An Efficient Strategy for Task Duplication in Multiport Message-Passing Systems

Dingchao Li[1], Yuji Iwahori[1], Tatsuya Hayashi[2] and Naohiro Ishii[3]

[1] Educational Center for Information Processing
[2] Department of Electrical and Computer Engineering
[3] Department of Intelligence and Computer Science
Nagoya Institute of Technology
liding@center.nitech.ac.jp

Abstract. important problem in compilers for parallel machines. In this paper, we present a duplication strategy for task scheduling in multiport message-passing systems. Through a performance gain analysis, we establish a condition under which duplicating a parent task of a task is beneficial. We also show that, by incorporating this strategy into two well-known priority-based scheduling algorithms, significant reductions in the execution time can be achieved.

1 Introduction

Scheduling is a sequential optimization technique used to exploit the parallelism inherent in programs. In this paper, we consider the problem of scheduling the tasks of a given program in a multiport message-passing system with the aim of minimizing the overall execution time of the program. Researchers have investigated two versions of this problem, depending on whether or not task duplication (or recomputation) is allowed. In general, for the same program, task scheduling with duplication produces a schedule with a smaller makespan (i.e., total execution time) than when task duplication is not allowed.

Some duplication based scheduling algorithms have been introduced in [2,4, 5,7]. However, all of these algorithms assume the availability of unlimited number of processors. If the number of processors available is less than the number of the processors they require, there could be a problem. In this paper, we aim to develop an efficient and practical algorithm with task duplication for scheduling tasks onto a bounded number of available processors. In this area, the work described in [8] is close to ours. The algorithm determines the tasks on the critical paths of a given program before scheduling and tries to select such tasks for duplication in every scheduling step. However, a critical path may be shortened and new critical paths may be generated during the scheduling process, because assigning a task and a parent task of the task to a single processor makes the communication overhead between them become zero. Therefore, our algorithm does not attempt to identify whether a task is critical. Instead, it dynamically analyzes the increase over the program execution time, created possibly by duplicating a parent task of a task, to guide a duplication decision. We will show that incorporating this strategy with some known scheduling algorithms such as the mapping heuristic [9] and the dynamic level method [10] can improve their performance without introducing additional computational complexity.

The remainder of the paper is organized as follows: system and program models are described in Section 2 and in Section 3, we present a duplication strategy for task scheduling. The performance evaluation of the strategy is discussed in Section 4. Finally, Section 5 gives some concluding remarks.

2 System and Program Models

We consider a system consisting of m identical programmable processors $P_i (i = 1 \cdots m)$, each with a private or local memory and fully connected by a multiport interconnection network. In this system, the tasks of a parallel program mapped to different processors communicate solely by message-passing. In every communication step, a processor can send distinct messages to other processors and simultaneously receive multiple messages from these processors. We assume that communication can be overlapped by computation and the communication time can be neglected if two tasks are mapped to a processor.

A parallel program is modeled as a weighted directed acyclic graph, or task graph. Let $G = (\Gamma, A, \mu, c)$ denote a task graph, where Γ is a finite set of tasks $T_i (i = 1, \cdots, n)$, A is a set of directed arcs representing dependence constraints among tasks, μ is an execution time function whose value $\mu(T_i)$ (time units) is the execution time of task T_i, and c is a communication cost function whose value $c(T_i, T_j)$ denotes the amount of communication from T_i to T_j. The arc (T_i, T_j) from T_i to T_j asserts that task T_j cannot start execution until the message from T_i arrives at the processor executing T_j. If $(T_i, T_j) \in A$, T_i is said to be a parent task of T_j and T_j is said to be a child task of T_i. In task graph G, tasks without parent tasks are known as entry tasks and tasks without child tasks are known as exit tasks. We assume, without loss of generality, that G has exactly one entry task and exactly one exit task.

For task T_i, let $est(T_i)$ and $lst(T_i)$ denote the earliest start and completion times, respectively. The computation of the earliest start and completion times can be determined in a top-down fashion, starting with the entry task and terminating at the exit task. Mathematically,

$$est(T_i) = \min_{(T_j,T_i)\in A} \max_{(T_k,T_i)\in A, k\neq j} \{ect(T_j), ect(T_k) + c(T_k, T_i)\}, \tag{1}$$

$$ect(T_i) = est(T_i) + \mu(T_i). \tag{2}$$

Similarly, let $lst(T_i)$ and $lct(T_i)$ denote the latest start and completion times of T_i, respectively. The latest start and completion times of T_i can be given by

$$lct(T_i) = \min_{(T_i,T_j)\in A} \{lst(T_j) - c(T_i, T_j)\}, \tag{3}$$

$$lst(T_i) = lct(T_i) - \mu(T_i). \tag{4}$$

Note that the latest start time of T_i indicates how long the start of T_i can be delayed without increasing the minimum execution time of the graph.

3 Scheduling with Duplication

For machine and program models described above, this section develops a duplication based algorithm in an attempt to find a one-to-one mapping from each task to its assigned processor such that the program execution time is minimized. Our algorithm consists of two parts. The first part decides which ready task should be assigned to an idle processor. This part is heuristic in nature and we can make the choice by applying one of the existing strategies such as HLF(Highest Level First), ETF(Earliest Task First), LP(Longest Path), LPT(Longest Processing Time), and CP(Critical Path). Given the task in the first part, the second part decides whether a parent of the task should be duplicated.

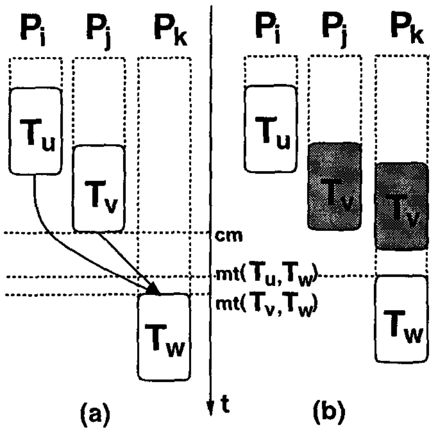

Fig. 1. The task duplication strategy. (a) The schedule before duplication. (b) The schedule after duplication.

Before discussing our task duplication strategy, we introduce *cm* to represent the current time of the event clock, $P(T_u)$ to represent the processor that processes task T_u and $st(P(T_u), T_u)$ to represent the time when T_u starts execution on $P(T_u)$. We also assume that processor P_k is idle at *cm* and task T_w is selected for scheduling, as shown in Fig. 1. The start time at which task T_w can run on processor P_k in case of no task duplication is thus given by

$$\begin{aligned} st_{nocopy}(P_k, T_w) = & \max_{(T_u,T_w)\in A, P(T_u)\neq P_k} \\ & \{cm, st(P(T_u), T_u) + \mu(T_u) + c(T_u, T_w)\}. \end{aligned} \tag{5}$$

Consider the fact that if the start time of a task is later than the latest start time of the task, then the overall completion time of the whole graph will be increased. Thus, we can define the increase incurred due to the execution of task T_w on processor P_k as follows.

$$\delta_{nocopy}(P_k, T_w) = st_{nocopy}(P_k, T_w) - lst(T_w). \tag{6}$$

Obviously, if $\delta_{nocopy}(P_k, T_w) \leq 0$, we should stop duplicating the parent tasks of T_w and assign T_w onto processor P_k, since duplicating its parent tasks is unable to shorten the schedule length even if its start time is reduced.

Assume now that $\delta_{nocopy}(P_k, T_w) > 0$. To minimize the overall execution time, we need to consider task duplication for reducing this increase as much as possible. Let us use $mt(T_u, T_w)$ to denote the time when the message for task T_w from T_u arrives at processor $P(T_w)$; i.e., $mt(T_u, T_w) = st(P(T_u), T_u) + \mu(T_u) + c(T_u, T_w)$ if $P(T_u) \neq P(T_w)$. From Fig. 1, it is clear that we should select a task T_v, as the candidate for duplication, such that

$$mt(T_v, T_w) = \max_{(T_u, T_w) \in A, P(T_u) \neq P(T_w)} \{st(P(T_u), T_u) + \mu(T_u) + c(T_u, T_w)\}.$$

Further let $it(P_k)$ denote the time when processor P_k becomes idle. Thus, the time at which T_v can start execution on P_k can be calculated below.

$$\begin{aligned} st(P_k, T_v) = & \max_{(T_s, T_v) \in A, P(T_s) \neq P_k} \\ & \{it(P_k), st(P(T_s), T_s) + \mu(T_s) + c(T_s, T_v)\}. \end{aligned} \tag{7}$$

Consequently, in this case, the time at which task T_w can start execution on processor P_k is that

$$\begin{aligned} st_{copy}(P_k, T_w) = & \max_{(T_u, T_w) \in A, P(T_u) \neq P_k, u \neq v} \\ & \{cm, st(P(T_u), T_u) + \mu(T_u) + c(T_u, T_w), st(P_k, T_v) + \mu(T_v)\}. \end{aligned} \tag{8}$$

This produces an increase as follows.

$$\delta_{copy}(P_k, T_w) = st_{copy}(P_k, T_w) - lst(T_w). \tag{9}$$

Comparing $\delta_{copy}(P_k, T_w)$ with $\delta_{nocopy}(P_k, T_w)$, we can finally establish the following condition: duplicating a parent task of T_w to the current processor should only occur if

$$\delta_{copy}(P_k, T_w) < \delta_{nocopy}(P_k, T_w). \tag{10}$$

The above duplication strategy clearly takes $O(p)$ times in the worst case, where p is the maximum number of parent tasks of every task in G. Therefore, incorporating this strategy into some known scheduling algorithms such as the mapping heuristic [9] and the dynamic level method [10] does not introduce additional time complexity, since these algorithms generally require $O(nm)$ times for selecting a ready task and an idle processor for assignment, where n is the number of tasks and m is the number of processors.

4 Experimental Study

This section presents an experimental study of the effects of the task duplication strategy described in Section 3. By incorporating the strategy into the mapping heuristic (MH) and the dynamic level method (DL) without global clock heuristic, we implemented two scheduling algorithms on a Sun workstation using the programming language *C*. Whereas we compared the performance of each enhanced algorithm with that of the original algorithm under various parameter settings, due to space limitations, here we show only some of the salient results.

In our experiments, we generated 350 random task graphs and scheduled them onto a machine consisting of eight identical processors, fully interconnected through full-duplex interprocessor links. The task graphs generated randomly ranged in size from 40 to 160 nodes with increments of 20. Each graph was constructed by generating a directed acyclic graph and removing all transitive arcs. For each vertex, the execution time and the number of the child tasks were picked from an uniform distribution whose parameters are input data. The communication time of each arc was determined alike. The communication-to-computation (C/C) ratio value was settled to be 5.0, where the C/C ratio of a task graph is defined as the average communication time per arc divided by the average execution time per task.

The performance analysis was based on the average schedule length obtained by each algorithm. Tables 1 and 2 show the experimental results, where *Impr.* represents the percentage performance improvement of each enhanced algorithm compared with that of the original algorithm. It can be seen that the performance of each algorithm with duplication is consistently better than the algorithm without duplication. The performance improvement compared with the original mapping heuristic varies from 4.01% to 8.17%, and the performance improvement compared with the original dynamic level algorithm varies from 3.42% to 6.90%. These results confirm our expectation that the proposed duplication strategy is efficient and practical.

Table 1. Results from the mapping heuristics with and without duplication.

No. of graphs	No. of tasks	Avg. sche. leng. MH without dup.	Avg. sche. leng. MH with dup.	Impr. (%)
50	40	649.58	609.64	6.15
50	60	860.88	826.34	4.01
50	80	1093.38	1023.16	6.42
50	100	1310.74	1231.68	6.03
50	120	1515.46	1401.70	7.51
50	140	1711.06	1583.58	7.45
50	160	1962.92	1802.51	8.17

Table 2. Results from the dynamic level algorithms with and without duplication.

No. of graphs	No. of tasks	Avg. sche. leng. DL without dup.	Avg. sche. leng. DL with dup.	Impr. (%)
50	40	602.60	572.41	5.01
50	60	788.61	761.66	3.42
50	80	1037.65	993.82	4.22
50	100	1226.73	1172.94	4.38
50	120	1454.92	1367.88	5.98
50	140	1643.25	1536.92	6.47
50	160	1891.74	1761.24	6.90

5 Conclusions

We have presented a novel strategy for task duplication in multiport message-passing systems. The strategy is based on the analysis of the increase over the total program execution time, created possibly by duplicating a parent task of a task. Experimental results have shown that incorporating it into two well-known scheduling algorithms either produces the same assignments as the original algorithms, or it produces better assignments. In addition, the costs, both in implementation effort and compile time, are very low. Future work includes a more complete investigation of the impact of varying the communication-to-computation ratio. Experimental studies on various multiprocessor platforms are also needed.

Acknowledgment

This work was supported in part by the Ministry of Education, Science and Culture under Grant No. 09780263, and by a Grant from the Artificial Intelligence Research Promotion Foundation under contract number 9AI252-9. We would like to thank the anonymous referees for their valuable comments.

References

1. B. Kruatrachue and T.G. Lewis, Grain Size Determination for Parallel Processing, IEEE Software, pp. 23-32, Jan., 1988.
2. J.Y. Colin and P. Chritienne, C.P.M. Scheduling with Small Communication Delays and Task Duplication, Operations Research, vol. 39, no. 4, pp. 680-684, July, 1991.
3. Y.C.Chung and S.Ranka, Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors, Proc. of Supercomputing'92, pp. 512-521, 1992.
4. H.B. Chen, B. Shirazi, K. Kavi, and A.R. Hurson, Static Scheduling Using Linear Clustering with Task Duplication, Proc. of ISCA International Conference on Parallel and Distributed Computing and Systems, pp. 285-290, 1993.

5. J. Siddhiwala and L.F. Chao, Path-Based Task Replication for Scheduling with Communication Costs, Proc. of the 1995 International Conference on Parallel Processing, vol. II, pp. 186-190, 1995.
6. M. A. Palis, J. Liou and D. S. L. Wei, Task Clustering and Scheduling for Distributed Memory Parallel Architectures, IEEE Trans. on Parallel and Distributed Systems, vol. 7, no. 1, pp. 46-55, Jan. 1996.
7. S. Darbha and D.P. Agrawal, Optimal Scheduling Algorithm for Distributed-Memory Machines, IEEE Trans. on Parallel and Distributed Systems, vol. 9, no. 1, pp. 87-95, Jan. 1998.
8. K.K. Kwok and I. Ahmad, Exploiting Duplication to Minimize the Execution Times of Parallel Programs on Message-Passing Systems, Proc. of the sixth IEEE Symposium on Parallel and Distributed Processing, pp. 426-433, 1994.
9. H.El-Rewini and T.G.Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," J. Parallel and Distributed Computing 9, pp. 138-153, 1990.
10. G.C.Sin and E.A.Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," IEEE Trans. on Parallel and Distributed Syst., vol. 4, no. 2, pp. 175-187, 1993.