Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 29 (2012) 3820 – 3825

Procedia Engineering

www.elsevier.com/locate/procedia

2012 International Workshop on Information and Electronics Engineering (IWIEE)

Reduced Communications Fault Tolerant Task Scheduling Algorithm for Multiprocessor Systems

Nabil Tabbaa*, Reza Entezari-Maleki, Ali Movaghar

Department of Computer Engineering, Sharif University of Technology Tehran, Iran

Abstract

Multiprocessor systems have been widely used for the execution of parallel applications. Task scheduling is crucial for the right operation of multiprocessor systems, where the aim is shortening the length of schedules. Fault tolerance is becoming a necessary attribute in multiprocessor systems as the number of processing elements is getting larger. This paper presents a fault tolerant scheduling algorithm for task graph applications in multiprocessor systems. The algorithm is an extension of a previously proposed algorithm with a reduced communications scheme. Simulation results show the efficiency of the proposed algorithm despite its simplicity.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of Harbin University of Science and Technology

Keywords: Multiprocessor systems; task scheduling; fault tolerance

1. Introduction

Multiprocessor systems are commonly deployed for executing computationally intensive parallel applications with various computing requirements. Mapping the tasks to the processors and specifying their execution time is one of the essential steps in parallel processing. This step, called task scheduling, determines the efficacy of the application's parallelization in multiprocessor systems [1].

Component failures may occur in multiprocessor systems. Consequently, there is a growing need for developing fault tolerant techniques. Actually, the likelihood of failure occurrences is increased by the fact that many parallel applications are getting larger, and might require long period of time for execution.

* Corresponding author.
E-mail address: tabbaa@ce.sharif.edu.

1877-7058 © 2011 Published by Elsevier Ltd.
doi:10.1016/j.proeng.2012.01.577

Hence, a technique is required to enable a multiprocessor system to continue the applications' execution even in the presence of one or more processor failures.

Task scheduling is among the techniques which can be used to achieve the required fault tolerance. The main objective of fault tolerant task scheduling algorithms is to find a suitable mapping of tasks to the processing elements of a multiprocessor system and tolerate a given number of processor failures [2].

In this paper, a reduced communications fault tolerant task scheduling algorithm is proposed. This algorithm is an extension of a previously presented algorithm [3] with a reduced communications scheme. The original algorithm aimed at tolerating multiple processor failures and achieving a minimum possible schedule length. The goal of the new extension is to reduce the communications duplication between the replicas of the tasks, while keeping the fault tolerance of the required number of processor failures.

2. Multiprocessor systems

The underlying multiprocessor system consists of a finite processor set $P = \{P_1, P_2, \ldots, P_m\}$. The processors are fully connected with each other via a reliable link. Thus, the delay of a communication is the same between any pair of the processing elements. Each processing element is composed of an application processor, a local memory, and an I/O processor. Because I/O processors handle communications separately from application processors, the processing elements can perform computation and communication simultaneously [4].

The processing elements may be heterogeneous or homogeneous. The heterogeneity of the processors means that they have different speeds or processing capabilities. However, it is assumed that every task of the application can be executed on any processor, even though the completion times on different processors may be different. The heterogeneity of processing capability is modeled by a function $C{:}P\rightarrow R^{+}$, where the processing capability of a processor P_k is given by $C(P_k)$ [4].

3. The original algorithm

The objective of the algorithm proposed in [3] was to map the tasks of directed acyclic graph (DAG) application to processors with diverse capabilities in a distributed computing system. The algorithm aimed to minimize the schedule length while tolerating a given number (*npf*) of processor failures. To achieve this, active replication scheme was used to allocate *npf*+1 copies of each task to different processors.

The original algorithm mainly used the familiar heuristic found in DAG scheduling algorithms that is called list scheduling. In that algorithm, each node was scheduled to multiple processors to achieve the required fault tolerance.

In the algorithm proposed in [3], free nodes were ordered by a priority value equals to *tlevel*+*blevel* of the node, where *tlevel* denotes the dynamic top level and it is computed using (1).

$$tlevel\,(n_i) = \max_{n_j \in pred\,(n_i)} \{FT\,(n_j, Proc\,(n_j)) + c(n_i, n_j)\}, \tag{1}$$

where $FT(n_j, Proc(n_j))$ is the finish time of node n_j (a predecessor of n_i) which has been previously scheduled on processor $Proc(n_j)$, and $c(n_i, n_j)$ is the data communication cost between n_i and n_j.

The *blevel* denotes the static bottom level and it is computed by (2).

$$blevel\ (n_i) = \max_{n_j \in succ\ (n_i)} \{\overline{w(n_i)} + c(n_i, n_j) + blevel\ (n_j)\}, \tag{2}$$

where $\overline{w(n_i)}$ is the average execution time of node n_i on all the system's processors. The expected finish time of free node n that has the highest priority is calculated on all of the processors using (3).

$$FT(n,P_l)=C(P_l)\times w(n)+\max\{\max_{n_j\in pred(n)}[\min_{1\le k\le npf+1}(FT(n_j^k,Proc(n_j^k)))+c(n_j,n)],r(P_l)\}, \tag{3}$$

where $r(P_l)$ is the ready time of the processor P_l. The predecessor nodes are already scheduled onto *npf*+1 processors, and n_j^k denotes the k^{th} replica of node n_j.

Then, the node n is scheduled on the *npf*+1 processors which deliver the minimum finish time for that node. The schedule length *SL* can be computed using (4).

$$SL = \max_{n\in V}\{\min_{1\le k\le npf+1}[FT(n^k,Proc(n^k))]\} \tag{4}$$

4. Communications reduction

By using the active replication scheme in the algorithm described above, each task in the graph is replicated *npf*+1 times. For any task n_i of the DAG, the *npf*+1 replicas of each predecessor n_j should send their own data to the *npf*+1 replicas of n_i. Therefore, each communication between two tasks in precedence is replicated $(npf+1)^2$ times.

It is mandatory to duplicate each task *npf*+1 times to achieve the required fault tolerance and resist to *npf* processor failures. But, duplicating the communications between all the tasks replicas is not obligatory.

For example, let us take a task n_i with two predecessors n_x and n_y. In the non-fault tolerant schedule, there is only one replica of each predecessor, and this replica will send one data message to the only replica of task n_i. So, the total number of messages in this example is two.

In Fig. 1(a), the schedule is supposed to tolerate two processor failures (*npf*=2). Therefore, each task is replicated three times (i.e., *npf*+1) on different processors. With duplicating all the communications between the replicas of n_i and the replicas of its predecessors n_x and n_y, the total number of messages is increased to 18, so it is duplicated 3^2 times.

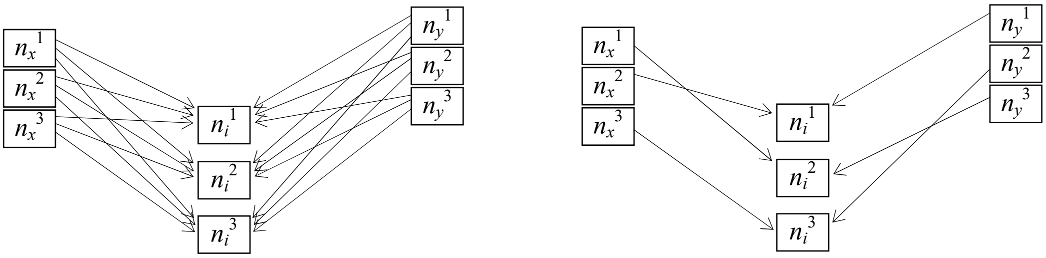

Fig. 1. (a) Fault tolerance with full communications replication; (b) Fault tolerance with reduced communications replication

Suppose the worst case where *npf* processors failed during the execution of these tasks. If *h* numbers of these failed processors were among the *npf*+1 processors where the replicas of the predecessor n_x are scheduled, then, the number of completed replicas of n_x will be *npf*+1-*h*.

As the total number of failures is *npf*, there will be at most *npf*-*h* failed processors among the *npf*+1 processors where the replicas of the task n_i are scheduled. Therefore, the number of working replicas of the predecessor n_x which is *npf*+1-*h* is always greater than the number of failed replicas of task n_i which is *npf*-*h*. Consequently, the required fault tolerance can be achieved if each replica of the predecessor n_x sent its data message to only one replica of the task n_i, because always there will be a communication link between two working processors. This idea can be applied to all predecessors, as it is shown in Fig. 1(b), and the total number of data messages is increased to six instead of 18. Therefore, the original data messages are duplicated three times only. Using the reduced communications replication scheme, each communication between two tasks in precedence is duplicated (*npf*+1) times instead of $(npf+1)^2$.

In the original algorithm, with the full communications replication scheme, the finish time of task *n* on processor P_l was calculated by Eq. (3), where every replica of the task *n* can start execution after receiving the data message from the earliest replica of the predecessor n_j. While, in the reduced communications scheme, each replica of the task *n* will receive the data message from only one replica of the predecessor n_j. So, the finish time of each replica of the task *n* will be calculated by Eq. (5).

$$FT(n,P_l) = C(P_l) \times w(n) + \max\{\max_{n_j \in pred(n)}[FT(n_j^k, Proc(n_j^k)) + c(n_j,n)], r(P_l)\} \qquad (5)$$

where, the data message is received only from the k^{th} replica of the predecessor n_j (the minimum is not used here), and this replica *k* is chosen in a way that minimizes this finish time.

5. Performance evaluation

To evaluate the proposed fault tolerant task scheduling algorithm, the algorithm in its original and reduced communications versions, is simulated and compared to FTBAR algorithm [5] which is the closest algorithm found in the literature.

The most important metrics of the performance of the algorithms are the normalized schedule length (NSL) and the fault tolerance overhead. NSL is obtained by dividing the output schedule length by the length of the critical path of the DAG. Moreover, the fault tolerance overhead caused by the active replication, can be computed in the following way: *overhead*=(*FTSL-nonFTSL*)/*FTSL*100*, where *FTSL* is the fault tolerant schedule length and the *nonFTSL* is the schedule length obtained when the number of tolerated failures *npf* is set to zero.

The algorithms have been simulated with a set of randomly generated graphs. The execution times of tasks are randomly selected from a uniform distribution. Similarly, the communication times of data dependency are randomly selected from a uniform distribution. Each point in the results represents an average over 60 random graphs. The number of processors of the system is set to 10, and their processing powers are randomly selected from a uniform distribution. The number of processor failures, *npf*, that the obtained schedule is expected to tolerate is set to five.

Fig. 2(a) shows the comparison of the NSL between the algorithm, in its two versions, and the FTBAR algorithm as a function of the number of tasks, which is varied uniformly in the range [20, 200], while CCR is set to one. In Fig. 2(b), the comparison of the NSL is shown as a function of the CCR, which is varied uniformly in the range [0.2, 5], while the number of tasks is set to 100. As it can be seen in Fig. 2, the algorithm, in its two versions, shows better NSL results compared to the FTBAR algorithm for any number of tasks and any value of CCR.

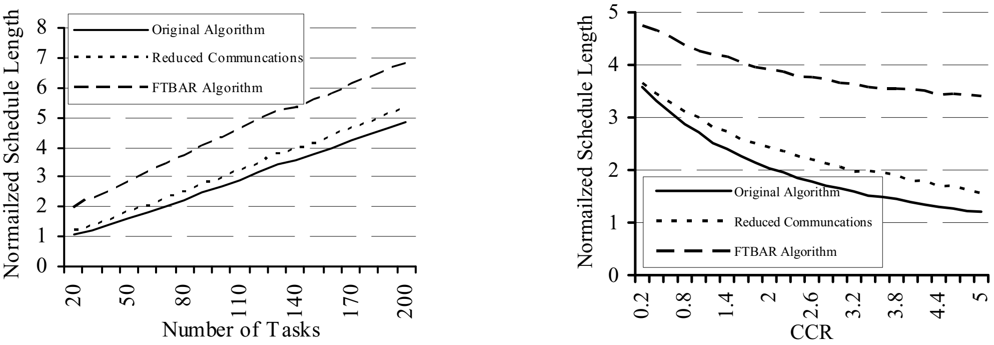

Fig. 2. (a) Normalized schedule length as a function of the number of tasks; (b) Normalized schedule length as a function of CCR

Fig. 3(a) shows the comparison of the overhead between the algorithm, in its two versions, and the FTBAR algorithm as a function of the number of tasks, which is varied uniformly in the range [20, 200], while CCR is set to one. In Fig. 3(b), the comparison of the overhead is shown as a function of the CCR which is varied uniformly in the range [0.2, 5], while the number of tasks is set to 100.

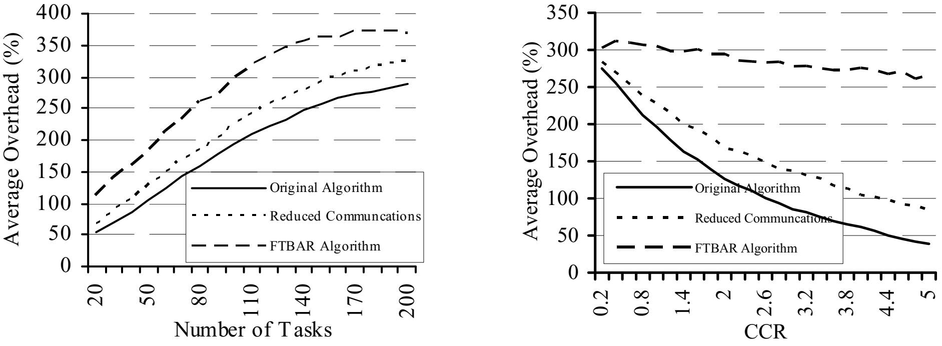

Fig. 3. (a) Fault tolerance overhead as a function of the number of tasks; (b) Fault tolerance overhead as a function of CCR

As shown in Fig. 3, the algorithm in its two versions, shows better overhead results compared to the FTBAR algorithm for any number of tasks. We can see that for small values of CCR there is a little difference in the overhead between the proposed algorithm and the FTBAR algorithm, but for higher values of CCR, the proposed algorithm has a significantly better overhead than the FTBAR algorithm.

6. Conclusions and future work

In this paper a reduced communications fault tolerant algorithm is proposed, for scheduling DAG tasks in multiprocessor systems. The proposed algorithm is based on a previously published algorithm which used active replication to schedule *npf*+1 replicas of each task on different processors to tolerate a given number *npf* of processor failures. In the original algorithm, communications between tasks are duplicated (*npf*+1)^2 times. The reduced communications version proposed in this paper decreases the duplication in communications to (*npf*+1).

In the proposed algorithm, the processors are considered fully connected with identical and non-faulty links. While this can be appropriate in multiprocessor systems, extensions might be added to this algorithm to take communication links heterogeneity and failures into account, and make the algorithm relevant to other distributed computing systems. Another extension that can be added to the proposed algorithm is the consideration of deadline for the task completion times. These deadlines might be considered when sorting the nodes in the list scheduling process. Additionally, the algorithm should be able to detect the feasibility of these deadlines through the scheduling process, an interesting feature of the algorithm when scheduling large applications.

References

[1] Sinnen O. *Task Scheduling for Parallel Systems.* 1st ed. New Jersey: John Wiley & Sons Inc.; 2007.

[2] Dubrova E. *Fault Tolerant Design: An Introduction.* Draft, Kluwer Academic Publishers; 2008.

[3] Tabbaa N, Entezari-Maleki R, Movaghar A. A Fault Tolerant Scheduling Algorithm for DAG Applications in Cluster Environments. In: *International Conference on Digital Information Processing and Communication*, Springer-CCIS, Ostrava, Czech Republic, 2011; **188**:189-199.

[4] Kwok YK, Ahmad I. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys* 1999;**31**(4):406-471.

[5] Girault A, Kalla H., Sighireanu M, Sorel Y. An Algorithm for Automatically Obtaining Distributed and Fault Tolerant Static Schedules. In: *International Conference on Dependable Systems and Networks*, San Francisco, CA, USA, 2003, p. 159-168.