A Novel Task Scheduling Algorithm for Heterogeneous Computing

Vinay Kumar
SC&SS
Jawaharlal Nehru University
New Delhi, INDIA

C. P.Katti
SC&SS
Jawaharlal Nehru University
New Delhi, INDIA

P. C. Saxena
SC&SS
Jawaharlal Nehru University
New Delhi, INDIA

ABSTRACT
The grid computing system can support the execution of computationally intensive parallel and distributive applications. The main characteristics of grid computing and heterogeneous computing system are similar. A novel scheduling algorithm, called NHEFT is proposed in this paper to enhance the functions of heterogeneous Earliest-Finish time (HEFT) algorithm. The NHEFT algorithm works for a bounded number of heterogeneous processors, the main objective of NHEFT is getting high performance and fast scheduling. The algorithm selects the tasks with a rank system at each step of execution of algorithm, which minimize earliest finish time with the minimization of cost.

General Terms
Distributed Computing, Heterogeneous Computing, Grid Computing.

Keywords
Task Scheduling Problem, NP Problems, Dynamic Scheduling.

1. INTRODUCTION
Most of previous research in parallel and distributive computing was focused on homogeneous computing system. In recent years, most of the research on scheduling algorithm can support the heterogeneous computing system [1], [2]. In a heterogeneous environment, it is assumed that the bandwidth of every communication channel and computing power of every processor can be different. A task scheduling problem can be represented by a direct acyclic graph (DAG) [3], in which tasks are represented by nodes and inter-task data dependencies are represented by edges. Each node label shows expected computation time of the task and each edge label shows inter-task expected communication time between tasks. The objective function of this problem is to map task onto processors and order their executions so that the task precedence requirements are satisfied and a minimum overall completion time is obtained. In general case a task scheduling problem is NP-complete problem [4].

In this paper, a new scheduling algorithm is proposed for a bounded number of fully connected heterogeneous processors. The motivation behind algorithm is to deliver good quality of schedule with lower costs. In this study the NHEFT algorithm selects the task with highest sum of top rank and down rank at each step. The selected task is assigned to processor which minimizes its earliest finished time with an insertion based approach [5].

The remainder of this paper is organized as follows: section 2 defines task scheduling problem and related terminology. Section 3 introduces proposed scheduling algorithm. Section 4 presents a comparison study of proposed algorithm with the related algorithms like MH [3], DLS [6], LMT [6] and HEFT [14]. The comparison study is based on randomly generated task graphs which have several real applications. The summary of the research is presented in section 5.

2. TASK SCHEDULING PROBLEM AND RELATED WORK
The application of task scheduling system is represented by a directed acyclic graph, G= (V, E) where V is the set of tasks and E is the set of edges between tasks. Each edge (i, j) ∈ E represents the constraint such that the task n_i should complete its execution before task n_j start. Here data is a v × v matrix of communication data, where v is total number of tasks. In a task graph, a task without any parent is called an entry task (n_{entry}) and a task without any child is called an exit task (n_{exit}).

In the target computing environment it is assumed that the set Q of q number of heterogeneous processors connected in a fully connected topology, so all inter-processor communications are assumed to be performed without contention. It is also assumed that computation can be overlapped with communication. As in previous terminology on task scheduling problem [6], W is a v × q computation cost matrix in which each $w_{i,j}$ gives the estimated execution time to complete task n_i on processor p_j. Before scheduling, the tasks are labeled with the average execution cost.

The average computation cost of a task n_i is defined as [5]

$$\overline{w}_i = \sum_{j=1}^{q} \frac{w_{i,j}}{q} \quad (1)$$

The data transfer rates between the processors are stored in matrix *B* of size $q \times q$. The communication startup cost of processor is given as a *q*-dimensional vector *L*. The communication cost of the edge *(i, k)* which is for transferring data from task n_i (scheduled on p_m) to task n_k (scheduled on p_n) by transfer rate $B_{m,n}$ is defined by

$$C_{i,k} = L_m + \frac{Data_{i,k}}{B_{m,n}} \quad (2)$$

where L_m is communication startup cost of processor p_m.

If task n_i and n_k are on the same processor, then $C_{i,k}$ = 0. The average communication cost of an edge *(i, k)* is defined by

$$\bar{C}_{i,k} = \bar{L} + \frac{Data_{i,k}}{\bar{B}} \quad (3)$$

where $\bar{L}$ is the average communication startup time.

EST (n_i ,p_j) and *EFT* (n_i ,p_j) are the earliest execution start time and the earliest execution finish time of task n_i on processor $p_j$respectively. The *EST* and *EFT* values are computed recursively, starting from the entry task as shown in equation (4) and (5). In order to compute the *EFT* of a task n_i , all immediate predecessor task of n_i must have been scheduled.

$EST(n_i,p_j)$ =

$$max\{avail\ (j),\ max_{n_m \in pred(n_i)}(AFT\ (n_m)\ + C_{m,i})\} \quad (4)$$

$$EFT\ (n_i,p_j) = w_{i,j} + EST\ (n_i,p_j) \quad (5)$$

For the entry task

$$EST\ (n_{entry}, p_j) = 0 \quad (6)$$

Where *avail (j)* is the time that processor p_j is free and it is ready to execute task n_i , the inner *max* block in equation (4) returns the ready time i.e., the time when all data needed by task n_i has arrived at processor p_j. After all tasks in a graph are scheduled, the scheduled length (overall completion time) will be the *AFT* of the exit task (n_{exit}). If there are multiple exit tasks and the convention of inserting a pseudo exit task is not applied, the schedule length, called makespan, is defined as

$$\text{makespan} = max\ [AFT(n_{exit})] \quad (7)$$

The next section presents the task scheduling algorithms that support heterogeneous processors. In Dynamic level scheduling (DLS) algorithm [6], at each step, algorithm selects the pair that maximizes the value of the dynamic level which is equal to

$$DL\ (n_i\ ,p_j) = rank\ _u^s\ (n_i) - EST\ (n_i\ ,p_j).$$

The computation cost of a task is the median value of the computation costs of the task on the processors. In this algorithm upward rank calculation does not consider the communication costs.

In mapping heuristic (MH) [3], the communication cost of a task on a processor is computed by the number of instructions to be executed in the task divided by the speed of processor. The algorithm uses static upward ranks to assign priorities.

The Levelized-Min Time (LMT) algorithm [6] is a two phase algorithm [8], The first phase groups the tasks that can be executed in parallel using level attribute. The second phase assigns each task to the fastest available processor. A task in lower level has higher priority than a task in a higher level. Within the same level, the task with the highest computation cost has the highest priority. Each task is assigned to a processor that minimizes the sum of the task's computation cost and the total communication cost with tasks in the previous levels.

These results can be modified by applying a new approach to calculate the rank of tasks. With the new upper rank and lower rank, the minimum makespan is achieved as compare to previous algorithms like DLS, MH, LMT and HEFT.

3. NEW HETROGENEOUS EARLIEST-FINISH TIME (NHEFT) ALGORITHM

In this proposed algorithm, tasks are ordered by scheduling priorities that are based on top ranking and down ranking. The top ranking of a task n_i is recursively defined as

$$R_u(n_i) = max_{n_j \in suc(n_i)}\{\bar{C}_{i,j} + \bar{w}_j + R_u(n_j)\} \quad (8)$$

where $\bar{C}_{i,j}$ is average communication cost between node n_i and n_j. The *suc(n_i)* is the set of immediate successors of task n_i. The $\bar{w}_j$ is the average communication cost of taskn_j. For exit task, top rank value is equal to

$$R_u(n_{exit}) = 0 \quad (9)$$

Top rank computed recursively by traversing the task group upward. Similarly, down rank of a task n_i is recursively define as

$$R_d(n_i) = \bar{w}_i + max_{n_j \in pred(n_i)}\{\bar{C}_{j,i} + R_d(n_j)\} \quad (10)$$

Where *pred(n_i)* is the set of immediate predecessor of task n_i. Down ranks are calculated recursively by traversing the task graph downward starting from entry task. The down rank of entry task is equal to

$$R_d(n_{entry}) = \bar{w}_{entry} \quad (11)$$

This algorithm has two phases, first is task prioritizing phase that calculate priority of all tasks. Second phase is processor selection phase, in which scheduling each selected task on its best processor. In task prioritizing phase, priority of each task is sum of its top rank and down rank, which is based on mean computation and mean communication costs. Arrange all tasks in a queue in decreasing order of its priorities. A tie-breaking is done randomly.

1) set the mean comp. cost of tasks and mean comm. cost of edges
2) compute sum of R_u and R_d of all tasks by traversing graph
3) sort tasks in a scheduling queue by decreasing order of tasks
4) ***while*** there are unscheduled tasks in the queue
5) ***do*** select first task, n_i from queue
6) ***for*** (each processor p_k in the processor set do compute *EFT* (n_i ,p_k) value using insertion based policy)
7) assign task n_i to processor p_j that minimize *EFT* of task n_i .
8) end while

This algorithm has insertion based policy as defined in HEFT algorithm, which considers the possible insertion of NHEFT Algorithm

a task in an earliest idle time slot between already schedule tasks on a processor. In the processor selection phase, the search of an appropriate idle time slot of a task n_i on processor p_j, i.e. the time when all input data of task n_i that were sent by n_i's immediate predecessor task have arrived at processor p_j. The search continue until finding the first idle time slot that is capable of holding the computation cost of task n_i. The NHEFT algorithm has an O(e × q) time complexity for e edges and q processors. For a dense graph, number of edges replace by v^2. Thus for a dense graph time complexity is O(v^2 × q), where v is a total number of tasks.

4. EXPERIMENTAL RESULTS

An example given in Figure 1 that presents a scheduling problem in Directed acyclic graph *(DAG)* form. Table 1 represent a computation cost matrix *W* of order *v* × *q*. The top rank and down rank for each tasks are given in Table 2. The task schedules obtained by NHEFT algorithm for this DAG present in Figure 2.

The first phase of algorithm is task prioritizing, for this calculate the top rank and down rank from (8) and (10). Because initially take the top rank of exit task is zero and down rank of entry task is mean computation cost of entry task, then R_u and R_d is calculated by recursively. A complete list is given in Table 2. Now sort the task in decreasing order of rank (R_u + R_d) and tie-breaking solve by randomly chosen. For given example scheduling queue is as [n_1 , n_2 , n_9 , n_{10} , n_3 , n_7 , n_8 , n_4, n_5 , n_6].

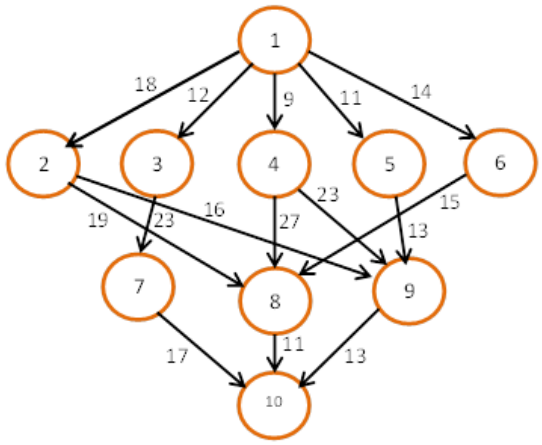

Fig 1: DAG form of a task scheduling problem

Table 1: Computation Cost Matrix

	P1	P2	P3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

Table 2: Ranks for given directed acyclic graph

	R_u	R_d	R_u + R_d
n_1	94.8	13	107.8
n_2	60.2	47.6	107.8
n_3	67.2	39.3	106.5
n_4	67.2	34.6	101.8
n_5	57.2	35.6	92.8
n_6	50.6	39.6	90.2
n_7	31.6	73.3	104.9
n_8	25.6	76.6	102.2
n_9	27.6	80.2	107.8
n_{10}	0	107.8	107.8

For the second phase of algorithm, select the unscheduled tasks from scheduling queue one by one. In our example first select n_1. Then compute *EFT* of this task for each processor by (6), take minimum *EFT* for this task and schedule it that processor.

For given example

$$EFT\ (n_1\ ,p_1) = 14$$

$$EFT\ (n_1\ ,p_2) = 16$$

$$EFT\ (n_1\ ,p_3) = 9$$

So select here n_1 on p_3 , then recursively execute algorithm for all unscheduled tasks from scheduling queue. This example takes schedule length 76. This is better than other algorithms like HEFT, DLS, MH and LMT.

The NHEFT algorithm is applied on five standard examples given by Siegel [9]. It can be easily seen that in Table 3 that scheduling length for NHEFT algorithm is better than other algorithms like HEFT, DLS, MH and LMT. A graph representation of these examples is shown in Figure 3.

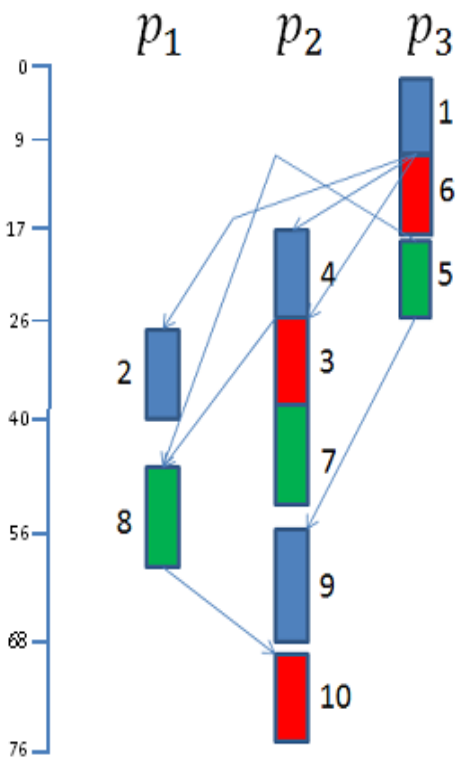

Fig 2: Tasks schedule length for given DAG

Table 3: Comparison with various algorithms

	HEFTT	DLS	MH	LMT	NHEFT
Makespan (Ex-1)	80	91	91	95	76
Makespan (Ex-2)	71	85	81	86	67
Makespan (Ex-3)	92	101	102	110	92
Makespan (Ex-4)	136	152	150	150	133
Makespan (Ex-5)	106	121	129	139	106

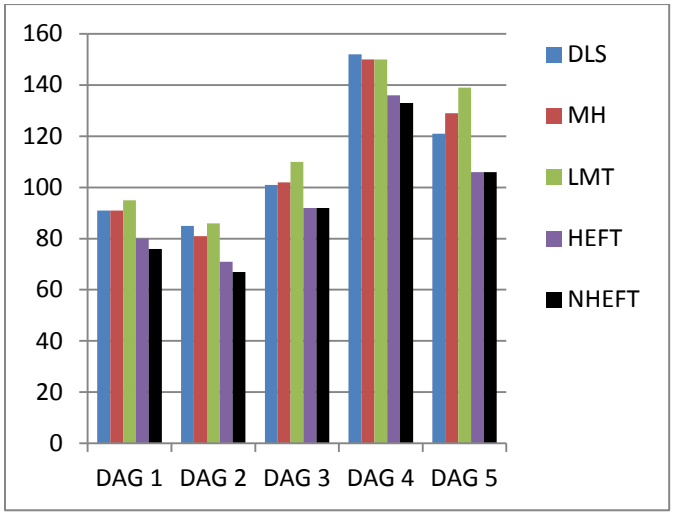

Fig 3: Comparison of HNEFT with various algorithms

5. CONCLUSION

The paper presented a new algorithm called NHEFT for scheduling graphs onto a system of heterogeneous processors. Experimental work shows that the NHEFT algorithm significantly outperformed the other algorithms like MH, DLS, LMT and HEFT. Because of its robust performance and low running time, the NHEFT algorithm is a viable solution for the DAG scheduling problem on heterogeneous systems. The NHEFT algorithm can be extended in future for rescheduling task in response to changes in processor and network loads. Although given algorithm assume a fully connected network. It is also extend this algorithm for arbitrary-connected networks.

6. REFERENCES

[1] Foster I and Kesselman C (editors), 1999, The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA.

[2] Braun R, Siegel H, Beck N, Boloni L, Maheswaran M, Reuther A, Robertson J, Theys M, Yao B, Hensgen D and Freund R, 2001, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, International Journal of Parallel and Distributed Computing, Vol.61(6): 810-837.

[3] Maheswaran M, Ali S, Siegel H.J, Hensgen D. and Freund R. F,1999, Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems, International Journal of Parallel and Distributed Computing,Vol. 59(2):107-131.

[4] Casanova H, Legrand A, Zagorodnov D and Berman F,(2000), Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, In. Proc. Of the 9th heterogeneous Computing Workshop: 349-363, Cancun,Mexico.

[5] Stone H S,1977, Multiprocessor Scheduling with the aid of network flow algorithms, IEEE Trans. Software Eng. 3: 85-93.

[6] Stone H S, Bukhara S H,1978, Control of distributed Processes, Computer: 97-106.

[7] M. M. Eshaghian, ed., 1996, Heterogeneous Computing, Artech House, Norwood, MA.

[8] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L Wang, 1993, Heterogeneous computing: Challenges and opportunities, IEEE Computer, Vol. 26, No. 6, pp. 18-27.

[9] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, 1996, Heterogeneous computing, in Parallel and Distributed Computing Handbook, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, pp. 725-761.

[10] H. J. Siegel, H. G. Dietz, and J. K. Antonio, 1997, Software support for heterogeneous computing, in The Computer Science and Engineering Handbook, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, pp. 1886-1909.

[11]G.C. Sih and E. A.Lee, 1994, A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures, IEEE Trans. Parallel and Distributed System vol. 5, no. 2, pp. 113-120.

[12] H. El-Rewini and T.G. Lewis, 1999, Scheduling Parallel Program Tasks onto Arbitrary Target Machines, Journal of Parallel and Distributed Computing, vol. 9, pp. 138-153.

[13] M. Iverson, F. Ozguner, and G. Follen, 1995, Parallelizing Existing Applications in a Distributed Hetrogeneous Environment", Proc. Hetrogeneous Computing Workshop, pp 93-100.

[14] Topcuoglu, H. Hariri, S. Min-You Wu, 2002, Performance effective and low complexity task scheduling for heterogeneous computing" IEEE Trans. Parallel and Distributed System vol. 13, no. 3, pp. 260-274.