Volume 1 No. 1, May 2011
International Journal of Information and Communication Technology Research

http://www.esjournals.org

A New Algorithm for Static Task Scheduling For Heterogeneous Distributed Computing Systems

[1]Nirmeen A. Bahnasawy, [2]Fatma Omara, [3]Magdy A. Koutb, [4]Mervat Mosa
[1,3,4]Faculty of Engineering, Menoufia University.
[2]Faculty of computers and science, Cairo University.
nirmeen_a_wahab@hotmail.com

ABSTRACT

Effective task scheduling is essential for obtaining high performance in **He**terogeneous **D**istributed **C**omputing **S**ystems (HeDCSs). However, finding an effective task scheduling in HeDCSs should take into consideration the heterogeneity of processors and inter-processor communication over head, which results from non-trivial data movement between tasks scheduled on different processors. In this paper, a new high performance task scheduling algorithm called **S**orted **N**odes in **L**eveled **D**AG **D**ivision (SNLDD) is presented for HeDCSs with considering a bounded number of processors. The main concept of the proposed algorithm is to divide the **D**irected **A**cyclic **G**raph DAG into levels and tasks in each level are sorted in descending order according to their computation size. A new attribute has been introduced and used to efficiently select tasks for scheduling in HeDCSs. This selection of tasks enables the proposed SNLDD algorithm to generate high-quality task schedule in a heterogeneous computing environment. To evaluate the performance of the proposed SNLDD algorithm, a comparison study has been done between it and the **L**ongest **D**ynamic **C**ritical **P**ath (LDCP) algorithm which is considered the most efficient algorithm. According the comparative results, it is found that the performance of the proposed algorithm provides better performance than the LDCP algorithm in terms of speedup, efficiency, complexity, and quality.

Keywords: *Task scheduling; directed acyclic graph; heuristics; parallel processing; heterogeneous distributed computing systems*

I. INTRODUCTION

A **D**istributed **C**omputing **S**ystem, or DCS, is a group of processors connected via a high speed network that supports the execution of parallel applications [1]. The efficiency of executing parallel applications on the DCSs critically depends on the used method to schedule the tasks of the parallel application onto the available processors [2]. In the DCSs, inter-processor communication is an unavailable overhead of the execution of parallel programs [3]. This overhead occurs when tasks allocated to different processors exchange data. Therefore, creation of high quality task schedules becomes more critical when the parallel applications are executed on the heterogeneous distributed computing systems [4]. In addition to the tradeoff between the gained speedup through parallelization and the overhead of inter-processor communication, scheduling algorithms for the HeDCSs have to consider the various execution times of the same task on different processors. A faulty scheduling decision in HeDCSs may limit the performance of the system by the capabilities of the slowest processors [5]. In general, task scheduling algorithms for DCSs are classified into two classes; static and dynamic. According to static scheduling algorithms, all information needed for scheduling, such as the structure of the parallel application, the execution times of individual tasks and the communication costs between tasks must be known in advance [5]. There are several techniques to estimate such information [6], [4]. Static task scheduling takes place during compile time before running the parallel application [2], [7]. In contrast, scheduling decisions in dynamic scheduling algorithms are made at run time [8]. The objective of dynamic scheduling algorithms includes not only creating high quality task schedules, but also minimizing the run time scheduling overheads [9], [10]. The static scheduling is addressed in this paper. Moreover, in typical scientific and engineering applications, compile time, including the static scheduling time, is much lower than that the run time [4]. By increasing scheduling complexity to create high quality task schedules, which reduce the run time of the parallel applications, will improve the overall performance of DCSs [11].

Examples of existing task scheduling algorithms are; Heterogeneous Earliest Finish Time (HEFT) [12], Critical Path On a Processor (CPOP) [13], Critical Path On a Cluster (CPOC) [13], Dynamic Level Scheduling (DLS) [13], Modified Critical Path (MCP) [5], Mapping Heuristic (MH) [12] and Dynamic Critical Path (DCP) [4]. [13] has presented a performance comparative study among the HEFT, CPOP, DLS, and MH algorithms for different values of DAG size. According to their study, the performance of the HEFT algorithm outperforms the CPOP, DLS, and MH algorithms. Moreover, the performance of the DLS algorithm outperforms the MH algorithm. The CPOP algorithm and the DLS algorithm achieved comparable results. Also, the performance of the HEFT and Heterogeneous N-predecessor Decisive Path (HNPD) algorithms is compared in [14], where the latter combines both list-based scheduling and multiple task duplication. When the number of processors is equal to one-forth the number of tasks, the HEFT algorithm

outperforms the HNPD algorithm. On the other hand, for unlimited number of processors the HNPD algorithm outperforms the HEFT algorithm. Since the HNPD algorithm employs multiple task duplication, the HNPD algorithm requires a greater number of processors than that the HEFT algorithm to achieve the same schedule length [13].

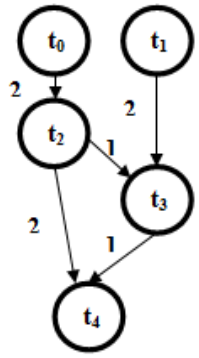
Recently, a new algorithm called Longest Dynamic Critical Path (LDCP) has been introduced [13]. According to the LDCP algorithm, a new attribute has been used to accurately identifying the priorities of tasks in the HeDCSs. The performance of the LDCP algorithm is compared to the HEFT [12] and the DLS [5]algorithms and outperformed them.

In this paper, a new algorithm called Sorted Nodes in Leveled DAG Division (SNLDD) is introduced for static task scheduling for the HeDCSs with limited number of processors. The motivation behind this algorithm is to generate the high quality task schedule that is necessary to achieve high performance in the HeDCSs. The main concept of the proposed algorithm is to divide the Directed Acyclic Graph (DAG) into levels and the tasks in each level are sorted according to their computation size in descending order. To evaluate the performance of the proposed SNLLD algorithm, a comparative study has been done between it and the LDCP algorithm. According to the comparative study, the SNLDD algorithm outperforms the LDCP algorithm in terms of schedule length, speedup, efficiency, and quality of system behavior.

The remainder of this paper is organized as follows: in section 2, the task scheduling problem and some necessary terms are defined. Section 3, the LDCP algorithm for task scheduling in the HeDCSs is introduced. The new proposed SNLDD algorithm is introduced in section 4. The comparative results of the proposed SNLDD algorithm and the LDCP algorithm are presented in section 5, and finally, conclusions are given in section 6.

II. PROBLEM DEFINITION

In static task scheduling for HeDCSs, the parallel application is represented by DAG. DAG is defined by the tuple (*T*, *E*), where T is a set of n tasks and E is a set of e edges. Each task $t_i \in T$ represents a task in the parallel application, and each edge $(t_i, t_j) \in E$ represents a precedence constraint and a communication message between tasks t_i and t_j. If $(t_i, t_j) \in E$, then the execution of $t_j \in T$ cannot be started before $t_i \in T$ finishes its execution. The source task t_i of an edge (t_i, t_j) is a parent of the sink task t_i, while t_j is a *child* of t_i. A task with no parents is called an *entry task*, and a task with no children is called an *exit task*. Associated with each edge (t_i, t_j), there is a value $d_{i,j}$ that represents the amount of data to be transmitted from task t_i to task t_j [5], [12]. The HeDCSs is represented by a set used P of m processors that have diverse capabilities. The $n \times m$ computation cost matrix W stores the execution costs of n tasks in m processors. Each element $w_{i,j} \in W$ represents the estimated execution time of task t_i on processor p_j. All processors are assumed to be fully connected. Communications between processors occur via independent communication units; this allows for concurrent execution of computation tasks and communications between processors [16]. The computation costs of tasks are assumed to be monotonic. In other words, if the computation cost of task t_i on processor p_j is higher than that on processor p_k, then the computation costs of any task on p_j is higher than or equal to that on processor p_k. The communication cost between two processors p_k and p_l depends on the network initialization at processors p_k and p_j in addition to the communication time on the network. The time required to initialize the network at the sender and receiver processors is considered to be ignorable compared to the communication time on the network [8]. The data transfer rate between any two processors on the network is assumed to be fixed and constant [5]. Therefore, the communication cost of an edge (t_i, t_j) is equal to the amount of data transmitted from task t_i to task t_j, or $d_{i,j}$ divided by the data transfer rate of the network. Without loss of generality, the data transfer rate of inter-processor network is assumed to be unity [15]. Hence, the communication cost of an edge (t_i, t_j) is equal to $d_{i,j}$ given that tasks t_i and t_j are scheduled on different processors. Since the data transfer rate of the intra-processor bus is much higher than the data transfer rate of the inter-processor network, the communication cost between two tasks scheduled on the same processor is taken as zero. A task can start execution on a processor only when all data from its parents become available to that processor; at that time the task is marked as ready. Tasks must be scheduled and assigned to processors in a way that minimizes the total run time, or the *schedule length*, of the parallel application [10], [12]. An example of a DAG of a parallel application and a computation cost matrix with two processors is shown in Fig. 1.



(a)

Task	P0	P1
t_0	5	8
t_1	11	15
t_2	4	7
t_3	6	7
t_4	1	2

(b)

Fig. 1 An example of a DAG and computation cost matrix

III. THE LONGEST DYNAMIC CRITICAL PATH (LDCP) ALGORITHM

The most recent algorithm called Longest Dynamic Critical Path (LDCP) algorithm has been

introduced by [13]. According to the LDCP algorithm (see Fig. 2), each scheduling step consists of three phases; task selection, processor selection and status update.

Task Selection Phase

A set of tasks that play an important role in determining the ***provisional*** schedule length is identified. To compute the LDCPs, a *directed acyclic graph that corresponds to a processor* (DAGP) is constructed for each processor in the system according to definition 1. These DAGPs are constructed at the beginning of the scheduling process.

Definition 1: Given a DAG with *n* tasks and *e* edges and a HeDCS with *m* heterogeneous processors $\{p_0, p_1, \ldots, p_{m-1}\}$, the **D**irected **A**cyclic **G**raph that corresponds to **P**rocessor p_j, called $\mathbf{DAGP}_j$, is constructed using the structure of the DAG, with sizes of tasks set to their computation costs on processor $\boldsymbol{p_j}$.

Processor Selection Phase

In this phase, the selected task is assigned to a processor that minimizes its finish execution time.

Status Update Phase

When a task is scheduled on a processor, the status of the system must be updated to reflect the new changes. The scheduling of task $\boldsymbol{t_i}$ on processor $\boldsymbol{p_j}$ means that the computation cost of $\boldsymbol{t_i}$ is no longer unknown. Hence, the sizes of the nodes that identify $\boldsymbol{t_i}$ are set to the computation cost of $\boldsymbol{t_i}$ on $\boldsymbol{p_j}$ on all DAGPs. Moreover, a value of zero is assigned to all edges that extend between the nodes that identify $\boldsymbol{t_i}$ and the nodes that identify its parents that are scheduled on processor $\boldsymbol{p_j}$. This must be done for all DAGPs to indicate the zero communication cost between tasks scheduled on the same processor. The insertion of task $\boldsymbol{t_i}$ into processor $\boldsymbol{p_j}$ will result in new execution constraints.

Construct DAGPs for all processors in the system

While *there are unscheduled tasks* ***do***

Find the key DAGP

Find the key node in the key DAGP

If *the key node has no unscheduled parents then*

Identify the selected task using the key node

Else

Find the parent key node

Identify the selected task using the parent key node

End if

Compute the finish time of the selected task on every processor in the system

Find the selected processor that minimizes the finish time of the selected task

Assign the selected task to the selected processor

Update the size of the nodes that identify the selected task on all DAGPs

Update the communication costs on all DAGPs

Update the execution constrains on all DAGPs

Update the temporary zero-cost edges on the DAGP associated with the selected processor

Update the URank values of the nodes that identify the scheduled tasks on all DAGPs

End while

Fig. 2 Longest Dynamic Critical Path (LDCP) algorithm

IV. The Sorted Nodes in Leveled DAG Division Algorithm (SNLDD)

According to the work in this paper, a new task scheduling algorithm called Sorted Nodes in Leveled DAG Division (SNLDD) has been developed. The developed SNLDD algorithm is based on dividing DAG into levels with considering the dependency priority conditions among tasks in the DAG. The tasks in each level will be sorted into a list based on their computation size. The tasks will be assigned to the earliest processors according to their priority in the list. The computation size of each task is calculated by the following equation:

$$S_j(n_i) = \left(w_j(n_i)\right)_{f\ p} + \{c_j[(n_i)_j, \sum_{k=1}^{t}(n_k)_{j-1}] + c_j[(n_i)_j, \sum_{x=1}^{q}(n_x)_{j+1}]\} \quad (1)$$

Where; $\boldsymbol{S_j\ (n_i)}$ is the computation size of the specified task ($\boldsymbol{n_i}$) in the ***j*** level where **1<=** ***j*** **<=** ***R, R*** is the total number of levels and ***1<= i <=T***, where ***T*** is the total number of tasks. The first part of the equation computes the execution time of task $\boldsymbol{n_i}$ from ***j*** level by the fastest processor ***p*** in the system. While the second part determines the sum of communication between the task $\boldsymbol{n_i}$ in ***j*** level and all of its parents in ***j-1*** level individually, and the sum of communications of its Childs in ***j+1*** level. Fig. 3 shows the pseudo code of the developed SNLDD algorithm.

According to the LDCP algorithm, the required tasks of DAG based on the longest path computation. These computations are repeated after assigning each task

which is caused a lot of arithmetic computations of communication overheads [13]. Therefore, our developed SNLDD algorithm is based on dividing the DAG into levels and tasks in each level are assigning to processors. So, the computations of communication overhead are elevated. By dividing the DAG into levels based on dependency conditions and the tasks in each level are sorted according to computation sizes in our developed (SNLDD) algorithm, this leads to simplify the method for classification of tasks according to the priority, which is considered more efficient than that the LDCP algorithm because the required time for choosing the returned task to be assigned will be computed in each step. A high quality schedule has been created using the proposed SNLDD algorithm without introducing runtime overheads which could be resulted from updating the extracting valuable task at every assigning step as in the LDCP algorithm.

On the other hand, the computation size of tasks not only allows deciding which task will be chosen and ordering the tasks according to their computation sizes, but also allows to generate complete system of classification tasks according to many properties such as its communication cost, dependency, its computation, and its order among the tasks in DAG, so that the choice of task in our developed SNLDD algorithm will reduce the total required time. In addition, sorting computation sizes of the tasks according to their computation sizes in descending ordering leads to get red of the heaviest tasks first to reduce the complicated communications dependency between them**.** If the computation sizes of more than one task are equal, the tie is solved by choosing tasks with large number of communication link.

Generally, by dividing DAG into levels and assigning tasks in each level, our developed SNLDD algorithm is become more efficient than that the LDCP algorithm for the following reasons:

- The LDCP algorithm needs to update the whole tasks, paths, processing time, and communication links after each assigning step which is not needed in our developed SNLDD algorithm, then the run time overheads is eliminated in the SNLDD algorithm.
- Assigning the tasks to processors according to computation size satisfy not only efficient task scheduling but also allows to generate complete system of classification of tasks according to many properties such as its communication cost ,dependency , and its computation time.
- Sorting the tasks in each level according to its computation size leads to get red off the task with heaviest computation size first which reduces the dependency between tasks.
- The sleek time of processors is minimized because of dividing the DAG in to levels and tasks in each level are assigning to processors.
- On the other hand, the authors in [13] have proved that the LDCP algorithm is considered more efficient than that the HEFT and LCD algorithms. On the other hands, our developed SNLDD algorithm is considered more efficient than that the LDCP algorithm, then our developed SNLDD algorithm is considered more efficient than that LDCP, HEFT, and LCD algorithms.
- Many ideas of most existing algorithms such as sorted list algorithm [14], clustering algorithms [3] and hierarchy as tree algorithms [10], are verified in our algorithm.

According to the developed SNLDD algorithm, the computation size for all tasks in the DAG is computed only once, while in the LDCP algorithm the longest path is computed at every assigning step, and the updating of the task selection, processor selection, and the communication status are also computed on each step. These will take time and calculations more than that in developed SNLDD algorithm.

Generate the DAG

Divide the DAG into levels according their communicated dependency ***/*DAG division*/***

Sort the constructed levels according to dependency ordering ***/*dependency conditions*/***

Compute the computation size of each task in each level according to the next equation

$$S_j(n_i) = (w_j(n_i))_{p \atop f} + \{c_j[(n_i)_j, \sum_{k=1}^{t}(n_k)_{j-1}] + c_j[(n_i)_j, \sum_{x=1}^{q}(n_x)_{j+1}]\} \quad (1)$$

Sort tasks according to[their direct communication of its next level then their computation sizes] in descending order

While *there are unscheduled levels* ***do***
/*levels assigning*/

While *there are unscheduled tasks* ***do***
/*task assigning*/

If *there are tasks highly communicated with tasks in direct next level*

Assign this parent with its Childs at the earliest processor

If *there are tasks in level = number of processors in system*
Then
Assign the largest computation size of these tasks with these (parent and childs) in earliest processor
Else
Assign tasks from the list of tasks
If $S_j(n_i)=S_j(n_{i+1})$
Then
Choose the highest communication lines first
Endif
Endif

Else
Compute their computation sizes and sort them in descending order according to equation (1)
Assign them with its childs at earliest processor
Endif
Compute the finish time of selected task in all processors in the system
Select the earliest processor
If *there is an idle time according to the communication of task and its parent*
Assign this task to the next earliest processor which will overcome an idle time
Else
Assign this task to this processor
If
There are more than one task are equal in Sj(ni) and all their procedures conditions
Then
Assign them to processors exchanging them taking into account the
update of unscheduled levels
End if
find the selected processor that minimizes the finishing time of selected task
update the computation size of nodes of tasks in the level
End if
Else
Assign the next task
End while
Assign the next level
End while

Fig. 3 The pseudo code of developed SNLDD algorithm

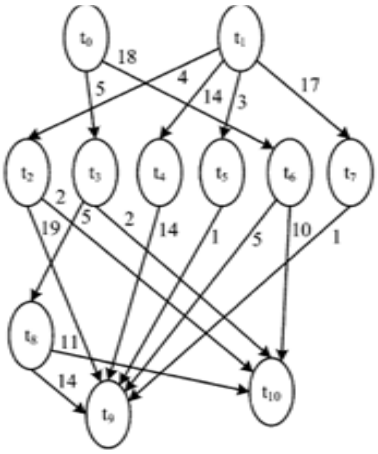
We can conclude that the time complexity of SNLDD algorithm is ***Θ(mxn^2)*** while the time complexity of LDCP algorithm is ***Θ(mxn^3)***, where *m* is the number of processors, and *n* is the number of tasks.

Example 1

By considering the application DAG and the computation cost matrix in Fig. 1. The schedule length according to the proposed SNLDD algorithm is **23** units; whenever the LCDP algorithm is **24** units.

Example 2

Considering the application DAG and the computation cost matrix as shown in Fig. 4. The generated schedule along with stepwise trace of the LCDP algorithm and SNLDD algorithm are shown in Fig. 5 and 6 respectively.



Task	p0	p1
t0	4	6
t1	15	22.5
t2	4	6
t3	13	19.5
t4	10	15
t5	7	10.5
t6	8	12
t7	4	6
t8	12	18
t9	6	9
t10	9	13.5

Figure 4(a) A sample DAG and (b) Computation cost

The schedule generated by SNLDD algorithm has length of **63,** while the schedule length generated by LCDP algorithm is **64**. So, the SNLDD algorithm has shorter execution length than that the LDCP algorithm. Also, by using the SNLDD algorithm, there is no idle time within processors which leads to good utilization of processors in the system. So, the SNLDD algorithm achieves high performance and quality than that the LDCP algorithm.

V. SIMULATION RESULTS

To evaluate the performance of our developed SNLDD algorithm, a simulator of a heterogeneous distributed system has been built using C# ver.5.1 and core 2 due processor with 1.73 Mega Hertz.

SNLDD algorithm schedules an unknown number of tasks for processing on a distributed system with a minimum execution time on the processors of the heterogeneous distributed system. The processors of the distributed system are heterogeneous, so, SNLDD algorithm is measured by Standard Task Graph Set (STG) which is a kind of benchmark for evaluation of multiprocessor scheduling algorithms. STG is proposed for every researcher to evaluate their algorithms under the same conditions covering various task-graph (TG) generation methods including task graphs generated from actual application programs [17].

An explicit comparison with some other well-known scheduling algorithms for HeDCSs, such as CPOP, MH and LMT, is not carried out as the HEFT and DLS algorithms have already been tested against them, and have given better or at worst very similar results [13].
To test the performance of the scheduling algorithms, SNLDD algorithm and LDCP a simulation environment for computer clusters is built and run on an acer core due processor with 1.73 speedup computer. The LDCP algorithm as well as the SNLDD algorithm is implemented. Two sets of parallel application graphs, which correspond to both random application DAGs and

DAGs of parallel numerical applications, are created. The scheduling algorithms are run on the application graphs to generate output schedules. Finally, a group of performance metrics is applied to the schedules generated by the two scheduling algorithms. A set of randomly generated graphs is created by varying a set of parameters that determines the characteristics of the generated DAGs.

These parameters are described below:

• ***DAG size; n***: The number of tasks in the DAG.

• ***Communication to computation cost ratio;*** *CCR*:
The average communication cost divided by the average computation cost of the application DAG.

•With four different numbers of processors varying from 2, 4, 8, and16 processors. For each number of processors, five different DAG sizes have been used varying from 20 to 100 nodes with an increment of 20.

To test the SNLDD algorithm the data collected for task graphs of 20 to 100 nodes and processor graphs of 2, 4,8,and 16 nodes. Fig. 7, 8, 9, and 10 show schedule length of LDCP and SNLDD algorithms. According to the result, the schedule length decreases, the running time of program decreases, so the system required memory decreases as a result the memory efficiency increases these results satisfied in just the SNLDD algorithm. So, the SNLDD algorithm is more efficient than the LDCP algorithm.

The SNLDD algorithm formalized in Fig. 3 has a time complexity of *O(mxn^2)* where *m* is the number of processors, and *n* is the number of tasks. In a comparison, the time complexity of the LDCP algorithm is In the LDCP algorithm, each scheduling step consists of three phases: task selection, processor selection and status update *O(mxn^3)* .

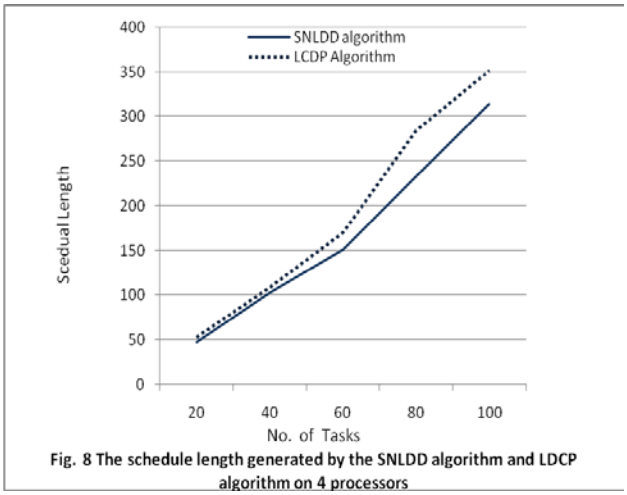


Fig. 8 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 4 processors

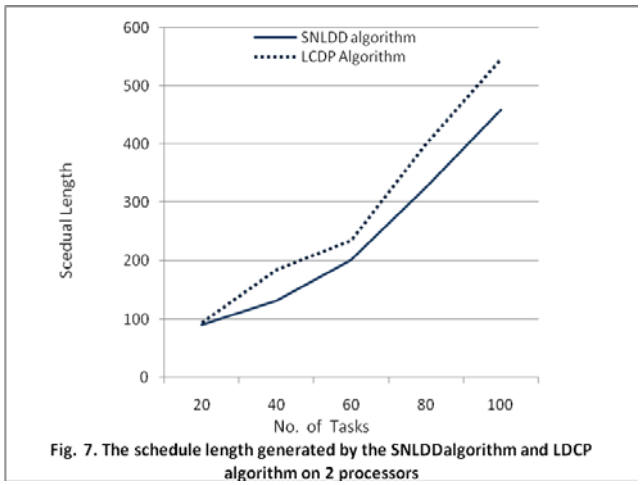


Fig. 7. The schedule length generated by the SNLDDalgorithm and LDCP algorithm on 2 processors

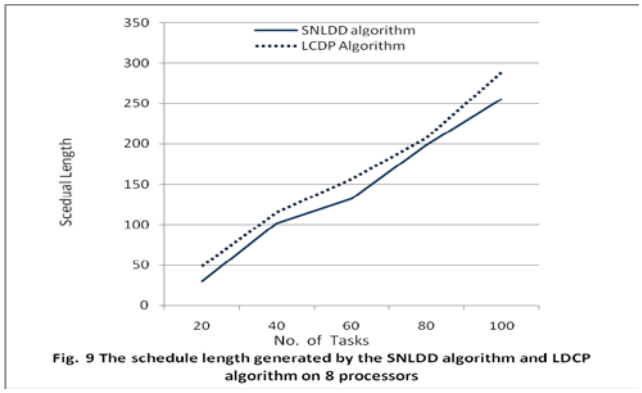


Fig. 9 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 8 processors

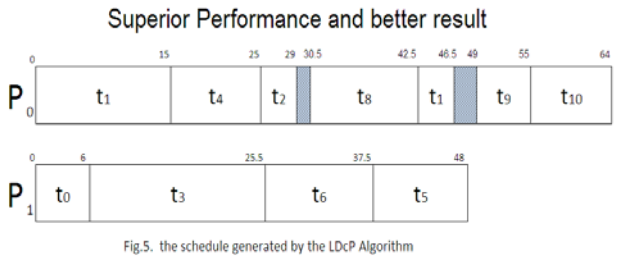


Fig.5. the schedule generated by the LDcP Algorithm

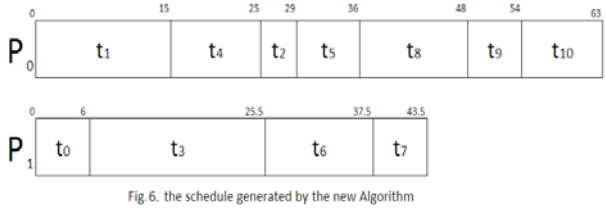


Fig.6. the schedule generated by the new Algorithm

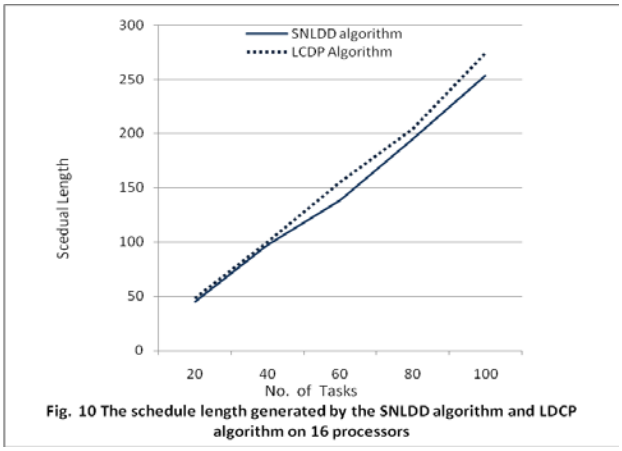


Fig. 10 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 16 processors

According to the work in this paper, a comparative study has been done between the SNLDD and LDCP algorithms. We will study the scheduling of a parallel program that has been represented in the form of DAG. The performance of the two approaches will be reported using the performance criteria described below.

Speedup is a good measure for the execution of an application program on a parallel system. The speedup of a schedule is defined as the ratio of the schedule length obtained by assigning all task to the fastest processor, to the parallel execution time of the task schedule.

Linear speedup means that the value of speedup increases as the number of processors in the parallel system increases.

Assume *T(1)* is the time required for executing a program on a fastest processor and *T(m)* is the time taken for executing the same program on *m* processors. Thus the speedup can be estimated *as*

$$S(m)=T(1)/T(m) \qquad 1 \leq S(m) < m$$

In ideal case, $S(m) = m$ but in actual case $1 \leq S(m)$.

The results of the comparative study according to the speedup parameter have been presented in Figures 11, 12, 13, 14 and 15.

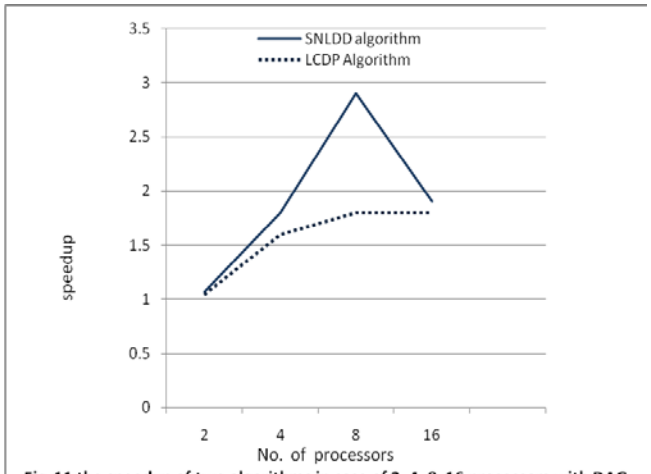


Fig. 11 the speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 20 tasks

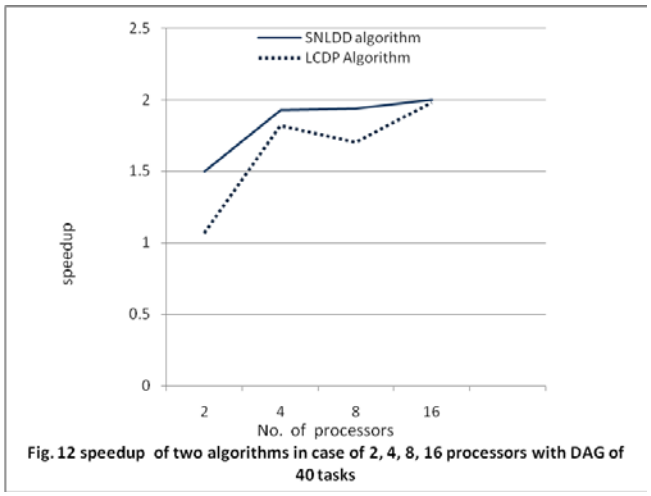


Fig. 12 speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 40 tasks

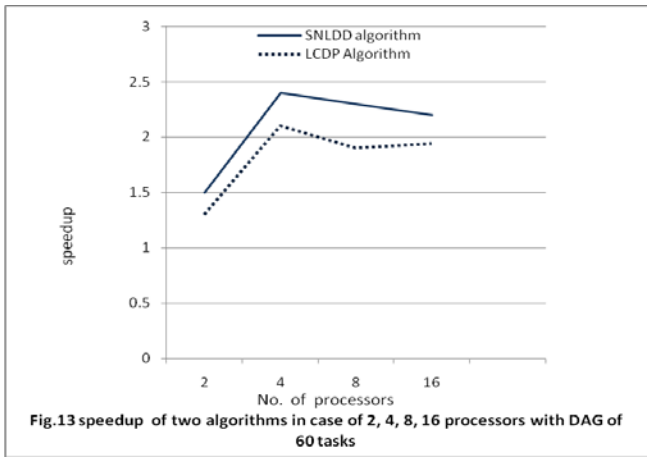


Fig.13 speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 60 tasks

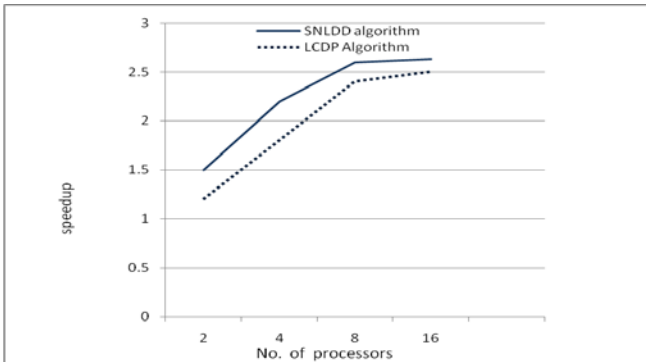


Fig.14 speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 80 tasks

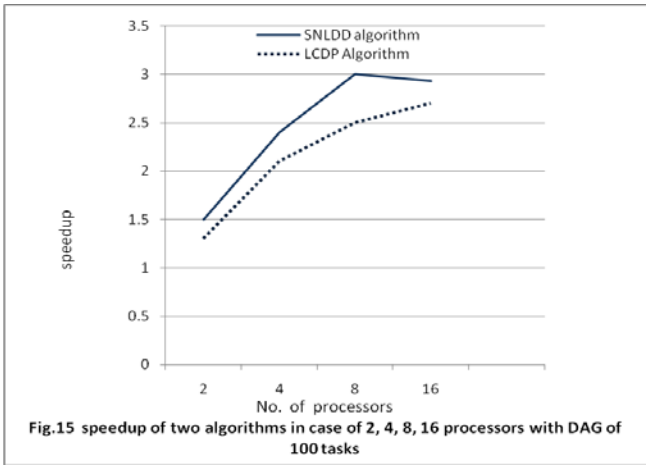


Fig.15 speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 100 tasks

According to the results, it is clear that the SNLDD algorithm outperforms the LDCP algorithm by 21.3%

Also the efficiency of the parallel computers is an indication to what percentage of a processors time is being spent in useful computation. The efficiency of a parallel computer containing m processors can be defined as:

$$E(m)=S(m)/m \qquad 1/m \leq E(m) \leq 1$$

Maximum efficiency $E(m) = 1$ is achieved when all the processors are fully utilized during all time periods of the program execution. The lowest efficiency result, in the case of the program is not suitable to be executed in parallel computer. Quality of parallelism is directly proportional to the speedup and efficiency [3]. The quality is always supper-bound by the speedup.

The comparative study between the two scheduling algorithms SNLDD and LDCP has been implemented using different number of processors (2, 4, 8, and16). The comparison results are presented in Figures 16, 17, 18, 19 and 20.

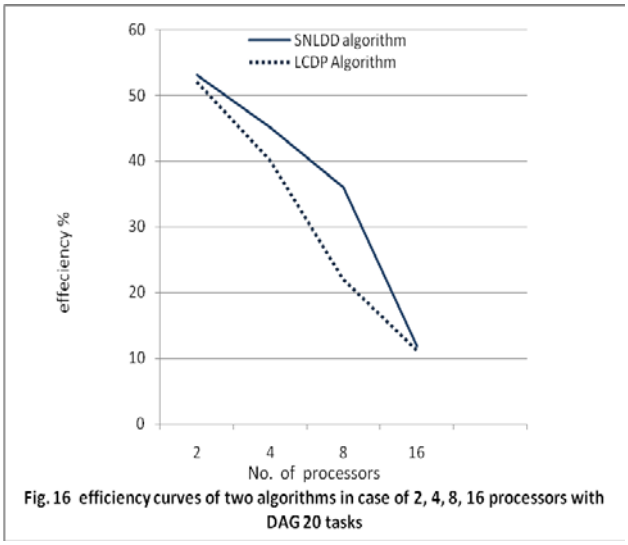


Fig. 16 efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG 20 tasks

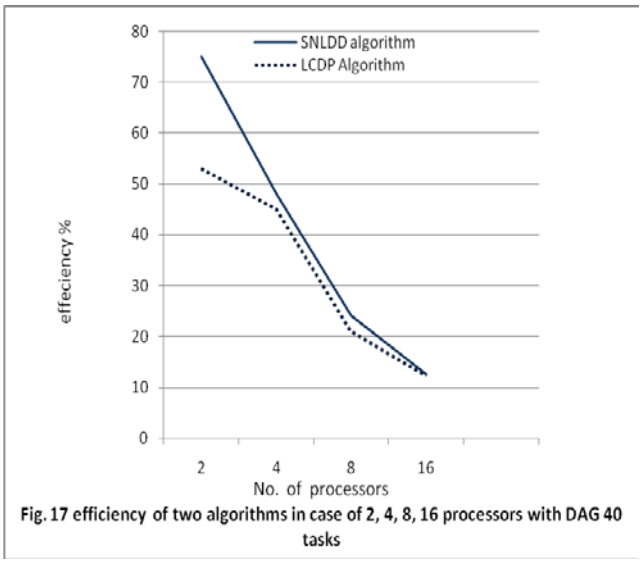


Fig. 17 efficiency of two algorithms in case of 2, 4, 8, 16 processors with DAG 40 tasks

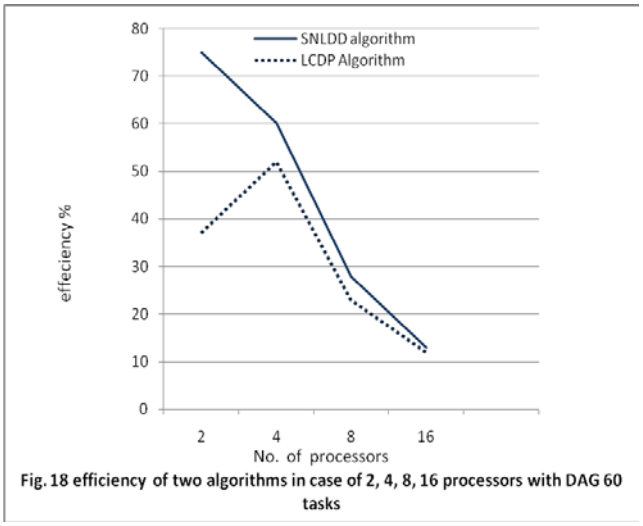


Fig. 18 efficiency of two algorithms in case of 2, 4, 8, 16 processors with DAG 60 tasks

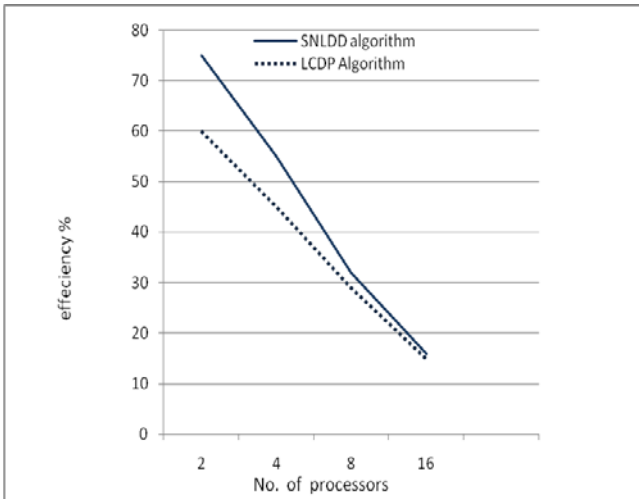


Fig.19 efficiency of two algorithms in case of 2, 4, 8, 16 processors with DAG 80 tasks

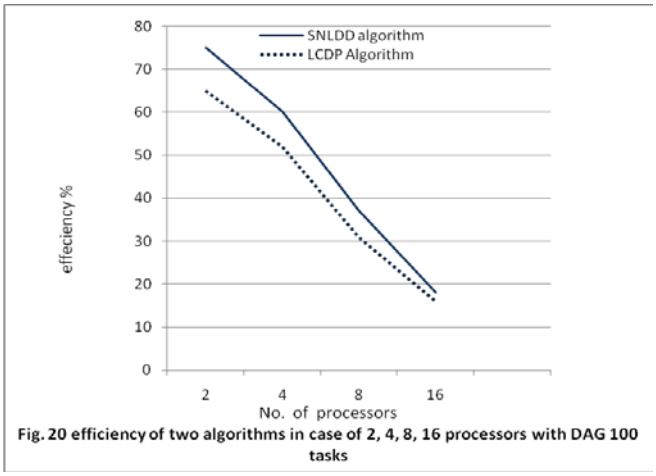


Fig. 20 efficiency of two algorithms in case of 2, 4, 8, 16 processors with DAG 100 tasks

According to the comparative results, it is found that our intelligent SNLDD algorithm behaves better than that the LDCP algorithm by 31.3%.

According to the results in figures 7-20, it is clear that our proposed SNLDD algorithm is always outperforms the LDCP algorithm in terms of schedule length conditions, speedup conditions, and efficiency conditions.

From these results, it is cleared that our proposed SNLDD algorithm has satisfied the mean ratio of the performance improvement resulted by 28.6% than LDCP algorithm. According to the schedule length parameter, our proposed SNLDD algorithm achieves better performance than LDCP algorithm in the DAGs which have levels less than paths number which is considered the most famous existing kinds of DAGs.

VI. CONCLUSIONS

In this paper, a new scheduling algorithm is presented for heterogeneous distributed computing systems HeDCSs. This algorithm uses a new attribute based on dividing the DAG into levels according to the precedence relations, and packing each level in descending order, and then the task is chosen from that level according to its computation size, to accurately identify the priorities of task in HeDCSs.

The performance of the SNLDD algorithm is compared to the LDCP algorithm, which is considered the best existing scheduling algorithm for HeDCSs [13].

According to the simulation results, it is found that the SNLDD algorithm outperforms and superior the LDCP algorithm in terms of schedule length, speedup, efficiency, complexity and quality parameters which are considered the most important performance measures for evaluating a parallel computer system. Generally, the performance improvement which has been achieved by SNLDD algorithm outperforms the LDCP algorithm by 28.6%.

REFERENCES

[1] Gamal Attiya and Yskandar Hamam, "Task Allocation for Maximizing Reliability of Distributed Systems: A simulated Annealing Approach," J. Parallel Distributed Computer, Vol. 66, pp. 1259-1266, 2006.

[2] S. [11]l, P. Kumar, K. Singh," Dealing With Heterogeneity Through Limited Duplication For Scheduling Precedence Constrained Task Graphs", J. Parallel Distributed Computer, Vol. 65 (4), pp. 479–491, 2005.

[3] R. Bajaj, D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," IEEE Trans. Parallel Distributed Systems, Vol. 15 (2), pp.107-116, 2004.

[4] S. Shivle, R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaan, W. Saylor, D.Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco,"Static mapping of subtasks in a heterogeneous ad hoc grid environment," Proc. of Parallel and Distributed Processing Symposium, Apr. 2004.

[5] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, New York: McGraw-Hill, Inc., 1993.

[6] M. Mezmaz, N. Melab, and E.-G. Talbi, "An Efficient Load Balancing Strategy for Grid-Based Branch and Bound Algorithm, "Parallel Computing, Vol. 33, pp. 302-313. February 2007.

[7] S. Baskiyar, C. Dickinson," Scheduling Directed A-Cyclic Task Graphs On A Bounded Set Of Heterogeneous Processors Using Task

Duplication," J. Parallel Distributed Computer, Vol. 65 (8), pp. 911–921, 2005.

[8] E. Ilavarasan, P. Thambidurai, R. Mahilmannan, "Performance Effective Task Scheduling Algorithm For Heterogeneous Computing System", Proceedings of the Fourth International Symposium on Parallel and Distributed Computing, France, pp. 28–38, 2005.

[9] G.C. Sih, E.A. Lee," A Compile-Time Scheduling Heuristic For Interconnection-Constrained Heterogeneous Processor Architectures," IEEE Trans. Parallel Distributed Systems, Vol. 4 (2), pp. 175–187, 1993.

[10] J. Kim, J. Rho, J.-O. Lee, M.-C. Ko, CPOC: "Effective Static Task Scheduling For Grid Computing," Proceedings of the 2005 International Conference on High Performance Computing and Communications, Italy, pp. 477–486, 2005.

[11] Yu-Kwong KWOK,"High-Performance Algorithms for Compile-Time Scheduling of Parallel Processors," The Hong Kong University of Science and Technology in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science Hong Kong, May 1997.

[12] H. Topcuoglu, S. Hariri, M.Y. Wu," Performance-Effective and Low Complexity Task Scheduling for Heterogeneous Computing," IEEE Trans. Parallel Distributed Systems, Vol. 13 (3), pp.260–274, 2005.

[13] Mohammad I. Daoud, Nawwaf Kharma "A High Performance Algorithm For Static Task Scheduling In Heterogeneous Distributed Computing Systems," IEEE Trans. Parallel Distributed Systems, Vol. 28, pp39-49, July 2007.

[14] W.F. Boyer, G.S. Hura, "Non-Evolutionary Algorithm for Scheduling Dependent Tasks in Distributed Heterogeneous Computing Environments," J. Parallel Distributed Computer, Vol. 65 (9), pp. 1035–1046, 2005.

[15] Kamer Kaya a, Bora Ucar a, Cevdet Aykanat, Murat Ikinci, "Task Assignment In Heterogeneous Computing Systems," J. Parallel Distributed Computers, 66, Vol 8 pp. 32 – 46, 2006.

[16] Yuan Tian, Jarupan Boangoat, Eylem Ekici, and Fu¨sun O¨ zgu¨ner; "Real-Time Task Mapping and Scheduling for Collaborative In-Network Processing in DVS-Enabled Wireless Sensor Networks," IEEE Trans. Parallel Distributed Systems, Rhodes, Greece (IPDPS 2006).

[17] http://www.Kasahara.Elec.Waseda.ac.jp/schedule