Efficient Scheduling of Task Graphs to Multiprocessors Using A Combination of Modified Simulated Annealing and List based Scheduling

Mahboobeh Houshmand*, Elaheh Soleymanpour*, Hossein Salami*,
Mahya Amerian** and Hossein Deldari*
*Dept. of computer engineering, Ferdowsi University
Mashhad, Iran
** Dept. of computer engineering, Mazandaran University
Babol, Iran
Ma.Hooshmand@stu-mail.um.ac.ir, el_so941@stu-mail.um.ac.ir, hosein.salami@stu-mail.um.ac.ir
,mahya.ameryan@gmail.com, hd@ferdowsi.um.ac.ir

Abstract- **Multiprocessor task scheduling is a well known NP-hard problem and numerous** methods have been proposed to optimally solve it. **The objective is makespan minimization, i.e. we want the last task to complete as early as possible. Simulated Annealing (SA) has been considered a very good tool for complex nonlinear optimization problem, such as multiprocessor task scheduling. However, a major disadvantage of the technique is that it is extremely slow. List-based scheduling algorithms are regarded as having acceptable results. In this paper we use a list scheduling based algorithm to find an initial solution and in the neighborhood generation phase of simulated annealing. We also parameterize SA and use a modified version of it. Simulation results show that our approach significantly improves the initial solution in considerably low time for different number of tasks; i.e. it efficiently outperforms the used list based scheduling approach.**

Keywords: List based scheduling algorithms, modified simulated annealing, NP hard problems, task scheduling problem.

I. INRTRODUCTION

Multiprocessor task scheduling has been well known as one of the hardest combinatorial optimization problems[1]. The problem of scheduling of tasks in multiprocessor systems is to determine *when* and on *which processor* a given task executes[2].

It was Kirkpatrick et al[3] who first proposed Simulated Annealing, SA, as a method for solving combinatorial optimization problems. It is reported that SA is very useful for several types of such these problems[4]. SA is a global optimization technique which traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation which lowers fitness is accepted probabilistically[2].

SA, single or in hybrid methods, has been used in multiprocessor task scheduling in some approaches [2, 5-10].

The most remarkable disadvantages of simulated annealing is it needs a lot of time to find the optimum solution and it is very difficult to determine the proper cooling schedule.

There are two approaches to shorten the calculation time in SA. One is determining the cooling schedule properly. The other approach is to perform SA on parallel computers[11].

A comprehensive study of best practices of simulated annealing for task mapping is presented in[12]. In this book chapter, cooling schedule of SA for the task scheduling problem is analyzed. The authors show that SA is a well performing algorithm if used properly. We use some methods of parameter selections of it. In addition, we apply some more improvements, as come in the following, to optimally solve the problem.

List based scheduling algorithms are generally regarded as having a good cost performance trade-off because of their low cost and acceptable results [13-15]. They assign priority levels to the tasks and map the highest priority task to the best fitting processing element. In our new approach, we use a modified version of SA to further improve the result, obtained by list based scheduling.

We apply the list based scheduling mechanism to generate an initial solution, and in the neighbor generation phase of SA to have a more informed search.

We are also inspired by the concepts in parallel simulated annealing[16], and use a concept, that we call agent. Simulation results show that this concept can also efficiently improve our proposed method, in terms of cost (makespan), and time, even while our approach is run on one single computer.

To sum up, we show that the combination of list scheduling based mechanism and a modified version of SA that we used can provide us a very good solution, in considerably low time; so can be used for scheduling of applications with large number of tasks.

The remainder of this paper is organized as follows. Section II states the problem. In section III, we introduce SA. Section IV describes our proposed task scheduling algorithms. Section V presents the experiment results. Then in Section VI, we conclude the paper.

II. PROBLEM STATEMENT

A homogeneous multiprocessor system is composed of a set P={p1…pm} of *M* identical processors. They are connected by a fully connected communication network

978-0-7695-4020-7/10 $26.00 © 2010 IEEE
DOI 10.1109/IITSI.2010.137

where all links are identical. Task preemption is not allowed.

The application program is modelled as directed acyclic graph (DAG), *G(T, E)* where T=$\{T_i : i = 1,...,n\}$ is a set of *N* tasks and *E* is a set of directed edges among the tasks representing the precedence.

For any two tasks, i, j $\in$ T, i< j means task *j* cannot be scheduled until *i* has been completed, *i* is a predecessor of *j* and *j* is a successor of *i.* Weights associate with nodes represent the computation time and weights associated with edges represent the communication cost. The multiprocessor scheduling is to assign the set of tasks *T* onto the set of processors *P* in such a way that precedence constraints are maintained and to determine the start and finish time of each task with the objective to minimize the completion time. We assume that the communication system is content free and allows the overlap of computation with communication. Task execution is started only after all data have been received from its precedence nodes. The communication links are full duplex. Communication is zero when two tasks are assigned to the same processor; otherwise they incur the communication cost equal to the edge weight[17].

III. INTRODUCTION OF SA

SA [3] is based on the analogy between statistical mechanics and combinatorial optimization. It can be viewed simply as an enhanced version of the familiar techniques of local optimization or iterative improvement, in which an initial solution is repeatedly improved by making small local perturbations until no further improvements.

The algorithm begins with an initial solution at high temperature 'T'. A second point is created by using perturbation scheme. The difference in the function values (delta) at these two points is calculated. If the second one has a smaller function value, this is accepted, otherwise it is accepted with a probability exp (-delta/T). The structure of the SA is as shown:

Step1: Get an initial solution S

Step2: Set an initial temperature, T>0

Step 3: While not frozen do the followings:

Step 3.1. Do the following n times:

Step 3.1.1. Sample a neighbor S' from S

Step 3.1.2. If delta <=0

Then set S=S'

Else set S=S'

With probability of exp(-delta/T)

Step 3.2 Set T=r*T, where r is the reduction factor

Step 4: Return S[18].

The advantages and disadvantages of SA are well summarized in[19].

IV. THE PROPOSED METHOD

In this section we present our proposed approach. The pseudo code of our method is shown in Fig. 1.

```
1.  $S_0 \longleftarrow$ list_CriticalPathFirst_EarliestStartTime(CPFES)
2.  $S \longleftarrow S_0$
3.  $C \longleftarrow$ Cost ($S_0$)
4.  $S_{best} \longleftarrow S$
5.  $C_{best} \longleftarrow C$
6.  for $i \longleftarrow$ 0 to 500//the iterations
7.      for $j \longleftarrow$ 1 to $A$ //$A$ denotes the number of agents
8.          $R \longleftarrow 0$
9.          $S' \longleftarrow S$
10.         $C' \longleftarrow C$
11.         $T \longleftarrow$ Temperature (i)
12.         Repeat
13.                 $w \longleftarrow$ generate a random number between
                    1 and $N$   //$N$ denotes the number of tasks
14.                 $z \longleftarrow$ generate a random number between
                    1 and $N$
15.                 tempSchedule $\longleftarrow$ alter S' by exchanging
                    $w$, $z$ on their processors
16.         Until feasible (tempSchedule)
17.         apply list_random_ EarliestStartTime to
            schedule after (w) and after (z) and generate
            a neighbor S" of S
18.         $C'' \longleftarrow$ Cost (S")
19.         $\Delta C \longleftarrow C'' - C'$
20.         $r \longleftarrow$ generate a random number between
            0 and 1
21.         $p \longleftarrow$ prob($\Delta C$, T)
22.         If $\Delta C < 0$ or $r < p$
23.             if $C'' < C_{best}$
24.                     $S_{best}[j] \longleftarrow S''$
25.                     $C_{best}[j] \longleftarrow C''$
26.             $S' \longleftarrow S''$
27.             $C' \longleftarrow C''$
28.             $R \longleftarrow 0$
29.         Else $R \longleftarrow R+1$
30.         if $R > R_{max}$
31.             break
32.     $S \longleftarrow$ the best in $S_{best}$
33.     $C \longleftarrow$ the best in $C_{best}$
34. Return S
```

Figure1. Pseudo-code of the proposed method

Before explaining the approach we describe three functions used in the pseudo code: *cost*, *after* and *feasible.*

after (i) returns the task numbers which are scheduled on the same processor as *i* in the current scheduling, and whose start time is after or equal to the finish time of the task *i*, in an ascending order.

feasible(S) gets the scheduling *S* and verifies if it is a feasible one, i.e. it conforms to the dependency relations between tasks in the DAG. *Cost(S)* returns the complementation time of the scheduling *S*.

We produce an initial solution by applying the list scheduling technique, which consists in the following steps: a) determine the available tasks to schedule b) define a priority to them and c) until all tasks are scheduled, select the task with higher priority and assign it to the processor that allows the Earliest Start-time.

Examples of heuristics for assigning priority are critical path [20, 21], job length [22]. The **Critical-Path-First, (CPF)** method tries to select tasks on the critical path prior to other tasks. The flavor of this method can be seen in [23-25].

We generate partial-order task sequences based on CPF ordering. We call the approach we used to construct the initial solution, CPFES.

Our initial solution is improved by SA. In the neighbour generation phase, first we repeatedly select two tasks randomly and exchange them on their processors until the obtained schedule is feasible. This method is introduced in [12]. Then we again apply a list based scheduling. (See the pseudo-code for more details). In this phase, when more than a partial-order sequence exists that conforms to the dependency relations between tasks, we randomly choose one. The list based scheduling algorithm we use in this phase helps us have a more informed search and not ruin the good solution that has been produced previously by our approach.

Being inspired by the concepts of parallel simulated annealing[16], we use a concept, that we call agent. We suppose there are some concurrent agents that all get a scheduling *S* and apply the modified SA algorithm on it a number of times and produce another scheduling *S"*. Each agent ends its work when sufficient number of consecutive moves has been rejected. Then, the schedulings *S"* of the agents are gathered and the best of them is broadcasted to all agents to be used in the next iteration (the loop with *i* index in the pseudo-code indicates the iterations).There is a total number of 500 iterations that this action repeats.

Although we run our approach on one single computer, simulation results show that this simulated concept makes further improvements, in terms of time and cost.

It is very difficult to determine the proper cooling schedule for SA. To determine it, many preparatory trials are needed [11]. In[12] a comprehensive study of the best practices of simulated annealing for task mapping is presented.

We use a part of the proposed method in this book chapter to parameterize SA. This method automatically selects parameters for a modified simulated annealing algorithm to save optimization effort.

TABLE I shows the parameters that we used. ΔC_{max} in this table shows maximum expected cost change. As estimation we consider it as (1).

$$\sum_{i=1}^{N} computationTime(Task\ i) \qquad (1)$$

Where *computationTime(Task)* returns the time needed to execute this task, and *N* is the number of processors.
C_0 (in the TABLE I) is the cost of the initial solution.

TABLE I. The used SA parameters

Parameter	Value	Meaning
T=Temperature (i)	$T_0 \times (0.95)^{\left\lfloor \frac{i}{R_{max}} \right\rfloor}$	Return temperature T at iteration i
prob (ΔC, T)	$\frac{1}{1 + \exp\left(\frac{-\Delta C}{C_0 T}\right)}$	The acceptance probability function
T_0	$\frac{\Delta C_{max}}{\ln\left(\frac{1}{0.45} - 1\right)}$	The initial temperature
R_{max}	M(N-1)(*M* denotes the number of processors)	Maximum Number of consecutive rejected moves

V. SIMULATION RESULTS

We implemented our approach in Borland C++ 5.02, using a system with 2 GBs of RAM and the CPU 2.2 GHz.
We defined a parameter α as equation (2):

$$\alpha = \frac{\sum_{i=1}^{N} computationTime(Task\ i)}{M} \qquad (2)$$

Where *computationTime (Task)* returns the time needed to execute this task, and *M* is the number of processors.

We generated DAGs with random structures with the different number of nodes (tasks). We assigned weights to the nodes (the computation time) in such a way that α =50. We also made random numbers between 0 and 5 to assign weights to the edges (the communication cost). These constraints enabled us to have a more exact compare.

Since we applied CPFES, as the initial solution and applied the modified SA on it, we considerably improved the initial solution, which itself is regarded to be an acceptable one [13-15, 23-25].

There are three main parameters in our approach, *M*, *N*, *A*, that denote the number of processors, tasks and agents respectively.

In the first step we wanted to study how our approach works when the number of tasks increases. So we fixed *A* and *M* to 5 and 15 respectively. We produced three DAGs (with the specifications mentioned in the first paragraph of this section) with 100, 150 and 200 nodes (tasks), respectively. Fig. 2 and TABLE II show our approach produces very good improvements in considerably low time, for different number of tasks.

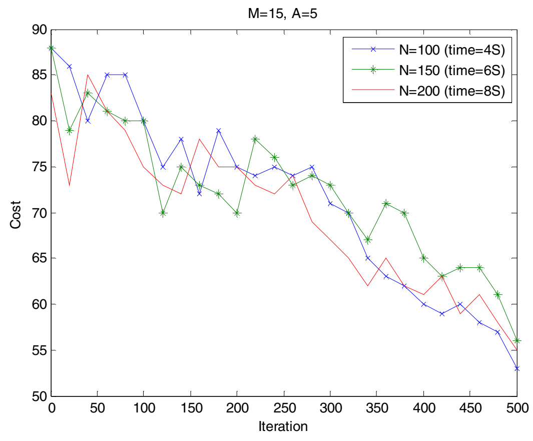

Figure 2. Simulation results, when *N* varies

TABLE II summarizes the results shown in Fig. 2.

TABLE II. Simulation results, A=5, M=15.

N	Relative improvement of initial solution (CPFES)	Time
100	39.7 %	4S
150	36.4%	6S
200	31.9%	8S

In the second step, we wanted to study the role of the new concept, agent, what we used. So we produced a DAG (with the specifications mentioned in the first paragraph of this section) with 150 nodes. We fixed *M* to 15. We applied our approach to the DAG three times and set the parameter *A* to 1, 5 and 10, respectively. When *A* is set to 1, it is like we did not use the concept. Fig. 3 shows when the number of agents increases our approach improves in terms of simulation time and cost.

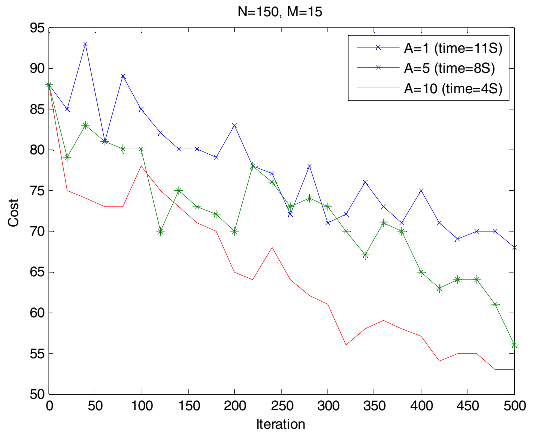

Figure 3. Simulation results, when *A* varies

TABLE III summarizes the results shown in Fig.3.

TABLE III. Simulation results, N=150, M=15.

A	Relative improvement of initial solution (CPFES)	Time
1	20 %	11S
5	36.4%	8S
10	39.8%	4S

In the third step, we wanted to study how our approach works when the number of processors varies. So, we used the DAG generated in the second step and fixed the parameter *A* to 5. We applied our approach to the DAG three times and set the parameter *M* to 10, 15 and 20, respectively. Simulation results show when the number of processors increases, the simulation time does not increase, and the cost decreases.

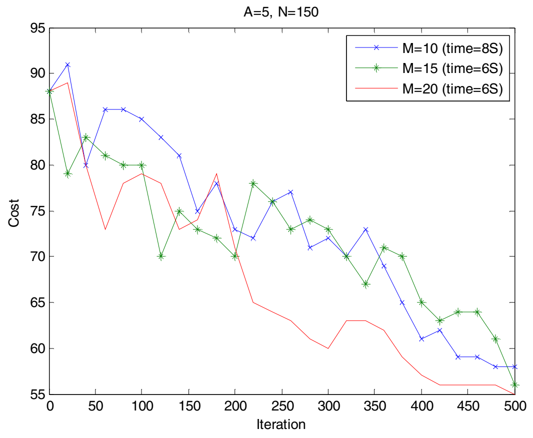

Figure 4. Simulation results, when *M* varies

TABLE IV summarizes the results shown in Fig. 4.

TABLE IV. Simulation results, A=5, N=150.

M	Relative improvement of initial solution (CPFES)	Time
10	34.1 %	8S
15	36.4%	6S
20	38%	6S

VI. CONCLUSION

In this paper we used list based scheduling with the critical path priority that schedules each task on a processor in a way that minimizes the start time, as an initial solution of SA. This method produces an acceptable result, which we further improved by a modified version of SA. Based on a previously proposed method, we parameterized SA, to make **algorithm scalable with respect to application and platform size. In the neighbor generation phase of SA, we again applied the list based scheduling algorithm to some tasks, to have a more informed search.**

Being inspired by the concepts in parallel simulated annealing we used a concept that we called 'agent'. This concept also improved our results, in the term of cost and time.

Simulation results showed that our proposed method produced very good solutions, in considerably low time, even while running on the DAGs with a large number of tasks.

REFERENCES

[1] D. Applegate and W. Cook, "A computational study of the Job-shop Scheduling problem," *ORSA Journal on Computing,* vol. 3, pp. 149-156, 1991.

[2] S. Sutar, J. Sawant, and J. Jadhav, "Task scheduling for multiprocessor systems using memetic algorithms," in *4th International Working Conference Performance Modeling and Evaluation of Heterogeneous Networks (HET-NETs '06)*, 2006.

[3] S. Kirkpatrick, G. J. C. D., and M. P. Vecchi, "Optimization by Simulated Annealing," vol. 220, pp. 671-680, 1983.

[4] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing* Wiley, 1989.

[5] H. Orsila, T. Kangas, E. Salminen, and T. D. H¨am¨al¨ainen, "Parameterizing Simulated Annealing for Distributing Task Graphs on Multiprocessor SoCs," in *Symposium on System-on-Chip, Tampere*, Finland, 2006, pp. 73-76.

[6] S. Wanneng and Z. Shijue, "A parallel genetic simulated annealing hybrid algorithm for task scheduling " *Wuhan University Journal of Natural Sciences,* pp. 1378-1382, 2008.

[7] G. E. Nasr, A. Harb, and G. Meghabghab, "Enhanced Simulated Annealing Techniques for Multiprocessor Scheduling," in *Proceedings of the Twelfth International FLAIRS Conference*, 1999.

[8] A. K. Nanda, D. DeGroot, and D. L. Stenger, "Scheduling directed task graphs on multiprocessors using simulatedannealing," in *Proceedings of the 12th International Conference on Distributed Computing Systems, 1992.*, Yokohama, Japan.

[9] M. E. Aydin and T. C. Fogarty, "A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application " *Journal of Intelligent Manufacturing,* pp. 805-814, 2005.

[10] M. König and U. Beißert, "Construction Scheduling Optimization by Simulated Annealing," *26th International Symposium on Automation and Robotics in Construction (ISARC 2009),* 2009.

[11] M. Miki, T. Hiroyasu, M. Kasai, K. Ono, and T. Jitta, "Temperature parallel simulated annealing with adaptive neighborhood for continuous optimization problem," *The International computational Intelligence and applications,* pp. 149-154, 2002.

[12] H. Orsila, E. Salminen, and T. D. Hämäläinen, "Best Practices for Simulated Annealing in Multiprocessor Task Distribution Problems (book chapter)," in *Simulated Annealing*: I-Tech Education and Publishing KG, 2008, pp. 321-342.

[13] Y.-C. Chung and S. Ranka, "Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors," in *Supercomputing '92: Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, 1992, pp. 512–521.

[14] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling dags on multiprocessors," *Journal of Parallel and Distributed Computing,* pp. 276–291, 1992.

[15] J.-C. Liou and M. A. Palis, "A comparison of general approaches to multiprocessor scheduling," in *Proc. International Parallel Processing Symposium*, 1997, pp. 152–156.

[16] D. J. RAM, T. H. SREENIVAS, and K. G. SUBRAMANIAM, "Parallel Simulated Annealing Algorithms," *Journal of Parallel and Distributed Computing,* vol. 37, pp. 207–212, 1996.

[17] M. K. Dhodhi, I. Ahmad, and I. Ahmad, "A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms," in *IEEE International Conference on Evolutionary Computation*, 1996.

[18] S. Benedict and V. Vasudevan, "Scheduling of scientific workflows using simluated anealing algorithm for computional grids," *International journal of soft computing,* vol. 2, pp. 606-611, 2007.

[19] L. Ingber, "Simulated Annealing: Practice versus Theory," *Mathl. Comput. and Modelling,* vol. 18, pp. 29-57, 1993.

[20] K. R. Pattipati, R. T. Kurien, T. Lee, and P. B. Luh, "On Mapping a Tracking Algorithm Onto Parallel Processors," *IEEE Transactions on Aerospace and Electronic Systems,* vol. 26, pp. 774–791, 1990.

[21] T. Pop, P. Eles, and Z. Peng, "Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems," in *Proceedings of the tenth International Symposium on Hardware/software Codesign*, 2002, pp. 187–192.

[22] Y. Shin and K. Choi, "Enforcing schedulability of multitask systems by hardware-software codesign," in *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*, 1997, pp. 3–7.

[23] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments.," in *Proc. International Conf. on Parallel Processing Workshops*, 2003, pp. 149–155.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performanceeffective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distributed Syst,* vol. 13, pp. 260–274, 2002.

[25] T. Hagras and J. Janecek., "An approach to compile-time task scheduling in heterogeneous computing systems," in *Proc. International Conf. on Parallel Processing Workshops*, 2004, pp. 182–189.