Chapter 4

An Edge Priority-based Scheduling Algorithm

In this chapter, we propose a clustering-based task scheduling algorithm called Edge Priority Scheduling (EPS) for multiprocessor environments. The proposed algorithm extends the idea of edge zeroing heuristic and uses the concept of edge priority to minimize the makespan of the task graph.

The intuitive idea proposed here in this chapter leads to the following interesting and useful observations and consequent contributions:

- The idea of computing priority of edges based on their associated computation and communication costs gives preference to that edge which links communication-intensive clusters while performing further clustering. Our scheme of prioritization is aimed at providing an appropriate technique for obtaining meaningful clustering.
- The proposed work successfully attains minimization of makespan, as demonstrated by the results in comparison to results obtained by such similar approaches. Makespan is an important optimization criterion for the problem of scheduling precedence-constrained tasks of a parallel application in multiprocessor environments.

- A comparative study is done to know the performance improvement of the proposed EPS algorithm over six well-known algorithms such as EZ [3], LC [23], CPPS [27], DCCL [69], RDCC [67], and LOCAL [71]. This contribution demonstrates the applicability of the proposed EPS algorithm for ready reference and comparison.

- We present results for two types of task graphs namely, randomly generated benchmark task graphs [79] and task graphs generated from real-world applications such as Gaussian Elimination [10, 11, 39] and Fast Fourier Transform (FFT) [10, 40]. These results were obtained for the said two types of task graphs unlike the first five algorithms (EZ, LC, CPPS, DCCL, RDCC) that considered only random task graphs. It is only the algorithm known as LOCAL presents the results in similar fashion comparing the performance for both types of graphs.

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

4.1 The EPS Algorithm

In this section, we present our proposed Edge Priority Scheduling (EPS) algorithm for a multiprocessor environment to schedule tasks for their possible parallel execution, on an unlimited number of fully connected homogeneous processors. The proposed algorithm is shown as Algorithm 1 and performs a series of clustering refinement steps. The primary step assigns each task of the task graph to a distinct cluster. At each step, the algorithm attempts to enhance its preceding clustering by identifying and merging suitable clusters into one. A merging operation is carried out through zeroing an edge cost linking two selected clusters. The primary aim here is to minimize the makespan of the input task graph satisfying the necessary precedence constraints. The proposed algorithm extends and enriches the idea of Sarkar′s algorithm [3] and defines and uses the concept of edge priority also. In Sarkar′s algorithm, edges are examined sequentially from the sorted list maintained according to their communication costs and possibly zeroed ensuring that makespan of the task graph does not increase. The authors of CPPS algorithm [27] viewed

Sarkar′s algorithm as a priority based algorithm in which communication cost of edges are considered as priorities for edges, and they extend this idea to characterize priority as a function between cluster pairs. In CPPS, priority between cluster pairs is determined by using total computation costs of clusters and all communication costs between them. Here in the proposed algorithm, we also consider Sarkar′s algorithm for enriching the same as a priority based algorithm and define a priority function for the edges of the task graph which depends on the communication cost of the edges. The priority function defined in the proposed EPS algorithm is different from the priority function used in the CPPS algorithm. In EPS algorithm, when an edge between clusters is selected for merging, it involves grouping, of two tasks belonging to two different clusters, that results in merging of both clusters into one. Thus, for defining the priority of an edge, we use only the communication cost of the edge and the execution costs of the tasks associated with that edge instead of using communication and computation costs of the clusters. At each clustering step, the algorithm zeroes an edge if makespan of the graph decreases. The proposed algorithm has the following characteristics:

- after initial clustering, at each step of the iteration, it merges a pair of selected clusters, creating a new cluster.
- when makespan obtained in the current step, after performing edge zeroing, is greater than the one obtained in the previous step, it backtracks to the previous step and searches for the next possibilities,
- an intermediate makespan is updated after every step, until final makespan of the task graph is obtained,
- it implicitly provides the feasible schedule at every step of the clustering.

In the following, we provide some understandings that are used in the design of our proposed algorithm. In the first subsection, we illustrate how edges are prioritized. In the second subsection, we talk about our approach for clustering explaining how clusters are selected for merging and how new clusters are formed. The detailed description of the proposed EPS algorithm is provided in the third subsection. In the fourth subsection, the complexity analysis is given. An illustrative example,

which demonstrates the working of the EPS algorithm is given at the end of this section.

4.1.1 Edge Prioritization

Here we introduce the idea of priority of an edge to be used in our proposed algorithm. To define edge priority, we use communication time of the edge and execution time of the tasks associated with it. The priority of an edge $e_{i,j}$ that connects the tasks T_i and T_j can be calculated as given by Eq. 4.1:

$$p(e_{i,j}) = \frac{CT(e_{i,j})}{ET(T_i) + ET(T_j)} \tag{4.1}$$

For example, in Fig. 2.1, the edge $e_{0,1}$ has communication time 3 and it is associated with the tasks T_0 and T_1 that have execution or computation time 2 and 3 respectively. Thus the priority of edge $e_{0,1}$ can be calculated as $3/(2+3) = 0.6$.

Here, $p(e_{i,j})$ shows the communication per computation cost of the tasks involved with the edge $e_{i,j}$. For non-zeroed edges, priority must be greater than zero. There are three types of non-zeroed edges according to their priority values: (1) computation-intensive priority edges, (2) balanced priority edges, and (3) communication-intensive priority edges. The computation-intensive priority edges are those in which sum of the computation time of the tasks associated with an edge have a higher value than the communication time of that edge and hence the priority for these edges ranges between 0 and 1. The balanced priority edges have equal values of the communication time of an edge and the sum of the computation time of the tasks associated with that edge, and consequently $p(e_{i,j})$ is 1. The communication-intensive priority edges have a higher value of communication time of an edge in comparison to the sum of computation time of its associated tasks and the value of $p(e_{i,j})$ is greater than 1 for these types of edges.

4.1.2 Clustering

After prioritizing edges, an edge is selected for zeroing and taken up for merging of two tasks from such a pair cluster. Due to zeroing of the edge, now the tasks

Algorithm 1 The EPS Algorithm

```
1: Initially one cluster for each of the tasks is formed
2: Determine initial makespan
3: Compute priority of each edge using Eq. 4.1
4: Sort the edges in non-increasing order according to their priorities
5: repeat
6:     for all edges in the sorted list do
7:         Zero an edge if makespan reduces
8:         When two clusters are grouped, the order among tasks is decided by
   comparing their bottom-levels with each other
9:         if bottom-level of one task is equal to the bottom-level of other task then
10:            Both tasks are ordered according to their topological-order in the clus-
   ter
11:        end if
12:        Update makespan
13:        break
14:    end for
15:    Update the sorted list by deleting the entry of an edge by which makespan
   is reduced.
16: until makespan decreases
```

will have to be executed on the same cluster, and the communication between tasks would become local. This is useful for reducing the total completion cost of the task graph as a possible heavy communication cost is eliminated. When two clusters are merged into one, the order of the tasks should be maintained. To maintain the order among tasks within a cluster, their bottom-levels are compared (which can be calculated by using Eq. 2.2 and Eq. 2.3) and a pseudo edge with zero communication cost is added from the task having higher bottom-level value to the task with lower bottom-level value. If the tasks have same bottom-level value, they must be ordered according to their topological order in their clusters and a pseudo edge with zero communication cost is added from the higher order task to the lower order task.

4.1.3 The Algorithm

In this subsection, we formalize the proposed algorithm as Algorithm 1.

This algorithm begins with the initial clustering and assigns each task of the task graph in a distinct cluster at line 1. It then determines the initial makespan of the

input task graph and calculates the priority of each edge by using Eq. 4.1 in lines 2 and 3. The priority is static, and it will remain constant at each scheduling step of the algorithm. The edges of the task graph are sorted in non-increasing order according to their priorities in line 4. The algorithm performs the steps from lines 6 to 15 repeatedly, as long as makespan decreases. In these steps, the algorithm selects the edges one by one from the sorted list and zeroes them if makespan of the task graph reduces. Here, sorting is done by using heap sort technique. When some edges have same priority value, the edge with the highest communication cost gets the preference among all. Whenever clusters are grouped, the order among tasks will be decided by comparing their bottom-levels with each other, i.e., the task with higher bottom-level will become predecessor of the task having lower bottom-level. If bottom-levels of tasks are equal, the tasks will be ordered according to their topological-order in the cluster, i.e., the task having higher topological-order in the cluster will become predecessor of the task having lower topological-order in the cluster. The makespan will be updated if it is reduced after edge zeroing. Whenever makespan is updated, the sorted list is updated by deleting an edge by which makespan is reduced.

In the for loop, the algorithm selects edges for edge zeroing from the sorted list that gives preference to the communication-intensive priority edges over computation-intensive priority edges. By doing this, the algorithm performs two things: (1) it tries to group into one cluster two tasks from different clusters that have heavy communication cost between each other with respect to their execution costs, (2) it would not attempt to put two tasks, from different clusters, together if they have heavy execution costs with respect to the communication cost between them.

4.1.4 Algorithm Complexity Analysis

This subsection presents a time complexity analysis of the EPS algorithm explained above.

Line 1 is an initialization step which would be performed $|V|$ times. Line 2 determines the initial makespan in $(|V| + |E|)$ times. In line 3, the priority of the edges can be calculated via Eq. 4.1 in $O(|V| + |E|)$ time. The sorting of the edges in

line 4 would be performed $O(|E|log|E|)$ time via heap sort. The for loop starting at line 6 will occur at most $|E|$ times when all edges are non-zeroed edges. Line 7 can be executed in $(|V| + |E|)$ times. From lines 8 to 11, when two clusters are merged, it can be done in $O(|V|)$ time. Line 12 updates makespan in constant time. Line 15 updates the sorted list in constant time. The total number of iterations for lines 6 to 15 occurs $|V|$ times. Thus, the complexity of the EPS algorithm is $O(|V||E|(|V| + |E|))$.

4.1.5 An Illustrative Example

Consider the task graph given in Fig. 2.1. The graph contains fifteen tasks labeled T_0 to T_{14} with their execution times. The tasks T_0 and T_{14} are the start and end tasks of the task graph respectively which represent the beginning and end of the application. The edges of the graph are labeled with the communication costs.

The algorithm first places each task into the distinct cluster as shown in Fig. 4.1a and computes the initial makespan that is 39. The clusters are shown with different color boxes. It then determines the priority of each edge of the task graph and maintains a sorted list of the edges according to their priority. At each step, the algorithm tries to zero an edge if makespan reduces. The execution trace of the EPS algorithm only for the steps when edge zeroing takes place is shown in Fig. 4.1.

Fig. 4.1b-Fig. 4.1g show the intermediate clusters and the partial makespan of the task graph. In Fig. 4.1h, the makespan comes out to be 23 that can′t be reduced after merging any edge in the task graph. Thus, it is the final makespan of the task graph, and the clustering obtained in this step reflects the final schedule of the example task graph.

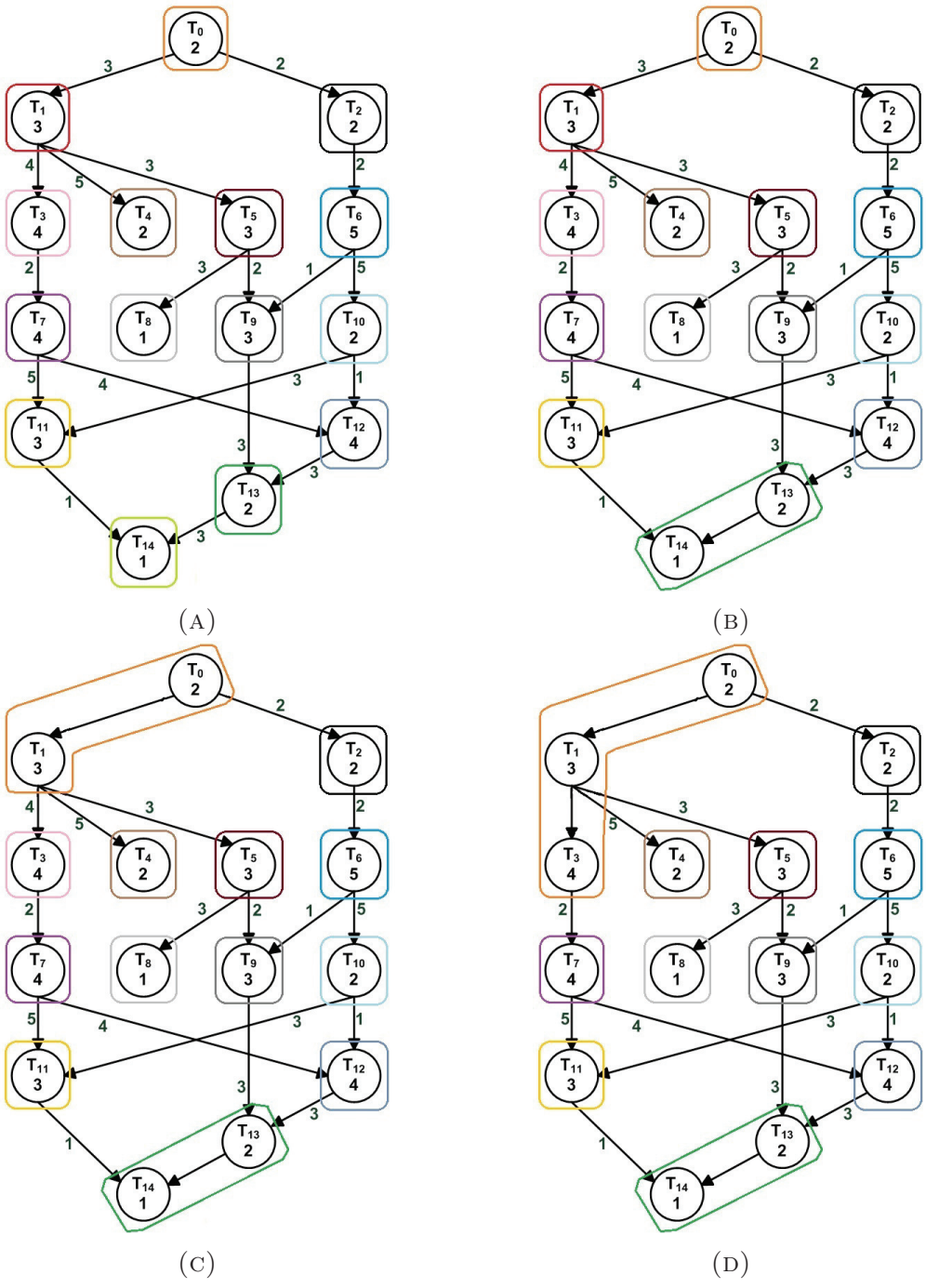

FIGURE 4.1: Clustering steps of the example task graph in Fig. 1 with the EPS algorithm (a) Initial clustering (initial makespan = 39), (b) clustering after zeroing the edge $e_{13,14}$ (partial makespan = 36), (c) clustering after zeroing the edge $e_{0,1}$ (partial makespan = 33), (d) clustering after zeroing the edge $e_{1,3}$ (partial makespan = 31)

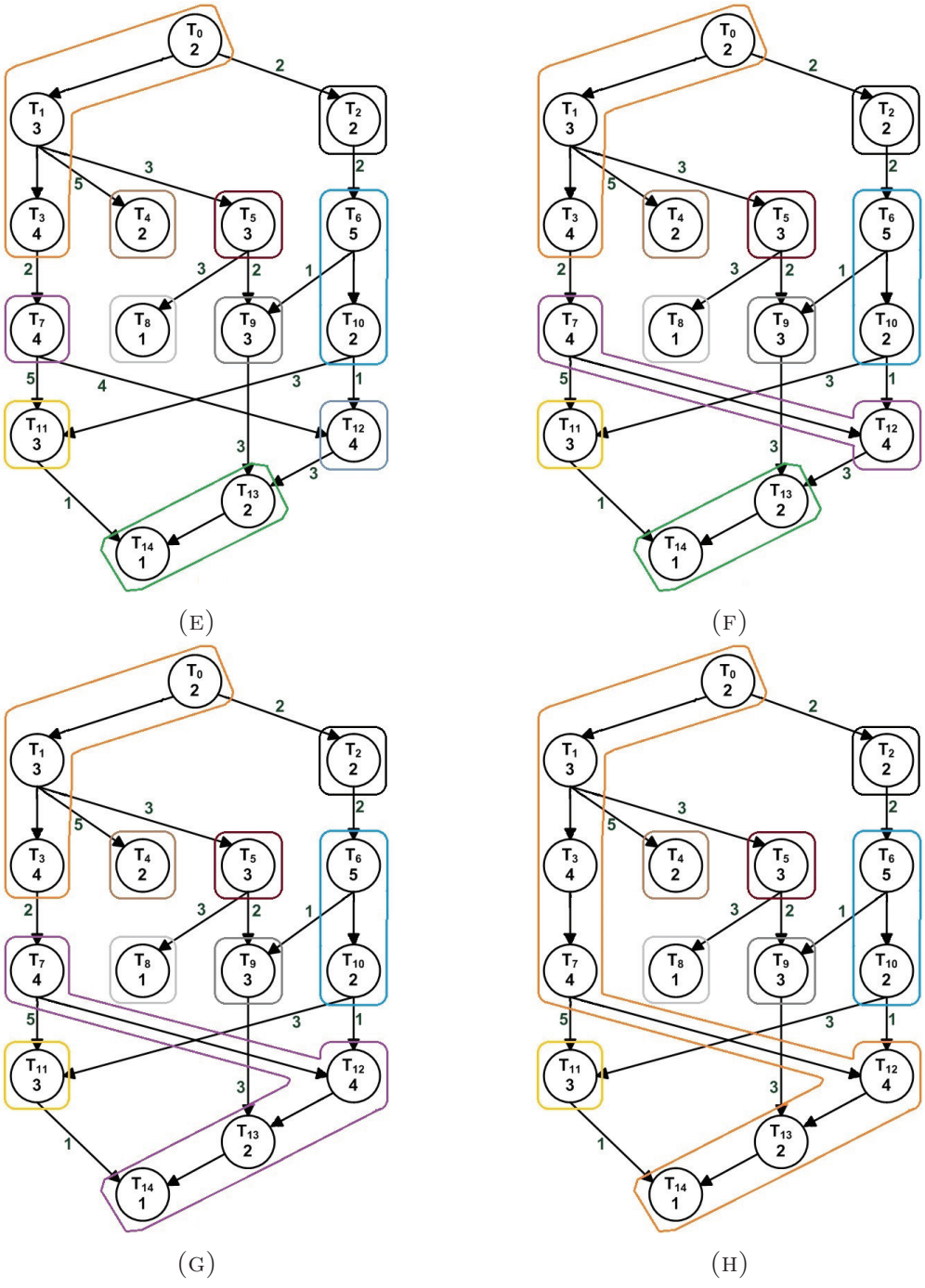

FIGURE 4.1: Clustering steps of the example task graph in Fig. 1 with the EPS algorithm (e) clustering after zeroing the edge $e_{6,10}$ (partial makespan = 29), (f) clustering after zeroing the edge $e_{7,12}$ (partial makespan = 26), (g) clustering after zeroing the edge $e_{12,13}$ (partial makespan = 25), (h) clustering after zeroing the edge $e_{3,7}$ (final makespan = 23).

4.2 Performance Metrics

The comparisons of the task scheduling algorithms are based on the following performance metrics:

4.2.1 Normalized Schedule Length

As more task graphs with various properties are utilized, it is needed to normalize the makespan to the lower bound. Hence, the Normalized Schedule Length (NSL) [72] of a schedule can be defined as the ratio of the makespan and the sum of computation costs on the critical path of the graph that is given in Eq. 4.2:

$$NSL = \frac{makespan}{\sum_{T_i \in CP} ET(T_i)} \tag{4.2}$$

The sum of computation costs on the critical path give the lower bound on the makespan. So the following inequality will also hold for the NSL:

$$NSL \gg 1 \tag{4.3}$$

4.2.2 Speedup

The speedup [10] is the enhancement in speed of computation when task graph is computed on single and various processors. It is defined as the ratio of the sequential computation time and the makespan and is given in Eq. 4.4:

$$Speedup = \frac{\sum_{T_i \in V} ET(T_i)}{makespan} \tag{4.4}$$

4.2.3 CCR

The CCR (Communication to Computation Ratio) [72] is defined as the ratio of the mean communication cost and the mean execution cost in the task graph and is

given in Eq. 4.5:

$$CCR = \frac{\left(\sum_{e_{i,j} \in E} CT(e_{i,j})\right)/|E|}{\left(\sum_{T_i \in V} ET(T_i)\right)/|V|} \tag{4.5}$$

A task graph is called communication intensive task graph if its CCR value is high.

4.2.4 Percentage Improvement in NSL

The percentage improvement in NSL of EPS algorithm over algorithm A [24] is defined as follows:

$$\%NSL_{improvement} = \left(1 - \frac{NSL_{EPS}}{NSL_A}\right) * 100 \tag{4.6}$$

where NSL_{EPS} represents the NSL generated by the EPS algorithm and NSL_A is the NSL generated by the compared algorithms.

4.2.5 Percentage Improvement in Speedup

The percentage improvement in speedup of EPS algorithm over algorithm A is defined as follows:

$$\%Speedup_{improvement} = \left(1 - \frac{Speedup_A}{Speedup_{EPS}}\right) * 100 \tag{4.7}$$

where $Speedup_{EPS}$ represents the speedup generated by the EPS algorithm and $Speedup_A$ is the speedup generated by the compared algorithms.

4.3 Experimental Results and Discussion

In this section, we provide the performance evaluation of the EPS algorithm with six well-known clustering based task scheduling algorithms, the EZ, the LC, the CPPS, the DCCL, the RDCC, and the LOCAL using various performance metrics. For this purpose, two types of task graphs are considered: randomly produced task

TABLE 4.1: t-value used for the estimation of confidence intervals when confidence level is 95 %

Sample size	**t-value**
10	2.262
20	2.093
30	2.045

graphs and the task graphs produced from real-world applications. The experiments are performed 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 use a confidence interval (CI) to characterize the results. Table 4.1 gives the t-values corresponding to different sample sizes used for the estimation of confidence interval when confidence level is 95 %.

4.3.1 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. These graphs are also available online [80].

The average NSL results of random graphs for the different number of nodes are shown in Fig. 4.2. For each set of task graphs, the EPS algorithm produces schedules similar to CPPS and LOCAL but better than others. The CPPS is based on the computation of priority of clusters pairs, and the EPS is based on the determination of priority of edges between tasks of different clusters. The priority computation in both algorithms is founded on the concept which attempts to reduce the communication cost of a given task graph. Overall, the EPS algorithm produces an improvement of 19.06, 5.70, 0.69, 14.46, 12.47, and 0.35 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

The average speedup results for random graphs with different number of nodes are shown in Fig. 4.3. For each set of task graphs, the EPS algorithm produces better speedup than other algorithms. Overall, the EPS algorithm produces an

improvement of 19.88, 5.53, 0.86, 19.07, 16.25, and 0.52 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

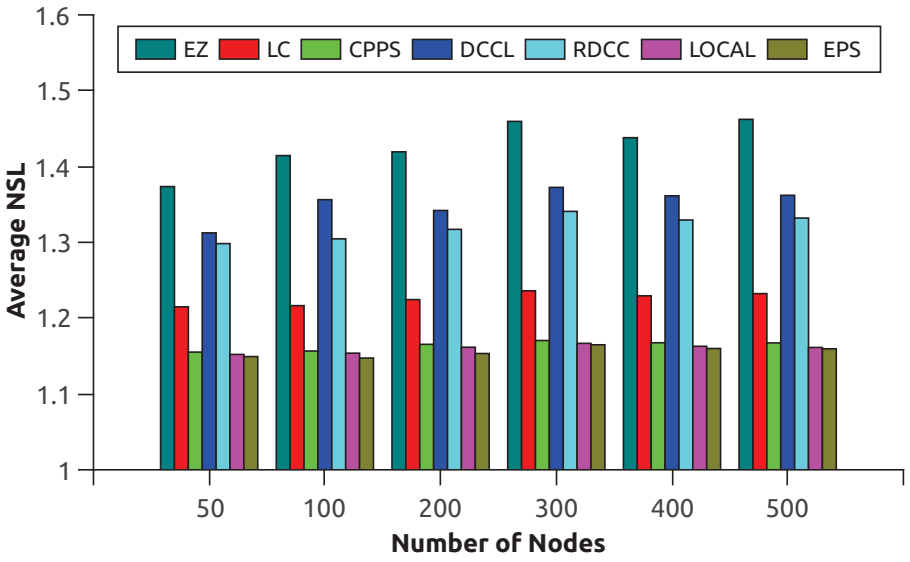

FIGURE 4.2: Average NSL results obtained for random task graphs as a function of number of nodes.

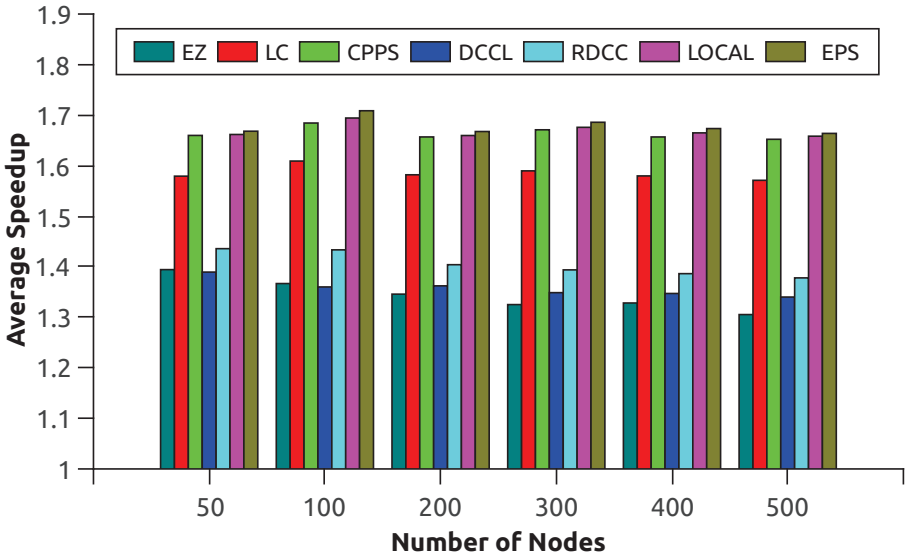

FIGURE 4.3: Average Speedup results obtained for random task graphs as a function of number of nodes.

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 4.2 and Table 4.3 present the statistical analysis of

NSL and speedup results obtained for random task graphs respectively. The results in both tables are presented for confidence level 95 %.

Table 4.2: Statistical analysis of NSL obtained for random task graphs.

Algo	**Nodes**	**Sample**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Minimum**
EZ	50	10	1.412300	0.241223	1.239752 - 1.584849	1.252485
		20	1.354175	0.229191	1.246911 - 1.461438	1.267355
		30	1.373550	0.230760	1.287392 - 1.459707	1.287412
	100	10	1.521630	0.188945	1.386477 - 1.656784	1.392079
		20	1.360804	0.152298	1.289527 - 1.432080	1.294742
		30	1.414413	0.179504	1.347392 - 1.481433	1.350515
	200	10	1.551651	0.175344	1.426226 - 1.677076	1.450462
		20	1.353301	0.167398	1.274958 - 1.431645	1.291011
		30	1.419418	0.192211	1.347653 - 1.491183	1.352453
	300	10	1.582101	0.173938	1.457682 - 1.706520	1.497450
		20	1.398232	0.205427	1.302090 - 1.494374	1.314184
		30	1.459522	0.211683	1.380487 - 1.538557	1.382097
	400	10	1.576699	0.218937	1.420091 - 1.733306	1.443889
		20	1.368710	0.177386	1.285692 - 1.451729	1.287898
		30	1.438040	0.213157	1.358455 - 1.517625	1.363918
	500	10	1.604642	0.217859	1.448806 - 1.760478	1.460553
		20	1.390985	0.186573	1.303667 - 1.478303	1.328066
		30	1.462204	0.219158	1.380378 - 1.544030	1.392759
LC	50	10	1.227063	0.190934	1.090486 - 1.363640	1.108911
		20	1.208858	0.160788	1.133608 - 1.284108	1.146110
		30	1.214926	0.168309	1.152085 - 1.277767	1.179283
	100	10	1.224101	0.110698	1.144918 - 1.303284	1.156663
		20	1.212714	0.096240	1.167673 - 1.257755	1.207772
		30	1.216510	0.099504	1.179359 - 1.253661	1.203960
	200	10	1.245163	0.124162	1.156349 - 1.333976	1.157340
		20	1.214159	0.106798	1.164176 - 1.264141	1.173371
		30	1.224493	0.111705	1.182786 - 1.266200	1.196721
	300	10	1.243298	0.108624	1.165598 - 1.320997	1.170538
		20	1.232539	0.134619	1.169537 - 1.295542	1.177083

continues ...

... continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Minimum
		30	1.236126	0.124746	1.189550 - 1.282701	1.205850
	400	10	1.244818	0.127431	1.153666 - 1.335970	1.165379
		20	1.221834	0.118939	1.166169 - 1.277499	1.186218
		30	1.229495	0.120122	1.184646 - 1.274345	1.199769
	500	10	1.248058	0.105482	1.172607 - 1.323510	1.204990
		20	1.224518	0.107938	1.174002 - 1.275034	1.176635
		30	1.232365	0.105894	1.192827 - 1.271902	1.197867
DCCL	50	10	1.508373	0.277199	1.310091 - 1.706656	1.432308
		20	1.214561	0.170463	1.134782 - 1.294339	1.151376
		30	1.312498	0.250458	1.218986 - 1.406010	1.269430
	100	10	1.576924	0.289600	1.369771 - 1.784077	1.390456
		20	1.245968	0.135092	1.182743 - 1.309192	1.184808
		30	1.356286	0.251326	1.262450 - 1.450122	1.279039
	200	10	1.571309	0.226760	1.409106 - 1.733512	1.477785
		20	1.227372	0.113307	1.174343 - 1.280401	1.198202
		30	1.342018	0.227075	1.257236 - 1.426799	1.279039
	300	10	1.587716	0.228248	1.424448 - 1.750983	1.476486
		20	1.264866	0.140901	1.198923 - 1.330809	1.206798
		30	1.372482	0.230514	1.286417 - 1.458548	1.317752
	400	10	1.591460	0.271386	1.397335 - 1.785584	1.432862
		20	1.246296	0.137606	1.181895 - 1.310696	1.203784
		30	1.361350	0.250302	1.267896 - 1.454804	1.272940
	500	10	1.586508	0.236335	1.417456 - 1.755560	1.460140
		20	1.249830	0.123873	1.191856 - 1.307803	1.217577
		30	1.362056	0.231182	1.275741 - 1.448371	1.308855
RDCC	50	10	1.398595	0.153012	1.289144 - 1.508045	1.338645
		20	1.248220	0.134060	1.185478 - 1.310961	1.192523
		30	1.298345	0.155689	1.240216 - 1.356474	1.256147
	100	10	1.439962	0.169549	1.318682 - 1.561242	1.334665
		20	1.236936	0.107052	1.186834 - 1.287037	1.201186
		30	1.304611	0.160952	1.244517 - 1.364705	1.251192

continues ...

. . . continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Minimum
	200	10	1.436473	0.177231	1.309698 - 1.563247	1.344197
		20	1.257476	0.123368	1.199738 - 1.315213	1.208851
		30	1.317141	0.164576	1.255695 - 1.378588	1.265962
	300	10	1.469224	0.175742	1.343514 - 1.594933	1.355477
		20	1.276681	0.126017	1.217703 - 1.335658	1.222222
		30	1.340862	0.168855	1.277817 - 1.403906	1.299044
	400	10	1.454369	0.187821	1.320019 - 1.588719	1.347493
		20	1.267015	0.118504	1.211554 - 1.322476	1.218882
		30	1.329467	0.167982	1.266748 - 1.392185	1.278259
	500	10	1.464328	0.186106	1.331205 - 1.597451	1.401068
		20	1.265896	0.105317	1.216607 - 1.315185	1.227149
		30	1.332040	0.164523	1.270613 - 1.393467	1.277476
CPPS	50	10	1.160622	0.153806	1.050603 - 1.270641	1.063333
		20	1.152535	0.134542	1.089568 - 1.215502	1.095713
		30	1.155230	0.138623	1.103473 - 1.206987	1.045558
	100	10	1.169500	0.072585	1.117580 - 1.221420	1.119764
		20	1.150271	0.074212	1.115539 - 1.185003	1.200837
		30	1.156681	0.072996	1.129427 - 1.183935	1.132435
	200	10	1.198311	0.113435	1.117170 - 1.279451	1.128143
		20	1.149066	0.074912	1.114006 - 1.184126	1.118868
		30	1.165481	0.090706	1.131614 - 1.199347	1.136612
	300	10	1.186416	0.084298	1.126117 - 1.246715	1.265751
		20	1.162575	0.094184	1.118496 - 1.206654	1.128154
		30	1.170522	0.090265	1.136820 - 1.204224	1.139830
	400	10	1.191329	0.105173	1.116097 - 1.266560	1.117219
		20	1.155657	0.087853	1.114541 - 1.196773	1.119444
		30	1.167547	0.093713	1.132558 - 1.202537	1.139123
	500	10	1.192013	0.090673	1.127154 - 1.256872	1.161655
		20	1.155081	0.073699	1.120589 - 1.189573	1.122083
		30	1.167391	0.080148	1.137467 - 1.197316	1.148160
LOCAL	50	10	1.147004	0.134154	1.051043 - 1.242966	1.079681

continues . . .

... continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Minimum
		20	1.142767	0.121439	1.085932 - 1.199601	1.091725
		30	1.144179	0.123498	1.098070 - 1.190289	1.102667
	100	10	1.150912	0.059168	1.108588 - 1.193235	1.110377
		20	1.142831	0.070133	1.110008 - 1.175654	1.123251
		30	1.145524	0.065757	1.120973 - 1.170076	1.121249
	200	10	1.180451	0.098954	1.109669 - 1.251234	1.112734
		20	1.141228	0.070711	1.108135 - 1.174321	1.082628
		30	1.154302	0.081660	1.123813 - 1.184791	1.126139
	300	10	1.172867	0.079148	1.116252 - 1.229482	1.130276
		20	1.155354	0.089425	1.113502 - 1.197206	1.113973
		30	1.161192	0.085170	1.129392 - 1.192991	1.130276
	400	10	1.176872	0.098959	1.106086 - 1.247659	1.166218
		20	1.147297	0.081679	1.109070 - 1.185524	1.105347
		30	1.157155	0.087242	1.124582 - 1.189729	1.124897
	500	10	1.178754	0.086513	1.116871 - 1.240638	1.152414
		20	1.148960	0.071273	1.115603 - 1.182316	1.118131
		30	1.158891	0.076518	1.130322 - 1.187460	1.144024
EPS	50	10	1.163024	0.147602	1.057443 - 1.268605	1.063333
		20	1.151508	0.136039	1.087841 - 1.215176	1.088734
		30	1.155347	0.137538	1.103995 - 1.206699	1.127490
	100	10	1.155821	0.069744	1.105933 - 1.205710	1.119174
		20	1.147685	0.076503	1.111881 - 1.183489	1.119174
		30	1.150397	0.073207	1.123064 - 1.177730	1.123107
	200	10	1.194586	0.111724	1.114669 - 1.274503	1.116185
		20	1.150784	0.079822	1.113427 - 1.188142	1.114197
		30	1.165385	0.092138	1.130984 - 1.199786	1.139344
	300	10	1.194352	0.081357	1.136157 - 1.252547	1.15653
		20	1.160636	0.091933	1.117610 - 1.203661	1.128600
		30	1.171874	0.088616	1.138788 - 1.204960	1.140978
	400	10	1.185795	0.101848	1.112942 - 1.258647	1.133604
		20	1.153129	0.085465	1.113131 - 1.193127	1.115972

continues ...

. . . continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Minimum
		30	1.164018	0.090830	1.130105 - 1.197930	1.133604
	500	10	1.191027	0.096709	1.121850 - 1.260204	1.155586
		20	1.154372	0.073181	1.120122 - 1.188621	1.121079
		30	1.166590	0.081977	1.135983 - 1.197198	1.150555

Table 4.3: Statistical analysis of speedup obtained for random task graphs.

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Maximum
EZ	50	10	1.931205	0.511606	1.565249 - 2.297160	2.274566
		20	1.125895	0.130150	1.064984 - 1.186807	1.177097
		30	1.394332	0.491338	1.210884 - 1.577780	1.501437
	100	10	1.880736	0.540078	1.494414 - 2.267058	2.101695
		20	1.109802	0.102983	1.061606 - 1.157999	1.154832
		30	1.366780	0.483840	1.186132 - 1.547429	1.399395
	200	10	1.855628	0.506204	1.493536 - 2.217719	2.191745
		20	1.090919	0.073735	1.056411 - 1.125428	1.124746
		30	1.345822	0.466388	1.171690 - 1.519955	1.472318
	300	10	1.797535	0.456817	1.470770 - 2.124299	2.101799
		20	1.088950	0.080150	1.051439 - 1.126461	1.103231
		30	1.325145	0.429413	1.164818 - 1.485472	1.465809
	400	10	1.816379	0.423623	1.513358 - 2.119399	2.109320
		20	1.084298	0.072881	1.050189 - 1.118407	1.117607
		30	1.328325	0.427058	1.168877 - 1.487773	1.456193
	500	10	1.750689	0.421104	1.449471 - 2.051908	1.835347
		20	1.082748	0.073957	1.048136 - 1.117361	1.116514
		30	1.305395	0.401471	1.155501 - 1.455290	1.394957
LC	50	10	2.208487	0.545568	1.818238 - 2.598736	2.592476
		20	1.264608	0.214344	1.164293 - 1.364922	1.327327
		30	1.579234	0.572084	1.365638 - 1.792830	1.739130
	100	10	2.326704	0.635235	1.872316 - 2.781093	2.629690
		20	1.250296	0.188931	1.161874 - 1.338717	1.336662
		30	1.609098	0.644186	1.368583 - 1.849614	1.809659

continues . . .

... continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Maximum
	200	10	2.306748	0.612477	1.868639 - 2.744857	2.658309
		20	1.219541	0.160761	1.144303 - 1.294779	1.281521
		30	1.581943	0.636459	1.344312 - 1.819574	1.692154
	300	10	2.289940	0.605835	1.856582 - 2.723298	2.710760
		20	1.239595	0.183880	1.153538 - 1.325653	1.323968
		30	1.589710	0.624240	1.356641 - 1.822779	1.798618
	400	10	2.300047	0.561897	1.898118 - 2.701976	2.696489
		20	1.219558	0.169991	1.140001 - 1.299115	1.290182
		30	1.579721	0.620724	1.347965 - 1.811477	1.793492
	500	10	2.242974	0.535807	1.859708 - 2.626241	2.518831
		20	1.235040	0.187036	1.147505 - 1.322574	1.301753
		30	1.571018	0.587847	1.351537 - 1.790499	1.757753
DCCL	50	10	1.726488	0.257669	1.542176 - 1.910801	1.905530
		20	1.220932	0.157989	1.146992 - 1.294872	1.288630
		30	1.389451	0.309377	1.273941 - 1.504961	1.482332
	100	10	1.720134	0.217427	1.564607 - 1.875661	1.862132
		20	1.180030	0.125133	1.121467 - 1.238593	1.225368
		30	1.360064	0.303299	1.246823 - 1.473306	1.458459
	200	10	1.745572	0.298201	1.532266 - 1.958877	1.807128
		20	1.170548	0.107393	1.120287 - 1.220809	1.220592
		30	1.362222	0.333415	1.237737 - 1.486708	1.431571
	300	10	1.703435	0.271868	1.508966 - 1.897904	1.873124
		20	1.171434	0.118443	1.116002 - 1.226867	1.225733
		30	1.348768	0.311757	1.232369 - 1.465167	1.393139
	400	10	1.722034	0.249349	1.543673 - 1.900395	1.884597
		20	1.159677	0.104612	1.110717 - 1.208636	1.198528
		30	1.347129	0.314906	1.229554 - 1.464704	1.430851
	500	10	1.682591	0.259239	1.497156 - 1.868026	1.789894
		20	1.168830	0.113280	1.115814 - 1.221846	1.211053
		30	1.340084	0.299904	1.228111 - 1.452057	1.443505
RDCC	50	10	1.920326	0.367714	1.657298 - 2.183355	2.088384

continues ...

. . . continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Maximum
		20	1.193513	0.197077	1.101280 - 1.285747	1.227920
		30	1.435784	0.434566	1.273533 - 1.598036	1.528796
	100	10	1.931478	0.431031	1.623158 - 2.239797	2.157632
		20	1.184425	0.157765	1.110589 - 1.258260	1.250879
		30	1.433442	0.449735	1.265527 - 1.601357	1.574135
	200	10	1.903880	0.376334	1.634686 - 2.173075	2.156146
		20	1.154144	0.152615	1.082719 - 1.225569	1.218331
		30	1.404056	0.434088	1.241983 - 1.566129	1.550043
	300	10	1.856636	0.348348	1.607461 - 2.105812	1.919217
		20	1.162414	0.152651	1.090972 - 1.233856	1.217206
		30	1.393821	0.404621	1.242750 - 1.544892	1.533240
	400	10	1.863914	0.328702	1.628791 - 2.099037	2.058264
		20	1.147561	0.142451	1.080892 - 1.214229	1.200491
		30	1.386345	0.405949	1.234778 - 1.537912	1.480553
	500	10	1.847676	0.327795	1.613203 - 2.082150	1.999554
		20	1.143135	0.155342	1.070434 - 1.215837	1.203179
		30	1.377982	0.404062	1.227120 - 1.528845	1.512102
CPPS	50	10	2.338080	0.611579	1.900613 - 2.775547	2.608833
		20	1.320690	0.200104	1.227039 - 1.414340	1.413366
		30	1.659820	0.616654	1.429583 - 1.890056	1.765203
	100	10	2.423202	0.626333	1.975182 - 2.871223	2.863233
		20	1.314929	0.185138	1.228282 - 1.401575	1.389177
		30	1.684353	0.653119	1.440502 - 1.928204	1.849952
	200	10	2.397690	0.646087	1.935539 - 2.859840	2.802870
		20	1.286415	0.169703	1.206992 - 1.365837	1.350244
		30	1.656840	0.657501	1.411352 - 1.902327	1.837219
	300	10	2.392998	0.619131	1.950129 - 2.835866	2.773472
		20	1.309940	0.186564	1.222626 - 1.397253	1.384615
		30	1.670959	0.641424	1.431474 - 1.910444	1.843508
	400	10	2.397724	0.577239	1.984821 - 2.810627	2.809108
		20	1.286341	0.174823	1.204522 - 1.368159	1.367818

continues . . .

... continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Maximum
		30	1.656802	0.638263	1.418497 - 1.895106	1.865615
	500	10	2.346182	0.557112	1.947676 - 2.744688	2.619181
		20	1.305178	0.185502	1.218361 - 1.391994	1.382158
		30	1.652179	0.606624	1.425688 - 1.878671	1.850629
LOCAL	50	10	2.357192	0.594878	1.931672 - 2.782713	2.638288
		20	1.331941	0.208731	1.234253 - 1.429629	1.413718
		30	1.673692	0.616450	1.443531 - 1.903852	1.775764
	100	10	2.465562	0.645829	2.003596 - 2.927529	2.907807
		20	1.323625	0.188005	1.235637 - 1.411613	1.362051
		30	1.704271	0.672589	1.453150 - 1.955392	1.850848
	200	10	2.433602	0.654951	1.965111 - 2.902094	2.852178
		20	1.295177	0.171708	1.214816 - 1.375537	1.352995
		30	1.674652	0.671101	1.424087 - 1.925217	1.859149
	300	10	2.422237	0.630109	1.971516 - 2.872958	2.489628
		20	1.318422	0.191962	1.228582 - 1.408262	1.406057
		30	1.686360	0.653800	1.442255 - 1.930466	1.858997
	400	10	2.428729	0.589678	2.006927 - 2.850530	2.850037
		20	1.295696	0.178567	1.212125 - 1.379267	1.375045
		30	1.673374	0.651093	1.430279 - 1.916469	1.877395
	500	10	2.373866	0.568557	1.967173 - 2.780559	2.688279
		20	1.312331	0.188770	1.223985 - 1.400677	1.396949
		30	1.666176	0.618640	1.435198 - 1.897154	1.865590
EPS	50	10	2.329498	0.602154	1.898773 - 2.760223	2.667742
		20	1.322486	0.204289	1.226877 - 1.418095	1.417582
		30	1.658157	0.610730	1.430132 - 1.886182	1.801724
	100	10	2.460166	0.673036	1.978738 - 2.941594	2.873440
		20	1.317871	0.184065	1.231727 - 1.404015	1.388148
		30	1.698636	0.680249	1.444655 - 1.952617	1.855127
	200	10	2.403648	0.642585	1.944002 - 2.863293	2.794428
		20	1.284514	0.166997	1.206358 - 1.362671	1.346418
		30	1.657559	0.659044	1.411495 - 1.903622	1.845987

continues ...

. . . continued

Algo	Nodes	Sample	Mean	Std. Dev.	CI (95 %)	Maximum
	300	10	2.372696	0.597442	1.945342 - 2.800051	2.400585
		20	1.312415	0.190520	1.223250 - 1.401580	1.385877
		30	1.665842	0.626890	1.431784 - 1.899900	1.841024
	400	10	2.411914	0.596533	1.985210 - 2.838618	2.827791
		20	1.289160	0.176541	1.206538 - 1.371783	1.371483
		30	1.663412	0.648571	1.421259 - 1.905565	1.858991
	500	10	2.348796	0.558330	1.949418 - 2.748173	2.656194
		20	1.306341	0.188716	1.218021 - 1.394662	1.376162
		30	1.653826	0.608191	1.426749 - 1.880903	1.860349

4.3.2 Real-World Application Graphs

Besides the random graphs, we evaluate and compare the performance of the algorithms for some real-world applications, namely Gaussian Elimination [10, 11, 39] and Fast Fourier Transform [10, 40].

4.3.2.1 Gaussian Elimination

The structure of Gaussian Elimination is known; hence, we use different values for the matrix size as [5, 10, 15, 20, 25, 30] and CCR as [0.1, 1, 10] to perform experiments.

The average NSL results of Gaussian Elimination graphs for different matrix sizes are shown in Fig. 4.4. For matrix size 5, CPPS and LOCAL give similar and better results than others, but when matrix size increases, the EPS produces best results for these graphs. The proposed EPS algorithm makes its decision by selecting a higher priority edge in each iteration. By doing this, the EPS perform two things; first, it tries to merge two heavily communicating clusters into one cluster and second it tries to keep separate two clusters having heavy computation costs. Overall, the EPS algorithm produces an improvement of 13.87, 29.87, 3.81, 48.55, 41.30, and 2.27

percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

The average NSL results of Gaussian Elimination graphs for the different values of CCR are shown in Fig. 4.5. For computation-intensive graphs (CCR = 0.10), EZ, LC, CPPS, LOCAL, and EPS give similar and better schedules than other two algorithms. The EPS produces an improvement of 3.94, 66.11, and 50.22 percent over the EZ, DCCL and RDCC scheduling algorithms respectively. For CCR = 1, the EPS algorithm produces an improvement of 19.32, 4.44, 1.62, 63.43, 48.15, and 1.39 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For communication-intensive graph (CCR = 10), the EPS gives best schedules and produces an improvement of 12.57, 38.93, 3.58, 26.29, 32.99, and 2.20 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. Due to use of a priority function, EPS tries to reduce communication costs of the graph by combining heavily communicating clusters into one.

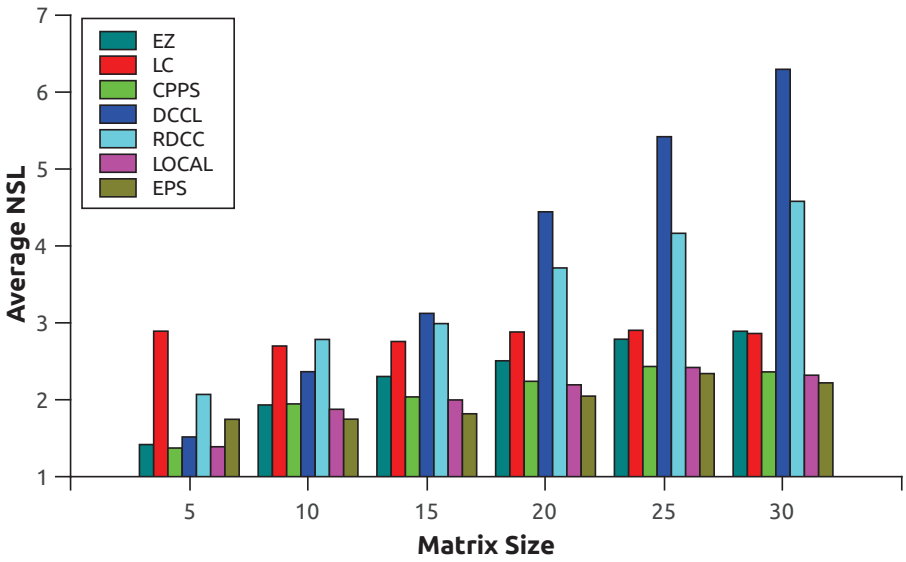

FIGURE 4.4: Average NSL results obtained for Gaussian Elimination task graphs as a function of matrix size.

The average speedup results of Gaussian Elimination graphs for the different matrix sizes are shown in Fig. 4.6. The EPS algorithm produces better schedules than other algorithms for each matrix size used in the experiment except for matrix size 5. Overall, the EPS algorithm produces an improvement of 13.22, 8.40, 2.43,

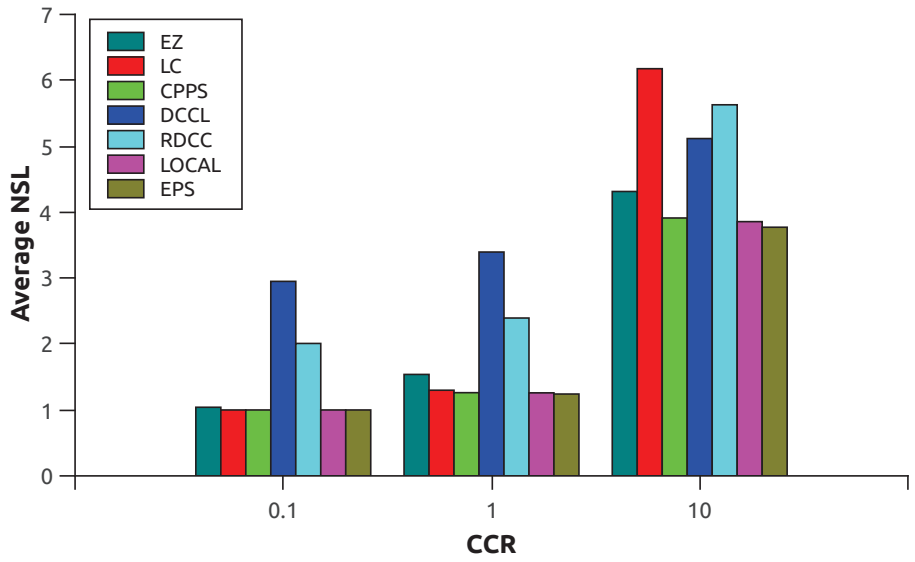

FIGURE 4.5: Average NSL results obtained for Gaussian Elimination task graphs as a function of CCR.

61.20, 51.09, and 3.94 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

The average speedup results of Gaussian Elimination graphs for the different values of CCR are shown in Fig. 4.7. For CCR = 0.10, the EPS algorithm produces an improvement of 4.14, 65.90, and 51.00 percent over the EZ, DCCL and RDCC scheduling algorithms respectively. For CCR = 1, the EPS algorithm produces an improvement of 19.88, 4.01, 1.06, 64.23, 49.88, and 1.80 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For CCR = 10, the EPS algorithm produces an improvement of 16.46, 39.99, 6.51, 28.83, 36.56, and 5.05 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

Along with the above average results for Gaussian Elimination task graphs, a statistical analysis of the obtained results is also presented. We use confidence intervals to show the significance of the results. Table 4.4 and Table 4.5 present the statistical analysis of NSL results obtained for Gaussian Elimination task graphs with CCR value 1 and 10 respectively. All NSL values obtained for Gaussian Elimination task graphs with CCR value 0.1 for LC, CPPS, LOCAL and EPS are equal to 1 and lower than the other three algorithms′ NSL values. Hence, we have not given the

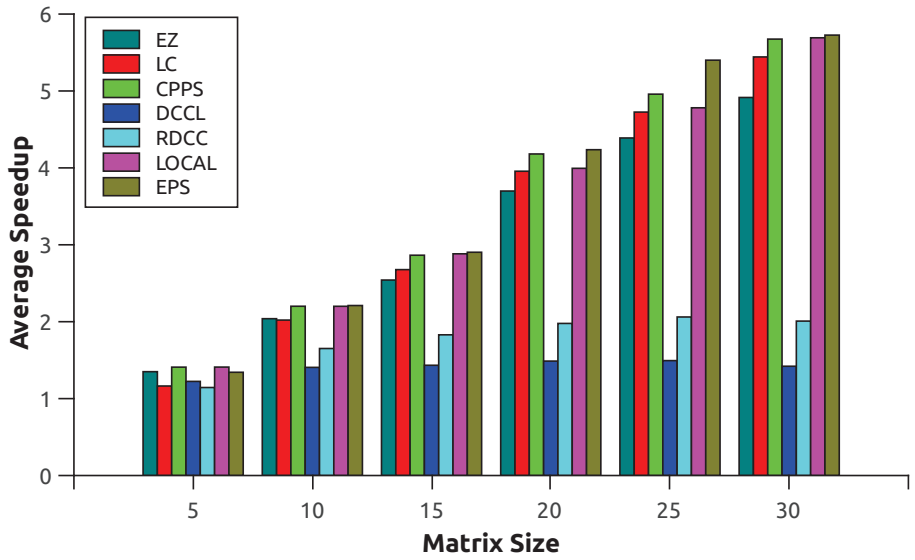

FIGURE 4.6: Average Speedup results obtained for Gaussian Elimination task graphs as a function of matrix size.

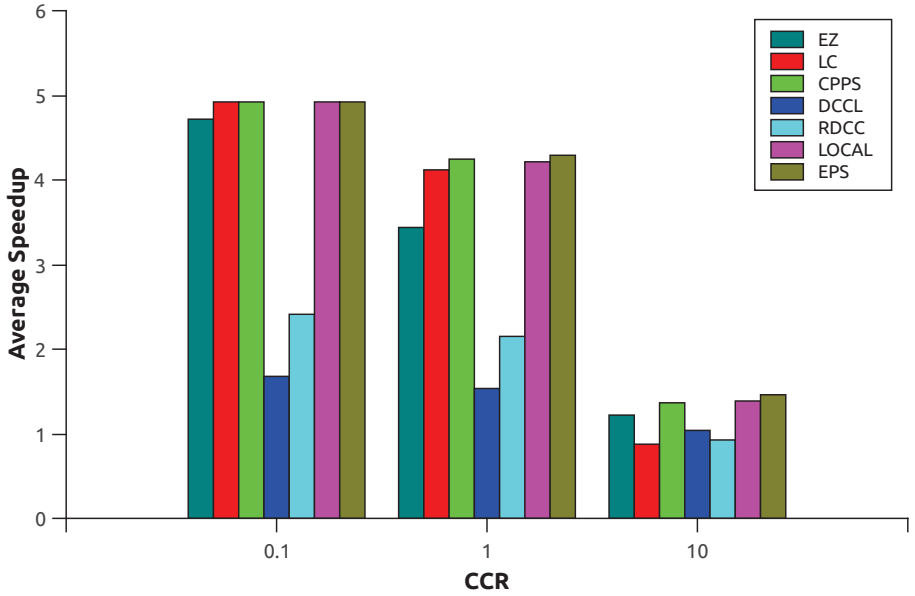

FIGURE 4.7: Average Speedup results obtained for Gaussian Elimination task graphs as a function of CCR.

table for that. Table 4.6, Table 4.7 and Table 4.8 present the statistical analysis of speedup results obtained for Gaussian Elimination 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 4.4: Statistical analysis of NSL obtained for Gaussian Elimination task graphs with CCR value 1.

Algo	**Matrix Size**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Min**
EZ	5	1.364442	0.108448	1.286868 - 1.442015	1.289157
	10	1.478177	0.106853	1.401744 - 1.554609	1.437500
	15	1.548794	0.056209	1.508587 - 1.589000	1.512820
	20	1.564562	0.076537	1.509815 - 1.619309	1.515942
	25	1.607019	0.063226	1.561793 - 1.652244	1.571429
	30	1.655259	0.062463	1.610579 - 1.699939	1.611452
LC	5	1.253708	0.093929	1.186520 - 1.320896	1.187500
	10	1.284395	0.062094	1.239979 - 1.328811	1.256944
	15	1.310255	0.106020	1.234418 - 1.386093	1.237918
	20	1.289732	0.051841	1.252649 - 1.326814	1.252874
	25	1.330477	0.064495	1.284343 - 1.376611	1.295567
	30	1.328390	0.043677	1.297148 - 1.359633	1.300874
DCCL	5	1.673593	0.257110	1.489681 - 1.857506	1.580000
	10	2.155984	0.339210	1.913344 - 2.398623	2.105882
	15	2.944279	0.401265	2.657252 - 3.231307	2.702055
	20	3.849678	0.252127	3.669330 - 4.030027	3.718354
	25	4.388062	0.365617	4.126534 - 4.649591	4.286354
	30	5.252866	0.390838	4.973297 - 5.532435	5.140316
RDCC	5	1.476197	0.132685	1.381287 - 1.571108	1.426667
	10	1.657437	0.114589	1.575471 - 1.739404	1.577778
	15	1.990625	0.180653	1.861403 - 2.119848	1.896341
	20	2.596001	0.296357	2.384014 - 2.807987	2.444444
	25	3.057056	0.328035	2.822410 - 3.291702	2.828889
	30	3.572607	0.317979	3.345154 - 3.800059	3.377922
CPPS	5	1.205080	0.072838	1.152979 - 1.257182	1.200000
	10	1.266069	0.068765	1.216881 - 1.315257	1.234637
	15	1.257705	0.077870	1.202004 - 1.313406	1.204461
	20	1.263174	0.066591	1.215542 - 1.310807	1.218182
	25	1.295029	0.051055	1.258509 - 1.331550	1.263039
	30	1.284631	0.034976	1.259612 - 1.309649	1.276074

continues ...

... continued

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Min
LOCAL	5	1.205900	0.072838	1.153799 - 1.258002	1.200820
	10	1.262569	0.068765	1.213381 - 1.311757	1.231137
	15	1.255741	0.077870	1.200040 - 1.311442	1.202497
	20	1.260879	0.066591	1.213246 - 1.308511	1.215886
	25	1.294389	0.051055	1.257869 - 1.330910	1.262399
	30	1.282467	0.034976	1.257449 - 1.307486	1.273911
EPS	5	1.211570	0.102223	1.138449 - 1.284691	1.187500
	10	1.251555	0.058847	1.209461 - 1.293648	1.232639
	15	1.255451	0.071369	1.204400 - 1.306502	1.204461
	20	1.269832	0.059218	1.227473 - 1.312191	1.229885
	25	1.285503	0.055644	1.245700 - 1.325305	1.252822
	30	1.271159	0.046242	1.238082 - 1.304236	1.239544

TABLE 4.5: Statistical analysis of NSL obtained for Gaussian Elimination task graphs with CCR value 10.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Min
EZ	5	2.232636	0.331204	1.995723 - 2.469549	2.020202
	10	3.300205	0.265817	3.110064 - 3.490345	3.125000
	15	4.296298	0.373965	4.028798 - 4.563798	4.090698
	20	5.013111	0.458353	4.685248 - 5.340975	4.861702
	25	5.243487	0.523879	4.868753 - 5.618221	4.892966
	30	5.800028	0.504573	5.439103 - 6.160952	5.506831
LC	5	5.978889	0.588582	5.557872 - 6.399906	5.577778
	10	6.164788	0.541442	5.777491 - 6.552085	5.831326
	15	6.221564	0.519511	5.849953 - 6.593174	6.269755
	20	6.320848	0.488288	5.971572 - 6.670124	6.109043
	25	6.260010	0.397358	5.975777 - 6.544243	5.991292
	30	6.200770	0.297148	5.988218 - 6.413322	6.101877
DCCL	5	2.199103	0.291995	1.990237 - 2.407969	2.020202
	10	3.257980	0.272996	3.062704 - 3.453256	3.117073
	15	4.537538	0.393493	4.256070 - 4.819007	4.451111

continues ...

. . . continued

Algo	**Matrix Size**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Min**
	20	5.832176	0.483229	5.486519 - 6.177833	5.682693
	25	7.040415	0.487993	6.691351 - 7.389480	6.727679
	30	7.838082	0.432959	7.528383 - 8.147781	7.543766
RDCC	5	4.436718	0.876192	3.809971 - 5.063465	4.178082
	10	5.155364	0.603275	4.723837 - 5.586891	5.040816
	15	5.543875	0.608681	5.108481 - 5.979269	5.195312
	20	5.894719	0.436261	5.582659 - 6.206780	5.683230
	25	6.206294	0.220344	6.048681 - 6.363908	6.122222
	30	6.424276	0.414308	6.127918 - 6.720633	6.472222
CPPS	5	2.286873	0.470518	1.950309 - 2.623438	2.020202
	10	3.418680	0.902260	2.773287 - 4.064074	2.782258
	15	3.707189	0.801310	3.134006 - 4.280371	3.158228
	20	4.470646	0.729897	3.948545 - 4.992746	4.356137
	25	4.731812	0.876887	4.104568 - 5.359055	4.114659
	30	4.699579	0.788340	4.135674 - 5.263484	4.168605
LOCAL	5	2.287693	0.470518	1.951129 - 2.624258	1.892712
	10	3.415180	0.902260	2.769787 - 4.060574	2.778758
	15	3.705225	0.801310	3.132042 - 4.278407	3.156264
	20	4.468350	0.729897	3.946249 - 4.990451	4.353841
	25	4.731172	0.876887	4.103928 - 5.358415	4.114019
	30	4.697416	0.788340	4.133511 - 5.261321	4.166442
EPS	5	2.657847	1.022414	1.926508 - 3.389187	2.058824
	10	3.026856	0.452565	2.703133 - 3.350579	2.765060
	15	3.720635	0.654761	3.252280 - 4.188990	3.272480
	20	4.344039	0.607549	3.909455 - 4.778623	4.023936
	25	4.700954	0.522459	4.327236 - 5.074673	4.423645
	30	4.310873	0.447729	3.990610 - 4.631137	4.171908

Table 4.6: Statistical analysis of speedup results obtained for Gaussian Elimination task graphs with CCR value 0.1.

Algo	**Matrix Size**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Max**
EZ	5	1.990764	0.265870	1.800585 - 2.180943	2.170418
	10	2.993985	0.275830	2.796682 - 3.191288	3.156872
	15	4.050384	0.293446	3.840481 - 4.260288	4.131944
	20	5.295668	0.326797	5.061908 - 5.529428	5.454071
	25	6.429518	0.272920	6.234297 - 6.624739	6.509206
	30	7.429668	0.284417	7.226222 - 7.633113	7.339616
LC	5	2.046907	0.275074	1.850145 - 2.243669	2.297872
	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
DCCL	5	1.585609	0.177702	1.458498 - 1.712720	1.699029
	10	1.689625	0.082955	1.630287 - 1.748963	1.735016
	15	1.772601	0.047230	1.738817 - 1.806385	1.798561
	20	1.711791	0.082773	1.652583 - 1.770999	1.769587
	25	1.698297	0.071159	1.647396 - 1.749197	1.702666
	30	1.609812	0.101918	1.536909 - 1.682715	1.669125
RDCC	5	1.871518	0.208960	1.722047 - 2.020988	1.962209
	10	2.316407	0.222942	2.156935 - 2.475879	2.423698
	15	2.472155	0.112581	2.391625 - 2.552685	2.467964
	20	2.628374	0.110816	2.549107 - 2.707641	2.679612
	25	2.557905	0.188295	2.423216 - 2.692594	2.692101
	30	2.428904	0.181373	2.299167 - 2.558641	2.528646
CPPS	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

continues ...

... continued

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
LOCAL	5	1.993754	0.265870	1.803575 - 2.183933	2.173408
	10	2.999355	0.275830	2.802052 - 3.196658	3.162242
	15	4.061756	0.293446	3.851853 - 4.271660	4.143316
	20	5.305516	0.326797	5.071756 - 5.539276	5.463920
	25	6.442598	0.272920	6.247376 - 6.637819	6.522286
	30	7.455534	0.284417	7.252088 - 7.658979	7.462466
EPS	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

TABLE 4.7: Statistical analysis of speedup results obtained for Gaussian Elimination task graphs with CCR value 1.

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
EZ	5	1.540221	0.224019	1.379979 - 1.700464	1.694561
	10	2.206874	0.228472	2.043446 - 2.370302	2.355072
	15	2.980529	0.311809	2.757490 - 3.203568	3.051282
	20	3.879847	0.237309	3.710098 - 4.049596	3.996176
	25	4.628990	0.304162	4.411421 - 4.846560	4.673077
	30	5.221720	0.266694	5.030952 - 5.412488	5.305623
LC	5	1.675460	0.236057	1.506607 - 1.844313	1.784141
	10	2.536784	0.265849	2.346621 - 2.726948	2.693370
	15	3.531662	0.365589	3.270154 - 3.793170	3.573574
	20	4.669476	0.350823	4.418529 - 4.920422	4.764438
	25	5.588448	0.271811	5.394020 - 5.782877	5.737898
	30	6.508392	0.387783	6.231008 - 6.785776	6.724638
DCCL	5	1.242038	0.072523	1.190162 - 1.293914	1.267606
	10	1.520611	0.126534	1.430101 - 1.611122	1.608911
	15	1.555322	0.112813	1.474626 - 1.636018	1.630435

continues ...

. . . continued

Algo	**Matrix Size**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Max**
	20	1.582796	0.089596	1.518707 - 1.646884	1.642221
	25	1.669169	0.069972	1.619118 - 1.719221	1.695743
	30	1.624923	0.061392	1.581009 - 1.668838	1.668589
RDCC	5	1.405209	0.165236	1.287014 - 1.523404	1.492537
	10	1.961613	0.178288	1.834083 - 2.089144	2.070064
	15	2.286774	0.105105	2.211591 - 2.361956	2.232645
	20	2.330783	0.120055	2.244907 - 2.416660	2.392405
	25	2.403339	0.150273	2.295848 - 2.510830	2.475248
	30	2.394262	0.145944	2.289867 - 2.498656	2.491944
CPPS	5	1.743912	0.261948	1.556538 - 1.931285	1.901408
	10	2.575129	0.278451	2.375951 - 2.774307	2.655738
	15	3.674746	0.380468	3.402595 - 3.946898	3.672840
	20	4.818484	0.253144	4.637409 - 4.999559	4.929245
	25	5.742957	0.354076	5.489684 - 5.996230	5.905225
	30	6.726351	0.345334	6.479331 - 6.973371	6.838235
LOCAL	5	1.543211	0.224019	1.382969 - 1.703454	1.697551
	10	2.212244	0.228472	2.048816 - 2.375672	2.360442
	15	2.991901	0.311809	2.768862 - 3.214940	3.062654
	20	3.889696	0.237309	3.719947 - 4.059444	4.006025
	25	4.642070	0.304162	4.424501 - 4.859639	4.763813
	30	5.247586	0.266694	5.056818 - 5.438354	5.331489
EPS	5	1.737394	0.265410	1.547544 - 1.927244	1.657459
	10	2.554259	0.243351	2.380189 - 2.728330	2.571429
	15	3.675646	0.440201	3.360767 - 3.990525	3.672840
	20	4.792433	0.302521	4.576037 - 5.008828	4.904051
	25	5.782085	0.416057	5.484477 - 6.079693	6.078799
	30	6.797980	0.314994	6.572662 - 7.023297	6.863905

TABLE 4.8: Statistical analysis of speedup results obtained for Gaussian Elimination task graphs with CCR value 10.

Algo	**Matrix Size**	**Mean**	**Std. Dev.**	**CI (95 %)**	**Max**
EZ	5	1.001744	0.005516	0.997799 - 1.005690	1.000000
	10	1.021866	0.027107	1.002476 - 1.041256	1.037736
	15	1.102288	0.108017	1.025023 - 1.179553	1.151222
	20	1.247146	0.108644	1.169432 - 1.324859	1.307885
	25	1.451475	0.088027	1.388508 - 1.514441	1.431992
	30	1.486830	0.120333	1.400755 - 1.572906	1.558093
LC	5	0.376359	0.056857	0.335689 - 0.417028	0.415385
	10	0.550201	0.060818	0.506698 - 0.593704	0.582437
	15	0.759926	0.055832	0.719988 - 0.799863	0.775750
	20	0.976200	0.040810	0.947009 - 1.005392	1.004486
	25	1.212297	0.070057	1.162184 - 1.262409	1.228431
	30	1.384120	0.045116	1.351848 - 1.416391	1.411192
DCCL	5	1.007592	0.016506	0.995786 - 1.019399	1.000000
	10	1.025750	0.015136	1.014923 - 1.036578	1.032864
	15	1.020693	0.019911	1.006450 - 1.034936	1.031746
	20	1.048748	0.023240	1.032124 - 1.065372	1.048835
	25	1.057182	0.037941	1.030042 - 1.084321	1.074983
	30	1.084887	0.034581	1.060151 - 1.109623	1.101614
RDCC	5	0.513685	0.100382	0.441881 - 0.585489	0.534653
	10	0.653223	0.067769	0.604747 - 0.701698	0.695279
	15	0.840734	0.082981	0.781377 - 0.900090	0.894040
	20	1.037117	0.088454	0.973845 - 1.100389	1.067781
	25	1.199174	0.082449	1.140197 - 1.258150	1.236327
	30	1.326238	0.084253	1.265971 - 1.386505	1.356533
CPPS	5	0.985784	0.044954	0.953628 - 1.017940	1.000000
	10	1.029767	0.191720	0.892628 - 1.166905	1.098170
	15	1.307535	0.199349	1.164939 - 1.450131	1.410658
	20	1.427548	0.220981	1.269478 - 1.585617	1.503979
	25	1.644583	0.274463	1.448258 - 1.840908	1.606663
	30	1.856913	0.221237	1.698661 - 2.015166	1.916961

continues ...

... continued

Algo	Matrix Size	Mean	Std. Dev.	CI (95 %)	Max
LOCAL	5	1.004734	0.005516	1.000789 - 1.008680	1.002990
	10	1.027236	0.027107	1.007846 - 1.046626	1.043106
	15	1.113660	0.108017	1.036395 - 1.190925	1.162594
	20	1.256994	0.108644	1.179281 - 1.334708	1.317734
	25	1.464555	0.088027	1.401588 - 1.527521	1.445072
	30	1.512696	0.120333	1.426621 - 1.598772	1.583959
EPS	5	0.913353	0.223744	0.753307 - 1.073398	1.000000
	10	1.069342	0.162086	0.953400 - 1.185284	1.176471
	15	1.314262	0.219753	1.157071 - 1.471453	1.409214
	20	1.474012	0.211063	1.323037 - 1.624987	1.603905
	25	1.647633	0.123603	1.559218 - 1.736047	1.673040
	30	2.003550	0.164218	1.886084 - 2.121017	2.081340

4.3.2.2 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 perform experiments.

The average NSL results of FFT graphs for the different number input points are shown in Fig. 4.8. For input points, 2 to 16, EPS, CPPS and LOCAL yielded the similar results to that of EZ but for input points 32, EZ performs best. From these sample of results, we can conclude that EPS does not achieve better results than EZ when all tasks belong to a critical path, i.e., when all paths are critical. Overall, the EPS algorithm produces an improvement of -3.11, 32.51, 3.33, 44.75, 32.70, and 1.74 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

The average NSL results of FFT graphs for the different values of CCR are shown in Fig. 4.9. For CCR = 0.10, the EPS algorithm produces an improvement of 1.62, 0.08, 1.07, 84.86, 75.21, and 0.51 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For CCR = 1, the EPS algorithm produces an improvement of 1.79, 0.97, 4.72, 73.37, 64.04, and 3.36 percent over

the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For CCR = 10, the EPS algorithm produces an improvement of -4.92, 37.45, 8.86, 31.73, 17.32, and 8.00 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

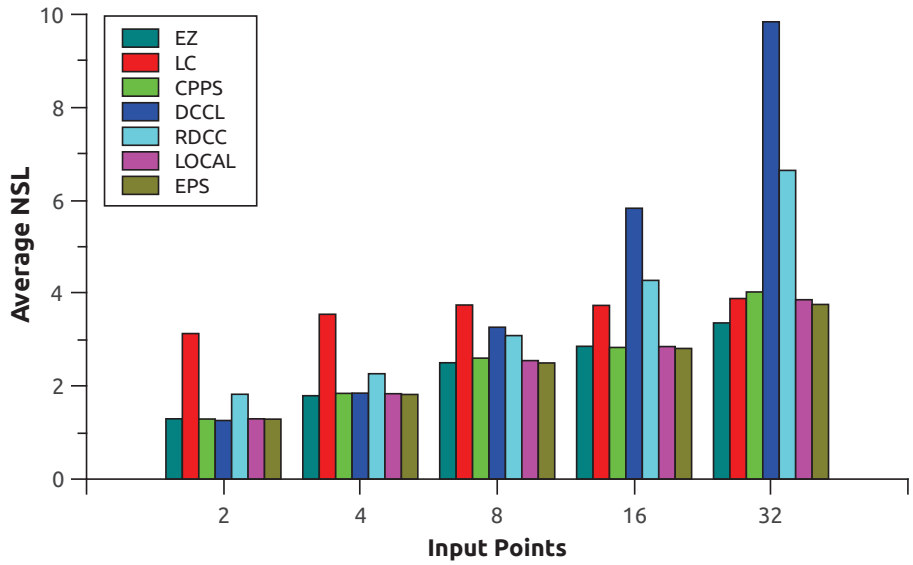

FIGURE 4.8: Average NSL results obtained for FFT task graphs as a function of input points.

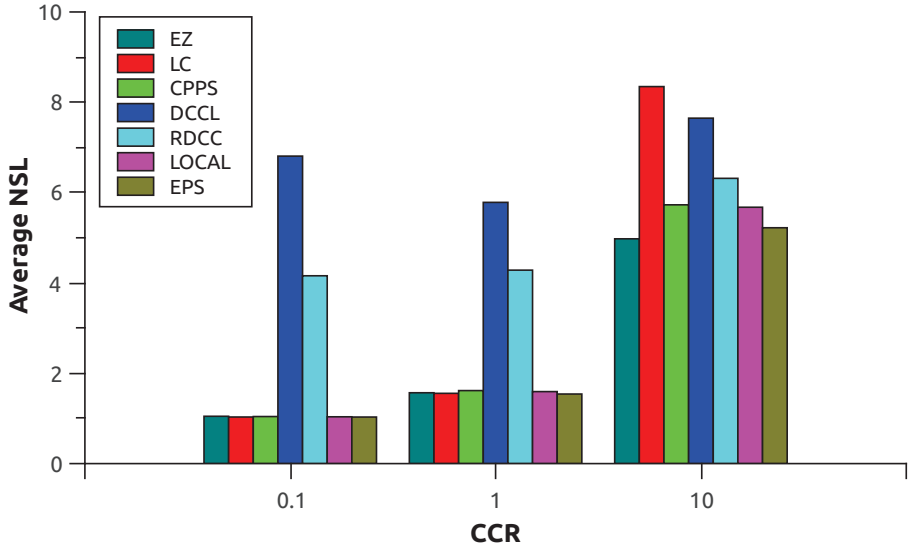

FIGURE 4.9: Average NSL results obtained for FFT task graphs as a function of CCR.

The average speedup results of FFT graphs for the different number of input points are shown in Fig. 4.10. For each set of task graphs, the EPS algorithm produces better schedules than other algorithms. Overall, the EPS algorithm produces an improvement of 2.62, 1.55, 66.03, 56.68, 1.08 percent over the LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

The average speedup results of FFT graphs for the different values of CCR are shown in Fig. 4.11. For CCR = 0.10, the EPS algorithm produces an improvement of 2.95, 0.55, 2.57, 85.27, 77.97, and 1.85 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For CCR = 1, the EPS algorithm produces an improvement of 4.68, 0.29, 9.39, 77.28, 69.66, and 3.20 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively. For CCR = 10, the EPS algorithm produces an improvement of -0.77, 30.22, 12.14, 41.43, 34.62, and 6.96 percent over the EZ, LC, CPPS, DCCL, RDCC and LOCAL scheduling algorithms respectively.

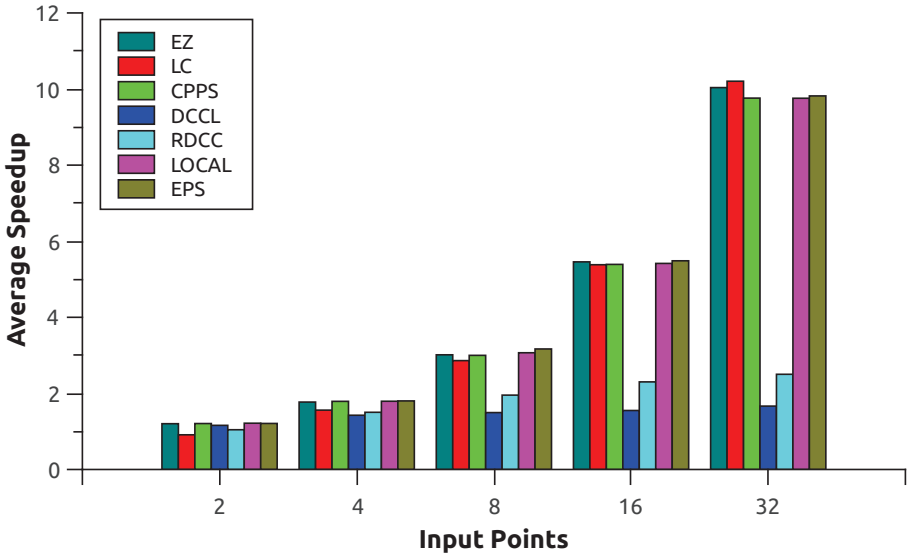

FIGURE 4.10: Average speedup results obtained for FFT task graphs as a function of input points.

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 4.9 and Table 4.10 present the statistical analysis of NSL results obtained for FFT task graphs with CCR value 1 and 10 respectively. All NSL values obtained for FFT task graphs with CCR value 0.1 for all algorithms

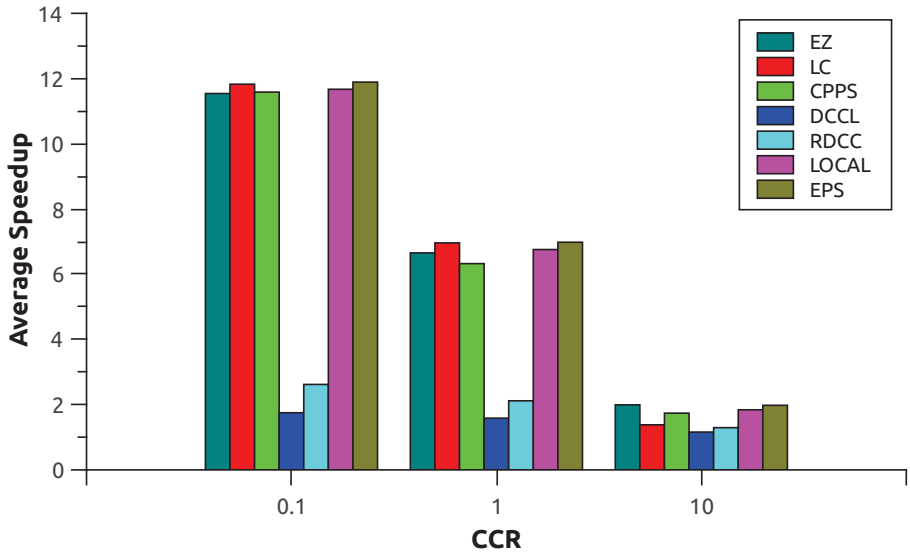

FIGURE 4.11: Average speedup results obtained for FFT task graphs as a function of CCR.

except DCCL and RDCC are similar and nearly equal to 1. Hence, we have not given the table for that. Table 4.11, Table 4.12 and Table 4.13 present the statistical analysis of speedup results obtained 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 4.9: Statistical analysis of NSL obtained for FFT task graphs with CCR value 1.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Min
EZ	2	1.334699	0.175496	1.209165 - 1.460233	1.210526
	4	1.539419	0.172479	1.416044 - 1.662795	1.465116
	8	1.534179	0.151103	1.426094 - 1.642264	1.509652
	16	1.558178	0.131256	1.464290 - 1.652067	1.500873
	32	1.739315	0.097966	1.669240 - 1.809391	1.674157
LC	2	1.468005	0.118234	1.383432 - 1.552579	1.398148
	4	1.540018	0.154786	1.429299 - 1.650738	1.473684
	8	1.566167	0.113329	1.485101 - 1.647232	1.500000
	16	1.514619	0.098751	1.443982 - 1.585256	1.456480
	32	1.650943	0.118648	1.566073 - 1.735813	1.595506

continues . . .

. . . continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Min
DCCL	2	1.276542	0.137729	1.178023 - 1.375060	1.201923
	4	1.640082	0.145315	1.536137 - 1.744027	1.574219
	8	2.649998	0.182628	2.519363 - 2.780634	2.614776
	16	4.642568	0.377325	4.372665 - 4.912471	4.513793
	32	8.652548	0.696100	8.154622 - 9.150473	8.302158
RDCC	2	1.266636	0.108954	1.188700 - 1.344571	1.201342
	4	1.502011	0.107605	1.425040 - 1.578982	1.528000
	8	1.888603	0.139163	1.789059 - 1.988147	2.054348
	16	2.976687	0.423252	2.673931 - 3.279442	2.776736
	32	6.564482	1.239240	5.678045 - 7.450920	5.819512
CPPS	2	1.325345	0.160509	1.210531 - 1.440158	1.263158
	4	1.477627	0.135435	1.380749 - 1.574505	1.406977
	8	1.614581	0.205646	1.467481 - 1.761681	1.475155
	16	1.644476	0.087366	1.581983 - 1.706969	1.607330
	32	1.813140	0.133371	1.717739 - 1.908541	1.763889
LOCAL	2	1.330022	0.167215	1.210412 - 1.449632	1.210526
	4	1.460814	0.129247	1.368362 - 1.553265	1.436842
	8	1.531361	0.155123	1.420401 - 1.642322	1.420849
	16	1.601327	0.099901	1.529867 - 1.672787	1.536828
	32	1.669389	0.110696	1.590207 - 1.748570	1.647197
EPS	2	1.320826	0.156363	1.208979 - 1.432673	1.210526
	4	1.483410	0.148621	1.377100 - 1.589719	1.442105
	8	1.610995	0.219558	1.453943 - 1.768046	1.525097
	16	1.630806	0.094973	1.562872 - 1.698741	1.578182
	32	1.804828	0.133970	1.708999 - 1.900658	1.722222

Table 4.10: Statistical analysis of NSL obtained for FFT task graphs with CCR value 10.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Min
EZ	2	1.535178	0.264862	1.345720 - 1.724636	1.351351
	4	2.798782	0.299760	2.584361 - 3.013202	2.739726

continues . . .

. . . continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Min
	8	4.931114	0.616373	4.490218 - 5.372010	4.642857
	16	5.964534	1.087059	5.186953 - 6.742116	5.244019
	32	7.287859	0.761986	6.742805 - 7.832913	6.813187
LC	2	6.907552	1.100070	6.120664 - 7.694439	6.329412
	4	8.062897	1.093623	7.280620 - 8.845173	7.283105
	8	8.640522	0.864660	8.022025 - 9.259020	8.593074
	16	8.663774	1.253405	7.767205 - 9.560344	7.822472
	32	8.977027	0.876386	8.350142 - 9.603913	8.750831
DCCL	2	1.395272	0.149356	1.288437 - 1.502108	1.315789
	4	2.380979	0.188038	2.246473 - 2.515484	2.334630
	8	4.331932	0.213359	4.179315 - 4.484549	4.227273
	16	7.568642	0.473967	7.229610 - 7.907674	7.576710
	32	10.707136	1.121147	9.905171 - 11.509101	9.559165
RDCC	2	3.180650	1.274628	2.268899 - 4.092400	3.361905
	4	4.110171	0.543537	3.721375 - 4.498966	3.905406
	8	5.511448	0.216027	5.356922 - 5.665973	5.375000
	16	6.486051	0.396696	6.202291 - 6.769810	6.315920
	32	7.328359	0.536937	6.944285 - 7.712434	6.955102
CPPS	2	1.535178	0.264862	1.345720 - 1.724636	1.351351
	4	3.025705	0.805035	2.449858 - 3.601552	2.459016
	8	5.160364	1.769384	3.894711 - 6.426017	3.903615
	16	5.800724	1.419780	4.785146 - 6.816303	4.934109
	32	9.218547	1.270326	8.309874 - 10.127221	8.420168
LOCAL	2	1.535178	0.264862	1.345720 - 1.724636	1.351351
	4	2.857245	0.502939	2.497489 - 3.217000	2.610465
	8	4.726687	0.961209	4.039127 - 5.414247	4.220588
	16	5.882629	0.813182	5.300955 - 6.464304	5.723485
	32	7.443456	0.821129	6.856097 - 8.030816	7.088028
EPS	2	1.535178	0.264862	1.345720 - 1.724636	1.351351
	4	3.400680	1.943873	2.010214 - 4.791147	2.459016
	8	5.600593	1.711376	4.376434 - 6.824752	4.393939

continues . . .

... continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Min
	16	6.708163	1.521448	5.619860 - 7.796466	5.932990
	32	9.026162	0.876068	8.399503 - 9.652820	8.537815

TABLE 4.11: Statistical analysis of speedup results obtained for FFT task graphs with CCR value 0.1.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
EZ	2	1.529400	0.040971	1.500093 - 1.558706	1.526252
	4	2.695470	0.065141	2.648874 - 2.742066	2.741228
	8	4.917232	0.113443	4.836085 - 4.998379	4.949239
	16	9.265878	0.235649	9.097316 - 9.434439	9.350394
	32	17.465185	0.492418	17.112956 - 17.817415	17.765385
LC	2	1.532930	0.046523	1.499652 - 1.566208	1.558326
	4	2.712915	0.054276	2.674090 - 2.751739	2.740978
	8	4.982905	0.081156	4.924853 - 5.040956	5.034423
	16	9.402434	0.155662	9.291088 - 9.513780	9.396637
	32	17.90611	0.395848	17.622958 - 18.189263	18.137454
DCCL	2	1.419751	0.138866	1.320419 - 1.519083	1.507159
	4	1.841614	0.146828	1.736587 - 1.946640	1.939416
	8	1.817917	0.106455	1.741769 - 1.894065	1.871401
	16	1.843650	0.083336	1.784039 - 1.903261	1.867873
	32	1.810839	0.065111	1.764264 - 1.857413	1.836147
RDCC	2	1.519893	0.053328	1.481747 - 1.558039	1.554174
	4	2.363988	0.184874	2.231746 - 2.496230	2.463863
	8	2.748028	0.127099	2.657113 - 2.838944	2.781344
	16	2.917023	0.312501	2.693489 - 3.140557	3.102885
	32	3.057757	0.191249	2.920955 - 3.194558	3.190141
CPPS	2	1.536325	0.043546	1.505176 - 1.567474	1.558326
	4	2.731317	0.053977	2.692707 - 2.769927	2.767528
	8	4.973397	0.131598	4.879264 - 5.067530	5.012853
	16	9.232749	0.236540	9.063550 - 9.401947	9.382054
	32	17.561844	0.311008	17.339377 - 17.78431	17.739084

continues ...

. . . continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
LOCAL	2	1.532862	0.040587	1.503830 - 1.561894	1.533474
	4	2.733581	0.054876	2.694329 - 2.772834	2.767528
	8	4.992277	0.112970	4.911469 - 5.073085	5.069461
	16	9.249313	0.211736	9.097857 - 9.400769	9.314143
	32	17.795164	0.371443	17.529468 - 18.060859	18.060103
EPS	2	1.540020	0.038639	1.512381 - 1.567659	1.558326
	4	2.733727	0.053118	2.695731 - 2.771722	2.767528
	8	4.963503	0.130156	4.870402 - 5.056604	5.043103
	16	9.327308	0.246617	9.150901 - 9.503714	9.500000
	32	17.65928	0.347687	17.410577 - 17.907983	17.90206

Table 4.12: Statistical analysis of speedup results obtained for FFT task graphs with CCR value 1.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
EZ	2	1.080478	0.076218	1.025958 - 1.134997	1.229508
	4	1.620973	0.100288	1.549236 - 1.692710	1.679104
	8	3.073904	0.115539	2.991258 - 3.156550	3.145161
	16	5.484071	0.278087	5.285153 - 5.682989	5.621302
	32	10.076141	0.508053	9.712727 - 10.439555	10.388199
LC	2	0.980119	0.100938	0.907917 - 1.052320	1.024590
	4	1.619242	0.109351	1.541023 - 1.697462	1.630435
	8	3.004226	0.083203	2.944710 - 3.063742	3.046875
	16	5.632963	0.246418	5.456698 - 5.809227	5.668258
	32	10.623179	0.505977	10.261250 - 10.985108	10.686901
DCCL	2	1.060520	0.083783	1.000589 - 1.120450	1.086957
	4	1.435838	0.148217	1.329818 - 1.541859	1.515152
	8	1.648045	0.109797	1.569507 - 1.726584	1.642797
	16	1.736723	0.109535	1.658371 - 1.815074	1.814362
	32	1.819045	0.102385	1.745808 - 1.892281	1.842366
RDCC	2	1.066790	0.081151	1.008742 - 1.124838	1.117318
	4	1.559226	0.059126	1.516932 - 1.601519	1.590909

continues . . .

... continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
	8	2.312413	0.138865	2.213081 - 2.411744	2.407407
	16	2.738951	0.331636	2.501730 - 2.976173	2.950311
	32	2.458218	0.412207	2.163363 - 2.753072	2.369819
CPPS	2	1.086857	0.074718	1.033410 - 1.140303	1.098901
	4	1.685791	0.113646	1.604499 - 1.767083	1.721939
	8	2.935446	0.252130	2.755095 - 3.115796	2.849687
	16	5.192878	0.375625	4.924190 - 5.461565	5.388979
	32	9.676248	0.510972	9.310746 - 10.041750	9.911111
LOCAL	2	1.083667	0.074502	1.030375 - 1.136959	1.117318
	4	1.706995	0.132827	1.611983 - 1.802008	1.800029
	8	3.094643	0.136358	2.997105 - 3.192180	3.179348
	16	5.338474	0.287634	5.132728 - 5.544221	5.488722
	32	10.580427	0.448217	10.259814 - 10.90104	10.657336
EPS	2	1.090015	0.071179	1.039100 - 1.140930	1.117318
	4	1.681263	0.127836	1.589821 - 1.772705	1.721939
	8	2.945844	0.255285	2.763237 - 3.128451	3.095238
	16	5.234277	0.319923	5.005434 - 5.463120	5.459770
	32	9.716942	0.416238	9.419204 - 10.01468	9.970194

Table 4.13: Statistical analysis of speedup results obtained for FFT task graphs with CCR value 10.

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
EZ	2	1.000000	0.000000	1.000000 - 1.000000	1.000000
	4	1.000000	0.000000	1.000000 - 1.000000	1.000000
	8	1.051848	0.070553	1.001381 - 1.102315	1.022727
	16	1.621286	0.084040	1.561172 - 1.681400	1.680141
	32	2.572384	0.083878	2.512386 - 2.632383	2.612771
LC	2	0.221912	0.006914	0.216966 - 0.226858	0.225632
	4	0.348623	0.018524	0.335373 - 0.361874	0.347544
	8	0.598029	0.026377	0.579161 - 0.616896	0.604839
	16	1.110043	0.030415	1.088287 - 1.131799	1.124527

continues ...

. . . continued

Algo	Input Points	Mean	Std. Dev.	CI (95 %)	Max
	32	2.086623	0.058837	2.044536 - 2.128709	2.163848
DCCL	2	1.000000	0.000000	1.000000 - 1.000000	1.000000
	4	1.000000	0.000000	1.000000 - 1.000000	1.000000
	8	1.021324	0.017550	1.008770 - 1.033878	1.021611
	16	1.071553	0.046343	1.038403 - 1.104703	1.096998
	32	1.368257	0.109262	1.290101 - 1.446413	1.425224
RDCC	2	0.547813	0.312567	0.324231 - 0.771394	0.382263
	4	0.586828	0.078402	0.530747 - 0.642909	0.579151
	8	0.803388	0.043217	0.772474 - 0.834301	0.833333
	16	1.251252	0.075300	1.197390 - 1.305114	1.287961
	32	1.990791	0.108476	1.913198 - 2.068385	2.012232
CPPS	2	1.000000	0.000000	1.000000 - 1.000000	1.000000
	4	0.955970	0.139235	0.856374 - 1.055566	1.000000
	8	1.090776	0.304889	0.872686 - 1.308865	1.300000
	16	1.746214	0.460873	1.416548 - 2.075879	2.075369
	32	2.046483	0.164667	1.928696 - 2.164271	2.105539
LOCAL	2	1.000000	0.000000	1.000000 - 1.000000	1.000000
	4	0.998979	0.078515	0.942817 - 1.055142	1.050459
	8	1.146896	0.174667	1.021956 - 1.271836	1.264447
	16	1.683750	0.222723	1.524435 - 1.843065	1.822826
	32	2.673507	0.120791	2.587104 - 2.759910	2.754677
EPS	2	1.000000	0.000000	1.000000 - 1.000000	1.000000
	4	0.932120	0.214656	0.778575 - 1.085665	1.000000
	8	0.994237	0.278254	0.795200 - 1.193274	1.162098
	16	1.503636	0.413891	1.207576 - 1.799695	1.315789
	32	2.080871	0.184032	1.949231 - 2.212510	2.194882

4.4 Summary

We have proposed and illustrated here an edge-based clustering algorithm that makes use of the concept of edge zeroing heuristics, and we name it as Edge Priority

Scheduling (EPS) algorithm, for the problem of scheduling in multiprocessors. The EPS algorithm goes for edge zeroing by using edge priorities for clustering the tasks. The complexity of the EPS algorithm works out to be $O(|V||E|(|V| + |E|))$, where $|E|$ represents the number of edges and $|V|$ denotes the number of tasks in the task graph. The performance of this EPS algorithm is compared with six well-known clustering based scheduling algorithms, namely EZ, LC, CPPS, DCCL, RDCC, and LOCAL. The comparative study is based on two types of task graphs, randomly generated benchmark task graphs and task graphs that correspond to some real-world applications. The task graphs derived from real-world applications are Gaussian Elimination and Fast Fourier Transform.

It is concluded from the experimental results that for randomly generated application graphs, the EPS algorithm proposed here produces considerably encouraging results than EZ, DCCL, and RDCC and gives similar but slightly better performance than CPPS and LOCAL in terms of average normalized schedule length and average speedup. For Gaussian elimination application graphs, EPS produces best among the shown results for all matrix sizes used in the experiments except for matrix size 5, and it also gives outstanding among the shown results for communication-intensive graphs. In case of Fast Fourier Transform application graphs, for input points, 2 to 16, EZ, CPPS, LOCAL and EPS perform similarly well but for input points 32, EZ gives slightly better results than other three algorithms. For CCR value 0.1 and 1, again EZ, CPPS, LOCAL and EPS provide similar schedules while for CCR value 10, EZ give slightly better schedules than other three algorithms as in case of FFT application graphs, all paths are critical. Therefore, except LC, DCCL, and RDCC, all algorithms including proposed one perform well and produce similar results. 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. The proposed task scheduling algorithm may further be considered for suitably extending the same for heterogeneous multiprocessors, or, may be integrated with the existing duplication based task scheduling strategies for different real-world applications.

In this chapter, we have seen how edges can be ordered in non-increasing order as per the edge priority considering all edges of the task graph. It will be interesting to

see whether a clustering-based algorithm be possible examining and using possibly fever number of edges. The idea is made use of in next chapter for working out a scheduling algorithm based on the consideration of the critical path.