A Task Scheduling Algorithm for Multi-core Processors

Xuanxia Yao, Peng Geng
School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China 100083
yaoxuanxia@163.com, 15261801997@yeah.net

Xiaojiang Du
Department of Computer and Information Sciences
Temple University
Philadelphia, PA, USA, 19122
dux@temple.edu

***Abstract*—With the widespread use of multi-core processors, task scheduling for multi-core processors has become a hot issue. Many researches have been done on task scheduling from various perspectives. However, the existing task scheduling algorithms still have some drawbacks, such as low processor utilization rate, high complexity, and so on. This paper presents a task scheduling algorithm for multi-core processors, which is based on priority queue and task duplication. In the proposed algorithm, the Directed Acyclic Graph (DAG) is used to build a task model. Based on the model, task critical degree, task reminder, task execution time and the average communication time are all considered as the priority metrics. A priority based task dispatching list is set up by comprehensive analysis and calculating the priority for each task. Then interval insertion and task duplication strategies are employed to map tasks to processors, which can decrease the communication cost, improve the processor utilization rate and shorten the schedule length. Our experiments show that the proposed algorithm has better performance and lower complexity than the existing scheduling algorithms.**

***Index Terms*—Multi-core Processors, priority, task duplication, normalized schedule length.**

I. Introduction

Task scheduling for multi-core processors is a key factor on the performance of the multi-core processors. Most traditional task scheduling strategies are focused on single-core processor, and they can't work well for multi-core processors system. With the widespread use of multi-core processors, designing an effective task scheduling strategy has been a hot issue [1].

Task scheduling for multi-core processors should meet two requirements [2], which are load balancing and processor affinity. Load balancing means making even and full use of the processors resources in the system. Processor affinity refers to minimizing the task migration. A processor should allocate some cache for each task running on it so as to store the intermediate data or results of them, by which the execution speed can be improved. But when a task migrates to another processor, the cache allocated to it by the previous processor becomes invalid and the current processor has to re-allocate cache for it, so it is important to avoid task migration as far as possible.

It seems that the two requirements of task scheduling for multi-core processors always conflict with each other. On the one side, to keep load balancing always needs to migrate tasks from one processor to another, which leads to the loss of affinity, and on the other side, considering the processor affinity much or limiting task migration may break load balancing. How to deal with the relations between the two requirements is an important and difficult issue in task scheduling for multi-core processors.

In order to counteract the existing problems in task scheduling algorithm for multi-core processors, we propose a new task scheduling scheme based on priority queue and task duplication, which not only exploits the advantages of the existing algorithms but also introduces some new ideas to improve the task dispatching effects for multi-core processors.

The remainder of this paper is organized as follows: In Section II, we review the related works on task scheduling. In Section III, the mathematical model, notations and assumptions are given. In Section IV, we describe the proposed task scheduling algorithm. In Section V, we make detailed comparison analyses on the performances among the proposed algorithm and the two typical existing schemes. Section VI concludes this paper.

II. Related Works

At present, task scheduling for multi-core processors is considered a NP (Non-Deterministic Polynomial) problem and unable to obtain the optimal solution. The existing scheduling algorithms can only get the suboptimal solutions based on some constraints. Heuristic task scheduling algorithms is generally considered as the most suitable algorithms to find the suboptimal solutions for NP problems. The commonly used heuristic algorithms can be classified into three kinds, which are list based task scheduling algorithm, task-duplication based algorithm and cluster based.

The list based scheduling algorithm can be divided into two kinds, which are list scheduling with static priorities (LSSP) and list scheduling with dynamic priorities (LSDP). For LSSP, tasks are scheduled to the processor that can make it starts earliest in the order of their priorities which are computed in advance. For LSDP, the priorities for the unscheduled tasks should be re-computed after each scheduling. In fact, the

978-1-4799-2419-6/13 $31.00 © 2013 IEEE
DOI 10.1109/PDCAT.2013.47

priorities are for task-processor pairs. LSDP is more complex than LSSP but usually have shorter scheduling length that LSSP. For instance, ISH [3] needs to analyze whether a task can be inserted into the free time between the scheduled tasks on a processors except. The simple list based scheduling algorithms always can't provide ideal scheduling length.

In task-duplication based scheduling algorithms, in order to decrease the communication time between tasks on different processors, some tasks may be duplicated redundantly on the processors that their successor tasks are assigned to after they have been assigned to the different processors, which can further shorten the total scheduling length. Compared with the list based scheduling algorithms, the task-duplication based scheduling algorithms usually have shorter scheduling length and higher complexity. TDS [4] should be the earliest task-duplication scheduling algorithm, which duplicates the predecessors of a critical task to its processor so as to make the task complete earlier. The key factor to shorten the scheduling length is to determine which tasks should be duplicated.

Cluster-based scheduling algorithms usually include two stages, which are mapping and scheduling. In mapping stage, the tasks are mapped into different clusters according to the specified policy. In scheduling stage, the tasks in a same cluster are scheduled to the same processor. MD [5] and DSC [6] are typical ones. In order to short the scheduling length, they all try to make the critical tasks be assigned to the same processor as possible.

In this paper, we try to take advantage of LSDP and task-duplication scheduling to shorten the total scheduling length.

III. The Mathematical Model, Notations and Assumptions

A. DAG Based Task Model

Since the relationship among tasks can be described clearly by a DAG [7], we adopt a DAG to model the tasks in our paper. In order to facilitate description, we use a five-tuple to denote a DAG G, which is $G = (T, E, C, W, V)$. The meaning of each element in the five-tuple is given as following.

- T is the collection of the vertices in graph G, which is denoted by $T=\{T_i|1\le i\le n\}$, here $n=|T|$ is the total number of the tasks. Each vertex T_i represents one task.
- E is the collection of the directed edges in graph G, which is denoted by $E=\{E_{i,j}|1\le i,j\le n\}$. The directed edge $E_{i,j}$ between vertex T_i and T_j indicates that task T_i must be done before starting task T_j. We call that T_j is the successor of T_i and T_i is the predecessor of T_i. $|E|$ is the total number of the directed edges.
- C is the collection of the communication costs, which is denoted by $C=\{C_{i,j}|1\le i,j\le N\}$ and $C_{i,j}$ represents the communication cost from task T_i to T_j.
- W is the collection of tasks' execution time，which is denoted by $W=\{W_i|1\le i\le n\}$, where W_i is the execution time of task T_i.
- V is the collection of the tasks' critical degree, which is denoted by $V=\{V_i|1\le i\le n,1\le i\le 3\}$,where V_i represents task T_i's critical degree. If task T_i is a critical task, its V_i will be assigned to be 3, which is the highest critical degree value, and if T_i is on a path directed to a critical task, its V_i should be set 2. Other, the critical degree of task T_i is 1. In this paper, we use the critical degree as a key factor to measure a task's priority.

B. Notations

In order to describe the algorithm clearly, we define some notations as following.

- $\text{pred}(T_i)=\{T_j|E_{j,i}\in E\}$ is the collection of T_i's predecessors, |pred(Ti)| represents the number of T_i's predecessors.
- $\text{succ}(T_i)=\{T_j|E_{i,j}\in E\}$ is the collection of T_i's successors, $|\text{pred}(T_i)|$ represents the number of T_i's successors.
- $\text{est}(T_i)$ is the earliest starting time of task T_i in the DAG based task graph, if $\text{pred}(T_i)$ is null, $\text{est}(T_i)$ is 0, else $\text{est}(T_i)=\max\{C_{j,i}+\text{est}(T_j)+W_j|\ T_j\in\text{pred}(T_i)\}$.
- $\text{lst}(T_i)$ is the latest starting time of task T_i in the DAG based task graph, if $\text{succ}(T_i)$ is null, $\text{lst}(T_i)=\text{est}(T_i)$, else $\text{lst}(T_i)=\min\{\text{est}(T_j)-C_{i,j}|\ T_j\in\text{succ}(T_i)\}-W_i$.
- EFT (T_i,P_i): the earliest completion time of task T_i on processor P_i when both the interval insertion and task duplication strategies are not adopted.
- $\text{EFT}_{\text{insert}}$ (T_i,P_i): the earliest completion time of task T_i on processor P_i when the interval insertion strategy is adopted.
- EFT_{copy} (T_i,P_i): the earliest completion time of task T_i on processor P_i when task duplication strategy is adopted.
- EFT_{all} (T_i,P_i): is the earliest time that processor P_i completes all the tasks dispatched on it before T_i being mapped to it.
- EST (T_i,P_i): the earliest starting time of task T_i on processor P_i.
- SPT (T_i,P_i): the waiting time of task T_i on processor P_i.
- $PST_{j,i}$ represents the prescheduling time from task T_j to T_i, $PST_{j,i} = W_j+C_{j,i}$, T_j is the predecessor of T_i .
- P is the collection of processors, which is denoted by $P=\{P_i|1\le i\le p\}$, P_i represent processor i. $p=|P|$ is the number of processors.
- TR_i is the remainder of T_i, which is calculated by $TR_i=\text{est}(T_i)-\text{lst}(T_i)$. The smaller the TR_i, the more urgent T_i.

C. Constraints and Assumptions

In order to describe our scheduling strategy, we give the constraints and make assumptions as following.

- Each DAG only has one entry node and one exit node. If there is more than one entry node, a null node should be added as these entries' predecessor. The computation and communication overheads of the null node are all zero. If there are more than one exit nodes, take the same approach to do with it.
- The FCFS(First Come First Service) strategy is taken on a processor after the tasks are assigned to it. Any task is not allowed to preempt the time slot.

- All the processors are homogeneous and have the same performance. In addition, they can communicate with each other.
- A task can't send message to its successors until it has been completed. If a task and its successor task are on the same processor, the communication cost between them is zero.
- A task can't start until all of its predecessors have been completed and it has received the messages from all of them.
- The current scheduling task refers to the task that will be scheduled immediately. The ready task refers to the task that its predecessor tasks have been scheduled. Since the predecessor tasks of a ready task may be duplicated, we don't consider whether they have been completed or not in the task mapping stage.
- There are enough processors available, which means the number of processors is many enough.

IV. The Algorithm Description

A. Priority Calculation

Different task scheduling algorithms have different methods and metrics to measure the task's priority. For example, the remainder of the ready task is used as a metric to compute the priority in each scheduling so as to realize dynamical priority [8]. In addition, the average computation cost and communication cost are always used as important parameters to calculate the task priority [9]. In this paper, we consider the task's priority from multiple aspects. For one thing, we introduce a concept of the task critical degree, which is used to indicate how much the task can contribute to reduce the length of tasks scheduling. The greater, the contribution more, its priority should be higher. For another, not only the task remainder but also the computation or execution time and the average communication time are all introduced into measuring a task's priority. The reason is that a task's computation and communication costs are all critical factors to advance the earliest starting time of its successor tasks. Based on these considerations, the priority of a task can be calculated by Eq.1.

$$\text{priority}(T_i)=\begin{cases} V_i*\dfrac{1}{TR_i}*\dfrac{1}{W_i+\dfrac{1}{|\text{succ}(T_i)|}*\sum\limits_{j\in\text{succ}(T_i)}C_{i,j}}, TR_i\neq 0 \\ V_i*\dfrac{1}{W_i+\dfrac{1}{|\text{succ}(T_i)|}*\sum\limits_{j\in\text{succ}(T_i)}C_{i,j}}, TR_i=0 \end{cases} \quad (1)$$

According to the definition for task critical degree in section 2.1, obtaining the critical tasks and critical paths are the perquisites of getting a task's critical degree. So it is necessary to find out the critical tasks and critical paths of the given task graph using the existing critical path algorithm so as to get the critical degree of each task node before using the Eq.1 to compute the priority of a task. Due to space limitations, it is not described here.

It is obvious that the task with the highest priority in the ready task queue should be chosen as the current scheduling task.

B. Interval Insertion and Task Duplication

Interval insertion means to insert the current scheduling task into the free time interval of a processor. For this purpose, it should check whether there is a free time interval greater than the execution time of the current scheduling task on a processor in current task mapping state, if yes, the interval insertion strategy may be used. To be sure, the interval insertion strategy has two premises, one is that the predecessor task of the current scheduling task has completed, the other is that there is no dependency relation between the inserted task and the waiting task on the processor.

Task duplication strategy is mainly used to deal with the communication delay among the tasks on different processors. This delay is very common in embedded systems. Since there is no communication cost among tasks on the same processor, duplicating the tasks with heavy communication cost to the same processor can reduce the communication cost among them, which is a method to use computation for communication.

Currently, most of the existing tasks scheduling algorithms use only one of the two strategies. For instance, HEFT [10] only adopts the interval insertion strategy, which may lead to the tasks that the interval insertion strategy can't use on them are unable to be scheduled well. In addition, since interval insertion can't decrease the communication costs caused by the tasks on different processors, the total time completion time can only be shortened limitedly. OSA [11] only adopts the task duplication policy, which can decrease the communication costs effectively, but can't avoid generating the redundant tasks. These the redundant tasks may not only be unable to shorten the total tasks completion time, but also waste the processor resources.

C. Task Mapping

Since the interval insertion strategy and task duplication strategy have their own advantages and disadvantages respectively, we try to combine the two strategies to advance the total completion time. The basic idea is computing a task's earliest completed time for different strategies (including neither of them being used) and then choosing one that has the earliest completion time to schedule the task. For this purpose, the following steps should be done.

Step 1. The earliest completion time of task T_i on processor P_i is computed by Eq. 2 when neither of the two strategies is used.

$$\text{EFT}(T_i,P_i)=\text{EFT}(T_j,P_j)+C_{j,i}+W_i \quad (2)$$

Here, T_j is the task with maximum costs (including execution time and communication cost) of all its predecessor tasks, and P_j is the processor that T_j is scheduled to.

Step 2. If the interval insertion strategy can be used to schedule T_i on P_i, $\text{EFT}_{\text{insert}}(T_i,P_i)$ is computed by Eq.3, otherwise, it is set to ∞. T_j in Eq.3 is the latest task communicating with T_i.

$EFT_{insert}(T_i,P_i)$

$$=MAX\{ EFT(T_j,P_j)+C_{j,i} , EFT_{all}(T_i,P_i)\}+ W_i \quad (3)$$

Step 3. For the task T_i to be scheduled to processor P_i, $SPT(T_i,P_i)$ is calculated by Eq.4. If it is less than a threshold, $EFT_{copy}(T_i,P_i)$ is set to be ∞ and go to step 5. Otherwise, check whether there is a T_i's predecessor task T_j with maximum sum of execution time and communication can meet the Inequality 5 and 6 or not, if yes, it indicates that the task duplication strategy can be used to duplicate T_i's predecessor task to processor P_i, and the T_j with the maximum critical degree is chosen to be duplicated. Otherwise, $EFT_{copy}(T_i,P_i)$ is set to be ∞ and go to step 5.

$$SPT(T_i,P_i)= EFT(T_i,P_i) - W_i - EFT_{all}(T_i,P_i) \quad (4)$$

$$W_j<SPT(T_i ,P_i) \quad (5)$$

$$EFT(T_j,P_i)<EST(T_i,P_i) \quad (6)$$

Step 4. $EFT_{copy}(T_i,P_i)$ is calculated by Eq.7, where Task T_k is the task with the second-most prescheduling time in $Pred(T_i)$ and P_k is processor that T_k is dispatched to. Let T_j be the current task to be scheduled on P_i and go back to step 3.

$EFT_{copy}(T_i,P_i)$

$$= \begin{cases} EFT(T_k,P_k)+C_{k,i}+W_i, \\ (T_k \text{ has not sent data to } T_i \text{ after } T_j \text{ is completed}); \\ EFT(T_j,P_i)+W_i, \\ (T_k \text{ has not sent data to } T_i \text{ after } T_j \text{ is completed}). \end{cases} \quad (7)$$

Step 5. Choose the scheduling strategy corresponding to the smallest one among $EFT(T_i,P_i)$, $EFT_{copy}(T_i,P_i)$ and $EFT_{insert}(T_i,P_i)$ to be scheduling strategy for scheduling task T_i to P_i.

D. Algorithm Description

For the sake of clarity, we describe the proposed task scheduling algorithm according to its execution steps as following.

Step 1. If there is a ready task in the task list, add it to the ready task list, otherwise, go to step 5;

Step 2. Compute the priority for each task in the ready task list and choose the task T_i with the highest priority to schedule.

Step 3. For each P_i , on which the task T_i's mapping computing have not been done, perform the five steps in the subsection of task mapping.

Step 4. Choose the processor P_i where task T_i has the earliest completion time to be the T_i's executing processor. That is to say scheduling task T_i to processor P_i.

Step 5. If all tasks have been scheduled, stop, otherwise, go back to step 1.

E. Time Complexity Analysis

According to the function of the proposed task scheduling algorithm, it can be divided into two phases, they are priority computing phase and the task mapping phase. In order to obtain the time complexity of the algorithm, we analyze the two phase's time complexity respectively.

In priority computing phase, the most complicated computations are finding the critical path, computing the execution time and the average communication time of these tasks. For a directed acyclic task graph with n vertices and e edges, the time complexity to find its critical path is $O(n+e)$. And in order to compute the execution time and the average communication time of these tasks in the task graph, we need to traverse all the vertices and edges in it, the time complexity for graph traversal is $O(d\cdot n)$, here, d is the biggest in-degree of a node in the DAG. Accordingly, the time complexity of priority computing phase is $O(n+e)+O(d\cdot n)$.

In task mapping phase, there are mainly three things to do. One is to compute the earliest completion time of a task on a processor without using any of the two strategies, whose time complexity isO ($d\cdot n$). The other is to compute the earliest completion time of a task on a processor when using interval insertion strategy, whose time complexity of this step is also $O(d\cdot n)$. The rest one is to compute the earliest completion time of a task on a processor when using task duplication strategy, which is different from the former two. Since it necessary to check recursively whether there is a current task's predecessor can be duplicated to the processor, the time complexity of this stage should be $O(m\cdot d\cdot n)$, here m is the number of recursion m, which is equal to the DAG's layers. Accordingly, the time complexity of the task mapping on a single processor is $O(2\cdot d\cdot n+m\cdot d\cdot n)$. In the worst case, m is equal to the number of the tasks n, and the time complexity will be $O(2\cdot d\cdot n +d\cdot n^2)$, that is $O(n^2)$. Let p be the number of the processors, the total time complexity of task mapping phase is $O(p\cdot n^2)$, which can also be denoted by $O(n^2)$.

To sum up, the proposed algorithm's time complexity is $O(d\cdot n)+ O(n+e)+ O(n^2)$, that is $O(n^2)$.

V. Performance Analysis

In order to analyze and evaluate the performance of the proposed algorithm objectively, we use the TDS and CPFD [12] task scheduling algorithms for reference and make comparison analyses among our algorithm and them from three aspects. In addition, for the sake of easy description, the proposed algorithm is denoted by PQTD in this section.

A. Scheduling Analysis Based on a Given Task Graph

We make the task graph shown in Fig.1 as an example to analyze and compare the scheduling results of the three task scheduling algorithms.

In Fig.1, there are ten task nodes, the number inside the node is its identification and the number beside the node represents the execution time of the task. In addition, the directed edge from node i to node j indicates that task j shouldn't start until task i is completed, and the number on the directed edge is the communication time from one task to the other.

We schedule the tasks in Fig.1 using TDS, CPFD and PQTD respectively. The scheduling results of the three algorithms are shown individually in Fig.2, Fig.3 and Fig.4. For easy understanding, the scheduling results figures are explained as following.

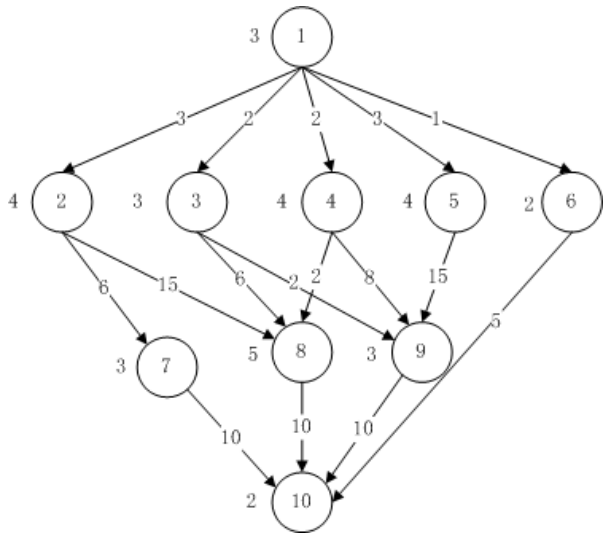

Fig.1 A task graph

In Fig.2, Fig.3 and Fig.4, the abscissa represents the scheduling length and the ordinate represents the processors used. The denotation of "T*X*/*Y*" represents a task node, where *X* denotes the identification of the task and *Y* is the execution time of it. The directed connection between two task nodes means the end task node couldn't begin until the starting point task finished. For the sake of clarity, only the dependence relationships that need long communication time are labeled. In addition, the task node with shadow is a duplicated task.

It can be seen from Fig.2, Fig.3 and Fig.4 that our scheduling length is 24, which is the same as CPFD's but is less than the TDS's 30. In addition, TDS and CPFD all need 6 processors while our algorithm only needs 5. If the two aspects are taken into consider together, our algorithm is better than the other two.

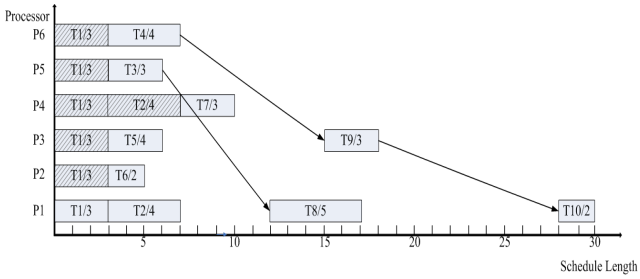

Fig.2 TDS Scheduling Result

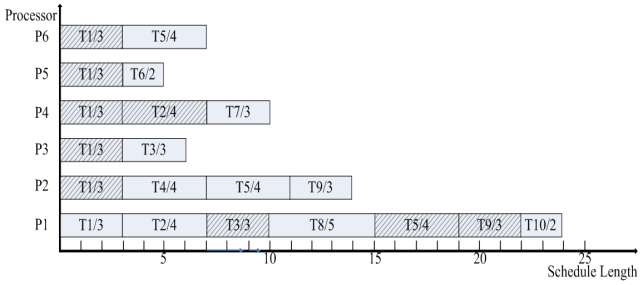

Fig.3 CPFD Scheduling Result

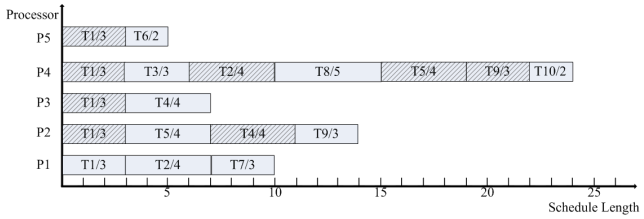

Fig.4 PQTD (The Proposed Algorithm) Scheduling Result

B. Scheduling Analysis Based on Random Task Graphs

In order to make an objective evaluation of the algorithm more generally, we make the random task graphs with 10 to 20 task nodes and the *CCR*(Communication to Computation Ratio) [13] being equal to 0.1, 0.5, 1, 2, 3 and 5 as the scheduling examples to analyze their scheduling results. At the same time, *NSL* (Normalized Schedule Length) [14] is employed as a key factor to evaluate the performance, because *NSL* is the ratio of the critical path length and scheduling length, which can reflect the improvement in the scheduling performance well.

The results of the three scheduling algorithms on the random task graphs with different *CCR* are shown in Fig. 5.

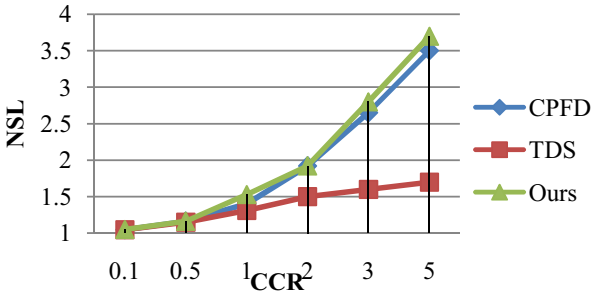

Fig.5 The Scheduling Results on the Random Task Graphs with Different *CCR*

The three algorithms are all based on task duplication, that is to say that they all use task-duplication strategy to decrease the communication cost and accordingly shorten the task scheduling length, so the communication cost has great impact on their scheduling performances [15]. As shown in Fig 5, the *NSL*s of the three algorithms didn't have an observably improvement when the CCR is 0.1, which means the performance didn't improve significantly and can't reflect the advantage of multi-core processors. However, with the increase of *CCR*, all the three scheduling algorithm's performance are all improved significantly. Especially, the improvements of our algorithm and CPFD are much faster than TDS's. And the proposed algorithm's performance is a little higher than CPFD's with the increase of *CCR*. The main reason is that TDS algorithm only considers merging the best predecessor and not taking the other predecessors into account, and at the same time, only task duplication strategy is used in CPFD, there may be some free time that are suitable for use interval insertion strategy with the increasing of *CCR*, which may lead to lose some opportunities to have a shorter scheduling length [16]. In the proposed algorithm, not only the task duplication strategy but also the interval insertion strategy is taken into account,

which can utilize the free time better, so it has a higher performance than CPFD.

C. Comparison in Time Complexity

Since time complexity is also an important criterion to assess the performance of an algorithm, we made a comparison in time complexity among the three task scheduling algorithms, which is shown in Table I.

TABLE I. COMPARISON IN TIME COMPLEXITY

Algorithm	Time Complexity
TDS	$O(n^2)$
CPFD	$O(n^4)$
PQTD(Our algorithm)	$O(n^2)$

The time complexity of TDS is from literature [4] and the time complexity of CPFD is from literature [12].

It can be seen from Table I that the time complexity of our scheduling algorithm is same as TDS's but is much lower than CPFD's.

VI. CONCLUSION

Most existing task scheduling algorithms have the problems of low processor utilization rate and/or high complexity. In this paper, based on priority queue and task duplication, we proposed a new task scheduling algorithm to deal with the problems. We make two contributions. The first contribution is that we introduce the concepts of critical degree as an important metrics to calculate a task's priority, which can reflect the importance of a task well. In addition, the existing priority metrics such as task remainder, the average communication cost and the execution time are all integrated into computing a task's priority, which makes the priority of a task objective and accurate. The second contribution is that we integrate the interval insertion strategy and task duplication strategy to map a task to a processor, which can take full uses of the processors resources and shorten the scheduling length. The comparison experiments showed that the proposed algorithm has lower time complexity and better performance than two popular scheduling algorithms.

ACKNOWLEDGMENT

This work is supported by Chinese National Scholarship Fund and Chinese National High Technology Research and Development Program 863 under Grant No. 2012AA121604, as well as the US National Science Foundation (NSF) under grants CNS-0963578, CNS-1022552, and CNS-1065444.

REFERENCES

[1] Ya-Shu Chen, Han Chiang Liao, and Ting-Hao Tsai, "Online Real-Time Task Scheduling in Heterogeneous Multicore System-on-a-Chip," IEEE Transactions On Parallel and Distributed Systems, 2013,24(1):118-130.

[2] Wu Jia-Jun, "Research On Task Scheduling for Multi-core and Multi-thread processor[PHD. Thesis]," Institute of Computing Technology of the Chinese Academy of Sciences, 2006.

[3] El-Rewinin H, Lewis T G, and Ali H H, "Task Scheduling in Parallel and Distributed Systems," Englewood Cliffs, New Jersey Prentice Hall , 1994, 401-403.

[4] DARBHA S, and AGRAWAL D P, "Optimal Scheduling Algorithm for Distributed-memory Machines," IEEE Transactions on Parallel and Distributed Systems, 1998, 9(1):87-95.

[5] MY WU, Dgajski D, and Hypertool, "A programming Aid for Message-Passing Systems," IEEE Transactions on Parallel and Distributed Systems, 1990,1(3):330-343.

[6] Yang T, and Gerasoulis A, "Scheduling Parallel Tasks on An Unbounded Number of Processors," IEEE Transactions on Parallel and Distributed Systems, 1994,5(9):951-967.

[7] Ahamd I, and Ranka S, "Using game theory for Scheduling Tasks on Multi-core Processors for Simultaneous Optimization of Performance and Energy," IEEE International Symposium on Parallel and Distributed Processing, 2008:2645-2650.

[8] Meng Xian-Fu, Yan Ling-ling, and Liu Wei-Wei, "Research on Task Scheduling Algorithm in Grid Computing Systems based on Dynamic Task Priority," Journal of Dalian University of Technology, 2012,52(2):277-284.

[9] Li Jing-Mei, and Jin Sheng-Nan, "Research on Static Task Scheduling based on Heterogeneous Multi-core Processors," Computer Engineering and Design, 2013,34(1):178-184.

[10] Topcuoglu H, Hariri S, and Wu M Y, "Task scheduling algorithms for heterogeneous processor," In: Proceedings of the Heterogeneous Computing Workshop, San Juan, Puerto Rico, 1999.3-14.

[11] Park C-I, and Choe T-Y, "An Optimal Scheduling Algorithm based on Task Duplication," IEEE Transactions on Computers. 2002, 51(4):444-448.

[12] Ahmad I, and Kwork Y, "On Exploiting Task Duplication in Parallel Programs Scheduling," IEEE Transactions on Parallel and Distributed Systems, 1998,9(9):872-892.

[13] Ruan You-Lin, Liu Gan, Zhu Guang-xi, and Lu Xiao-feng, "Duplication Based Scheduling Algorithm for Dependent Tasks," Chinese Mini-Micro Systems, 2005,26(3):335-339.

[14] Lan Zhou, and Sun Shi-Xin, "An Algorithm of Allocating Tasks to Multiprocessors Based on Dynamic Critical Task," Chinese Journal of Computers, 2007,30(3):454-462.

[15] Xu Cheng, Zhao Lin-Xiang and Yang Zhi-Bang, "Scheduling Algorithm of Multiple-processor Based on Task Clustering and Duplication," Application Research of Computers, 2012,29 (8):2931-2934.

[16] Meng Xian-Fu, and Liu Wei-Wei, "A DAG Scheduling Algorithm Based on Selected Duplication of Precedent Tasks," Journal of Computer-Aided Design & Computer Graphics, 2010,22(6):1056-1062.