計算機アーキテクチャ 134－27
（１９９９．８．４）

並列度に依存しないスケジューリング

Kirilka Vassileva Nikolova, 前田敦司, 曽和将容
電気通信大学 大学院 情報システム学研究科

概要: 既存の静的スケジューリングアルゴリズムの多くは、特定の(固定された)プロセッサ数を前提にタスクをスケジュールする。本稿では Parallelism-Independent Scheduling と呼ぶ、並列度が異なる計算機上においても最適に近い実行時間が得られる手法を提案する。それぞれ次の３つのフェーズからなる２つの手法を提案する: リストスケジューリング、同一レベル内でのタスク順序入れ替えによる最適化、スケジュールの直列化。効果を実証するため、ランダムな DAG を用いたシミュレーションを行ない、Critical Path Algorithm を各並列度ごとに適用した場合と結果を比較した。

Parallelism-Independent Scheduling

Kirilka Vassileva Nikolova, Atusi Maeda and Masahiro Sowa
Graduate School of Information Systems, University of Electro-Communications
Chofugaoka 1-5-1, Chofu-shi, Tokyo 182, Japan
Phone: (+81) 424-43-5635 Fax: (+81) 424-43-5851
e-mail: {nikol, maeda, sowa}@sowa.is.uec.ac.jp

Abstract: Most existing static scheduling algorithms order the tasks for one fixed number of processors. In this paper we propose a new scheduling method, called Parallelism-Independent Scheduling Method which enables the scheduled program to be executed efficiently on parallel computers with any degree of parallelism (DOP). We propose two variants, each of which has the following phases: obtaining a parallel schedule by list scheduling heuristics, optimization by rearranging the tasks in each level to adapt different DOP, and serialization of the schedule. To prove the efficiency, we have made simulations with random DAGs and compared the results to those obtained by the Critical Path Algorithm applied separately for each DOP.

Key Words: Instruction Scheduling, Static Scheduling Algorithms, Multiprocessor Scheduling, Degree of Parallelism, Directed Acyclic Graphs

1. Introduction

The performance of parallel computers depends to a great degree on the compiler's ability to generate code that can be executed by the hardware in an appropriate order. Compilers incorporate instruction scheduling phase in which the instructions are arranged statically, so that if they are executed in this order on the target machine, the execution time will be optimized. When executing a program on a parallel computer, the source code of the program is translated by the parallelising compiler into a parallel object code which has the same degree of parallelism as that of the parallel computer. The compiler generates a precedence task graph after flow analysis, then uses an instruction scheduling method like Critical Path Method (CPM) [2] to generate the program's object code. The scheduling is done with the assumption of a fixed degree of parallelism and that is the reason why the generated parallel object code cannot be executed efficiently or at all on a computer with different degree of parallelism. To execute the program on a computer with different degree of parallelism in minimum time, the source code of the program needs to be translated again with another parallelism assumption.

To solve this problem, we propose a static scheduling method called a *Parallelism-Independent Scheduling Method.* The Parallelism-Independent Scheduling Method produces a parallelism-independent object code which can be executed on parallel computers with any degree of parallelism with near-optimal time.

We propose two Parallelism-Independent Scheduling algorithms developed under the assumptions that the program code does not include

branches and loops, each instruction has an unit execution time, all the processors in the multiprocessor system are homogeneous and there is no inter-processor communication overhead. In order to evaluate the performance of the proposed algorithms we have made simulations using random graphs with different input parameters as number of nodes and edges and different degree of parallelism. For evaluation purposes a trace of random graphs with equal degrees of parallelism have been examined and the average values of the results in terms of schedule length have been taken. The program is scheduled only once using a Parallelism-Independent scheduling algorithm and the schedule lengths for its execution with different degrees of parallelism are compared to the schedule lengths obtained by using the CP algorithm but applied separately for each possible degree of parallelism.

The remainder of this paper is organized as follows: in Section 2 we present previous work related to static scheduling for multiprocessor systems. In Section 3 we describe in detail the basic idea of the Parallelism-Independent Method and the proposed by us two algorithms: Parallelism-Independent Simple (PIS) and Parallelism-Independent Critical Path (PICP) algorithm. In Section 4 we present the simulation results and comparison of the two algorithms, and draw the conclusions in the last section.

2. Related Work

The utilization of the big potential of the multiprocessor architectures depends very much on the effective instruction scheduling and that is the reason why there are many researches in this area. The problem of scheduling a set of precedence-related tasks forming a directed acyclic graph (DAG) is known to be NP-complete except in some restricted cases: 1. scheduling a tree-structured DAG with identical instruction execution times to arbitrary number of processors (applying Hu's algorithm which uses a level number - the length of the longest path from a node to the exit node); 2. scheduling an arbitrary DAG with identical instruction execution times to two processors, shown by Coffman [1], and 3. scheduling an interval-ordered DAG to an arbitrary number of processors, proved by Papadimitriou and Yannakis. Because of the NP-completeness of the problem, there have been a large number of static scheduling heuristics described in the literature under different assumptions for the execution time and communication costs [4].

The most common technique for DAG scheduling is the list scheduling: assigning priorities to the nodes of the DAG and allocating the nodes for execution to the available processors in order of their priorities. The quality of the algorithms depends on the accuracy with which the priorities of the nodes are defined. Adam, Chandy and Dickson [5] suggest that using a level number as node priority is the nearest solution to optimal. Many task scheduling schemes use the classical Critical Path heuristic because the tasks belonging to the CP determine the shortest possible time for execution of the whole program. The steps of the CP algorithm consist of defining the longest exit path for each graph node, rank the nodes according to the length of these paths and assign nodes for execution in their rank order. The CP algorithm does not consider the weight of the nodes in the node's subtrees. Shirazi, Wang and Pathak [2] propose a Weighted Length Algorithm which takes into account the relationships among the nodes at different levels and the rank of the node depends on the length of the exit path, the number of the children and the weights of the children and their descendants.

Instruction scheduling has a significant role in superscalar processors too because their performance depends to a great degree on the exploitation of the instruction level parallelism (ILP). The scheduling helps for exposing more ILP in the instruction stream and although the superscalar processors use hardware to schedule instructions dynamically, their performance often depends on the compiler-specified, static scheduling. So scheduling can be done both at compile time and dynamically with the purpose of enabling as many instructions as possible to be issued for execution per cycle. Some researches extend the traditional static instruction scheduler so that it takes advantage of the hardware resources for out-of order instruction issue.

However, all the static scheduling algorithms so far assume fixed degree of parallelism and the code obtained by using these algorithms could be executed efficiently only on a machine with the same degree of parallelism. To the best of our knowledge, we are the first to propose a scheduling method which enables program execution effectively on parallel computers with different degrees of parallelism without necessity to reschedule the programs for each number of processors they are run on.

3. Parallelism-Independent Scheduling Method

The Parallelism-Independent Scheduling Method is a method for producing a parallelism-independent code as a list of serially ordered tasks separated by special boundary markers called *parallel execution limits*. They specify the number of instructions (tasks) that

can be executed simultaneously with different degrees of parallelism. In this way parallel computers with different degree of parallelism can exploit efficiently all the instruction-level parallelism that appears in this list of serially ordered tasks. The ready instructions are assigned for execution to the processors in a serial order until information for a parallel execution limit corresponding to the degree of parallelism of the computer is met, then the assigned tasks are executed in parallel. After that another set of tasks is assigned and executed in parallel. The process is repeated until the end of the task list.

We propose three algorithms for PI scheduling: Parallelism-Independent Simple (PIS) Scheduling and Parallelism-Independent Critical Path (PICP) Scheduling. We have developed these algorithms under the assumption that the precedence task graph representing the program is a directed acyclic graph. Each node in the graph represents an instruction and the edges between the nodes represent the data dependencies between the instructions. All the instructions have uniform unit execution times. The processor are homogeneous and the inter-processor communication is ignored.

The scheduling framework for the proposed algorithms is as follows:

1. Assigning priorities to the nodes of the DAG according to some heuristics and building a sorted priority list of instructions (instructions with higher priority appear earlier in the priority list);
2. Forming blocks of instructions taking the data dependencies between them and the instruction priorities into account. Instructions which can be executed simultaneously appear in one block:
3. Optimization phase in which the instructions are rearranged in the blocks so that they can be executed efficiently with different degrees of parallelism;
4. Forming a serially ordered list of tasks by connecting all the blocks from top to bottom;
5. Attaching information for the parallel execution limits: the serialized sequence of instructions is examined from top to bottom whether the instructions can be executed with different degrees of parallelism and if not, a marker indicating this is inserted.

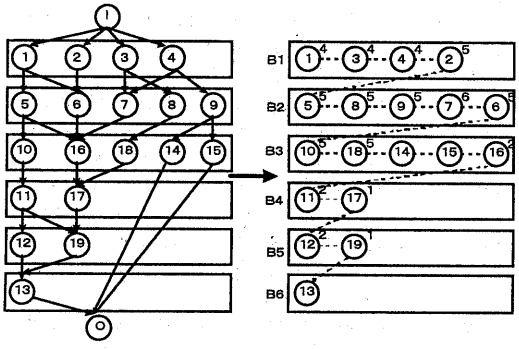
3.1 Parallelism-Independent Simple Scheduling (PIS) Algorithm

We use in this algorithm two functions for ordering the tasks: their co-levels (the length of the longest path from the task to the input node) for forming of blocks, and the distance between predecessor and successor tasks in two successive blocks for determining the exact position of each task in the blocks. The purpose is to arrange the tasks in such a way that the distances between immediate predecessors and successors in two successive blocks is maximum. This maximum distance will make possible the execution of the instructions with different degrees of parallelism.

The steps of this algorithm are briefly explained below through a simple example of a sample DAG:

1. At this stage the number of the processors is assumed to be unlimited. The instructions are ordered into blocks according to the length of the path from them to the input node, so that instructions which can be executed simultaneously are ordered into one block as shown in Fig.1a.
2. In the first block instructions with greater number of successors have higher priority and appear earlier in it. If the instructions are with equal number of successors, then the instructions appear in the block according to their number. The arrangement of instructions in the successive blocks is done according to their distance from their immediate predecessors in the previous block. The distance between successor and predecessor shows the number of instructions that have to be assigned for execution between them. Those instructions which have greater distance from their intermediate predecessor in the previous block appear earlier in the next block. If the instructions have the same distance from their predecessor, the instruction with greater number of successors is given a higher priority (Fig. 1b).The distances are shown above the number of the instructions in that figure. For example $dist\text{-}pos1_{1\text{-}5} = 4$ is the distance between task 1 and its successor task 5, if task 5 is placed on first position in the second block B2. That distance is equal to 4 because four ready tasks with higher priority (tasks 1, 2, 3, 4) have to be assigned for execution, before task 4 is assigned to a processor and executed. In the same way we can determine $dist\text{-}pos1_{2\text{-}6} = 1$, $dist\text{-}pos1_{4\text{-}7} = 2$, $dist\text{-}pos1_{3\text{-}8} = 3$, and $dist\text{-}pos1_{4\text{-}9} = 2$. Task 5 is put on first position in the second block B2 because its distance from task 1 is greater than the distances of all the other tasks in block 2, if placed at first position. In this way all the tasks in the second block B2 are ordered according to their distances to their predecessors in block B1. Then the value of the distances between predecessor and successor is readjusted according to the position of the task: $dist\text{-}pos1_{1\text{-}5} = 4$, $dist\text{-}pos2_{3\text{-}8} = 4$, $dist\text{-}pos3_{4\text{-}9} = 4$, $dist\text{-}pos4_{4\text{-}7} = 5$, $dist\text{-}pos5_{2\text{-}6} = 5$, as shown in Fig. 3b. For task 4 the smaller distance to its immediate successor task 9 is taken. We have to note that tasks 7 and 9 will have equal distances, if placed at position 3, but task 9 has greater number of successors and that is the reason it is given a greater priority than 7.

Similarly the process is repeated until the positions of all the tasks are determined in all the blocks.



(a) (b)

Fig. 1 (a) Forming of blocks; (b) Rearranging of the instructions in the blocks

1 3 4 2 [x5] 5 8 9 7
6 10 18 14 15 16 11 [x4] 17
[x5] 12 [x3] [x4] 19 [x3] [x4] [x5] 13

Fig. 2 Insertion of parallel execution limits in the serialized schedule

The schedules for the DAG of Fig. 1 when it is executed on different degrees of parallelism are given on Fig. 3a. The schedules when the sample DAG is scheduled using CP algorithm for each degree of parallelism are presented in Fig. 3 b. We can see in Fig. 3a that at some cycles there are idle processors which if utilized, could lead to a better schedule. This problem is solved by the next algorithms described in Section 3.2. On Fig. 2 we show the phase in which the markers for the execution limits are attached. [x5] means that the instructions after it cannot be executed with the instructions before it with degree of parallelism 5.

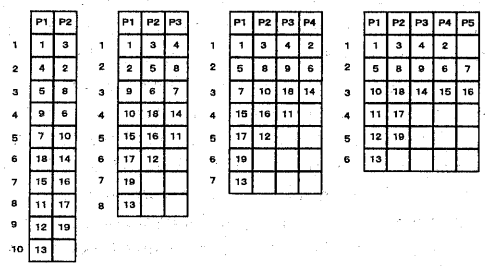


Fig. 3a Length of the schedules for different degrees of parallelism using PIS Scheduling Algorithm

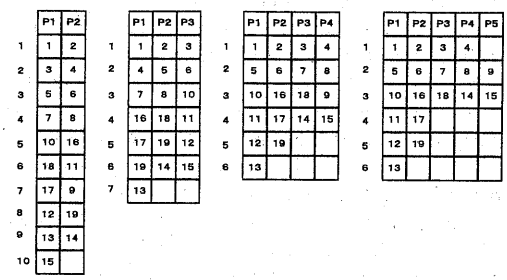


Fig. 3b Length of the schedules for different degrees of parallelism when using CP algorithm applied separately for each degree of parallelism.

3.2 Parallelism-Independent Critical Path (PICP) Scheduling Algorithm

The difference of PICP algorithm and the PIS algorithm is that in the first step the program is scheduled using the CP algorithm. The CP algorithm is applied for degree of parallelism p equal to the minimum number of processors necessary for execution of the program with schedule length equal to the CP in the DAG.

We define p as follows: $p = \lceil N/\text{Length of CP} \rceil + \alpha$ where N is the total number of instructions in the graph, and α is a non-negative number which cannot be determined easily. In order to determine p exactly first the program is scheduled for number of processors equal to (N/Length of CP) and if the schedule length is longer than the CP, then the process is repeated for number of processors P+1 and so on until the schedule length becomes equal to the CP in the DAG. For our sample DAG, the value of p is four. The rearrangement of the tasks in the blocks is shown on Fig. 4.

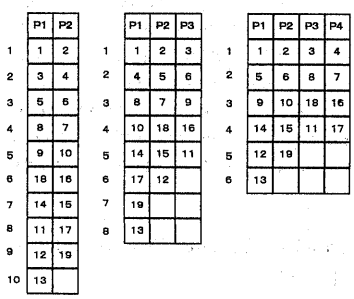


Fig. 4 Length of the schedules for different degrees of parallelism using the PICP scheduling algorithms

4. Simulations

To prove the effectiveness of our algorithms we have

made simulations using random graphs and compared the results to those given by the CP algorithm applied separately for each degree of parallelism. The CP algorithm is used for comparison because it has been shown to provide schedules within 5% of the optimal in 95% of the random cases. We have used for the simulations ANSI C in Unix environment.

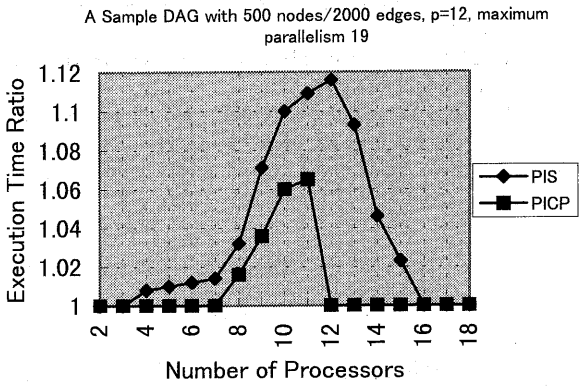
In order to carry out our experiments and 25 random graphs with 250 nodes/1000 edges (p=8), 25 random graphs with 500 nodes/2000 edges (p=12). For each of these graph types the average deviations from the results of the CP method have been taken.

Graphs with a specific degree of parallelism (p=8 or 12) have been obtained as we separate the nodes into levels and the nodes in each level are chosen successors randomly from the nodes in a limited number of levels below. Of all the generated random graphs, we have taken for our experiments only those which can be executed with time equal to the length of the CP on minimum number of processors equal to 8, or 12 respectively. This has been done to make it possible to take average results from the experiments. The ratio between the number of the nodes and the edges is taken to be 4 as a typical value in DAGs representing a user's program.

When using the PIS or PICP scheduling algorithms, the DAGs are scheduled only once and from the so generated list of serially ordered tasks, schedules for different degrees of parallelism can be formed. The schedule lengths for the different degrees of parallelism are compared to the schedule lengths obtained using the CP algorithm applied separately for each degree of parallelism.

The maximum values for the execution time ratio for the PIS algorithm are reached for degree of parallelism equal to p, and for the other two algorithms for degree of parallelism equal to (p-1). The execution time ratio becomes equal to 1 for the PICP for degrees of parallelism equal to p or (p-1). For the PIS algorithm the time execution ratio becomes 1 for degree of parallelism greater than p which depends on the maximum degree of the parallelism in the graph. On Fig. 5a and 5b we present the results for 2 sample DAGs with p=12, the maximum deviation for the PIS algorithm 11.6% for the first graph and 15.1% for the second graph are achieved for number of 12 and 11 processors respectively. The increase of the maximum parallelism in the graph affects the results of the PIS algorithm at a great degree as seen from those two figures.

In Fig. 6 we present the results for the PICP algorithm for 4 sample graphs with same p=8 but different ratio between the number of the edges and the number of the nodes. With increase of the number of the edges in the graph, the Execution Time Ratio for the PICP algorithm increases as well.



a) DAG with maximum parallelism 19

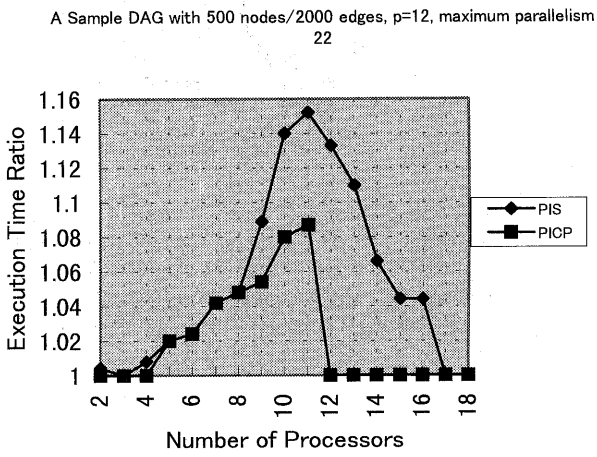


Fig. 5 Results for 2 Sample DAGs with different degrees of parallelism. With increase of the degree of parallelism of the graph, the deviation of the results of the PIS algorithm from that of the CP algorithm become greater.

The average results we have obtained for 25 graphs with 250 nodes/1000 edges and 500 nodes/2000 edges are presented in Fig. 7 and 8 and could be summarized as given below, in brackets we give the values between which the maximum deviation fluctuates:

1. 250 nodes/1000 edges (p=8), results for 25 random DAGs - Fig. 12:

- PIS Scheduling algorithm: Maximum average deviation 12.6% (min 6% - max 17.6%)
- PICP Scheduling algorithm: Maximum average deviation 7.7% (min 5.6% - max 9.7%)

2. 500 tasks/2000 edges (p=12), results for 25 random graphs- Fig. 13:

- PIS Scheduling algorithm: Maximum average deviation 14.1% (min 9.1% - max 17.4 %)
- PICP Scheduling algorithm: Maximum average deviation 9.5% (min 6.5% - max 13%)

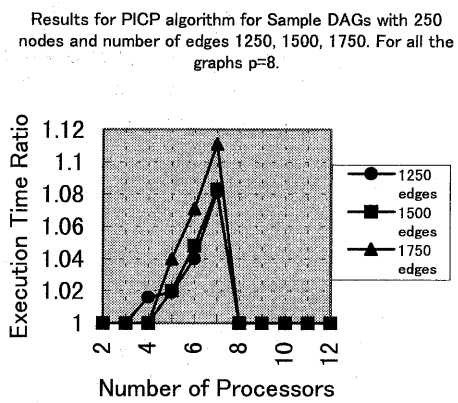


Fig. 6 Results for PICP algorithm for DAGs with 250 nodes and p=8. The Ratio between the number of the nodes and the number of the edges is 5, 6 and 7 respectively. The maximum execution ratio is 8.1%, 8.3% and 11.1% respectively for 7 processors.

5. Conclusion

The main contribution of this paper is the proposition of static scheduling algorithms not depending on the degree of parallelism which generate schedule enabling the execution of the program on variable number of processors with schedule length comparable to the optimal. The results show that when the programs scheduled by the PICP algorithm are executed on variable number of processors, the execution time does not exceed that of the time obtained by the CP algorithm more than 10% in most of the cases.

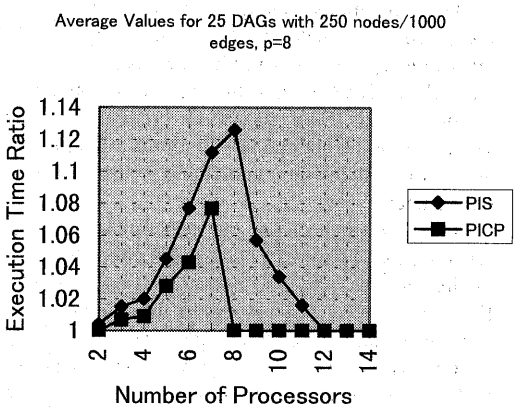


Fig. 7 Average results taken for 25 DAGs with 250 nodes/1000 edges, p=8.

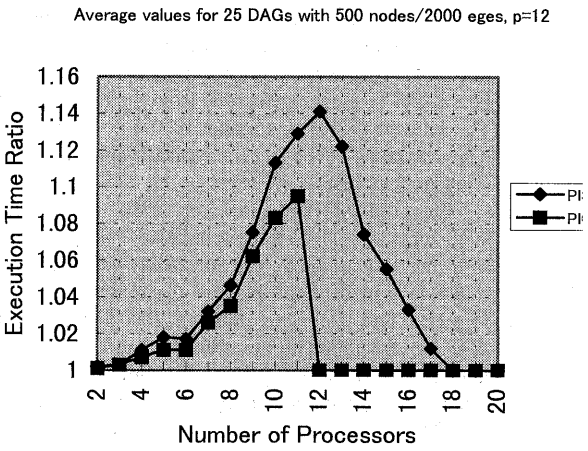


Fig. 8 Average results taken for 25 DAGs with 500 nodes/2000 edges, p=12

References

[1] E. G. Coffman, Computer and Job-Shop Scheduling Theory. Wiley, New York, 1976.

[2] B. Shirazi, M. Wang and G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Scheduling," Journal of Parallel and Distributed Computing, no. 10, pp. 222-232, 1990, March 1995, pp. 303 - 313.

[3] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on Unbounded Number of processors," IEEE Trans. On Parallel and Distributed Systems, vol.5, no.9, pp.951-967, Sep. 1994.

[4] Yu-Kwong Kwok, Ishfaq Ahmad, and Jun Gu, "FAST: A Low Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors," 1996 International Conference on Parallel Procesing, pp. II-150-II157

[5] T.L. Adam, K.M. Chandy, and J. R. Dickson, "Comparison of List Schedules for Parallel Processing Systems," Comm ACM, vol. 17, no. 12, pp. 685-690, Dec. 1974