

# S3MP: A Task Duplication Based Scalable Scheduling Algorithm for Symmetric Multiprocessors

Oh-Han Kang\*  
Department of Computer Education  
Andong National University  
Andong, Kyungbuk, 760-749 Korea  
ohkang@andong.ac.kr

Dharma P. Agrawal†  
Department of ECECS  
University of Cincinnati  
Cincinnati, OH 45221-0030  
dpa@ececs.uc.edu

## Abstract

*We present a task duplication based scalable scheduling algorithm for Symmetric Multiprocessors (SMP), called S3MP (Scalable Scheduling for SMP), to address the problem of task scheduling. The algorithm pre-allocates network communication resources so as to avoid potential communication conflicts, and generates a schedule for the number of processors available in a SMP. This algorithm employs heuristics to select duplication of tasks so that schedule length is reduced/minimized. The performance of the S3MP algorithm has been observed by comparing the schedule length under various number of processors and the ratio of communication to computation cost. This algorithm also has been applied to some practical DAGs.*

## 1. Introduction

The emergence of several high performance workstations has led to the development of parallel computing platform using clusters of workstations. The scalability of clusters of workstations gives them a major advantage over other types of systems. Many future workstations will be SMP with more than one processor. Also SMP systems will be the basis of the clusters of SMP. There has been various hardware technologies and communication protocols designed to speed up communication in this environment. However, one of the major limitations of SMP is the high cost for interprocessor communication, which can be reduced by having an efficient task partitioning and scheduling algorithm.

In parallel computer systems, an efficient task partition-

---

\*This work was done when the author was visiting University of Cincinnati.

†Supported by the National Science Foundation under grant No. CCR-9902748.

ing and scheduling strategy has been regarded as one of the important issues. A task partitioning algorithm partitions an application into tasks and represents them in the form of a directed acyclic graph (DAG). A task scheduling algorithm assigns the tasks, represented as a DAG, onto the processors. Efficient task scheduling of parallel programs can utilize system resources and reduce program parallel execution time [7]. Since an optimal task scheduling problem on the multiprocessors has been proven to be an NP-Complete [10], many scheduling algorithms based on heuristics has been proposed [1, 3]. Until now, many scheduling algorithms based on priority [4], clustering [5, 10], and task duplication [2, 6, 8, 11, 12] have also been proposed. Most of the previous scheduling algorithms assume the presence of a fully-connected network where in all processors can communicate with each other directly. Task scheduling on bus-based SMP is different from that on a fully-connected network. In this topology, two independent communication tasks cannot take place at the same time.

In this paper, we extend STDS algorithm [11], task scheduler of a DAG for a multiprocessor system, to a SMP environment. Also, we propose a new heuristic algorithm, called a scalable scheduling for SMP (S3MP), based on task duplication to solve the scheduling problem on a SMP. The algorithm duplicate certain critical tasks which can reduce schedule length and pre-allocate network communication resources so as to avoid potential communication conflicts. The proposed algorithm can scale down the schedule to the available number of processors on a SMP. To select the duplication tasks, the algorithm uses heuristics so as to reduce schedule length.

The rest of this paper is organized as follows: DAG model and definitions are described in the next section. In section 3, our proposed algorithm, S3MP, is presented, and the running trace of the algorithm on an example DAG is illustrated. Simulation results are presented in section 4. Finally section 5 provides the conclusions.



the *exit node*. While performing the task assignment process, only critical tasks which are essential to establish a path from a particular node to the *entry node* are duplicated. If the favorite predecessor has already been assigned to another processor, it is not duplicated except that there are no other predecessors of the current task or all the other predecessors of the current task have been assigned to another processor. Table 1 shows the values of the mathematical expressions using the example DAG in Figure 1.

Table 1: Values of Mathematical Expressions for DAG

task	level	est	ect	last	lact	fpred	ffpred
$t_1$	20	0	3	0	3	-	-
$t_2$	10	3	9	6	12	1	-
$t_3$	17	3	7	3	7	1	-
$t_4$	12	3	8	3	8	1	-
$t_5$	9	7	12	7	12	3	1
$t_6$	13	7	11	8	12	3	1
$t_7$	4	15	18	15	18	6	3
$t_8$	9	11	19	13	21	6	3
$t_9$	7	12	18	12	18	6	3
$t_{10}$	1	21	22	21	22	8	6

The STDS algorithm generates task clusters which are assigned to a different processor. Based on the values computed in Table 1, the task clusters and the schedule lengths obtained by the algorithm applied to the DAG in Figure 1 are shown in Figure 2. In this schedule, the processor 1,  $P_1$ , has the tasks 1, 3, 6, 8, and 10 for each computation cost of 3, 4, 4, 8, and 1 respectively. Processors  $P_2$  and  $P_3$  have to communicate with  $P_1$  to execute task 7 and task 9 respectively. The communication costs represented by  $C$  are 8 and 3 for each communication, and each communication can begin simultaneously. On a fully connected network such as multiprocessor systems, schedule length becomes 26 units by applying the STDS algorithm to the DAG in Figure 1.

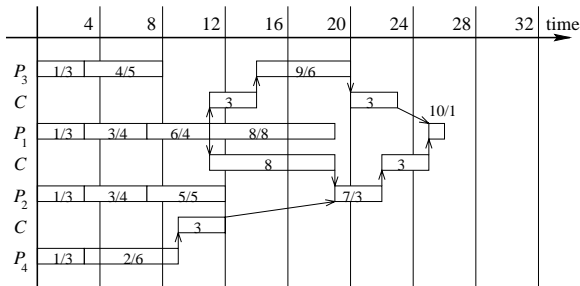


Figure 2. Schedule of the DAG in Figure 1 on fully connected multiprocessor systems

But the algorithms used in fully connected networks are not suitable for a bus-based SMP environment where net-

work conflict could possibly affect communication time seriously. Therefore, we have modified the STDS algorithm to make it appropriate for a bus-based SMP environment. Figure 3 presents the schedule of the example DAG shown in Figure 1 by applying the STDS algorithm on SMP environment.

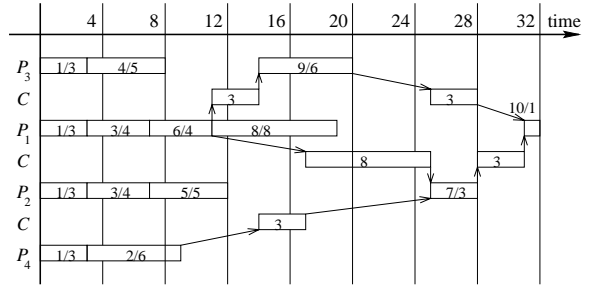


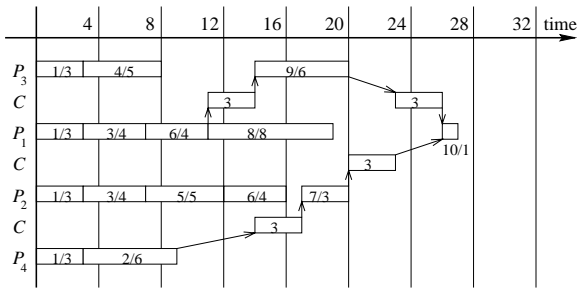
Figure 3. Schedule of the DAG in Figure 1 by STDS algorithm on SMP

The modified STDS algorithm differs from the STDS algorithm used for multiprocessor systems in its scheduling of network communication resources. The modified STDS algorithm pre-allocate network communication resources to avoid potential communication conflict. As shown in Figure 3, network communication resources cannot be shared by two independent tasks in a bus-based SMP. Communications between two pairs of independent processors cannot take place simultaneously. For example,  $P_2$  and  $P_3$  cannot communicate with the first processor  $P_1$  after completion of task 6 due to the network communication conflicts. When the schedule is employed on a bus-based SMP, the actual schedule length becomes 32 units due to network conflicts as shown in Figure 3.

### 3.2. Description of the Proposed Algorithm

Many scheduling algorithms based on the task duplication have been proposed for multiprocessors. But they are not suitable for a bus-based SMP where network conflict does affect execution time seriously. As shown in Figure 3, when the STDS algorithm is employed on a bus-based SMP the schedule length becomes 32 units due to network conflicts. By applying the proposed S3MP algorithm to the DAG in Figure 1, the schedule length can be reduced to 27 units as shown in Figure 4. The S3MP algorithm differs from the modified STDS algorithm in its assigning of duplication task. As shown in Figure 4, after completion of task 6 in  $P_1$  the S3MP algorithm reduces communication cost between  $P_1$  and  $P_2$  by duplicating the task 6 on  $P_2$ .

Using the heuristics, the S3MP algorithm duplicate the favorite predecessor selectively if it can reduce the schedule length. The following heuristics are used in selecting



**Figure 4. Schedule of the DAG in Figure 1 by S3MP**

which tasks to be duplicated. If a favorite predecessor of  $n_i$  already has been assigned to another processor (task cluster), the S3MP algorithm only duplicates it in three cases.

- (1) It is only predecessor of  $n_i$ .
- (2) All the other predecessors of  $n_i$  have been assigned to another processor.
- (3) A favorite predecessors of  $fpred(i)$ ,  $ffpred(i)$ , can be assigned with  $n_i$  to the same processor.

In the S3MP algorithm, the basic parameters are computed in step 1 using relations (1)-(11). Using the parameters computed in step 1, step 2 generates the duplicated task clusters. In this step, heuristics are used to select duplication tasks. If the number of required processor in application is greater than the available processor on SMP, the processor reduction procedure is executed in step 3. In step 4,  $rst$  and  $rct$  of each tasks are generated, and schedule lengths are calculated.

### 3.3. Illustration of the S3MP Algorithm

The steps involved in scheduling the example DAG in Figure 1 are explained below. The schedule of the DAG by the S3MP algorithm is shown in Figure 4. In step 1, algorithm calculates  $level$ ,  $est$ ,  $ect$ ,  $last$ ,  $lact$ , and  $fpred$  for all tasks  $i \in V$ . The  $est$  of the *entry node* is zero, since  $pred(1)=0$ .  $n_1$  completes execution at time three. The  $est$  and  $ect$  of other tasks can be computed by using (4) and (5). For example,  $t_9$  has tasks  $t_4$  and  $t_6$  as predecessors. Thus  $est(9) = \min\{\max\{(ect(4) + c_{49}), ect(6)\}, \max\{(ect(6)+c_{69}), ect(4)\}\} = 12$ . The  $est$  and  $ect$  of all the tasks of the DAG are shown in Table 1. Starting from the *exit node*, the latest allowable start and completion times are computed by using (7)-(9) and are shown in Table 1.

In step 2, algorithm generates clusters which have the duplicated tasks. The search is performed by tracing a path from the initial task selected from *queue* to the *entry node*

by following the favorite predecessors along the way. The elements in the *queue* are the nodes of the DAG sorted in smallest *level* first order. In case above a favorite predecessor of current task has already been assigned to another processor, it is still duplicated if  $ffpred$  of the current task can be allocated to the same processor with the current task. For this example DAG, the array *queue* is  $queue=\{t_{10}, t_7, t_9, t_8, t_5, t_2, t_4, t_6, t_3, t_1\}$ . Starting from the first task in *queue*,  $t_{10}$  in this example, the first cluster is generated. The search process visits through the favorite predecessors of each task. The first processor  $P_1$  is allocated tasks  $t_{10}$ ,  $t_8$ ,  $t_6$ ,  $t_3$ , and  $t_1$ . The next cluster starts from the first unassigned task in *queue*, which is  $t_7$ . the favorite predecessor of  $t_7$  is  $t_6$ , which has already been assigned to  $P_1$ , the search continues with  $t_5$ . Proceeding with the search yields the allocation  $t_7$ ,  $t_5$ ,  $t_3$  and  $t_1$  to  $P_2$ . In this search, the  $t_6$  which is the favorite predecessor of  $t_7$  is not duplicated in this processor  $P_2$ . Hence, the algorithm checks whether the favorite predecessor of  $t_6$ , i.e.  $t_3$ , is assigned in  $P_2$ . Since  $t_3$  which is  $ffpred(7)$  is assigned in  $P_2$ ,  $t_6$  is duplicated to this processor. It can be observed that even though  $t_3$  has already been assigned to  $P_1$ , it is still duplicated on  $P_2$  because it is the only predecessor of  $t_5$ . The next search starts with  $t_9$ . The favorite predecessor of  $t_9$  is  $t_6$ . The task is assigned to  $P_1$  and  $P_2$ . The processor 3,  $P_3$ , is allocated  $t_9$ ,  $t_4$  and  $t_1$ .  $t_6$  which is  $ffpred(9)$  is not duplicated in  $P_3$  because  $t_3$  which is  $ffpred(9)$  is not assigned to this processor. The rest of the allocations are generated by the same process until all tasks have been assigned to a processor. Step 2 generates four clusters, and each cluster is assigned to the processors  $P_1$ ,  $P_2$ ,  $P_3$ , and  $P_4$  respectively as shown in Figure 4.

In step 3, if the number of available processors is less than the number of required processors then the task clusters of different processors have to be merged. Each processor  $i$  is assigned a value of  $exec(i)$ , which is the sum of computing cost of all the tasks on that processor. The algorithm sorts the processors in ascending order of  $exec(i)$  and merges the clusters in logarithmic fashion.

Using the results of step 3, scheduling sequence is determined by step 4. Each scheduling progress in step 4 is explained as in the following. In Figure 4, the first task  $t_1$  in  $P_1$  is selected. A task can start execution only after all its precedent relations of data are satisfied. Since it is *entry node*, it can be scheduled to  $P_1$ . After  $t_1$  is scheduled to  $P_1$ ,  $t_3$  is scheduled to  $P_1$  because its predecessor  $t_1$  is scheduled in  $P_1$ .  $t_6$  and  $t_8$  are scheduled to  $P_1$ , and  $rst$  values of  $t_6$  and  $t_8$  become 7 and 11 respectively. The next scheduling task is  $t_{10}$  which has the unscheduled predecessors  $t_7$  and  $t_9$ . To satisfy data precedence,  $t_7$  and  $t_9$  must be scheduled before  $t_{10}$ . Therefore, the tasks assigned to next processor  $P_2$  are scheduled.  $t_1$ ,  $t_3$ ,  $t_5$ ,  $t_6$  are scheduled sequentially to  $P_2$ .  $rst$  values of  $t_1$ ,  $t_3$ ,  $t_5$ ,  $t_6$  become 0, 3, 7,

and 12 respectively. After  $t_6$  is scheduled to  $P_2$ ,  $t_7$  cannot be scheduled directly because it has an unscheduled predecessor  $t_2$ . Thus, another processor  $P_3$  is selected.  $t_1$  and  $t_4$  are scheduled to  $P_3$  with  $rst$  values 0 and 3 respectively. The next candidate task is  $t_9$  which has two predecessors  $t_4$  and  $t_6$ .  $t_9$  can start because scheduling of the predecessors  $t_4$  and  $t_6$  is completed.  $t_9$  needs to communicate with  $t_4$  and  $t_6$ . Communication cost between task  $t_9$  to  $t_4$  becomes 0 because they are scheduled the same processor  $P_3$ .  $t_6$  consumes network communication resource from time 11 to time 14 to communicate with  $t_9$ .  $rst$  and  $rct$  values of  $t_9$  becomes 14 and 20 respectively. After  $t_9$  is scheduled to  $P_3$ ,  $t_1$  and  $t_2$  in  $P_4$  are scheduled. Next, unscheduled tasks in  $P_1$  can be scheduled.  $t_{10}$  cannot be scheduled because its predecessor  $t_7$  is not scheduled.  $t_7$  can be scheduled because scheduling of its predecessors  $t_2$ ,  $t_5$ , and  $t_6$  is finished. Communication from  $t_2$  to  $t_7$  can start from time 14 which is the earliest start time when the communication resource is free after completion of task 2.  $rst$  and  $rct$  values of  $t_7$  become 17 and 20 respectively. Since the scheduling of tasks in  $P_3$  and  $P_4$  is completed,  $t_{10}$  in  $P_1$  becomes the candidate tasks.  $t_{10}$  needs communication with  $t_7$  and  $t_9$ . Communication from  $t_7$  to  $t_{10}$  will start from time 20 which is the earliest time when the network communication resource is available after  $rct(7)$ . Communication from  $t_9$  to  $t_{10}$  can start from time 23 and consumes channel from time 23 to 26.  $rst(10)$  and  $rct(10)$  become 26 and 27 respectively.

The S3MP algorithm uses heuristics to select duplication tasks in step 2. By duplicating the favorite predecessors selectively, the algorithm can reduce schedule length.  $t_6$  is duplicated to  $P_2$  because it is favorite predecessor of  $t_7$  and  $ffpred(7)$ ,  $t_3$ , is scheduled to the same processor  $P_3$ . Communication from  $t_6$  in  $P_1$  to  $t_7$  in  $P_2$  is required if  $t_6$  which is  $ffpred(7)$  is not duplicated to  $P_2$ . In this case, it consumes channel from time 17 to time 25 and it extends schedule length. At the end of step 4, a schedule length become 27 units by applying the S3MP algorithm to the DAG.

## 4. Performance Comparison

The proposed algorithm has been applied to four practical DAGs. The four applications are Bellman-Ford algorithm (BF), Systolic algorithm (SY), Master-slave algorithm (MS) and Cholesky decomposition algorithm (CD). The first three practical DAGs are part of the ALPES project [9]. The number of tasks in each DAG is around 2,500, and the number of edges varies from 4,902 to 20,298. The number of predecessors and successors varies from 1 to 140 and the computation costs vary from 1 to 20,000 time units. The characteristics for each application are shown in Table 2.

To obtain a wider variation in the communication costs,

$ccr$  has been varied from 1 to 2, 5, and 10 times in practical applications. For each of these cases, the results of S3MP algorithm is compared against modified STDS algorithm, which is based on task duplication. But, unlike STDS algorithm, S3MP algorithm duplicates the favorite predecessor selectively to reduce schedule length.

Table 2: Performance parameters for different applications

Characteristics	BF	CD	SY	MS
$ccr$	0.1261	1.5179	0.0068	0.0068
Number of nodes	2,921	2,925	2,502	2,262
Number of edges	20,298	5,699	4,902	6,711
Processors required by STDS	1,171	432	97	53
Processors required by S3MP	201	75	50	54

For each of  $ccr$  values, the number of processors required by the algorithms to execute these applications are shown in Figure 5. It is shown that the number of processors required by S3MP is much less than that of STDS, while the two algorithms require similar number of processors for the Master-slave algorithm.

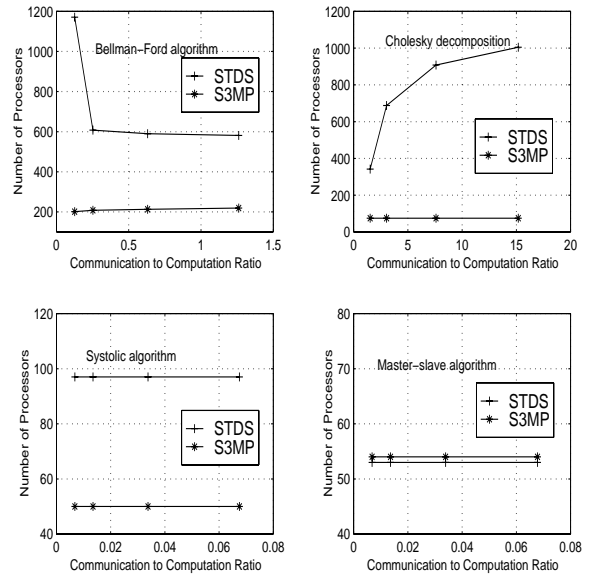
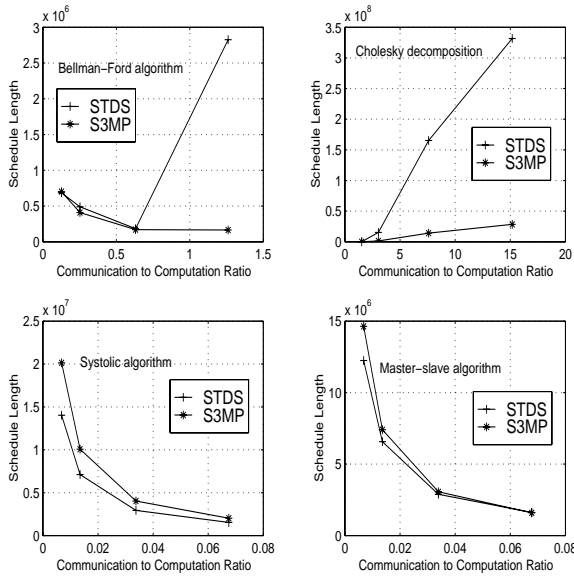


Figure 5. Required number of processors vs.  $ccr$  for practical DAGs

Figure 6 show the schedule lengths generated by the algorithms using each number of those processors. Because the two algorithms require different number of processors for these applications, it is not appropriate to compare the schedule lengths of the two algorithms directly. But we can observe the important characteristics of two algorithms from the results. It can be observed that the schedule length

generated by the S3MP algorithm is similar or less than that of the STDS. On the other hand, the number of processors required by the S3MP algorithm is much lower than that of the STDS.



**Figure 6. Schedule length generated using the required number of processors**

In the Bellman-Ford and Cholesky decomposition algorithms, it is shown that the schedule length of S3MP is lower than that of STDS, and the gap becomes larger as  $ccr$  increases. In the Systolic and Master-slave algorithms, the schedule lengths generated by S3MP are almost same or less than that of STDS. This shows that S3MP algorithm does not perform much better than STDS. This can be explained as follows. For a small  $ccr$  value, the favorite predecessors which had been duplicated to reduce schedule length extend schedule length because communication cost is much smaller relatively than computation cost of these tasks. But we can see that the gap of schedule length of two algorithm becomes almost the same as  $ccr$  value increases.

## 5. Conclusions

This paper presents a task duplication based scalable scheduling algorithm (S3MP) to schedule the tasks of a DAG onto a bus-based SMP environment. The S3MP algorithm duplicates certain critical tasks which can reduce schedule length using the heuristics. The proposed scheduling algorithm can be scaled down to the available number of processors in SMP. It has been observed that the S3MP algorithm performs very well in reducing the number of processors when the available number of processors is low.

The S3MP algorithm has been compared to the STDS algorithm in terms of the schedule length. The performance of this algorithm has been observed by its application on some practical DAGs for different communication to computation ratios.

## References

- [1] A. Gerasoulis, and T. Yang, *A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors*, in *Journal of Parallel and Distributed Computing*, vol. 16, pp. 276-291, 1992.
- [2] B. Shi, H-B. Chen, and J. marquis, *Comparative Study of Task Duplication Static Scheduling versus Clustering and Non-Clustering Techniques*, in *Concurrency: Practice and Experiences*, vol. 7, no. 5, pp.371-389, 1995.
- [3] C.L. McCreary, A.A. Khan, J.J. Thompson, and M.E. McArdle, *A Comparison of Heuristics for Scheduling DAGS on Multiprocessors*, in *Proceedings of Eighth International Conference on Parallel Processing*, pp. 446-451, 1994.
- [4] G.C. Sih and E.A. Lee, *A Compile-time Scheduling Heuristic for Interconnection-constrained Heterogeneous Processor Architectures*, in *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp.175-187, May 1993.
- [5] G.L. Park, B. Shirazi, and J. Marquis, *DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems*, in *Proceedings of Parallel Processing Symposium*, pp.157-166, 1997.
- [6] H. Chen, B. Shirazi, and J. Marquis, *Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication*, in *Proceedings of International Conference on Parallel and Distributed Systems*, pp. 270-275, Dec. 1993.
- [7] H. El-Rewini, H.H. Ali, and T. Lewis, *Task Scheduling in Multiprocessing Systems*, in *IEEE Computer*, vol. 28, no. 12, pp. 27-37, Dec. 1995.
- [8] I. Ahmad, and Y.K. Kwok, *A New Approach to Scheduling parallel Program using Task Duplication*, in *Proceedings of International Conference on Parallel Processing*, vol. II, Aug. pp. 47-51, 1994.
- [9] J.P. Kitajma, and B. Plateau, *Building Synthetic Parallel Programs: The Project (ALPES)*, in *Proceedings of the IFIP WG 10.3 Workshop on Programming Environments for Parallel Computing*, pp. 161-170, 1992.
- [10] R.L. Graham, L.E. Lenstra, J.K. Lenstra, and A.H.Kan, *Optimization and approximation in deterministic sequencing and scheduling; A survey*, in *Annual Discrete Mathematics*, pp. 287-326, 1979.
- [11] S. Darbha, and D.P. Agrawal, *A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems*, in *Journal of Parallel and Distributed Computing*, vol. 46, pp. 15-26, 1997.
- [12] S. Darbha, and D.P. Agrawal, *SDBS: A Task Duplication Based Optimal Scheduling Algorithm*, in *Proceedings of Scalable High-Performance Computing Conference*, pp. 756-763, 1994.