A Leveled Dag Critical Task Firstschedule Algorithm in Distributed Computing Systems

Amal EL-NATTAT
Computer Science & Eng. Dept.,
Faculty of Electronic Eng.
Menouf 32952, Egypt

Nirmeen A. El-Bahnasawy
Computer Science & Eng. Dept.,
Faculty of Electronic Eng.
Menouf 32952, Egypt

Ayman EL-SAYED, IEEE Senior Mem.
Computer Science & Eng. Dept
Faculty of Electronic Eng.
Menouf 32952, Egypt

***Abstract*—In distributed computing environment, efficient task scheduling is essential to obtain high performance. A vital role of designing and development of task scheduling algorithms is to achieve better makes pan. Several task scheduling algorithms have been developed for homogeneous and heterogeneous distributed computing systems. In this paper, a new static task scheduling algorithm is proposed namely; Leveled DAG Critical Task First (LDCTF) that optimizes the performance of Leveled DAG Prioritized Task (LDPT) algorithm to efficiently schedule tasks on homogeneous distributed computing systems. LDPT was compared to B-level algorithm which is the most famous algorithm in homogeneous distributed systems and it provided better results. LDCTF is a list based scheduling algorithm which depends on sorting tasks into a list according to their priority then scheduling one by one on the suitable processor. LDCTF aims to improve the performance of the system by minimizing the schedule length than LDPT and B-level algorithms.**

***Keywords*—Task scheduling; Homogeneous distributed computing systems; Precedence constrained parallel applications; Directed Acyclic Graph; Critical path**

I. Introduction

Distributed systems have emerged as powerful platforms for executing parallel applications. A distributed system can be defined as a collection of computing systems that appears to its users as a single system, these systems collaborate over a network to achieve a common goal [1]. There are two types of distributed systems; homogeneous (in which processors are identical in capabilities and functionality) and heterogeneous (in which processors are different).

In distributed computing environment, an application is usually decomposed into several independent and/or interdependent sets of cooperating tasks. Dependent tasks are represented by a Directed Acyclic Graph (DAG). DAG can be defined as a graph consists of a set of vertices or nodes and a set of edges G(V, E) in which each node represents a task and each edge represents a communication between two tasks (the two tasks are dependent on each other). The computation cost of the task is represented by a weight associated with each node and the communication cost between two tasks is represented by a weight associated with each edge. The communication cost between two dependent tasks is considered to equal zero if they are executed on the same processor. Figure 1 shows an example of a simple task graph (DAG). In the Figure, t0 is called predecessor (or parent) of t2 and t2 is called successor (or child) of t0. The edge between t0 and t2 means that t2 can start execution only after t0 finishes its execution. Efficient task scheduling of application tasks is essential to achieve high performance in parallel and distributed systems. The basic function of task scheduling is to determine the allocation of tasks to processors and their execution order in order to satisfy the precedence requirements and obtain minimum schedule length (or make span) [2].Task-scheduling algorithms are broadly classified into two basic classes: static and dynamic. In static scheduling, the characteristics of an application, such as execution time of tasks and data dependencies between tasks are known in advance (during compile time before running the application). In dynamic scheduling, some information about tasks and their relations may be undeterminable until run-time [3].

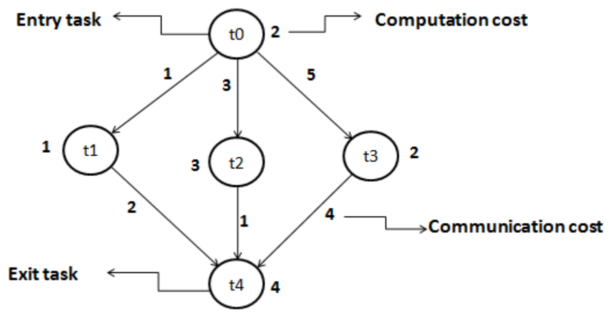

Fig. 1. Example of a DAG

Over the past few decades, researchers have focused on designing task scheduling algorithms for homogenous and heterogeneous systems with the objective of reducing the overall execution time of the tasks. Topcuoglu et al. [2] have presented HEFT and CPOP scheduling algorithms for heterogonous processors. Luiz et al. [4] have developed lookahead-HEFT algorithm, which look ahead in the schedule to make scheduling decisions. Eswari, R. and Nickolas, S. [5] have proposed PHTS algorithm to efficiently schedule tasks on the heterogeneous distributed computing systems. Rajak and Ranjit [6] have presented a queue based scheduling algorithm called TSB to schedule tasks on homogeneous parallel multiprocessor system. Ahmed, S.G.; Munir, E.U.; and Nisar, W. [7] have developed genetic algorithm called PEGA that provide low time complexity than standard genetic algorithm (SGA). Xiaoyong Tang; Kenli Li; Renfa Li; and Guiping Liao [8] have presented a list-scheduling algorithm called HEFD for heterogeneous computing systems. Nasri, W. and Nafti, W. [9] have developed a new DAG scheduling algorithm for heterogeneous systems that provide better performance than some well-known existing task scheduling algorithms.

In homogeneous distributed systems, researchers have developed many heuristic task-scheduling algorithms such as ISH [10], ETF [11], DLS [12], MH [13],B-level [14] and some heuristics that depend on the critical path such as MCP [15], FCP [16], and CNPT [17]. Among these algorithms, B-level provides the best performance in terms of schedule length, speedup, and efficiency. LDPT (Leveled DAG Prioritized Task) algorithm [18]was compared to B-level algorithm which is the most famous algorithm in homogeneous distributed systems and it provided better results.

In this paper, the problem of scheduling precedence constrained parallel tasks on homogeneous physical machines (PMs) is addressed. A new static scheduling algorithm called LDCTF is proposed. The goal of LDCTF is to optimize the performance of LDPT [18] algorithm in order to provide better system performance. LDCTF is a list scheduling algorithm. It depends on dividing the DAG into levels then sorting tasks in each level into a list according to their priority and finally, picking tasks from the list one by one to schedule it on the suitable processor. LDCTF is compared to LDPT and B-level algorithms and it provided better results in terms of schedule length, speedup, and efficiency.

The remainder of this paper is organized as follows. Section II provides an overview of the related work algorithm. The proposed algorithm is discussed in section III. Section IV presents performance evaluation results of the proposed algorithm. Finally, conclusion and future work is reviewed in section V.

II. LDPT Algorithm

LDPT is a list based scheduling algorithm. It depends on dividing the DAG into levels with considering the dependency conditions among tasks in the DAG. The algorithm has two phases: (1) Task prioritization phase, (2) Processor selection phase. LDPT algorithm depends on giving a priority to each task as shown in Figure 2 then; scheduling each task on one processor with taking into consideration the insertion-based policy. Figure 2 shows the pseudo code of LDPT algorithm.

```
Generate the DAG
Divide the DAG into levels according to their
communicated dependency
Sort the constructed levels according to dependency
ordering
Sort tasks according to [their computation costs then their
direct communication of its next level] in descending order
While there are unscheduled levels do
     While there are unscheduled tasks do
            For each level do
            Find the task with the highest computation cost
            If there are tasks have equal computation cost
               Then
Choose the task with the highest communication cost with
                      its Childs in next level
            End if
               Find the processor that minimizes the Earliest
               Start Time of the selected task
               Assign the task to the selected processor
               Remove the selected task from the list
               Repeat
               Until all tasks are scheduled
End for each
End while
```

Fig. 2. LDPT algorithm [18]

III. Leveled DAG Critical Task First (LDCTF) Algorithm

LDCTF is a theoretical task scheduling algorithm. LDCTF, LDPT, and B-level algorithms are applied on Standard Task Graph STG [19] as a bench mark, and it was found that LDCTF algorithm is more efficient than LDPT and B-level algorithms.

LDCTF is a list based scheduling algorithm. It depends on dividing the DAG into levels with considering the dependency conditions among tasks in the DAG then, applying the Min-min method [20] which means calculating the minimum completion time (MCT) for each task on all processors then selecting the task with the lowest MCT to schedule. The algorithm has two phases: (1) Task prioritization phase, (2) Processor selection phase.

A. *Task prioritization phase:*

In this phase, the critical path [2] is calculated for the DAG (critical path is the longest path from the entry task to the exit task in the graph) then, the DAG is divided into levels and the tasks in each level will be sorted into a list based on their priority. The priority for each task is given as follow:

1) First, the critical task (task located on the critical path) in each level will have the highest priority.

2) Then, the expected Earliest Finish Time (EFT) is calculated for the other tasks in the same level and the task with the lowest EFT will have the highest priority. If tow tasks have equal EFT value then, the task with the lowest task number will have the highest priority. EFT of a task tion processor pj is computed as follow:

$$EFT(t_i, P_j) = w_{i,j} + EST(t_i, P_j) \text{-----------------} (1)$$

3) Finally, tasks in each level are sorted into the list in ascending order according to their EFT value.

B. *Processor Selection Phase:*

In this phase, the tasks are picked from the list one by one and assigned to the processor that will minimize the earliest start time of the task, with taking into consideration the insertion-based policy. The insertion policy means that if there is an idle time slot on the processor between two already scheduled tasks and it was enough for executing the task, then the task is assigned on that processor in this idle slot without violating precedence constraints. In other words, a task can be scheduled earlier if there is a period of time between two tasks already scheduled on processor (P), where P runs idle.If two processors provide the same start time for the task then, the task is assigned to the first processor that will minimize the EST of it.The Earliest Start Time of a task $n_i$on a processor $P_j$is defined as:

$$\boldsymbol{EST(n_x, P_m)=max[TAvailable(P_m),max\{AFT(n_i)+c_{x,i}\}]} \quad (2)$$

Where TAvailable(P_m) is the earliest time at which processor $P_m$is ready. AFT(n_i) is the Actual Finish Time of a task n_i (the parent of task nx) on the processorP_m. $c_{n,i}$is the communication cost from task n_i to task n_x,$c_{k,i}$ equal zero if the predecessor task $t_k$is assigned to processor P_m. For the entry task,EST(n_{entry}, P_m)= 0. Figure 3shows the pseudo code of LDCTF algorithm.

C. *Case Study*

Consider the DAG shown in Figure 4; assume the system has two processors (P0, P1). The critical path for the DAG in Figure 4 is (t0, t1, t3, t6, t8). Table 1 shows the computation cost for each task. Both LDPT and LDCTF algorithms generate a list of tasks that shows the execution order of them. Table 2 shows the lists generated by LDPT and LDCTF algorithms. For LDCTF algorithm the critical task in each level will be scheduled first as shown in table 2. Figure (5.a, 5.b) shows the Gantt chart generated by LDPT and LDCTF algorithms respectively. Both algorithms assign the selected task to the processor that minimizes the start time (EST) of it. For example, in Figure 5.a, the EST for task t2 on p0 is 5 and the EST for t2 on p1 is 4, so the task t2 is scheduled on p1.In Figure 5.b, the same manner if followed with taking into consideration the insertion-based policy. From Figure 5, it is shown that the schedule length (the finish time of the last task scheduled from the DAG) resulted from LDPT and LDCTF algorithms is 25, and 23 unit of time respectively.

```
Generate the DAG
Calculate the critical path for the DAG
Divide the DAG into levels according to their communicated
dependency
Sort the constructed levels according to dependency ordering
Determine the critical task for each level
While there are unscheduled levels do
   While there are unscheduled tasks do
          For each level do
          For each task in level
          Calculate the expected EFT of selected task
          End for
          Sort level tasks in Tasks Ordered List according to
          1-Critical task
          2-Expected EFT in ascending order
          If there are tasks have equal Earliest Finish Time
             Then
            Choose the task with the lowest task number
          End if
          For each task in Tasks Ordered List
             Find the processor that minimizes the Earliest Start Time
             of the selected task
             Assign the task to the selected processor
             Remove the selected task from the list
             Repeat
             Until all tasks are scheduled
End for each
End while
```

Fig. 3. Leveled DAG Critical Task First algorithm

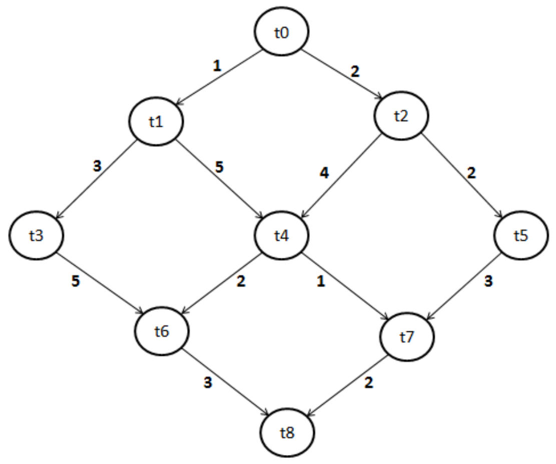

Fig. 4. Sample DAG

TABLE I. COMPUTATION COST

Task	Computation Cost
t_0	2
t_1	3
t_2	1
t_3	4
t_4	3
t_5	5
t_6	2
t_7	4
t_8	6

Figure 5 depicts the Gantt chart generated by LDPT and LDCTF algorithms. From the Figure, it is shown that the schedule length generated from LDPT algorithms is 25 unit time while the schedule length generated from LDCTF algorithm is 23 unit time. In case of LDCTF, we observe that there is less periods in which processors are idle than LDPT. According to this result, the overall running time of the application will be decreased and the efficiency of the system will be improved.

TABLE II. TASK LISTS FOR LDPT AND LDCTF ALGORITHMS

Execution	LDPT	LDCTF
1	t_0	t_0
2	t_1	t_1
3	t_2	t_2
4	t_5	t_3
5	t_3	t_5
6	t_4	t_4
7	t_7	t_6
8	t_6	t_7
9	t_8	t_8

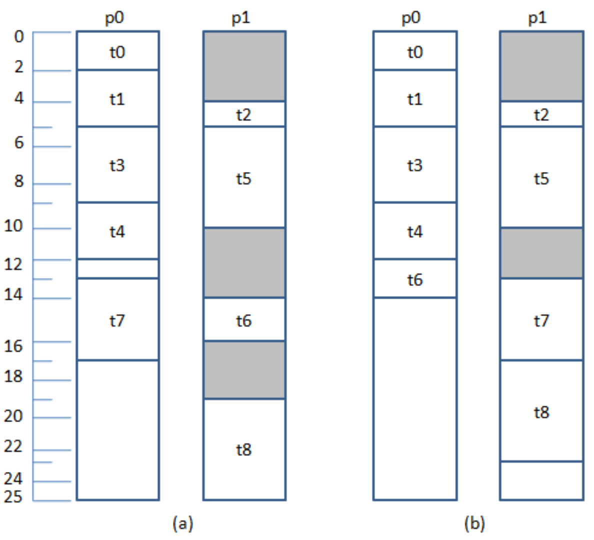

Fig. 5. The schedules generated by (a) LDPT algorithm (b) LDCTF algorithm for sample DAG

IV. Results and Performance Evaluation

A. *Simulation Environment*

To evaluate the performance of LDCTF algorithm, a simulator had been built using visual C# .NET 4.0 on machine with: Intel(R) Core(TM) i3 CPU M 350 @2.27GHz, RAM of 4.00 GB, and the operating system is window 7, 64-bit.To test the performance of LDPT and LDCTF algorithms, a set of randomly generated graphs is created by varying a set of parameters that determines the characteristics of the generated DAGs. These parameters are described as follows: DAG size (n: the number of tasks in DAG).Density (d: the probability of existence edge between ni in levelj and nx in the next level levelj+1 for DAG. Where, i, x=1,2,…, N, and N is the number of tasks, j=1, 2,…, T, and T is the number of levels inDAG).With six different numbers of processors varying from 2, 4, 8, 16, 32 and 64 processors. For each number of processors, six different DAG sizes have been used varying from 10, 20,40,60,80 and 100 nodes.

B. *Evaluation Metricsa*

The most important metrics for evaluating performance of scheduling algorithms are schedule length, speed up, and efficiency. Schedule length is the maximum finish time of the last task (exit task) scheduled from the DAG.

$$\textbf{\textit{Schedule length= Max(AFT(nexit))}} \text{ ----------------------------(3)}$$

Where AFT(nexit) is the actual finish time of the exit task. Speedup is defined as the ratio of the schedule length generated from executing the application on one processor to the schedule length generated from executing the application on multiple parallel processors.

$$\textbf{\textit{Speed up=}} \frac{\underset{pj\epsilon P}{Min}[\sum_{ni\epsilon V} w(i,j)]}{SL} \text{---------------------------------------(4)}$$

Where $w(i, j)$means the weight of task ni on processor pjand SL means the schedule length. Efficiency is the inverse of speed up.

$$\boldsymbol{Efficiency} = \frac{\boldsymbol{speedup}}{\boldsymbol{Number\ of\ processors}} \text{-------------------------(5)}$$

C. *Experimental Results*

The schedule length generated byLDPT and LDCTF algorithms is shown in Figure 6, 7, 8, 9, 10, 11 for 10, 20, 40, 60, 80, 100 tasks respectively and the results are recorded in table 3. According to the results, the schedule length is decreased that will minimize the running time of the application. The improvement ratio in schedule length is (2.75%). Figure 12, 13, 14, 15, 16, 17 show a comparative study of the speed up of LDPT and LDCTF algorithms in case of 2, 4, 8, 16, 32, 64 processors respectively. Table 4 shows the speedup results of LDPT and LDCTF algorithms. From the results, we can see that the improving ratio in speed up is (3.2%). Table 5 shows the efficiency results of LDPT and LDCTF algorithms. From Figure 18, 19, 20, 21, 22, 23 we can see that LDCTF is more efficient than LDPT algorithms with improving ratio (1.9%). The schedule length generated by B-level, LDPT, and LDCTF algorithms is shown in Figure 24, 25, 26, 27, 28, 29. Figure 30, 31, 32, 33, 34, 35 shows a comparative study of the speed up of B-level, LDPT, and LDCTF algorithms. The efficiency results of B-level, LDPT, and LDCTF algorithms are shown in Figure 36, 37, 38, 39, 40, 41.

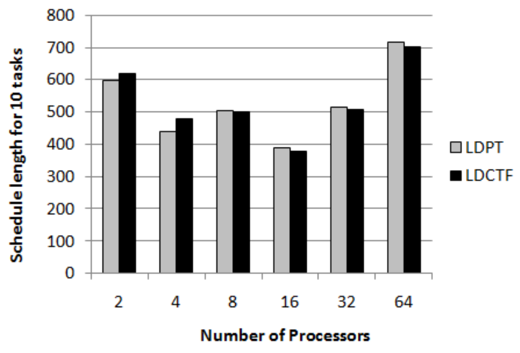

Fig. 6. Schedule length for 10 tasks

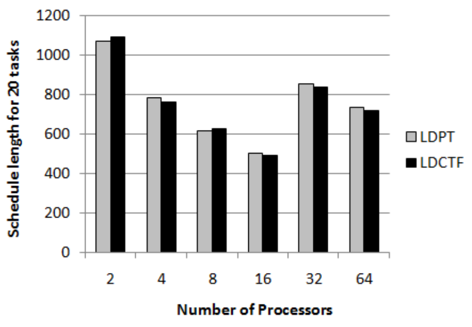

Fig. 7. Schedule length for 20 tasks

Figure6, 7, 8, 9, 10, 11 depict the schedule length versus number of tasks with varying number of processors 2, 4, 8, 16, and 32 processors. It is shown that the schedule length in case of applying LDCTF algorithm is less than LDPT algorithm. This is because the periods in which processors are idle in case of LDCTF are less than LDPT algorithm.

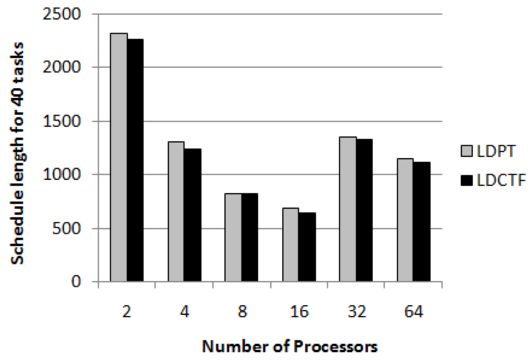

Fig. 8. Schedule length for 40 task

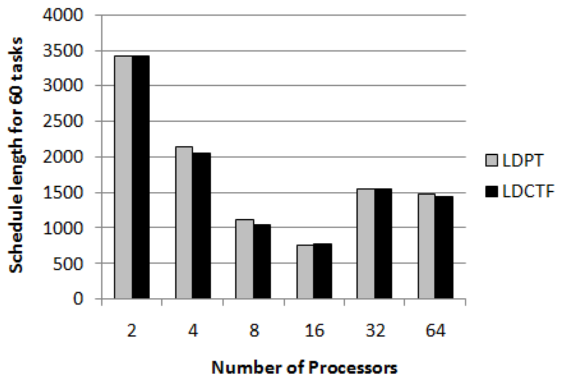

Fig. 9. Schedule length for 60 task

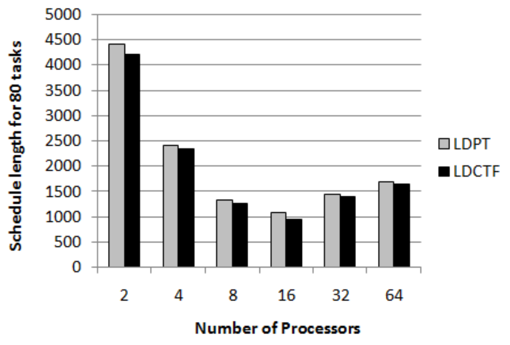

Fig. 10. Schedule length for 80 task

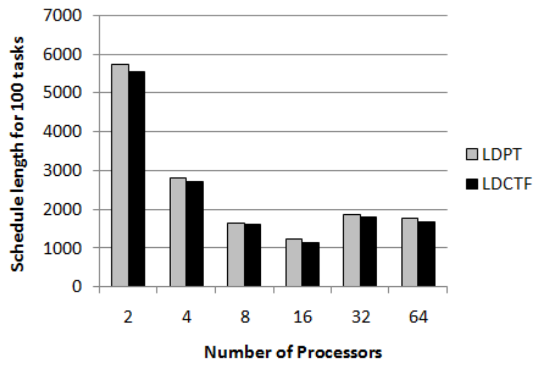

Fig. 11. Schedule length for 100 task

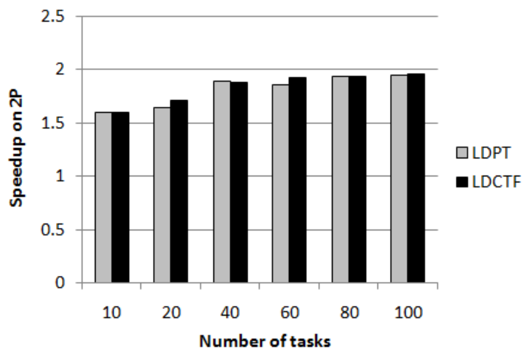

Fig. 12. Speedup on 2 processors

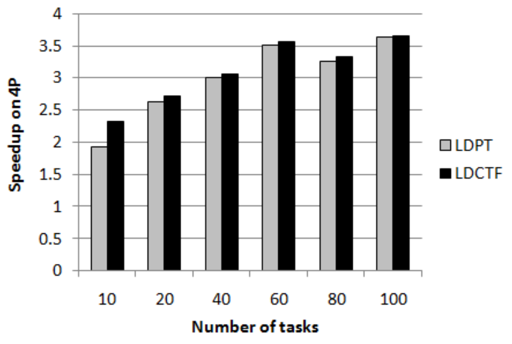

Fig. 13. Speedup on 4 processors

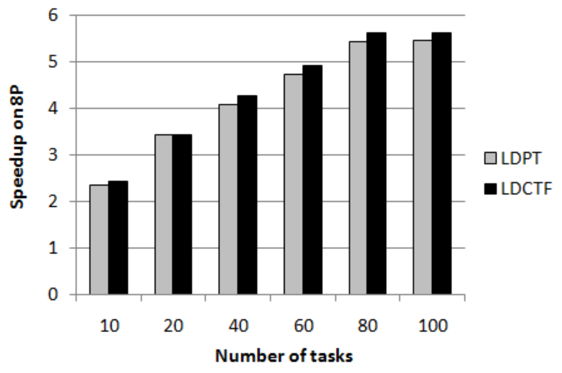

Fig. 14. Speedup on 8 processors

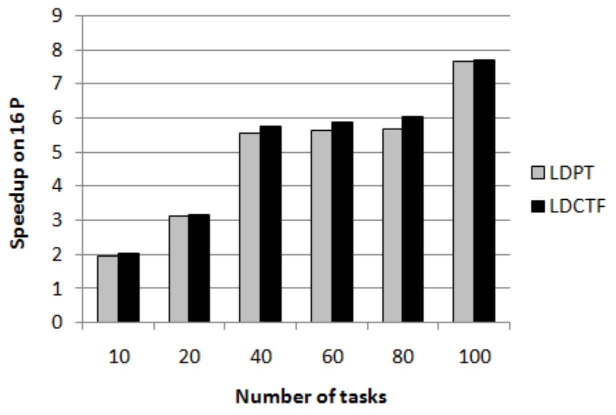

Fig. 15. Speedup on 16 processors

TABLE III. SCHEDULE LENGTH RESULTED FROM LDPT AND LDCTF ALGORITHMS

Number of tasks	2 processor		4 processor		8 processor		16 processor		32 processor		64 processor	
	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF
10	598	619	438	481	503	501	390	377	516	508	718	701
20	1070	1091	784	763	615	629	504	490	856	838	733	720
40	2319	2265	1304	1236	827	818	691	637	1348	1325	1151	1117
60	3427	3430	2141	2055	1114	1053	765	783	1557	1544	1469	1441
80	4408	4204	2403	2349	1330	1261	1072	955	1442	1403	1690	1642
100	5734	5551	2821	2724	1654	1604	1218	1124	1858	1806	1755	1666

TABLE IV. SPEEDUP RESULTED FROM LDPT AND LDCTF ALGORITHMS

Number of processors	10 tasks		20 tasks		40 tasks		60 tasks		80 tasks		100 tasks	
	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF
2	1.597101	1.604076	1.646914	1.710256	1.889304	1.884024	1.861015	1.925334	1.941921	1.938553	1.95121	1.957603
4	1.926573	2.32981	2.625984	2.722449	3.01626	3.05922	3.509259	3.567059	3.261836	3.330391	3.648107	3.658973
8	2.339703	2.421978	3.42637	3.42637	4.071071	4.258639	4.71821	4.918985	5.438854	5.611392	5.46696	5.615385
16	1.965385	2.015779	3.126603	3.172358	5.570213	5.749634	5.649007	5.88856	5.671501	6.030069	7.672043	7.707561
32	1.965385	2.015779	4.363834	4.402198	4.718067	4.892601	5.672481	5.883417	7.498576	7.695907	7.63357	7.985985
64	2.066937	2.088115	2.782361	3	4.601407	4.774939	5.362133	5.529858	6.167453	6.291099	6.755631	7.024131

TABLE V. EFFICIENCY RESULTED FROM LDPT AND LDCTF ALGORITHMS

Number of processors	10 tasks		20 tasks		40 tasks		60 tasks		80 tasks		100 tasks	
	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF	LDPT	LDCTF
2	61.59624	63.94366	55.79577	56.94572	52.51707	53.2254	52.22963	53.03263	51.05813	51.52983	51.24817	51.62996
4	38.20755	40	34.6574	34.95531	30.53475	30.58641	28.24712	28.19847	26.86957	26.90683	27.48918	27.43998
8	40.21632	40.21632	23.46253	24.34109	17.45132	17.8457	15.93451	16.25871	14.7612	15.02103	14.97816	15.22779
16	32.63062	32.63062	20.13423	23.2318	12.44079	13.06407	10.4242	10.83525	8.181933	8.51226	8.97423	9.236988
32	35.92677	35.92677	14.66799	14.66799	10.17964	11.00299	7.395091	7.854315	7.148139	7.394627	5.821122	6.124305
64	37.26462	37.26462	18.53759	18.53759	8.372093	8.372093	5.611434	5.593789	4.385628	5.140007	4.311791	4.666996

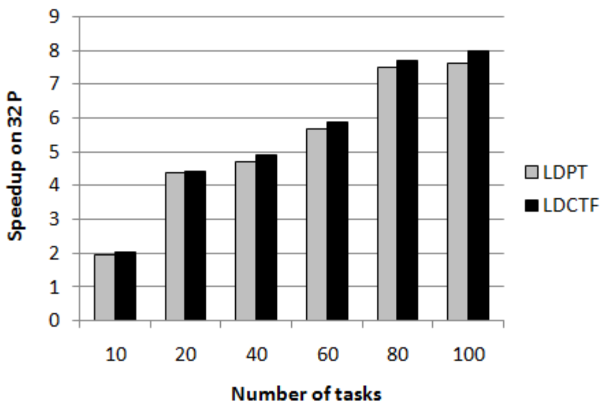

Fig. 16. Speedup on 32 processors

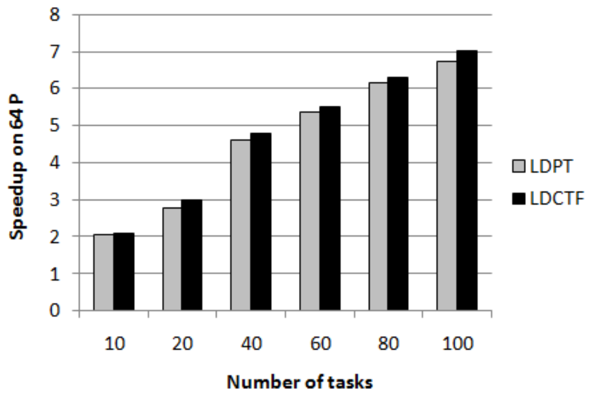

Fig. 17. Speedup on 64 processors

Figure 12, 13, 14, 15, 16, 17 depict speedup versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm provides better speed up than LDPT algorithm. This is because in case of LDCTF algorithm, all processors have finished the execution of tasks earlier than LDPT algorithm.

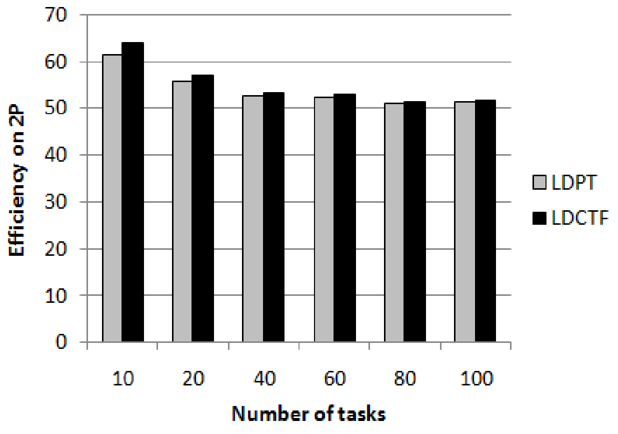

Fig. 18. Efficiency on 2 processors

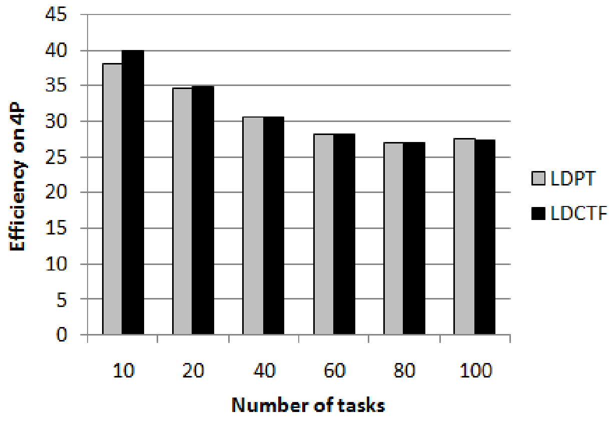

Fig. 19. Efficiency on 4 processors

Figure 18, 19, 20, 21, 22, 23 depict efficiency versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm is more efficient and provides better performance than LDPT algorithm. Most of processor elements have been perfect utilized in our algorithm because of the communication among tasks is not affected in algorithm breadth procedures.

Figure 24, 25, 26, 27, 28, 29 depicts the schedule length versus number of tasks with varying number of processors 2, 4, 8, 16, 32, and 64 processors. It is shown that the schedule length in case of applying LDCTF algorithm is less than LDPT and B-level algorithms.

Figure 30, 31, 32, 33, 34, 35 depicts speedup versus number of processors with varying number of tasks (10, 20, 40, 60, 80, 100). It is shown that LDCTF algorithm provides better speed up than LDPT and B-level algorithms. This is because in case of LDCTF algorithm, all processors have finished the execution of tasks earlier than LDPT and B-level algorithms.

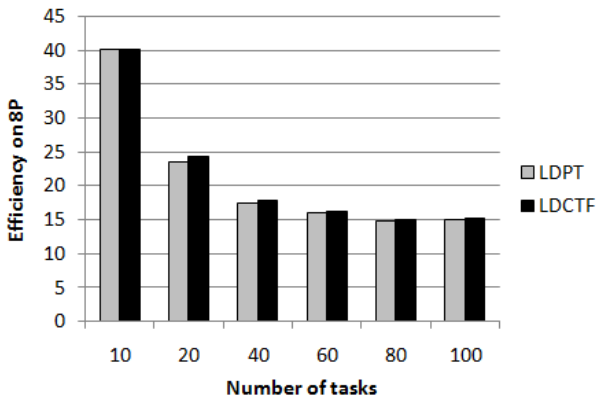

Fig. 20. Efficiency on 8 processors

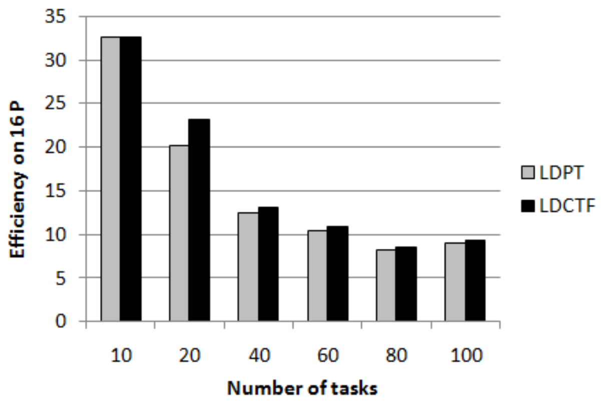

Fig. 21. Efficiency on 16 processors

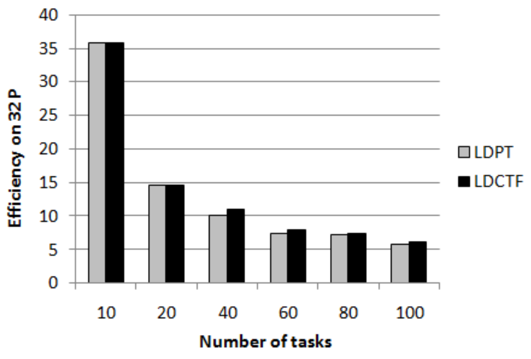

Fig. 22. Efficiency on 32 processors

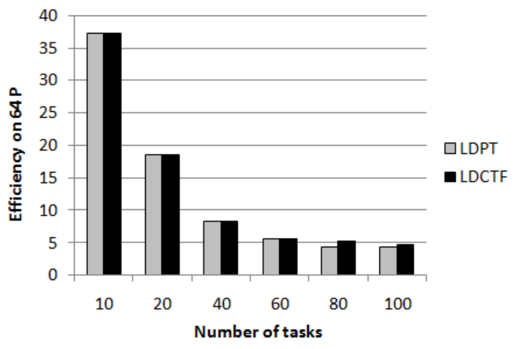

Fig. 23. Efficiency on 64 processors

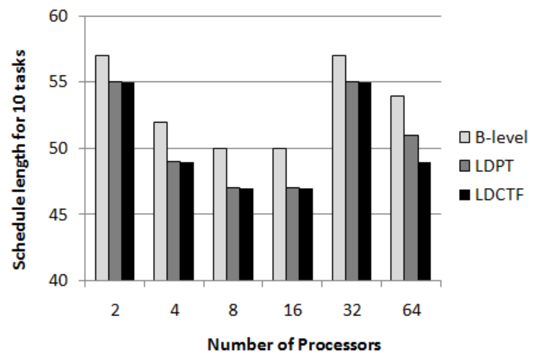

Fig. 24. Schedule length for 10 tasks

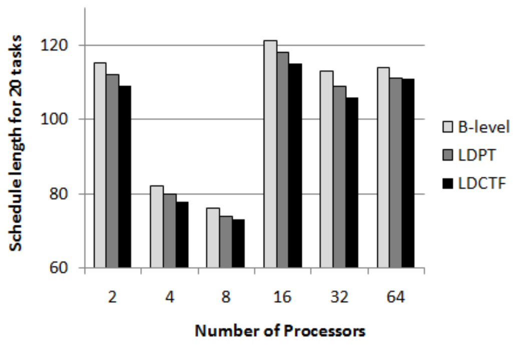

Fig. 25. Schedule length for 20 tasks

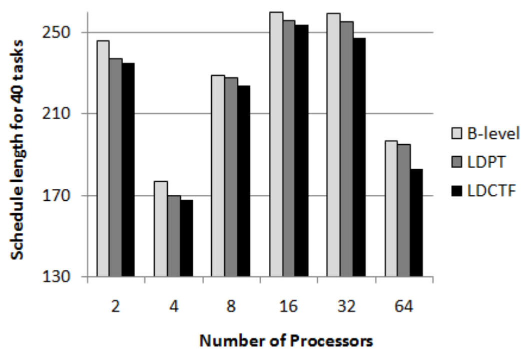

Fig. 26. Schedule length for 40 tasks

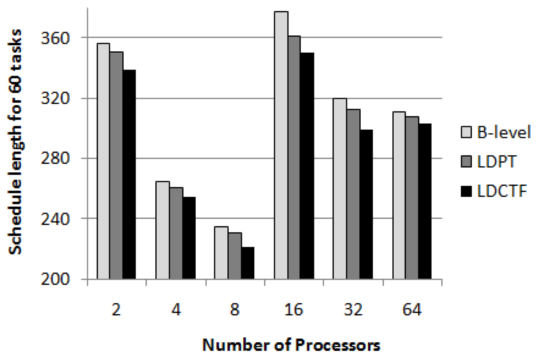

Fig. 27. Schedule length for 60 tasks

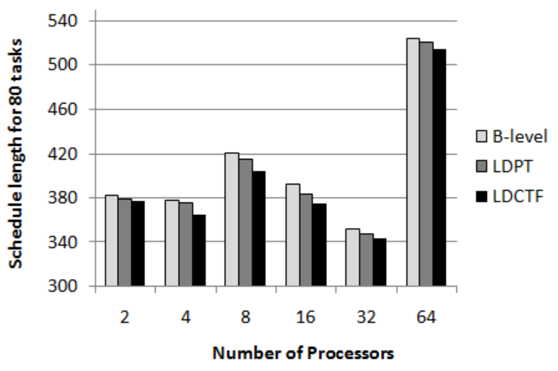

Fig. 28. Schedule length for 80 tasks

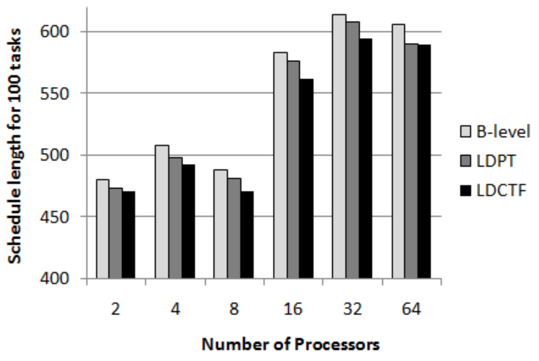

Fig. 29. Schedule length for 100tasks

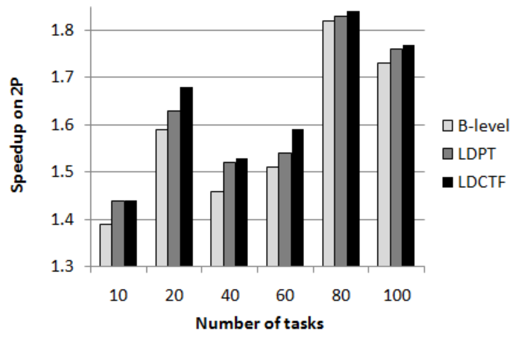

Fig. 30. Speedup on 2 processors

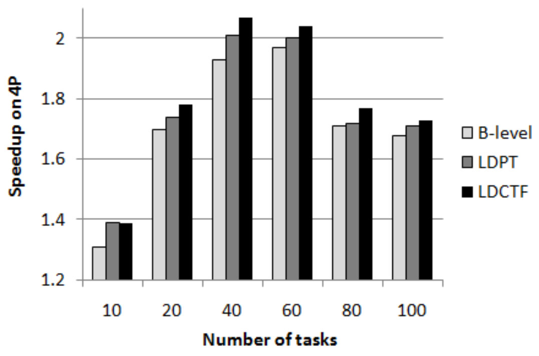

Fig. 31. Speedup on 4 processors

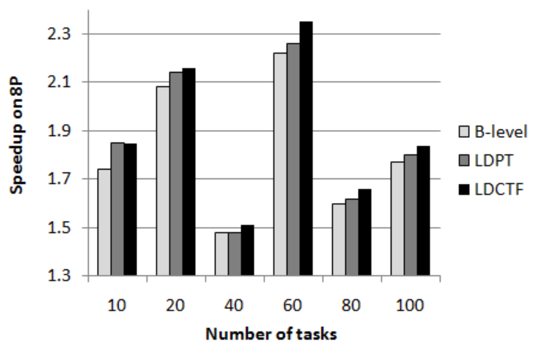

Fig. 32. Speedup on 8 processors

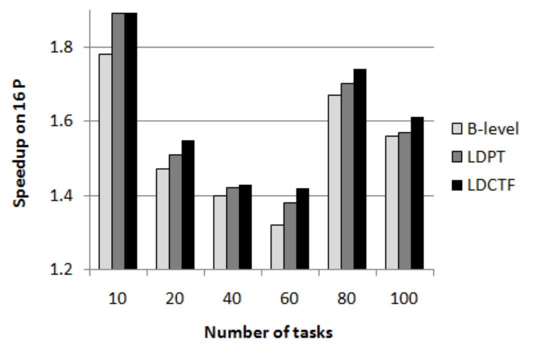

Fig. 33. Speedup on 16 processors

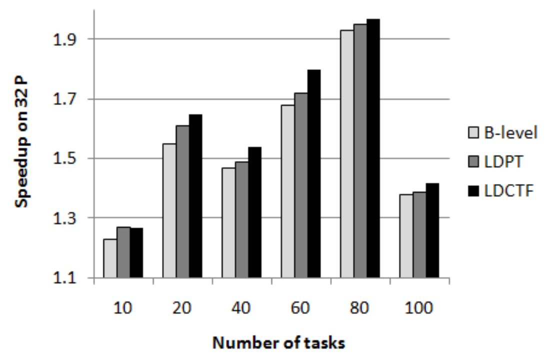

Fig. 34. Speedup on 32 processors

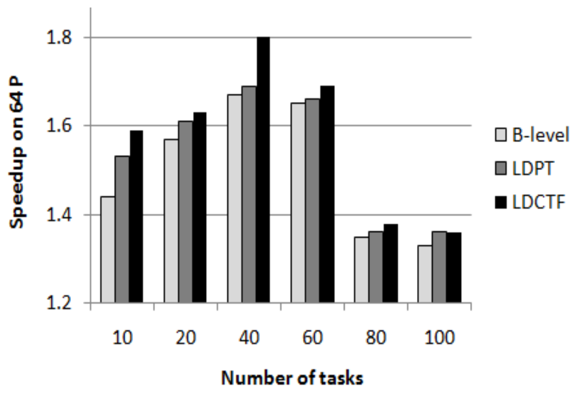

Fig. 35. Speedup on 64 processors

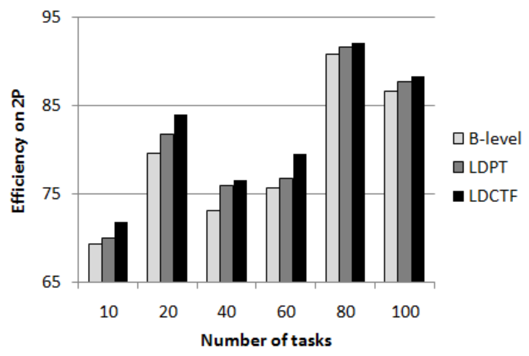

Fig. 36. Efficiency on 2 processors

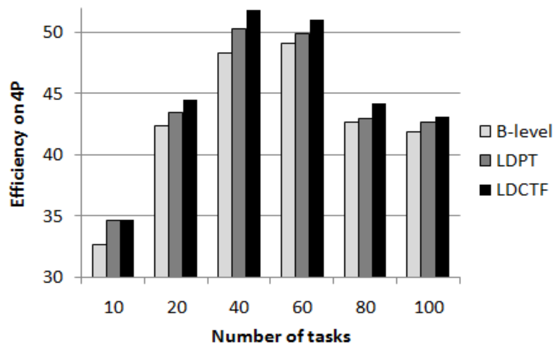

Fig. 37. Efficiency on 4 processors

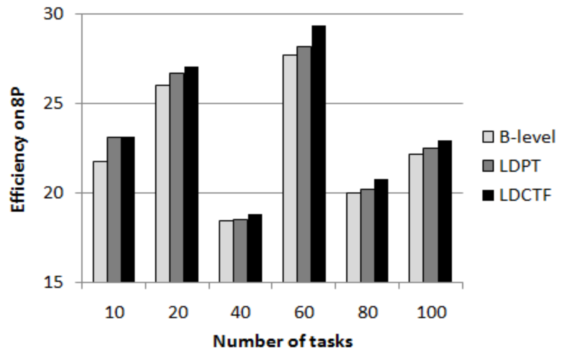

Fig. 38. Efficiency on 8 processors

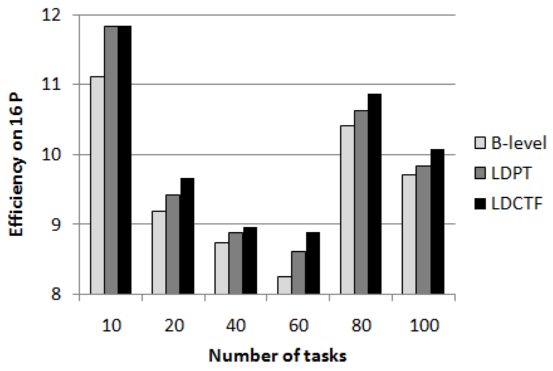

Fig. 39. Efficiency on 16 processors

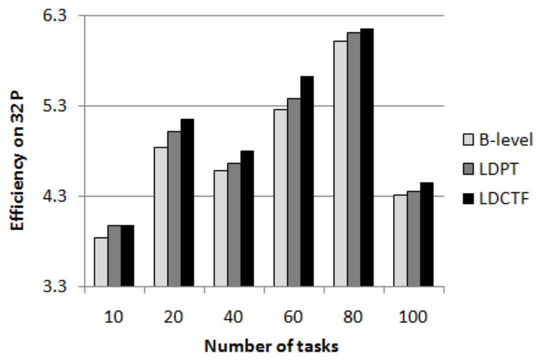

Fig. 40. Efficiency on 32 processors

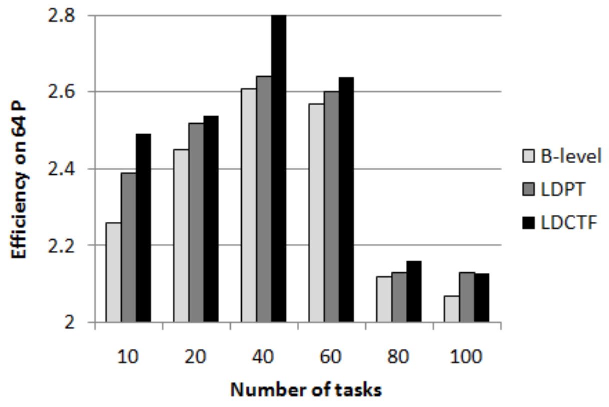

Fig. 41. Efficiency on 64 processors

Figure 36, 37, 38, 39, 40, 41 depicts efficiency versus number of processors with varying number of tasks (20, 40, 60, 80, 100). It is shown that LDCTF algorithm is more efficient and provides better performance than LDPT and B-level algorithms.

D. *Discussion of Results*

First, LDCTF algorithm is compared to LDPT algorithm and it provided better results in terms of schedule length, speed up, and efficiency. This is because in case of LDCTF, the critical path is taken into account and the critical task will be scheduled first in each level. This means that the task with the highest computation and communication cost will be scheduled first resulting in minimum schedule length, higher speed up, and higher efficiency.

Finally, LDCTF is compared to B-level algorithm and it provided better results in terms of schedule length, speed up, and efficiency. This is because B-level algorithm depends on paths idea and this will increase the communication overhead during assigning tasks on processors. On the other side, LDCTF algorithm depends on levels idea that will minimize the communication overhead during assigning tasks on processors. Another reason is that B-level algorithm must calculate the b-level value for each task before scheduling so that, the arithmetic calculation in LDCTF is less than B-level algorithm which leads to minimize the complexity factor.

V. Conclusion and Future Work

In this paper, a new static scheduling algorithm (LDCTF) is developed for homogeneous distributed computing systems. The performance of LDCTF algorithm is compared with LDPT algorithm. LDCTF is evaluated for different DAGs and found to be giving better results than LDPT algorithm in terms of schedule length, speed up, and efficiency with improving ratio 2.75%, 3.2%, and 1.9% respectively.

The performance of LDCTF is also compared with B-level and LDPT algorithms and found to be giving better results in terms of schedule length, speed up, and efficiency. LDCTF, LDPT, and B-level algorithms are applied on Standard Task Graph STG as a bench mark, and it was found that LDCTF algorithm is more efficient than LDPT and B-level algorithms.The future scope of the idea can be as follows:

- In this paper LDCTF algorithm is applied on Directed Acyclic Graph (DAG). In the future it can be applied on Directed Cyclic Graph (DCG).
- LDCTF can be applied on Heterogeneous Distributed Computing Systems (HDCS).
- LDCTF can be applied in a dynamic strategy instead of static strategy.
- Finally, duplication technique can be applied with LDCTF algorithm to minimize the communication overhead.

References

[1] Journal of Theoretical and Applied Information Technology. (2011, April 9). [Online]. Available: http://www.jatit.org/distributed-computing/grid-vs-distributed.htm.

[2] H.Topcuoglu, S. Hariri, and M.Y. Wu, “Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,” IEEE Trans. Parallel and Distributed Systems, Vol. 13, No.3, pp. 260-274, March 2002.

[3] Solomon Raju Kota, Chandra Shekhar, Archana Kokkula, Durga Toshniwal, M. V. Kartikeyan And R. C. Joshi, "Parameterized Module Scheduling Algorithm For Reconfigurable Computing Systems" In 15th International Conference On Advanced Computing And Communications, PP 473-478, 2007.

[4] Luiz F. Bittencourt, RizosSakellariou. "DAG Scheduling Using a Look ahead Variant of the Heterogeneous Earliest Finish Time Algorithm", 18th Euromicro International Conference onParallel, Distributed and Network-BasedProcessing(PDP), pp. 27-34, 2010.

[5] Eswari, R. and Nickolas, S. "Path-Based Heuristic Task Scheduling Algorithm for Heterogeneous Distributed Computing Systems".Advances in Recent Technologies in Communication and Computing (ARTCom), International Conference on 2010. P: 30-34.

[6] Rajak and Ranjit. "A Novel Approach for Task Scheduling in Multiprocessor System".International Journal of Computer Applications (IJCA), Vol.44, No. 11, pp. 12-16.April 2012.

[7] Ahmad, S.G.; Munir, E.U. and Nisar, W. PEGA "A Performance Effective Genetic Algorithm for Task Scheduling in Heterogeneous Systems".High Performance Computing and Communication& 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), IEEE 14th International Conference on 2012. Pp. 1082-1087.

[8] Tang, X., et al., "List scheduling with duplication for heterogeneous computing systems", Journal of Parallel and Distributed Computing (JPDC), Vol. 70, No.4,pp. 323-329.2010.

[9] Nasri,W. and Nafti, W. "A new DAG scheduling algorithm for heterogeneous platforms". Parallel Distributed and Grid Computing (PDGC), second IEEE International Conference on 2012. Pp. 114-119.

[10] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," IEEE Software, vol. 5, no. 1, pp. 23-32, May 1988.

[11] J. J. Hwang. Y.C. Chow. F. D. Anger and C.-Y. Lee. "Scheduling precedence graphs In systems with interprocessor communication times." SLAM Journal of Computing, vol. 18, no. 2. pp. 244-257. 1989.

[12] G.C. Slh and E. A. Lee. "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures." IEEE Transactions on Parallel and Distributed Systems, vol. 4. no. 2, pp. 75-87. Feb. 1997.

[13] H. El-Rewini and T.G .Lewis, " Scheduling parallel programs onto arbitrary target machines." Journal of Parallel and Distributed Computing, vol. 9. no. 2, pp. 138-153, June 1990.

[14] Panos M. Pardalos, SanguthevarRajasekaran, José D. P. Rolim, " Randomization Methods in Algorithm Design: DIMACS Workshop", vol. 43, pp. 12-14, December 1997.

[15] M. Y. Wu and D. D. Gajski, "Hypercool: a programming aid for message passing systems," IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 3 pp. 330-343, July 1990.

[16] A. Radulesu, J. C. Arjan and V. Gemund. "Low-cost task scheduling fordistributed-memory machines," IEEE Transactions on Parallel andDistributed Systems. vol. 13.no. 6. pp. 648-658. June 2002.

[17] T. Hagras and J. Janecek. "A high performance, low complexity algorithmfor compile-timejobscheduling in homogeneous computing environments."Proceedings of the International Conference on Parallel ProcessingWorkshops (ICPPW'03), pp 149.155. Oct. 2003.

[18] Amal EL-Nattat, Nirmeen A. El-Bahnasawy, Ayman EL-Sayed, "A new task scheduling algorithm for maximizing the distributed systems efficiency"; International Journal of Computer Applications, vol.110. no. 9, January 2015.

[19] http://www.kasahara.elec.waseda.ac.jp/schedule/index.html.

[20] F. Xhafa, L. Barolli and A. Durresi, “Batch Mode Schedulers for Grid Systems.International Journal of Web andGrid Services”, Vol. 3, No. 1, pp.19-37, 2007.