Duplication based List Scheduling in Heterogeneous Distributed Computing

R.S.Singh,
Department of Computer Engineering
Institute of Technology
BHU, Varanasi, India

A.K.Tripathi
Department of Computer Engineering
Institute of Technology
BHU, Varanasi, India

S.Saurabh
Department of Computer Engineering
Institute of Technology
BHU, Varanasi, India

V.Singh
Department of Computer Engineering
Institute of Technology
BHU, Varanasi, India

ABSTRACT
Whenever tasks of an application are scheduled in Heterogeneous Distributed Computing environment, idle slots on processors are efficiently utilized to minimize the overall running time. Since task assignment problem has been proved to be NP-complete problem, many heuristics have been given in the literature caring empty slots on processors as well as dependencies among tasks. This paper presents an efficient and effective way to allocate tasks of an application in the Heterogeneous Distributed Computing environment. Generally in list based static scheduling where computation time and communication time are known a-priori. First tasks are prioritized and then the processors that minimize the cost function are assigned to the appropriate tasks. Duplication based scheduling is another category of static scheduling. In this category communication costs among the processors are avoided by duplicating the tasks on same processor. This paper presents a duplication based list scheduling that overwhelms the existing scheduling algorithms in both the categories.

Keywords
Distributed Computing, Critical Path, Heterogeneous System, Static Scheduling.

1. INTRODUCTION
Heterogeneous Distributed Computing involves independent resources of diverse capabilities interconnected by a high speed network to solve computationally intensive parallel and distributed applications. It becomes important to be able to maximize the utilization of computing and communication infrastructure for justifying the cost that may have gone into creation of these software and hardware resources. Fortunately computational demands of jobs and applications have also been increasing quite fast and the trend is likely to continue into the future as well.

The performance of a compute intensive application on such platforms is highly dependent on the allocation of the tasks onto these resources. One of the major problems in Heterogeneous Distributed Computing is to schedule the tasks of an application such that overall running time is minimized. Task scheduling problem is proven to be NP-complete [1,2]. Many heuristics have been proposed in the literature for tasks scheduling problem as there is no exact solution to NP-complete problem. Task scheduling is characterized into two categories: static scheduling and dynamic scheduling. In static scheduling, which is done at compile time, all the information associated with a parallel program such as task processing time, communication time and data dependencies are known a-priori. In dynamic scheduling, many scheduling decisions of a parallel application are taken at run time. Thus the objective of dynamic scheduling is not only to schedule tasks but also consider the scheduling overhead, fault tolerance issues etc. This paper has consideration to static scheduling.

Various static scheduling heuristics have been proposed in the literature. On basis of the approaches these heuristics use, they have been classified into four groups: list scheduling algorithms, cluster based algorithms, duplication based algorithms and random search algorithms.

In list based heuristics, tasks are put in a priority list with each task having unique priority value. Priority of a task depends upon priorities of its ancestors. Now tasks from the priority list are taken one by one following three phases: task selection phase, processor selection phase and status update phase. In task selection phase highest priority task is taken for scheduling. In processor selection phase, extracted task is assigned to a processor that optimizes some predefined cost function. The status update phase updates the status of the system. HEFT, CPOP, LDCP etc. are list based heuristics [3-7].

In cluster based algorithms, a set of tasks, communicating with each other are grouped together to form a cluster [8,9]. If the number of clusters is greater than the number of processors available, the two communicating clusters are grouped to make a single cluster and this process is repeated until the number of clusters available equals the number of resources available. Now each cluster is allocated to the processors in such a way that overall running time is minimized.

In duplication heuristics the highly communicating tasks are redundantly allocated on the same processors. This is to effectively reduce the start time of the waiting tasks and thus improve overall running time of the applications. Duplication based heuristics are useful in case of Heterogeneous Distributed Computing System having high communication latencies and low bandwidths [10-16].

Guided random search techniques are based on enumerative techniques to search guided by some additional information. It uses principle of evolution and natural genetics. A genetic algorithm is one type of evolution computation that is commonly used [17-19].

This paper combines list based scheduling and duplication based scheduling approaches and gives a hybrid static scheduling algorithm. Task duplication approach can effectively be used in any list based heuristic. Idea is to effectively use the time slot on processors during which no task has been scheduled by a list based scheduling algorithm. We analyze tens of thousands of experimental runs to explain that why the proposed algorithm performs better in certain settings.

The rest of the paper is organized as follows: Section 2 and Sectioan 3 cover the basic DAG (Directed Acyclic Graph) model to represent a parallel application. DAGs to some important problem have also been given in this section. Performance metrics are also given in this section. Section 4 gives basic idea of random task graph generator. This random task graph generator is based on certain important parameters. Parameters are also listed in the section. Section 5 gives the new hybrid static scheduling algorithm and theoretically explains why it should work better in certain settings. Experimental evaluation of our work is given in Section 6. Conclusion and planned future work is presented in Section 7.

2. PRELIMINARIES

DAG Model

A parallel application is represented in form of a DAG. A DAG G= (V,E) consists of a set of nodes V and a set of directed edges E. Set V represents tasks and set E represents dependencies among tasks. A directed edge (i,j) E G, represented dependency of task j(child) on task i(parent). This dependency shows that child task can not start execution before execution of its parent as well as child task has to receive data from the parent task. If there is more than one parent to a child task, child task shall only be executed when its entire parents are executed and data from all parent task have been received by the child task. Data is an e*e matrix where Data (i,j) is the communication cost between task i and task j. If both task i and task j reside on same processor, Data (i,j) is assumed to be zero.

In a DAG a task without any parent is called entry task and a task without any child is called an exit task. Some scheduling algorithms may require single-entry and single-exit task graph, so without loss of generality it is assumed that there is one entry node to the tasks graph and one exit node to the task graph. If there is more than one entry node in a DAG, they are connected by a zero cost pseudo entry node by zero cost pseudo edges. In these connections pseudo entry node is parent to all the entry nodes. In the similar fashion there is a pseudo exit node of zero cost and this exit node is set child of entire exit nodes by zero cost pseudo edges. Fig. 1 gives a DAG corresponding to Fast Fourier Transform Algorithm while Fig.2 corresponds to Gaussian Elimination algorithm.

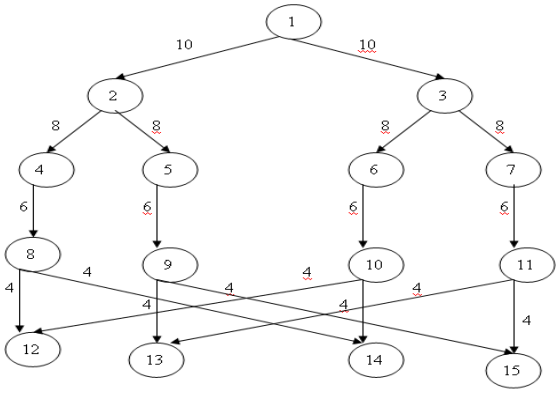

Fig 1: A DAG for FFT Algorithm

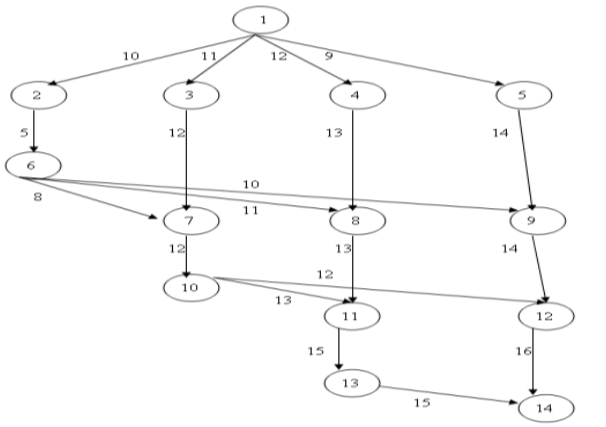

Fig 2: A DAG for Gaussian Elimination Algorithm

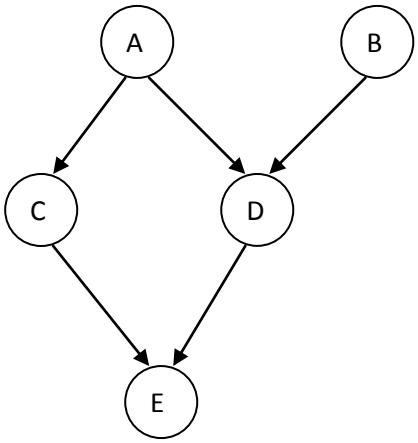

Fig 3: An Example DAG

Resource Model

Heterogeneous Distributed Computing environment has diversely capable for solving parallel and distributed applications. It is assumed that dedicated communication channels are available i.e. bandwidth is contention free. Communication cost between processors P1 and P2 depends on network initialization at P1 and P2 in addition to the communication time. Time for network initialization is considered to be negligible with respect to the communication time in the network. It is also assumed that data transfer rate is fixed and constant. So the communication cost of an edge (i,j) is equal to the amount of data to be transmitted from task i to task j divided by data transfer rate of the network. Without loss of generality, data transfer rate is assumed to be unity. We also assume that heterogeneous processors are fully connected i.e. a processor is connected to every other processor in the network. Computation cost matrix C has running time of all the tasks on all the processors. C (i,j) represents the computation cost of task i on processor j. In a heterogeneous system it is not necessary that if a task i takes lesser time on processor j than processor k, then every task will have lesser time on processor j than k. This is due to diverse capabilities of the processors. Fig. 3 shows an example DAG and Fig. 4 is computation cost matrix.

The DAG has five tasks labelled A, B, C, D and E. Dependencies among these tasks are also there for example

Task D is dependent upon task A and task B. Computation cost matrix gives running time of tasks on processors P1 and P2.

3. PERFORMANCE METRICS

Our scheduling objective is to optimize certain metrics. These metrics are schedule length ratio, speed up and running time. These metrics are generally used for evaluation of the scheduling algorithms. These metrics are described below.

Schedule Length Ratio (SLR)

SLR is one of the important performance measures of scheduling algorithm. SLR is defined by

SLR= makespan/ $\sum_{Ni \varepsilon CPmin} \min_{Pj \varepsilon Q} \{Wi,j\}$

where makespan is the overall schedule length. The denominator is the sum of minimum computation costs of tasks on the CPmin. CPmin is the critical path which is obtained by minimum computation cost assignment to the nodes of DAGs. The SLSR can never be less than one, since the denominator is the lower bound. Algorithm that gives smallest SLR of a graph, is the best algorithm with respect to performance.

Speed up

Another performance measure of scheduling algorithm is speed up. Speed up is defined by

Speed up= $\min_{Pj \varepsilon Q} \{\sum_{Ni \varepsilon V} Wi,j\}$ / makespan

The numerator is sequential execution time. Sequential execution time is computed by assigning all tasks to single processor that minimizes the overall computation time. The denominator is parallel execution time. Higher the speed up of an algorithm decides the goodness of the algorithm with respect to speed p.

	P1	P2
A	2	3
B	3	6
C	10	10
D	12	3
E	5	12

Fig 4: A computation cost matrix for example DAG

Average Running Time

Average Running Time is sum of running times of different DAGs divided by number of DAGs. So an algorithm with respect to Average running time is better if it has smaller Average running time.

4. RANDOM TASK GRAPH GENERATOR

For the comparison of the results of proposed algorithm with the results of some good existing algorithms, a Random Task Graph Generator (RTGG) has been designed. RTGG avoids biasing towards any specific algorithms. RTGG generates a large number of task graphs with controlled variation over the various graph properties. Our framework first executes the RTGG program to generate the task graphs. Then by executing the different programs related to different algorithms, schedules are generated. Finally a program computes the performance metrics based on the schedules generated.

Our RTGG requires the following input parameters to build weighted DAGs:

i)Task size in the graph(v): the value of v is assigned from the set (20,40,60,80)

ii)Shape parameter of the graph (α): we assume that height of a DAG is $\sqrt{v}/\alpha$. α gets its value from (0.5, 1.0, 2.0). the width of each level is randoml selected from a uniform distribution with mean value to $\sqrt{v} * \alpha$. If $\alpha >> 1.0$, dense graph i.e. high parallelism degree graph is generated else if $\alpha << 1.0$, long graph with low parallelism degree is generated.

iii)Out degree of a node (out_degree): out_degree sets its value from 1 to v.

iv)Communication to computation cost ratio (CCR): it is ratio of the average communication cost to average computation cost. If CCR value is very low for a graph, the graph represents a compute intensive application. CCR gets its value from (0.1, 0.5, 1.0, 5.0, and 10.0).

v)The average computation cost of task graph: this is selected randomly from a predefined set. Computation cost of each task in a graph is selected randomly from [0, 2*WDAG] where WDAG is average computation cost of the given graph.

5. PROPOSED ALGORITHM

Selection of nodes for duplication is different than duplication based algorithm in proposed algorithm. To reduce the start time of nodes, some algorithms duplicate only the parent nodes as well as some algorithms try to duplicate ancestors at higher level. We have implemented the concept of Task Duplication in the pre-existing CPOP Algorithm [4]. CPOP algorithm has two phases the task prioritzation and processor selection phase . The application of Task duplication comes in the processor selection phase. Whenever a task is being scheduled, then we find out its Very important parent and check whether duplicating this on any processor will reduce the earliest finish time of task. If yes, then duplicate the parent task on processor and then schedule the task. We must notice that since the purpose of duplication is only to reduce the communication cost hence whenever we try to find EST of n_i on any processor pj then the duplication is checked only on p_j because if the parent node is duplicated on any other processor then also there is no improvement in EST of n_i.

We made a function to check whether n_k can be duplicated on p_j or not, it checked the availability of p_j at that time as well as checked that is it really worthy to duplicate n_k. Rest of the procedure is same as it used to be in the case of classic HEFT and CPOP algorithms [4].

1. Set computation cost of task and communication costs of edge with mean values.

2. Compute upward rank (ranku) by traversing graph upward starting from the exit task.

3. Compute downward rank (rankd) by traversing graph downward, starting from entry task.

4. Compute priority(ni) = rankd(ni) + ranku(ni) for each task ni in the graph.

5. |CP| = priority(nentry), where nentry is the entry task.

6. SETCP = {nentry}, where SETCP is the set of task on the critical path.

7. nk <-nentry

8. while nk is not exit task do

9. Select nj where ((nj € successor(nk)) and (priority(nj) == |CP|))

10. SETCP = SETCP Union {nj}

11. nk <-nj

12. end while

13. Select critical path processor {PCP} while minimizes

$\sum Nj\varepsilon SETCP\ wi,j,$ for pj € Q.

14. Initialize the priority queue with entry task.

15. while there is an unscheduled task in the priority queue do

16. Select the highest priority task ni from priority queue.

17. if ni € SETCP then

18. Assign the task ni on PCP

19. else

20. for all processors pi

21. if VIP(ni) can be duplicated on pj , duplicate it on pj

22. EST (ni,pj) = EFT(nk,pj)

23. else EST(ni,pj) = AFT(nk) + ci,k

24. EFT(ni,pj) = EST(ni,pj) + wi.j

25. Assign processor with least EFT to nj

26. end for

27. end while

6. RESULTS

For the CPOP algorithm, which is a list based scheduling duplication of tasks sufficiently, decreases the execution time of an application[4]. The reason is that in CPOP all the tasks that lie on critical path must be scheduled on critical path processor. The CCR value was kept 0.1 and the value of alpha was kept 0.5. For each value of parameters thousands of graphs were generated and scheduled using HEFT, CPOP and duplication based CPOP algorithms [4]. Then the average values of SLR, speedup and execution time for each test case were calculated. The average execution time of duplication based CPOP is better than the HEFT, CPOP algorithms. Fig. 5 gives graph for number of nodes vs. average execution time. It is clear that with increase in number of nodes, average execution time in duplication based CPOP decreases faster than any other algorithm. Similarly on average duplication based CPOP has better average schedule length ratio. Fig. 6 depicts the relation between number of nodes and average schedule length ratio. Fig. 7 depicts that quality of schedule with respect to speed is also better in proposed algorithm than any other algorithm.

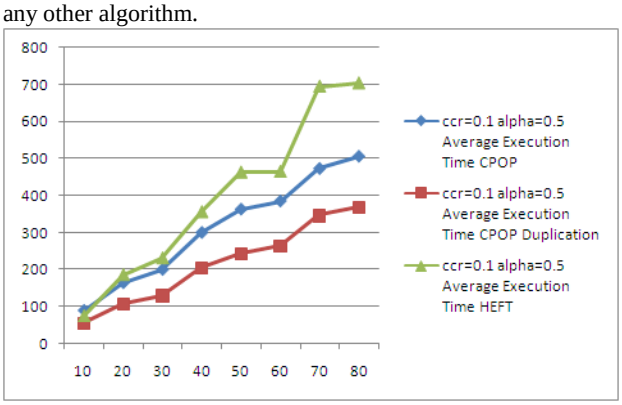

Fig 5: No. of nodes vs. average execution time

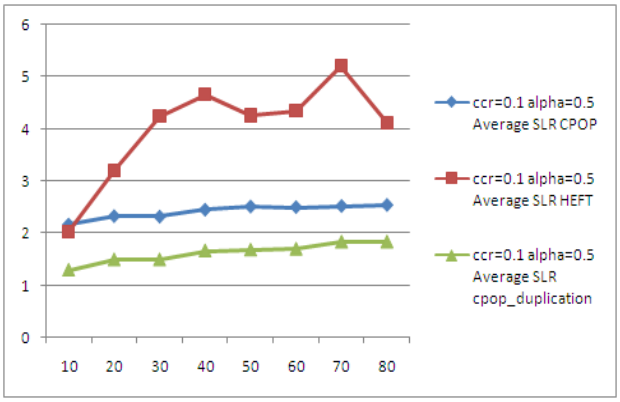

Fig 6: No. of nodes vs. average SLR

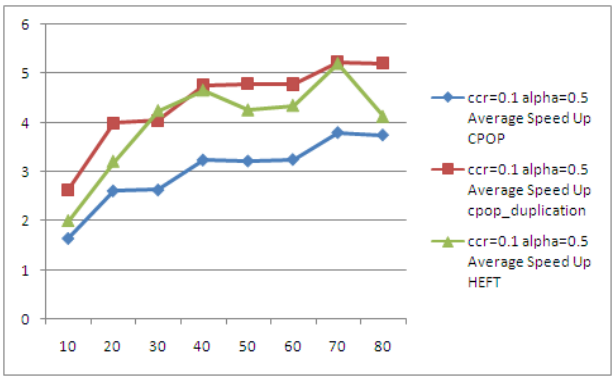

Fig 7: No. of nodes vs. average speedup

The results obtained are about 30-40% better than those produced by CPOP algorithm and about 40-50% better than HEFT algorithm. The results get increasingly better as the number of nodes is increased from 20 to 90.

7. CONCLUSION AND FUTURE WORK

In this paper we present a duplication based list scheduling algorithm for scheduling task of an application onto a heterogeneous computing system. To avoid biasing towards proposed algorithm random task graph generator has been used to compare the scheduling results of proposed method and some existing efficient methods. Random task graphs are generated by deciding important input parameters like number of nodes, communication to computation cost ratio and shape parameters. This selection makes a wide range of task graphs

with various characteristics. Speedup, average running time and schedule length ratio are the three metrics that decide goodness of a scheduling algorithm. Experimental results how that duplication based list scheduling heuristic often outperforms many other scheduling algorithms in duplication based algorithm category as well as in list based scheduling algorithm category.

In future we will extend the work to partially connected resources. Availability based scheduling is another future direction to work, where some processors may have internal job queues.

8. REFERENCES

[1] Garey, M. R. and Johnson D. S. 1979 Computers and Intrctability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA

[2] Yang, T. and Gerasoulis, A. 1994 DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans. Parallel Distrib. Syst., 5(9):951–967.

[3] Ilavarasan, E. and Thambidurai, P. 2007 Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments. Journal of Computer Sciences 3 (2): 94-103

[4] Topcuoglu, H., Hariri, S., and Wu, M.Y. 2002 Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst., 13(3):260–274.

[5] Adam, T. L., Chandy K. M., and Dickson J. R. 1974 A comparison of list schedules for parallel processing systems. Commun. ACM, 17(12):685–690.

[6] Kwok, Y.K. and Ahmad, I. 1999 Benchmarking and comparison of the task graph scheduling algorithms. J. Parallel Distrib. Comput., 59(3):381–422.

[7] Liu, G. Q., Poh, K. L. and Xie., M. 2005 Iterative list scheduling for heterogeneous computing. J. Parallel Distrib. Comput., 65(5):654–665.

[8] Liou, J. and Falls, M. A. 1996 An efficient clustering heuristic for scheduling dags on multiprocessors. In Proc. Symp. Parallel and Distributed Processing.

[9] Bittencourt,L. F., Madeira, E. R. M., Cicerre, F. R. L. , and Buzato, L. E. 2005 A Path Clustering Heuristic for Scheduling Taks Graphs onto a Grid. 3rd ACM International Workshop on Middleware for Grid Computing, Grenoble, France. Nov.

[10] Ahmad, I. and Kwok, Y.K. 1994 A new approach to scheduling parallel programs using task duplication. In Proc. Int'l Conf Parallel Processing, volume 2, pages 47–51.

[11] Ahmad, I. and Kwok,Y.K.1998 On exploiting task duplication in parallel program scheduling. IEEE Trans. Parallel Distrib. Syst, 9(9):872–892.

[12] Park, G.L., Shirazi, B. and Marquis, J. 1997 DFRN: A new approach for duplication based scheduling for distributed memory multiprocessor systems. In IPPS '97: Proceedings of the 11th International Symposium on Parallel Processing, pages157–166, Washington, DC, USA, IEEE Computer Society.

[13] Kwok, Y.K. and Ahmad, I. 1994 Exploiting duplication to minimize the execution times of parallel programs on message-passing systems. In Proceedings of the 6th Symposium on Parallel and Distributed Processing, pages 426–433, Los Alamitos, CA, USA, October, IEEE Computer Society Press.

[14] Ranaweera , S. and Agrawal, D. P. 2000 A task duplication based scheduling algorithm for heterogeneous systems. In 14th International Parallel and Distributed Processing Symposium (SPDP'2000), pages 445–450, Washington - Brussels - Tokyo, May.

[15] Darbha, S. and Agrawal, D. P. 1997 A task duplication based scalable scheduling algorithm for distributed memory systems. J. Parallel Distrib. Comput, 46(1):15–27.

[16] Bansal, S., Kumar, P. and Singh, K. 2003 An Improved Duplication Stratgy for Scheduling Precedence Constrained Graphs in Multiprocessors. IEEE Trans. Parallel and Distributed Systems,Vol.31, No.4,pp.533-544,june.

[17] Goldberg, D. E. 1989 Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, Massachusetts.

[18] Wang, L., Siegel, H. J., Roychowdhury, V. P., and Maciejewski, A. A. 1997 Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. Journal of Parallel and Distributed Computing,47(1):8–22, November.

[19] Ali, S., Sait, S. M., and Benten, M. S.T. 1994 GSA: Scheduling and allocation using genetic algorithm. In Proceedings of the 1994 European Design Automation Conference, pages 84–89, Toronto, Canada, September.