Research on Hierarchical Multi-core Scheduling Algorithm Based on Task Replication

Yin Haijing
School of Computer Science and Engineering
Xi'an Technological University
Xi'an, China
E-mail: 1391275494@qq.com

Huang Shujuan
School of Computer Science and Engineering
Xi'an Technological University
Xi'an, China
E-mail: 349242386@qq.com

Wang Jianguo
School of Computer Science and Engineering
Xi'an Technological University
Xi'an, China
E-mail: 2269261628@qq.com

***Abstract*—The rapid development of multi-core systems makes task scheduling in multi-core systems a new research topic. While tasks are running in parallel, how to improve the efficiency of the system and maintain the load balance of the system is the focus of research in the new era. Aiming at the problem that the multi-core scheduling algorithm based on task duplication does not consider the load balance of each CPU, which leads to the problem of reduced CPU utilization. This paper combines a hierarchical idea on the basis of task replication, and proposes a new hierarchical multi-core scheduling algorithm TDLS algorithm based on task replication. This algorithm is based on the idea of hierarchical scheduling. According to the fact that there is no dependency relationship between tasks at the same layer after layering, the task scheduling sequence is adjusted to reduce the waste on the core, shorten the waste between cores caused by communication time, and reduce the number of processors. , Thereby greatly improving the CPU utilization rate, using the least time and the least number of cores to complete scheduling, making the load of multi-core scheduling more balanced. Experiments show that under the same experimental conditions, compared with the traditional multi-core scheduling algorithm based on task replication, the improved algorithm TDLS reduces the number of processor cores, and also shortens the scheduling length of the total task. Its performance is better than the traditional multi-core scheduling algorithm based on task replication.**

Keywords-Load Balancing; Task Scheduling; Task Duplication; Hierarchical Scheduling

I. INTRODUCTION

Multi-core processor technology mainly integrates two or more processor cores on a single chip to enhance computing performance. Multi-core processors improve system performance by distributing load on multiple CPU cores, and relying on high-speed on-chip interconnection and high-bandwidth pipelines of memory and input/output (I/O). Under the same conditions, multi-core processors can bring more performance and productivity advantages than current single-core processors. Therefore, the research of scheduling algorithms under multi-core platforms is also a future development trend.

Multi-core processor task scheduling refers to how to allocate multiple tasks to multiple cores for parallel execution through a scheduling algorithm, so as to minimize the total time for task completion. Multi-core task scheduling has long been proved to be an NP problem [1], and it is difficult to find the optimal solution in polynomial time. The most common task scheduling algorithm is based on heuristic scheduling algorithm. Heuristic scheduling algorithms mainly include table scheduling algorithm based on critical path [2-5], task duplication algorithm [6-8], processor allocation algorithm based on task duplication [9], improved multi-core scheduling based on task duplication Algorithm [10]，clustering algorithm [11-13] and so on.

Since the communication overhead between tasks on the same processor can be ignored, scheduling based on task duplication is an effective strategy for reducing communication overhead. The characteristic of the task copy method is to reduce the communication time between processors by copying the predecessor tasks that have a communication relationship, thereby reducing the execution time of the system as a whole.

When using reasonable and effective duplication rules and strategies, scheduling algorithms based on task duplication have been proven to have better scheduling effects than other scheduling algorithms. However, the scheduling algorithm does not consider the factor of load balancing, and in the DAG graph, there is no dependency between nodes in the same layer. According to the adjustment of the scheduling sequence between nodes in the same layer, the idle time is reduced and the CPU is increased. Utilization, while coordinating the load in each CPU to make it more balanced. Therefore, this paper proposes a hierarchical scheduling algorithm based on task duplication to solve the shortcomings of unbalanced load of traditional scheduling algorithms based on task duplication.

II. TASK SCHEDULING MODEL

The task scheduling problem is a kind of combinatorial optimization problem in mathematics, that is, an abstract task model of a computer application is established, and then based on the constraints of the task model, through a reasonable scheduling strategy, a scheduling sequence is generated and the tasks are assigned to the processing cores for calculations. Get the least total task execution time and maximize the parallel execution advantages of multi-core systems.

The task scheduling model is mainly divided into two aspects: system model and task model. The system model is a mathematical abstraction of information such as the topological structure and computing capabilities of a multi-core system, and the task model is a mathematical abstraction of computer application programs. It mainly includes information such as the constraint relationship between tasks and the characteristics of the task itself. The following are two parts Detailed discussion.

A. System model

The system model is an abstraction of the actual computing system. The actual computing system in this article is a multi-core system, that is, a system composed of multiple processing cores.

The system is generally expressed as $\mathrm{P}=\{\mathrm{p}_1,\mathrm{p}_2,\cdots,\mathrm{p}_i,\cdots,\mathrm{p}_n\}$.

Among them, P represents a collection of processing cores in a multi-core system, which p$_i$ represents the i processing core, and n represents that the system contains a total of n cores.

B. Mission model

The relationship between multi-core tasks is generally represented by DAG (Directed Acyclic Graph), and when there is a dependency between tasks, a weighted DAG graph is used (as shown in Figure 1)

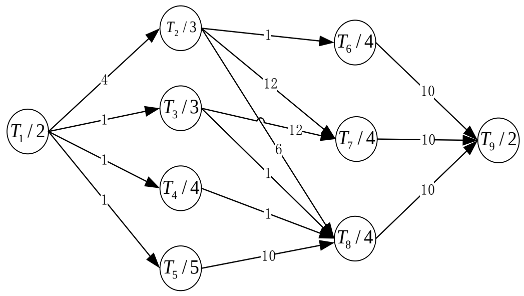

Figure 1. DAG diagram

Its mathematical description is:

$$G=\{T,E,t,c\}. \tag{1}$$

Among them, the formula $T=\{T_1,T_2,\cdots,T_i,\cdots,T_n\}$ represents the set of nodes in the graph, which is the first task; represents the set of nodes in the graph, which is the first task;

$E=\{E_{ij}\}$ Represents the set of directed edges that E_{ij} is a communication relationship between task T_i and task T_j, otherwise, they cannot communicate directly. $t=\{t_1,t_2,\cdots,t_i,\cdots,t_n\}$. This Set represents the set of node weights in the graph，in other words，t_i is execution time of the task *i*. Meanwhile, The set,$\{c=c_{ij}\}$,is the set of weights of directed edges that c_{ij} Indicates the communication time between task T_i and task T_j.

Since the communication time of tasks between different cores is much longer than the communication time between the same cores, when two tasks are on the same processor, the communication time is ignored, that is $c_{ij}=0$ in two related tasks on the same processor.

Definition:

- The earliest start time T^i_{begin} of task *i*: it represents the smaller value between the predecessor time of task execution and the maximum associated predecessor time in the task predecessor set. which is:

$$T^i{}_{begin}=\min\{T^i{}_{exe\_pre},\max\{T^i{}_{re\_pre}\}\}. \tag{2}$$

- The completion time T^i_{end} of task *i*: the time when the task is executed on the processing core is equal to the start time plus the task execution time, which means the time it takes to complete the task. which is:

$$T^i{}_{end}=T^i{}_{begin}+t_i. \tag{3}$$

- Associated predecessor *j* of task *i*: The set of tasks that must be completed before the task *i* is executed. That is, the task set on which the execution of the task depends. For example, the task set T_i, Tj in Figure 1 is the associated predecessor of task T_7. The associated predecessor time $T^i_{re\_pre}$ of task *i* is: if the associated predecessor task *j* and task *i* of the task are on the same core, the associated predecessor time is the completion time of the associated predecessor task *j*; if not on the same core, The associated predecessor The time is the completion time of the associated predecessor task *j* and the maximum value of the sum of communication values between task *i* and its associated predecessor task *j*.

$$T^{i}_{re\_pre} = \begin{cases} T^{j}_{end}, & i, j \text{ in the same core} \\ T^{j}_{end} + c_{ij}, & i, j \text{ in a different core} \end{cases}. \quad (4)$$

Where task *j* is a predecessor task of task *i*.

- The running result of a certain processor p^{i}_{end} indicates the total time spent by the processor after all tasks on the processor are scheduled.
- The running result of all processors P_{end}: it represents the final task scheduling result, namely: the total time for all tasks to complete.
- The successor task *next* of task *i*: the task related to task *i*, and it must be ensured that task *i* has been executed before it is executed.
- Execution predecessor *k* of task *i*: On the same core, task *k* to be executed before task execution is the execution predecessor of task *i*. And the execution predecessor time $T^{i}_{re\_pre}$ of task *i* is: the completion time of the execution predecessor of task *k*, that is:

$$T^{i}_{re\_pre} = T^{k}_{end}. \quad (5)$$

- The idle time T^{i}_{rest} of task *i*: represents the wasted time slice on the same core when the task is executed. which is:

$$T^{i}_{rest} = T^{i}_{begin} + T^{i}_{exe\_pre}. \quad (6)$$

- The communication start time $T^{next}_{begin\_comm}$ between task *i* and the successor task *next*: It mainly represents the communication relationship between task *i* and its successor task *next* that are not on the same processor. That is: after the execution of task *i* is over, after the communication time between processor cores, the start time of the successor task *next*.

C. Basic constraints of task scheduling

For a certain task, when it meets the following two necessary conditions, it can be executed on a specific processing core.

On one hand, All the predecessor tasks of the same processing core with which it is dependent have been executed, and the communication between the predecessor tasks that are not on the same processing core and this task has also been completed;

On the other hand, the time period occupied between the task start time and the end time does not conflict with other tasks on the processing core where the task is located.

- For task *i*, its earliest start execution time must not be less than the execution completion time of all predecessor tasks, which is:

$$T^{i}_{begin} \geq \max\{T^{i}_{re\_pre}\}. \quad (7)$$

- The start time of the communication between the task and the successor task must be after the end time of the task, because only the task execution is completed before the relevant data can be provided to the successor task, which is:

$$T^{i}_{begin\_comm} \geq T^{i}_{end}. \quad (8)$$

- For a task graph, the final task scheduling result depends on the task that finishes executing at the latest. That is, the time used by the processor with the longest scheduling length among all processors. which is:

$$P_{end} = \max\{ p^{i}_{end} \}. \tag{9}$$

III. TASK DUPLICATION SCHEDULING ALGORITHM

The idea of task duplication scheduling algorithm is to generate copies of specific tasks through duplication. These copies will be allocated to the processing core according to a certain strategy. When the subsequent tasks of the copy are allocated to the same processing core, they can be offset. Consumption of communication between tasks to save time.

Task duplication can be divided into single-task duplication and multi-task duplication according to the number of duplication tasks. Single-task duplication copies and allocates tasks that restrict the start time of the current task to the processing core; multi-task duplication copies and allocates multiple predecessor tasks of the task to the processing core.

Taking Figure 1 as an example, the successor tasks of task T_1 are T_2, T_3, T_4, and T_5. Therefore, when T_1 and T_2, T_3, T_4, and T_5 are executed on different processing cores, there will be inter-core communication, in order to save inter-core During the communication time, a task duplication strategy is adopted to replicate T_1, and all three copies are allocated to the processing cores where T_2, T_3, T_4, and T_5 are located. Through this task duplication method, the task scheduling result is finally optimized.

As shown in Figure 2, if the T_1 task is not copied, and assuming that the tasks T_2 and T_1 are not on the same core, the earliest start time of the task T_2 is equal to the time W_1 waiting for the completion of the predecessor task T_1 and the task that is not on the same processor The sum of the communication time $C_{1,2}$ between T_1 and task T_2, that is, the earliest start time of T_2 is 6, and the scheduling result of task T_2 is 9; and if the task duplication strategy is adopted for scheduling, a copy is made on the processor where T_2 is located The copy of T_1, at this time, because the communication time is reduced by 4, the start time of T_2 becomes 2, and the scheduling result of task T_2 is reduced to 5 (as shown in Figure 3). At this time, the optimization effect is significant and the scheduling efficiency is greatly improved.

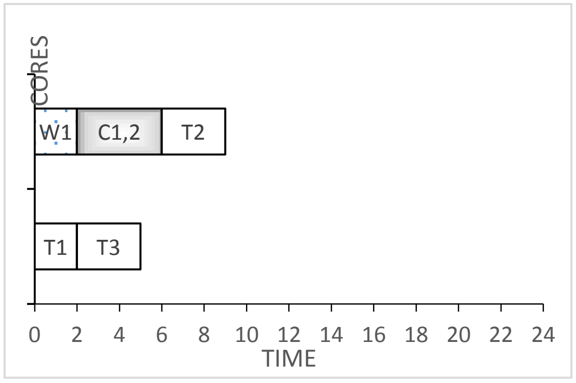

Figure 2. T1 does not use task duplication strategy T2 scheduling result diagram

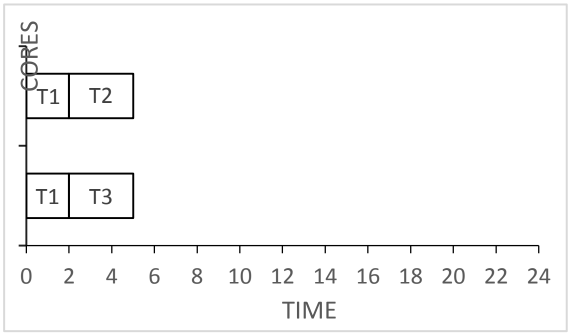

Figure 3. T1 adopts task duplication strategy T2 scheduling result graph

IV. TDLS SCHEDULING ALGORITHM

In the DAG diagram, tasks can be classified by layer. The tasks in the same layer are independent, and the tasks in the same layer are not dependent on each other, that is, if there is no priority difference, the tasks in the same layer, there is no difference in the execution of tasks. The TDLS algorithm mainly relies on hierarchical scheduling without dependencies between tasks on the same layer. By adjusting the scheduling sequence of tasks on the same layer, tasks with a smaller initial start time can be scheduled when the time slice comes, reducing the number of cores. At the same time, considering that the communication value between tasks with dependencies between different layers is too large, it will also affect the completion time of the task, so the predecessor tasks that have a greater impact on the task are adjusted to reduce the communication time between tasks, Thereby increasing the CPU utilization.

A. Steps of hierarchical multi-core scheduling algorithm based on task replication

Step 1: Calculate the in-degree of all tasks in the DAG graph;

Step 2: Determine whether the in-degree of all tasks is 0, and put all tasks with in-degree of 0 into one layer, that is, the k layer is obtained;

Step 3: Remove the tasks that have been layered in the DAG graph and their related edges to get a DAG graph, and make $k = k + 1$, repeat steps 1~3, until the task in the DAG graph is empty, that is The DAG graph completes the layering operation.

B. The Steps of TDLS Algorithm

Step 1: According to the in-degree of the tasks in the DAG, use the hierarchical algorithm to perform hierarchical operations on all tasks;

Step 2: According to the layering result, the scheduling sequence is initially obtained; because the tasks in the same layer do not have mutual dependence, the tasks of the same layer can be scheduled according to the earliest start time T^{i}_{begin} of the task in the order of scheduling from small to large to reduce idle time , Improve CPU utilization.

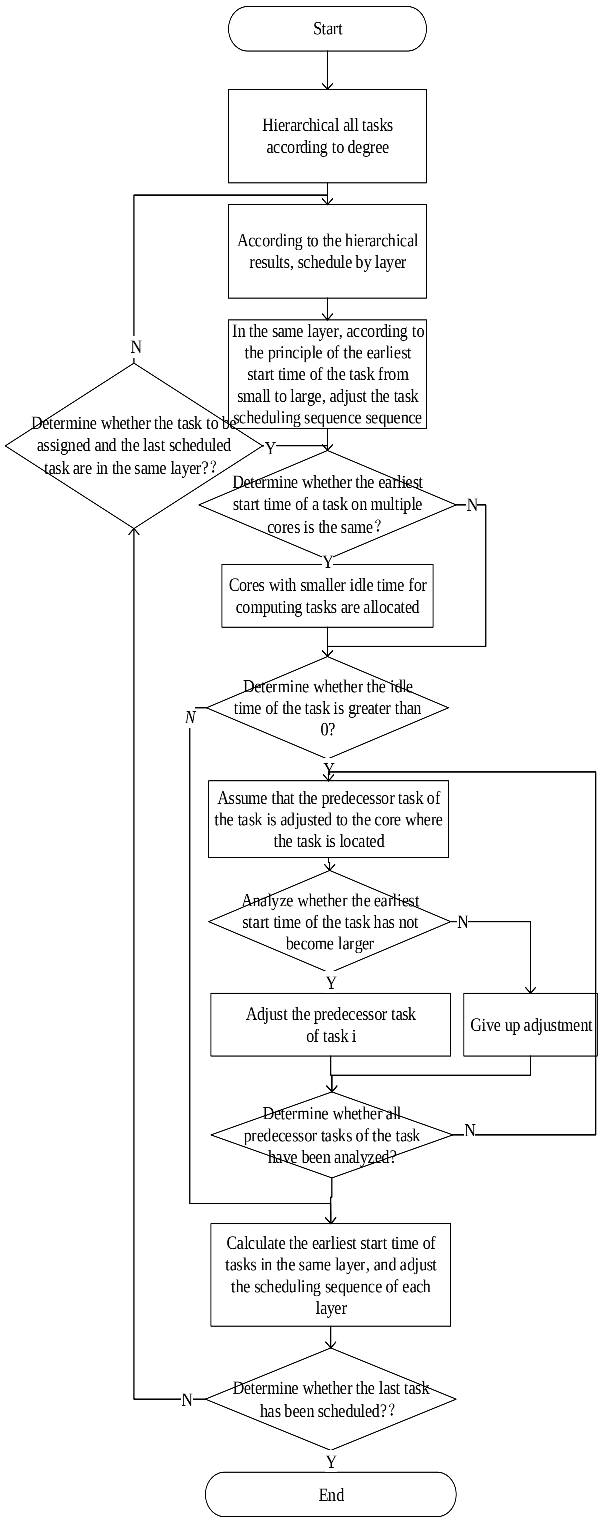
Start
Hierarchical all tasks according to degree
According to the hierarchical results, schedule by layer
In the same layer, according to the principle of the earliest start time of the task from small to large, adjust the task scheduling sequence sequence
Determine whether the task to be assigned and the last scheduled task are in the same layer??
N
Y
Determine whether the earliest start time of a task on multiple cores is the same?
N
Y
Cores with smaller idle time for computing tasks are allocated
Determine whether the idle time of the task is greater than 0?
N
Y
Assume that the predecessor task of the task is adjusted to the core where the task is located
Analyze whether the earliest start time of the task has not become larger
N
Y
Adjust the predecessor task of task i
Give up adjustment
Determine whether all predecessor tasks of the task have been analyzed?
N
Calculate the earliest start time of tasks in the same layer, and adjust the scheduling sequence of each layer
Determine whether the last task has been scheduled??
N
Y
End

Figure 4. TDLS algorithm flow chart

Step 3: Schedule each task in turn according to the adjusted task scheduling sequence sequence. The polling method assigns the task T_i to each core in turn, calculates and compares the earliest start time T^i_{begin} of the task on each core, so as to find out which core takes the least time, that is, allocate T_i to the core.

If the task's earliest start time T^i_{begin} is not single, consider the core with the smaller idle time T^i_{rest} for priority allocation, and when the idle time T^i_{rest} is greater than 0, it means that the communication time between a predecessor task $T^i_{re\_pre}$ of the task and the task is relatively large. When they are not on a core, the earliest start time T^i_{begin} of the task is relatively large, so the precursor task $T^i_{re\_pre}$ that affects the start time of the task is adjusted to the core where the task is located for analysis and comparison. If the earliest start time of the task is not If it changes or becomes smaller, adjust the predecessor task $T^i_{re\_pre}$ of the task to the core where the task is located; otherwise, give up the adjustment. The final scheduling sequence is the optimal scheduling sequence (the algorithm flow chart is shown in Figure 4).

C. Analysis of the time complexity of TDLS algorithm

When selecting an algorithm for a task scheduling problem, it is usually necessary to evaluate the time complexity of the algorithm. Time complexity refers to the increasing trend of the execution time required by the algorithm when the scale of the problem expands. Generally, O is used to represent the time complexity. Among them, O(1) means that the time complexity of the algorithm is constant, that is, no matter how much the problem scale increases, the time consumed by the algorithm remains the same; O(n^2) means that the time complexity of the algorithm is square, that is, when the problem scale When the problem is enlarged by 2 times, the time required for the algorithm to solve the problem is 4 times; O(2^n) means that the time complexity of the algorithm is exponential of 2, and once the scale of the problem increases, the time consumed by the algorithm will be exponential increase.

Assume that the DAG task graph has n nodes. By traversing the task graph to schedule each node, the time complexity required for each node is O(n). For each node of each layer, it is necessary to perform simulation scheduling before confirming the scheduling to determine whether the scheduling reduces idle time, reduces CPU waste, and improves CPU utilization. When planning to schedule, the scheduling sequence of each layer needs to be adjusted. At this time, assuming that there are m nodes in a certain layer, the time complexity required at this time is O(m), so the TDLS algorithm is extremely In this case, the time complexity of the algorithm is not greater than O(n^2).

V. EXPERIMENTAL RESULTS AND ANALYSIS

This chapter will use the TDLS algorithm and the traditional task replication-based scheduling algorithm (CPTD and TDMC algorithm) to conduct experiments in the same environment to schedule the specific DAG graph (Figure 1), and use TDLS, CPTD and TDMC to perform the experiments respectively. The task scheduling distribution diagram of the three algorithms available for scheduling is shown in Figures 5~7, and the two performance indicators and the algorithm time complexity of the number of processors used and the CPU utilization rate are compared. Each of the three algorithms is the comparison of performance evaluation parameters is shown in Table 1.

A. Performance evaluation parameters

The performance of the task scheduling algorithm can be evaluated with the following performance evaluation parameters:

- The number of processors used to complete all tasks. After all tasks in the task graph are scheduled, the fewer processors are used, the more resources can be saved.
- Scheduling length. After all tasks in the task graph are scheduled, the length of time used, the smaller the scheduling length, the better the explanation of resources, and the better the algorithm.

B. Task diagram example analysis

For a specific task graph example (Figure 1), call TDLS, CPTD, and the (TDMC algorithm for short) in Literature 14 (referred to as TDMC algorithm [14]) respectively based on task duplication scheduling algorithm, draw a scheduling diagram, and compare these 3 algorithms Various performance indicators.

The DAG graph shown in Fig. 1 has 4 layers, including 9 tasks $T_1 \sim T_9$. Among them, each circle in the figure represents a task node, and the nodes in the node represent the task and the time required to execute the task. The line segment with arrows in the figure represents the communication dependency between tasks. Where the start position of the arrow line segment is the predecessor task node, the end position is the successor task node, and the number on the straight line represents the communication time between tasks (because the execution time of two tasks on the same CPU is much shorter than that of different CPU The execution time of the two tasks between the two, therefore, when two dependent tasks are on the same CPU, the communication time can be ignored).

TABLE I. COMPARISON OF SCHEDULING RESULTS OF TDLS, CPTD, AND TDMC ALGORITHMS

Algorithms	**Scheduling Length**	**Number of Processors**	**Algorithm Time Complexity**
TDLS	23	2	O(n^2)
CPTD	23	3	O(n^2)
TDMC	24	5	O(n^2)

Use TDLS, CPTD and TDMC three algorithms to schedule the DAG graph respectively, and the task scheduling and distribution diagram corresponding to the three algorithms can be obtained, as shown in Figure 5~7, and the comparison of each performance evaluation parameter under the three algorithms the situation is shown in Table 1.

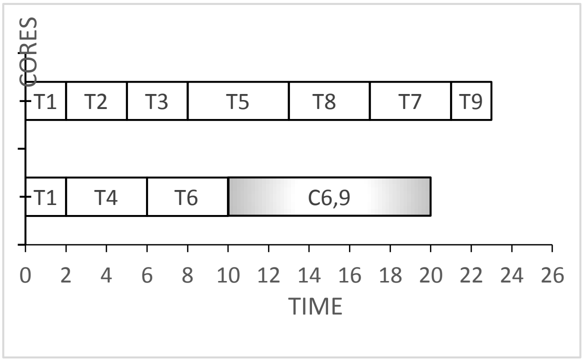

Figure 5. TDLS algorithm scheduling diagram

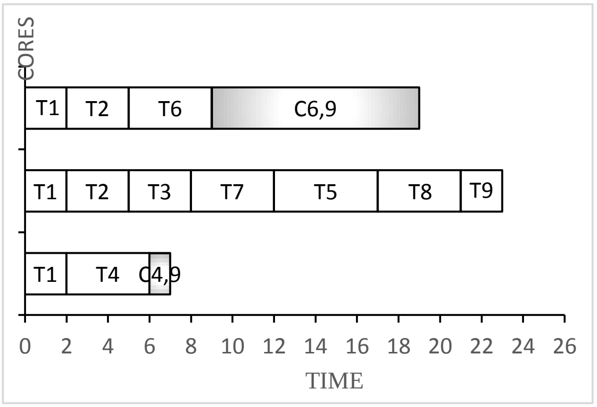

Figure 6. CPTD algorithm scheduling diagram

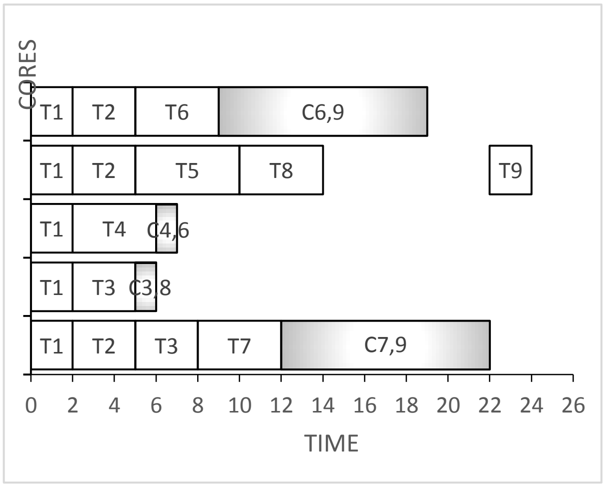

Figure 7. TDMC algorithm scheduling diagram

It can be seen from Figures 5 to Figures 7 and Table 1 that the TDMC algorithm is used to schedule the task graph. The total execution time is 24, and the number of processor cores is 5. The main reason why the CPU is not fully utilized is T_7 and T_9. The communication time of the node is too long, resulting in waste between T_9 and T_8, thereby increasing the total length of task scheduling. The CPTD algorithm is used to schedule the task graph. The total execution time is 23, and the number of processor cores is 3. The algorithm first finds the critical path, and then adopts the early completion strategy of the preceding node group to merge the scheduling sequences of nodes T_7 and T_8 , So that the execution time of T_9 can be advanced, and the communication time can be better controlled. The scheduling of T_6 and T_4 is completed through task replication, but the processors where these two tasks are located mainly complete the scheduling of T_6 and T_4, which greatly reduces the CPU Utilization rate makes the CPU load unbalanced.

The total execution time used by the improved TDLS algorithm is 23, which only occupies 2 processor cores. It is mainly used in the scheduling process of the third layer, by adjusting the scheduling sequence, according to the T_7 and T_8 nodes are unrelated For nodes on the same layer, T_7 is adjusted to T_8 before scheduling, which shortens the waste of time caused by too long communication time, improves CPU utilization, and makes the load more balanced than the previous algorithm.

Compared with the TDMC algorithm, the TDLS algorithm on the one hand shortens the total scheduling length of task execution (the scheduling length is reduced from 24 to 23), and on the other hand, it reduces the number of CPU required to complete all tasks (the number of CPU is reduced from 5 to 2), the algorithm reduces the waste of resources on the whole and improves the utilization of CPU; at the same time, compared with the CPTD algorithm, the TDLS algorithm reduces the number of redundant tasks on the one hand (such as: Reduction in the number of tasks T_1 and task T_2 redundancy). On the other hand, it also reduces the number of CPU required to complete the task (the number of CPU is reduced from 3 to 2). It also reduces the waste of time slices between tasks and greatly improves each CPU The utilization rate makes the load more balanced. Through the comparison and analysis of the above two groups of experiments, it can be seen that the TDLS algorithm has better scheduling performance than the CPTD algorithm and the TDMC algorithm.

VI. CONCLUSION

Aiming at the problem of load imbalance in the traditional multi-core scheduling algorithm based on task duplication under certain circumstances, this paper proposes a hierarchical scheduling algorithm based on task replication, TDLS. TDLS is based on no scheduling sequence between tasks in the same layer. The scheduling sequence of tasks in the same layer can shorten the idle time of the CPU, shorten the execution time of the program, and then improve the utilization of multiple CPU. It is proved through comparative experiments that the TDLS algorithm is compared to the other two traditional scheduling algorithms based on task replication. The load is more balanced, and it also has a certain significance for the improvement of the scheduling performance of the multi-core parallel computer system.

The hierarchical task scheduling algorithm based on task replication proposed in this paper overcomes some inherent shortcomings of traditional task replication-based algorithms in the experimental environment of simulated scheduling, and improves the efficiency of task scheduling and CPU utilization. The research of this algorithm is completely based on assumptions and simulated environment. However, the real situation is more complicated and changeable. For multi-core systems, its load balancing, power consumption, and communication congestion between cores need to be considered under actual conditions. Therefore, it is necessary to further expand the scope of research in future research, apply it to real-time systems, and make more comprehensive considerations for the problems in the actual situation.

REFERENCES

[1] Han Yingjie. Research on Multi-core Task Scheduling Based on Comprehensive Scheduling Critical Path [D]. Harbin University of Science and Technology, 2014.

[2] Ren Liangyu, Zhao Chengping, Yan Hua. Multi-core scheduling algorithm based on task duplication and redundancy elimination [J]. Computer Engineering, 2019, 45(05):59-65.

[3] Shi Wei, Zheng Weimin. A balanced dynamic critical path scheduling algorithm based on related task graphs [J].Chinese Journal of Computers, 2001(09):991-997.

[4] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger,Chung-Yee Lee. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. [J]. SIAM J. Comput.,1989,18(2):

[5] Wu M Y, Gajski D D. Hypertool: a programming aid for message-passing systems [J]. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(3):330-343.

[6] Liu Y, Jia P, Yang Y. Efficient scheduling of DAG tasks on multi-core processor based parallel systems[C]// Tencon IEEE Region 10 Conference. IEEE, 2016.

[7] S. Darbha and D. P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines," in IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 1, pp. 87-95, Jan. 1998, doi: 10.1109/71.655248.

[8] An optimal scheduling algorithm based on task duplication [J]. Journal of Systems Engineering and Electronics, 2005(02):445-450.

[9] Harbin. An Algorithm of Processor Pre-Allocation Based on Task Duplication [J]. Chinese Journal of Computers, 2004.

[10] Ye Jia, Zhou Mingzheng. An improved multi-core scheduling algorithm based on task replication［J］. Computer Engineering and Applications, 2015, 51(12): 31-37.

[11] Boeres C, Filho J V, Rebello V. A Cluster-based Strategy for Scheduling Task on Heterogeneous Processors[C]// Symposium on Computer Architecture & High Performance Computing. IEEE, 2004.

[12] Palis M A, Liou J C, Wei D. Task Clustering and Scheduling for Distributed Memory Parallel Architectures. IEEE Transactions on Parallel and Distributed Systems, 7(1):46-55 [J]. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(1):46-55.

[13] Lan Zhou. Research on scheduling algorithms in distributed systems［D］. Chengdu: University of Electronic Science and Technology of China, 2009.

[14] Zhiqiang Xie, Lei Zhao, Yu Xin, Jing Yang. A Scheduling Optimization Algorithm Based on Task Duplication for Multi-core Processor [J]. Energy Procedia,2011,13:

[15] Ahmad Wakar, Alam Bashir. An efficient list scheduling algorithm with task duplication for scientific big data workflow in heterogeneous computing environments [J]. Concurrency and Computation: Practice and Experience, 2020,33(5):

[16] Computing - Supercomputing; Findings in the Area of Supercomputing Reported from Harbin Institute of Technology (Linear and Dynamic Programming Algorithms for Real-time Task Scheduling With Task Duplication) [J]. Computer Weekly News,2019:

[17] Cao Zhebo, Li Qing. Research and design of multi-core processor parallel programming model [J]. Computer Engineering and Design, 2010, 31(13): 2999-3002+3056.

[18] Chen Gang, Guan Nan, Lu Mingsong, Wang Yi. A review of real-time multi-core embedded systems [J]. Journal of Software, 2018, 29(07): 2152-2176.