Egyptian Informatics Journal (2011) **12**, 219–229

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Optimization procedure for algorithms of task scheduling in high performance heterogeneous distributed computing systems

Nirmeen A. Bahnasawy [b,*], Fatma Omara [a], Magdy A. Koutb [b], Mervat Mosa [b]

[a] *Faculty of Computer and Science, Cairo University, Egypt*
[b] *Faculty of Engineering, Menoufia University, Egypt*

Received 23 May 2011; revised 3 October 2011; accepted 4 October 2011
Available online 25 November 2011

KEYWORDS

Task scheduling;
Directed acyclic graph;
Parallel processing;
Heterogeneous distributed computing systems;
Sleek time

Abstract In distributed computing, the schedule by which tasks are assigned to processors is critical to minimizing the execution time of the application. However, the problem of discovering the schedule that gives the minimum execution time is NP-complete. In this paper, a new task scheduling algorithm called Sorted Nodes in Leveled DAG Division (SNLDD) is introduced and developed for HeDCSs with consider a bounded number of processors. The main principle of the developed algorithm is to divide the Directed Acyclic Graph (DAG) into levels and sort the tasks in each level according to their computation size in descending order. To evaluate the performance of the developed SNLDD algorithm, a comparative study has been done between the developed SNLDD algorithm and the Longest Dynamic Critical Path (LDCP) algorithm which is considered the most efficient existing algorithm. According to the comparative results, it is found that the performance of the developed algorithm provides better performance than the LDCP algorithm in terms of speedup, efficiency, complexity, and quality. Also, a new procedure called Superior Performance Optimization Procedure (SPOP) has been introduced and implemented in the developed SNLDD

* Corresponding author.
E-mail address: nirmeen_a_wahab@hotmail.com (N.A. Bahnasawy).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.
doi:10.1016/j.eij.2011.10.001

Production and hosting by Elsevier

algorithm and the LDCP algorithm to minimize the sleek time of the processors in the system. Again, the performance of the SNLDD algorithm outperforms the existing LDCP algorithm after adding the SPOP procedure.

1. Introduction

A Distributed Computing System, 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 [4,6]. Static task scheduling takes place during compile time before running the parallel application [2,3,7]. 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 [5,9]. The static scheduling is addressed in this paper. Moreover, in typical scientific and engineering applications, compilation time, including the static scheduling time, is much lower than the run time [5]. 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 [10].

Examples of existing task scheduling algorithms are; Heterogeneous Earliest Execution time (HEFT) [11], Critical Path On a Processor (CPOP) [6], Critical Path On a Cluster (CPOC) [6], Dynamic Level Scheduling (DLS) [5], Modified Critical Path (MCP) [5], Mapping Heuristic (MH) [11] and Dynamic Critical Path (DCP) [4]. Topcuoglu et al. [11] have presented a 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 are achieved comparable results. Also, the performance of the HEFT and Heterogeneous N-predecessor Decisive Path (HNPD) algorithms is compared in [6], 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 large number of processors than the HEFT algorithm to achieve the same schedule length [6].

Recently, a new algorithm called Longest Dynamic Critical Path (LDCP) has been introduced [6]. 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 [11] and the DLS [5] algorithms.

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 principle of the developed algorithm is to divide the Directed Acyclic Graph (DAG) into levels and sort the tasks in each level according to their computation size in descending order. So, to evaluate the SLNDD algorithm, a comparative study has been done between the developed SNLDD algorithm and the LDCP algorithm. According to the comparative results, the SNLDD algorithm outperforms the LDCP algorithm in terms of schedule length, speedup, efficiency, and quality of system behavior.

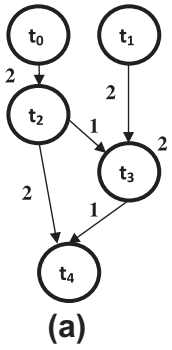
The LDCP algorithm and the developed SNLDD algorithm have been modified by introducing a new procedure called Superior Performance Optimization Procedure (SPOP) to minimize the sleek time of the processors by using the idle time of the processors during assigning tasks to generate high-quality task schedules, and minimize the schedule length. Again, the two modified algorithms have been compared, and the modified SNLDD algorithm has verified better performance than the modified LDCP algorithm.

The remainder of this paper is organized as follows; in Section 2, the task scheduling problem and some necessary terms are defined. In Section 3, the LDCP algorithm for task scheduling in the HeDCSs is introduced. The new developed algorithm SNLDD is introduced in Section 4. Section 5 represents the new procedure SPOP which is applied on both LDCP and SNLDD algorithms. The comparative study between the developed algorithm and the existing LDCP algorithm is presented in Section 6, and finally, conclusions are given in Section 7.

2. 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 T represents a task in the parallel application, and each edge (t_i, t_j) E

represents a precedence constraint and a communication message between tasks t_i and t_j. If (t_i, t_j) E, then the execution of t_j T cannot be started before t_i 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 [5,11]. 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 tasks n in processors m. Each element $w_{i,j}$ 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 [3,12,13]. 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 [14]. The data transfer rate between any two processors on the network is assumed to be fixed and constant [3,5]. 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 [14,15]. 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 [3,9,11]. An example of a DAG of a parallel application and a computation cost matrix with two processors is shown in Fig. 1.



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

(b)

Figure 1 An example of a DAG and computation cost.

3. The Longest Dynamic Critical Path (LDCP) algorithm

The most recent algorithm called Longest Dynamic Critical Path (LDCP) algorithm has been introduced by Daoud et al. [12]. According to the LDCP algorithm (see Fig. 2), each scheduling step consists of three phases; task selection, processor selection and status update.

3.1. 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 directed acyclic graph that corresponds to processor p_j, called DAGP_j, is constructed using the structure of the DAG, with sizes of tasks set to their computation costs on processor p_j.

3.2. Processor selection phase

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

3.3. 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 t_i on processor p_j means that the computation cost of t_i is no longer unknown. Hence, the sizes of the nodes that identify t_i are set to the computation cost of t_i on p_j on all DAGPs. Moreover, a value of zero is assigned to all edges that extend between the nodes that identify t_i and the nodes that identify its parents that are scheduled on processor 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 t_i into processor p_j will result in new execution constraints.

4. 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:

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

where $S_j(n_i)$ is the computation size of the specified task (n_i) in the j level where $1 \leqslant j \leqslant R$, R is the total number of levels and $1 \leqslant i \leqslant T$, where T is the total number of tasks. The first part

```
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 execution time of the selected task on every processor in the system
  Find the selected processor that minimizes the execution 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
```

Figure 2 Longest Dynamic Critical Path (LDCP) algorithm.

of the equation computes the execution time of task 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 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 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 [6]. Therefore, the developed SNLDD algorithm is based on dividing the DAG into levels and the tasks in each level are assigned 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 the developed (SNLDD) algorithm, this leads to simplify the classification of tasks according to the priority, which is considered more efficient than the LDCP algorithm because the time for choosing the returned task to be assigned will be computed in each step. A high quality schedule is created 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 the developed SNLDD algorithm will reduce the total required time. In addition, sorting the computation sizes of tasks according to their computation sizes in descending ordering leads to get red of the heaviest tasks first to reduce the complicated dependency of childs on 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, the developed SNLDD algorithm is become more efficient than 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 the 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 schedule 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 into levels and tasks in each level are assigned to processors.
- On the other hand, the authors in [6] have proved that their LDCP algorithm is considered more efficient than those the HEFT and LCD algorithms and, in the same time, the developed SNLDD algorithm is considered more efficient than the LDCP algorithm, then the 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 [6], clustering algorithms [2] and hierarchy as tree algorithms [9], are verified in the algorithm, this means that; SNLDD algorithm is considered as a collection of a lot of algorithms.

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. We can conclude that the time complexity of SNLDD algorithm is $\Theta(m \times n\hat{}2)$ while the time complexity of LDCP algorithm is $\Theta(m \times n\hat{}3)$, where m is the number of processors, and n is the number of tasks.

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

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

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 execution 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 $S_j(n_i)$ *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

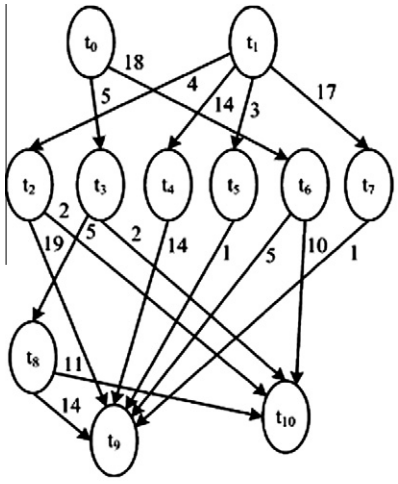
Figure 3 The pseudo code of developed SNLDD algorithm.

Example 1. By considering the application DAG and the computation cost matrix in Fig. 1. The schedule length according to the SNLDD algorithm is 23 units; whenever the LDCP 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 LDCP algorithm and SNLDD algorithm are shown in Figs. 5 and 6 respectively. The schedule generated by SNLDD algorithm has length of 63, while the schedule length generated by LDCP algorithm is 64. So, the SNLDD algorithm has shorter execution length than 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 the LDCP algorithm.

5. Implementing the Superior Performance Optimization Procedure (SPOP)

A new performance optimization procedure (SPOP) has been added to the developed SNLDD algorithm and the LDCP algorithm. The SPOP is based on the availability of selection the task when it is assigned to a processor that minimizes its finish execution time using the insertion based scheduling policy [6]. When a processor p_j is assigned a task t_i, the insertion-based scheduling policy considers all possible idle time slots on p_j to find a time slot of equal or greater length than the execution time of t_i. This must be done without violating the precedence constraints among tasks. An idle time slot on processor p_j is defined as the time space between the finish execution time and start execution time of two consecutively scheduled tasks on p_j. The search starts from a time equal to the ready time



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

(b)

Figure 4 (a) A sample DAG and (b) computation cost matrix.

of t_i on p_j, and proceeds until it finds the first idle time slot with the sufficient length for the computation cost of t_i on p_j. If no idle time slot is found, the selected task is inserted after the last scheduled task on p_j.

6. Performance evaluations

6.1. Comparative results without adding SPOP procedure

To evaluate the performance of the 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 MHz.

Empirical results on benchmark [16] task graphs of several well-known parallel applications, which have been validated by the use of non-parametric statistical tests, show that the developed algorithm significantly outperforms several related algorithms in terms of the schedule quality. Further experiments are carried out to reveal that the developed algorithm is able to maintain high performance within a wide range of parameter settings.

A comparative study has been done between the developed SNLDD algorithm and the LDCP algorithm. Two sets of parallel application graphs, which correspond to both random application DAGs and DAGs of parallel numerical applications are used. Also, the Standard benchmark Task Graph Set (STG) has been used [16].

Some parameters have been determined, these parameters are:

- *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.
- Using four different numbers of processors varying from 2, 4, 8, and16 processors. For each number of processors, five different DAG sizes varying from 20 to 100 nodes with an increment of 20 are used.

The results of the comparative study between the developed SNLDD algorithm and the LDCP using task graphs of 20 to 100 nodes and processor graphs of 2, 4, 8, and 16 nodes are shown in Figs. 7–10. According to the results, the schedule length, the running time of program, and the system required memory is decreased in the developed SNLDD algorithm and then memory efficiency increases. So, the developed SNLDD algorithm is more efficient than the LDCP algorithm.

The performance of the developed SNLDD algorithm and the LDCP algorithm will be reported using the performance criteria:

6.2. Speedup

The speedup of a schedule is defined as the ratio of the schedule length obtained by assigning all task to the fastest proces-

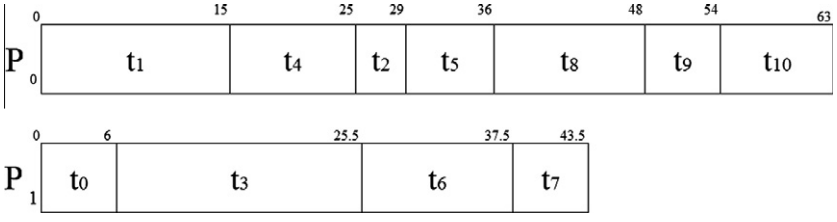


Figure 5 The schedule generated by the developed algorithm.

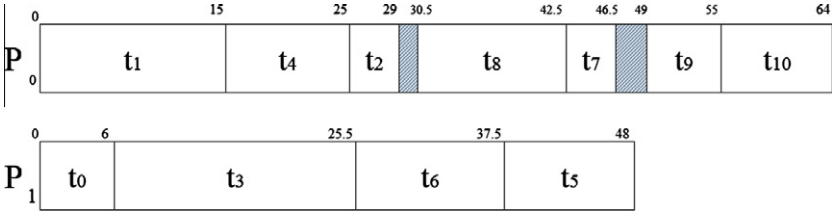


Figure 6 The schedule generated by the LDCP algorithm.

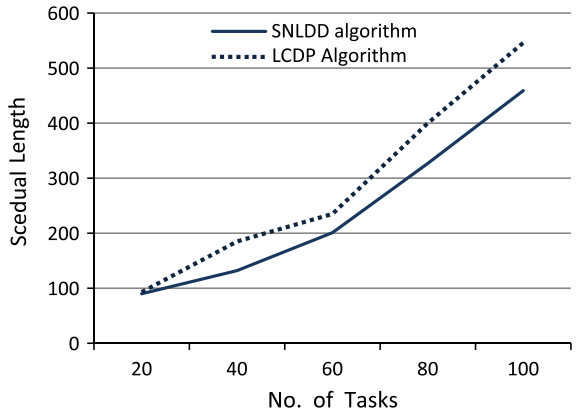


Figure 7 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 2 processors.

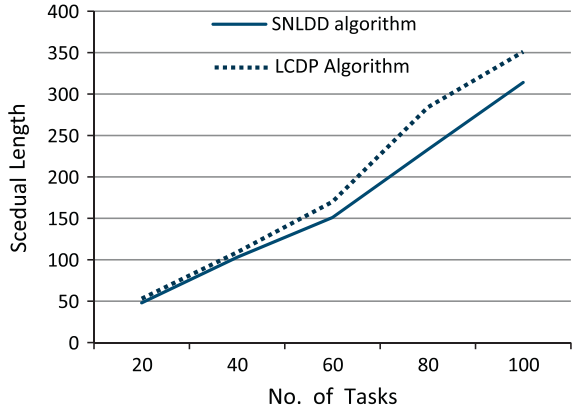


Figure 8 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 4 processors.

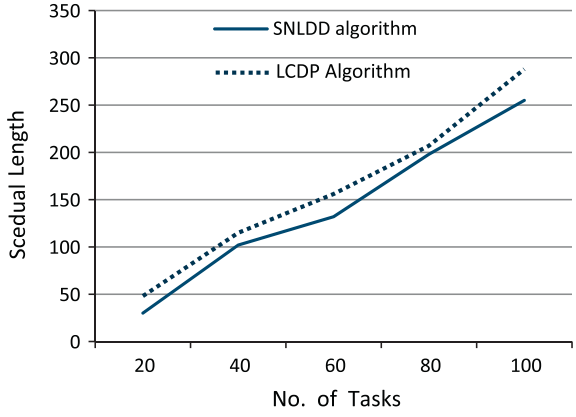


Figure 9 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 8 processors.

sor, to the parallel execution time of the task schedule [6]. Linear speedup means that the value of speedup increases as the number of processors in the parallel system increases [5].

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.

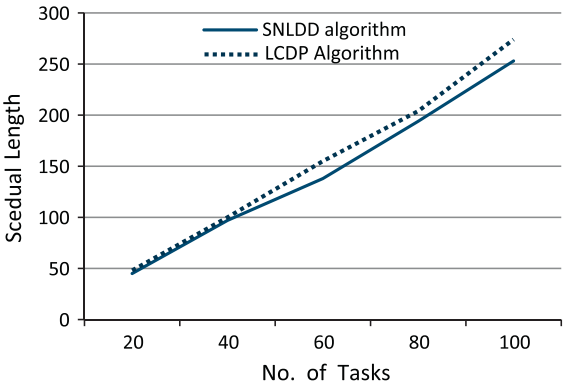


Figure 10 The schedule length generated by the SNLDD algorithm and LDCP algorithm on 16 processors.

Speedup can be estimated as

$$S(m) = T(1)/T(m) \quad \leqslant S(m) < m \tag{2}$$

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

The results of the comparative study according to the speedup are shown in Figs. 11–15.

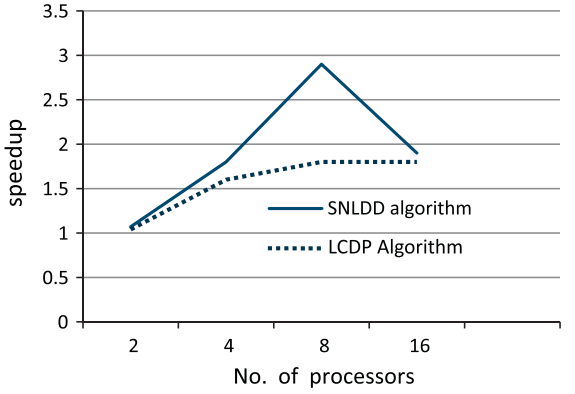


Figure 11 The speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 20 tasks.

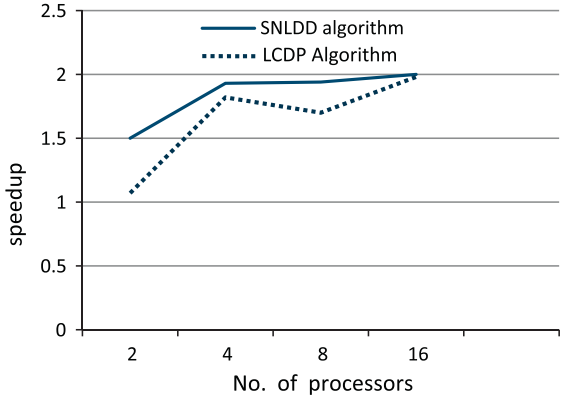


Figure 12 The speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 40 tasks.

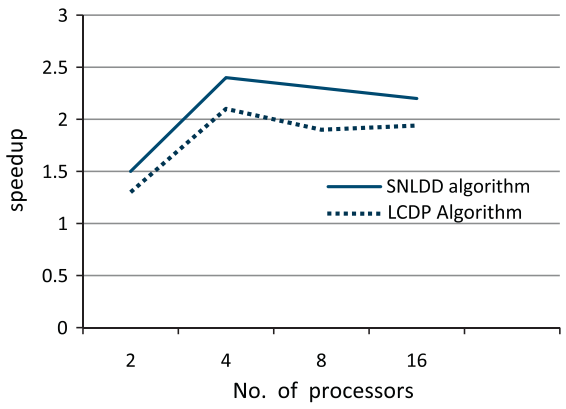


Figure 13 The speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 60 tasks.

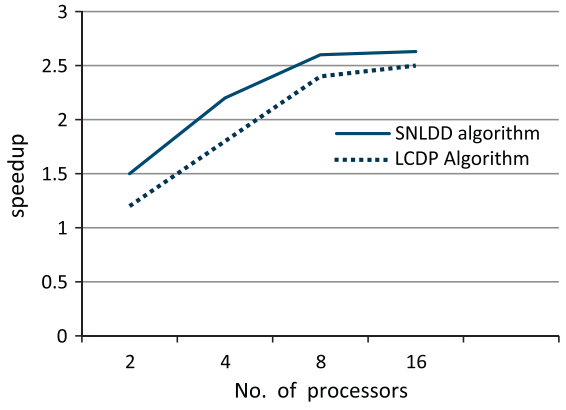


Figure 14 The speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 80 tasks.

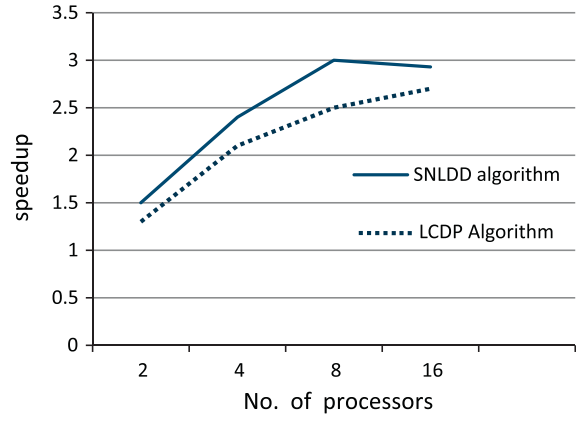


Figure 15 The speedup of two algorithms in case of 2, 4, 8, 16 processors with DAG of 100 tasks.

6.3. Efficiency

The efficiency of the parallel computers is an indication to what percentage of a processors time is being spent in useful computation [5]. The efficiency of a parallel computer containing m processors can be defined as:

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

Maximum efficiency $E(m) = 1$ is achieved when all the processors are fully utilized during all time periods of the program execution. Quality of parallelism is directly proportional to the speedup and efficiency [5]. The quality is always supper-bound by the speedup.

The results of the comparative study according to the efficiency are shown in Figs. 16–20.

According to the results in Figs. 7–20, it is clear that the developed SNLDD algorithm is always outperformed the LDCP algorithm in terms of schedule length conditions, speedup conditions, and efficiency conditions. These results show the important performance measures of evaluating parallel system [6].

6.4. Comparative results with adding SPOP procedure

The performance of the SNLDD algorithm and the SNLDD algorithm after adding SPOP procedure has been evaluated. Figs. 21 and 22 represent the comparative results of the LDCP algorithm before and after adding SPOP procedure. According to the results in Figs. 21 and 22, the performance of the modified LDCP algorithm does not improved, while the perfor-

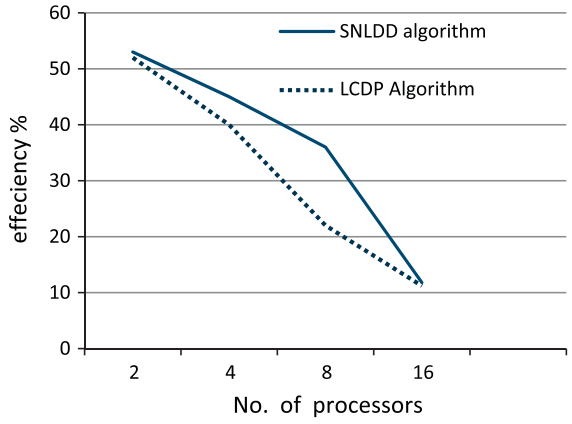


Figure 16 Efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG of 20 tasks.

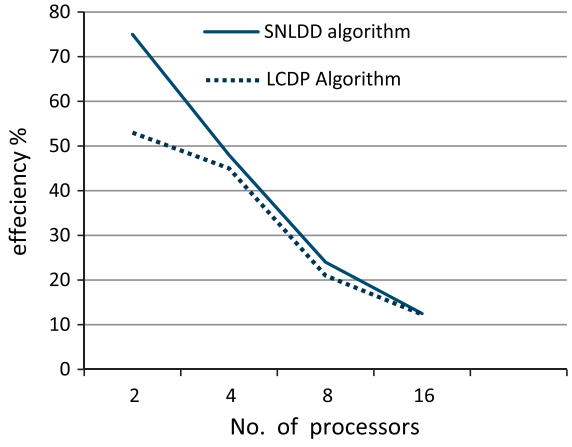


Figure 17 Efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG of 40 tasks.

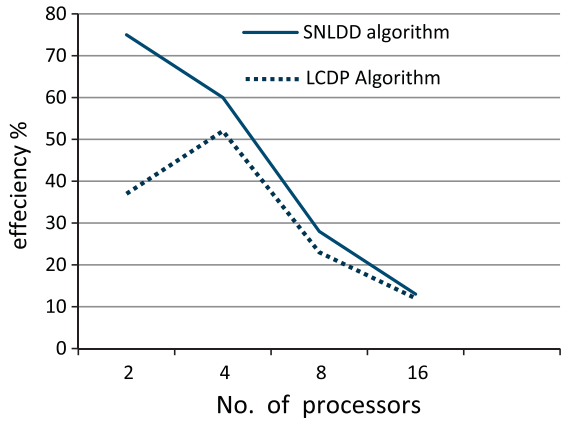


Figure 18 Efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG of 60 tasks.

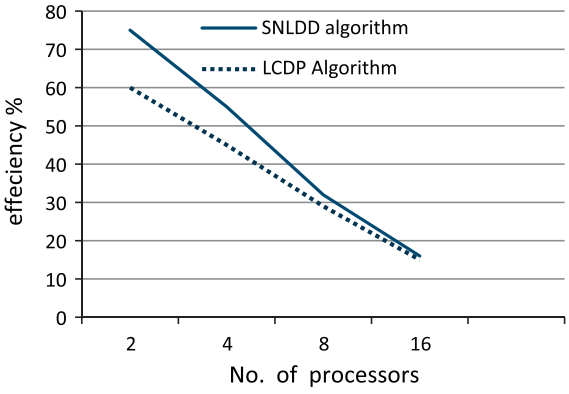


Figure 19 Efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG of 80 tasks.

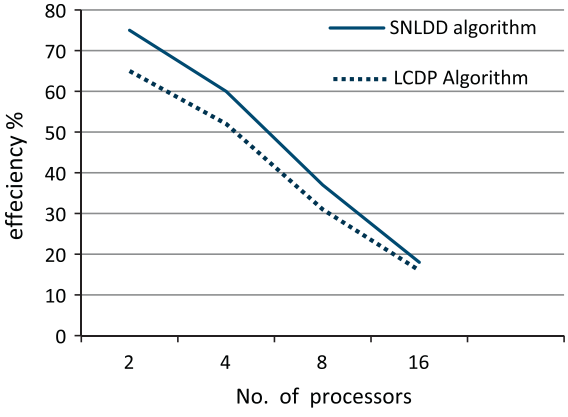


Figure 20 Efficiency curves of two algorithms in case of 2, 4, 8, 16 processors with DAG of 100 tasks.

mance of the modified SNLDD algorithm has been increased by 7.5%according to the schedule length parameter.

Figs. 23–26 represent the comparative results of the SNLDD algorithm and the LDCP algorithm after adding SPOP procedure using task graphs of 20–100 nodes and processor graphs of 2, 4, 8, and 16 nodes. According to the results

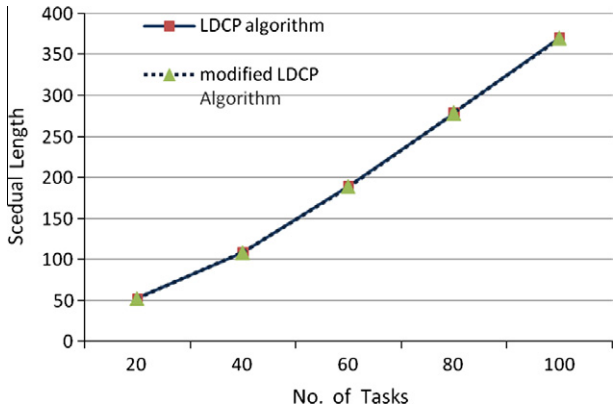


Figure 21 The schedule length generated by the modified LDCP algorithm and LDCP algorithm on 4 processors.

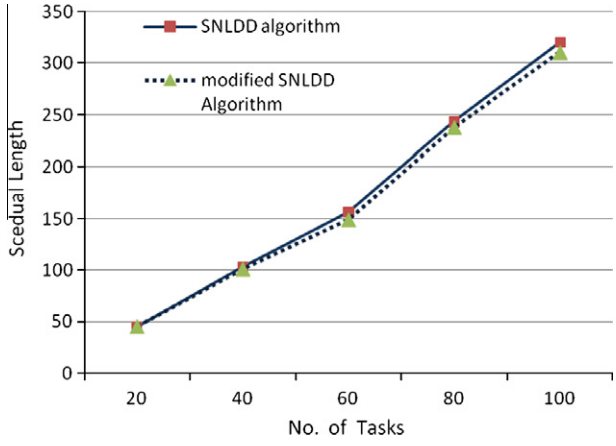


Figure 22 The schedule length generated by the modified SNLDD algorithm and SNLDD algorithm on 4 processors.

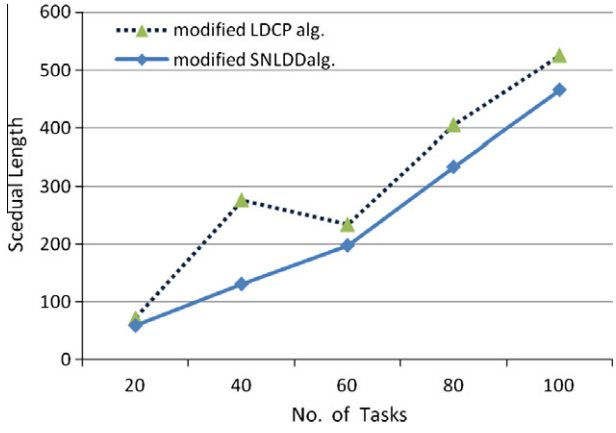


Figure 23 The schedule length generated by the modified LDCP algorithm and SNLDD algorithm on 2 processors.

in Figs. 23–26, it is shown that the number of schedule length, the running time of program, and the system required memory are decreased; then, memory efficiency is increased by using the modified SNLDD algorithm.

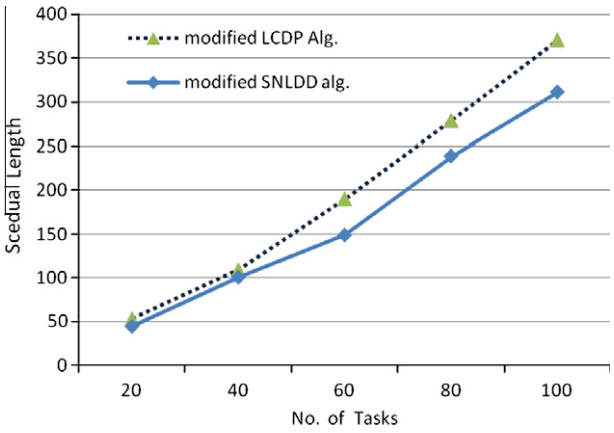


Figure 24 The schedule length generated by the modified LDCP algorithm and SNLDD algorithm on 4 processors.

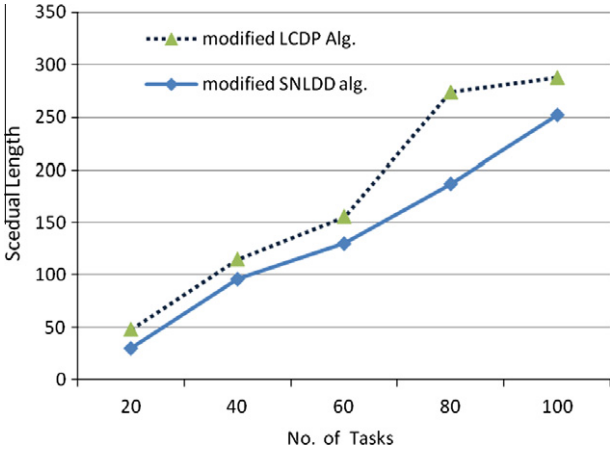


Figure 25 The schedule length generated by the modified LDCP algorithm and SNLDD algorithm on 8 processors.

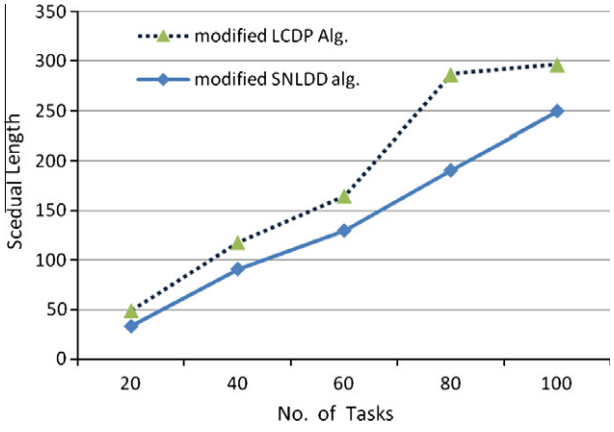


Figure 26 The schedule length generated by the modified LDCP algorithm and SNLDD algorithm on 16 processors.

The modified SNLDD algorithm guarantees smart allocations in acceptable computation time, and overcomes the low solutions quality that may be obtained by using modified LDCP. It also overcomes the computational time complexity of the modified LDCP algorithm. Furthermore, the developed algorithm improves the efficiency of using the system memory.

From Figs. 23–26, the modification SNLDD algorithm is considered better than the modification LDCP algorithm under schedule length conditions, for 2, 4, 8, 16 processors. So the algorithm is most efficient than other. It also overcomes the computational time complexity of this algorithm. Furthermore, the developed algorithm improves the efficiency of using the system memory.

One of the major advantages of the algorithm over LDCP is that balancing of workload of the system among the processors can improve system performance.

7. Conclusions

In this paper, a new scheduling algorithm is presented for heterogeneous distributed computing systems HeDCSs. According to this algorithm, the DAG is divided into levels according to the priority of precedence relations, and tasks are sorted in each level in descending order, and then the task is chosen from that level according to its the computation size, to accurately identify the priorities of task in HeDCSs.

The performance of the developed SNLDD algorithm is compared to, which is considered the best existing scheduling algorithm for HeDCSs because is outperformed both the HEFT and the DLS algorithms.

The comparative study between the developed SNLDD algorithm and the LDCP algorithm has been done using standard application DAGs. It is found that the developed SNLDD algorithm outperforms and superior the LDCP algorithm in terms of schedule length, speedup, efficiency, complexity and quality parameters which are considered most important performance measures for evaluating a parallel computer system.

Also, the developed SNLDD algorithm and the LDCP algorithm have been modified by adding Superior Performance Optimization Procedure (SPOP) to minimize the sleek time in the processors, and then minimize the execution length. According to the simulation, it is found that the developed SNLDD algorithm significantly outperforms and superior the LDCP algorithm in terms of schedule length, speedup, and, efficiency of running time of programs and memory quality parameters which are most important performance measures of evaluating a parallel computer system.

Generally, the performance improvement ratio of the SNLDD algorithm outperforms the LDCP algorithm by 16% according to schedule length parameter, and 21.3% according to speedup parameter, but after adding (SPOP) procedure the performance improvement ratio of the SNLDD algorithm outperforms the LDCP algorithm by 22% according to schedule length parameter, and 28.6% according to speedup parameter.

References

[1] Attiya Gamal, Hamam Yskandar. Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. J Parallel Distrib Comput 2006;66:1259–66.

[2] Bansal S, Kumar P, Singh K. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J Parallel Distrib Comput 2005;65(4):479–91.

[3] Bajaj R, Agrawal DP. Improving scheduling of tasks in a heterogeneous environment. IEEE Trans Parallel Distrib Syst 2004;15(2):107–16.

[4] Shivle S, Castain R, Siegel HJ, Maciejewski AA, Banka T, Chindam K, et al. Static mapping of subtasks in a heterogeneous ad hoc grid environment. In: Proc of parallel and distributed processing symposium; April 2004.
[5] Hwang K. Advanced computer architecture: parallelism, scalability, programmability. New York: McGraw-Hill, Inc; 1993.
[6] Mezmaz M, Melab N, Talbi E-G. An efficient load balancing strategy for grid-based branch and bound algorithm. Parallel Comput 2007;3:302–13.
[7] Boyer WF, Hura GS. Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. J Parallel Distrib Comput 2005;65(9):10350–1046.
[8] Zelkowitz Marvin. ''Advances in computers'' Elsevier journal. Computer Science, vol. 78; 2010. p. 368.
[9] Kim J, Rho J, Lee J-O, Ko M-C. CPOC: effective static task scheduling for grid computing. In: Proceedings of the 2005 international conference on high performance computing and communications, Italy; 2005. p. 477–86.
[10] Kwok Yu-Kwong. 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.
[11] Topcuoglu H, Hariri S, Wu MY. Performance-effective and low complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 2005;13(3):260–74.
[12] Daoud Mohammad I, Kharma Nawwaf. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. IEEE Trans Parallel Distrib Syst 2007;28:39–49.
[13] Tian Yuan, Boangoat Jarupan, Ekici Eylem, Özgüner Füsun. 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).
[14] Ilavarasan E, Thambidurai P, Mahilmannan R. Performance effective task scheduling algorithm for heterogeneous computing system. In: Proceedings of the fourth international symposium on parallel and distributed computing, France; 2005. p. 28–38.
[15] Kaya Kamer, Ucar Bora, Aykanat Cevdet, Ikinci Murat. Task assignment in heterogeneous computing systems. J Parallel Distrib Comput 2006;66(8):32–46.
[16] http://www.Kasahara.Elec.Waseda.ac.jp/schedule/.