Send Orders for Reprints to reprints@benthamscience.ae

The Open Cybernetics & Systemics Journal, 2014, *8*, 835-842

Open Access

Research on the Possibility of Regulating Scheduling Length after Eliminating Intra-Iteration Dependency

Wu Huixin[1,*], Feigao Li[2] and Nan Sun[3]

[1]*Department of Information Engineering, North China University of Water Resources and Electric Power, Zhengzhou, 450046, China;* [2]*Department of Electrical Engineering, Henan Polytechnic, Zhengzhou, 450046, China;* [3]*College of Computer and Information Engineering, Henan University of Economics and Law, Zhengzhou, 450002, China.*

Abstract: Recently, scheduling algorithm which is related to computing tasks and communication transactions is widely studied. In multi-core systems, adopting certain scheduling algorithm to execute schedule can decrease scheduling length effectively after eliminating intra-iteration dependency of the computing tasks. In this case, regulating execution order of computing tasks can make further compression of the scheduling length in consideration of the effect generated by communication tasks. The experiment proved the necessity of decreasing scheduling length by regulating execution order of computing tasks according to the specific situation after achieving the initial scheduling length by adopting certain scheduling algorithm without the intra-iteration dependence of computing tasks. This study has great significance for the design of scheduling algorithm in the multi-core environment.

Keywords: Multi-core, scheduling length, task scheduling, task execution order.

1. INTRODUCTION

In recent years, scheduling algorithm which contains computing tasks and communication transactions has obtained extensive research. For cycle application running on multi-core systems, dependency of computing tasks affects the parallelism, which reduces the utilization rate of multi-core [1-3]. Dependency of computing task contains intra-iteration dependency and extra-iteration dependency [4]. Only intra-iteration dependency will produce the fore mentioned affection, but extra-iteration dependency will not. Therefore the researchers start to consider eliminating intra-iteration dependency of computing tasks [5], and carrying out a large number of scheduling algorithm designs based on the elimination of data dependency of intra-iteration [6-9]. Computing task execution order can be regulated liberally after eliminating the intra-iteration dependency of the computing tasks. The adjustment of computing task execution order may affect scheduling length especially when considering the communication transactions. After eliminating intra-iteration of computing tasks, whether the adjustment of computing task execution order can reduce the scheduling length is a problem worth studying.

2. RELEVANT DEFINITION AND MODELLING

When infra-iteration dependency of computing tasks is eliminated, the definition of scheduling length and the scheduling length of data dependency with data iteration is different. That is because communication happens in different iteration of the loop. The definition of scheduling length without infra-iteration data dependency will be given in the following section.

Definition 1: On considering scheduling of communication tasks, scheduling length is the latest finish time of all the computing tasks and communication transactions in the task graph.

Before the study on computing task execution order, directed acyclic graph standing for application program will be introduced first: $G=(\Gamma,E,D,C)$, Γ represents task set, E represents the set of directed edge $e_{ij}(i,j\in n)$, W represents the set of weight of direct edge, that is to say w_i represents the execution time of computing task τ_i. $c_{ij}\in C$ represents the communication time between computing task τ_i and τ_j. Fig. (**1**) shows one task graph instance. In Fig. (**1**), nodes represent computing tasks and the numbers beside the computing tasks represent the number of clock cycles required by executing computing tasks. The directed edge connecting two computing tasks represents the communication edge of the two computing tasks. The computing task from the communication edge are father computing task (father node), the computing task pointed by the arrows from the communication edge are child computing task(child node), the tagged numbers beside the computing tasks represent the number of clock cycles required by executing communication. The directed edge connecting two computing tasks represents the communication edge of the two computing tasks. For the situation two computing tasks from connection of communication edge are assigned to the same processing core, the communication time between the two computing

1874-110X/14

tasks will be zero. In the multi-core system, if intra-iteration dependency of computing tasks are eliminated and the original computing tasks with intra-iteration dependences map to the processing core in accordance with certain scheduling algorithm, then scheduling length could be the time of the last one computing task finished in all processing core, or also could be the time of the last one communication task finished on the bus. This is because once the intra-iteration dependency of computing tasks are eliminated, the finish time of communication tasks and computing tasks may overlap, or the finish time of communication tasks may be later than the final completion time of computing tasks. Thence scheduling length can be calculated by the following formula:

$$makespan = \max(\max(finish(\tau_i, p_k)), \max(finish(e_{ij}))) \quad (i \in 1,2,\cdots n, j \in 1,2,\cdots n, k \in 1,2,\cdots m) \tag{1}$$

makespan --scheduling length.

$finish(\tau_i, p_k)$ --the finish time of computing task τ_i on processing core p_k

$finish(e_{ij})$ --the completion time of communication transaction e_{ij}

m --the number of the processing core

The completion time used by computing task τ_i on the processing core p_k is the sum of the start time and execution time of this task on the processing core p_k, and is expressed as follow.

$$finish(\tau_i, p_k) = start(\tau_i, p_k) + w_{ik} \tag{2}$$

$start(\tau_i, p_k)$ --start time

w_{ik} --execution time

The completion time $finish(e_{ij})$ of communication transaction e_{ij} is the sum of the start time and execution time of this communication transaction.

$$finish(e_{ij}) = start(e_{ij}) + c_{ij} \tag{3}$$

$start(e_{ij})$ --start time

c_{ij} --communication time

3. BACKGROUND KNOWLEDGE AND PLATFORM INTRODUCTION

3.1. Principle of Retiming Technique

Retiming technology principle is put forward by Leisers in 1991 and this technique was originally used to optimize sequential circuits [10]. While keeping the original function elements and their connection way invariable, It will remove delays from each of the input side of one node and add to each of the output side of the node by rearranging the registers or in turn so that circuits can be optimized. Later this technology is brought in parallelization and schedule of embedded system [11, 12] to optimize data flow diagram gained by the abstraction of application program. Actually its essence of the improvement of program parallelism is though reassembling the loop body, and making dependency of computing tasks in the original task graph exist in different iterations. This technology achieves the purpose of optimizing loop by rearranging the delays under the situation of keeping the tasks of the original data flow diagram and dependency among iterations invariable. This technology has eliminated the intra-iteration dependency of the original data flow diagram. The loop body can be optimized according to the retiming data flow diagram after the use of this technology on data flow diagram. In the retiming data flow diagram, if there is no delay at the directed edge which connects two nodes, the intra-iteration data dependency will represent the relationship the two computing tasks. Two extra-iterations existed in different loops will be represented by the edge with delay, " | "which connects two different nodes represents delay, the number of " | " represents how many cycle times of discrepancy between the two nodes connected on one edge. For instance, the number of delay in the edge that connects computing task τ_i and τ_j is $\rho(e_{ij}) > 0$, showing the result produced by the computing task τ_j at m-th loop depends on the result produced by the computing task τ_i at $(m - \rho(e_{ij}))$-th loop.

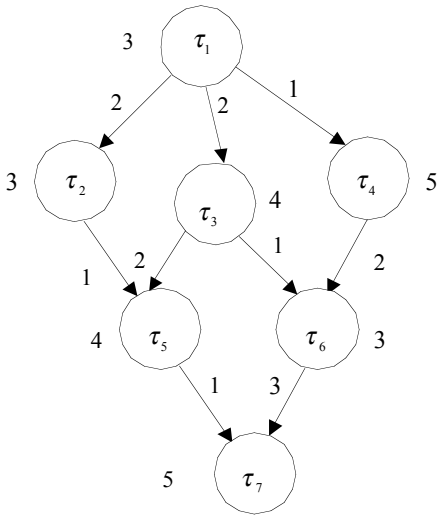

Fig. (1). An example of task graph.

Fig. (**2a**) provides a loop program. Fig. (**2b**) provides the relevant data flow diagram G of this loop program (one node represents one computing task). The tagged number beside each computing task represents the periodicity of this computing task, here assuming that add operation needs one clock cycle, multiply operation requires two clock cycles. Fig. (**2c**) provides one deformational loop program. Fig. (**2d**) provides the relevant retiming data flow diagram G_r. In this retiming data flow diagram, the retiming value of node A is 3, the retiming value of node B is 2, the retiming value of node C is 1, the retiming value of node D is 2, and the retim-

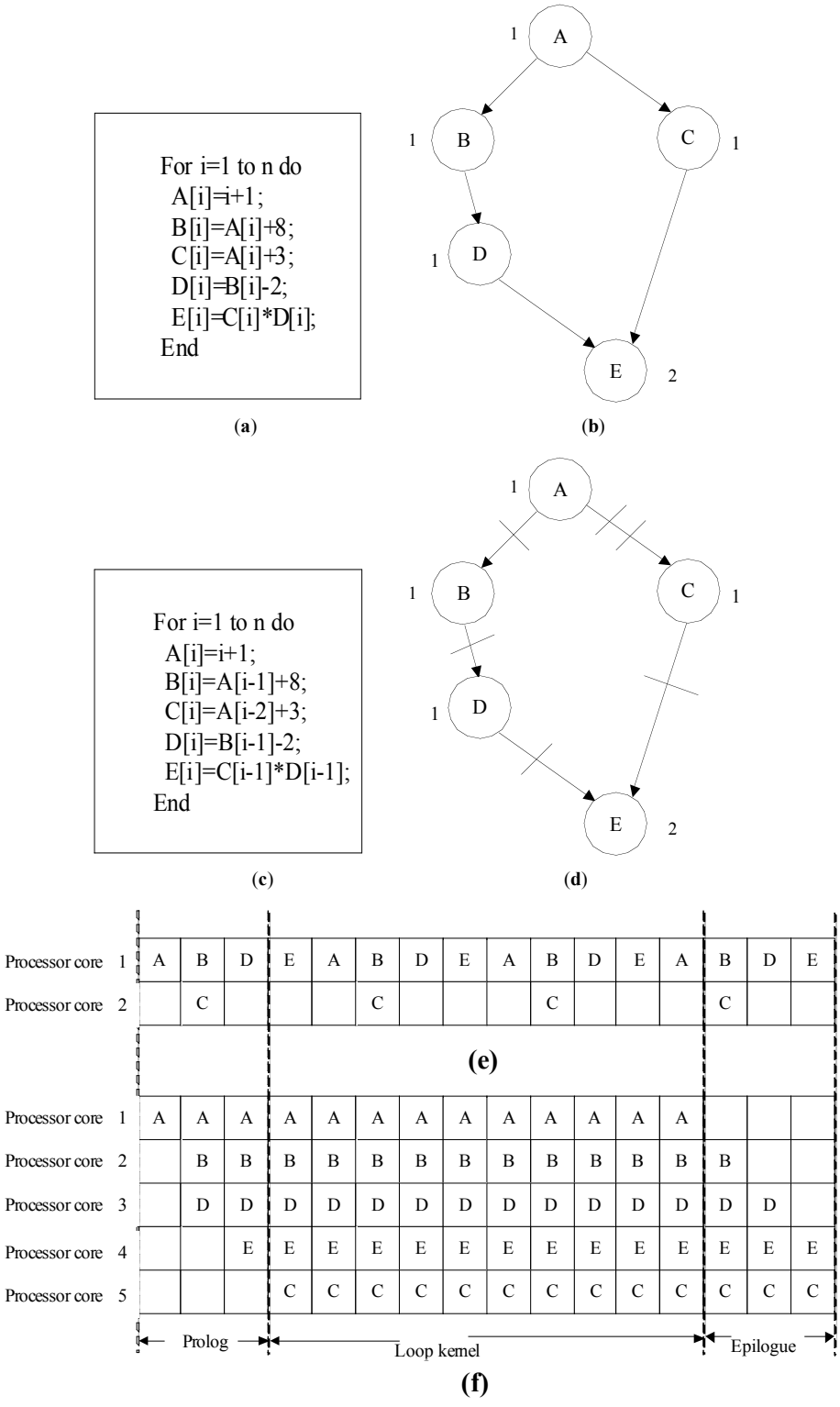

Fig. (2). (**a**) A loop program, (**b**) The relevant data flow graph G, (**c**) The deformational loop program, (**d**) The relevant retiming data flow graph, (**e**) The scheduling result obtained by data flow graph G, (**f**) Assembly line scheduling result gained though retiming data flow diagram G_r.

ing value of node E is 0. Fig. (**2e**) provides the scheduling result gained though the data flow diagram G. Fig. (**2f**) provides the assembly line scheduling result gained though the retiming data flow diagram G_r.

3.2. RDAG

RDAG algorithm is designed based on retiming principle whose aim is to obtain computing task retiming value [4]. This algorithm can transform a periodic acyclic graph into a

retiming data flow diagram effectively. In the retiming data flow diagram, each node has the smallest retiming value so that the intra-iteration dependency among tasks can be eliminated effectively. Formula (4) gives the calculation formula of retiming values.

$$r[\tau_i]=\begin{cases}\max\{r[\tau_i],r[\tau_j]+1 & \text{if } \hat{o}_i \text{ is parent node of } \tau_j \\ 0 & \text{if } \hat{o}_i \text{ is a leaf node}\end{cases} \quad (4)$$

The basic idea of RDAG algorithm is that: first the retiming value of each node will be initialized to zero. Then all of the leaf nodes found from the original task graph will be put in one array named after Q. Link list tail of Q is stored in variable named after tail. After that every time one node τ_j will come out from array Q, for the father node τ_i of the node τ_j ,its retiming value can be computed according to the formula(4). Finally, determine whether τ_j is tail node, if it is not, τ_i will be put in the array again, but if it is, the node τ_i will be seemed as the link list tail. When the queue is empty, the corresponding retiming task graph can be obtained. In this algorithm, the calculation process of retiming is carried out in breadth first manner.

3.3. List Scheduling Algorithm

Scheduling of directed acyclic graph in multiprocessor/multi-core system is a very complicated problem. Heuristic scheduling algorithm is a good solution. List scheduling algorithm is one kind of classic static heuristic scheduling algorithm. Lots of heuristic scheduling algorithms are designed on the basis of list scheduling algorithm. The basic idea of list scheduling algorithm is that a priority for each node which can form a scheduling list will be arranged. In this scheduling list, computing tasks are stored according to the priority descending order [13]. There are many ways to decide the priority of nodes such as HLF (Highest Level First), LP (Longest Path), LPT(Longest Processing Time), CP (Critical Path) and so forth. The scheduling of list scheduling algorithm will be completed in compliance with the following two rules [14].

(1) Get one node out in accordance with the deposited order from the scheduling graph;

(2) Map this node to the processing core which can make this node have the earliest start time.

This is the executive process of the traditional list scheduling algorithm and scheduling list cannot be changed once formed in the traditional list scheduling algorithm. In other words, the priority of tasks is the pre-determined static priority. With the further study of scheduling algorithm, some improved list scheduling algorithms based on dynamic scheduling list have been put forward. In these algorithms, after finishing the mapping of each node, the priority of the nodes that have not scheduled will be recomputed, then the next node will be chose according to the new priority, and this process will not stop until every node is scheduled [14].

The list scheduling algorithm in the modified TORSHE in this paper is based on the maximum completion time priority. The aim of this priority method is to ensure that the maximum completion time is minimal. In this paper, after eliminating the intra-iteration dependency, whether the analysis of adjustment of computing task execution order can reduce the scheduling length is suitable for the scheduling result gained by any scheduling algorithms.

3.4. Platform Introduction

Multi-core embedded system can be divided into homogeneous multi-core embedded system and heterogeneous multi-core embedded system based on chip processing core style. The method proposed in this paper is applicable to not only homogeneous multi-core embedded system but also heterogeneous multi-core embedded system. The system studied in this paper uses the shared bus as communication architecture. All of the processing cores share one bus. In this kind communication architecture with multi-core system, the data sent by computing tasks will possess the bus exclusively. Therefore, it is necessary to set a bus arbiter to distribute the use right of the bus when the tasks running on different processing cores apply for the bus simultaneously. The structure of the bus in this paper is as same as the one in literature [15].

The principle of bus arbitration in this paper is similar to the principle in literature [16]:

(1) If there is no computing tasks possessing the bus, and at this time a computing task applying for the use right of the bus can gain right to use the bus.

(2) If one computing task has gained the right to use the bus, and other tasks apply for the bus at the same time, the bus arbiter will not response until the data transmission is finished.

(3) If there are multiple computing tasks applying for the right to use the bus at the same time, the bus arbiter will distribute the use right to the task with high priority.

4. RELATED WORKS

Due to the elimination of intra-iteration data dependency, the parallelism and performance can be improved, besides that, the energy consumption also can be reduced. Therefore the scheduling algorithm without intra-iteration has aroused wide concern. Literature [17] proposes a scheduling algorithm which can meet the double restrictions of time and resource. With the computing task as the scheduling object, this algorithm adopts main loop scheduling strategy to carry out the retiming operations for other nodes waiting for being scheduled and form the channel without feedback without violating the time constraints. Literation [18] proposes a method that can realize implicit retiming with cycle rolling so that a transfer under the resource constrain can be realized though rolling cycle. This scheduling algorithm in the service of computing tasks can compress scheduling cycle and choose a best scheme from a variety of scheduling scheme obtained. Considering the voltage conversion overhead and dynamic power consumption, Literature [19] proposes a scheduling algorithm which can reduce power consumption though combining the technology of dynamic voltage scaling and cyclic rotation scheduling. For the purpose of optimizing the energy consumption and performance simultaneously, Literature [4] puts forward a two-stage algorithm based on

overhead perception. In the first stage, intra-iteration dependency will be eliminated by loop body deformation. Then in the second stage, task scheduling and voltage selection will be adjusted iteratively with spring algorithm [4]. This method synthesizes dynamic voltage scaling, dynamic power consumption and software pipelining technology to achieve the power consumption optimization in multi-core embedded system. To optimize computing tasks and communication transactions on multi-core system, literature [6] uses retiming technology to adjust computing tasks and communication transactions simultaneously which provides the possibility of minimizing the scheduling length. Literature [20] introduces the real-time loop scheduling technology which reduces energy consumption by dynamic voltage scaling. The technology includes IDVS algorithm and DVLS algorithm. With IDVS algorithm conversion cost can be included in the optimization scheduling scheme. And based on rotation scheduling, DVLS algorithm reorganizes the loop body repeatedly to achieve the purpose of reducing energy consumption as much as possible within the given time limit.

It can be seen from the above analysis that the research into computing tasks or communication transactions respectively has been done for long, but there is less done on the study of both aspects at the same time. Besides, the research results are just confined to the homogeneous multicore system. However the research result proposed in this paper is suitable for not only homogeneous multicore system but also heterogeneous multi-core system.

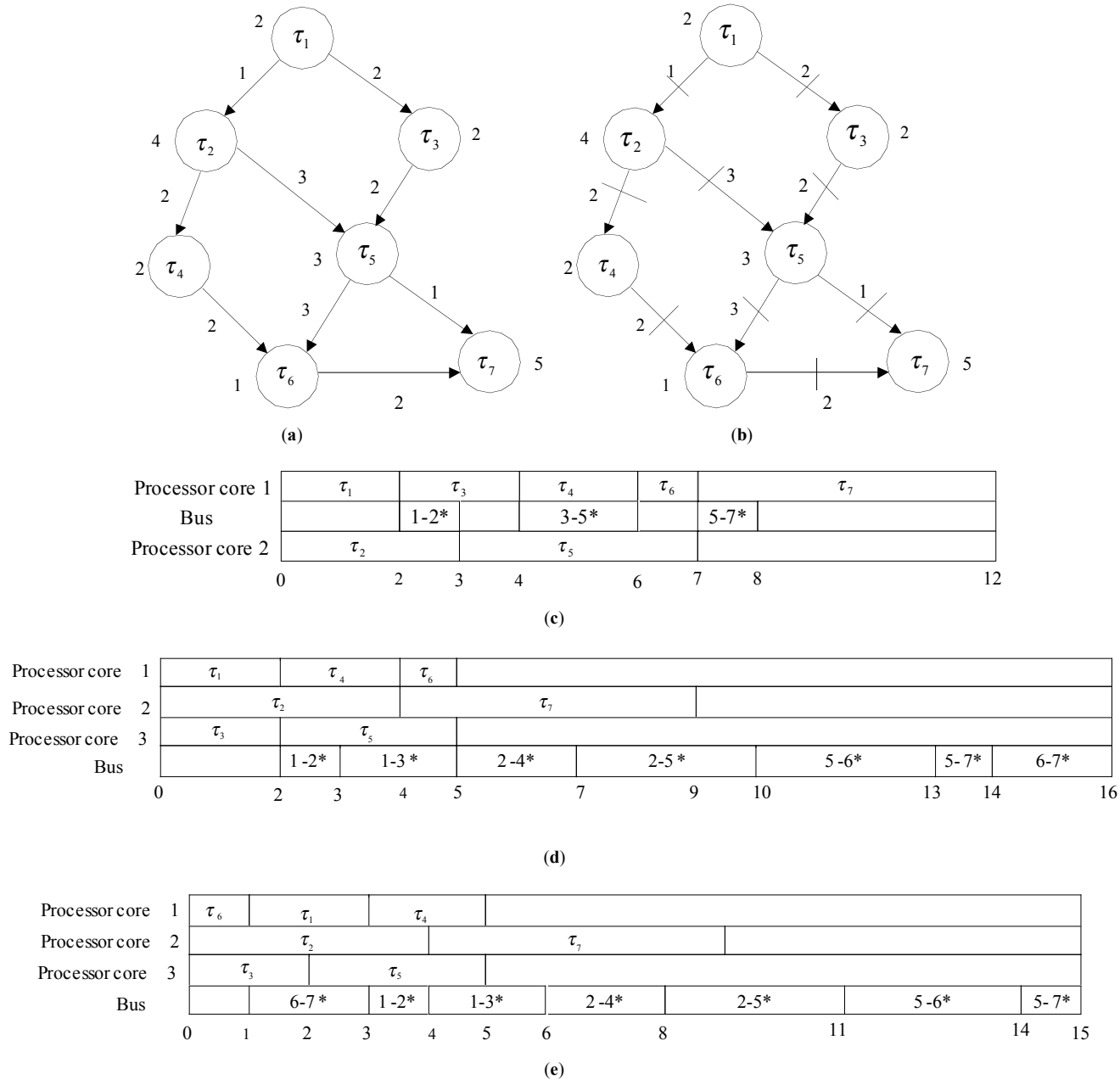

Fig. (3). (**a**) The task graph with 7 computing tasks. (**b**) The corresponding retiming task graph. (**c**) The scheduling result on multi-core system with 2 processing cores. (**d**) The scheduling result on multi-core system with 3 processing cores. (**e**) The scheduling result on multi-core system with 3 processing cores after adjusting the execution order of computing tasks.

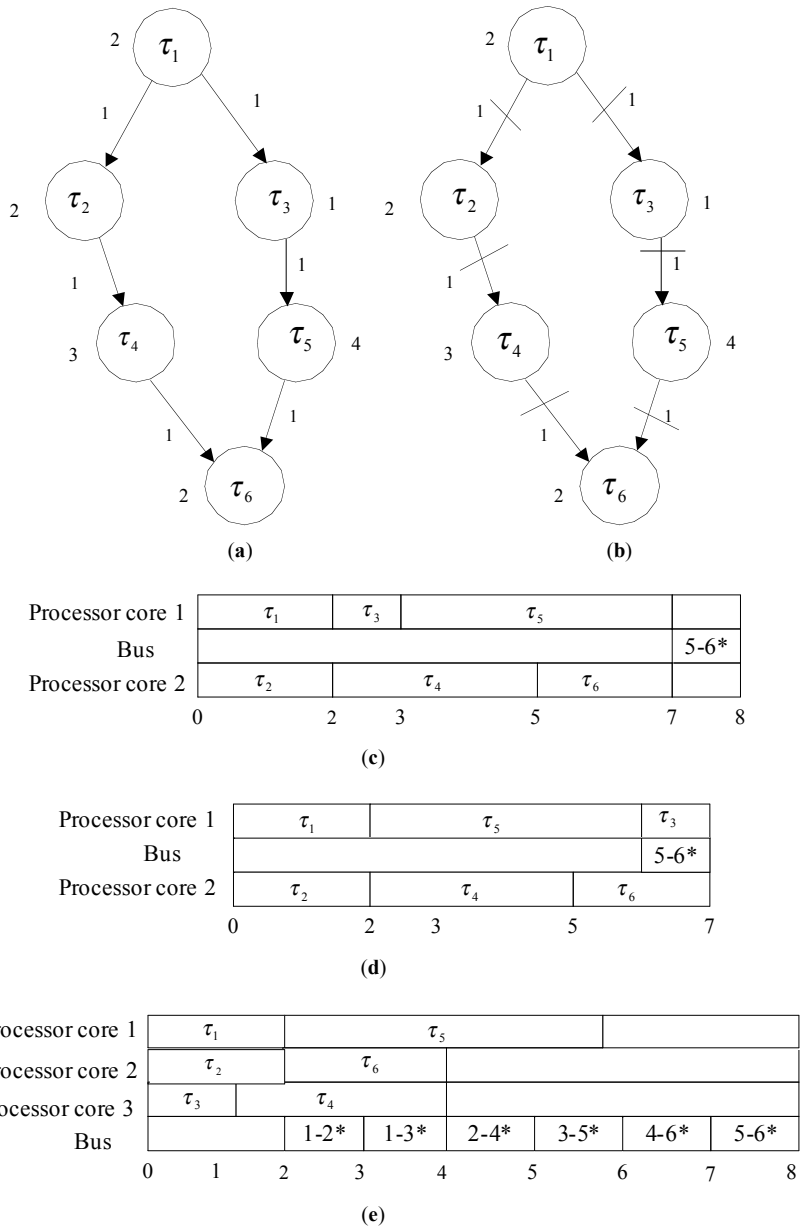

Fig. (4). (**a**) The task graph with 6 computing tasks. (**b**) The corresponding retiming task graph. (**c**) The scheduling result on multi-core system with 2 processing cores. (**d**) The scheduling result on multi-core system with 2 processing cores after adjusting the execution order of computing tasks. (**e**) The scheduling result on multi-core system with 3 processing cores.

5. CHOICE OF SCHEDULING LENGTH AND DETERMINATION OF TASK PRIORITY

For the application which contains dependency computing tasks, the most popular multi-core scheduling algorithms usually adopt list scheduling strategy, heuristic algorithm and so forth to obtain the scheduling scheme and determine the priority of the computing tasks. For the tasks eliminating intra-iteration dependency, there are no more free time slots existing in the processing core, but the communication between the computing tasks of the father node and the child node in different iteration still exists because of extra-dependency existing among the computing tasks. It is possible that the scheduling length is the finish time of the last communication transaction. For the same application with the intra-iteration dependency, the scheduling result can make sure that the optimized scheduling length can be obtained within the time limit. But after eliminating the intra-iteration dependency among the computing tasks, it is necessary to improve the scheduling result and adjust the execution of the computing tasks to obtain the optimized scheduling length within the time limit. In this paper, we set the initial priority of the computing tasks the same with the priority of the list scheduling algorithm in TORSCHE. After obtaining the initial mapping with list scheduling algorithm, in order to make sure the scheduling length is the shortest, it is necessary to adjust the execution order of the computing tasks. When multiple computing tasks apply for the use right of bus at the same time, the execution order of computing tasks gained by breadth traversal will determine which communication transaction has the right to use the bus. For

the communication transactions sent by the same computing task, the use right of the bus is determined bases on the breadth traversal too.

6. EXPERIMENTAL RESULTS

In this paper, we mainly focus on the two task graphs based on Gaussian elimination mentioned in literature [23, 24] and transform the list scheduling algorithm proposed in literature [22] based on the maximum completion time method, and then install the modified TORSCHE on the MATLAB software environment. Firstly, we get the corresponding retiming task graphs of this two task graphs using the method in literature [4] and obtain a loop scheduling result on a respective multi-core system with 2-3 processing cores with the modified list scheduling algorithm. Then, we will analyze the scheduling result and find out whether the scheduling length can be compressed further by adjusting the execution order of the computing tasks. In the following experiment, we mainly discuss impacts of execution order on scheduling length. Fig. (**3a**) shows a task graph with 7 computing tasks. For this task graph, firstly use the retiming technology to get its corresponding retiming task graph and then the scheduling result with the revised list scheduling algorithm. Fig. (**3b**) is the retiming task graph of this task graph after eliminating the intra-iteration dependency. Fig. (**3c**) shows the scheduling result for the retiming task graph on the multi-core system with 2 processing cores. From Fig. (**3c**), we can see that the scheduling length cannot be compressed by adjusting the execution order of computing tasks. From Figs. (**3d**) and (**3e**), we can see that the scheduling length can be compressed by adjusting the execution order of computing tasks on the multi-core system possessing 2 processing cores. For the task graph with 6 computing tasks, we can obtain the scheduling result by its corresponding retiming task graph. Fig. (**4a**) shows a task graph with 6 computing tasks. Fig. (**4b**) is the retiming task graph of the task shown in Fig. (**4a**) after eliminating the intra-iteration dependency. Fig. (**4c**) shows the scheduling result with the revised list scheduling algorithm on the multi-core system with 2 processing cores. Fig. (**4d**) shows the scheduling result after adjusting the execution order of the computing tasks. From Fig. (**4c**) and Fig. (**4d**), we can see that the scheduling length can be compressed by adjusting the execution order of computing tasks. However, for the multi-core system with 3 processing cores, the scheduling length cannot be compressed by adjusting the execution order of computing tasks (shown in Fig. (**4e**)).

CONCLUSION

In this paper, we mainly discuss the influence of computing task execution order on scheduling length on the multi-core systems. We respectively test the task graph with 6 computing tasks and 7 computing tasks on the multi-core systems with 2-3 processing cores. The experimental result shows that when considering communication transactions, we should analyse the necessity for adjusting the execution order of computing tasks to reduce scheduling length in view of different variable characteristics. The conclusion of this article is of great significance to optimize regulating scheduling length after eliminating intra-Iteration dependency on the multi-core systems.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work is supported by National Nature Science Foundation of China (No.61171051), Science and Technology Innovation Team Foundation of Zhengzhou city (No. 131PCXTD595).

REFERENCES

[1] LIU Wei, YIN Hang, DUAN Yu-Guang, DU Wei, WANG Wei, ZENG Guo-Sun, "Adaptive Threshold-Based Energy-Efficient Scheduling Algorithm for Parallel Tasks on Homogeneous DVS-Enabled Clusters," *Chinese Journal of Computers*, (02) ,pp.393-407, 2013.

[2] Li DaiPing, Luo ShouWen, Zhang XinYi, Fang HaiXiang, "Research of Parallel Task Partition Strategy on Grid," *Application Research of Computers*,(10),pp.80-82, 2005.

[3] Wang Qianjin, Gao Yong, Li Cunhua, "Research on Multi-Core-Based Multitask Parallel Processing Technology," *Computer Applications and Software*, 29(7), pp.141-143, 2012.

[4] Liu Hui, Shao Zili, Wang Meng, et al, "Overhead-Aware System-Level Joint Energy and Performance Optimization for Streaming Applications on Multiprocessor Systems-on-Chip," *Euro micro Conference on Real-Time Systems*, pp.92-101, 2008.

[5] ZHANG Lei, TAO Bin-xian, QIAN Ju, "Using Points-to Combinations to Optimize Dependence Graph Construction," *Computer Science*,(01), pp.139-143. 2013.

[6] Wang Yi, Liu Duo, Wang M, "Optimal Task scheduling by removing inter-core communication overhead for streaming applications on MPSoC," *IEEE Transactions on Computers*, 62(2), pp.336-350, 2013.

[7] Liu D, Wang Y, Shao Z, *et al.* "Optimally Maximizing Iteration-Level Loop Parallelism," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 23(3), pp.564-572.

[8] Wang Y, Liu H, Liu D, *et al.*, "Overhead-Aware Energy Optimization for Real-Time Streaming Applications on Multiprocessor System-on-Chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16(2),pp.1-32, 2011.

[9] Zhang Jun, Deng Tan, Gao Qiuyan, *et al.* "Optimizing Data Placement of Loops for Energy Minimization with Multiple Types of Memories," *Signal Processing Systems*, 72(3), pp.151-164, 2013.

[10] Leiserson C E, Saxe J B, "Retiming synchronous circuitry," *Algorithmica*, (6): 5-35, 1991.

[11] Chao L F, LaPaugh A S, Sha E H M, "Rotation scheduling: A loop pipelining algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems(TCAD)*, 16(3), pp.229-239.

[12] ZHOU Ben-hai, QIAO Jian-zhong, LIN Shu-kuan, "Research on Parallel Scheduling Algorithm of Task Graph Model on Multi-core Processing Platform," *Mini-micro Systems*,(11), pp.2485-2492, 2012.

[13] Wang Li-Zhe, von Laszewski Gregor, *et a1*. "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. Melbourne, Australia, 368-377, 2010.

[14] Zamani R, Afsahi A, Qian Y, Hamacher C, "A feasibility analysis of power awareness and energy minimization in modern interconnects for high performance computing," *Proceedings of the 9th IEEE International Conference Cluster Computing* (Cluster 07). Austin, USA, pp.l18-128, 2007.

[15] Zhang Fa, Antonio Fernandez Anta, Wang Lin, Hou ChengYing, Iiu Zhi Yong, "Network energy consumption models and energy efficient algorithms," *Chinese Journal of Computers*, 35(3), pp.603 - 615, 2012.

[16] Orso A, Sinha S, Harrold M J, "Classifying data dependences in the presence of pointers for program comprehension, testing, and debugging." *ACM Transactions on Software Engineering and Methodology*, 13(2), pp.199-239, 2004.

[17] Yang Dayu, Lin Zhenghui, "A Fast Loop Pipelining Algorithm," *JOURNAL OF SHANGHAI JIAOTONG UNIVERSITY*, (12), pp.1717-1720, 2002.

[18] Xu Bao-wen, Qian Ju, Zhang Xiao-fang, *et a1*., "A brief survey of program slicing," *ACM SIGSOFT Software Engineering Notes*, 30(2), pp.10-45, 2005.

[19] Godefroid P, Nori A V, Rajamani S K, et a1, "Compositional may must program analysis unleashing the power of alternation," *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 43-56, 2010.

[20] Shao Z, Wang M, Chen Y, *et al.*, "Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems," *IEEE Transactions on Circuits and Systems Ⅱ* (TCAS-II), 54(5), pp.445-449, 2007.

[21] Hua G, Wang M, Shao Z, Liu H, *et al.* "Real-Time Loop Scheduling with Energy Optimization *Via* DVS and ABB for Multi-core Embedded System," *International Conference on Embedded and Ubiquitous Computing*, pp.1-12, 2007.

[22] http://rtime.felk.cvut.cz/scheduling-toolbox/.

[23] Koji Hashimoto, Tatsuhiro Tsuchiya, Tohru Kikuno, "Fault-Secure Scheduling of Arbitrary Task Graphs to Multiprocessor Systems," *International Conference on Dependable Systems and Networks*, pp.1-10, 2000.

[24] Zong Ziliang, Qin Xiao, Xiaojun Ruan, *et al*. "Energy- Efficient Scheduling for Parallel Applications Running on Heterogeneous Clusters," *the 36th International Conference on Parallel Processing*, pp.1-8, 2007.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014