Chapter 2

Preliminaries and Literature Review

This chapter first presents some preliminary concepts related to the research problem addressed in the thesis and then discusses some related work regarding task scheduling algorithms for distributed computing in multiprocessor environments. Here, we attempt to provide a comprehensive understanding of the existing task scheduling algorithms by comparing the concepts used in the algorithms.

2.1 Preliminaries

The following assumptions, definitions, concepts, etc. are being given here under for the sake of completeness as associated terms could be referred to, in the rest of thesis.

2.1.1 Assumptions

The following are certain assumptions that are used by the scheduling system model of our proposed algorithms in this chapter:

- The multiprocessor environment for distributed computing is composed of the unlimited number of homogeneous processors.
- The processors are loosely-coupled and communicate each other through message passing. Also, the processors are fully connected with each other and there is non-zero communication overhead between any two processors.
- A processor can execute a single task at any time.
- Each task is computed by a single processor such that the task can′t be preempted, once a processor starts executing it.
- The tasks used in this model are dependent tasks.
- After finishing the execution, the task transmits output data to all immediate successor tasks simultaneously.
- The computation and communication operations can be performed concurrently by a node in the system.
- No duplication of the task is allowed and each task of a cluster must execute on the same processor.

2.1.2 Definitions

The following definitions are being provided for the purpose of clarity with respect to their usage in rest of the work.

Definition 2.1. **($pred(T_i)$).** It represents the set of immediate predecessors of task T_i in a given task graph. A task is called an entry task, T_{entry}, if it doesn′t have any predecessor task. If a task graph consists of many entry tasks, a dummy entry task with zero execution time and edges with zero communication time can be added to the task graph.

Definition 2.2. **($succ(T_i)$).** It represents the set of immediate successors of task T_i in a given task graph. A task is called an exit task, T_{exit}, if it doesn′t have any successor task. Similar to the entry task, if a task graph consists of many exit tasks, a dummy exit task with zero execution time and edges with zero communication

time from current many exit tasks to this dummy task can be added to the task graph.

The uniqueness of T_{entry} and T_{exit} is not presumed in this work.

Definition 2.3. **($AFT(T_i)$).** It is the Actual Finish Time of a task T_i and is defined as the point of time where the execution of task T_i is completed by some assigned processor.

Definition 2.4. (makespan). makespan or schedule length represents the completion time of the exit task in the scheduled task graph and is defined by

$$makespan = max\Big\{AFT(T_{exit})\Big\} \tag{2.1}$$

where $AFT(T_{exit})$ denotes the Actual Finish Time of the exit task. If a task graph has many exit tasks and no redundant task is added, the makespan is computed as the maximum AFT of all exit tasks. Schedule length, makespan and parallel execution time are used interchangeably in this work.

Definition 2.5. (Critical Path). Critical Path (CP) of a task graph is defined as the longest path from the entry task to the exit task in the graph where the length of a path in the task graph is the sum of the execution times of its tasks and communication times of its edges.

Definition 2.6. **($BL(T_i)$).** It denotes the Bottom Level of a task T_i and is defined as the length of the longest path from the task T_i to the exit task in the task graph. It is obtained as given in following equation:

$$BL(T_i) = ET(T_i) + \max_{T_j \in succ(T_i)} \Big\{BL(T_j) + CT(e_{i,j})\Big\} \tag{2.2}$$

$CT(e_{i,j})$ becomes zero, when T_i and T_j are in a same cluster. For the exit task,

$$BL(T_i) = ET(T_i) \tag{2.3}$$

Definition 2.7. **($TL(T_i)$).** It denotes the Top Level of a task T_i and is defined as the length of the longest path from the entry task to the task T_i in the task

graph excluding the execution time of task itself. It is obtained as given in following equation:

$$TL(T_i) = \max_{T_j \in pred(T_i)} \left\{ TL(T_j) + ET(T_j) + CT(e_{j,i}) \right\} \tag{2.4}$$

$CT(e_{i,j})$ becomes zero, when T_i and T_j are in a same cluster. For the entry task,

$$TL(T_i) = 0 \tag{2.5}$$

Definition 2.8. **($EST(T_i)$).** It denotes the Earliest Start Time of a task T_i and is defined as the earliest time at which T_i can start its execution, after all its predecessor tasks are finished their execution and associated dependencies have been transferred to T_i. It is obtained as given in following equation:

$$EST(T_i) = \max_{T_j \in pred(T_i)} \left\{ EFT(T_j) + CT(e_{j,i}) \right\} \tag{2.6}$$

$CT(e_{j,i})$ becomes zero, when T_j and T_i are in a same cluster. For the entry task,

$$EST(T_{entry}) = 0 \tag{2.7}$$

Definition 2.9. **($EFT(T_i)$).** It denotes the Earliest Finish Time of a task T_i and is defined as the earliest time where the computation of task T_i is completed. It is obtained as given in following equation:

$$EFT(T_i) = EST(T_i) + ET(T_i) \tag{2.8}$$

We assume that all processors are homogeneous and initially available. Thus, the AFT and EFT of a task will be equal.

Definition 2.10. **($LFT(T_i)$).** It denotes the Latest Finish Time of a task T_i and is defined as the latest time where the computation of task T_i is completed. It is obtained as given in following equation:

$$LFT(T_i) = \min_{T_j \in succ(T_i)} \left\{ EST(T_j) - CT(e_{i,j}) \right\} \tag{2.9}$$

Definition 2.11. ($slack(T_i)$). It denotes the slack of a task T_i and is defined as the amount of time the task T_i can be delayed without delaying the makespan. It is obtained as given in following equation:

$$slack(T_i) = LFT(T_i) - EST(T_i) - ET(T_i) \tag{2.10}$$

2.1.3 Task Graph

An application is a group of interrelated tasks modeled with a Task Graph (TG) that is a Directed Acyclic Graph (DAG), $G = (V, E)$, where V denotes the set of nodes and each node represents a task that consists of instructions which must be computed on the same machine. E denotes the set of communication edges between tasks. In the task graph, each task $T_i \in V$ is associated with its execution time or computation time, denoted by $ET(T_i)$ and each edge $e_{i,j} \in E$ from $T_i \in V$ to $T_j \in V$ is associated with its communication time, denoted by $CT(e_{i,j})$. Each edge $e_{i,j}$ represents the precedence constraint between tasks T_i and T_j such that task T_j cannot begin its execution until task T_i completes its execution. When two tasks belong to the same cluster, the communication time between them becomes zero. An instance of a task graph containing 15 tasks, for a multiprocessor system having homogeneous processors, is shown in Fig. 2.1 and is taken from literature [26]. Another instance of a task graph containing 10 tasks, for a multiprocessor system having heterogeneous processors, is shown in Fig. 2.2. The corresponding matrix showing execution times of the tasks on each processor for a 3-processor system for this task graph is also given in Fig. 2.2. In Fig. 2.1, a node denotes a task, and the value inside the node represents the execution time of that task. The value written with a directed edge between tasks T_i and T_j denotes the communication time from T_i to T_j. For example, the execution time of T_7 is 4, and the communication time from T_7 to T_{11} is 5. T_1 is immediate predecessor of T_3, T_4, and T_5. T_{11} is immediate successor of T_7 and T_{10}. T_0 is an entry task, and T_4, T_8, and T_{14} are exit tasks.

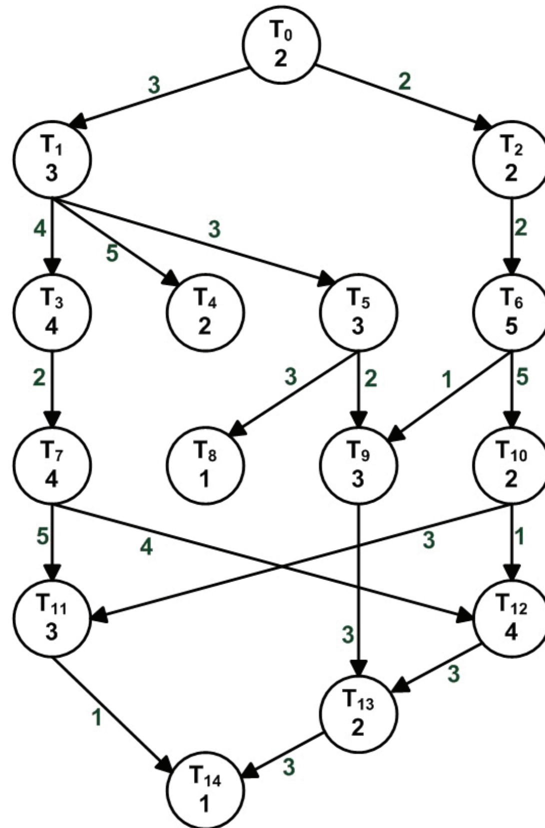
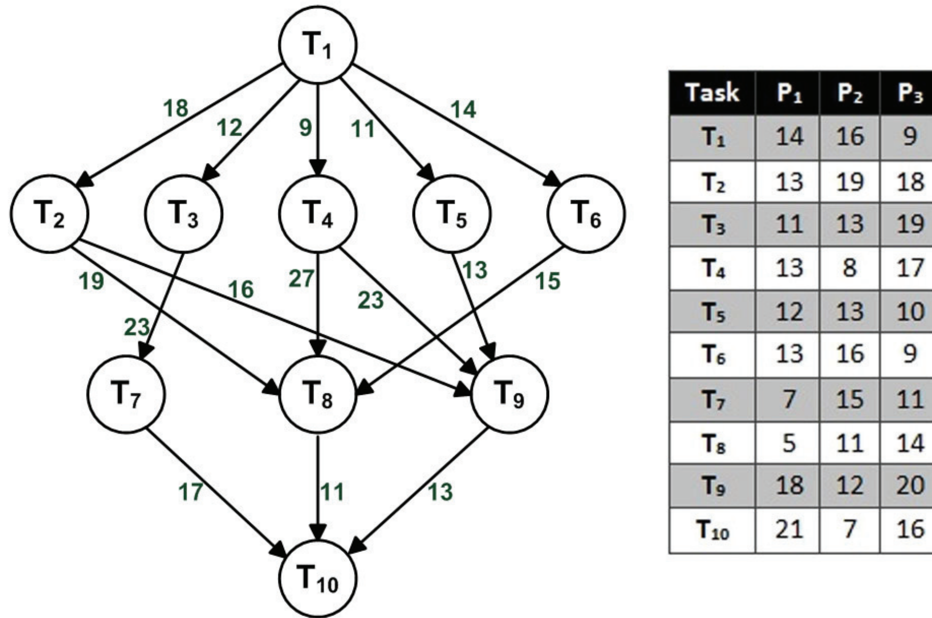


FIGURE 2.1: A sample task graph containing 15 tasks for a multiprocessor system having homogeneous processors.

2.1.4 Task Scheduling Problem

The purpose of the task scheduling problem is to schedule tasks of an application onto processors such that the precedence constraints are satisfied and the makespan is minimized. When each task of the given application is scheduled, the makespan will be the AFT of the end task, as defined by Eq. 2.1. For a given task graph, precedence constraint means that a task can′t starts its execution until its predecessor tasks have completed their execution and all essential information has been communicated to it.



Task	P1	P2	P3
T1	14	16	9
T2	13	19	18
T3	11	13	19
T4	13	8	17
T5	12	13	10
T6	13	16	9
T7	7	15	11
T8	5	11	14
T9	18	12	20
T10	21	7	16

FIGURE 2.2: A sample task graph containing 10 tasks for a multiprocessor system having heterogeneous processors and the corresponding matrix showing execution times of the tasks on each processor for a 3- processor system.

2.1.5 Real-world Application Graphs

We use different real-world application graphs in this thesis to perform experiments regarding task scheduling algorithms. For sake of completeness, we are, here giving a brief description of these application graphs.

2.1.5.1 Gaussian Elimination

In Gaussian Elimination application, the size of the DAG is characterized by a new parameter called matrix size m and is determined as $(m^2 + m - 2)/2$. For the experiments, m is used instead of DAG size [10, 11, 39]. A Gaussian Elimination DAG for matrix size 5 is shown in Fig. 2.3.

2.1.5.2 Fast Fourier Transform

In Fast Fourier Transform (FFT) application, the DAG size is characterized by a new parameter called input points and is determined as $(2m-1)+mlog_2m$ where m

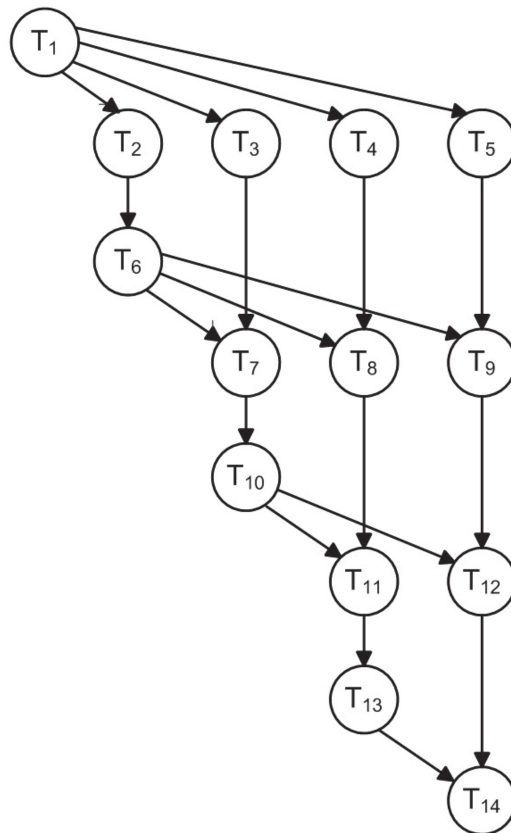


FIGURE 2.3: A Gaussian Elimination DAG for matrix size 5.

is the input points and $m = 2^k$ for some integer k. The first part in the expression gives a number of recursive call tasks, and the second part gives the number of butterfly operation tasks [10, 40]. An FFT task graph for four input points is shown in Fig. 2.4. In this figure, the tasks above dashed line are recursive call tasks, and the tasks below dashed line are butterfly tasks.

2.1.5.3 Systolic Array

In systolic array graphs , the number of nodes is n^2 and number of edges are $2n(n+1)$ where n is the number of nodes on a path from the start node to the centre node [41]. A systolic array DAG for $n = 3$ is shown in Fig. 2.5.

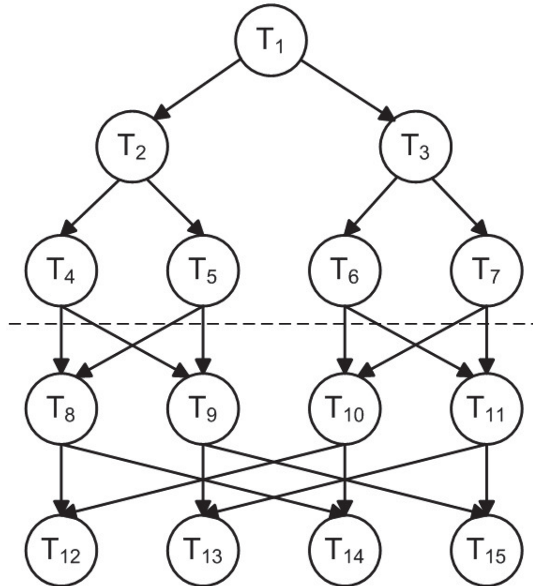


FIGURE 2.4: A Fast Fourier Transform DAG for four input points.

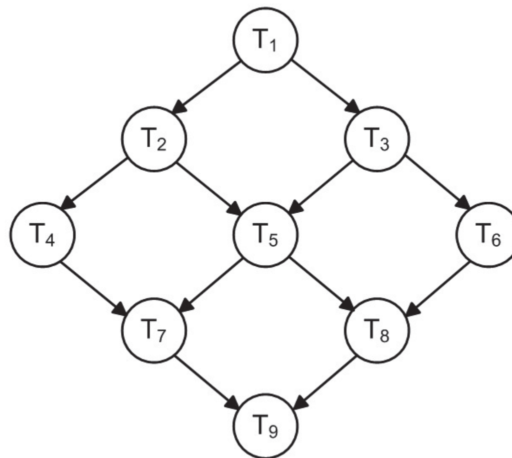


FIGURE 2.5: A systolic array DAG for n = 3.

2.1.5.4 Montage Workflow

The Montage workflow is an astronomical application created by NASA/IPAC. It is used to produce custom mosaics of the sky by stitching together multiple input images [42, 43]. A comprehensive description of this workflow is given by Juve et al. [44]. A Montage workflow graph having 25 nodes is shown in Fig. 2.6.

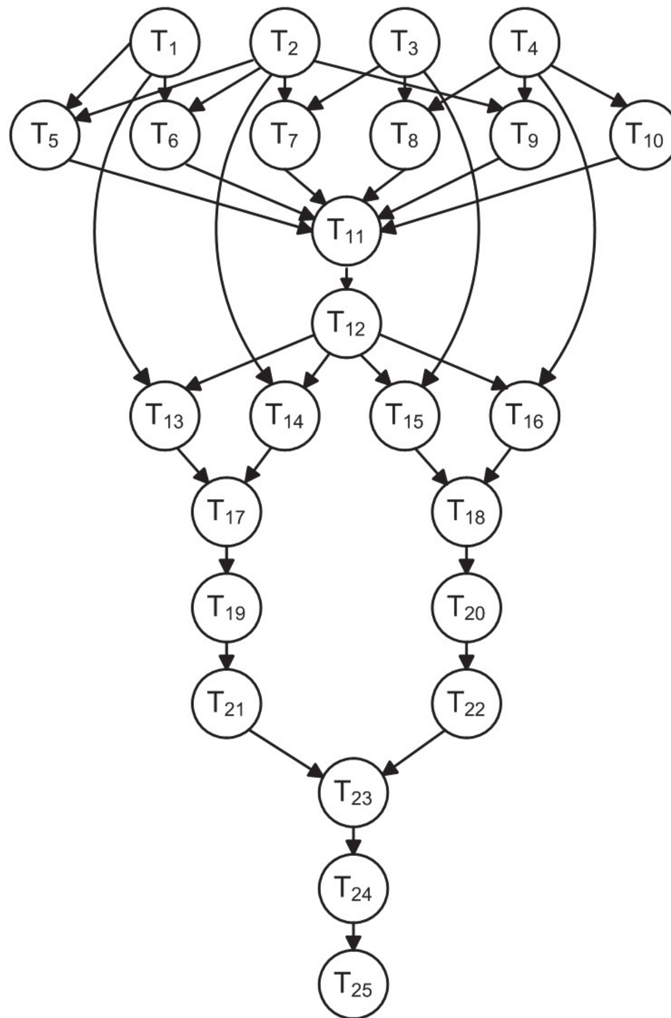


FIGURE 2.6: A Montage workflow graph having 25 nodes.

2.1.5.5 Epigenomics Workflow

The Epigenomics workflow is built by the USC Epigenome Center and the Pegasus Team. It is used to map the epigenetic state of human cells on the genome-wide scale. It is an extremely pipelined application with many pipelines operating on independent chunks of data in parallel [45]. A comprehensive description of this workflow is given by Juve et al. [44]. An Epigenomics workflow graph having 24 nodes is shown in Fig. 2.7.

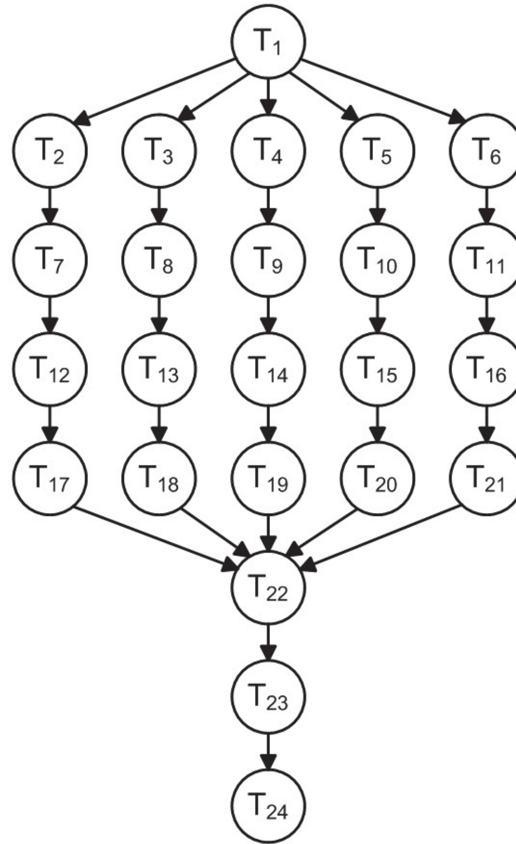


Figure 2.7: An Epigenomics workflow graph having 24 nodes.

2.2 Literature Review

Many task scheduling algorithms have been proposed in the literature. This section briefly reviews various categories of existing task scheduling algorithms such as list scheduling, duplication-based scheduling and clustering-based scheduling algorithms developed for distributed computing on multiprocessors.

2.2.1 List Scheduling Algorithms

In this section, we give a brief description regarding some known list scheduling algorithms.

LMT (Levelized Min Time) algorithm [46] is designed as a two-phase algorithm, where the first phase carries out the level sorting based on the ordering of the nodes

using their precedence constraints. The level sorting technique performs grouping of nodes which can execute in parallel such that the tasks in each level have no precedence constraints. In the second phase, the tasks are allocated level by level, using the Min-Time algorithm which attempts to allocate each task to the best processor without using precedence information.

In [10], Topcuoglu et al. proposed two algorithms: HEFT (Heterogeneous Earliest Finish Time) and CPOP (Critical Path On a Processor). Both algorithms consist of two phases: task prioritizing and processor selection. In HEFT, the task prioritizing phase prioritizes the tasks by computing their upward rank while in CPOP, the first phase computes the rank of tasks by computing the sum of their upward and downward ranks. After computing rank of tasks, both algorithms make a sorted list of tasks according to their ranks. In the processor selection phase, HEFT selects a processor for a task which gives the smallest value of earliest finish time (EFT) for that task while CPOP schedules the critical tasks on the critical path processor if the task is on the critical path. For non-critical tasks, CPOP schedules tasks on processors, as in HEFT. To compute EFT, both algorithms use insertion-based scheduling policy that attempts to include a task in the free time window between two tasks that are previously assigned to a processor, if the window is capable of scheduling that task and preserves the precedence constraints. The time complexity of both algorithms is the same, but the scheduling result of CPOP is worse.

The HCPT (Heterogeneous Critical Parent Trees) algorithm [47] divides the task graph into a set of unlisted-parent trees. The root of each unlisted-parent tree is a critical path node (CPN). It consists of two phases: listing tasks and processor assignment. In the first phase, tasks are listed in a queue in decreasing order of their ALST (Average Latest Start Time) values whereas, in the second phase, each task of the queue is assigned to a processor.

The HPS (High-Performance Task Scheduling) [48] algorithm is divided into three phases, a level sorting, task prioritization, and processor selection phase. The first phase traverses the graph from top-to-bottom to sort tasks at each level and performs grouping of independent tasks. In the second phase, priority is computed for each task using the attributes DLC (Down Link Cost), ULC (Up Link Cost), and Link Cost. The Link Cost of a task is the sum of DLC, ULC, and maximum link cost for all its direct predecessor tasks. At each level, the tasks are sorted in decreasing order

according to their link cost values. The third phase is associated with the processor selection based on minimum EFT for a task execution with Link Cost value.

The ILS (Iterative List Scheduling) algorithm [49] improves the solution iteratively after producing an initial solution with moderate quality. The primary step of ILS is same as the HEFT algorithm. During the iterative steps, the results of the previous iteration are used to compute and update the execution time of tasks and communication time of the edges in order to construct a new list. It stores and returns the best solution observed during the iterations.

The PETS (Low complexity performance effective task scheduling) algorithm [15] maintains a priority queue by sorting each task on each scheduling level according to average computation cost, data transfer cost, and rank of predecessor tasks. The algorithm calculates the EFT of each task and allocates a task in the idle time slot between already scheduled tasks on processors using insertion-based policy.

The LDCP (Longest Dynamic Critical Path) algorithm [9] consists of three phases: (i) task selection, (ii) processor selection and (iii) status update. The first phase constructs DAGPs (DAGP is a Directed Acyclic Graph that corresponds to a Processor) for all processors and identifies the selected task using the key node or the parent key node. The second phase computes the finish time of the selected task on all processors and assigns the tasks on processors according to lowest completion time. In the third phase, the status of the system is updated.

The Lookahead scheduling algorithm [50] is an improvement over the HEFT algorithm [10]. In this algorithm, the processor selection strategy used in the second phase of HEFT is modified which integrates the notion of lookahead for current task and its children and reduces the finishing time of the current task and its children. To choose a processing unit for a task t, the algorithm iterates over all available processing units and calculates the EFT of its children tasks on all processing units. Task t will be allocated on the processing unit which reduces the highest EFT for all its children which were allocated using HEFT.

The CEFT (Constrained Earliest Finish Time) algorithm [51] introduces and uses the concept of constrained critical path (CCP) that is a critical path which includes only ready tasks. A ready task is a task whose all parents have been completed. The algorithm finds all CCPs by analyzing and pruning each critical path of the graph.

A queue is maintained by picking one or many ready tasks from the longest CCP. The procedure is performed again for the next longest CCP, and so on until each task is added to the queue in a round robin fashion. Each CCP is allocated to the processing unit which yields the smallest completion time.

The PEFT (Predict Earliest Finish Time) algorithm [7] is founded on the notion of lookahead and optimistic cost table (OCT). The OCT is represented by a matrix in which the rows and columns give information about the tasks and processing units respectively. The algorithm works in two phases as like HEFT. In the initial phase, tasks priorities are decided from the sorted average of OCTs on each processing unit. In the second phase, a processing unit for a task is chosen from the outcomes of task optimistic EFT calculated from the least summation of OCTs and EFT. It also uses insertion-based scheduling policy as like HEFT.

In [52], the authors proposed an HSV (Heterogeneous Selection Value) algorithm with an accurate analysis of time on computing heterogeneity by observing the upward rank value and the earliest finish time based on the classic model. The algorithm sorts the tasks on the basis of their HPRV (Heterogeneous Priority Rank Value) value and assigns tasks to the processors according to their HSV values.

The IPEFT (Improved Predict Earliest Finish Time) algorithm [53] is an improvement over the PEFT algorithm [7] that calculates the task priority with a pessimistic cost table, implements the feature prediction with a critical node cost table, and assigns the best processor for the task that has at least one immediate successor as the critical node.

The PATS (Predict and Arrange Task Scheduling) algorithm [54] executes in two steps: EFT with level-based task scheduling and idle slot reduction. In the first step, tasks are scheduled according to their predicted EFT from the candidate task list and their dependencies. Scheduling is performed level by level starting from the top level. In the second step, the idle time slots in each processing unit are minimized.

Table 2.1 shows a comparison of some known list scheduling algorithms in terms of the concept used and the time complexity. In Table 2.1, $|E|$ and $|V|$ represent the number of edges and the number of nodes in the task graph, respectively, p represents the number of processors, s_{max} denotes the maximum number of iterations, deg_{in}

indicates the in-degree of the task graph and l represents the number of levels in the task graph.

TABLE 2.1: A comparison of some known list scheduling algorithms.

Algorithm	**Concept**	**Complexity**
LMT [46]	Level sorting	$O(\|V\|^2p^2)$
HEFT [10]	Upward rank	$O(\|V\|^2p)$
CPOP [10]	Upward and downward rank	$O(\|V\|^2p)$
HCPT [47]	Average Latest Start Time	$O(\|V\|^2p)$
HPS [48]	Link cost	$O(\|V\|+\|E\|)(p+log\|V\|)$
ILS [49]	Bottom-level	$O(s_{max}(p\|E\|+\|V\|^2))$ in the worst case
PETS [15]	Rank of predecessor task	$O(\|E\|(p+log\|V\|))$
LDCP [9]	Upward rank	$O(\|V\|^3)$
Lookahead [50]	Lookahead	$O(\|V\|^4p^3)$ in worst case
CEFT [51]	Constrained critical path	$O(\|V\|.p(\|V\|+\|E\|+deg_{in}))$
PEFT [7]	Optimistic cost table	$O(\|V\|^2.p)$
HSV [52]	Heterogeneous priority rank value	$O((\|V\|log\|V\|)p+\|E\|)$
IPEFT [53]	Critical node cost table and pessimistic cost table	$O(\|V\|^2p)$
PATS [54]	Level-based approach	$max(O((\|V\|^2p)/l),O(\|V\|^2))$

2.2.2 Duplication-based Scheduling Algorithms

In this section, we give a brief description regarding some known duplication-based task scheduling algorithms.

Papadimitriou and Yannakakis (PY) [2] have proposed a scheduling algorithm for random DAGs that produces a schedule whose length is at most twice optimal. It combines the task clustering technique with duplications. The algorithm first computes the lower bound values of nodes, starting from the entry nodes, on their earliest possible start time. Then clusters for nodes are found which allow nodes to be started as early as possible. After that clusters are mapped to the appropriate processors. In this algorithm, all possible ancestors of a node are duplicated. Palis

et al. [55] have presented an algorithm similar to PY algorithm [2] except for the cluster finding step. To find the clusters, the algorithm uses a simple greedy strategy which grows the cluster one node at a time.

The CPFD (Critical Path Fast Duplication) algorithm [56] classifies the nodes in a DAG into three categories: Critical Path Node (CPN), In-Branch Node (IBN), and Out-Branch Node (OBN). A CPN is a node on the critical path. An IBN is a node from which there is a path to a CPN. An OBN is a node which is neither a CPN nor an IBN. It duplicates the predecessor nodes of each CPN, which may be CPNs or IBNs, in descending order of message arrival times. The algorithm applies the duplication technique recursively upward from the predecessor nodes so that the CPN being considered can potentially start at the earliest possible time. It performs OBN binding and schedules OBNs after scheduling all the IBNs and CPNs.

In [57], authors classified the existing duplication-based algorithms into SFD (Scheduling with Full Duplication) and SPD (Scheduling with Partial Duplication). The authors attempt to combine good features of both approaches and proposed DFRN (Duplication First and Reduction Next) algorithm. It first duplicates all the duplicable parents at once and tries to delete them one by one if the duplicated task does not meet certain conditions. The DFRN applies the duplication only for the critical processor with the hope that the critical processor is the best candidate for the join node where a join node is a node having incoming degree greater than 1.

The TDS (Task Duplication-based Scheduling) algorithm [58] first calculates certain parameters for each task and based on these parameters, it generates the tasks clusters that may contain two independent tasks and may be assigned to the same processor. The formation of the cluster is finished by searching similar to the depth-first search beginning from the starting task and following the favorite predecessors. If the favorite predecessor is not yet assigned to a processor, then it is selected. Otherwise, the favorite predecessor may or may not be duplicated onto the current processor.

LDBS (Levelized Duplication-Based Scheduling) algorithms [59] utilizes a level sorting technique which arranges independent tasks at each level. Beginning from the top level, the tasks are scheduled on to processors. It uses an existing meta-task

scheduling algorithm, min-min to schedule tasks. In LDBS, two versions of duplication heuristic are given. The LDBS1 algorithm does an exhaustive search among all the tasks within a given level to look for the most suitable task-machine pair while scheduling each task. In the LDBS2, tasks within a level are sorted with bottom level priority and then scheduled one by one.

The SD (Selective Duplication) algorithm [18] combines the ease of list scheduling techniques with the performance of duplication-based techniques. To decrease the waiting time of a task on the processor, SD duplicates the direct predecessors of that task into the idle time slots of the processor. It selectively duplicates the nodes and attempts to avoid unnecessary duplications, only if it helps in improving the performance.

The TANH (Task duplication based scheduling Algorithm for Network of Heterogeneous systems) algorithm [19] first calculates certain parameters for each node and then it generates the clusters and simultaneously chooses a processor for their execution. The final schedule length is generated based on the comparison of the number of available processors and the number of needed processors. If the number of available processors is greater, additional duplication is carried out. Otherwise, the number of clusters is scaled down to the number of available processors.

The HCPFD (Heterogeneous Critical Parents with Fast Duplicator) algorithm [20] consists of a listing technique, which is a modified version of the CNPT (Critical Nodes Parent Trees) algorithm [60], and a machine assignment mechanism based on task-duplication. Both CNPT and HCPFD work on the concept that delaying a critical task will delay its children critical tasks and will delay the exit task. Unlike the general idea of duplication-based algorithms, the HCPFD first selects the machine then checks for duplication. Also, it examines one task and one parent at each step instead of examining one task. The HCPFD assigns higher priority values to the tasks on the critical path, and here, only critical parent of a task is duplicated.

The HLD (Heterogeneous Limited Duplication) algorithm [61] extends the selective duplication approach [18], which improves the performance of the list scheduling algorithms with the addition of limited and effective duplication. The algorithm

schedules the tasks strictly in the order of their global priority and restricts duplications to the most crucial immediate parents to avoid redundant replications.

The HNPD (Heterogeneous N-predecessor Duplication) [62] uses and combines the insertion-based scheduling with multiple task duplication. In order to duplicate a node, it requires idle time on a specific processor. It assigns the highest priority to critical path nodes (CPN) and then to those predecessors of CPNs that include the CPN in their decisive path where the decisive path to a node n_i is the longest path from entry node to n_i excluding the computation cost of the node itself.

The DBUS (Duplication-based Bottom-Up Scheduling) [63] traverses the DAG from bottom-to-top and schedules all children of a task before scheduling the task on as many processors necessary which results in making better duplication decisions as all copies of the parent task are considered at the same time. Here, stopping criterion for duplication is decided by the quality of the current schedule not by the number of duplications already accomplished. DBUS does not impose any restrictions on the number of task duplication.

In HEFD (Heterogeneous Earliest Finish with Duplicator) [21], authors incorporated duplication-based techniques into list scheduling for the problem of scheduling. It uses a modified version of HEFT as a list scheduling approach. It uses a new weight of task nodes and edges using variance to accurately identify the priorities of tasks. The HEFD reduces start execution time of tasks by duplicating its heavily communicating immediate parent tasks to the same processor.

The RADS (Resource-Aware Scheduling Algorithm with Duplications) algorithm [64] concentrates on the removal of redundant task duplications dynamically during the scheduling process. In this, when all children of a task have been assigned, the task is reconsidered to determine whether its copies are necessary or not. Authors have also introduced an optimizing scheme for a schedule generated by the RADS that may further search and reduce resource consumption without degrading the schedule length.

Table 2.2 shows a comparison of some known duplication-based task scheduling algorithms in terms of the concept used and the time complexity. In Table 2.2, $|E|$ and $|V|$ represent the number of edges and the number of nodes in the task graph,

respectively, p represents the number of processors and d_{max} denotes the maximum in/out degree of a task in the task graph.

Table 2.2: A comparison of some known duplication-based task scheduling algorithms.

Algorithm	**Concept**	**Complexity**
PY [2]	Lower-bound	$O(\|V\|^2(\|V\|log\|V\| + \|E\|))$
CPFD [56]	OBN binding	$O(\|E\|\|V\|^2p)$
Greedy PY [55]	Lower-bound	$O(\|V\|(\|V\|log\|V\| + \|E\|))$
DFRN [57]	MAT, CIP and DIP	$O(\|V\|^3)$
TDS [58]	Smallest level first ordering	$O(\|V\|^2)$
LDBS [59]	Level sorting	$O(\|V\|^3\|E\|p^3)$ for version 1 and $O(\|V\|^3\|E\|p^2)$ for version 2
SD [18]	Selective duplication	$O(p\|V\|^2d_{max} + \|E\|)$
TANH [19]	Processor reduction	$O(\|V\|^2)$
HCPFD [20]	Critical parents	$O(\|V\|^2p)$
HLD [61]	Bottom-level priority	$O(p\|V\|^2d_{max} + \|E\|)$
HNPD [62]	Decisive Path	$O(\|V\|^2p)$
DBUS [63]	Critical Path	$O(p^2\|V\|^2)$
HEFD [21]	Earliest Finish Time	$O(\|V\|\|E\|p)$
RADS [64]	Redundancy deletion	$O(\|V\|^2p)$

2.2.3 Clustering-based Scheduling Algorithms

In this section, we give a brief description regarding some known clustering-based task scheduling algorithms.

One of the most famous clustering-based algorithms is Sarkar′s algorithm [3] that utilizes the concept of edge zeroing for clustering the tasks. In this concept, the tasks which involve large communications are grouped and executed on the same processor to minimize the makespan of the task graph. This algorithm initially kept each task in a separate cluster and sorted the edges by their weights in non-increasing order, then scrutinizes edges one-by-one and zeroes them if the makespan does not increase. After edge zeroing concept, Kim and Browne [23] came with the idea of

linear clustering that finds the critical path in the task graph and merges all the nodes of a critical path in the same cluster.

Yang and Gerasoulis [24, 65] presented an algorithm, called Dominant Sequence Clustering (DSC) that groups the nodes of the task graph by their priorities which is determined by using the idea of the bottom level and top level. The top level of a node is the length of the longest path from an entry node up to that node (excluding the weight of the node) in the task graph. The greedy version of DSC algorithm is given by Dikaiakos et al. [66].

Wu and Gajski [11] proposed MCP (Modified Critical Path) algorithm for a limited number of processors that can also perform as an edge-zeroing heuristic when used for an unlimited number of processors. It exploits the idea of As-Late-As-Possible (ALAP) binding which assigns the latest possible execution time to the task and determines the ALAP time for each task by moving downward through the DAG. The MCP maintains an increasing lexicographical sorted list of tasks according to their ALAP time and schedules the first task from the list at each step. In [11], Wu and Gajski proposed another scheduling algorithm, called MD (Mobility Directed) that assigns tasks to processors according to their relative mobilities.

Like MCP, the DCP (Dynamic Critical Path) algorithm [14] is given for a bounded number of processors and becomes an edge-zeroing heuristic when it is utilized for an unbounded number of processors. At each iteration, it determines the difference between the absolute EST and absolute latest start time for each task of the CP and schedules the task which has the smallest difference among all tasks. The authors called the intermediate CP as the dynamic CP to differentiate it from the initial CP of the DAG.

Kadamuddi and Tsai [26] presented a Clustering Algorithm for Synchronous Communication (CASC) that efficiently parallelizes the input application on multiprocessors. It includes 4 steps, that is, Initialize, ForwardMerge, BackwardMerge, and EarlyReceive. It increases the performance that occurred because of synchronous communications. The CASC prevents deadlocks and handles the problems of blocking synchronous sends at the clustering phase.

Mishra and Tripathi [27] defined a priority function for cluster pairs in the task graph and proposed an algorithm, named Cluster Pair Priority Scheduling (CPPS).

The CPPS initially determines the priorities for all cluster pairs and sorts them in decreasing order, then inspects each cluster pair one-by-one and groups them if makespan decreases. The priorities of cluster pairs change whenever a grouping of tasks take place; the CPPS performs above steps until makespan decreases.

Mishra et al. [67] proposed a randomized algorithm, called RDCC (Randomized Dynamic Computation Communication) for task scheduling. The RDCC is a randomization of the CCLC (Computation Communication Load Clustering) algorithm given in [68] and uses the concept of dynamic priority from DCCL (Dynamic Computation Communication Load) algorithm [69]. The CCLC determines the CCLoad (Computation-Communication-Load) for each task and sorts them according to their CCLoad values. The algorithm initially places all tasks in one cluster, then extracts them one-by-one at each step and places them in separate clusters if makespan decrease. The DCCL algorithm works similar to CCLC and utilizes the concept of priority which is based on the calculation of Dynamic CCLoad.

Khaldi et al. [70] presented an algorithm, Bounded Dominant Sequence Clustering (BDSC), that extends the DSC algorithm for a limited number of processors. The BDSC considers several major attributes like memory constraints, dependency constraints, processor availability, etc. The algorithm reduces the number of clusters to match the processor quantities and assigns each cluster to available processors which provide the smallest top level. The algorithm utilizes an extra heuristic to decide the precedence among tasks by their bottom level when tasks have equal priorities.

In [71], the authors proposed a general randomized task scheduling algorithm called LOCAL(A, B) for multiprocessor environments. It uses a local neighborhood search and gives a hybrid of two known task scheduling algorithms (A, and B). As an instance, the authors selected DSC as algorithm A and CPPS as algorithm B.

Table 2.3 shows a comparison of the existing clustering-based task scheduling algorithms in terms of the concept used and the time complexity. In Table 2.3, $|E|$ and $|V|$ represent the number of edges and the number of nodes in the task graph, respectively, $'a'$ represents the number of randomization steps and $'b'$ denotes a limit on the number of clusters formed.

Table 2.3: A comparison of some known clustering-based task scheduling algorithms.

Algorithm	**Concept**	**Complexity**
EZ [3]	Edge zeroing	$O(\|E\|(\|V\|+\|E\|))$
LC [23]	Critical Path	$O(\|V\|(\|V\|+\|E\|))$
DSC [24, 65]	Dominant Sequence	$O((\|V\|+\|E\|)log(\|V\|))$
GDS [66]	Greedy Dominant Sequence	$O(\|V\|(\|V\|+\|E\|))$
MCP [11]	ALAP binding	$O(\|V\|^2 log(\|V\|))$
MD [11]	Relative mobility	$O(\|V\|^3)$
DCP [14]	Critical Path	$O(\|V\|^3)$
CASC [26]	Synchronous Communication	$O(\|V\|(\|E\|^2+log(\|V\|)))$
CPPS [27]	Cluster Pair Priority	$O(\|V\|\|E\|(\|V\|+\|E\|))$
CCLC [68]	CCLoad	$O(\|V\|^2(\|V\|+\|E\|)log(\|V\|+\|E\|))$
DCCL [69]	Dynamic CCLoad	$O(\|V\|^2(\|V\|+\|E\|)log(\|V\|+\|E\|))$
RDCC [67]	Randomization	$O(ab\|V\|(\|V\|+\|E\|)log(\|V\|+\|E\|))$
BDSC [70]	Dominant Sequence	$O(\|V\|^3)$
LOCAL [71]	Local Search	$O(\|V\|\|E\|(\|V\|+\|E\|))$

2.2.4 Outcome of Literature Review

Efficient scheduling for distributed computing is critical for achieving improved performance of parallel applications. As task scheduling problem belongs to NP-complete class, researchers proposed many algorithms for this problem and newer algorithms keep coming. Consequently, various scheduling algorithms such as list scheduling, duplication-based scheduling and the clustering-based scheduling algorithms have been proposed in the literature.

List scheduling algorithms mainly focus on ranking the tasks and selecting the best processor for execution of the tasks. Most of the existing list scheduling algorithms utilizes the combinations of bottom-level and top-level for computing priorities of tasks. For task prioritization, these algorithms does not consider the number of successor tasks, priorities of predecessor tasks and quantity of the data to be communicated between the tasks.

Duplication-based scheduling algorithms focus on reducing the communication cost between tasks by duplicating the crucial or all predecessor tasks on to the processors of their successor tasks. The concept of task duplication plays an effective role in improving the performance of applications having high CCR values. These algorithms mostly used in the combination with list scheduling algorithms.

Clustering-based scheduling algorithms focus on minimizing the communication cost between tasks through clustering of tasks. These algorithms are nonbacktracking algorithms. In other words, clusters that have been merged in some step of the algorithm are never unmerged at a later time. This approach simplifies the clustering heuristics and significantly reduces their complexity. Clustering-based scheduling algorithms are also more advantageous as they embrace a more global view, whereas list scheduling algorithms, in contrast, tend only to make local optimization decisions. This thesis concentrates on developing some alternative clustering-based algorithms for scheduling tasks having precedence constraints on multiprocessors.

2.3 Summary

This chapter first gives preliminary concepts related to the task scheduling along with assumptions, notation and definitions for the sake of completeness. Further, a literature review regarding task scheduling algorithms have been given to understand the current research state of the existing scheduling algorithms for distributed computing in multiprocessor environments. In the next chapter, we compare and understand existing scheduling algorithms to obtain a way of benchmarking task scheduling algorithms. In the next to next chapters, we attempt to give alternative solutions to address task scheduling problem for distributed computing in multiprocessor environments.