization of the fitness evaluation, the most CPU intensive task.

According to our simulation results, the proposed algorithm not only obtains similar performance as the original genetic algorithm, but also spends less time doing the scheduling. This feature also makes the proposed algorithm to be more scalable and extends its practicability.

The proposed algorithm uses a straightforward encoding scheme and generates a randomized initial population. The fitness function uses the maxspan, the balance of load among the processors and the communication costs while evaluating the schedules. Stochastic sampling with partial replacement selection, an extension of the roulette wheel selection, is used to increase the possibility of survival of the fitter solutions. Cycle cross over preserves the characteristics of the parent chromosomes in the children there by leading to exploration of search space. Random swapping is done to prevent the GA to get struck in a local maximum.

4 References

[1] GOLDBERG D. E., *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

[2] M Nowostawski, R Poli, Parallel Genetic Algorithm Taxonomy, - *3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems* 1999.

[3] AJ Page, TJ Naughton Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing, - *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05).*

[4] M Golub, S Kasapovic Scheduling Multiprocessor Tasks with Genetic Algorithms, - *Applied Informatics-Proceedings*- 2002.

[5] Tao Yang and Apostolos Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, pp. 951-967, Sep. 1994.

[6] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, November 1999.

[7] A. Y. Zomaya, M. Clements, and S. Olariu. A framework for reinforcement-based scheduling in parallel processor systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):249–260, March 1998.

[8] E. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, February 1994.

[9] A. Y. Zomaya and Y.-H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899–911, September 2001.

[10] Sean Luke, "Genetic programming produced competitive soccer softbot teams for robocup97," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, Eds., University of Wisconsin, Madison, Wisconsin,USA, 22-25 July 1998, pp. 214–222.

[11] E Cantu-Paz ,A survey of parallel genetic algorithms, *Calculateurs Paralleles, Reseaux et Systems Repartis,* 1998

[12] Mariusz Nowostawski, "Parallel genetic algorithms in geometry atomic cluster optimisation and other applications," M.S. thesis, School of Computer Science,The University of Birmingham, UK, September 1998,http://studentweb.cs.bham..ac.uk/˜mxn/gzipped/mpga-v0.1.ps.gz.

[13] I. Ahmad, Y.-K. Kwok, I. Ahmad, and M. Dhodhi. Scheduling parallel programs using genetic algorithms. In A. Y. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, chapter 9, pages 231–254. John Wiley and Sons, New York, USA, 2001.

[14] A. Y. Zomaya, C. Ward, and B. Macey. Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795–812, August 1999.

[15] Moore, M., An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster, IPDPS, 2003.

[16] Moore, M., *Parallel Genetic Algorithms to Find Near Optimal Schedules for Tasks on Multiprocessor Architectures*, Communicating Process Architectures, IOS Press, 2001, pgs, 27-36.

[17] Abramson D., Mills G., Perkins S., Parallelisation of a genetic algorithm for the computation of efficient train schedules *Proceedings of the 1993 Parallel Computing and Transputers Conference*, p. 139–149, 1993.

[18] Yi-Hsuan Lee and Cheng Chen, A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems, Taiwan: National Chiao Tung University, 2003. http://parallel.iis.sinica.edu.tw/cthpc2003/papers/CTHPC2003-18.pdf

[19] I.M. Oliver, D.J. Smith and J.R.C. Holland (1987), Study of Permutation Crossover Operators on the TSP, Genetic Algorithms and Their Applications: Proceedings of the Second International Conference, 224-230.

[20] Designing efficient master-slave parallel genetic algorithms E Cantu-Paz, Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25, 1998