Enhanced Simulated Annealing Techniques for Multiprocessor Scheduling

G.E. Nasr[1], A. Harb[1] and G. Meghabghab[2]
[1]Department of Electrical and Computer Engineering
Lebanese American University, Byblos, Lebanon
e-mail : genasr@lau.edu.lb
[2]Department of Mathematics and Computer Science
Valdosta State University, Valdosta, GA 31698
e-mail :gmeghab@valdosta.edu

Abstract

The problem of multiprocessor scheduling can be stated as scheduling a general task graph on a multiprocessor system such that a set of performance criteria will be optimized. This study investigates the use of near optimal scheduling strategies in multiprocessor scheduling problem. The multiprocessor scheduling problem is modeled and simulated using five different simulated annealing algorithms and a genetic algorithm. In this paper, the comparison of the simulation results of the simulated annealing algorithm, the modified versions of simulated annealing algorithms, and the genetic algorithm is presented. In addition, results of sensitivity analysis on the simulated annealing algorithm, the modified simulated annealing algorithms, and the genetic algorithm are included.

Introduction

The multiprocessor scheduling problem can be described as the problem of fairly distributing a workload on a distributed computer system consisting of multiple processors with distributed local memories.

Many researchers have worked on the problem of multiprocessor scheduling and various algorithms have been proposed. Proposed solution methods in the literature fall under two major categories: Heuristics and physical optimization algorithms. Heuristic procedures are mainly geometry based methods whereas physical optimization algorithms are derived from the natural sciences.

In the case of heuristics, Hellstorn and Kanal (1992) proposed a solution for multiprocessor scheduling problem using an asymmetric mean-field neural network model.

In the case of physical optimization, Hou, Ansari and Ren (1993) proposed a genetic algorithm based solution and Driessche and Piessens (1992) developed parallel versions of the genetic algorithm. Hwang and Xu (1993) proposed a simulated annealing algorithm for solving the multiprocessor scheduling problem.

In this study, the behavior of simulated annealing and genetic algorithms on the multiprocessor scheduling problem are examined under different situations. In addition, the modified-uniform simulated annealing algorithm (MSA), the enhanced simulated annealing algorithm (Enh_SA) (Loganantharaj, 1997) the multi-thread simulated annealing algorithm (Mul_SA), and multi-thread modified-uniform simulated annealing algorithm (Mul_MSA) are investigated. Simulation results are presented and compared.

The Multiprocessor Scheduling Problem

The general problem of multiprocessor scheduling can be stated as scheduling a set of computational tasks onto a multiprocessor system so that a set of performance criteria will be optimized. The goal (or program) assigned to the multicomputer is divided into several program modules M_i. The interrelationship among M_i can be specified by a program graph after compilation. Each program module, M_i for i=1,2,…,m, corresponds to one node in the graph. The edge connections in the program graph correspond to communication paths among program modules. The module weight, W_i, reflects the code segment length and size of the data set used in M_i, giving an approximated estimate of the CPU cycles, needed to execute M_i. The edge weight, e_{ij}, reflects the expected communication cost between the two program modules (Nasr and Moujabber, 1997).

The assumptions made in this work are the following:

- The Multiprocessor is a distributed-memory message-passing multicomputer whose processors are connected by a static point-to-point interconnection network (Hwang, 1993).
- The Routing used is a wormhole routing (Hwang, 1993).
- The computation model used is a loosely synchronous computation model in which processors perform compute-communication cycles in SPMD (Single Program, Multiple Data) scheme (Fox et al., 1988).
- The edges of the graph are bi-directional between any pair of modules M_i and M_j. i.e. $e_{ij} = e_{ji}$.

- Bi-directional links are assumed between all pairs of computer nodes in a multicomputer.
- The distance between any two computer nodes is defined as the number of hops between them.

The Objective Function

The job of the objective function is to be a guide for the algorithm to achieve optimization (minimizing system execution time in the case of load balancing). In this study a system oriented objective function (OF_{so}) is used.
This cost function is defined as $\boldsymbol{OF_{so} = E_{imb} + E_{com}}$, where E_{imb} is the system load imbalance and E_{com} is the communication cost (Nasr and Moujabber, 1997).
For a given load distribution, an imbalance vector is defined $\Delta=(\delta_1,\delta_2,\delta_3,\ldots,\delta_n)$ over n computer nodes, where the load offset δ_i for the node C_i is defined as $\delta_i = |l_i - L|$ and L is the load average among n computer nodes (Total load/n). The system load imbalance is defined by:
$E_{imb} = \sum\delta_i$. The communication cost is the sum of the matrix expressed as $E_{com} = 1/2\ \Sigma[\Sigma C_{ij}]$ & $C_{ij} = d_{ij}\ \Sigma[\Sigma e_{xy}]$, where d_{ij} is the distance between the nodes and e_{xy} is the edge weight between all program modules M_x in C_i and M_y in C_j.

Scheduling Methods

Simulated Annealing Algorithm (SA)

SA is a physical optimization technique which accepts a bad criteria in order to reach a global minimum (or optimal solution).
SA starts with a randomly generated initial solution. In each iteration, the solution is randomly perturbed. If such a perturbation results in negative or zero change in the system energy, then it is accepted. To prevent premature convergence when trapped in a bad optimum, the simulated annealing accepts positive changes in energy with a certain temperature dependent probability.
In order to apply the simulated annealing algorithm for solving the multiprocessor scheduling problem the following steps are followed:

1. Give an initial value for the temperature T and assign a constant value for K (cooling schedule).
2. Distribute the jobs (tasks) on the processors randomly.
3. Evaluate the system weight, for this initial random generation, using the objective function OFso.
4. Select two processors a and b, randomly (the selected processes a and b must be different). Then select (also randomly) two tasks, one from a and another one from b.
5. Swap the selected tasks between the two processors a and b.
6. Evaluate again the system weight, for this perturbation, using the objective function OF_{so}.
7. If the new system weight is smaller than the previous one, accept this new state and use it as a new starting point for the next perturbation.
8. If not, generate a random number, RND, between 0 and 1. If
   $RND < e^{[[\text{Previous system weight - new system weight}]/T]}$,
   accept the situation and use it as a new starting point for the next perturbation. If not, reject and keep the previous one as the starting point for the next perturbation.
9. Repeat the steps starting from part 4 for n successive rejections (in this study n = 3).
10. Change the temperature T as follows: T = T * K.
11. Repeat the steps starting from part 4 for a new temperature until the temperature becomes less or equal to 1.

The value of the cooling factor is problem-dependent and has to be determined experimentally.

Enhanced Simulated Annealing Algorithm (Enh_SA)

It is clear that the SA technique achieves better solution than a local search method by incorporating a chance for accepting higher cost, or letting the algorithm to jump some short distance subject to some pre-set parameters (k, p, and temperature). The enhanced annealing technique draws parallel from selecting different solid samples at different energy levels, and heating each sample to a higher temperature and then cooling it down till it reaches the lowest energy equilibrium. The cooled down sample with the lowest energy level provides the solution (Loganantharaj, 1997).
In order to apply the enhanced simulated annealing algorithm for solving the multiprocessor scheduling problem the following steps are followed:

1. Distribute the jobs (tasks) on the processors randomly and evaluate the system weight (using the OF_{so}) for 100 times.
2. Find the mean of 100 system weights, which are found in part 1.
3. Select a threshold percentage such that any case having a system weight greater than 20% above the mean is selected as a seed to perform multi-thread annealing.
4. Collect enough number of seeds.
5. Apply the simulated annealing algorithm as discussed before for every seed.
6. Select the distribution that has the minimum system weight.

Multi-Thread Simulated Annealing Algorithm (Mul_SA)

The enhanced simulated annealing algorithm is reexamined under new seed selection criteria. The following modifications are adopted:

1. select the seeds randomly.
2. Reapply on the selected seed the simulated annealing algorithm.
3. Select the distribution that has the smallest system weight as a solution.

Modified-Uniform Simulated Annealing Algorithm (MSA)

As a variation on the simulated annealing algorithm, the Boltzman function in the simulated annealing algorithm is replaced by a uniform function. The modifications on the simulated annealing algorithm are done by replacing the Boltzman function in step 8 of SA by a uniform function and deleting steps 10 and 11 of the same algorithm.

Multi-Thread Modified-Uniform Simulated Annealing Algorithm (Mul_MSA)

This algorithm draws parallel from selecting different samples. So, instead of taking one seed and applying on it the modified-uniform simulated annealing algorithm (MSA), many seeds are selected for the application of MSA.

Genetic Algorithm (GA)

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics (Holland, 1975).

A genetic algorithm consists of the following steps:

1. Initialization: an initial population of the search nodes is randomly generated.
2. Evaluation of the fitness function: the fitness value of each node is calculated according to the fitness function or objective function.
3. Genetic operations: new search nodes are generated randomly by examining the fitness value of the search nodes and applying the genetic operators to the search nodes.

Repeat steps 2 and 3 until the algorithm converges.

GA is composed of three operators:

1. Reproduction: Reproduction is a process in which individual strings are copied according to their fitness function values. The fitness function is evaluated for all populations. The population which has the minimum fitness value is replaced by the one which has the maximum value.
2. Crossover: crossover may proceed in three steps. First, two random strings are chosen for the crossover procedure. Second, a random number k is generated with a value between 1 and the string length minus 1, $[1,l\text{-}1]$. In our case, k is generated with a value between the lowest and the highest task weight. Finally, two new strings are created by swapping all characters between position $k + 1$ and l inclusively.
3. Mutation: first, select randomly two processors a and b from one population. Second, select randomly 2 tasks, one from processor a and the other from processor b. Finally, swap these two selected tasks between the processors a and b.

The number of mutations depends on the mutation probability ρ defined by:

number of mutations = ρ * number of processor * number of tasks.

Simulation Results

Various tests are performed to evaluate and compare the performance of the above mentioned algorithms. The tests were conducted on 2 task graphs. These graphs were randomly generated. The first one was a graph of 32 jobs and the second one was a 64-jobs graph. Algorithm performances on the two graphs were studied using six different topologies. These topologies were 16-nodes Hypercube, 8-nodes Hypercube, 16-nodes Ring and 8-nodes Ring.

Algorithm performances are studied in terms of two measures. The first measure is the solution quality (Optimizing the system objective function OF_{so}) and the second measure is the execution time measured in unit of time. The results for each test are quoted in terms of the system weight, the time and the percentage enhancement with respect to the initial random distribution. In addition, the comparison between the means of the data using hypothesis testing and confidence interval analysis is performed (Douglas, 1996). All simulations are performed using *Mathematica* software. The parameters used in the simulations are listed below.

- The cooling factor $K = 0.75$ and the initial temperature T = 50 in the simulation of SA.
- The percentage range $r = 0.03$ in the simulation of MSA.
- A population size of 16 and a mutation probability ρ = 0.0005 in the simulation of GA.
- For the Enh_SA, a seed is accepted for the reapplication of SA if it falls above 20% of the system weight mean.
- For Mul_SA, 10 seeds are randomly selected for the reapplication of SA.
- For Mul_MSA, 10 seeds are randomly selected for the reapplication of MSA.

The following graphs show the simulation results of the mean of 20 multiprocessor scheduling tests using the above mentioned algorithms to distribute 64 jobs on 8 processors.

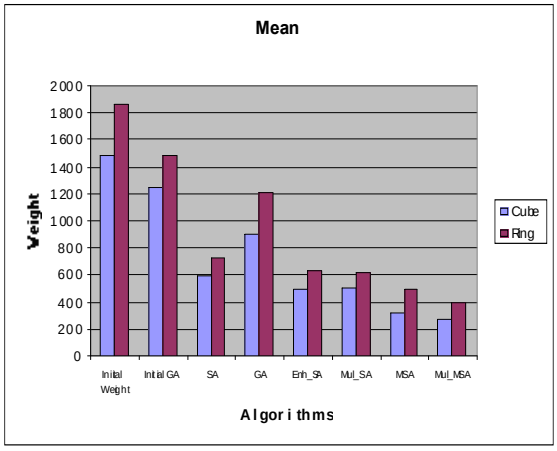

Fig. 1. The mean of the system weight

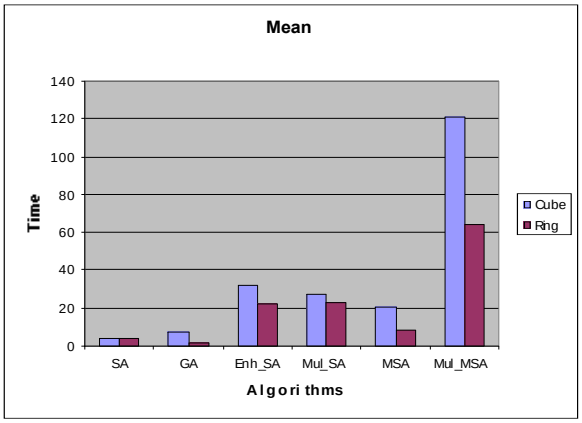

Fig. 2. The execution time mean

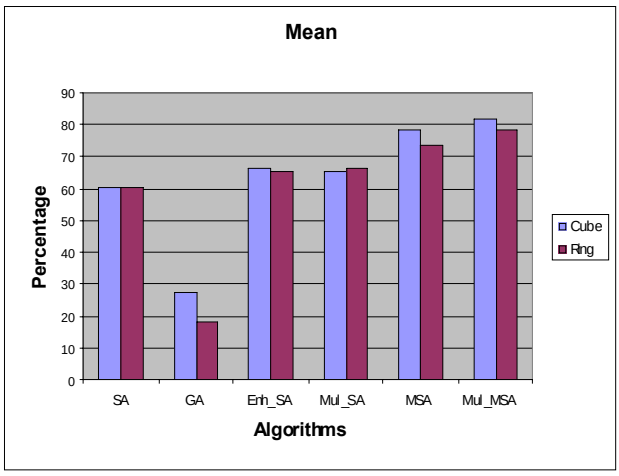

Fig. 3. Percentage enhancement from the initial system weight

The comparison of simulation results provides the following observations.

- The system weight enhancement (for all kinds of topologies) using all the discussed algorithms follows the below equation under hypothesis testing.

$\mu[\text{Enhancement(Mul\_MSA)}] > \mu[\text{Enhancement (MSA)}] \geq \mu[\text{Enhancement(Mul\_SA)}] = \mu[\text{Enhancement(Enh\_SA)}] > \mu[\text{Enhancement(SA)}] > \mu[\text{Enhancement(GA)}]$

- The same enhancement is obtained for Enh_SA and Mul_SA.
- The improvement, in minimizing the objective function OF_{so}, using MSA, SA, and GA follows the following equation:

  Improvement of MSA > Improvement of SA > Improvement of GA.

On the other hand, the execution time of MSA, SA, and GA follows the following equation:

Execution time of MSA > Execution time of SA > Execution time of GA.

In addition, sensitivity analysis for SA with respect to its cooling factor, MSA with respect to its percentage range, and GA with respect to its population size and mutation probability are considered. The sensitivity analysis revealed the following:

- SA is sensitive to the variation of the cooling factor. When the cooling factor increases, the enhancement from the initial system weight and the execution time also increase.
- MSA is sensitive to the variation of the percentage range. In particular, when the percentage range increases the enhancement from the initial system weight decreases and the execution time increases.
- A moderate mutation probability (0.0005) gives a better improvement than the lowest (0.00005) and the highest (0.005) mutation probabilities. In addition, GA appears not to be sensitive to the population size.

Conclusion

In this paper, the behavior of simulated annealing and genetic algorithm on the multiprocessor scheduling problem are examined under different situations. In addition, the modified-uniform simulated annealing algorithm (MSA), the enhanced simulated annealing algorithm (Enh_SA), the multi-thread simulated annealing

algorithm (Mul_SA), and multi-thread modified-uniform simulated annealing algorithm (Mul_MSA) are investigated. The simulation results presented in this paper showed that Mul_MSA provides more system weight enhancement than the other methods and that the same enhancement can be obtained when using either Enh_SA or Mul_SA. In addition, the enhancement from the initial system weight and the execution time were found to increase with the increase of the cooling factor for the simulated annealing algorithm. However, the enhancement from the initial system weight was found to decrease with the increase of the percentage range in the case of MSA. It should also be noted that no conclusive data was obtained to formulate a comparative statement on the execution time of the different algorithms.

References

Douglas, C., 1996, *Design and Analysis of Experiments*, John Wiley & Sons, Inc.

Fox, G. C., Johnson ,M., Lyzenga, G., Otto, S., Slamon, J., Walker, D., 1988, *Solving Problems on Concurrent Processors, Englewood Cliffs*, N, J, Prentice-Hall.

Helstron, B. and Kanal, L., 1992 Asymmetric Mean Field Neural Networks for Multiprocessor Scheduling, Neural Networks, 5, 671-686.

Holland, J., 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan press.

Hou, E., Ansari, N. and Hong, R., 1993 A Genetic Algorithm for Multiprocessor Scheduling, IEEE Transactions on parallel and Distributed Systems, 4 (1): 121-134.

Hwang, K. and Xu, J., 1993, Mapping Partitioned Program Modules onto Multicomputer Nodes Using Simulated Annealing, Int. Conf. Parallel Processing, 2, 292-293.

Hwang, K., 1993, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Singapore.

Logananthara j, R., 1997, A Method to Enhance A Simulated Annealing Technique, Proc. of the 10th Florida Artificial Intelligence Research Symposium, 217-221.

Nasr, G. E. and Moujabber, B. N., 1997, A New Deterministic Technique for Multiprocessor Scheduling, Proc. of 2nd LAAS International Conferences on Computer Simulation,. 117-122.

Van Driessche, R. and Piessens, R., 1992, Load Balancing with Genetic Algorithm, Parallel Solving from Nature, 2, Brussels, Belgium.