Applied Mathematical Modelling 36 (2012) 6243–6263

Contents lists available at SciVerse ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules

Pramod Kumar Mishra [a,*], Abhishek Mishra [b], Kamal Sheel Mishra [c], Anil Kumar Tripathi [b]

[a] Department of Computer Science & DST Centre for Interdisciplinary Mathematical Sciences, Banaras Hindu University, Varanasi 221 005, India
[b] Department of Computer Engineering, Institute of Technology, Banaras Hindu University, Varanasi 221 005, India
[c] Department of Computer Science, School of Management Sciences, Varanasi 221 011, India

ARTICLE INFO

Article history:
Received 17 July 2010
Received in revised form 15 January 2012
Accepted 3 February 2012
Available online 15 February 2012

Keywords:
Benchmarking
Clustering
Distributed computing
Homogeneous systems
Scheduling
Task allocation

ABSTRACT

In this paper we give some extensive benchmark results for some dynamic priority clustering algorithms for homogeneous multiprocessor environments. By dynamic priority we mean a priority function that can change with every step of the algorithm. Using dynamic priority can give us more flexibility as compared to static priority algorithms. Our objective in this paper is to compare the dynamic priority algorithms with some well known algorithms from the literature and discuss their strengths and weaknesses. For our study we have selected two recently proposed dynamic priority algorithms: CPPS (*Cluster Pair Priority Scheduling algorithm*) having complexity $O(|V||E|(|V|+|E|))$ and DCCL (*Dynamic Computation Communication Load scheduling algorithm*) having complexity $O(|V|^2(|V|+|E|)log(|V|+|E|))$ where $|V|$ is the number of nodes in the task graph, and $|E|$ is the number of edges in the task graph. We have selected a recently proposed randomized algorithm with static priority (RCCL: *Randomized Computation Communication Load scheduling algorithm*) and converted it into a dynamic priority algorithm: RDCC (*Randomized Dynamic Computation Communication load scheduling algorithm*) having complexity $O(ab|V|(|V|+|E|)log(|V|+|E|))$ where a is the number of randomization steps, and b is a limit on the number of clusters formed. We have also selected three well known algorithms from literature: DSC (*Dominant Sequence Clustering algorithm*) having complexity $O((|V|+|E|)log(|V|))$, EZ (*Edge Zeroing algorithm*) having complexity $O(|E|(|V|+|E|))$, and LC (*Linear Clustering algorithm*) having complexity $O(|V|(|V|+|E|))$. We have compared these algorithms using various comparison parameters including some statistical parameters, and also using various types of task graphs including some synthetic and real task graphs. Our results show that the dynamic priority algorithms give best results for the case of random task graphs, and for the case when the number of available processors are small.

1. Introduction

A parallel system is designed so that it can execute the applications faster than a sequential system. For this we need to parallelize the program. The three steps involved in the parallelization of a program [1] are: task decomposition, dependence analysis, and scheduling. By scheduling we mean both the temporal allocation (assigning a start time to the task) and the

* Corresponding author.
E-mail addresses: mishra@bhu.ac.in (P.K. Mishra), abhishek.rs.cse@itbhu.ac.in (A. Mishra), ksmishra@smsvaranasi.com (K.S. Mishra), aktripathi.cse@itbhu.ac.in (A.K. Tripathi).

0307-904X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.apm.2012.02.011

spatial allocation (assigning a processor to the task). The tasks are allocated on different processors to exploit the parallelism so that the parallel execution time of the tasks can be reduced.

A dependence relation among the tasks is represented as a directed acyclic graph, also known as the *task graph*. Nodes in the task graph represent the tasks and have a weight associated with them that represents the running time of the task. Edges in the task graph represent the dependence relation among the tasks and have a weight associated with them that represents the communication time among the tasks.

The problem of finding a schedule for a given task graph on a given set of processors that takes minimum time is NP-Complete [2, 3].

A fundamental scheduling heuristic is called the *list scheduling* heuristic. In list scheduling, first we assign a priority scheme to the tasks. Then we sort the tasks according to the priority scheme, while respecting the precedence constraints of the tasks. Finally each task is successively scheduled on a processor chosen for it. Some examples of list scheduling algorithms are: Adam et al. [4], Coffman and Graham [5], Graham [6], Hu [7], Kasahara and Nartia [8], Lee et al. [9], Liu [10], Wu and Gajski [11], Yang and Gerasoulis [12].

Another fundamental scheduling heuristic is called *clustering*. Basically it is a scheduling technique for an unlimited number of processors. It is often proposed as an initial step in scheduling for a limited number of processors. A cluster is a set of tasks that are scheduled on the same processor. Clustering based scheduling algorithms generally consist of three steps. The first step finds a clustering of the task graph. The second step finds an allocation of clusters to the processors. The last step finds a scheduling of the tasks. Some examples of clustering based scheduling algorithms are Mishra et al. [13], Yang and Gerasoulis [14], Kim and Browne [15], Kadamuddi and Tsai [16], Sarkar [2], Hanen and Munier [17].

Several benchmarking for task scheduling algorithms are proposed in literature [18, 19, 20, 21]. In this paper we give some extensive benchmark results for some dynamic priority clustering algorithms for homogeneous multiprocessor environments. By dynamic priority we mean a priority function that can change with every step of the algorithm. Using dynamic priority can give us more flexibility as compared to static priority algorithms. Our objective in this paper is to compare the dynamic priority algorithms with some well known algorithms from the literature and discuss their strengths and weaknesses. For our study we have selected two recently proposed dynamic priority algorithms: CPPS (*Cluster Pair Priority Scheduling algorithm*) [22], having complexity $O(|V||E|(|V|+|E|))$ and DCCL (*Dynamic Computation Communication Load scheduling algorithm*) [23], having complexity $O(|V|^2(|V|+|E|)log(|V|+|E|))$ where $|V|$ is the number of nodes in the task graph, and $|E|$ is the number of edges in the task graph. We have selected a recently proposed randomized algorithm with static priority RCCL (*Randomized Computation Communication Load scheduling algorithm*) [24], and converted it into a dynamic priority algorithm: RDCC (*Randomized Dynamic Computation Communication load scheduling algorithm*) having complexity $O(ab|V|(|V|+|E|)log(|V|+|E|))$ where a is the number of randomization steps, and b is a limit on the number of clusters formed. We have also selected three well known algorithms from literature: DSC (*Dominant Sequence Clustering algorithm*) [14], having complexity $O((|V|+|E|)log(|V|))$, EZ (*Edge Zeroing algorithm*) [2], having complexity $O(|E|(|V|+|E|))$, and LC (*Linear Clustering algorithm*) [15], having complexity $O(|V|(|V|+|E|))$. We have compared these algorithms using various comparison parameters including some statistical parameters, and also using various types of task graphs including some synthetic and real task graphs. Our results show that the dynamic priority algorithms give best results for the case of random task graphs, and for the case when the number of available processors are small.

The rest of the paper is organized in the following manner: Section 2 presents a description of algorithms used in the benchmarking, Section 3 gives a description of performance evaluation parameters used, Section 4 gives a description of task graphs used, Section 5 gives the performance results for peer set task graphs, Section 6 gives the performance results for random task graphs, Section 7 gives the performance results for systolic array task graphs, Section 8 gives the performance results for Gaussian elimination task graphs, Section 9 gives the performance results for divide and conquer task graphs, Section 10 gives the performance results for fast Fourier transform task graphs, Section 11 gives the performance results for small random task graphs with optimal solutions, and finally we conclude in Section 12.

2. A description of algorithms used in the benchmarking

2.1. The CPPS algorithm

Mishra and Tripathi [22] consider the EZ algorithm [2] for scheduling precedence constrained task graphs on parallel systems as a priority based algorithm in which the priority is assigned to edges. In this case, the priority can be taken as the edge weight. This can be viewed as a task dependent priority function that is defined for pairs of tasks. In the CPPS algorithm [22] this idea is extended in which the priority is a cluster dependent function of pairs of clusters (of tasks):

$$P_c(C_i, C_j) = comm(C_i, C_j) + comm(C_j, C_i) - comp(C_i) - comp(C_j), \tag{1}$$

where $P_c(C_i, C_j)$ is the priority function that is defined for a pair of clusters, $comm(C_i, C_j)$ is the total communication cost from the cluster C_i to the cluster C_j, $comm(C_j, C_i)$ is the total communication cost from the cluster C_j to the cluster C_i, $comp(C_i)$ is the total computation cost of the cluster C_i, and $comp(C_j)$ is the total computation cost of the cluster C_j. This can be viewed as a dynamic priority algorithm that depends on the current allocation in each step of the algorithm. The CPPS algorithm has complexity $O(|V||E|(|V|+|E|))$.

2.2. The DCCL algorithm

In the DCCL algorithm [23], priority of modules is dependent on the computation and the communication times associated with the module as well as the current allocation. *DCCLoad* of a module is defined as follows:

$$DCCLoad_i = (c\_in_i + c\_out_i)m_i - sum\_in_i - sum\_out_i, \tag{2}$$

where,

$$c\_in_i = \sum_{cluster[j] \neq cluster[i], 1 \leqslant j \leqslant n} 1, \tag{3}$$

$$c\_out_i = \sum_{cluster[i] \neq cluster[k], 1 \leqslant k \leqslant n} 1, \tag{4}$$

$$sum\_in_i = \sum_{cluster[j] \neq cluster[i], 1 \leqslant j \leqslant n} w_{ji} \tag{5}$$

and

$$sum\_out_i = \sum_{cluster[i] \neq cluster[k], 1 \leqslant k \leqslant n} w_{ik}. \tag{6}$$

For calculating $DCCLoad_i$ of a module M_i, we first multiply its running time (m_i) with the number of those incoming edges from, and outgoing edges to, ($c\_in_i + c\_out_i$) that are allocated on different clusters from M_i. Then we subtract the result by the sum of weight of incoming edges that are allocated on different clusters (sum_in_i) subtracted by the sum of weight of outgoing edges that are allocated on different clusters (sum_out_i).

Initially the modules are allocated in a single cluster. Modules are taken out in decreasing order of priority and the priorities are recalculated. This makes it a dynamic priority algorithm. The complexity of the DCCL algorithm is $O(|V|^2(|V| + |E|)log(|V| + |E|))$.

2.3. The DSC algorithm

The DSC algorithm [14] finds the critical path of the graph. The critical path is called the DS (*Dominant Sequence*). An edge from the DS is used to merge its adjacent nodes, provided the parallel time reduces. After merging, a new DS is computed and the clustering is tried again. The DSC algorithm has complexity $O((|V| + |E|)log(|V|))$.

2.4. The EZ algorithm

The EZ algorithm [2] reduces the parallel time at each step by considering the highest cost edge. This can be viewed as a static priority algorithm in which the priority is assigned to edges. The approach can summarized by three steps. The first step sorts the edges of the task graph in descending order of edge costs. The second step merges the clusters connected by the edge with the highest cost, if the parallel time does not increase. The third step repeats the second step until all edges are examined. During the second step, if the parallel time increases for an edge, then the two nodes connected by that edge are scheduled on separate processors. The EZ algorithm uses a level information to determine the parallel time and these levels are computed for each step. The EZ algorithm has complexity $O(|E|(|V| + |E|))$.

2.5. The LC algorithm

The LC algorithm [15] uses the CP (*Critical Path*) [15] information to create clusters in a parallel system. The approach can be summarized by four steps. The first step marks all the edges in the task graph as unexamined. The second step finds the CP composed of unexamined edges only. In the third step the nodes in the CP are clustered and all the edges incident upon the nodes in the newly created cluster are marked as examined The fourth step repeats the second and third steps until all the edges are examined. The LC algorithm has complexity $O(|V|(|V| + |E|))$.

2.6. The RDCC algorithm

The RDCC algorithm is the dynamic priority version (using the dynamic priority function of the DCCL algorithm proposed by Mishra et al. [23]) of the RCCL (*Randomized Computation Communication Load*) algorithm [24]. The RCCL algorithm is a generalization of the CCLC (*Computation Communication Load Clustering*) [13] algorithm with restricted number of clusters to reduce the complexity together with a randomization of the generalized algorithm. The RDCC algorithm has complexity $O(ab|V|(|V| + |E|)log(|V| + |E|))$ where a is the number of randomization steps, and b is a limit on the number of clusters formed. We have used $a = 100$, and $b = 10$.

3. A description of performance evaluation parameters used

3.1. Non-statistical parameters

By non-statistical parameters we mean the parameters that consider the performance of only a single algorithm or task graph. The CCR (*Communication to Computation Ratio*) [20] of a task graph is defined as the average communication cost divided by the average computation cost:

$$CCR = \left(\sum_{e\in E} w(e)/|E|\right) / \left(\sum_{v\in V} w(v)/|V|\right), \tag{7}$$

where E is the set of edges ($w(e)$ is the communication cost) and V is the set of vertices ($w(v)$ is the computation cost). The CCR of a task graph tells us whether the task graph is communication intensive ($CCR > 1$), computation intensive ($CCR < 1$), or is a balanced task graph ($CCR = 1$).

The NSL (*Normalized Schedule Length*) [20] of a schedule for a given task graph is defined as the SL (*Schedule Length*) divided by the sum of computation costs on the CP ($CP \subset V$, *Critical Path*) of the task graph:

$$NSL = SL/\sum_{v\in CP} w(v). \tag{8}$$

The sum of computation cost of the nodes on the CP gives a lower bound on the SL. So we have the following inequality for the NSL:

$$NSL \geqslant 1. \tag{9}$$

Besides these parameters we also consider the number of processors used and the running time of algorithms.

3.2. Statistical parameters

By statistical parameters we mean the parameters that consider the performance of all the algorithms or task graphs under consideration. The PDB (*Percentage Degradation from the Best solution*) [20], for SL's generated by an algorithm $A \subset G$ for a given group G of algorithms and a given task graph is defined as the percentage degradation from the least SL generated by the algorithms in G:

$$PDB = 100(SL(A) - \min(\{SL(X)|X \in G\}))/\min(\{SL(X)|X \in G\}). \tag{10}$$

Besides these parameters we also consider some standard statistical parameters like average, standard deviation, and correlation [25, 26]).

In order to evaluate the algorithms we use five types of task graphs: average NSL vs number of nodes, average NSL vs CCR, average percentage degradation from the best solution vs number of nodes, average number of processors used vs number of nodes, and average running time vs number of nodes.

4. A description of task graphs used

4.1. Peer set task graphs

The *peer set task graphs* are small sized task graphs that are used by various researchers and are documented to give examples of tracing the algorithms. We have used 9 such task graphs. These are (as listed in Table 1): a 16-node task graph (TG1) [27]; a 17-node task graph (TG2) [28]; a 9-node task graph (TG3) [29]; a 11-node task graph (TG4) [30]; a 18-node task graph (TG5) [11]; a 7-node task graph (TG6) [12]; a 7-node task graph (TG7) [13]; a 15-node task graph (TG8) [16]; and a 9-node task graph (TG9) [20].

Table 1
Schedule lengths generated for peer set task graphs.

Task graph	CPPS	DCCL	DSC	EZ	LC	RDCC
TG1	170	270	190	170	190	160
TG2	38	44	39	37	40	38
TG3	15	17	14	15	14	14
TG4	25	30	29	29	33	25
TG5	430	500	420	460	420	410
TG6	10	5	6	6	10	5
TG7	15	27	15	15	16	15
TG8	23	32	23	26	23	24
TG9	17	27	19	17	19	17
Average	82.56	105.78	83.89	86.11	85.00	78.67
Standard deviation	131.61	159.11	130.33	140.41	129.79	125.34

4.2. Random task graphs

We have selected 150 benchmark random task graphs of Davidovic and Crainic [19]. These task graphs are available online at Davidovic [31]. There are 30 task graphs each having number of nodes as 50, 100, 200, 300, 400, and 500. Out of these we have selected the task graphs having number of nodes as 50, 100, 200, 300, and 400 with a total of 150 task graphs.

4.3. Systolic array task graphs

Fig. 1 shows a sample systolic array task graph [32]. A systolic array task graph has n^2 nodes and $2n(n-1)$ edges where n is the number of nodes on a path from the start node to the center node. We have selected a total of 153 random systolic array task graphs with number of nodes as: 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, and 400. For each one of these (number of nodes) we have selected three task graphs each for the value of CCR as 1.00, 10.00, and 0.10 respectively.

4.4. Gaussian elimination task graphs

Fig. 2 shows a sample Gaussian elimination task graph [33]. A Gaussian elimination task graph has $(n^2+n+4)/2$ nodes and n^2+1 edges where n is the number of nodes on the second level of the task graph. We have selected a total of 144 random Gaussian elimination task graphs with number of nodes as: 93, 107, 122, 138, 155, 173, 192, 212, 233, 255, 278, 302, 327, 353, 380, and 408. For each one of these (number of nodes) we have selected three task graphs each for the value of CCR as 1.00, 10.00, and 0.10 respectively.

4.5. Divide and conquer task graphs

Fig. 3 shows a sample divide and conquer task graph [34]. A divide and conquer task graph has $3(2^{n-1})-2$ nodes and $2^{n+1}-4$ edges where n is the number of nodes on a path from the start node to the middle level of the task graph. We have selected a total of 180 random divide and conquer task graphs with number of nodes as: 10, 22, 46, 94, 190, and 382. For each one of these (number of nodes) we have selected ten task graphs each for the value of CCR as 1.00, 10.00, and 0.10 respectively.

4.6. Fast Fourier transform task graphs

Fig. 4 shows a sample fast Fourier transform task graph [35]. A fast Fourier transform task graph has $2+(n+1)2^n$ nodes and $(n+1)2^{n+1}$ edges where 2^n is the number of nodes on the second level of the task graph. We have selected a total of 150 random fast Fourier transform task graphs with number of nodes as: 6, 14, 34, 82, and 194. For each one of these (number of nodes) we have selected ten task graphs each for the value of CCR as 1.00, 10.00, and 0.10 respectively.

4.7. Small random task graphs with optimal solutions

We have selected a total of 180 small random task graphs with optimal solutions having number of nodes as: 5, 6, 7, 8, 9, and 10. For each one of these (number of nodes) we have selected ten task graphs each for the value of CCR as 1.00, 10.00, and 0.10 respectively.

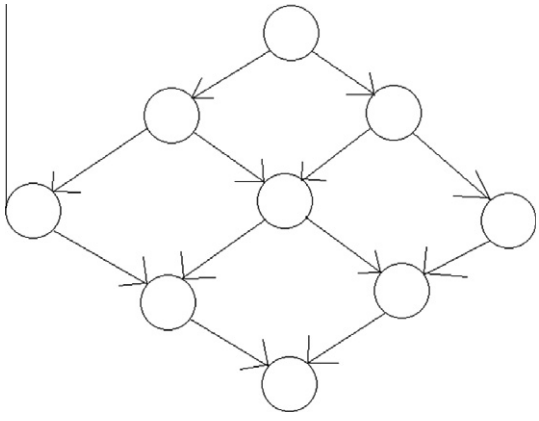

Fig. 1. A sample systolic array task graph for $n = 3$.

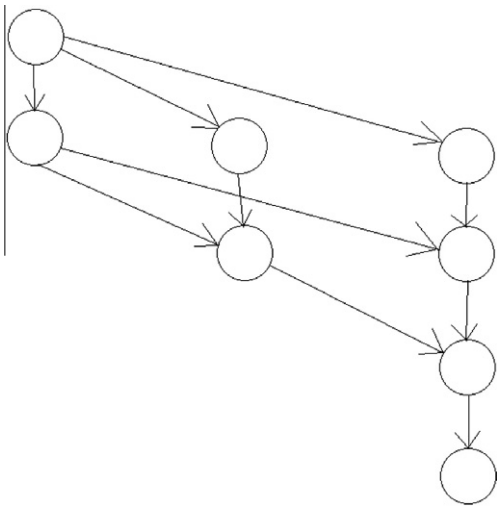

Fig. 2. A sample Gaussian elimination task graph for $n = 3$.

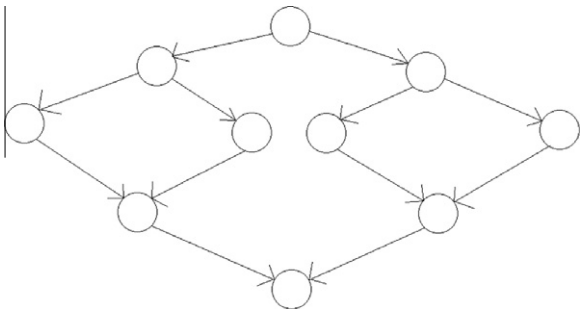

Fig. 3. A sample divide and conquer task graph for $n = 3$.

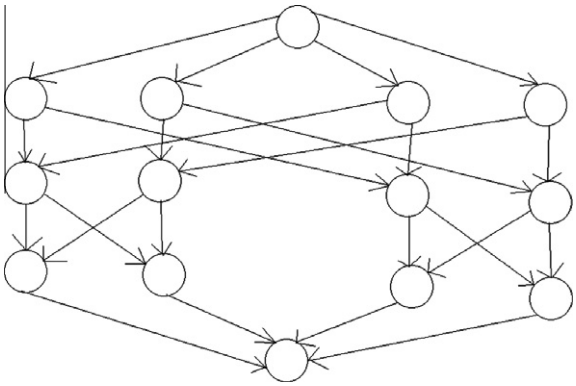

Fig. 4. A sample fast Fourier transform task graph for $n = 2$.

5. Performance results for peer set task graphs

Table 1 shows the schedule lengths obtained by running the six algorithms (CPPS, DCCL, DSC, EZ, LC, RDCC) on 9 selected peer set task graphs. As can be seen from the table, CPPS gives 4 best solutions (TG4, TG7, TG8, and TG9). DCCL gives 1 best solution (TG6). DSC gives 3 best solutions (TG3, TG7, and TG8). EZ gives 3 best solutions (TG2, TG7, and TG9). LC gives 2 best solutions (TG3, and TG8). RDCC gives 7 best solutions (TG1, TG3, TG4, TG5, TG6, TG7, and TG9). With the number of best solutions as the comparison parameter we can put them in the order: $DCCL < LC < DSC = EZ < CPPS < RDCC$. With the

Table 2
Correlation (ρ_{XY}) between the algorithms.

X/Y	DCCL	DSC	EZ	LC	RDCC
CPPS	0.990293	0.998357	0.999602	0.998406	0.999878
DCCL	–	0.996355	0.986802	0.995953	0.989693
DSC	–	–	0.996783	0.999933	0.998119
EZ	–	–	–	0.996859	0.999746
LC	–	–	–	–	0.998089

average of SL as the comparison parameter we can put them in the order: $RDCC < CPPS < DSC < LC < EZ < DCCL$. With the standard deviation of SL as the comparison parameter we can put them in the order: $RDCC < LC < DSC < CPPS < EZ < DCCL$.

Table 2 shows the correlation (ρ_{XY}) between the algorithms. From the table we observe that the algorithms DSC and LC are highly correlated. This is as expected because both the algorithms are designed using the concept of critical path [15, 14].

6. Performance results for random task graphs

Fig. 5 shows the average NSL vs number of nodes for random task graphs. Average NSL of CPPS ranges from 25.56 to 329.21 with an average of 155.67. Average NSL of DCCL ranges from 29.23 to 391.04 with an average of 183.06. Average NSL of DSC ranges from 25.74 to 330.56 with an average of 156.67. Average NSL of EZ ranges from 30.39 to 404.67 with an average of 189.63. Average NSL of LC ranges from 26.77 to 347.60 with an average of 163.83. Average NSL of RDCC ranges from 26.46 to 363.92 with an average of 169.85. The average performance order is: $CPPS < DSC < LC < RDCC < DCCL < EZ$.

Fig. 6 shows the average percentage degradation from the best solution vs number of nodes for random task graphs. Average percentage degradation from the best solution for CPPS ranges from 0.47 to 1.08 with an average of 0.69. Average percentage degradation from the best solution for DCCL ranges from 17.83 to 21.75 with an average of 19.96. Average percentage degradation from the best solution for DSC ranges from 0.80 to 1.76 with an average of 1.22. Average percentage degradation from the best solution for EZ ranges from 19.82 to 24.85 with an average of 22.66. Average percentage degradation from the best solution for LC ranges from 5.56 to 6.15 with an average of 5.86. Average percentage degradation from the best solution for RDCC ranges from 5.21 to 11.58 with an average of 9.42. The average performance order is: $CPPS < DSC < LC < RDCC < DCCL < EZ$.

Fig. 7 shows the average number of processors used vs number of nodes for random task graphs. Average number of processors used by CPPS ranges from 21.00 to 153.27 with an average of 81.05. Average number of processors used by DCCL ranges from 2.50 to 3.43 with an average of 3.04. Average number of processors used by DSC ranges from 15.20 to 107.73 with an average of 57.91. Average number of processors used by EZ ranges from 2.87 to 3.63 with an average of 3.31. Average number of processors used by LC ranges from 6.70 to 11.00 with an average of 9.24. Average number of processors used by RDCC ranges from 2.93 to 3.93 with an average of 3.44. The average performance order is: $DCCL < EZ < RDCC < LC < DSC < CPPS$.

Fig. 8 shows the average running time (in seconds) vs number of nodes for random task graphs. Average running time of CPPS ranges from 0.084 s to 2070.22 s with an average of 524.81 s. Average running time of DCCL ranges from 0.014 s to 8.21 s with an average of 2.52 s. Average running time of DSC ranges from 0.00062 s to 0.039 s with an average of 0.013 s. Average running time of EZ ranges from 0.0086 s to 16.47 s with an average of 4.74 s. Average running time of LC ranges from 0.000092 s to 0.0025 s with an average of 0.0012 s. Average running time of RDCC ranges from 1.90 s to 1550.97 s with an average of 470.45 s. The average performance order is: $LC < DSC < DCCL < EZ < RDCC < CPPS$.

7. Performance results for systolic array task graphs

Fig. 9 shows the average NSL vs number of nodes for systolic array task graphs. Average NSL of CPPS ranges from 7.30 to 191.87 with an average of 69.39. Average NSL of DCCL ranges from 8.23 to 229.22 with an average of 98.54. Average NSL of DSC ranges from 7.07 to 158.64 with an average of 58.85. Average NSL of EZ ranges from 7.23 to 159.24 with an average of 59.08. Average NSL of LC ranges from 13.88 to 219.49 with an average of 81.58. Average NSL of RDCC ranges from 7.17 to 183.02 with an average of 74.18. The average performance order is: $DSC < EZ < CPPS < RDCC < LC < DCCL$.

Fig. 10 shows the average NSL vs CCR for systolic array task graphs. For $CCR = 0.10$, average NSL of CPPS is 14.952, average NSL of DCCL is 49.78, average NSL of DSC is 14.65, average NSL of EZ is 14.954, average NSL of LC is 14.72, and average NSL of RDCC is 32.38. The average performance order is: $DSC < LC < CPPS < EZ < RDCC < DCCL$.

For $CCR = 1.00$, average NSL of CPPS is 52.08, average NSL of DCCL is 114.98, average NSL of DSC is 45.81, average NSL of EZ is 49.62, average NSL of LC is 48.35, and average NSL of RDCC is 74.03. The average performance order is: $DSC < LC < EZ < CPPS < RDCC < DCCL$.

For $CCR = 10.00$, average NSL of CPPS is 141.14, average NSL of DCCL is 130.85, average NSL of DSC is 116.10, average NSL of EZ is 112.65, average NSL of LC is 181.67, and average NSL of RDCC is 116.12. The average performance order is: $EZ < DSC < RDCC < DCCL < CPPS < LC$.

Fig. 11 shows the average percentage degradation from the best solution vs number of nodes for systolic array task graphs. Average percentage degradation from the best solution for CPPS ranges from 0.82 to 30.40 with an average of

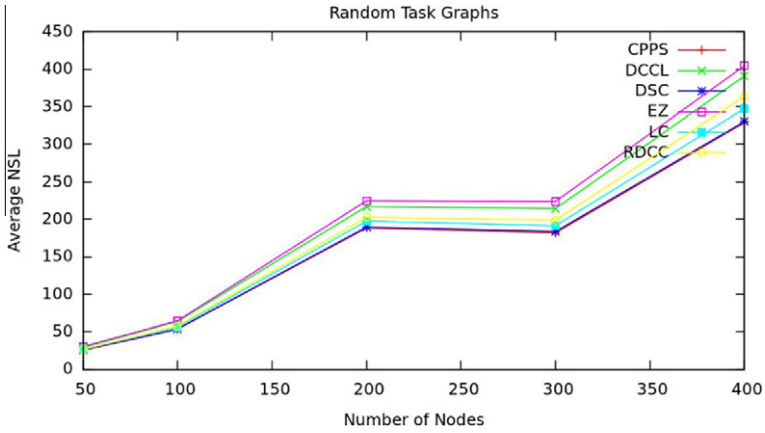

Fig. 5. Average NSL vs number of nodes for random task graphs. The average performance order is: *CPPS* < *DSC* < *LC* < *RDCC* < *DCCL* < *EZ*.

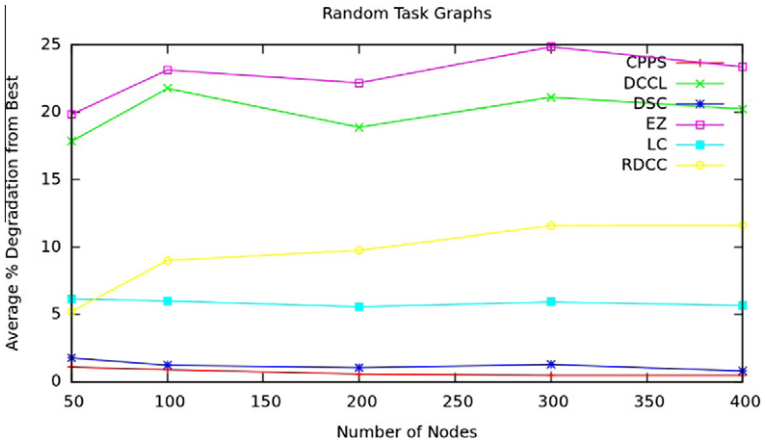

Fig. 6. Average percentage degradation from the best solution vs number of nodes for random task graphs. The average performance order is: *CPPS* < *DSC* < *LC* < *RDCC* < *DCCL* < *EZ*.

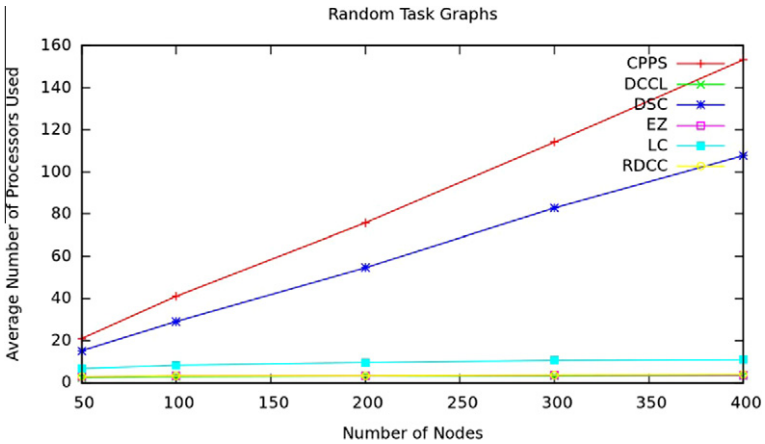

Fig. 7. Average number of processors used vs number of nodes for random task graphs. The average performance order is: *DCCL* < *EZ* < *RDCC* < *LC* < *DSC* < *CPPS*.

12.81. Average percentage degradation from the best solution for DCCL ranges from 17.56 to 210.00 with an average of 106.74. Average percentage degradation from the best solution for DSC ranges from 0.10 to 14.99 with an average of 5.04. Average percentage degradation from the best solution for EZ ranges from 2.75 to 9.49 with an average of 5.54. Average

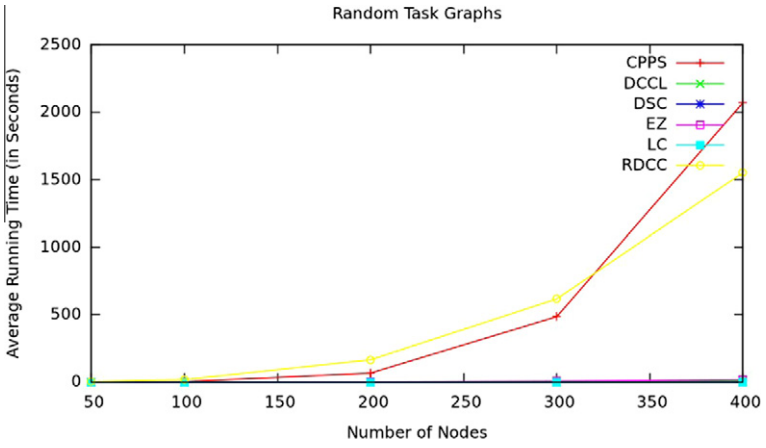

Fig. 8. Average running time (in seconds) vs number of nodes for random task graphs. The average performance order is: *LC* < *DSC* < *DCCL* < *EZ* < *RDCC* < *CPPS*.

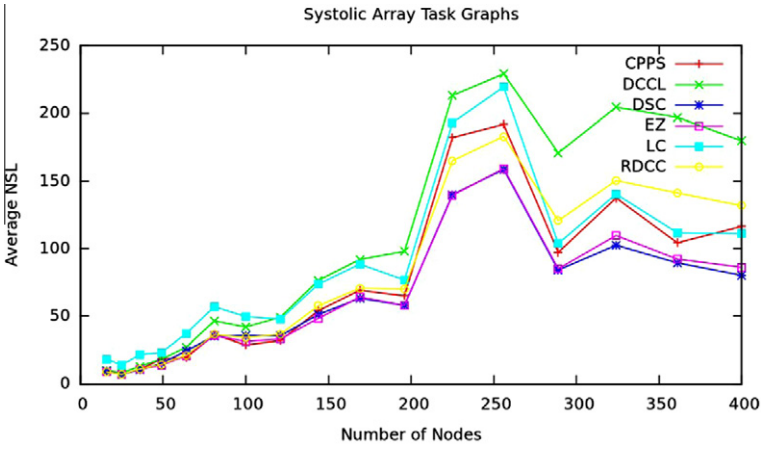

Fig. 9. Average NSL vs number of nodes for systolic array task graphs. The average performance order is: *DSC* < *EZ* < *CPPS* < *RDCC* < *LC* < *DCCL*.

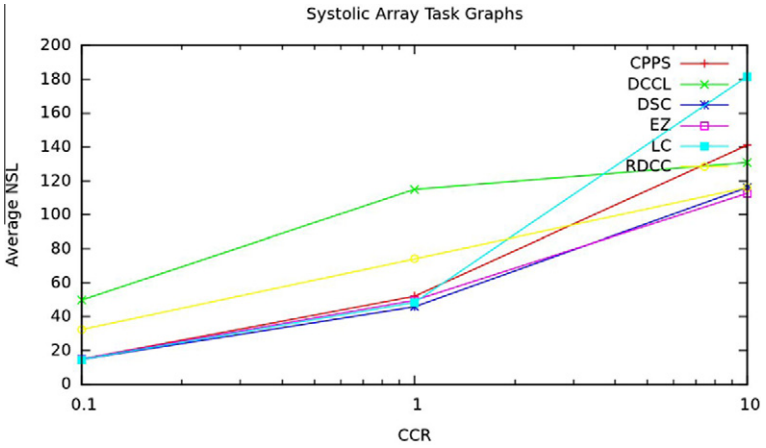

Fig. 10. Average NSL vs CCR for systolic array task graphs. The average performance order for *CCR* = 0.10 is: *DSC* < *LC* < *CPPS* < *EZ* < *RDCC* < *DCCL*. The average performance order for *CCR* = 1.00 is: *DSC* < *LC* < *EZ* < *CPPS* < *RDCC* < *DCCL*. The average performance order for *CCR* = 10.0 is: *EZ* < *DSC* < *RDCC* < *DCCL* < *CPPS* < *LC*.

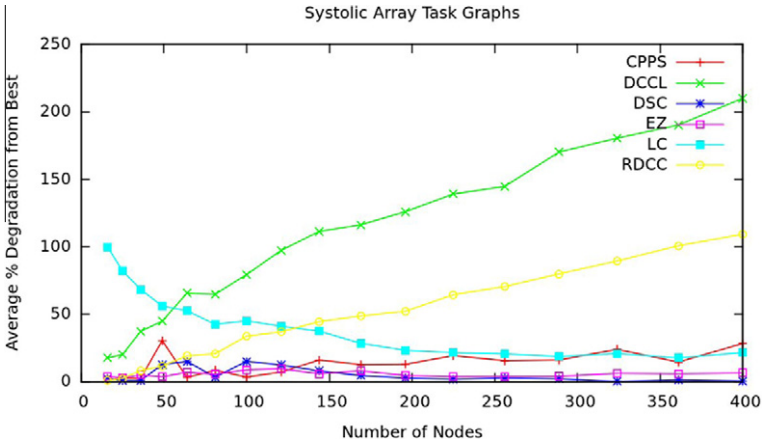

Fig. 11. Average percentage degradation from the best solution vs number of nodes for systolic array task graphs. The average performance order is: *DSC* < *EZ* < *CPPS* < *LC* < *RDCC* < *DCCL*.

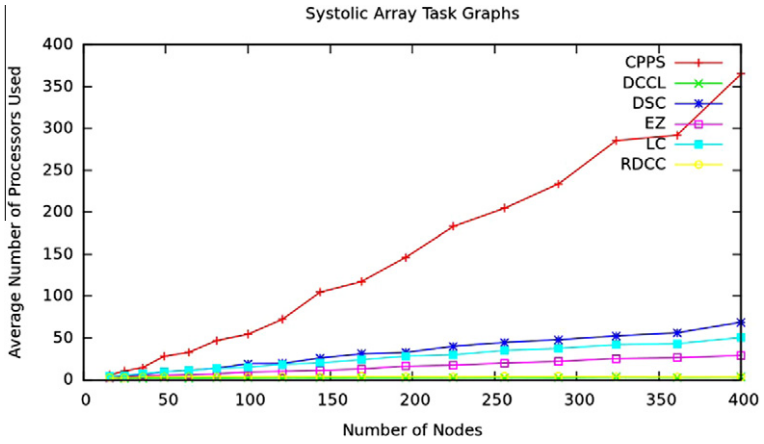

Fig. 12. Average number of processors used vs number of nodes for systolic array task graphs. The average performance order is: *DCCL* < *RDCC* < *EZ* < *LC* < *DSC* < *CPPS*.

percentage degradation from the best solution for LC ranges from 17.58 to 99.76 with an average of 40.98. Average percentage degradation from the best solution for RDCC ranges from 0.65 to 109.24 with an average of 46.61. The average performance order is: *DSC* < *EZ* < *CPPS* < *LC* < *RDCC* < *DCCL*.

Fig. 12 shows the average number of processors used vs number of nodes for systolic array task graphs. Average number of processors used by CPPS ranges from 5.78 to 365.11 with an average of 129.39. Average number of processors used by DCCL ranges from 2.11 to 3.22 with an average of 2.67. Average number of processors used by DSC ranges from 3.67 to 69.00 with an average of 28.92. Average number of processors used by EZ ranges from 2.67 to 29.22 with an average of 13.69. Average number of processors used by LC ranges from 4.11 to 50.78 with an average of 23.42. Average number of processors used by RDCC ranges from 2.67 to 4.11 with an average of 3.69. The average performance order is: *DCCL* < *RDCC* < *EZ* < *LC* < *DSC* < *CPPS*.

Fig. 13 shows the average running time (in seconds) vs number of nodes for systolic array task graphs. Average running time of CPPS ranges from 0.00032 s to 0.46 s with an average of 0.13 s. Average running time of DCCL ranges from 0.00026 s to 0.36 s with an average of 0.080 s. Average running time of DSC ranges from 0.000036 s to 0.0032 s with an average of 0.0010 s. Average running time of EZ ranges from 0.00010 s to 0.047 s with an average of 0.013 s. Average running time of LC ranges from 0.000016 s to 0.0014 s with an average of 0.00045 s. Average running time of RDCC ranges from 0.026 s to 50.24 s with an average of 11.55 s. The average performance order is: *LC* < *DSC* < *EZ* < *DCCL* < *CPPS* < *RDCC*.

8. Performance results for Gaussian elimination task graphs

Fig. 14 shows the average NSL vs number of nodes for Gaussian elimination task graphs. Average NSL of CPPS ranges from 19.82 to 61.27 with an average of 39.29. Average NSL of DCCL ranges from 34.31 to 146.80 with an average of 80.08. Average

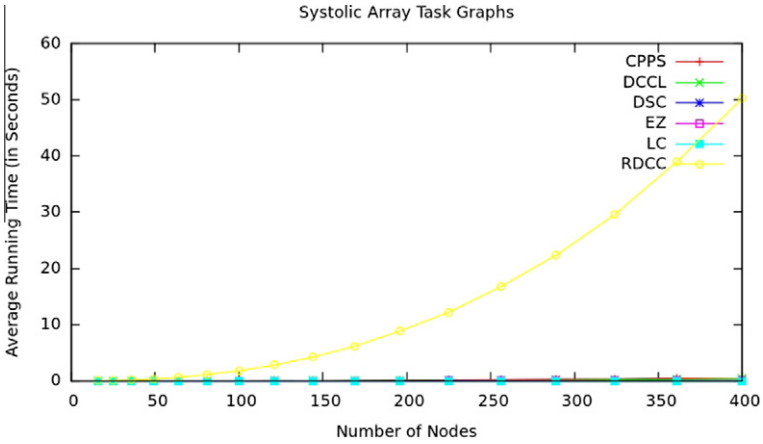

Fig. 13. Average running time (in seconds) vs number of nodes for systolic array task graphs. The average performance order is: $LC < DSC < EZ < DCCL < CPPS < RDCC$.

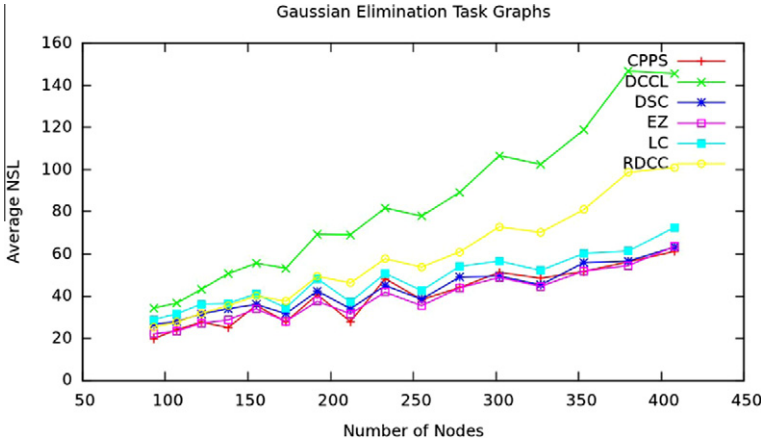

Fig. 14. Average NSL vs number of nodes for Gaussian elimination task graphs. The average performance order is: $EZ < CPPS < DSC < LC < RDCC < DCCL$.

NSL of DSC ranges from 26.67 to 63.36 with an average of 41.75. Average NSL of EZ ranges from 22.09 to 63.58 with an average of 38.60. Average NSL of LC ranges from 28.86 to 72.51 with an average of 46.56. Average NSL of RDCC ranges from 25.39 to 100.96 with an average of 55.62. The average performance order is: $EZ < CPPS < DSC < LC < RDCC < DCCL$.

Fig. 15 shows the average NSL vs CCR for Gaussian elimination task graphs. For $CCR = 0.10$, average NSL of CPPS is 12.84, average NSL of DCCL is 63.88, average NSL of DSC is 12.64, average NSL of EZ is 12.93, average NSL of LC is 12.71, and average NSL of RDCC is 40.25. The average performance order is: $DSC < LC < CPPS < EZ < RDCC < DCCL$.

For $CCR = 1.00$, average NSL of CPPS is 21.98, average NSL of DCCL is 74.15, average NSL of DSC is 19.19, average NSL of EZ is 22.37, average NSL of LC is 20.03, and average NSL of RDCC is 45.58. The average performance order is: $DSC < LC < CPPS < EZ < RDCC < DCCL$.

For $CCR = 10.00$, average NSL of CPPS is 83.06, average NSL of DCCL is 102.22, average NSL of DSC is 93.41, average NSL of EZ is 80.51, average NSL of LC is 106.94, and average NSL of RDCC is 81.03. The average performance order is: $EZ < RDCC < CPPS < DSC < DCCL < LC$.

Fig. 16 shows the average percentage degradation from the best solution vs number of nodes for Gaussian elimination task graphs. Average percentage degradation from the best solution for CPPS ranges from 3.68 to 12.73 with an average of 8.32. Average percentage degradation from the best solution for DCCL ranges from 121.05 to 331.84 with an average of 227.50. Average percentage degradation from the best solution for DSC ranges from 2.62 to 20.90 with an average of 9.91. Average percentage degradation from the best solution for EZ ranges from 5.66 to 12.63 with an average of 9.10. Average percentage degradation from the best solution for LC ranges from 11.29 to 30.15 with an average of 17.89. Average percentage degradation from the best solution for RDCC ranges from 49.07 to 183.05 with an average of 111.51. The average performance order is: $CPPS < EZ < DSC < LC < RDCC < DCCL$.

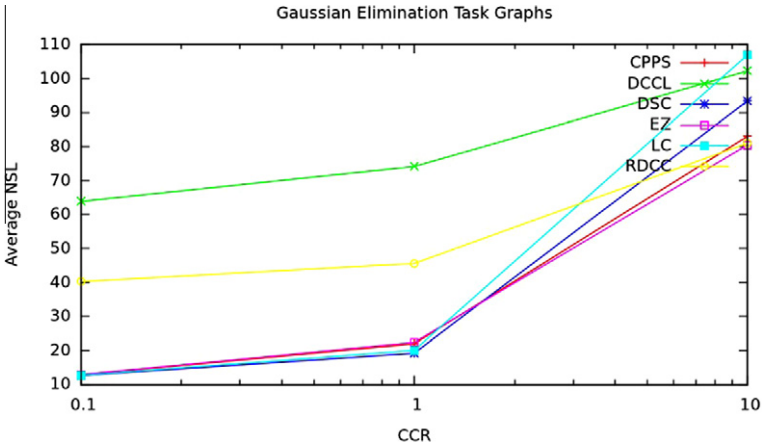

Fig. 15. Average NSL vs CCR for Gaussian elimination task graphs. The average performance order for $CCR = 0.10$ is: $DSC < LC < CPPS < EZ < RDCC < DCCL$. The average performance order for $CCR = 1.00$ is: $DSC < LC < CPPS < EZ < RDCC < DCCL$. The average performance order for $CCR = 10.0$ is: $EZ < RDCC < CPPS < DSC < DCCL < LC$.

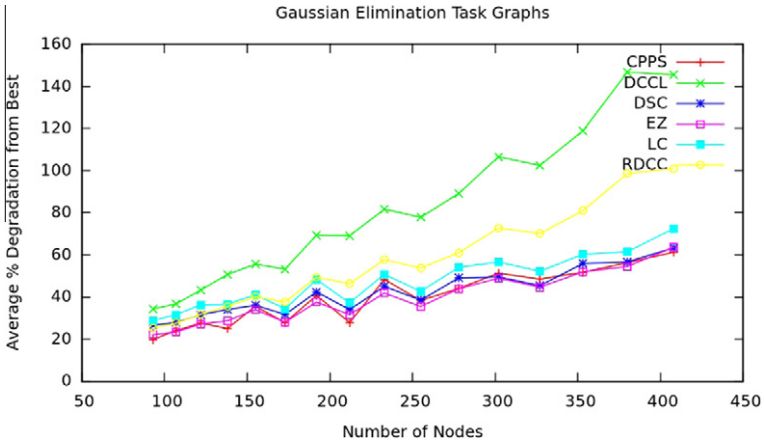

Fig. 16. Average percentage degradation from the best solution vs number of nodes for Gaussian elimination task graphs. The average performance order is: $CPPS < EZ < DSC < LC < RDCC < DCCL$.

Fig. 17 shows the average number of processors used vs number of nodes for Gaussian elimination task graphs. Average number of processors used by CPPS ranges from 59.67 to 355.33 with an average of 193.26. Average number of processors used by DCCL ranges from 2.78 to 3.33 with an average of 2.97. Average number of processors used by DSC ranges from 27.44 to 142.11 with an average of 80.81. Average number of processors used by EZ ranges from 10.11 to 41.78 with an average of 22.77. Average number of processors used by LC ranges from 13.00 to 28.00 with an average of 20.50. Average number of processors used by RDCC ranges from 3.67 to 4.33 with an average of 4.01. The average performance order is: $DCCL < RDCC < LC < EZ < DSC < CPPS$.

Fig. 18 shows the average running time (in seconds) vs number of nodes for Gaussian elimination task graphs. Average running time of CPPS ranges from 0.017 s to 0.49 s with an average of 0.17 s. Average running time of DCCL ranges from 0.011 s to 0.40 s with an average of 0.13 s. Average running time of DSC ranges from 0.00029 s to 0.0036 s with an average of 0.0015 s. Average running time of EZ ranges from 0.0040 s to 0.071 s with an average of 0.026 s. Average running time of LC ranges from 0.00015 s to 0.0011 s with an average of 0.00053 s. Average running time of RDCC ranges from 1.64 s to 55.98 s with an average of 18.63 s. The average performance order is: $LC < DSC < EZ < DCCL < CPPS < RDCC$.

9. Performance results for divide and conquer task graphs

Fig. 19 shows the average NSL vs number of nodes for divide and conquer task graphs. Average NSL of CPPS ranges from 3.22 to 25.19 with an average of 13.67. Average NSL of DCCL ranges from 2.73 to 94.73 with an average of 31.72. Average NSL of DSC ranges from 2.50 to 16.77 with an average of 10.29. Average NSL of EZ ranges from 2.52 to 18.37 with an average of

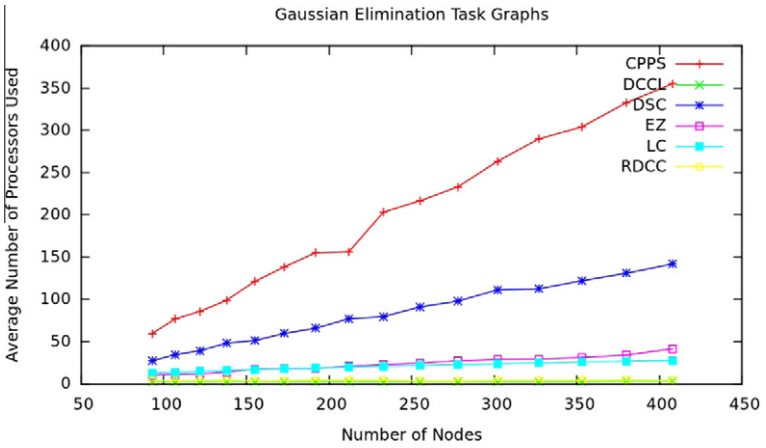

Fig. 17. Average number of processors used vs number of nodes for Gaussian elimination task graphs. The average performance order is: *DCCL* < *RDCC* < *LC* < *EZ* < *DSC* < *CPPS*.

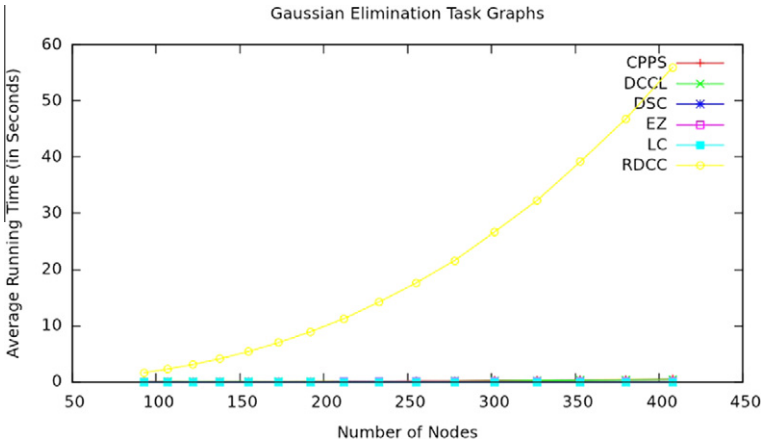

Fig. 18. Average running time (in seconds) vs number of nodes for Gaussian elimination task graphs. The average performance order is: *LC* < *DSC* < *EZ* < *DCCL* < *CPPS* < *RDCC*.

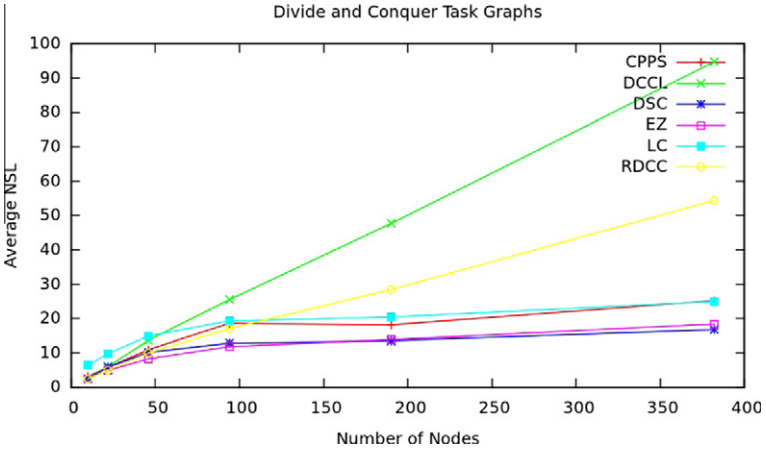

Fig. 19. Average NSL vs number of nodes for divide and conquer task graphs. The average performance order is: *EZ* < *DSC* < *CPPS* < *LC* < *RDCC* < *DCCL*.

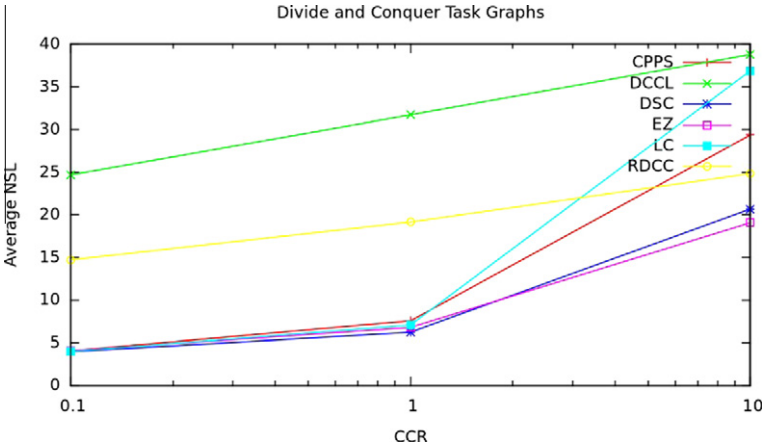

Fig. 20. Average NSL vs CCR for divide and conquer task graphs. The average performance order for $CCR = 0.10$ is: $DSC < LC < EZ < CPPS < RDCC < DCCL$. The average performance order for $CCR = 1.00$ is: $DSC < EZ < LC < CPPS < RDCC < DCCL$. The average performance order for $CCR = 10.0$ is: $EZ < DSC < RDCC < CPPS < LC < DCCL$.

9.98. Average NSL of LC ranges from 6.45 to 24.93 with an average of 15.98. Average NSL of RDCC ranges from 2.50 to 54.38 with an average of 19.57. The average performance order is: $EZ < DSC < CPPS < LC < RDCC < DCCL$.

Fig. 20 shows the average NSL vs CCR for divide and conquer task graphs. For $CCR = 0.10$, average NSL of CPPS is 4.0739, average NSL of DCCL is 24.68, average NSL of DSC is 3.96, average NSL of EZ is 4.0737, average NSL of LC is 3.99, and average NSL of RDCC is 14.72. The average performance order is: $DSC < LC < EZ < CPPS < RDCC < DCCL$.

For $CCR = 1.00$, average NSL of CPPS is 7.56, average NSL of DCCL is 31.71, average NSL of DSC is 6.23, average NSL of EZ is 6.79, average NSL of LC is 7.07, and average NSL of RDCC is 19.15. The average performance order is: $DSC < EZ < LC < CPPS < RDCC < DCCL$.

For $CCR = 10.00$, average NSL of CPPS is 29.38, average NSL of DCCL is 38.77, average NSL of DSC is 20.68, average NSL of EZ is 19.08, average NSL of LC is 36.88, and average NSL of RDCC is 24.84. The average performance order is: $EZ < DSC < RDCC < CPPS < LC < DCCL$.

Fig. 21 shows the average percentage degradation from the best solution vs number of nodes for divide and conquer task graphs. Average percentage degradation from the best solution for CPPS ranges from 13.76 to 35.54 with an average of 25.45. Average percentage degradation from the best solution for DCCL ranges from 12.79 to 746.26 with an average of 251.41. Average percentage degradation from the best solution for DSC ranges from 0.00 to 15.02 with an average of 6.27. Average percentage degradation from the best solution for EZ ranges from 2.24 to 9.37 with an average of 4.60. Average percentage degradation from the best solution for LC ranges from 27.37 to 118.43 with an average of 55.29. Average percentage degradation from the best solution for RDCC ranges from 1.27 to 395.69 with an average of 120.90. The average performance order is: $EZ < DSC < CPPS < LC < RDCC < DCCL$.

Fig. 22 shows the average number of processors used vs number of nodes for divide and conquer task graphs. Average number of processors used by CPPS ranges from 3.93 to 370.17 with an average of 113.07. Average number of processors used by DCCL ranges from 1.97 to 2.57 with an average of 2.27. Average number of processors used by DSC ranges from 2.90 to 96.10 with an average of 32.01. Average number of processors used by EZ ranges from 2.50 to 48.70 with an average of 17.64. Average number of processors used by LC ranges from 4.00 to 128.00 with an average of 42.00. Average number of processors used by RDCC ranges from 2.57 to 4.10 with an average of 3.47. The average performance order is: $DCCL < RDCC < EZ < DSC < LC < CPPS$.

Fig. 23 shows the average running time (in seconds) vs number of nodes for divide and conquer task graphs. Average running time of CPPS ranges from 0.000061 s to 0.11 s with an average of 0.024 s. Average running time of DCCL ranges from 0.000057 s to 0.29 s with an average of 0.058 s. Average running time of DSC ranges from 0.000022 s to 0.0028 s with an average of 0.00069 s. Average running time of EZ ranges from 0.000036 s to 0.031 s with an average of 0.0075 s. Average running time of LC ranges from 0.000009 s to 0.0028 s with an average of 0.00063 s. Average running time of RDCC ranges from 0.010 s to 40.35 s with an average of 8.21 s. The average performance order is: $LC < DSC < EZ < CPPS < DCCL < RDCC$.

10. Performance results for fast Fourier transform task graphs

Fig. 24 shows the average NSL vs number of nodes for fast Fourier transform task graphs. Average NSL of CPPS ranges from 3.08 to 8.40 with an average of 6.51. Average NSL of DCCL ranges from 3.08 to 26.11 with an average of 11.08. Average NSL of DSC ranges from 3.10 to 7.48 with an average of 5.60. Average NSL of EZ ranges from 3.08 to 7.42 with an average of 5.48. Average NSL of LC ranges from 6.49 to 8.47 with an average of 7.62. Average NSL of RDCC ranges from 3.04 to 15.02 with an average of 7.50. The average performance order is: $EZ < DSC < CPPS < RDCC < LC < DCCL$.

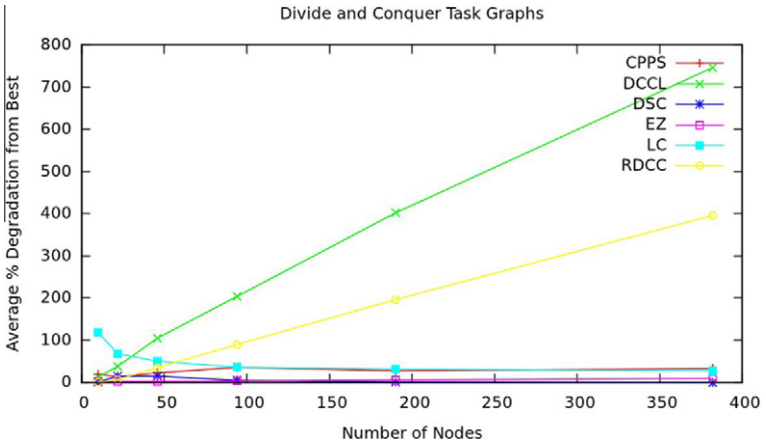

Fig. 21. Average percentage degradation from the best solution vs number of nodes for divide and conquer task graphs. The average performance order is: *EZ* < *DSC* < *CPPS* < *LC* < *RDCC* < *DCCL*.

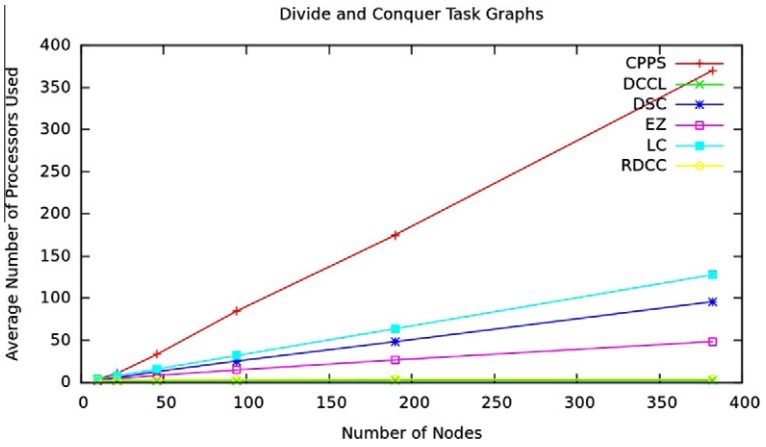

Fig. 22. Average number of processors used vs number of nodes for divide and conquer task graphs. The average performance order is: *DCCL* < *RDCC* < *EZ* < *DSC* < *LC* < *CPPS*.

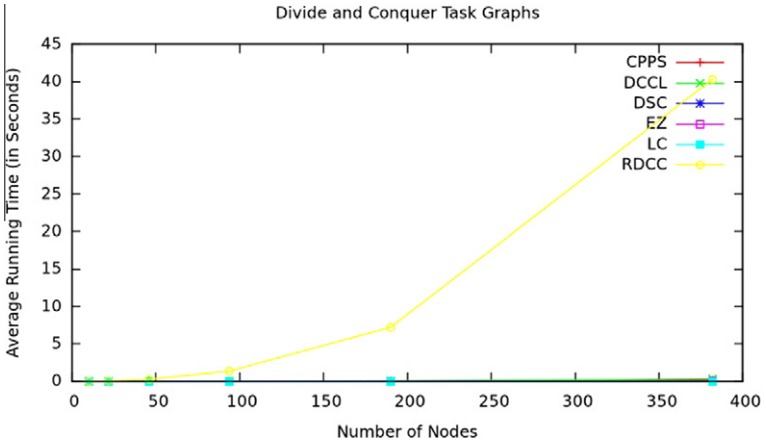

Fig. 23. Average running time (in seconds) vs number of nodes for divide and conquer task graphs. The average performance order is: *LC* < *DSC* < *EZ* < *CPPS* < *DCCL* < *RDCC*.

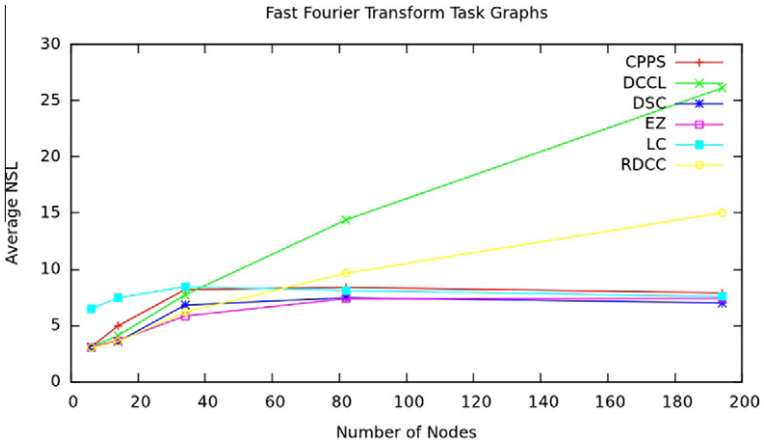

Fig. 24. Average NSL vs number of nodes for fast Fourier transform task graphs. The average performance order is: $EZ < DSC < CPPS < RDCC < LC < DCCL$.

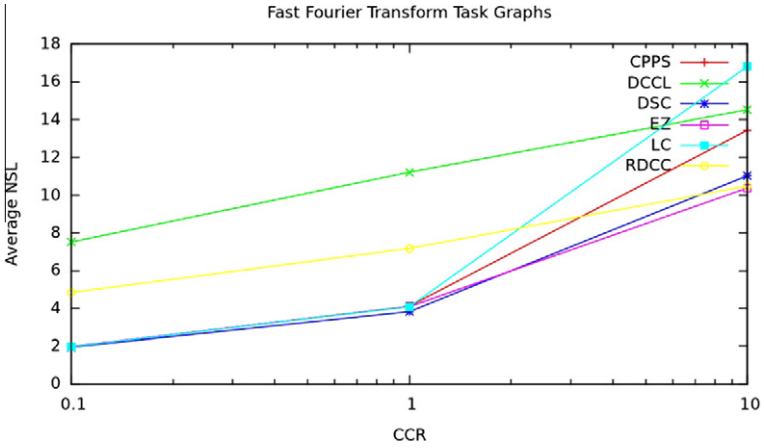

Fig. 25. Average NSL vs CCR for fast Fourier transform task graphs. The average performance order for $CCR = 0.10$ is: $DSC < LC < CPPS < EZ < RDCC < DCCL$. The average performance order for $CCR = 1.00$ is: $DSC < LC < EZ < CPPS < RDCC < DCCL$. The average performance order for $CCR = 10.0$ is: $EZ < RDCC < DSC < CPPS < DCCL < LC$.

Fig. 25 shows the average NSL vs CCR for fast Fourier transform task graphs. For $CCR = 0.10$, average NSL of CPPS is 1.969, average NSL of DCCL is 7.51, average NSL of DSC is 1.94, average NSL of EZ is 1.99, average NSL of LC is 1.965, and average NSL of RDCC is 4.83. The average performance order is: $DSC < LC < CPPS < EZ < RDCC < DCCL$.

For $CCR = 1.00$, average NSL of CPPS is 4.11, average NSL of DCCL is 11.22, average NSL of DSC is 3.82, average NSL of EZ is 4.085, average NSL of LC is 4.081, and average NSL of RDCC is 7.18. The average performance order is: $DSC < LC < EZ < CPPS < RDCC < DCCL$.

For $CCR = 10.00$, average NSL of CPPS is 13.45, average NSL of DCCL is 14.52, average NSL of DSC is 11.03, average NSL of EZ is 10.38, average NSL of LC is 16.82, and average NSL of RDCC is 10.49. The average performance order is: $EZ < RDCC < DSC < CPPS < DCCL < LC$.

Fig. 26 shows the average percentage degradation from the best solution vs number of nodes for fast Fourier transform task graphs. Average percentage degradation from the best solution for CPPS ranges from 1.14 to 29.10 with an average of 15.91. Average percentage degradation from the best solution for DCCL ranges from 1.70 to 545.64 with an average of 176.94. Average percentage degradation from the best solution for DSC ranges from 0.00 to 11.75 with an average of 3.42. Average percentage degradation from the best solution for EZ ranges from 1.62 to 6.43 with an average of 4.46. Average percentage degradation from the best solution for LC ranges from 5.89 to 105.09 with an average of 45.16. Average percentage degradation from the best solution for RDCC ranges from 0.03 to 287.56 with an average of 84.28. The average performance order is: $DSC < EZ < CPPS < LC < RDCC < DCCL$.

Fig. 27 shows the average number of processors used vs number of nodes for fast Fourier transform task graphs. Average number of processors used by CPPS ranges from 2.13 to 188.40 with an average of 60.25. Average number of processors used by DCCL ranges from 1.47 to 2.73 with an average of 2.30. Average number of processors used by DSC ranges from 2.07 to 79.97 with an average of 26.45. Average number of processors used by EZ ranges from 1.47 to 22.47 with an average of 8.50.

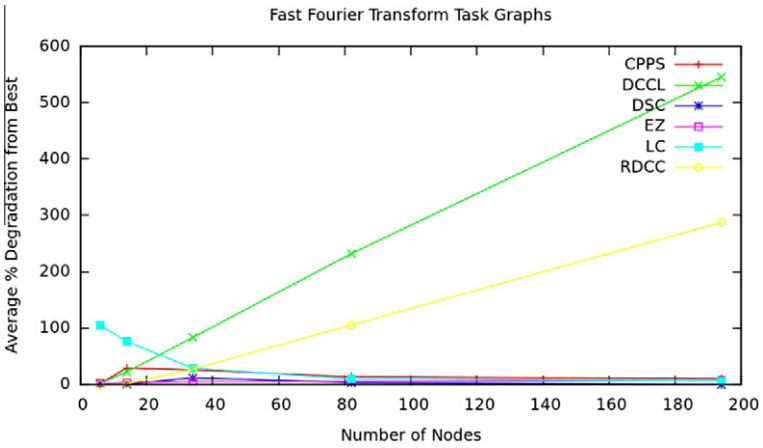

Fig. 26. Average percentage degradation from the best solution vs number of nodes for fast Fourier transform task graphs. The average performance order is: *DSC* < *EZ* < *CPPS* < *LC* < *RDCC* < *DCCL*.

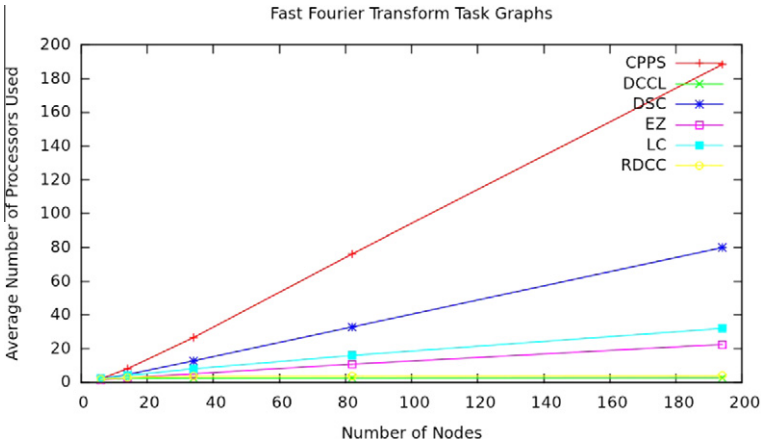

Fig. 27. Average number of processors used vs number of nodes for fast Fourier transform task graphs. The average performance order is: *DCCL* < *RDCC* < *EZ* < *LC* < *DSC* < *CPPS*.

Average number of processors used by LC ranges from 2.00 to 32.00 with an average of 12.40. Average number of processors used by RDCC ranges from 1.83 to 3.97 with an average of 3.21. The average performance order is: *DCCL* < *RDCC* < *EZ* < *LC* < *DSC* < *CPPS*.

Fig. 28 shows the average running time (in seconds) vs number of nodes for fast Fourier transform task graphs. Average running time of CPPS ranges from 0.000044 s to 0.033 s with an average of 0.0088 s. Average running time of DCCL ranges from 0.000029 s to 0.066 s with an average of 0.016 s. Average running time of DSC ranges from 0.000015 s to 0.0011 s with an average of 0.00032 s. Average running time of EZ ranges from 0.000019 s to 0.018 s with an average of 0.0046 s. Average running time of LC ranges from 0.0000060 s to 0.00098 s with an average of 0.00026 s. Average running time of RDCC ranges from 0.0040 s to 9.63 s with an average of 2.23 s. The average performance order is: *LC* < *DSC* < *EZ* < *CPPS* < *DCCL* < *RDCC*.

11. Performance results for small random task graphs with optimal solutions

Fig. 29 shows the average NSL vs number of nodes for small random task graphs with optimal solutions. Average NSL of CPPS ranges from 1.36 to 2.02 with an average of 1.65. Average NSL of DCCL ranges from 1.24 to 2.21 with an average of 1.67. Average NSL of DSC ranges from 1.24 to 2.00 with an average of 1.602. Average NSL of EZ ranges from 1.23 to 1.99 with an average of 1.598. Average NSL of LC ranges from 2.49 to 4.10 with an average of 3.12. Average NSL of RDCC ranges from 1.22 to 1.96 with an average of 1.58. Average NSL of OPT (the *Optimal algorithm*) ranges from 1.22 to 1.95 with an average of 1.57. The average performance order is: *OPT* < *RDCC* < *EZ* < *DSC* < *CPPS* < *DCCL* < *LC*.

Fig. 30 shows the average NSL vs CCR for small random task graphs with optimal solution. For $CCR = 0.10$, average NSL of CPPS is 1.248, average NSL of DCCL is 1.373, average NSL of DSC is 1.251, average NSL of EZ is 1.263, average NSL of LC is

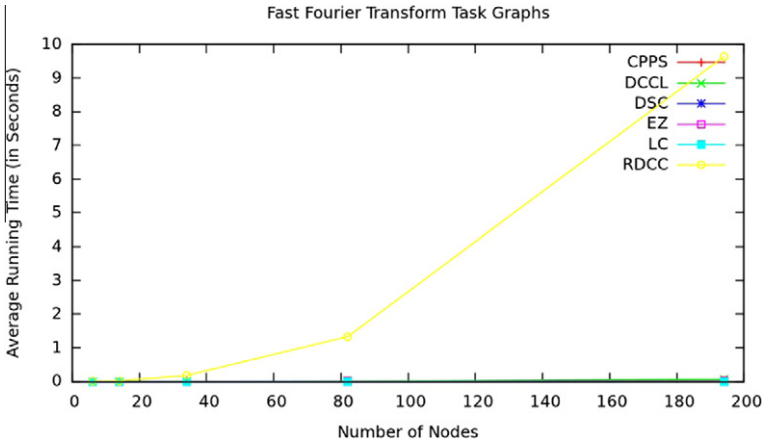

Fig. 28. Average running time (in seconds) vs number of nodes for fast Fourier transform task graphs. The average performance order is: $LC < DSC < EZ < CPPS < DCCL < RDCC$.

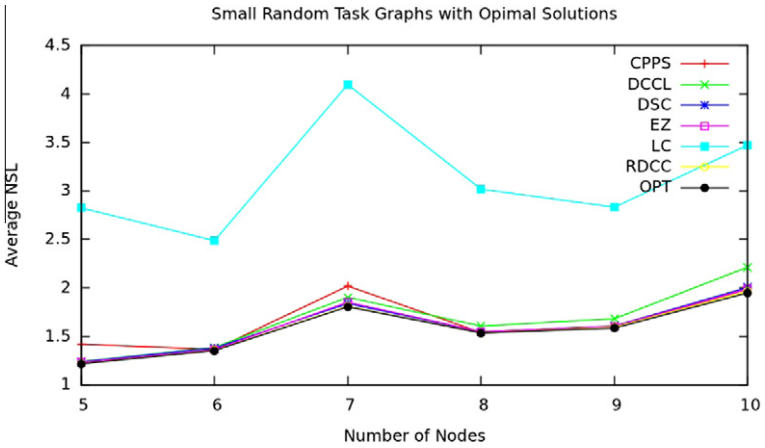

Fig. 29. Average NSL vs number of nodes for small random task graphs with optimal solutions. The average performance order is: $OPT < RDCC < EZ < DSC < CPPS < DCCL < LC$.

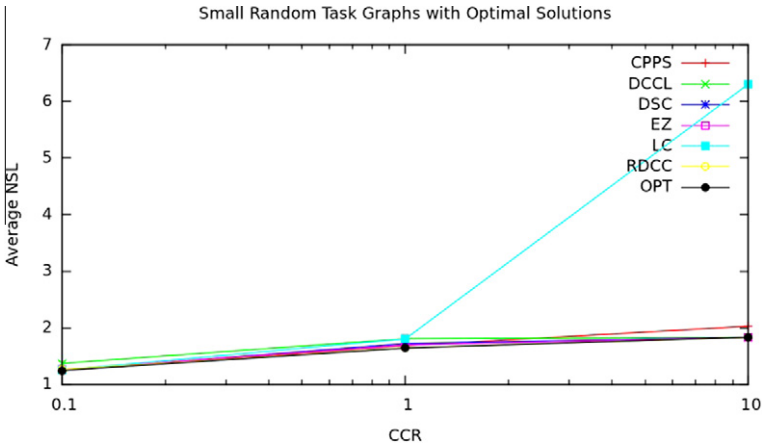

Fig. 30. Average NSL vs CCR for small random task graphs with optimal solutions. The average performance order for $CCR = 0.10$ is: $OPT < CPPS < DSC < LC < RDCC < EZ < DCCL$. The average performance order for $CCR = 1.00$ is: $OPT < RDCC < CPPS < EZ < DSC < LC < DCCL$. The average performance order for $CCR = 10.0$ is: $OPT = DCCL = DSC = EZ = RDCC < CPPS < LC$.

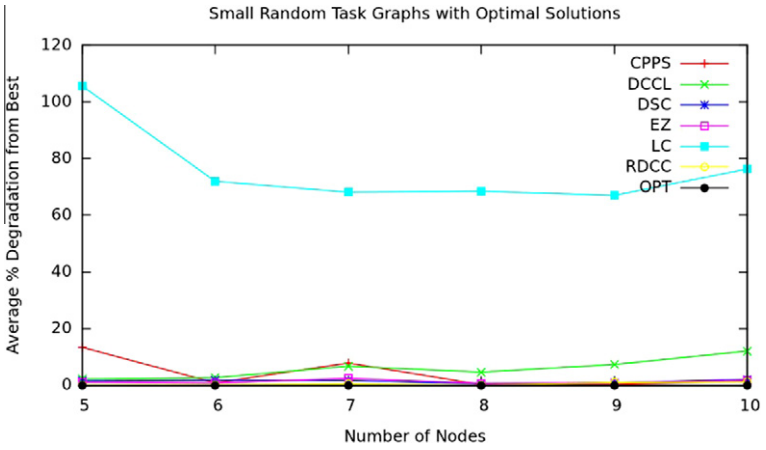

Fig. 31. Average percentage degradation from the best solution vs number of nodes for small random task graphs with optimal solutions. The average performance order is: *OPT* < *RDCC* < *EZ* < *DSC* < *CPPS* < *DCCL* < *LC*.

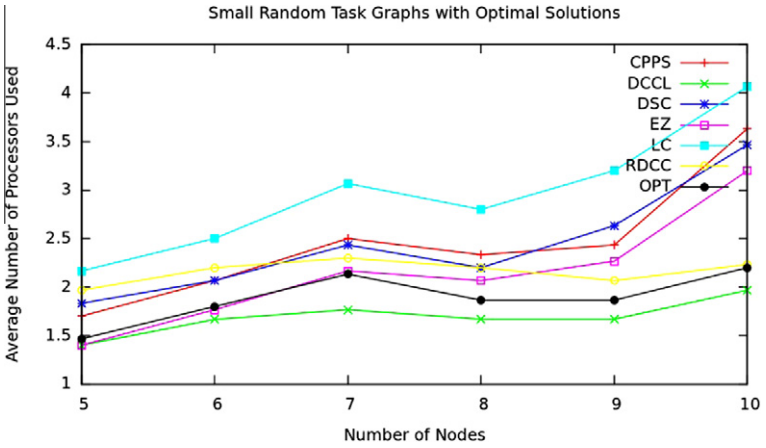

Fig. 32. Average number of processors used vs number of nodes for small random task graphs with optimal solutions. The average performance order is: *DCCL* < *OPT* < *EZ* < *RDCC* < *DSC* < *CPPS* < *LC*.

1.255, average NSL of RDCC is 1.261, and average NSL of OPT is 1.246. The average performance order is: *OPT* < *CPPS* < *DSC* < *LC* < *RDCC* < *EZ* < *DCCL*.

For *CCR* = 1.00, average NSL of CPPS is 1.684, average NSL of DCCL is 1.807, average NSL of DSC is 1.720, average NSL of EZ is 1.694, average NSL of LC is 1.803, average NSL of RDCC is 1.643, and average NSL of OPT is 1.641. The average performance order is: *OPT* < *RDCC* < *CPPS* < *EZ* < *DSC* < *LC* < *DCCL*.

For *CCR* = 10.00, average NSL of CPPS is 2.029, average NSL of DCCL is 1.835, average NSL of DSC is 1.835, average NSL of EZ is 1.835, average NSL of LC is 6.307494, average NSL of RDCC is 1.835, and average NSL of OPT is 1.835. The average performance order is: *OPT* = *DCCL* = *DSC* = *EZ* = *RDCC* < *CPPS* < *LC*.

Fig. 31 shows the average percentage degradation from the best solution vs number of nodes for small random task graphs with optimal solutions. Average percentage degradation from the best solution for CPPS ranges from 0.21 to 13.40 with an average of 4.03. Average percentage degradation from the best solution for DCCL ranges from 2.16 to 12.09 with an average of 5.93. Average percentage degradation from the best solution for DSC ranges from 0.56 to 2.11 with an average of 1.46. Average percentage degradation from the best solution for EZ ranges from 0.71 to 2.46 with an average of 1.30. Average percentage degradation from the best solution for LC ranges from 67.04 to 105.62 with an average of 76.27. Average percentage degradation from the best solution for RDCC ranges from 0.07 to 1.21 with an average of 0.52. Average percentage degradation from the best solution for OPT ranges from 0.00 to 0.00 with an average of 0.00. The average performance order is: *OPT* < *RDCC* < *EZ* < *DSC* < *CPPS* < *DCCL* < *LC*.

Fig. 32 shows the average number of processors used vs number of nodes for small random task graphs with optimal solutions. Average number of processors used by CPPS ranges from 1.70 to 3.63 with an average of 2.444. Average number of processors used by DCCL ranges from 1.40 to 1.97 with an average of 1.69. Average number of processors used by DSC ranges from 1.83 to 3.47 with an average of 2.439. Average number of processors used by EZ ranges from 1.40 to 3.20 with an average of

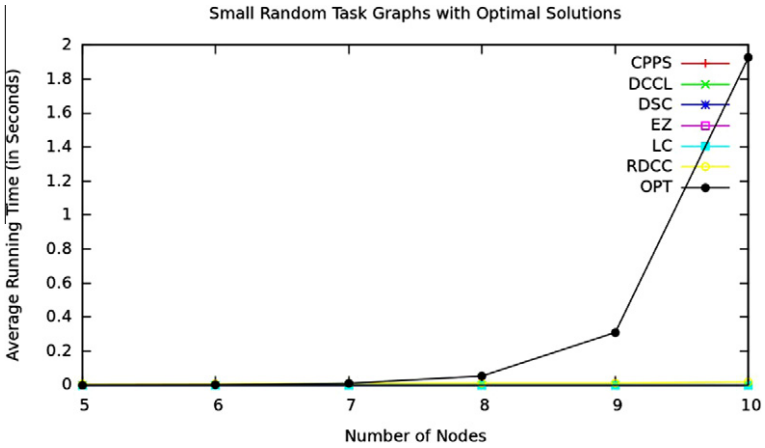

Fig. 33. Average running time (in seconds) vs number of nodes for small random task graphs with optimal solutions. The average performance order is: $LC < DSC < EZ < CPPS < DCCL < RDCC < OPT$.

2.14. Average number of processors used by LC ranges from 2.17 to 4.07 with an average of 2.97. Average number of processors used by RDCC ranges from 1.97 to 2.30 with an average of 2.16. Average number of processors used by OPT ranges from 1.47 to 2.20 with an average of 1.89. The average performance order is: $DCCL < OPT < EZ < RDCC < DSC < CPPS < LC$.

Fig. 33 shows the average running time (in seconds) vs number of nodes for small random task graphs with optimal solutions. Average running time of CPPS ranges from 0.000035 s to 0.000078 s with an average of 0.000059 s. Average running time of DCCL ranges from 0.000027 s to 0.00011 s with an average of 0.000064 s. Average running time of DSC ranges from 0.000021 s to 0.000031 s with an average of 0.000025 s. Average running time of EZ ranges from 0.000029 s to 0.000046 s with an average of 0.000037 s. Average running time of LC ranges from 0.000006 s to 0.000008 s with an average of 0.000007 s. Average running time of RDCC ranges from 0.0058 s to 0.018 s with an average of 0.011 s. Average running time of OPT ranges from 0.00052 s to 1.93 s with an average of 0.38 s. The average performance order is: $LC < DSC < EZ < CPPS < DCCL < RDCC < OPT$.

12. Conclusion

We evaluated the six algorithms (CPPS, DCCL, DSC, EZ, LC, and RDCC) for seven types of task graphs (peer set task graphs, random task graphs, systolic array task graphs, Gaussian elimination task graphs, divide and conquer task graphs, fast Fourier transform task graphs, and small random taks graphs with optimal solutions) using five types of comparison (average NSL vs number of nodes, average NSL vs CCR, average percentage degradation from the best solution vs number of nodes, average number of processors used vs number of nodes, and average running time vs number of nodes).

From the above results it can be concluded that the RDCC algorithm gives best results for the peer set task graphs (in terms of the number of best solutions) and for the case of small random task graphs with optimal solutions (in terms of average NSL). This is as expected because the number of randomization steps ($a = 100$) is too good for small task graphs. The CPPS algorithm gives best average NSL for random task graphs. The DSC algorithm gives best average NSL for systolic array task graphs. The EZ algorithm gives best average NSL for Gaussian elimination task graphs, divide and conquer task graphs, and fast Fourier transform task graphs.

For systolic array task graphs, Gaussian elimination task graphs, divide and conquer task graphs, and fast Fourier transform task graphs we also obseved that the DSC algorithm gives best average NSL for computation intensive task graphs ($CCR = 0.10$), and balanced task graphs ($CCR = 1.00$), whereas the EZ algorithm gives best average NSL for communication intensive task graphs ($CCR = 10.00$). This observation is as expected because of the edge zeroing principal [2].

We observed that the LC algorithm gives least running time because of its simplicity in implementation. We also observed that the DCCL algorithm uses the least number of processors. This observation is as expected because the DCCL algorithm starts with initial cluster on a single processor.

In summary we can say that the dynamic priority algorithms give best results for the case of random task graphs, and for the case when the number of available processors are small. For future work we should work for the benchmarking of heterogeneous task scheduling algorithms.

Acknowledgements

The authors are thankful to the anonymous referees for valuable comments and suggestions in revising the manuscript to the present form. The first two authors greatly acknowledge the financial assistance received under sponsored research project from the CSIR, New Delhi.

References

[1] O. Sinnen, Task Scheduling for Parallel Systems, John Wiley and Sons, 2007.
[2] V. Sarkar, Partitioning and scheduling parallel programs for multiprocessors, Research Monographs in Parallel and Distributed Computing, MIT Press, 1989.
[3] C.H. Papadimitriou, M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, SIAM J. Comput. 19 (1990) 322–328.
[4] T.L. Adam, K.M. Chandy, J.R. Dickson, A comparison of list schedules for parallel processing systems, Commun. ACM 17 (1974) 685–690.
[5] E.G. Coffman, R.L. Graham, Optimal scheduling for two-processor systems, Acta Inform. 1 (1972) 200–213.
[6] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (1969) 416–429.
[7] T.C. Hu, Parallel sequencing and assembly line problems, Oper. Res. 9 (1961) 841–848.
[8] H. Kasahara, S. Narita, Practical multiprocessor scheduling algorithms for efficient parallel processing, IEEE Trans. Comput. 33 (1984) 1023–1029.
[9] C.Y. Lee, J.J. Hwang, Y.C. Chow, F.D. Anger, Multiprocessor scheduling with interprocessor communication delays, Oper. Res. Lett. 7 (1988) 141–147.
[10] Z. Liu, A note on Graham's bound, Inform. Process. Lett. 36 (1990) 1–5.
[11] M.-Y. Wu, D.D. Gajski, Hypercool: a programming aid for message-passing systems, IEEE Trans. Parallel Distrib. Syst. 1 (1990) 330–343.
[12] T. Yang, A. Gerasoulis, List scheduling with and without communication delays, Parallel Comput. 19 (1993) 1321–1344.
[13] P.K. Mishra, K.S. Mishra, A. Mishra, A clustering heuristic for multiprocessor environments using computation and communication loads of modules, Int. J. Comput. Sci. Inform. Technol. 2 (2010) 170–182.
[14] T. Yang, A. Gerasoulis, A fast static scheduling algorithm for DAGs on an unbounded number of processors, in: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, 1991, pp. 633–642.
[15] S.J. Kim, J.C. Browne, A general approach to mapping of parallel computation upon multiprocessor architectures, in: Proceedings of 1988 International Conference on Parallel Processing, 3, 1988, pp. 1–8.
[16] D. Kadamuddi, J.J.P. Tsai, Clustering algorithm for parallelizing software systems in multiprocessors environment, IEEE Trans. Software Eng. 26 (2000) 340–361.
[17] C. Hanen, A. Munier, An approximation algorithm for scheduling dependent tasks on m processors with small communication delays, Disc. Appl. Math. 108 (2001) 239–257.
[18] P.E. Coll, C.C. Ribeiro, C.C. de Sousa, Test instances for scheduling unrelated processors under precedence constraints, 2002, www.ic.uff.br/celso/grupo/readme.ps.
[19] T. Davidovic, T.G. Crainic, Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems, Comput. Oper. Res. 33 (2006) 2155–2177.
[20] Y.K. Kwok, I. Ahmad, Benchmarking and comparison of the task graph scheduling algorithms, J. Parallel Distrib. Comput. 59 (1999) 381–422.
[21] T. Tobita, H. Kasahara, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, J. Sched. 5 (2002) 379–394.
[22] A. Mishra, A.K. Tripathi, An extension of edge zeroing heuristic for scheduling precedence constrained task graphs on parallel systems using cluster dependent priority scheme, J. Inform. Comput. Sci. 6 (2011) 83–96. An extended abstract of this paper appears in the Proceedings of IEEE International Conference on Computer and Communication Technology, 2010, pp. 647–651.
[23] P.K. Mishra, K.S. Mishra, A. Mishra, A clustering algorithm for multiprocessor environments using dynamic priority of modules, Ann. Math. Inform. 38 (2011) 99–110.
[24] P.K. Mishra, K.S. Mishra, A. Mishra, A.K. Tripathi, A randomized scheduling algorithm for multiprocessor environments, submitted for publication.
[25] D.R. Cox, D.V. Hinkley, Theoretical Statistics, Chapman and Hall, London, 1974.
[26] J.E. Freund, R.E. Walpole, Mathematical Statistics 4e, Prentice Hall, Englewood Cliffs, NJ, 1987.
[27] A. Al-Maasarani, Priority-based Scheduling and Evaluation of Precedence Graphs with Communication Times, M.S. thesis, King Fahd University of Petroleum and Minerals, Saudi Arabia, 1993.
[28] M.A. Al-Mouhamed, Lower bound on the number of processors and time for scheduling precedence graphs with communication costs, IEEE Trans. Software Eng. 16 (1990) 1390–1401.
[29] J.Y. Colin, P. Chretienne, C.P.M. scheduling with small computation delays and task duplication, Oper. Res. 39 (1991) 680–684.
[30] Y.C. Chung, S. Ranka, Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributedmemory multiprocessors, in: Proc. Supercomput., 1992, pp. 512–521.
[31] T. Davidovic, Benchmark task graphs, http://www.mi.sanu.ac.rs/~tanjad/sched_results.htm.
[32] O.H. Ibarra, S.M. Sohn, On mapping systolic algorithms onto the hypercube, IEEE Trans. Parallel Distrib. Syst. 1 (1990) 48–63.
[33] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation - Numerical Methods, Athena Sci. (1989).
[34] S. Madal, J.B. Sinclair, Performance of synchronous parallel algorithms with regular structures, IEEE Trans. Parallel Distrib. Syst. 2 (1991) 105–116.
[35] J.P. Kitajima, Modèles Quantitatifs d'Algorithmes Parallèles, Doctorate thesis, Institut National Ploytechnique de Grenoble, 1992.