Grenze International Journal of Computer Theory and Engineering , Special issue

Heterogeneous Laxity-based Improved Task Scheduling for DAG-based Heterogeneous Computing

Rashmi Basavaraddi[1] and Arshiya Sultana[2]
[1]rashmihb123@gmail.com
[2]arshiyasltn@gmail.com

***Abstract*—In heterogeneous computing taskmapping is necessary and has drawn major attention. Task scheduling algorithms at present are less efficient. This generally consists of two phases where first is pritorization and second is task assigning. In this paper a Heterogeneous laxity based improved task scheduling algorithm is proposed. This method of scheduling is represented by directed acyclic graph (DAG). This combines the method Heterogeneous Laxity Based Scheduling (HLBS) and Heterogeneous Scheduling with Improved Task Priority (HSIP) to improve the task scheduling efficiency. Here in this paper we come across with m-processers and n-tasks and we are concentrating on ideal time slot of any m-processers to schedule any n-tasks with shorter laxity and execution time to complete the task. To address the performance issue we are considering three parameters like makespan, Scheduling Length Ratio (SLR) and failure ratio.**

***Index Terms*— Directed Acyclic Graph (DAG), HLBS, HSIP, Makespan, SLR and Failure Ratio.**

I. Introduction

The availability of a network of processors makes a cost effective utilization of underlying parallelism for application like weather modeling, image processing, real-time and distributed database systems. A well recognized strategy in efficient execution of a huge application on a heterogeneous computing environment is that partition it into numerous independent tasks and plan such tasks over a set of available processors.
The task ranking algorithm divide the task into more number of tasks and an abstract replica of such a partitioned application can be represented using the graph. Each task in the DAG corresponds to a chain of operation and the directed edge represents the rank constraints between the tasks. Each task will be executed on a processor and in the directed edge shows transfer of relevant data from one processor to other. Task scheduling can be performed at compile instance or at run time. This includes execution times of tasks on unlike processors, the data size of the communication between the tasks and the task dependencies are known apriori. The objective of task scheduling is to map the tasks on the processors and order their execution so that task rank requirements are satisfied and a minimum overall completion time is obtained. Power performance optimization includes minimizing the present efficiency gap in processing throughput and power utilization. Power efficiency, a fresh focus for common purpose computing, has been a important technology driver in the application of phone and embedded areas for few time.
Different methods are done for optimizing the solution for task scheduling difficulty. The tasks are scheduled

in order to lessen the idle time on the machines and communication overhead. The scheduling of a DAG on the topological network not only involves the mapping of task nodes on the processor but also includes the mapping of the edges on the links of the network. When all the inputs are accessible, that is when the parent nodes have successfully executed, only then the task is executed. Each node has its own associated computation rate that designates the execution time of every node on the processor. In the case of homogeneous processor, the price is the similar for every kind of processor.

II. Related Work

The methods on the list scheduling are used in graph to assign the ranks to tasks and for listing these tasks according to priorities in a downward order. The task of high rank is given preference over the one having low priority by the aid of priority policy. A task can be assigned to any of the processors meanwhile a ready list is maintained for assigning the priorities.

Prof. Guoqi Xie et.al [1] proposed a fully heterogeneous task scheduling algorithm to address the above problems. The fundamentals of DAG model and corresponding algorithms are investigated. New concepts called Heterogeneous Upward Rank Value (HURV) and Heterogeneous Priority Rank Value are defined. An algorithm called Heterogeneous Select Value is proposed in paper. Both benchmark and extensive experimental evaluation demonstrate the significant improvements in proposed algorithm.

Mehdi Akbari et. al [2] proposed a task scheduling algorithm on heterogeneous computing systems using Efficient State Space Search Genetic Algorithm (ESSSGA). The basic idea of this approach is to exploit the advantages of heuristic-based algorithms to reduce space search and the time needed to find good solutions. The proposed algorithm uses a novel list scheduling heuristic-based algorithm while using a heuristic-based earliest finish time approach to search for a solution for the task-to-processor mapping. Here the results gives that makespan is better achieved.

Guan Wang, and Yuxin Wang [3] Heterogeneous Scheduling with Improved Task Priority (HSIP). Here the algorithm has two phases: a task prioritizing stage and it is to calculate task priorities and a processor selection stage for choosing the best processor to execute the current task. In this task duplication selection policy is used and it will consume memory.

Yuhei Suzuki and Takuya Azumi [4] presented the HLBS Algorithm, HLBS computes the rest time until deadlines, known as laxity, and preferentially assign a task with shorter laxity to the processor. This enables scheduling of multiple deadlines to reduce deadline miss rate. But makespan is higher in HLBS when compared to other algorithms hence in our paper using HLBS and HSIP Algorithm we reduced the makespan.

III. Methodology

In heterogeneous computing, task scheduling is major issue as different kinds of processors are used. Different kinds of techniques though were introduced but still efficient use of resources and time complexity still remains as it is and deadlines are missed many times. So there is need of efficient of scheduling algorithm that improves idle slot and reduces the deadline and time complexity. The overall proposed architecture of the model is as shown in Figure 1.

The method of scheduling is represented by DAG.A DAG is a directed graph that has no cycles. It is formed by a collection of vertices and edges, where the vertices are structure-less objects that are connected in pairs by edges. In the case of a directed graph, every edge has an orientation, from 1 vertex to another vertex. A path in a directed graph can be describe by a series of edges having the property that the ending vertex of every edge in the series is the same as the starting vertex of the next edge in the sequence; a path forms a cycle if the starting vertex of its first edge equals the ending vertex of its last edge.

We identify the execution time for particular task for each m-Processor and based on the criteria such as (1) Execution time (2) Computation Time and (3) Laxity scheduling work has taken place.

Here in this paper we are combining both the HLBS (Heterogeneous Laxity-Based Scheduling) and HSIP (Heterogeneous Scheduling with Improved Task Priority) algorithms. There are two phases to integrate both the algorithm: I. Task Prioritization. II. Task Assignment.

We now introduce the graph attributes used for ranking the task priorities. An application is represented by a DAG, $G = (V, E)$, where V is the set of n tasks, E is the set of e edges between tasks, w_{ip} represents the

weight of task n_i on processor p, which is the execution cost of task n_i, c_{ij} represents the communication delay from task n_i to task n_j ,
D_i represents the deadline attributed to end noden_i, and w_i and $\bar{c}_{ij}$ represent the average of the wi and cij dependent processors, respectively.

A. Task Prioritization

The scheduler gives a priority level to each task as pre-processing. Tasks are given priority $\text{prio}_{\text{heft}}(\text{T}_\text{i})$ which is recursively defined by

$$prio_{heft}(\text{T}_i) = \overline{w}_i p + \max_{\text{T}_j \in succ(\text{T}_i)}(rank(\text{T}_j) + c_{ij}). \quad (1)$$

A set of immediate successors to node vi is given by $\text{succ}(\text{n}_\text{i})$. $\text{prio}_{\text{heft}}$is computed recursively by traversing the task graph from end to entry node. The tasks are assigned a priority in ascending order of$\text{prio}_{\text{heft}}$. As shown in Equation 1, the HEFT algorithm computes the priority as the sum of the execution and communication times through a path. If the task n_i corresponds to an end node, $\text{prio}_{\text{heft}}$ is equal to

$$\text{rank}_{\text{heft}}(\text{T}_\text{i}) = \overline{\text{w}_\text{ı}\text{p}}. \quad (2)$$

B. Task Assignment

In the processor selection stage, according to the priority of task scheduling order, tasks are assigned to the lesser EFT processor to be executed. On the basis of the above strategy, we proposed two innovative policies, entry task duplication selection policy and idle time slots (ITS) insertion-based optimizing policy. They improve the efficiency of scheduling algorithm.
The allocation of tasks to a processor is performed using EST (earliest execution start time) and EFT (earliest execution finish time). $\text{EST}(\text{n}_\text{i}, \text{H}_\text{p})$, given by Equation 3, is an available time to start an execution of the task n_i on processor H_p and $\text{EFT}(\text{T}_\text{i}, \text{H}_\text{p})$, given by Equation 4, is the time to completion of task execution.
$\text{available}(\text{H}_\text{p})$is the earliest time at which processor H_p is ready for task execution. The set of immediate predecessor tasks of task $\text{n}_\text{i}$is represented by $\text{pred}(\text{n}_\text{i})$.

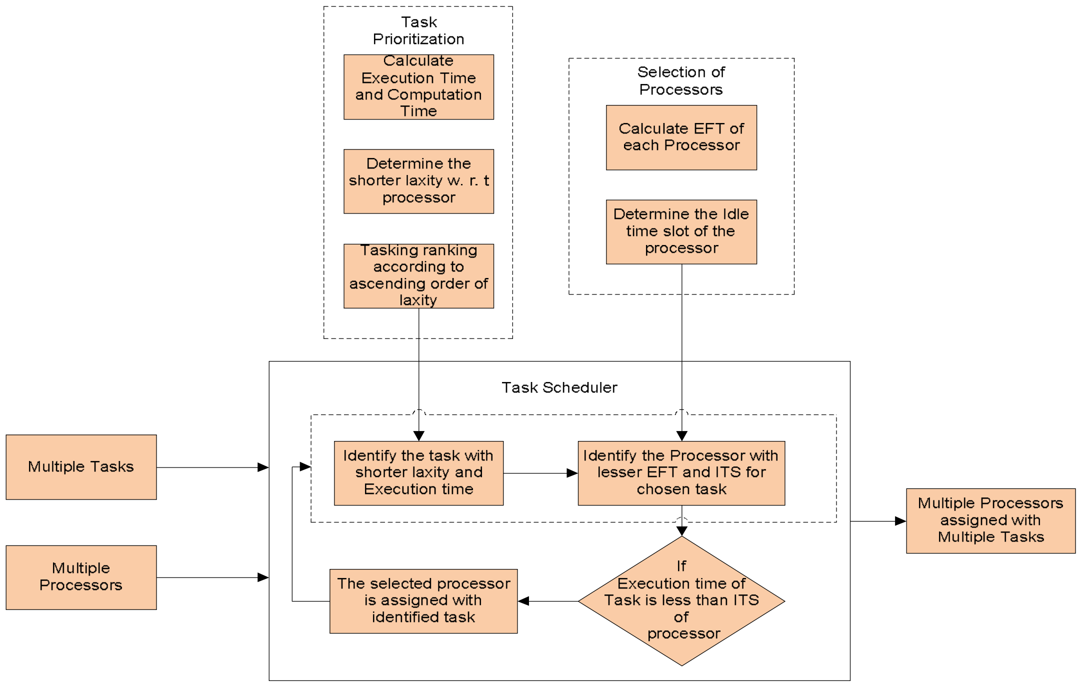
Task Prioritization
Calculate Execution Time and Computation Time
Determine the shorter laxity w. r. t processor
Tasking ranking according to ascending order of laxity
Selection of Processors
Calculate EFT of each Processor
Determine the Idle time slot of the processor
Task Scheduler
Multiple Tasks
Multiple Processors
Identify the task with shorter laxity and Execution time
Identify the Processor with lesser EFT and ITS for chosen task
If Execution time of Task is less than ITS of processor
The selected processor is assigned with identified task
Multiple Processors assigned with Multiple Tasks

Figure 1: Block Diagram of Proposed System

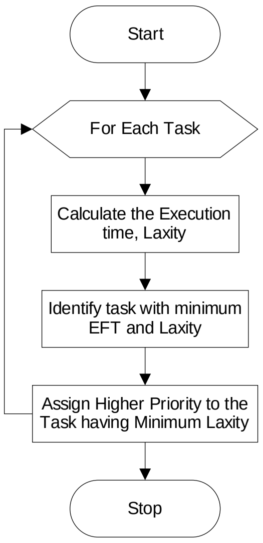
Start
For Each Task
Calculate the Execution time, Laxity
Identify task with minimum EFT and Laxity
Assign Higher Priority to the Task having Minimum Laxity
Stop

Figure 2: Flow Chart of task Prioritization

$$\mathrm{EFT}(T_i, H_p) = \max(\mathrm{available}(H_p) \\ \max_{n_j \in pred(n_i)}(\mathrm{EFT}(n_j, \mathrm{host}(T_j)) + c_{ij})) \qquad (3)$$

$$\mathrm{EFT}(n_i, H_p) = w_i p + \mathrm{EST}(n_i, H_p) \qquad (4)$$

Figure 5 shows the scheduling results from Figure 4 using HEFT algorithm, where the execution and communication costs are taken from Table I. The vertical axis shows the processor, and the horizontal axis shows time. The scheduling order of the tasks with respect to the HEFT algorithms is [T0, T4, T5, T1, T8, T11, T9, T12, T2, T6, T14, T3, T20, T15, T7, T18, T16, T19, T13, T17, and T10]. As seen from Figure 2[a], all end nodes are concentrated near the finish time of all tasks and task T13 fails to meet the deadline of 144. In HEFT, a deadline miss usually occurs because an end node via few nodes does not preferentially execute. The main measure of the performance of an algorithm is makespan, which is given by Formula 5:

$$\mathrm{makespan} = \max(\mathrm{AFT}(H_p, n_{end})) \qquad (5)$$

The execution finish time of a task that is assigned at the end of each processor is known as the AFT (actual execution finishes time) and is given as $\mathrm{AFT}(H_p, n_{end})$. Representing the processor as $H_p$and the last of the tasks that are assigned to the processor H_p as $n_i$makespan derives the finish time of the system.

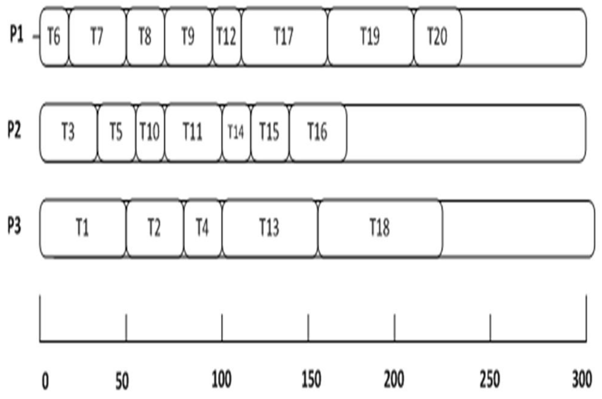

Figure 4: Scheduling results of Figure 4 with HEFT and HLBS algorithm

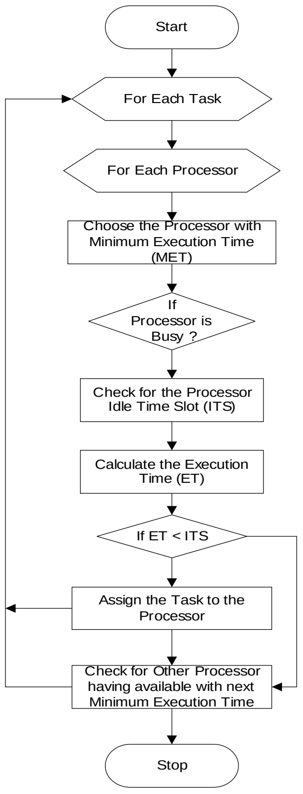
Start
For Each Task
For Each Processor
Choose the Processor with Minimum Execution Time (MET)
If Processor is Busy ?
Check for the Processor Idle Time Slot (ITS)
Calculate the Execution Time (ET)
If ET < ITS
Assign the Task to the Processor
Check for Other Processor having available with next Minimum Execution Time
Stop

Figure 3: Flow Chart of Selection of Process

In HLBS, the priority given to task v_i represents laxity and corresponds to the shortest time until the deadline, $laxity(v_i)$, given by Equations 6 and 7.

- If v_i corresponds to the end node:

$$laxity(v_i) = D_i - \overline{w_i p} \quad (6)$$

- If v_i does not correspond to the end node:

$$\mathrm{laxity}(v_i) = \min_{v_j \in succ(v_i)} (\mathrm{laxity}(v_j) - \overline{c_{ij}}) - \overline{w_i}p \quad (7)$$

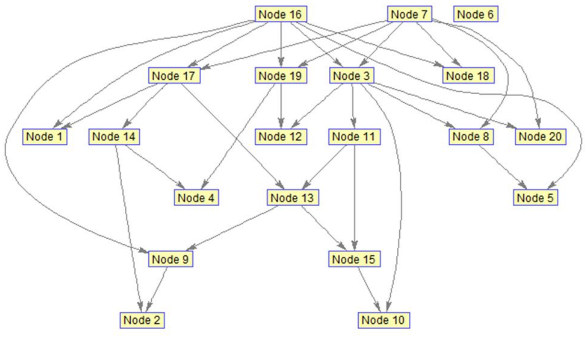

Figure 5: DAG Input Graph

Laxity is recursively computed by traversing the task graph from end node to entry node. The tasks are assigned high priority in ascending order of laxity. It means that the task could not afford the deadline has been preferentially assigned by the processor. This ranking algorithm addressed Richard's Anomalies which is a problem for fixed priority to increase makespan when increasing the number of processors.

IV. RESULT

In this section, we present a comparative evaluation of our algorithms and those of previous work using a randomly generating DAG tool.

TABLE I: PARAMETER OF EACH NODE

Task No.	Execution Cost P1	Execution Cost P2	Execution Cost P3	Communication Cost
T1	26	24	17	20
T2	19	12	9	2
T3	11	8	23	26
T4	16	13	6	29
T5	13	3	21	21
T6	3	4	6	23
T7	8	29	12	23
T8	4	29	19	12
T9	6	18	24	20
T10	8	2	3	6
T11	13	8	28	22
T12	2	11	24	1
T13	28	25	15	9
T14	29	1	14	2
T15	15	2	14	3
T16	15	6	10	25
T17	11	20	16	21
T18	28	22	16	10
T19	12	20	25	29
T20	4	14	24	2

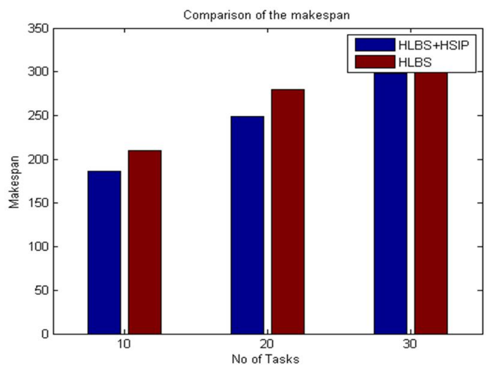

Figure 6: Comparison of Makespan

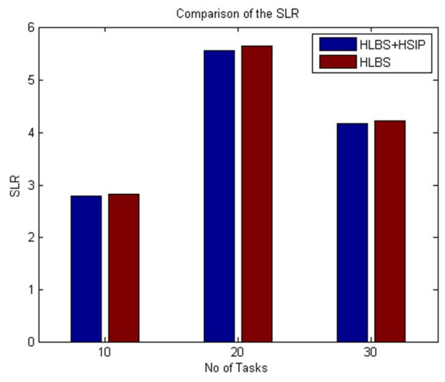

Figure 7: Comparison of SLR

Comparison with previous work as shown in Figure 6, the average makespan value of HLBS and HLBS HSIP combined. This difference is equivalent to one task. As the time for each of the number of tasks increased, the difference in the makespan value become larger because tasks on the critical path are not necessarily given a high priority in HLBS, as the main purpose is to ensure that deadline constraints are met.

Scheduling Length Ratio (SLR): SLR normalizes the scheduling length (makespan) to a lower bound. The task scheduling algorithm that gives the lowest SLR is considered the bestperformance algorithm. SLR is defined as the ratio of makespan to the sum of the computation time on the critical path (CP) and is calculated.

$$\mathrm{SLR} = \frac{\mathrm{makespan}}{\sum_{v_i \in CP_{MIN}} \min_{p_j \in Q}\{w_i\}} \qquad (8)$$

Average SLR values over several task graphs were used in our experiments.

Failure ratio (FR): We evaluated the performance in terms of a Failure ratio, defined as the ratio of the number of unschedulable task sets to the total number of task sets attempted. The failure ratio was defined by

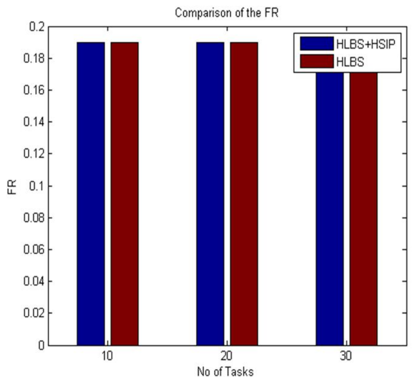

Figure 8: Comparison of Failure Rate

$$FR = \frac{the\ number\ of\ failure\ task\ sets}{the\ number\ of\ scheduled\ task\ sets} \quad (100)$$

Where the number of failure task sets is the set of tasks including any tasks that fail to meet deadline constraints on the end nodes.

V. Conclusion

In this paper we have proposed HLIS which combination of HLBS and HSIP. This approach helps us in scheduling task in heterogeneous efficiently. This helps us in executing the task soon, with shorter scheduling length. The combination of two approaches helps us improve in better way.

References

[1] Guoqi Xie, Renfa Li,Xiongren Xiao and Yuekun Chen, “A High-performance DAG Task Scheduling Algorithm for Heterogeneous Networked Embedded Systems” IEEE, Vol.7, Issue 3, 2014.

[2] Mehdi Akbari and Hassan Rashidi, “An Efficient Algorithm For Compile-Time Task Scheduling Problem On Heterogeneous Computing Systems”, Vol.7, Issue 1, 2015.

[3] Samia Ijaz, Ehsan Ullah Munir, Waqas Anwar, and Wasi Nasir, “Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment”, Vol. 10, Issue 5, 2013.

[4] Sukhjit Singh and Nirmal Kaur, “A Heterogeneous Static Hierarchical Expected Completion Time Based Scheduling Algorithm in Multiprocessor System”, Vol. 3, Issue 2, 2016.

[5] GuanWang, YuxinWang, Hui Liu, and He Guo, “HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing”, Hindwai, 2016.

[6] Prerit Chawda and Partha Sarathi Chakraborty, “An Improved Min-Min Task Scheduling Algorithm for Load Balancing in Cloud Computing”, Vol. 4, Issue 4, 2016.

[7] Anum Masood, Ehsan Ullah Munir, M. Mustafa Rafique and Samee U. Khan, “HETS: Heterogeneous Edge and Task Scheduling Algorithm for Heterogeneous Computing Systems”, Vol.3, Issue 32, 2014.

[8] Weiwei Lin, Wentai Wu, and James Z. Wang, “A Heuristic Task Scheduling Algorithm for Heterogeneous Virtual Clusters”, Hindawi, 2016.