An Independent Task Scheduling Algorithm in Heterogeneous Multi-core Processor Environment

Lindong Liu[1,2] , Deyu Qi[1]
1.Research Institute of Computer Systems at South China University of Technology, Guangzhou,China
2.Department of Computer Science, Guangdong University of Education,Guangzhou,China
hongox@163.com

***Abstract*—Multi-core processor architecture is increasingly being used in high-performance computing.Task scheduling problem for performance heterogeneous multi-core processor is a well-known NP-complete problem. Based on heterogeneous multi-core processor environment, independent online task scheduling is studied in this paper. In this paper, a scheduling model of heterogeneous multi-core processors based on weighted earliest finish time is proposed, and the wEFT algorithm is proposed. The wEFT algorithm selects the processor core to execute task with the minimum earliest completion time or the minimum weighted links of every processor core. Experiments show that compared with RR, wRR, LC and wLC algorithms under the environment of different number of tasks and processor cores, the makespan for the complete schedule of all tasks and average waiting time in wEFT algorithm is superior to other four algorithms.**

Keywords—Processor core;task scheduling;independent task; makespan;average waiting time

I. INTRODUCTION

Multi-core processor[1-4] integrates multiple fully functional processor cores in the same chip, and each processor core can be homogenous or heterogeneous. Multi-core processor architecture has become the mainstream microprocessor architecture and is widely used in high-performance computing environment. Heterogeneous multi-core processors can be divided into two types,they are functional heterogeneous multi-core and performance heterogeneous multi-core. Functional heterogeneous multi-core processor means different processor cores process different instruction sets, and performance heterogeneous multi-core processor means that all of the processor cores support the same instruction sets, but they differ in their performance.Performance heterogeneous multi-core processor have become a hot spot in academic and industrial research.

In multi-core processor environment,task scheduling depends on whether there are dependencies between the tasks that are scheduled. It can be divided into independent task scheduling and related task scheduling.Related task scheduling is often referred to as dependent task scheduling[5-6]. There is no dependency relationship and data communication among tasks in independent task scheduling[7-9].The tasks in the dependent task scheduling have some dependence,and there is data communication among tasks. Independent task scheduling is a study of how independent tasks set are scheduled on a heterogeneous multi-core processors set, the best performance of the whole scheduling system or a specific scheduling goal is required. In order to allocate the most appropriate processor core for the selected task, various constraints of the task are considered of the task scheduling strategy. Independent task scheduling is also divided into two strategies: online task scheduling and batch task scheduling. The common online task scheduling algorithms include Round-Robin scheduling algorithm, weighted Round-Robin scheduling algorithm, minimum Link-Counter scheduling algorithm and weighted minimum Link-Counter scheduling algorithm. The batch task scheduling include Min-Min[10] algorithm、Min-Max[10] algorithm, Sufferage algorithm, Genetic algorithm(GA) and Particle Swarm Optimization algorithm(PSO).

II. ONLINE TASK SCHEDULING ALGORITHMS

The common online task scheduling algorithms include Round-Robin scheduling algorithm[11], weighted Round-Robin scheduling algorithm, minimum Link-Counter scheduling algorithm and weighted minimum Link-Counter scheduling algorithm. For example ,there are 4 processor cores in the heterogeneous multi-core processors environment, we introduce the basic idea and task scheduling process of 4 online task scheduling algorithms respectively.

A. Round-Robin scheduling algorithm

Round-Robin scheduling algorithm is an online scheduling algorithm. Its basic idea is to schedule a selected task to different processor cores in turn. The advantages of the algorithm are simple and easy, but the disadvantage is that the processor core with heterogeneous performance is inefficient. As shown in Fig. 1, 4 task queues Q_1,Q_2,Q_3 and Q_4 are used to store the task scheduling queues of 4 processor cores(c_0,c_1,c_2,c_3). Scheduling tasks in sequence from c_0 firstly, and when scheduling to c_3, the next processor core is c_0. In the algorithm, the loop scheduling is handled by (*core_ID*+1)%*total_core*, where *core_ID* is the ID of current shceduled processor core and *total_core* is the number of the processor cores.

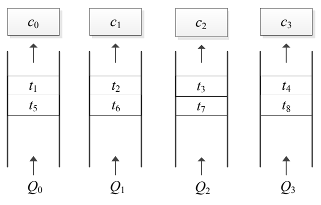

Fig. 1. RR task scheduling algorithm

B. Weighted Round-Robin scheduling algorithm

In order to overcome the disadvantages in the rotation scheduling algorithm, the computing performance of every processor core is represented by the corresponding weights w_0,w_1,w_2 and w_3, and the processor core with large weights will be assigned more task requests. Finally, the number of links of each processor core tends to the weight proportion corresponding to each node. The task scheduling algorithm is shown in Fig. 2. According to the idea of weighted Round-Robin scheduling algorithm, the number of tasks allocated on each processor core is T_j, $T_j = (w_j * total\_task) / \sum_{i=0}^{4} w_i$, variable j is the number of a processor core between 0 and 3, and variable *total_task* is the number of tasks scheduled. Obviously T_j is not an integer.

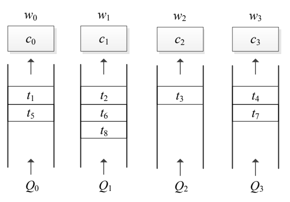

Fig. 2. wRR task scheduling algorithm

C. Minimum Link-Counter scheduling algorithm

The minimum Link-Counter algorithm is an online task scheduling algorithm, as shown in Fig. 3. The algorithm registers the number of scheduled tasks in each processor core by a load balancer. The number of scheduled tasks in each processor core is ts_0,ts_1,ts_2 and ts_3 respectively. When a task arrives, the load balancer assigns the early arrival time to the least connected processor core. Obviously, when the processor core has not been scheduled, the minimum Link-Counter scheduling algorithm is similar to the Round-Robin scheduling algorithm. In order to solve this problem, if there are more than two processor cores with minimum scheduled tasks, select the processor core with the minimum execution time to be scheduled.

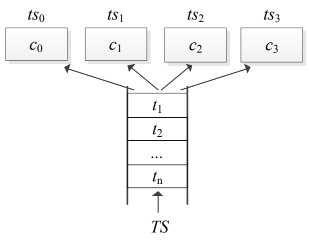

Fig. 3. LC task scheduling algorithm

D. Weighted minimum Link-Counter scheduling algorithm

Because LC algorithm does not consider the inconsistent processing power of the processor core, the weighted minimum Link-Counter algorithm can overcome this problem. The weighted minimum Link-Counter algorithm uses the corresponding weights to represent the computing power of each processor core, and assigns task to the processor core with the smallest connections to the weight ratio, as shown in Fig. 4. The number of tasks assigned on each processor core is T_j, T_j= $((ts_j / w_j)*total\_task)/\sum_{i=0}^{4}(ts_j / w_j)$,variable j is the number of a processor core between 0 and 3, and variable *total_task* is the number of tasks scheduled.

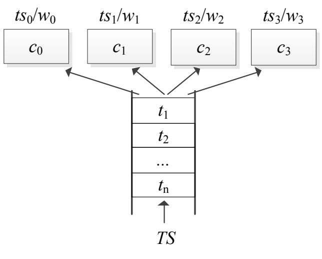

Fig.4. wLC task scheduling algorithm

III. TASK SCHEDULING MODEL AND ALGORITHM

A. Definitions

Definition 1. The task set T refers to the set of scheduled tasks, including the set of n tasks $\{t_1,t_2,\ldots,t_n\}$,each task t_i has the corresponding arrival time $AT(t_i)$.

Definition 2. The processor core set C, including the set of m processor cores $\{c_0,c_1,\ldots,c_{m-1}\}$,each processor core c_j has a corresponding weight of w_j.

Definition 3. Scheduling time $TC(t_i,c_j)$ refers to the execution time of task t_i on processor core c_j.

Definition 4. Arrival time of task t_i $AT(t_i)$ refers to the time that each task reaches the set of processor cores.

Definition 5. Makespan or scheduling length represents the completion time of the last task in the task set T with a certain scheduling algorithm based on the task set T, as shown in (1). $AFT(t_i)$ represents the actual finish time of task t_i (Actual Finish Time). Makespan is the maximum value of the actual finish time in all scheduled tasks.

$$Makespan = \max\{AFT(t_i)\} \quad (1)$$

Definition 6. Earliest start time $EST(t_i,c_j)$ refers to the earliest start time of task t_i on processor core c_j. $EST(t_i,c_j)$ take the larger value between $AT(t_i)$ and $EFT(c_j)$, as shown in (2).

$$EST(t_i,c_j) = \begin{cases} AT(t_i) \\ EFT(c_j) \end{cases} \quad (2)$$

Definition 7. Earliest finish time $EFT(t_i,c_j)$ refers to the earliest finish time of task t_i on processor core c_j. The value of earliest finish time is equal to the earliest start time of the task t_i on the processor core c_j, plus the scheduling time of t_i on the processorc_j, as shown in (3).

$$EFT(t_i,c_j) = EST(t_i,c_j) + ET(t_i,c_j) \quad (3)$$

Definition 8. Average waiting time AWT. Suppose the start time of the task t_i is ST_i, and the arrival time of task is AT_i during the scheduling process, $AWT=\sum_{j=1}^{n}(ST_j - AT_j)/n$.

B. Task scheduling model

wEFT task scheduling model is as shown in Fig. 5. The task scheduling model mainly include three modules: earliest finish time module, weighted link counter module and task scheduling module. The earliest finish time module calculates the minimum earliest finish time based on the task set T and the processor core set C. Weighted link counter module calculates the minimum value of the number and weight of the task being scheduled on the process core set. Based on the results of the first two modules, task scheduling module chooses the appropriate processor core from the core set C to schedule the corresponding task.

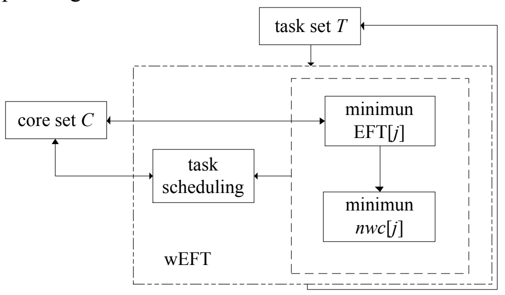

Fig. 5. wEFT task scheduling model

C. Task scheduling algorithm

The wEFT algorithm takes the earliest finish time (EFT) of tasks on each processor core and the weighted minimum link counter as the main parameters of task scheduling. Firstly, the algorithm finds the minimum earliest finish time of the scheduled task on each processor core. Secondly, if there are multiple processors cores with the same earlisest finish time, we further find the processor core of the weighted minimum link counter, and schedule the task to the processor core. Otherwise, the task is directly scheduled to the process core with the smallest earliest finish time. wEFT algorithm is shown in Table I. *n* tasks to be scheduled with *m* processor cores, and the time complexity of wEFT algorithm is O(*n***m*).

TABLE I. wEFT ALGORITHM

```
Input：task set $TS, w[j], TC[][]$;
Output：$tc[s]$;
int $i, ts[], nwc[], total\_core, counter, core\_ID$;
double $min\_EFT$,
select a task $t_i$ with the minimum arrive time from $TS$;
for(int $j$=0;$j<total\_core$;$j$++){
    $ts[j]$=0;
    read $w[j]$ of every process core;
}
while ($TS$ is not empty){
  for(int $j$=0;$j<total\_core$;$j$++){
      $nwc[j]=ts[j]/w[j]$;
      calculate EFT[$j$];
  }
  find minimum EFT[$j$] $min\_EFT$;
  if $min\_EFT$=EFT[$j$]
        $counter$++;
  if $counter$>1{
        select minimum $nwc[j]$;
        $core\_ID=j$;
  }
  assign task $t_i$ to core $core\_ID$;
  $tc[core\_ID]$=1;
  $ts[core\_ID]$++;
}
```

IV. ANALYSIS OF TASK SCHEDULING PROCESS

(1) Initialization of task set and processor core set

It is assumed that the task set *T* includes 8 tasks, that is n=8. There are 4 processor cores in core set *C*, and those processor cores are performance heterogeneity. The weights of the 4 processor cores are as follows: w_0=1.0,w_1=4.0,w_2=2.0,w_3=1.33. The greater the weight value, the stronger the computing power of the processor core.On contrary, the smaller the weight value, the weaker the computing power of the processor core. The scheduling time of the task set on the processor core set is shown in Table II.

TABLE II. THE SCHEDULING TIME OF THE TASK SET ON THE PROCESSOR CORE SET

task	c_0	c_1	c_2	c_3
t_1	200	50	100	150
t_2	40	11	21	32
t_3	21	5	11	16
t_4	35	9	18	27
t_5	60	14	31	43
t_6	100	26	49	77
t_7	120	31	62	90
t_8	150	38	75	112
t_9	90	22	44	68
t_{10}	110	28	56	85

(2)Arrival time of every task $AT(t_i)$.

The arrival time of each task is generated through a random program, and the arrival time of each task is taken as an integer. Arrival time of each task as shown in Table III.

TABLE III. ARRIVAL TIME OF EACH TASK

task	arrival time
t_1	20
t_3	0
t_4	15
t_5	11
t_6	35
t_7	21
t_8	22
t_9	23

(3) Scheduling tasks

RR, wRR, LC, wLC, and wEFT algorithms are used to schedule the task set to generate the scheduling relationship between the task set *T* and the processor core set *C*, as shown in Fig. 6, Fig. 7 and Fig. 8.

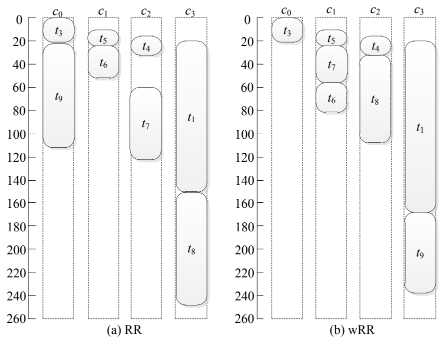

Fig.6. (a) RR algorithm task scheduling diagram (b) wRR algorithm task scheduling diagram

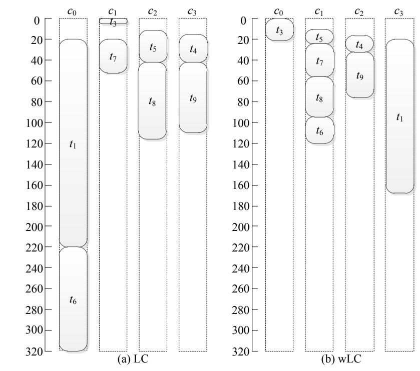

Fig. 7. (a)LC algorithm task scheduling diagram (b) wLC algorithm task scheduling diagram

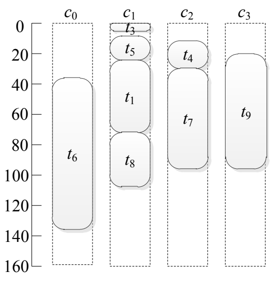

Fig. 8. wEFT algorithm task scheduling diagram

(4)Calculating the *Makespan* and *AWT* values

According to the scheduling relationship between the task set and the processor core set, as well as the Definition 5, the Definition 8, the corresponding *Makespan* and *AWT* values of the 5 algorithms are shown in Table IV.

TABLE IV. COMPARISON BETWEEN *MAKESPAN* AND *AWT*

Algorithms	RR	wRR	LC	wLC	wEFT

Makespan	247	238	320	170	135
AWT	18.5	22.88	28	13.8	8.75

V. SIMULATION EXPERIMENT AND RESULT ANALYSIS

A. Experimental purpose

In order to verify we proposed wEFT algorithm, the performance of the wEFT algorithm is compared with the existing task scheduling algorithms RR, wRR, LC and wLC algorithms under the same experimental conditions. The Makespan and the AWT of task set are compared.

B. Simulated environment

Based on the simulator toolkit provided by SimGrid[12-14], a simulation environment of heterogeneous multi-core processors is built in this paper.

(1)Interprocessor cores are interconnected through high speed networks.

(2)Each processor core can perform task execution at the same time and communicate with other processor cores without competition.

(3)Every task is not preempted on the processor core.

(4)The processor cores are heterogeneous,there are some difference between the same task is scheduled on the different processor core.

The computer used in the experiment is configured as:Intel Core i5-3210M@2.5GHz dual core pen processor, 8GB memory. The number of the processor cores in the experiment is 4 and 6.

C. Test Data Set

The input data of wEFT algorithm include task scheduling time two-dimensional array, task set and weights on processor cores. The task scheduling time array includes the execution time of 10 tasks and 4 processor cores, another case is 10 tasks and 6 processor cores. The number of tasks in the test task set starts from 50, increments 50 tasks each time until to 500 tasks, and the arrival time of each task $AT(t_i)$ is generated by a random program.

D. Analysis of Experimental Results

(1)Result analysis under 4 processor cores

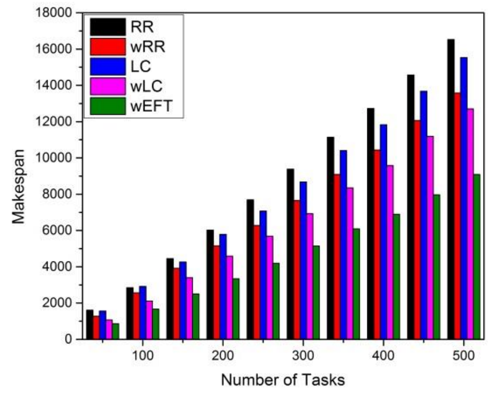

Fig. 9. comparison of *Makespan*

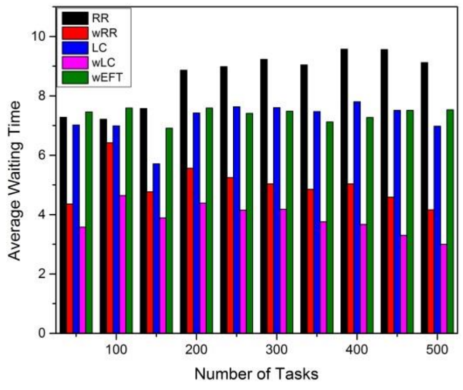

Fig. 10. Comparison of average waiting time

Fig. 9 and Fig. 10 compare the value of *Makespan* and *AWT* of 5 algorithms under various tasks. Obviously, for *Makespan*, the wEFT algorithm is better than the other 4 algorithms. When the number of tasks is less, the *Makespan* value is at least 16.3% lower than other 4 algorithms. When the number of tasks is more, the *Makespan* value is at least 28.5% lower than other 4 algorithms. That is, as the number of tasks increases, the performance of wEFT algorithm is better than other algorithms. In terms of average waiting time, the wEFT algorithm is close to the RR and LC algorithm, and the average waiting time of wRR and wLC is the shortest in the experiments.

(2) Result analysis under 6 processor cores

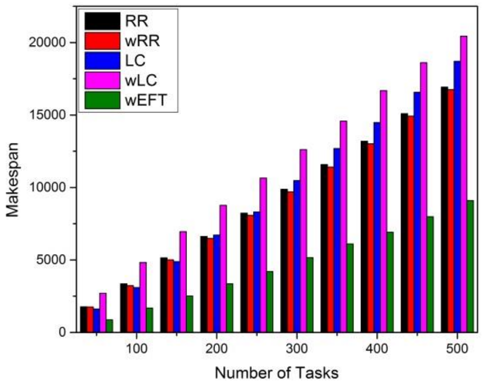

Fig. 11. comparison of *Makespan*

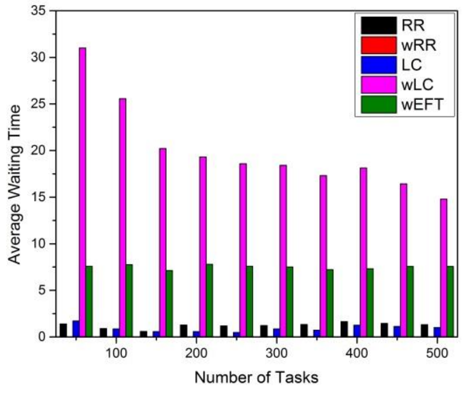

Fig. 12. Comparison of average waiting time

When the number of processor core is increased to 6, Fig. 11 and Fig. 12 compare the value of *Makespan* and *AWT* of the 5 algorithms. For the value of *Makespan*, wEFT algorithm is better than the other 4 algorithms. By comparing Fig. 9 and Fig. 11, with the increase of the processor cores, the *Makespan* will be reduced accordingly. The wEFT algorithm has better *Makespan* value under 6 processor cores environment than under 4 processor cores environment. When the number of tasks is small, the *Makespan* value is at least 46.3% lower than the other 4 algorithms, and when the number of tasks is more, the *Makespan* value is at least 45.6% lower than the other 4 algorithms. As the number of tasks increases, the performance of wEFT algorithm is not changed significantly. In terms of average waiting time, RR and LC algorithms are close, the average waiting time of the wRR is the shortest, the average waiting time of the wEFT algorithm is not better, and the average waiting time of the wLC algorithm is the largest.

VI. CONCLUSION

In this paper, 4 common task scheduling algorithms are analyzed for task scheduling in heterogeneous multi-core processor environment, and an online task scheduling model and a wEFT scheduling algorithm are proposed. Through simulation experiments, the *Makespan* value of wEFT algorithm is obviously better than that of RR, wRR, LC and wLC algorithms.

wEFT algorithm is not dominant in the average waiting time of task set. In the future research work, we need to further optimize and improve the algorithm.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No.61070015), Guangdong Province Natural Science Foundation Team Project (Grant No. 2014B010110004), Guangdong Province Advanced and Key Technology Creative Research Project (Grant No. 2014B010110004) and the Industry-University-Academy Collaborative Innovation Key Project of Guangzhou (Grant No. 201604016074).

REFERENCES

[1] H. F. Sheikh, I. Ahmad, and D. R. Fan, "An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors," IEEE Transactions on Parallel and Distributed System, vol. 27,issue 3, pp. 668-681, 2016.

[2] G. Y. Jia, G. J. Han, J. F. Jiang, N. Sun, and K. Wang, "Dynamic resource partitioning for heterogeneous multi-core-based cloud computing in smart cities," IEEE Access, vol. 4, pp. 108-118, 2016.

[3] Y. J. Chen, W. W. Chang, C. Y. Liu, C. E. Wu, B. Y. Chen,and M. Y. Tsai, "Processors allocation for MPSoCs with single ISA heterogeneous multi-core architecture," IEEE Access, vol. 5, pp. 4028-4036, 2017.

[4] C. W. Chang, J. J. Chen, T. W. Kuo, and H. Falk, "Real-time task scheduling on island-based multi-core platforms," IEEE Transactions on Parallel and Distributed System, vol. 26, issue 2, pp. 538-550, 2015.

[5] N. Q. Zhou, D. Y. Qi, and X. Y. Wang, "A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table," Concurrency & Computation Practice & Experience, in press.

[6] K. Chronaki, A. Rico, M. Casas, M. Moreto, R. M. Badia,E. Ayguade, et al, "Task scheduling techniques for asymmetric multi-core systems," IEEE Transactions on Paralle and Distributed System, vol. 28, issue 7, pp. 2074-2085, 2017.

[7] G. Lucarelli, F. Mendonca, and D. Trystram, "A new on-line method for scheduling independent tasks," 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Spain, pp.140-149, 2017.

[8] J. Wu, and X. J. Hong, "Energy-efficient task scheduling and synchronization for multicore real-time systems," 2017 IEEE 3rd International Conference on Big Data Security on Cloud. China, pp. 179-184, 2017.

[9] I. Yamazaki, J. Kurzak, P. R. Wu, M. Zounon, and J. Dongarra, "Symmetric indefinite linear solver using OpenMP task on multicore architectures," IEEE Transactions on Parallel and Distributed Systems. in press.

[10] T. D. Braun, H. J. Siegel, N. Beck, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," Journal of Parallel & Distributed Computing, vol.61, issue 6, pp.810-837, 2001.

[11] X. L. Zhang, "Study on scheduling algorithm of the independent and associated tasks for cloud computing," Chongqing University,2014.

[12] C. A. R. L. Brennand, J. M. Duarte, A. P. Silva, "SimGrid:a simulator of network monitoring topologies for Peer-to-Peer based computational grids," in press.

[13] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications:the SMPI approach," IEEE Transactions on Parallel and Distributed Systems, vol. 28, issue 8, pp.2387-2400, 2017.

[14] A. Mohammed, A. Eleliemy, and F. M. Ciorba, "Towards the reproduction of selected dynamic loop scheduling experiments using SimGrid-SimDag," IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems, Thailand, pp. 623-626, 2017.