Chapter 5

Effective Critical Path Scheduling Algorithm

This chapter attempts to propose another clustering-based task scheduling algorithm called Effective Critical Path (ECP) for multiprocessor environments. The proposed algorithm applies the idea of edge zeroing on the critical path of a task graph for obtaining minimization of makespan. The time complexity of the ECP algorithm is $O(|V|^2(|V|+|E|))$, where $|E|$ is the number of edges and $|V|$ is the number of tasks in the task graph. The contributions here may be indicated as follows:

- Our idea of applying edge zeroing concept on the critical path leads to reduction in the communication time among the tasks of a given task graph and finally provides a meaningful clustering.
- Makespan being a significant measure for scheduling applications in multiprocessor environments, the proposed algorithm provides an alternative approach towards its minimization.
- This work demonstrates improvement in performance of the proposed ECP over four well-known algorithms such as EZ [3], LC [23], CPPS [27], and LOCAL [71] and the algorithm proposed in the previous chapter 4.
- The results of the simulation carried out herein show distinctively better normalized schedule length and speedup for randomly generated benchmark task

graphs [79] and task graphs generated from real-world applications such as Gaussian Elimination, Fast Fourier Transform (FFT) and Systolic Array.

- A statistical analysis, using confidence intervals, is performed to know the significance of the obtained results

5.1 The ECP Algorithm

In this section, we introduce the proposed clustering-based scheduling algorithm named ECP for an unbounded number of processors for the task scheduling problem on the multiprocessor computing systems. Our proposed clustering based solution, to the scheduling problem, is implemented in Algorithm 4 below which makes use of Algorithms 2, and 3 and executes a chain of clustering refinement steps. The earliest step allocates each task of the task graph to a distinct cluster. At each step, the algorithm tries to improve its earlier clustering by merging suitable clusters into one. A merging operation is carried out by zeroing an edge cost linking two clusters. The main aim of the ECP algorithm is to minimize the makespan taking into consideration the precedence constraints. This algorithm uses the concept of edge zeroing on the critical path of the task graph to group the tasks. The idea of edge zeroing was initially given by Sarkar [3]. Sarkar′s algorithm examines the edges one by one from sorted list and performs edge zeroing if makespan does not increase. In Sarkar′s algorithm, the edge which is zeroed may not be on a path that determines the makespan. Here in the proposed algorithm, we repeatedly compute a critical path of the task graph and perform the edge zeroing steps only on the critical path edges with the aim of reducing makespan. The proposed algorithm has the following characteristics:

- it zeroes an edge of a critical path at each step, producing a new critical path, thereafter,
- it uses backtracking, after zeroing an edge at each step, when current makespan is more than the makespan in the previous step,
- it dynamically updates the schedule for each processor until makespan of the task graph does not increase, and

- it implicitly gives the feasible schedule of the clustering obtained at each step. Otherwise, current makespan would not be determined, on which clustering decisions are based.

In the following, we talk about a number of the concepts utilized in the design of our algorithm. In the first subsection, we describe the computation of the critical path and a method to select the edge for zeroing. In the second subsection, we discuss the method for merging of clusters. We describe the proposed algorithm in the third subsection. In the fourth subsection, the analysis about the complexity of the algorithms are presented, An illustrative example for proposed algorithm is given at the end of this section.

5.1.1 Critical Path Computation and Edge Selection

As given in Definition 2.5, a critical path of a task graph is the path from entry node to exit node that has the maximum sum of the execution and communication times. It determines the partial makespan of a task graph because the sum of computation time of the tasks belongs to a CP provides the lower bound on the makespan. In the proposed solution, at each step of the scheduling process, the CP of a task graph may change dynamically, because an edge on a CP in a particular step may not remain on the CP at the next step due to edge zeroing. We call the intermediate CP obtained during scheduling steps as the Effective Critical Path (ECP) to distinguish it from the initial CP of the unscheduled task graph. The ECP is computed as shown in Algorithm 2.

After the ECP computation, we need an approach to select an appropriate edge on the ECP for zeroing. To select an edge, we first sort the edges of the ECP according to their CT values in non-increasing order by using heap sort. Then choose edges from left to the right in the sorted list until schedule length does not reduce. If two edges have same CT value on the ECP, the order between edges is decided by the sum of the execution time of the associated tasks. The edge which has lower value will get higher order than the other in the sorted list. If this value is equal for both edges, the order is decided according to FCFS in the ECP. By doing this, we are giving lower priority to the computation-intensive tasks in comparison to

Algorithm 2 Algorithm for the ECP computation

1: topo_list $\leftarrow$ a list of all tasks $T_i \in V$ sorted in a reverse topological order
2: **for** each task T_i in topo_list **do**
3: Compute $BL(T_i)$ as Eq. 2.2
4: Store successor of T_i in $ecp\_succ(T_i)$ from which $BL(T_i)$ is computed
5: **end for**
6: $ecp[0] \leftarrow T_i$, where $T_i \in V$ with maximum $BL(T_i)$
7: $T_{current} \leftarrow T_i$
8: $count \leftarrow 0$
9: **while** $ecp\_succ(T_{current}) \neq \phi$ **do**
10: $ecp\_edges[count] \leftarrow e_{current,ecp\_succ(T_{current})}$
11: $count \leftarrow count + 1$
12: $ecp[count] \leftarrow ecp\_succ(T_{current})$
13: $T_{current} \leftarrow ecp\_succ(T_{current})$
14: **end while**

communication-intensive. Computation-intensive tasks are those tasks which spend more time in performing computation than communication.

5.1.2 Clustering

While we are able to identify an appropriate edge on an ECP for zeroing, we still require a method to merge two clusters after zeroing an edge. When two clusters are merged, their communication becomes local, and tasks of the cluster are ordered according to Algorithm 3.

The algorithm first verifies that whether the tasks of one cluster are descendant of other cluster's tasks. If yes, there is no need to order them explicitly. Otherwise, the algorithm compares their bottom level and adds a pseudo edge with zero communication time from the task having higher BL value to the task having lower BL value. If the BL value of the tasks comes out to be equal, order the tasks according to their topological order in their clusters and add a pseudo edge with zero communication time from the higher order task to the lower order task.

Algorithm 3 Algorithm for the merging of two clusters, C_1 and C_2

```
1: Let l1 is a list of tasks of C1 and l2 is a list of tasks of C2
2: i ← 0, j ← 0
3: while i ≤ l1.size and j ≤ l2.size do
4:     if l1[i] is descendant of l2[j] then
5:         j++
6:     else if l2[j] is descendant of l1[i] then
7:         i++
8:     else if BL(l1[i]) < BL(l2[j]) then
9:         add pseudo edge from l2[j] to l1[i] with zero communication time
10:        j++
11:    else if BL(l2[j]) < BL(l1[i]) then
12:        add pseudo edge from l1[i] to l2[j] with zero communication time
13:        i++
14:    else if l2[j] comes earlier in topological order than l1[i] then
15:        add pseudo edge from l2[j] to l1[i] with zero communication time
16:        j++
17:    else
18:        add pseudo edge from l1[i] to l2[j] with zero communication time
19:        i++
20:    end if
21: end while
22: Make Cluster C2 as Cluster C1
```

5.1.3 The ECP Algorithm

In this subsection, we formalize the proposed ECP algorithm in Algorithm 4. As stated earlier this algorithm makes use of Algorithms 2, and 3 for its purpose as indicated in the Algorithm 4 given below.

The algorithm starts with initial clustering of each task of the task graph at line 1. It then computes the initial schedule length of the unscheduled task graph. The algorithm repeats the steps from line 3 to 14, until schedule length does not increase. In these steps, the algorithm computes ECP of partially scheduled task graph via algorithm 1 and sorts the edges of the ECP according to their communication time by using heap sort. Then it selects the edges of the ECP from left to right in the sorted list and verifies whether both tasks of an edge belong to a cluster or not. Perform edge zeroing if both tasks of an edge are in different clusters and schedule length does not increase after clustering, otherwise do verification for next edges.

Algorithm 4 The ECP Algorithm

```
1: Initially, each task forms a separate cluster
2: Compute initial schedule length
3: repeat
4:     Compute the ECP via Algorithm 2
5:     Sort the edges of the ECP in non-increasing order according to their com-
   munication time
6:     for all edges of ECP in the sorted list do
7:         if both tasks of an edge belongs to same cluster then
8:             continue
9:         else
10:            Zero an edge if schedule length does not increase
11:            When two clusters are merged, use Algorithm 3
12:            Update schedule length
13:        end if
14:    end for
15: until schedule length does not increase
```

Whenever edge zeroing is performed, clusters are merged according to Algorithm 3 and schedule length is updated.

5.1.4 Algorithm Complexity Analysis

In this section, we present an analysis of the time complexity of the algorithms discussed above.

5.1.4.1 Analysis of Algorithm 2

This algorithm finds the ECP of a task graph. The topological sorting in line 1 can be performed in $O(|V| + |E|)$ time. Line 2 in the for loop iterates through the edges coming out of a task T_i. Lines 3 and 4 take constant time. Therefore, the total number of iterations for lines 2 to 5 occurs $|E|$ times. Line 6 to 8 takes constant time. The while loop from lines 9 to 14 find the tasks and edges of the ECP and execute $|V_{ecp}|$ times where V_{ecp} is the number of tasks on the ECP. In the worst case, this while loop will execute $|V|$ times when all tasks on the ECP. Thus, the upper bound of this algorithm is $O(|V| + |E|)$.

5.1.4.2 Analysis of Algorithm 3

This algorithm finds the ECP of a task graph. Line 2 is an initialization that can execute in constant time. The while loop beginning at line 3 will execute at most $|V|$ times, and the steps from lines 4 to 20 will take constant time. Thus, the total number of iterations for lines 3 to 21 occurs $|V|$ times. Line 22 will execute $|V|$ times. Thus, the upper bound of this algorithm is $O(|V|)$.

5.1.4.3 Analysis of Algorithm 4

Algorithm 3 is the proposed ECP algorithm computing the final schedule for a task graph. Line 1 is an initialization that can be done in $|V|$ times. Line 2 computes the initial schedule length in $(|V| + |E|)$ times. In line 4, the ECP can be computed via Algorithm 2 in $O(|V| + |E|)$ time. The sorting in line 5 can be performed via heap sort and completed in $O(|V|lg|V|)$ time. The for loop beginning at line 6 will execute at most $|V|$ times when all tasks belong to an ECP. Line 10 can be executed in $(|V| + |E|)$ times. In line 11, when two clusters are merged, a call is issued to Algorithm 3 and can be completed in $O(|V|)$ time. Line 12 updates schedule length in constant time. The total number of iterations for lines 3 to 15 occurs $|V|$ times. Thus, the complexity of the ECP algorithm is $O(|V|^2(|V| + |E|))$.

5.1.5 An Illustrative Example

Consider the task graph given in Fig. 2.1. The graph consists of fifteen tasks labeled T_0 to T_{14} with their execution times. The tasks T_0 and T_{14} are the entry and exit tasks of the task graph respectively which represent the starting and ending of the application. The edges of the graph are labeled with the communication times.

At each step of the execution of the algorithm, the ECP of the task graph is determined, and edges of the ECP are examined one-by-one in non-increasing order of their communication times and corresponding tasks (clusters) of that edge are merged, if schedule length does not increase. Clustering steps of the example task graph in Fig. 2.1 with the ECP algorithm is shown in Fig. 5.1.

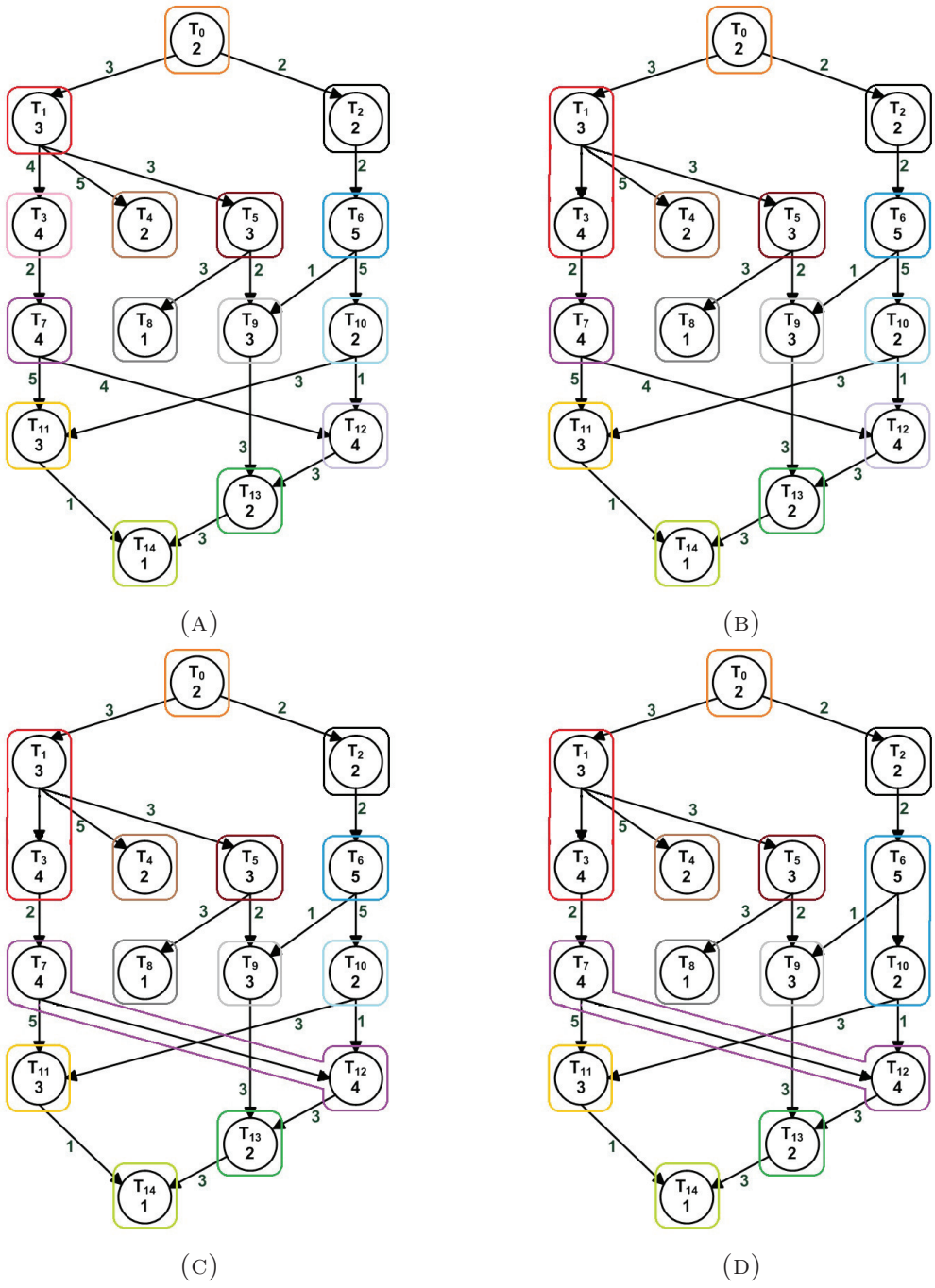

FIGURE 5.1: Clustering steps of the example task graph in Fig. 1 with the ECP algorithm (a) Initial clustering (initial schedule length = 39), (b) clustering after merging T_1 and T_3 (partial schedule length = 35), (c) clustering after merging T_7 and T_{12} (partial schedule length = 34), (d) clustering after merging T_6 and T_{10} (partial schedule length = 31).

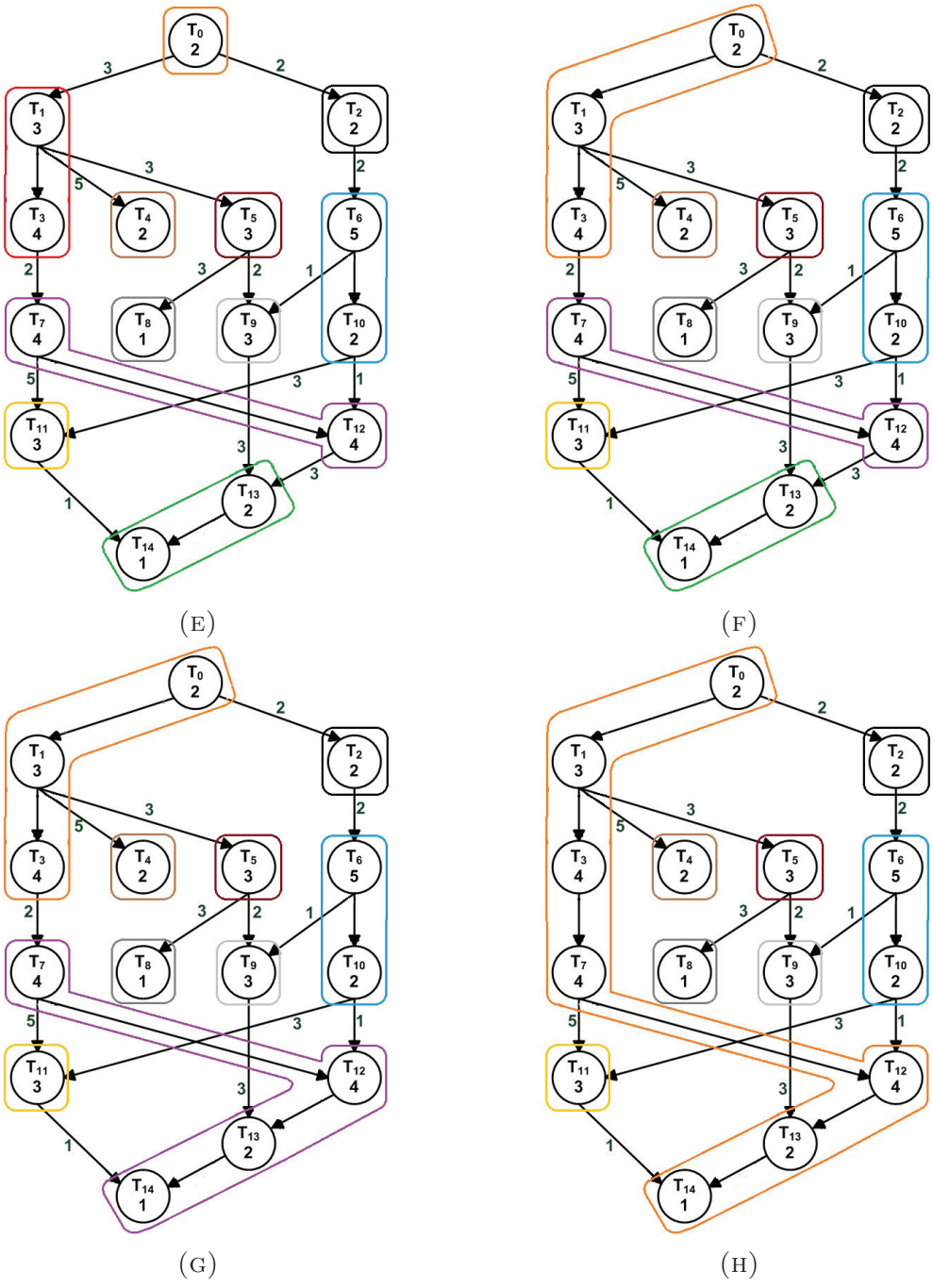

FIGURE 5.1: Clustering steps of the example task graph in Fig. 1 with the ECP algorithm (e) clustering after merging T_{13} and T_{14} (partial schedule length = 28), (f) clustering after merging T_0 and $\{T_1, T_3\}$ (partial schedule length = 26), (g) clustering after merging $\{T_7, T_{12}\}$ and $\{T_{13}, T_{14}\}$ (partial schedule length = 25), (h) clustering after merging $\{T_0, T_1, T_3\}$ and $\{T_{12}, T_{13}, T_{14}\}$ (final schedule length = 23).

Initially, each task is assumed to be in a distinct cluster as shown in Fig. 5.1a, in

which clusters are shown with different color boxes. Schedule length for the initial clustering comes out to be 39.

Fig. 5.1b-Fig. 5.1g show the intermediate clusters and the partial schedule length of the task graph. In Fig. 5.1h, the schedule length comes out to be 23 that can′t be reduced after merging any edge in the ECP. Thus, it is the final schedule length of the task graph, and the clustering obtained in this step reflects the final schedule of the example task graph.

5.2 Experimental Results and Discussion

This section provides the performance evaluation of the ECP algorithm with four well-known clustering-based task scheduling algorithms, the EZ, the LC, the CPPS, and the LOCAL and the algorithm proposed in the previous chapter 4, using various metrics. For this purpose, two types of task graphs are considered: (i) randomly generated and (ii) derived from real-world applications. The metrics used for comparison of the algorithms are same as used in Chapter 4. In this chapter, we are not considering DCCL and RDCC for the performance comparison with ECP. The reason is that both algorithms, DCCL and RDCC, perform worst for all types of graphs in the previous chapter. The experiments are carried out on a Dell PowerEdge R420 server with CentOS (version 7.3-1611), Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20 GHz processor, and 192 GB of memory. A statistical analysis for different sample sizes is also performed for the obtained results of the algorithms. We exploit a confidence interval (CI) to characterize the results and use 95 % as the confidence level. For the estimation of confidence interval, t-values are used from Table 4.1. For each type of task graph, we will not present a statistical analysis of the results of the algorithms in this chapter which have already been presented in the chapter 4.

5.2.1 Randomly Generated Task Graphs

In [79], authors proposed a set of 180 benchmark random task graphs for comparison and analysis of scheduling algorithms. The graphs are divided into 6 subsets each

of which contains 30 graphs and having the number of nodes as 50, 100, 200, 300, 400, and 500 respectively.

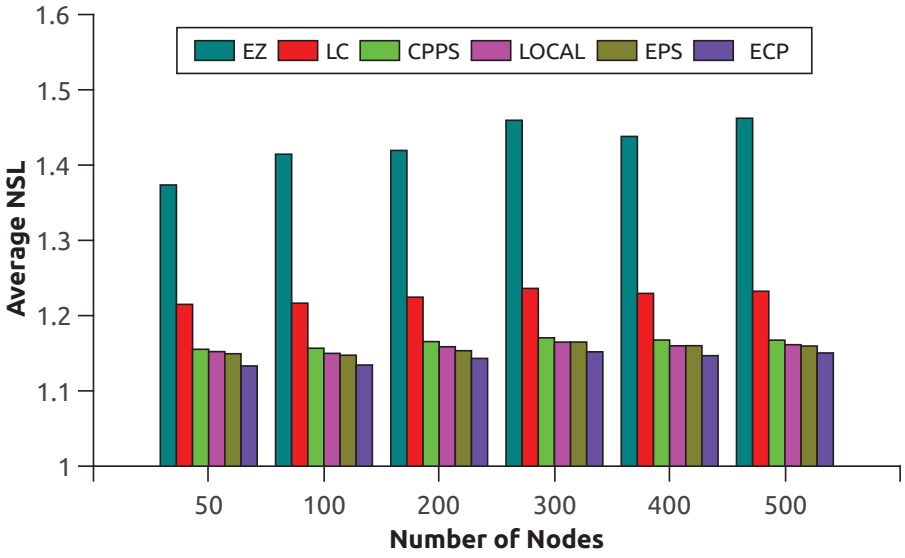

FIGURE 5.2: Average NSL results obtained for random task graphs as a function of number of nodes.

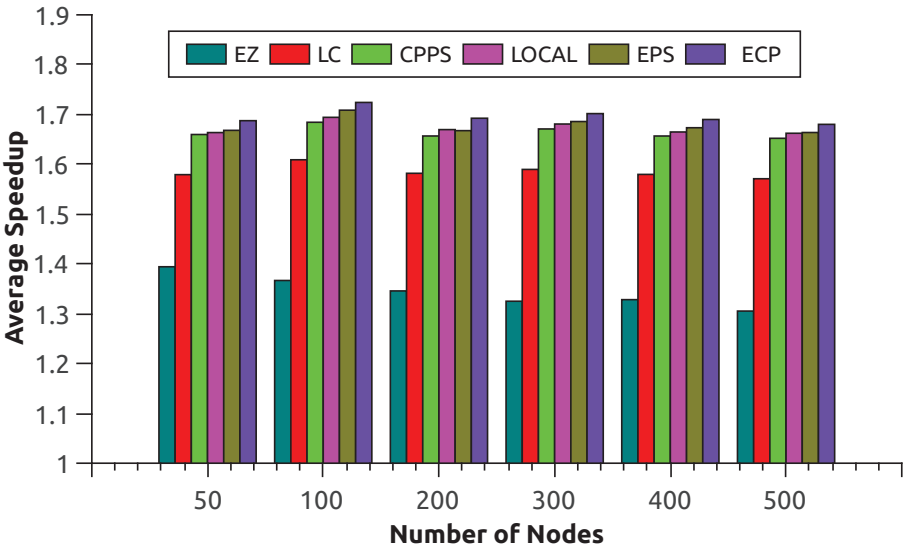

FIGURE 5.3: Average Speedup results obtained for random task graphs as a function of number of nodes.

The average NSL results of random graphs for the different number of nodes are shown in Fig. 5.2. For each set of task graphs, the ECP algorithm gives better

schedules than other algorithms. Overall, the ECP algorithm provides an improvement of 19.93, 6.72, 1.76, 1.25 and 1.08 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average Speedup results of random graphs for the different number of nodes are shown in Fig. 5.3. For each set of task graphs, the ECP algorithm produces better speedup than other algorithms. Overall, the ECP algorithm provides an improvement of 20.74, 6.54, 1.92, 1.38 and 1.07 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

Along with the above average results for random task graphs, a statistical analysis of the obtained results is also presented. We use confidence intervals to show the significance of the results. Table 5.1 and Table 5.2 present the statistical analysis of NSL and speedup results of ECP for random task graphs respectively. The results in both tables are presented for confidence level 95 %.

5.2.2 Real-World Application Graphs

Besides randomly generated graphs, we evaluated and compared the performance of the algorithms concerning real-world applications, namely Gaussian Elimination [10, 11, 39], Fast Fourier Transform [10, 40], and Systolic Array [41]. All of these applications are well-known and used in real-world problems.

5.2.2.1 Gaussian Elimination

The values of matrix size used in this experiment are varied from 5 to 30 with an interval of 5. As the structure of this graph is known, we use different values of CCR as [0.1, 1, 10] to carry out experiments.

The average NSL results of Gaussian Elimination graphs for different matrix sizes are shown in Fig. 5.4. For each set of task graphs, the ECP algorithm gives better schedules than other algorithms. Overall, the ECP algorithm provides an improvement of 18.90, 34.04, 11.25, 9.45 and 8.95 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

TABLE 5.1: Statistical analysis of NSL results of ECP for random task graphs.

Algo	**Nodes**	**Sample**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Min**
ECP	50	10	1.133387	0.115791	1.050560 - 1.216213	1.075697
		20	1.132999	0.110300	1.081377 - 1.184620	1.082290
		30	1.133128	0.110145	1.092004 - 1.174252	1.102667
	100	10	1.132323	0.052146	1.095023 - 1.169623	1.098936
		20	1.135391	0.066957	1.104054 - 1.166727	1.057544
		30	1.134368	0.061509	1.111403 - 1.157333	1.121791
	200	10	1.162592	0.087368	1.100097 - 1.225087	1.102312
		20	1.133390	0.066820	1.102118 - 1.164662	1.115665
		30	1.143124	0.074096	1.115459 - 1.170789	1.115665
	300	10	1.159318	0.076506	1.104593 - 1.214043	1.05425
		20	1.148133	0.085147	1.108283 - 1.187982	1.119893
		30	1.151861	0.081211	1.121540 - 1.182183	1.138346
	400	10	1.162416	0.094516	1.094809 - 1.230024	1.098137
		20	1.138937	0.075875	1.103427 - 1.174447	1.111111
		30	1.146763	0.081676	1.116269 - 1.177258	1.121101
	500	10	1.165496	0.083342	1.105881 - 1.225111	1.138908
		20	1.142839	0.068997	1.110547 - 1.175130	1.114178
		30	1.150391	0.073435	1.122973 - 1.177809	1.127503

The average NSL results of Gaussian Elimination graphs for different values of CCR are shown in Fig. 5.5. For CCR = 0.10, the ECP algorithm provides an improvement of 3.94 percent over the EZ algorithm. For CCR = 1, the ECP algorithm provides an improvement of 19.84, 5.07, 2.92, 2.04 and 0.66 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 22.18, 45.65, 15.48, 11.57 and 10.99 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average speedup results of Gaussian Elimination graphs for different matrix sizes are shown in Fig. 5.6. For each set of task graphs, the ECP algorithm gives better schedules than other algorithms. Overall, the ECP algorithm gives an improvement of 13.46, 8.47, 4.08, 4.19 and 2.55 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

TABLE 5.2: Statistical analysis of speedup results of ECP for random task graphs.

Algo	**Nodes**	**Sample**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Max**
ECP	50	10	2.376304	0.579798	1.961570 - 2.791038	2.667742
		20	1.343193	0.219397	1.240513 - 1.445873	1.426446
		30	1.687563	0.617434	1.457036 - 1.918091	1.818182
	100	10	2.507922	0.669565	2.028978 - 2.986867	2.952381
		20	1.332322	0.191353	1.242767 - 1.421877	1.412903
		30	1.724188	0.693421	1.465290 - 1.983087	1.909003
	200	10	2.469515	0.666359	1.992863 - 2.946166	2.570565
		20	1.303939	0.173983	1.222513 - 1.385365	1.352092
		30	1.692464	0.685529	1.436512 - 1.948416	1.874486
	300	10	2.451477	0.642766	1.991701 - 2.911252	2.570565
		20	1.326904	0.197707	1.234376 - 1.419433	1.352092
		30	1.701762	0.666749	1.452821 - 1.950702	1.874486
	400	10	2.459733	0.603382	2.028130 - 2.891337	2.890966
		20	1.305052	0.182722	1.219537 - 1.390567	1.377115
		30	1.689946	0.664353	1.441900 - 1.937991	1.889175
	500	10	2.401550	0.580784	1.986110 - 2.816989	2.758387
		20	1.319485	0.192120	1.229571 - 1.409399	1.392244
		30	1.680173	0.630896	1.444619 - 1.915727	1.880550

The average speedup results of Gaussian Elimination graphs for different values of CCR are shown in Fig. 5.7. For CCR = 0.10, the ECP algorithm provides an improvement of 4.14 percent over the EZ algorithm. For CCR = 1, the ECP algorithm provides an improvement of 20.96, 5.30, 3.52, 2.88 and 1.34 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 21.95, 43.93, 14.04, 9.37 and 6.57 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

Along with the above average results for Gaussian Elimination task graphs, a statistical analysis of the obtained results of ECP is also presented. We use confidence intervals to show the significance of the results. Table 5.3 and Table 5.4 present the statistical analysis of NSL results of ECP for Gaussian Elimination task graphs with

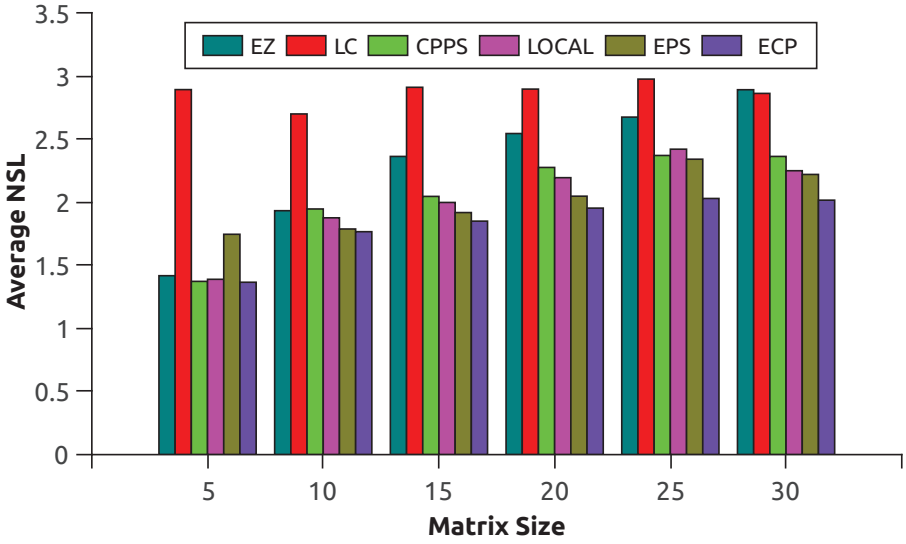

FIGURE 5.4: Average NSL results obtained for Gaussian Elimination task graphs as a function of matrix size.

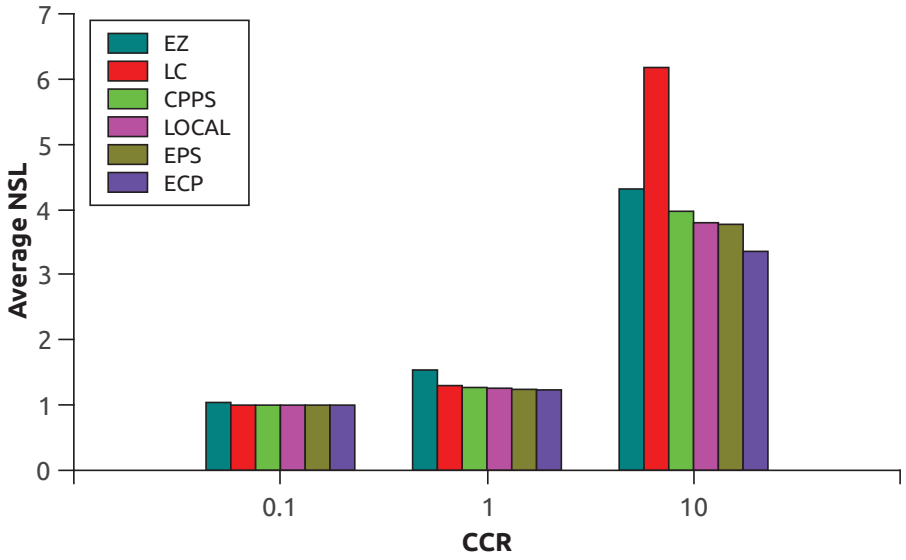

FIGURE 5.5: Average NSL results obtained for Gaussian Elimination task graphs as a function of CCR.

CCR value 1 and 10 respectively. For Gaussian Elimination task graphs with CCR value 0.1, all algorithms except EZ have NSL values equal to 1. EZ has slightly greater NSL values in comparison to others. Thus, we are not presenting the statistical analysis table for this case. Table 5.5, Table 5.6 and Table 5.7 present the statistical analysis of speedup results of ECP for Gaussian Elimination task graphs

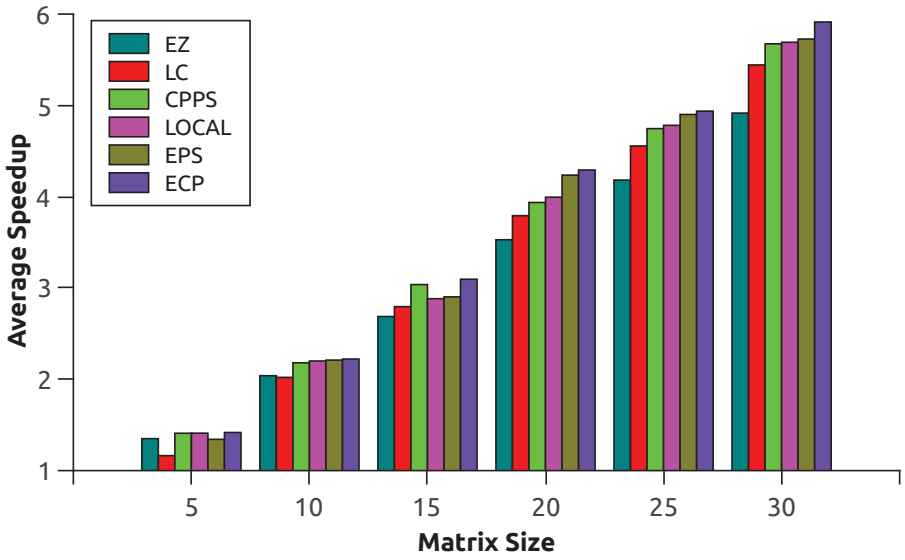

FIGURE 5.6: Average Speedup results obtained for Gaussian Elimination task graphs as a function of matrix size.

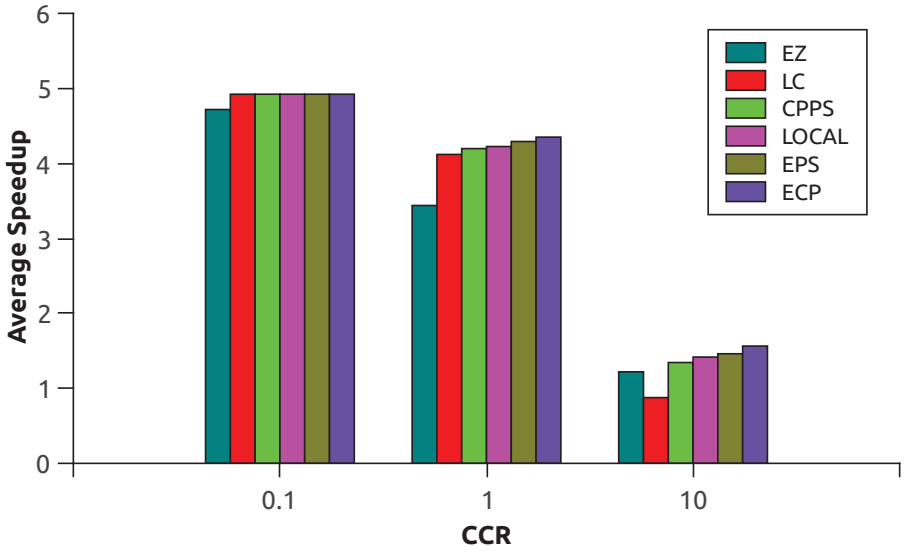

FIGURE 5.7: Average Speedup results obtained for Gaussian Elimination task graphs as a function of CCR.

with CCR value 0.1, 1 and 10 respectively. The results in these tables are presented for confidence level 95 % and sample size 10.

Table 5.3: Statistical analysis of NSL results of ECP for Gaussian Elimination task graphs with CCR value 1.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Min
ECP	5	1.181414	0.067026	1.133470 - 1.229358	1.160714
	10	1.248178	0.053935	1.209597 - 1.286758	1.229249
	15	1.228282	0.057918	1.186853 - 1.269711	1.193050
	20	1.234118	0.034252	1.209617 - 1.258618	1.218274
	25	1.255600	0.045557	1.223013 - 1.288188	1.243792
	30	1.243591	0.037688	1.216633 - 1.270549	1.220796

Table 5.4: Statistical analysis of NSL results of ECP for Gaussian Elimination task graphs with CCR value 10.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Min
ECP	5	2.232636	0.331204	1.995723 - 2.469549	2.020202
	10	3.055568	0.208813	2.906202 - 3.204933	3.068293
	15	3.492733	0.225209	3.331640 - 3.653826	3.360759
	20	3.638267	0.263412	3.449847 - 3.826688	3.452096
	25	3.789085	0.126596	3.698530 - 3.879640	3.787307
	30	3.852953	0.206340	3.705356 - 4.000550	3.767857

Table 5.5: Statistical analysis of speedup results of ECP for Gaussian Elimination task graphs with CCR value 0.1.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
ECP	5	2.046907	0.275074	1.850145 - 2.243669	2.216749
	10	3.107596	0.293095	2.897943 - 3.317249	3.271263
	15	4.228850	0.310621	4.006660 - 4.451039	4.302242
	20	5.531130	0.320292	5.302023 - 5.760237	5.683466
	25	6.733947	0.306428	6.514756 - 6.953137	6.798757
	30	7.766287	0.312919	7.542454 - 7.990120	7.786150

5.2.2.2 Fast Fourier Transform

The structure of FFT is known; hence, we use different values for input points as [2, 4, 8, 16, 32] and CCR as [0.1, 1, 10] to carry out experiments.

TABLE 5.6: Statistical analysis of speedup results of ECP for Gaussian Elimination task graphs with CCR value 1.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
ECP	5	1.779241	0.273709	1.583456 - 1.975027	1.956522
	10	2.609007	0.265933	2.418783 - 2.799231	2.731092
	15	3.764089	0.435542	3.452542 - 4.075635	4.041916
	20	4.902230	0.391155	4.622434 - 5.182026	5.097561
	25	5.923041	0.369982	5.658390 - 6.187692	6.148008
	30	6.949625	0.369950	6.684997 - 7.214253	7.160494

TABLE 5.7: Statistical analysis of speedup results of ECP for Gaussian Elimination task graphs with CCR value 10.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
ECP	5	1.001744	0.005516	0.997799 - 1.005690	1.017442
	10	1.103966	0.058733	1.061953 - 1.145978	1.136364
	15	1.352727	0.113130	1.271804 - 1.433650	1.380368
	20	1.687744	0.119892	1.601984 - 1.773503	1.763804
	25	2.000666	0.124770	1.911417 - 2.089915	2.010724
	30	2.229132	0.102849	2.155564 - 2.302701	2.246377

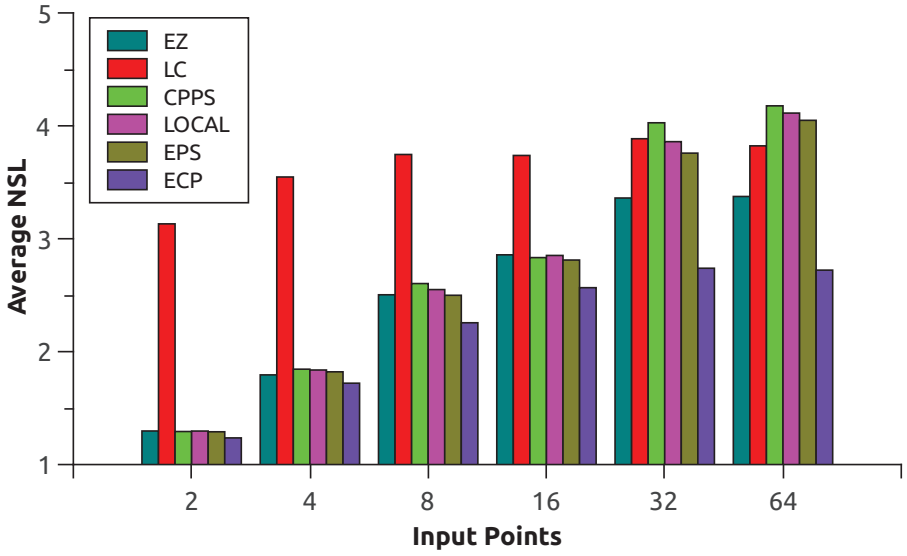

FIGURE 5.8: Average NSL results obtained for FFT task graphs as a function of input points.

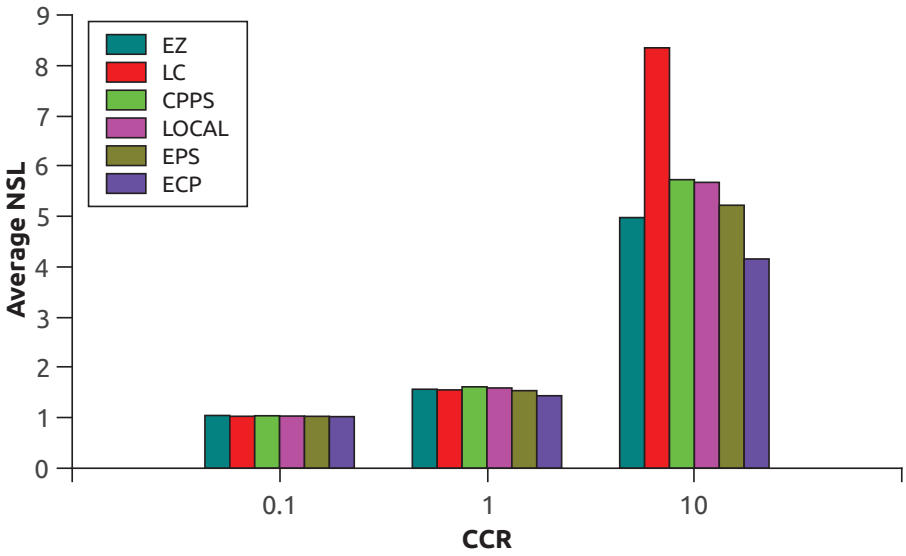

FIGURE 5.9: Average NSL results obtained for FFT task graphs as a function of CCR.

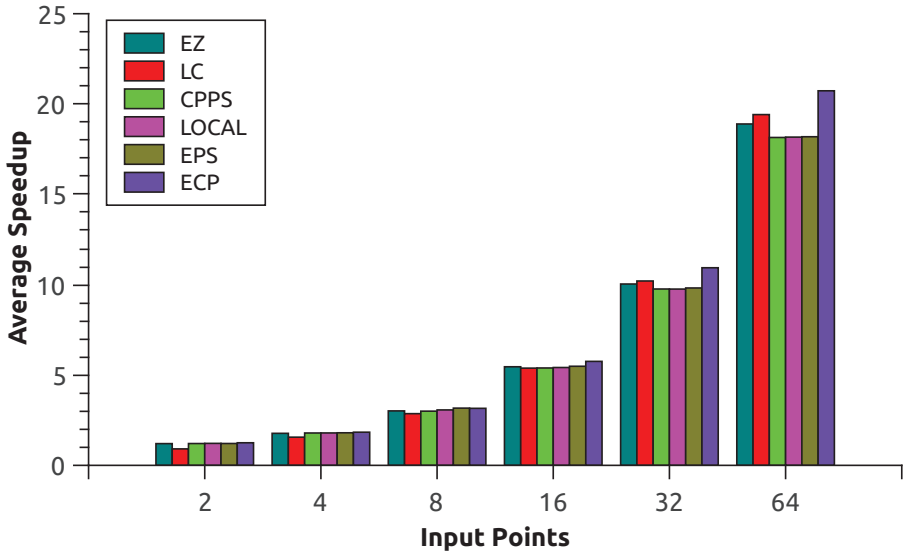

FIGURE 5.10: Average speedup results obtained for FFT task graphs as a function of input points.

The average NSL results of FFT graphs for different number input points are shown in Fig. 5.8. For each set of task graphs, the ECP algorithm produces better schedules than other algorithms. Overall, the ECP algorithm gives an improvement of 12.83, 39.47, 21.09, 19.80, and 18.42 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

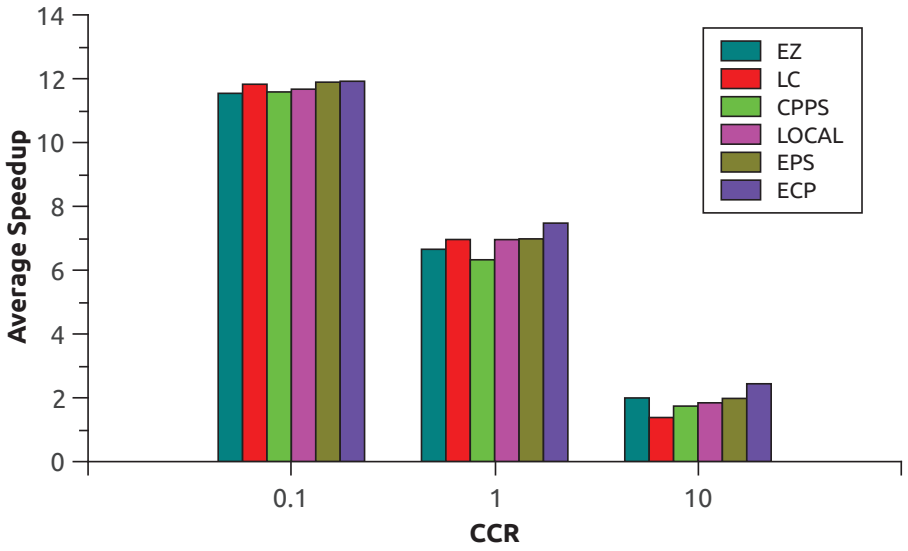

FIGURE 5.11: Average speedup results obtained for FFT task graphs as a function of CCR.

The average NSL results of FFT graphs for different values of CCR are shown in Fig. 5.9. For CCR = 0.10, the ECP algorithm provides an improvement of 2.23, 0.70, 1.68, 1.13 and 0.62 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 1, the ECP algorithm provides an improvement of 8.22, 7.46, 10.95, 9.69 and 6.55 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 16.51, 50.22, 27.48, 26.79 and 20.42 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average speedup results of FFT graphs for the different number of input points are shown in Fig. 5.10. For each set of task graphs, the ECP algorithm produces better schedules than other algorithms. Overall, the ECP algorithm gives an improvement of 7.53, 7.63, 10.00, 9.72 and 9.144 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average speedup results of FFT graphs for the different values of CCR are shown in Fig. 5.11. For CCR = 0.10, the ECP algorithm provides an improvement of 3.18, 0.79, 2.81 and 2.09 percent over the EZ, LC, CPPS and LOCAL scheduling algorithms respectively. For CCR = 1, the ECP algorithm provides an improvement of 10.97, 6.87, 15.37, and 5.98 percent over the EZ, LC, CPPS and LOCAL

scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 18.20, 43.35, 28.68, and 12.17 percent over the EZ, LC, CPPS and LOCAL scheduling algorithms respectively.

Along with the above average results for FFT task graphs, a statistical analysis of the obtained results is also presented. We use confidence intervals to show the significance of the results. Table 5.8 and Table 5.9 present the statistical analysis of NSL results of ECP for FFT task graphs with CCR value 1 and 10 respectively. For FFT task graphs with CCR value 0.1, all algorithms have almost similar NSL values and nearly equal to 1. Thus, we are not presenting the statistical analysis table for this case. Table 5.10, Table 5.11 and Table 5.12 present the statistical analysis of speedup results of ECP for FFT task graphs with CCR value 0.1, 1 and 10 respectively. The results in these tables are presented for confidence level 95 % and sample size 10.

Table 5.8: Statistical analysis of NSL results of ECP for FFT task graphs with CCR value 1.

Algo	**Input Points**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Min**
ECP	2	1.300748	0.162483	1.184522 - 1.416973	1.201342
	4	1.444000	0.133238	1.348694 - 1.539306	1.377049
	8	1.448141	0.130614	1.354712 - 1.541570	1.383663
	16	1.430542	0.095441	1.362272 - 1.498811	1.365079
	32	1.525637	0.094108	1.458321 - 1.592953	1.468165
	64	1.488624	0.077805	1.432969 - 1.544278	1.435567

Table 5.9: Statistical analysis of NSL results of ECP for FFT task graphs with CCR value 10.

Algo	**Input Points**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Min**
ECP	2	1.392771	0.220807	1.234827 - 1.550716	1.333333
	4	2.688784	0.301899	2.472834 - 2.904735	2.483444
	8	4.293009	0.377506	4.022977 - 4.563042	4.285714
	16	5.241556	0.670510	4.761935 - 5.721176	4.842105
	32	5.668365	0.430913	5.360130 - 5.976600	5.544014
	64	5.648005	0.612548	5.209845 - 6.086165	5.221239

Table 5.10: Statistical analysis of speedup results of ECP for FFT task graphs with CCR value 0.1.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
ECP	2	1.540020	0.038639	1.512381 - 1.567659	1.558326
	4	2.735846	0.055919	2.695847 - 2.775845	2.767528
	8	5.011157	0.108394	4.933622 - 5.088691	5.079448
	16	9.468638	0.170938	9.346365 - 9.590911	9.576613
	32	18.028484	0.444022	17.710872 - 18.346096	18.328767
	64	34.738064	0.441240	34.422441 - 35.053686	35.028791

Table 5.11: Statistical analysis of speedup results of ECP for FFT task graphs with CCR value 1.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
ECP	2	1.107189	0.067061	1.059220 - 1.155158	1.153846
	4	1.728200	0.161935	1.612367 - 1.844033	1.807229
	8	3.253840	0.118333	3.169195 - 3.338484	3.323864
	16	5.965139	0.270254	5.771824 - 6.158453	6.142241
	32	11.484607	0.442570	11.168033 - 11.80118	11.638831
	64	21.305718	1.080260	20.533000 - 22.078435	21.744681

Table 5.12: Statistical analysis of speedup results of ECP for FFT task graphs with CCR value 10.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
ECP	2	1.102043	0.081863	1.043486 - 1.160600	1.145038
	4	1.041988	0.034115	1.017586 - 1.066391	1.063830
	8	1.203016	0.063787	1.157389 - 1.248643	1.207430
	16	1.833249	0.115167	1.750869 - 1.915629	1.884921
	32	3.300531	0.162214	3.184498 - 3.416564	3.310789
	64	6.154693	0.286885	5.949482 - 6.359904	6.289231

5.2.2.3 Systolic Array

For systolic array graphs, the values of number of nodes, n, used in this experiment are varied from 5 to 20 with an interval of 5. As the structure of this graph is known, we use different values of CCR as [0.1, 1, 10] to carry out experiments.

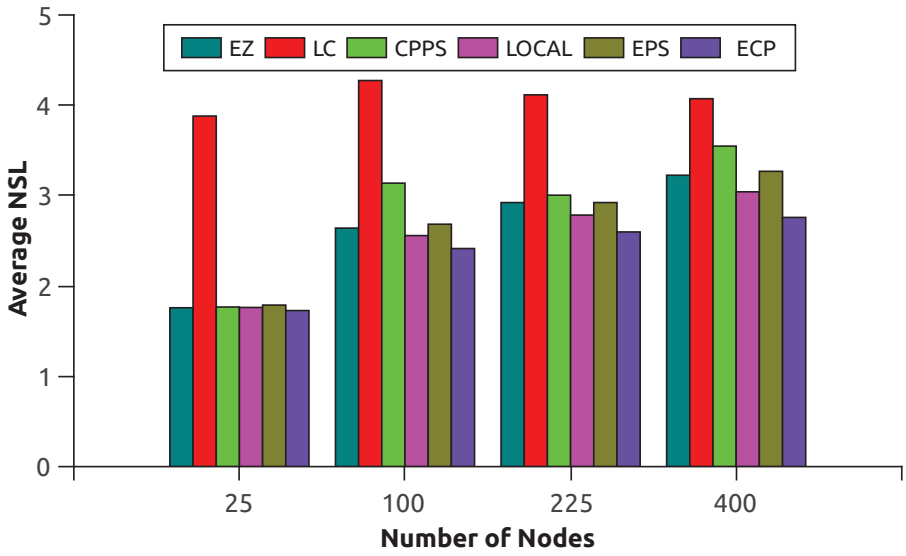

FIGURE 5.12: Average NSL results obtained for systolic array task graphs as a function of number of nodes.

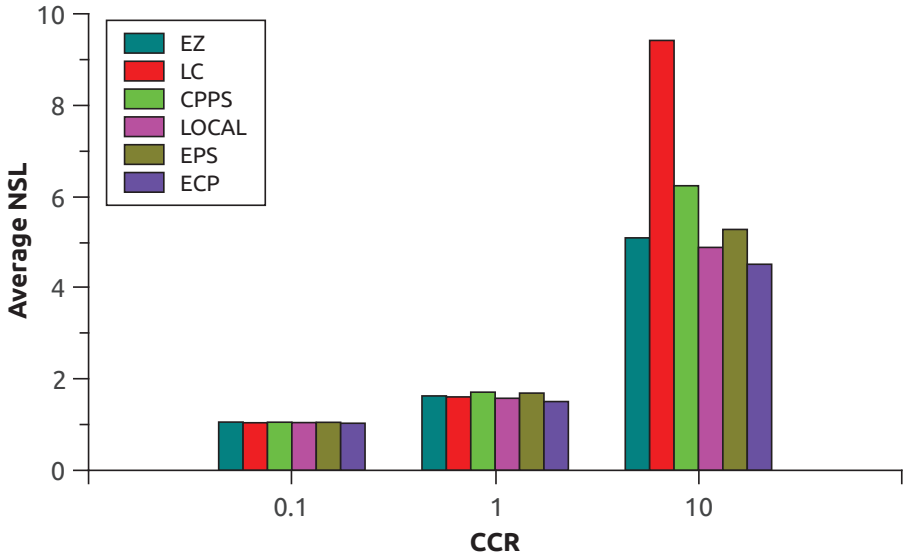

FIGURE 5.13: Average NSL results obtained for systolic array task graphs as a function of CCR.

The average NSL results of systolic array graphs for the different number of nodes are shown in Fig. 5.12. For each set of task graphs, the ECP algorithm produces better schedules than other algorithms. Overall, the ECP algorithm gives an improvement of 9.35, 41.55, 17.10, 6.39 and 10.93 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

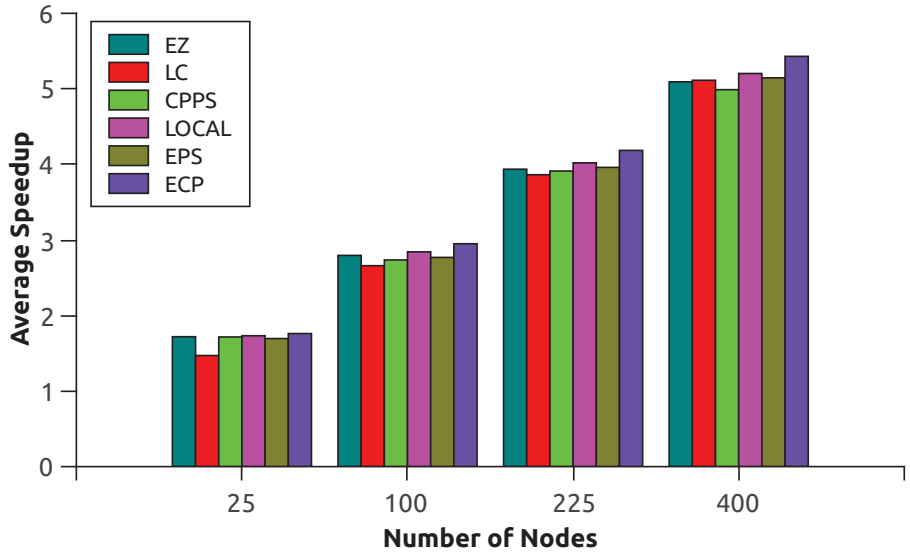

FIGURE 5.14: Average speedup results obtained for systolic array task graphs as a function of number of nodes.

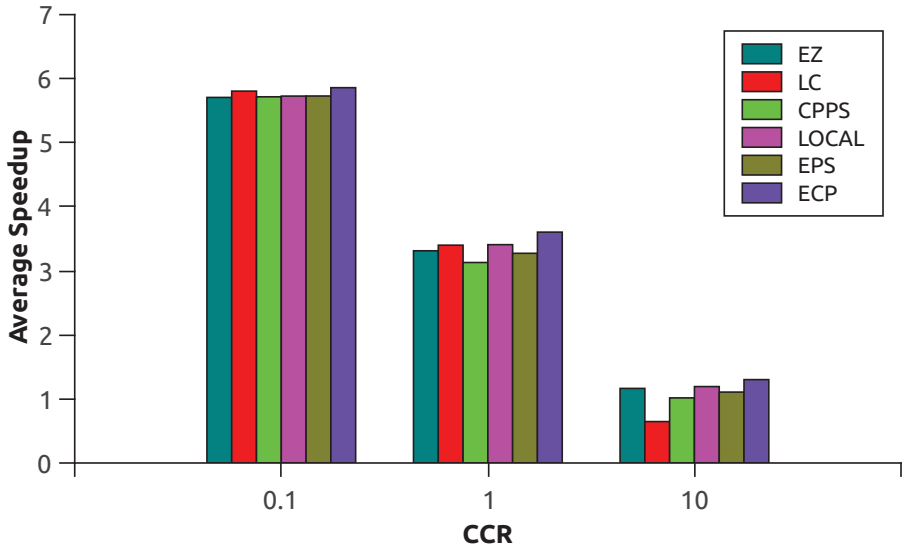

FIGURE 5.15: Average speedup results obtained for systolic array task graphs as a function of CCR.

The average NSL results of systolic array graphs for the different values of CCR are shown in Fig. 5.13. For CCR = 0.10, the ECP algorithm provides an improvement of 2.40, 1.04, 2.14, 1.38 and 1.90 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 1, the ECP algorithm provides an improvement of 7.58, 6.34, 12.02, 4.55 and 10.95 percent over the EZ, LC, CPPS,

TABLE 5.13: Statistical analysis of NSL results obtained for systolic array task graphs with CCR value 0.1.

Algorithm	**Nodes**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Minimum**
EZ	25	1.043603	0.015981	1.032172 - 1.055035	1.036585
	100	1.058077	0.011448	1.049888 - 1.066265	1.052689
	225	1.066388	0.009172	1.059827 - 1.072948	1.060298
	400	1.063987	0.009889	1.056913 - 1.071060	1.056964
LC	25	1.033850	0.014309	1.023614 - 1.044085	1.028094
	100	1.041283	0.011206	1.033267 - 1.049298	1.035223
	225	1.049304	0.009013	1.042857 - 1.055751	1.043243
	400	1.043973	0.007338	1.038723 - 1.049222	1.040595
CPPS	25	1.033285	0.020087	1.018917 - 1.047653	1.021622
	100	1.051902	0.012226	1.043157 - 1.060647	1.044371
	225	1.064057	0.016533	1.052230 - 1.075883	1.053598
	400	1.064329	0.017664	1.051694 - 1.076964	1.054602
LOCAL	25	1.035896	0.019801	1.021732 - 1.050060	1.037602
	100	1.045951	0.010345	1.038551 - 1.053351	1.043122
	225	1.052850	0.007908	1.047194 - 1.058507	1.047377
	400	1.050437	0.008060	1.044672 - 1.056202	1.047498
EPS	25	1.034091	0.022153	1.018245 - 1.049937	1.037602
	100	1.043500	0.012733	1.034392 - 1.052608	1.037662
	225	1.062591	0.016051	1.051109 - 1.074072	1.051713
	400	1.064412	0.016238	1.052797 - 1.076027	1.053013
ECP	25	1.025544	0.013467	1.015911 - 1.035177	1.018060
	100	1.033824	0.010001	1.026670 - 1.040978	1.029257
	225	1.037313	0.007265	1.032116 - 1.042510	1.032678
	400	1.034087	0.008115	1.028282 - 1.039891	1.029468

LOCAL and EPS scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 11.36, 52.06, 27.59, 7.56 and 14.45 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average Speedup results of systolic array graphs for the different number of nodes are shown in Fig. 5.14. For each set of task graphs, the ECP algorithm

TABLE 5.14: Statistical analysis of NSL results obtained for systolic array task graphs with CCR value 1.

Algorithm	**Nodes**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Minimum**
EZ	25	1.536343	0.123005	1.448356 - 1.624329	1.478632
	100	1.577889	0.090181	1.513382 - 1.642396	1.519231
	225	1.694573	0.068351	1.645681 - 1.743465	1.672000
	400	1.690534	0.070208	1.640314 - 1.740754	1.642857
LC	25	1.562495	0.167916	1.442383 - 1.682606	1.452991
	100	1.565057	0.078604	1.508831 - 1.621283	1.528846
	225	1.633111	0.083865	1.573121 - 1.693100	1.591111
	400	1.624552	0.088418	1.561306 - 1.687798	1.568898
CPPS	25	1.541131	0.173547	1.416991 - 1.665270	1.437909
	100	1.646739	0.139852	1.546702 - 1.746776	1.552885
	225	1.806706	0.122247	1.719262 - 1.894150	1.722944
	400	1.822393	0.081123	1.764365 - 1.880422	1.794355
LOCAL	25	1.537847	0.144797	1.434273 - 1.641421	1.482906
	100	1.515092	0.074957	1.461474 - 1.568709	1.474104
	225	1.623623	0.065294	1.576918 - 1.670327	1.601000
	400	1.616232	0.063890	1.570531 - 1.661933	1.578013
EPS	25	1.553972	0.177935	1.426694 - 1.681251	1.487180
	100	1.622474	0.126706	1.531840 - 1.713107	1.607143
	225	1.772353	0.136003	1.675069 - 1.869637	1.687943
	400	1.789816	0.090332	1.725201 - 1.854431	1.755639
ECP	25	1.433997	0.113605	1.352734 - 1.515259	1.393162
	100	1.452294	0.064896	1.405873 - 1.498714	1.415929
	225	1.550671	0.070803	1.500026 - 1.601317	1.518750
	400	1.539130	0.060871	1.495589 - 1.582672	1.510040

produces better schedules than other algorithms. Overall, the ECP algorithm gives an improvement of 5.39, 8.48, 6.80, 3.68 and 5.28 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

The average Speedup results of systolic array graphs for the different values of CCR

Table 5.15: Statistical analysis of NSL results obtained for systolic array task graphs with CCR value 10.

Algorithm	**Nodes**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Minimum**
EZ	25	2.656270	0.646072	2.194130 - 3.118410	2.329193
	100	4.954801	0.707600	4.448650 - 5.460953	4.831579
	225	6.095106	0.587341	5.674977 - 6.515236	5.816367
	400	6.727032	0.691736	6.232228 - 7.221835	6.242857
LC	25	9.037529	1.100222	8.250533 - 9.824526	9.545455
	100	9.672246	0.824791	9.082267 - 10.262225	9.105263
	225	9.710346	0.938377	9.039119 - 10.381574	9.100447
	400	9.769025	0.841663	9.166977 - 10.371072	9.235865
CPPS	25	3.220695	1.451986	2.182079 - 4.259311	2.329193
	100	6.154393	1.631067	4.987679 - 7.321106	5.492806
	225	6.606369	1.897791	5.248865 - 7.963872	6.118343
	400	9.057157	1.626923	7.893407 - 10.220907	8.599547
LOCAL	25	2.656270	0.646072	2.194130 - 3.118410	2.329193
	100	4.739726	0.631426	4.288063 - 5.191389	4.694737
	225	5.689939	0.494973	5.335881 - 6.043997	5.341237
	400	6.188127	0.568945	5.781156 - 6.595098	5.787848
EPS	25	2.656270	0.646072	2.194130 - 3.118410	2.329193
	100	5.637119	2.127578	4.115247 - 7.158991	4.363309
	225	7.837555	1.732167	6.598523 - 9.076586	6.959821
	400	8.800448	0.569974	8.392742 - 9.208155	8.652977
ECP	25	2.656270	0.646072	2.194130 - 3.118410	2.329193
	100	4.524651	0.584671	4.106431 - 4.942870	4.500000
	225	5.282772	0.486683	4.934644 - 5.630900	4.692307
	400	5.646422	0.476069	5.305886 - 5.986958	5.328767

are shown in Fig. 5.15. For CCR = 0.10, the ECP algorithm provides an improvement of 2.59, 0.92, 2.43, 2.24 and 2.22 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 1, the ECP algorithm provides an improvement of 8.08, 5.65, 13.15, 5.43 and 19.17 percent over the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively. For CCR = 10, the ECP algorithm provides an improvement of 10.55, 50.33, 21.94, 8.35 and 14.96 percent over

Table 5.16: Statistical analysis of speedup results obtained for systolic array task graphs with CCR value 0.1.

Algorithm	Nodes	Mean	Std. Dev.	CI (95 %)	Maximum
EZ	25	2.418199	0.465455	2.085255 - 2.751142	2.450980
	100	4.555786	0.458366	4.227913 - 4.883659	4.664179
	225	6.698784	0.473503	6.360083 - 7.037484	6.879331
	400	8.790735	0.444735	8.472613 - 9.108857	8.952551
LC	25	2.439738	0.463614	2.108112 - 2.771364	2.475248
	100	4.628291	0.452922	4.304312 - 4.952269	4.699248
	225	6.807234	0.471076	6.470270 - 7.144198	6.970260
	400	8.960863	0.490388	8.610085 - 9.311641	9.136592
CPPS	25	2.438944	0.448242	2.118314 - 2.759575	2.485089
	100	4.581392	0.447388	4.261372 - 4.901412	4.678727
	225	6.711152	0.427176	6.405590 - 7.016714	6.923077
	400	8.789980	0.474863	8.450307 - 9.129653	9.115337
LOCAL	25	2.446531	0.456025	2.120333 - 2.772728	2.497565
	100	4.621984	0.457358	4.294833 - 4.949135	4.702882
	225	6.818831	0.456202	6.492507 - 7.145156	7.018533
	400	8.918119	0.478933	8.575535 - 9.260703	9.209184
EPS	25	2.438375	0.456423	2.111892 - 2.764857	2.470356
	100	4.618137	0.449505	4.296603 - 4.939671	4.699248
	225	6.723107	0.474321	6.383822 - 7.062393	6.900429
	400	8.788097	0.453898	8.463421 - 9.112774	9.011715
ECP	25	2.459047	0.462799	2.128003 - 2.790090	2.510040
	100	4.662575	0.468143	4.327709 - 4.997441	4.741584
	225	6.886510	0.487417	6.537858 - 7.235163	7.073989
	400	9.046258	0.489592	8.696049 - 9.396467	9.267841

the EZ, LC, CPPS, LOCAL and EPS scheduling algorithms respectively.

Along with the above average results for systolic array task graphs, a statistical analysis of the obtained results is also presented. We use confidence intervals to show the significance of the results. Table 5.13, Table 5.14 and Table 5.15 present the statistical analysis of NSL results obtained for systolic array task graphs with

TABLE 5.17: Statistical analysis of speedup results obtained for systolic array task graphs with CCR value 1.

Algorithm	**Nodes**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Maximum**
EZ	25	1.604897	0.284400	1.401464 - 1.808331	1.785714
	100	2.728703	0.159595	2.614543 - 2.842862	2.824859
	225	3.824679	0.250646	3.645390 - 4.003967	3.750000
	400	4.932924	0.322725	4.702076 - 5.163772	5.154639
LC	25	1.585859	0.299720	1.371467 - 1.800251	1.764706
	100	2.751700	0.186976	2.617955 - 2.885446	2.832861
	225	3.973826	0.313346	3.749687 - 4.197965	4.095874
	400	5.132047	0.257079	4.948156 - 5.315937	5.231037
CPPS	25	1.598461	0.245987	1.422504 - 1.774417	1.760563
	100	2.625144	0.236862	2.455715 - 2.794573	2.665198
	225	3.601042	0.353394	3.348256 - 3.853827	3.693467
	400	4.577112	0.306049	4.358193 - 4.796031	4.482616
LOCAL	25	1.623612	0.237881	1.453453 - 1.793770	1.760563
	100	2.794579	0.203571	2.648963 - 2.940194	2.891878
	225	3.910939	0.300288	3.696141 - 4.125738	4.009627
	400	4.996027	0.285433	4.791854 - 5.200199	5.074576
EPS	25	1.586775	0.255398	1.404087 - 1.769463	1.760563
	100	2.663582	0.246255	2.487433 - 2.839730	2.695418
	225	3.664795	0.253608	3.483387 - 3.846203	3.843964
	400	4.656475	0.213382	4.503842 - 4.809109	4.803843
ECP	25	1.716118	0.282729	1.513881 - 1.918356	1.914894
	100	2.964013	0.190048	2.828070 - 3.099955	2.964427
	225	4.180836	0.279768	3.980716 - 4.380956	4.213483
	400	5.414941	0.302762	5.198373 - 5.631509	5.582278

CCR value 0.1, 1 and 10 respectively. Table 5.16, Table 5.17 and Table 5.18 present the statistical analysis of speedup results obtained for systolic array task graphs with CCR value 0.1, 1 and 10 respectively. The results in these tables are presented for confidence level 95 % and sample size 10.

TABLE 5.18: Statistical analysis of speedup results obtained for systolic array task graphs with CCR value 10.

Algorithm	Nodes	Mean	Std. Dev.	CI (95 %)	Maximum
EZ	25	1.000000	0.000000	1.000000 - 1.000000	1.000000
	100	1.026077	0.035104	1.000966 - 1.051187	1.030664
	225	1.195856	0.092685	1.129558 - 1.262154	1.261211
	400	1.421896	0.097505	1.352149 - 1.491642	1.458333
LC	25	0.293116	0.058825	0.251037 - 0.335194	0.311203
	100	0.524673	0.053814	0.486179 - 0.563167	0.558633
	225	0.749684	0.046175	0.716654 - 0.782713	0.770021
	400	0.976650	0.046126	0.943656 - 1.009644	1.008999
CPPS	25	0.900530	0.215846	0.746134 - 1.054926	1.022727
	100	0.881070	0.257462	0.696906 - 1.065234	0.982318
	225	1.177413	0.321269	0.947607 - 1.407220	1.369716
	400	1.096223	0.305940	0.877382 - 1.315064	1.280970
LOCAL	25	1.000000	0.000000	1.000000 - 1.000000	1.000000
	100	1.001998	0.141513	0.900773 - 1.103223	1.094823
	225	1.297104	0.157991	1.184092 - 1.410116	1.302607
	400	1.392950	0.144194	1.289806 - 1.496093	1.419802
EPS	25	1.000000	0.000000	1.000000 - 1.000000	1.000000
	100	0.986606	0.280290	0.786113 - 1.187099	1.173239
	225	0.963817	0.209505	0.813957 - 1.113678	1.105354
	400	1.089926	0.143495	0.987282 - 1.192569	1.078900
ECP	25	1.000000	0.000000	1.000000 - 1.000000	1.000000
	100	1.122925	0.065167	1.076311 - 1.169539	1.165703
	225	1.376795	0.071404	1.325718 - 1.427871	1.418663
	400	1.689675	0.082616	1.630579 - 1.748771	1.740039

5.3 Summary

We have proposed and explained here a clustering-based scheduling algorithm that makes use of critical path, and we name it Effective Critical Path (ECP) algorithm, for the problem of task scheduling in multiprocessors. The ECP algorithm goes for edge zeroing on critical paths for clustering the tasks. The complexity of the ECP

algorithm works out to be $O(|V|^2(|V| + |E|))$, where $|E|$ represents the number of edges and $|V|$ denotes the number of tasks in the task graph. The performance of this ECP algorithm is compared with four well-known clustering-based scheduling algorithms such as EZ, LC, CPPS, and LOCAL. The comparative study is based on two types of task graphs such as randomly generated benchmark task graphs and task graphs that correspond to real-world applications. The task graphs derived from real-world applications are Gaussian Elimination, Fast Fourier Transform and systolic array. The ECP algorithm proposed here significantly outperforms the said algorithms in terms of average NSL and average speedup for all types of task graphs considered in this work. A statistical analysis of the results obtained from the experiments is also performed along with average result analysis. We used confidence intervals to show the significance of the results. This analysis supports the conclusion of average result analysis. For future work, the proposed task scheduling algorithm may be suitably extended for heterogeneous multiprocessors or may be integrated with the existing duplication-based task scheduling strategies for different real-world applications.

The task scheduling algorithms proposed in this chapter and in the previous chapter address the issue of makespan minimization of the task graph. It will be interesting to develop a scheduling algorithm which addresses another issue of minimization of energy consumption, in the next chapter.