

Optimization Methods of Multi-Core Embedded System

Haorui Chen *

Software Institute, East China Normal University, Shanghai, 200062, China

* Corresponding Author Email: 10205101444@stu.ecnu.edu.com

Abstract. With the development of the Internet of everything technology, embedded system has become one of the most common computing systems. Embedded system has high portability, but there are often stronger limitations in energy consumption, real-time and so on. In this work, have improved some traditional optimization algorithms, and finally get a task scheduling sequence, which can reduce the total execution time and cost of the task. Firstly, the multi-module division is used to divide multiple tasks into different modules, improve the classical Kernighan-Lin (KL) algorithm and clustering algorithm implement this process. Second, this paper invokes a series of algorithms to calculate the priority value of the task. Finally, this paper call two multi-core scheduling algorithms to schedule the tasks within each module to the Central Processing Unit (CPU) in the module. The proposed algorithm is implemented in this paper. With directed acyclic graph as input, the improvement effect of the algorithm and the direct advantages and disadvantages of different algorithms are evaluated with different task quantity.

Keywords: Embedded system; Optimization algorithm; Multitask scheduling.

1. Introduction

Nowadays, with the development of science and technology, intelligent embedded system has become more and more common in our life. There are all kinds of embedded systems in mobile phones, smart homes, cars and even airplanes [1]. As people's needs grow, today's embedded systems need to handle more and more complex tasks. The timeout of a task may cause the user experience to deteriorate and even cause security accidents. Therefore, the optimization of intelligent embedded system has great research significance.

Embedded system optimization aims to meet the performance and cost requirements of embedded devices. A mainstream idea is to minimize the communication cost as much as possible in order to meet the real-time and energy consumption requirements of embedded systems [2]. At present, some existing algorithms focus on multi-task scheduling, and have achieved considerable results in reducing the communication cost of the system, but there are some shortcomings in the execution time and partition effect of the algorithm, and there is room for improvement [3-5].

In this paper, while keeping the process of first module partitioning and then multi-core scheduling unchanged, the task communication graph is used as the input to minimize the communication cost, and the execution time and the final partition effect of the algorithm are further optimized. Since the existing algorithms have been optimized to some extent in this paper, the optimal algorithm can be selected according to different scenarios in practical application, which has strong flexibility.

2. Optimal Design Methods

Embedded systems are often composed of multiple modules, each of which contains multiple independent CPU. In the actual task scheduling, the multi-module division should be carried out first [6]. Assume that the cost of communication between tasks in the same module is 0. Scheduling tasks with high communication cost into the same module reduces the performance loss caused by cross-module communication.

After the module partition is complete, the multi-core partition is performed. The task in the same module is assigned to different CPU for execution, and the processing efficiency of the task is improved by parallel execution [7-9].

2.1. Multi-Module Partition Algorithm

2.1.1 Improved KL Algorithm

KL algorithm is a dichotomous method based on graph. Based on the greedy principle, KL algorithm divides the known graph into two modules with minimal cost [1]. Starting from the initial solution, calculate all possible cost gains at a time, and choose the best of them to exchange [1]. Until the algorithm can't find a better solution from the current solution. The KL Algorithm is shown in Figure 1.

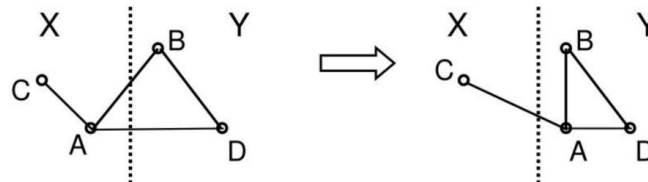


Fig. 1 An example of KL Algorithm [1]

The cost gain resulting from the swap is calculated as follows. A, B are two individual modules including several tasks. The communication cost between any two tasks i, j is defined as $Cost_{ij}$. For task $a \in A$, the internal communication cost is defined as $Cost_{in} = \sum Cost_{ai} (i \in A, i \neq a)$, the external communication cost is defined as $Cost_{ex} = \sum Cost_{aj} (j \in B)$; $D_a = Cost_{ex} - Cost_{in}$. Calculate D_a in the same way. Finally, can have cost gain $Gain_{ab} = D_a + D_b - C_{ab}$.

The procedure for executing the improved algorithm is as follows. Let the input be graph G . First, the elements in G are randomly assigned to two modules A and B . Then select the element pair $\{a_1, b_1\}$, $Gain_{a_1b_1} = \max(Gain_{ij}), i \in A, j \in B$. Then the process of selecting the maximum gain is repeated for $A - \{a_1\}, B - \{b_1\}$ until $\max(Gain_{ij}) = 0$. Resulting in the element pairs $\{a_1, b_1\}, \{a_2, b_2\}, \dots, \{a_n, b_n\}$, and the total gain $Gain_{sum} = \sum_{i=1}^n Gain_{a_i b_i}$. Swap all pairs of elements and start calculating the payoff $Gain_{sum}$ again until $Gain_{sum} = 0$. In the above steps, the execution time of the algorithm is reduced by reducing the process of the algorithm. The control flow diagram of the algorithm is shown in Figure 2.

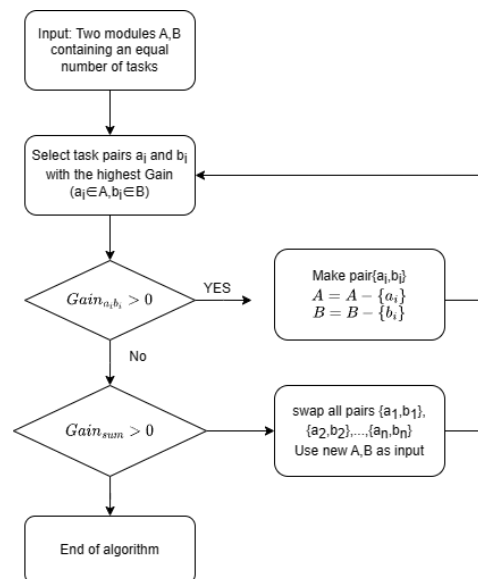


Fig. 2 The control flow diagram of improved KL Algorithm (Photo/Picture credit: Original)

Since the KL algorithm finds the local optimal solution, it can run the KL algorithm several times on a graph to select the partition with the best effect, to approach the global optimal solution.

2.1.2 Improved Clustering Algorithm

Clustering is to divide a data set into different classes according to a certain standard, so that the data in the same class is as similar as possible, and the data difference between different classes is as large as possible [2]. The communication cost is defined as the partitioning standard, and the smaller the communication cost is defined as the greater the difference [2]. After the clustering algorithm is executed, the elements with higher cross-module communication cost between each other will be divided into the same class [2]. Therefore, it is feasible to apply the clustering algorithm to multi-module partition and can effectively reduce the total communication cost. An example of clustering is shown in Figure 3.



Fig. 3 An example of clustering [2]

According to the three traditional clustering algorithms based on multi-module division, the author has made some improvements to make the algorithm more effectively divide the tasks with high cross-module communication cost into the same module, so as to reduce the total communication cost [2].

(1) Improved Single Link Clustering Algorithm

With the element set $D = \{t_1, t_2, \dots, t_n\}$, the adjacency matrix C of communication costs between elements. Then threshold $c = \text{Max}(C) + 1$, total number of target partitions k , current partition $K_c = \{\{t_1\}, \{t_2\}, \dots, \{t_n\}\}$ and the maximum value of the element EN within each partition. For each $K_i, K_j \in K_c$, there is $\text{larc}_{ij} = \text{Max}(C_{t_i t_j}), t_i \in K_i, t_j \in K_j$. If $\text{larc}_{ij} \geq c$, larc_{ij} is temporarily stored in a set Larcs [6].

After traversing once, select the two partitions $K_i, K_j, \text{larc}_{ij} = \text{Max}(\text{Larcs})$. Merge if $N_{\text{element}} \leq EN(N_{\text{element}} = N_i + N_j, N_i, N_j$ is the number of elements in K_i, K_j). Repeat the process of selection and merging until $N_{\text{element}} > EN$. The threshold value c is then subtracted by one for the next round of traversal and merge. If the number of currently divided groups $N_{\text{Groups}} = k$, the algorithm ends. If $c = 0$ but $N_{\text{Groups}} > k$, the Single Link Clustering Algorithm is invoked again with the current partition as the element set D . The control flow diagram of Single Link Algorithm is shown in figure 4.

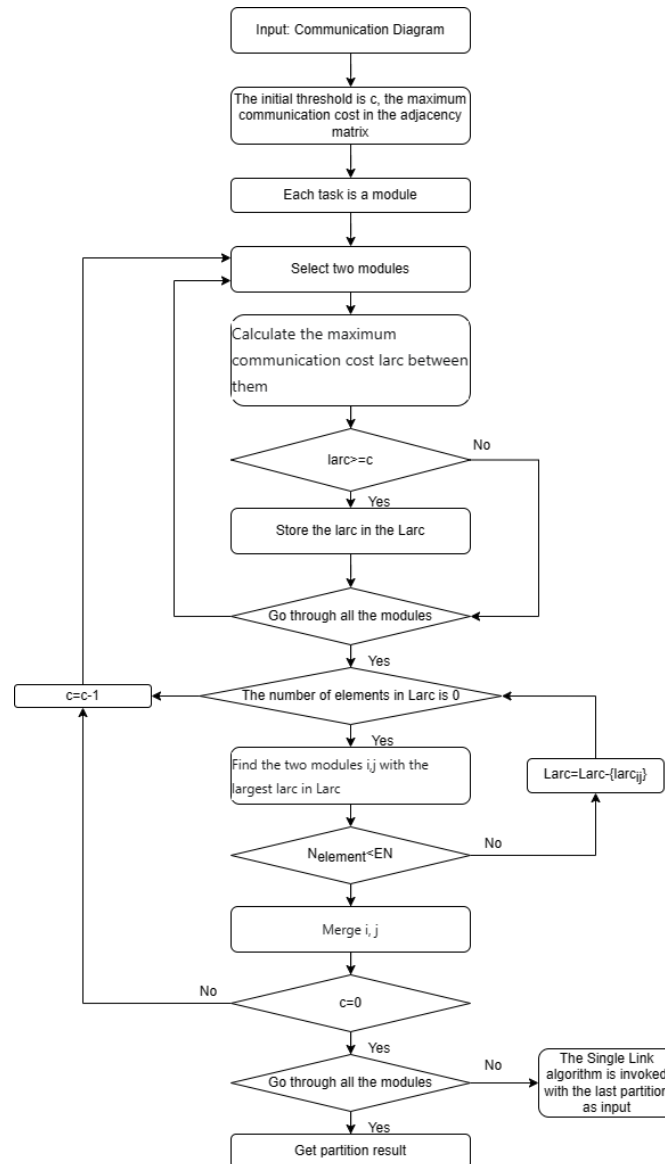


Fig. 4 The control flow diagram of Single Link Algorithm (Photo/Picture credit: Original)

(2) Improved All Link Clustering Algorithm

With the element set $D = \{t_1, t_2, \dots, t_n\}$, the adjacency matrix C of communication costs between elements. Then threshold $c = \text{Max}(C) + 1$, total number of target partitions k , current partition $K_c = \{\{t_1\}, \{t_2\}, \dots, \{t_n\}\}$ and the maximum value of the element EN within each partition. For each $K_i, K_j \in K_c$, there is $\text{Smac}_{ij} = \text{Min}(C_{t_i t_j}), t_i \in K_i, t_j \in K_j$. If $\text{Smac}_{ij} \geq c$, larc_{ij} is temporarily stored in a set Smacs [10].

After traversing once, select the two partitions $K_i, K_j, \text{Smac}_{ij} = \text{Max}(\text{Smacs})$. Merge if $N_{\text{element}} \leq EN (N_{\text{element}} = N_i + N_j, N_i, N_j \text{ is the number of elements in } K_i, K_j)$. Repeat the process of selection and merging until $N_{\text{element}} > EN$. The threshold value c is then subtracted by one for the next round of traversal and merge. If the number of currently divided groups $N_{\text{Groups}} = k$, the algorithm ends. If $c = 0$ but $N_{\text{Groups}} > k$, the All-Link Clustering Algorithm is invoked again with the current partition as the element set D . The control flow diagram of All Link Algorithm is shown in figure 5.

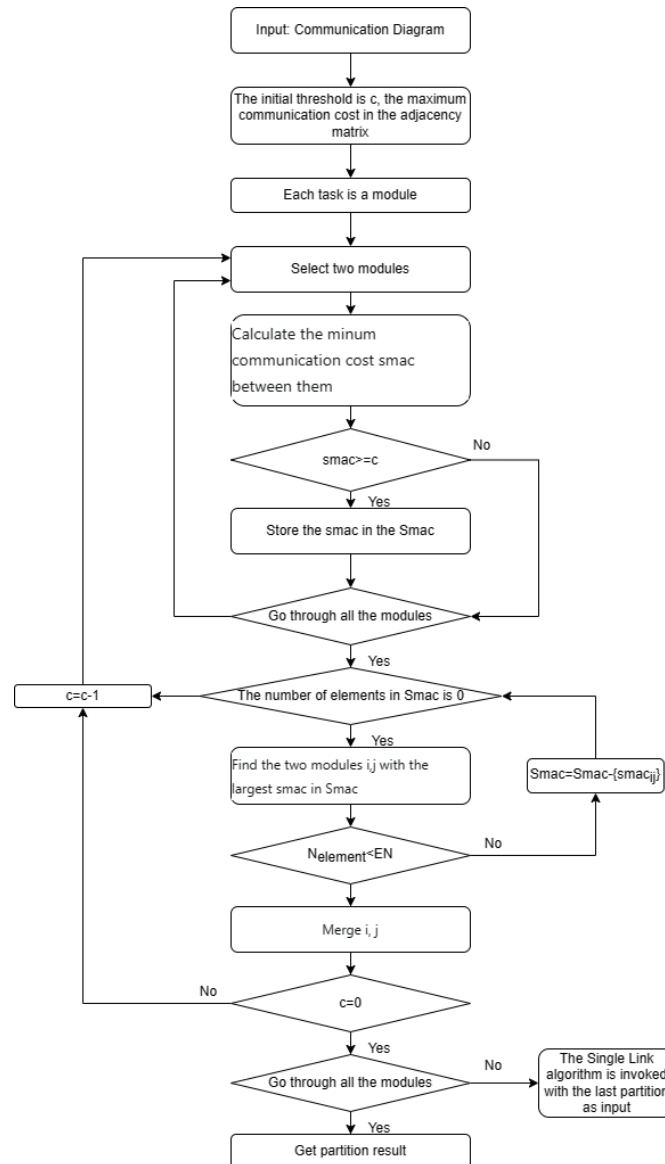


Fig. 5 The control flow diagram of All Link Algorithm (Photo/Picture credit: Original)

(3) Improved Average Link Clustering Algorithm

With the element set $D = \{t_1, t_2, \dots, t_n\}$, the adjacency matrix C of communication costs between elements. Then threshold $c = \text{Max}(C) + 1$, total number of target partitions k , current partition $K_c = \{\{t_1\}, \{t_2\}, \dots, \{t_n\}\}$ and the maximum value of the element EN within each partition. For each $K_i, K_j \in K_c$, there is $\text{avgc}_{ij} = \sum C_{t_i t_j} / (i * j)$, $t_i \in K_i, t_j \in K_j$. If $\text{avgc}_{ij} \geq c$, avgc_{ij} is temporarily stored in a set Avgcs .

After traversing once, select the two partitions K_i, K_j , $\text{larc}_{ij} = \text{Max}(\text{Avgcs})$. Merge if $N_{\text{element}} \leq EN$ ($N_{\text{element}} = N_i + N_j$, N_i, N_j is the number of elements in K_i, K_j). Repeat the process of selection and merging until $N_{\text{element}} > EN$. The threshold value c is then subtracted by one for the next round of traversal and merge. If the number of currently divided groups $N_{\text{Groups}} = k$, the algorithm ends. If $c = 0$ but $N_{\text{Groups}} > k$, the Average Link Clustering Algorithm is invoked again with the current partition as the element set D . The control flow diagram of Average Link Algorithm is shown in figure 6.

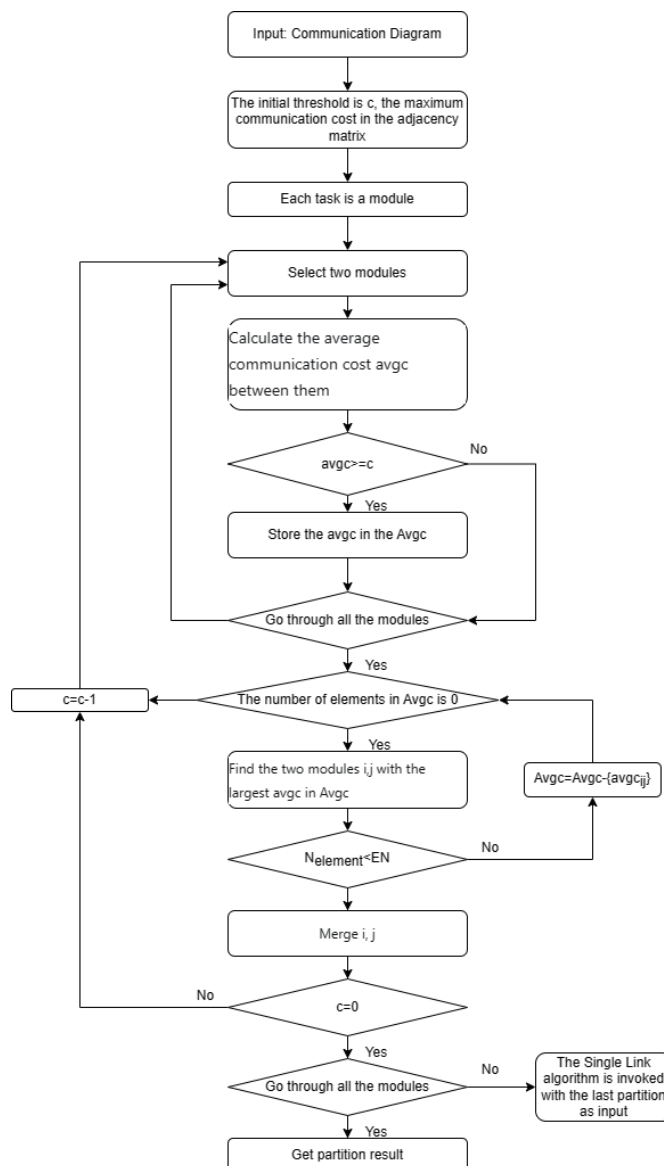


Fig. 6 The control flow diagram of Average Link Algorithm (Photo/Picture credit: Original)

2.2. Multi-core Partitioning Algorithm

Multi-core partitioning algorithm is an algorithm that allocates tasks to multiple CPU according to certain logic. By executing tasks in parallel, the total execution time of tasks is shortened, and the efficiency of task processing is improved. In the multi-core partition, it is necessary to pay attention to the dependencies between tasks and the release time [11, 12].

2.2.1 TaPSA Algorithm

TaPSA algorithm is a task priority table establishment algorithm proposed by Professor Chen Yixiang [12]. It takes the task priority value, release time, execution time and task serial number as inputs, and outputs a list of priority values.

The procedure for executing the algorithm is as follows [12]. First, the algorithm is sorted according to the priority values. If no priority values are equal, the algorithm ends. Otherwise, the tasks with the same priorities are sorted by release time. If the release time is the same, the tasks are sorted by execution time. If the execution time is the same, the task is sorted by task number.

2.2.2 Multicore Partitioning Algorithm MuPPA

MuPPA algorithm is a multi-core partitioning algorithm based on TaPSA algorithm [12]. Using the priority value table constructed by TaPSA algorithm as input, the multi-core partitioning method is output.

The algorithm is executed as follows [12]. First, check whether the precursor of the task is completed based on the task priority value table. If the task is completed, check whether the task release time is reached. If the release time has been reached, the task is assigned to the idle CPU with the smallest serial number.

3. Effect evaluation

3.1. Effectiveness Evaluation of Improved KL Algorithm

Mobile phone is one of the most common embedded systems at present, and the current mobile phone generally has multi-core architecture. People generally hope to reduce energy consumption as much as possible while improving the mobile phone running speed. Therefore, the mobile phone running optimization algorithm has research significance. Define such a cell phone that has four modules, each containing two cores. The operating system is responsible for scheduling tasks to different cores in different modules. All cores can run simultaneously. An example of a multi-core architecture is shown in figure 7.

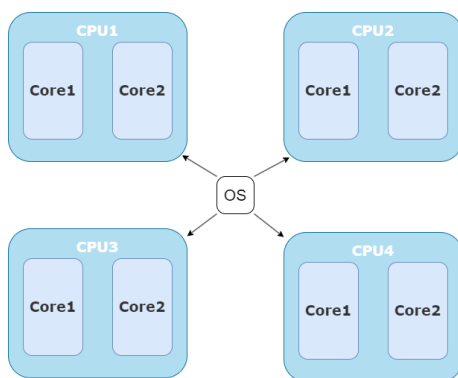


Fig. 7 An example of a multi-core architecture (Photo/Picture credit: Original)

Tasks are defined using a directed acyclic graph, as shown in the figure below. Nodes represent tasks, and directed edges represent dependencies between tasks [13]. A task cannot be executed until all its predecessor tasks have been executed. The weights on the sides represent the communication costs of tasks between different modules. An example of directed acyclic graph is shown in figure 8.

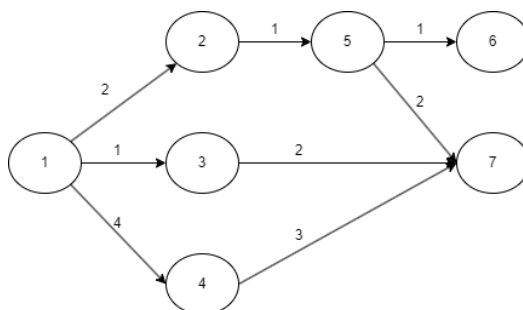


Fig. 8 An example of directed acyclic graph [13]

3.2. Algorithm effect evaluation

3.2.1 Effectiveness Evaluation of Improved KL Algorithm

As shown in the figure 9, the blue line represents the ratio of the execution time between the improved KL algorithm and the original KL algorithm, while the green line represents the ratio of the

partition result (that is, the total communication cost of the system) between the improved KL algorithm and the original KL algorithm [1]. Compared with the original KL algorithm, the execution time of the improved KL algorithm is doubled, and the final partition results are basically consistent with the original KL algorithm [1].

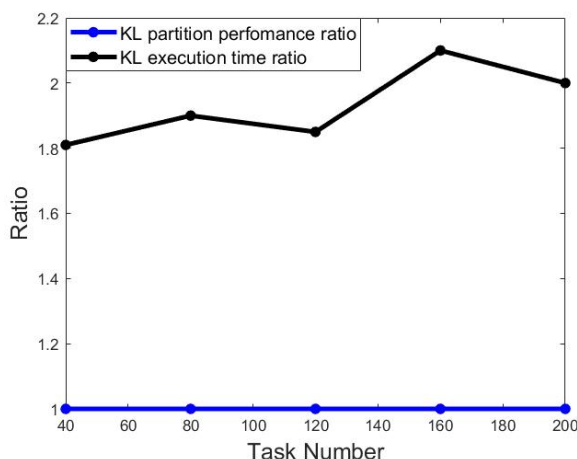


Fig. 9 Improve ratio of KL Algorithm (Photo/Picture credit: Original)

3.2.2 Effectiveness Evaluation of Improved Clustering Algorithm

As shown in the figure 10, the blue line is the communication cost ratio between the original single link algorithm and the improved algorithm, the orange line is the communication cost ratio between the original all link algorithm and the improved all link algorithm, and the yellow line is the communication cost ratio between the original average link algorithm and the improved average link algorithm. The improvement effect of the algorithm of all link and average link is more significant, reaching about 1.5 times of the original, while the improvement effect of single link is poor, only 1.2 times of the original

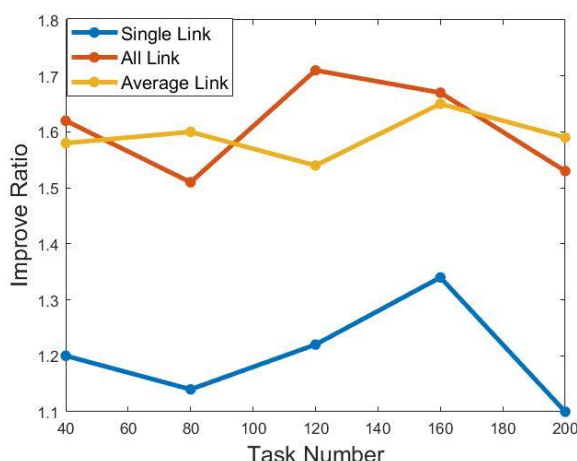


Fig. 10 Improve ratio of Clustering Algorithm (Photo/Picture credit: Original)

3.2.3 The Improved Effect of Multi-core Partition Algorithm

The improved effect of the multi-core partitioning algorithm is shown in the figure 11 below. The orange line is the multi-core scheduling algorithm with small serial number priority, and the blue line is the improved multi-core scheduling algorithm with long idle time priority. It can be observed that the improved task execution time is significantly shortened, and the effect becomes more significant with the increase of the number of tasks.

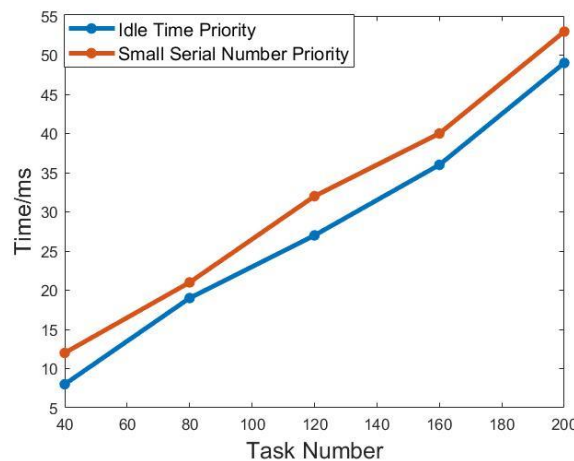


Fig. 11 Improve ratio of Multi-core Partition Algorithm (Photo/Picture credit: Original)

3.2.4 Overall Performance Evaluation

A method called communication cost ratio was used to evaluate the effectiveness of this work [14]. The communication cost ratio is the ratio of the total communication cost divided by an algorithm to the minimum communication cost divided by all algorithms. The communication cost ratio can effectively reflect the effectiveness of the algorithm.

The running results of the algorithm with different number of tasks are shown in the following figure 12. The lines in blue, orange, yellow and purple represent the communication cost ratio between Single link algorithm, All link algorithm, Average link algorithm and KL algorithm respectively. The effect ranking of the four algorithms from high to low is KL algorithm > Average link algorithm > All link algorithm > Single link algorithm,

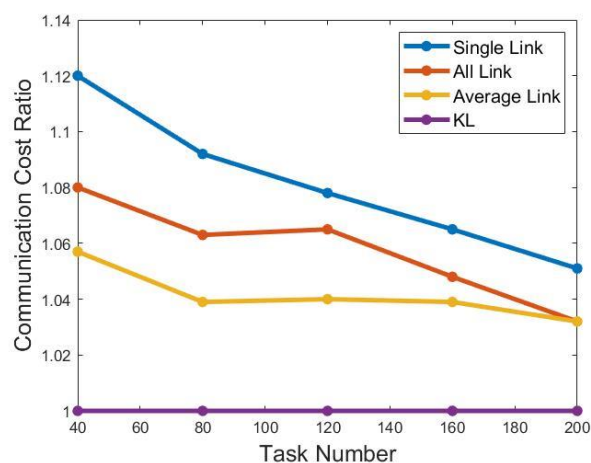


Fig. 12 Communication cost ratio of 4 algorithm (Photo/Picture credit: Original)

4. Summary

This paper introduces an optimization method applied to distributed embedded system, which improves the existing task scheduling algorithm and improves the performance of distributed embedded system with multi-module and multi-core more effectively. This algorithm takes the attributes and dependencies of the task as input, schedules the task to multiple modules by calling the module partitioning algorithm, and then performs multi-core partitioning, and finally gets the task partitioning in different CPUs in different modules. By reducing the number of cycles and performing more efficient merging, the result of module partitioning has a significant improvement in algorithm execution time and communication cost compared with the existing algorithms mentioned in the paper. Among them, the execution time of KL algorithm is the most significant improvement, and

the communication cost of clustering algorithm is the most significant improvement. From the partitioning results, the KL algorithm has the best partitioning effect, followed by the Average Link algorithm, then the All-Link algorithm, and the single link algorithm has the worst partitioning result.

At the same time, the improved algorithm proposed in this paper still has some shortcomings. From the view of task execution termination time, the optimization strategy proposed by this algorithm is not significantly improved compared with the existing algorithms. In the future, the method to reduce the termination time of task execution should be explored while the condition that the partition results remain unchanged. The division of hardware and software is introduced to make the distribution of hardware and software resources of distributed embedded system more balanced, to realize more effective use of resources.

References

- [1] Kernighan W B, Lin S. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 1970, 49(2): 291-307.
- [2] Sun Jigui, Liu Jie, Zhao Lianyu. Research on clustering algorithm. *Journal of Software*, 2008, 19(1): 48-61.
- [3] Stitt G, Vahid F, Nematbakhsh S. Energy savings and speedups from partitioning critical software loops to hardware in embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 2004, 3(1): 218-232.
- [4] Arató P, Mann Á Z, Orbán A. Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2005, 10(1): 136-156.
- [5] Govil N, Shrestha R, Chowdhury R S. PGMA: An algorithmic approach for multi-objective hardware software partitioning. *Microprocessors and Microsystems*, 2017, 54: 83-96.
- [6] Wu J, Srikanthan T, Lei T. Efficient heuristic algorithms for path-based hardware/software partitioning. *Mathematical and Computer Modelling*, 2009, 51(7): 974-984.
- [7] He Yanxiang, Luo Xianlin, Wu Si et al. Analysis of three typical Distributed task allocation Algorithms. *Minicomputer Systems*, 1997, (11): 2-7.
- [8] Han Dong, Liang Rubing. Research on multiprocessor allocation Algorithm in Distributed System. *Fujian Computer*, 2006, (07): 74-75.
- [9] Arafah B, Day K, Touzene A. A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems. *Journal of Systems Architecture*, 2007, 54(5): 530-548.
- [10] Cao Ying. Research and Implementation of Embedded System Task Scheduling Mechanism. *Dalian University of Technology*, 2010.
- [11] Xing Jiansheng, Wang Yongji, Liu Junxiang et al. A static minimum priority allocation algorithm. *Journal of Software*, 2007, (07): 1844-1854.
- [12] Xu Jinyi, Chen Yixiang, Li Kaixuan. Heterogeneous distributed optimization design method of the embedded system. *Journal of micro-nano electronic and intelligent manufacturing*, 2020, 2(01): 45-55.
- [13] Du Ming, Yang Anping, Zhou Junfeng, et al. K-step reachable-query optimization Algorithm on directed acyclic graphs. *Journal of Computer Applications*, 2019, 40(02): 426-433.
- [14] Chen Yixiang. Optimization Design of Intelligent Embedded System. *SCM and Embedded System Application*, 2022, 22(01): 6.