A Heuristic for Scheduling Parallel Programs with Synchronous Communication Model in the Network Computing Environments

Zhao Mingyu and Zhang Tianwen

School of Computer Science and Technology Harbin Institute of Technology, Harbin 150001,China
zhmy@21cn.com

Abstract. Most heuristics for scheduling address asynchronous communication DAG, but they are not suitable for the synchronous ones. The proposed PRGSC algorithm avoids the deadlock that is caused by the synchronous communication, but also it can alleviate impact of synchronous communicating delay. Simulation shows that the PRGSC algorithm has better performance than the CASC algorithm which deals with the same type of problems.

1 Introduction

Scheduling problem is fundamental and highly important in the parallel system research domain [1]. Most of scheduling heuristics target to the asynchronous scheduling problems. But applications in the network computing environment may depend on the synchronous communication to ensure the reliability. Nevertheless, synchronization introduces delay for the sender. Furthermore, synchronization may result in deadlock situations between communicating tasks.

With regards to synchronous scheduling, there have been few researches and algorithms proposed [3, 4]. Although these algorithms studied deadlock detection, they can merely manage the direct deadlock between processors, but not the circular deadlock among multiprocessors.

This paper proposed a heuristic: Parameters Relation Graph based Synchronous Clustering (PRGSC) for the synchronous scheduling problems. It can fully detect and avoid the deadlock situation, while has better scheduling quality.

2 Definitions

Parallel applications can be depicted as weighted Directed Acyclic Graph, $G = (V, E, \tau, \beta)$, where $V = \{n_1, n_2, \ldots, n_v\}$ represents the set of tasks to be executed, and the weighted, directed edges $(n_i, n_j) \in E$ represents communication between tasks. The weight of task n is $w(n)$, representing the execution time. Edge (n_i, n_j) represents a message sent from n_i to n_j , and the weight of the edge refers to the message transportation time, denoted $\beta(n_i, n_j)$. We use $tlevel(n_i) + blevel(n_i)$

to evaluate node n_i's relevant importance, where tlevel and blevel are defined in [1]. The execution of task can be divided into three phases: receiving, computing and sending. Therefore, four attributes are required to measure the start time and finish time of these three phases:

Definition 1. *$r(v)$ is the time that v starts receiving messages.*

Definition 2. *$R(v)$ is the time that v finishes receiving messages, and it is also the start time of computing phase.*

Definition 3. *$s(v)$ is the time that v starts sending messages, and it is also the finish time of computing phase.*

Definition 4. *$S(v)$ is end of the sending phase of a task when it receives acknowledgments from all the recipient tasks.*

These values can be evaluated by the equations defined in [3].

3 The Proposed Heuristic

3.1 Selection of Nodes and Processors

In the synchronous scheduling, unlike the asynchronous one, scheduling a node changes not only the time properties of unscheduled nodes, but also those of scheduled nodes. Thus, except for decreasing the start time of unscheduled part, every scheduling step has to evaluate the impact on scheduled part in synchronous model.

Therefore, PRGSC tries four methods of scheduling a node in every scheduling step:

1. Among the edges between scheduled nodes and ready nodes, choose the one (n_p, n_f) with highest priority. An edge's priority is defined as the sum of two nodes' priorities.
2. Choose n_p with highest priority among the scheduled nodes which have ready child nodes, firstly; then, select the critical child node n_f in the ready child nodes set of n_p.
3. Choose n_f with highest priority among the ready nodes, then select the critical parent nodes n_p in the set of parent nodes of n_f.
4. Assign the ready node with highest priority onto a new processor without changing the makespan.

PRGSC compares each zeroing result and uses the one that produces the lowest makespan and no deadlock in each step. PRGSC zeros the edge (n_p, n_f) by adding n_f into cluster $clust(n_p)$, i.e., setting n_f rightly behind the last node n_l. If n_l isn't a parent of n_f in the original DAG, produce a pseudo-edge (n_l, n_f) with weight of zero.

3.2 Deadlock Detection

Definition 5. *Deadlock is a status. Under this status, there is a subset D of tasks. For every element that belongs to D, its complete time is decided by the others. That is,* $D = \{v_i : 1 \leq i \leq m, m \geq 2\} \in V$*, for every task* $v_i \in D$*,* $S(v_i) = f_i(S(v_1), S(v_2), S(v_{i-1}), S(v_{i+1}), \ldots, S(v_m))$*, where* f_i *is a function.*

Every step changes the scheduling by zeroing a certain edge or adding a pseudo-edge. Represent the scheduled graph of step k with $G_k(V, E_k)$. Say that assign node n_f to processor P at step k, and n_l is the last node on P at step $k-1$, then $E_k = E_{k-1} - (n_i, n_f) + (n_l, n_f)$, where $n_i \in P$, $(n_i, n_f) \in E_k$, and $\beta(n_l, n_f) = 0$. Provide the definition accordingly:

Definition 6. *In the kth step, Parameters relation graph is* $G_k^{'}(V^{'}, E_k^{'})$*, where the elements of set* $V^{'}$ *are values of r,R, s and S of nodes n in V, denoted as n.r, n.R, n.s and n.S respectively. If the value of element a has some effect on that of element b, then* $(a, b) \in E_k^{'}$*. All possible members of* $E_k^{'}$ *are listed below:*

1. $\{(n.r, n.R), (n.R, n.s), (n.s, n.S)\} \subseteq E_k^{'}, n \in V$.
2. *If* $(n, c) \in E_k^{'}$ *and* $\beta(n, c) \neq 0$*, then* $\{(n.s, c.R), (c.r, n.S)\} \subseteq E_k^{'}$.
3. *If* $(n, c) \in E_k$ *and* $\beta(n, c)=0$*, then* $(n.S, c.r) \in E_k^{'}$.

According to definition 1-4 and the assumption that tasks on the same processor have no communication cost, $E_k^{'}$ of step k can be incrementally derive from $E_{k-1}^{'}$: $E_k^{'} = E_{k-1}^{'} - \{(n_i.s, n_f.R), (n_f.r, n_i.S)\} + \{(n_l.S, n_f.r)\}$.

Due to the construction scheme, we can derive two theorems following:

Theorem 1. $G_k^{'}$ *includes a circuit, iff the current schedule is deadlock.*

Theorem 2. *The schedule is deadlock, iff there is a circuit in* $G_k^{'}$ *through* $n_f.r$.

For the sake of space, the proofs are ignored.

As it need to traverse the PRG $G_k^{'}$ breadth-first to compute all the parameters and determine if exist a circuit in current schdule, the complexities of parameters computing and deadlock detection are both $O(v+e)$. For every node scheduling, deadlock determination and time parameters update are the major operations. Thus, the complexity of PRGSC is $O(v(v+e))$.

4 Simulation experiment

250 random generated graphs are used for validating the performance of PRGSC. These graphs are separated into 5 groups according to their CCR [5], values of which are 0.1, 0.5, 1.0, 2.0 and 10.0 respectively. Each group has 50 graphs, which are separated into 10 subgroups according to the number of nodes, valuing from 50 to 500, 50 incremental. Every subgroup contains 5 graphs.

Among all the previous researches, CASC is most similar to ours, but it can only determine and avoid the direct deadlock between 2 processors. We replace the deadlock detection function in CASC with the one mentioned above. Table 1

Table 1. The average improving comparisons for random generated DAGs

Size	CCR=0.1	CCR=0.2	CCR=1.0	CCR=2.0	CCR=10.0
50	10.13%	10.08%	13.78%	12.56%	9.28%
100	4.58%	6.69%	13.11%	11.70%	17.33%
150	6.31%	5.76%	10.33%	9.02%	9.28%
200	4.65%	7.72%	5.75%	10.36%	12.23%
250	2.08%	5.21%	6.05%	5.38%	9.51%
300	-0.07%	3.55%	5.55%	7.18%	7.36%
350	2.11%	3.15%	6.50%	5.15%	6.39%
400	3.03%	1.73%	3.80%	6.67%	5.49%
450	3.30%	3.78%	4.75%	4.45%	5.35%
500	3.47%	3.82%	3.32%	5.95%	6.82%

is the comparison of the scheduling quality of PRGSC and modified CASC. The measures in the table are the average makespan improvement of the 5 graphs in every subgroup. We can conclude from these experiment results that PRGSC has better performance than CASC consistently, except for the situation of CCR=0.1 and graph size of 300.

5 Conclusions

Parallel computing in the synchronous communication environment has different impacts on scheduling method. This paper presents a scheduling algorithm used for parallel application in synchronous communication, proves the validity of the deadlock determination method analytically and its efficiency experimentally. PRGSC has lower complexity, breaks the limitation of direct deadlock determination and has better scheduling result than CASC.

References

1. Y.K. Kwok and I. Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, Dec.1999, vol.31, no.4, pp. 406–471.
2. A. Gerasoulis and T. Yang, A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors, J. Parallel and Distributed Computing, Dec. 1992, vol. 16, no. 4, pp. 276–291.
3. D. Kadamuddi and J. Tsai, Clustering Algorithm for Parallelizing Software System in Multiprocessors, IEEE Transactions on Software Engineering, Apr.2000, vol. 26, pp. 340–361.
4. B.R. Arafeh, A Task Duplication Scheme for Resolving Deadlocks in Clustered DAGs, Parallel Computing, 2003, vol. 29, pp. 795–820.
5. Y.K. Kwok and I. Ahmad, Benchmarking and Comparison of the Task Graph Scheduling Algorithms, J. Parallel and Distributed Computing, Dec. 1999, vol.59, no. 3, pp. 381–422.