Extended Abstract

A Parallel Programming Tool for Single Program Multiple Data Model on Distributed Memory Multiprocessors

Yeh-Ching Chung
Department of Information Engineering
Feng Chia University
Taichung, Taiwan 40724, ROC
(04) 252-2250 x3714
e-mail : ychung@pine.iecs.fcu.edu.tw

Sanjay Ranka[*]
School of Computer and Information Science
Syracuse University
Syracuse, NY 13244-4100
(315) 443-4457
e-mail : ranka@top.cis.syr.edu

Abstract – As both the number of processors and the complexity of problems to be solved increase, programming *distributed memory multiprocessors* becomes difficult and error-prone. In a distributed memory multiprocessor, the program partitioning and scheduling play an important role for the performance of a parallel program. However, how to find the best program partitioning and scheduling so that the best performance of a parallel program on a distributed and memory multiprocessor can be achieved is not an easy task. In this paper, we propose a parallel programming tool, PPT, to aid the programmers to find the best program partitioning and scheduling and automatically generate the parallel code for the *single program multiple data* (SPMD) model on a distributed memory multiprocessor.

1. Introduction

Many commercial *distributed memory multiprocessors* have been introduced, such as NCUBE-2 [3] and the Connection Machine 5 (CM-5) [6]. However, it is not an easy task to design a parallel program for a distributed memory multiprocessor. The choice of the execution model, partitioning a problem into processes, grouping these processes into tasks, assignment of each task to a processor, and insertion of synchronization primitives for proper execution [7] have to be performed (manually or automatically). In this paper, we present a parallel programming tool, PPT, for the SPMD model on a distributed memory multiprocessor. The goal of PPT is to aid programmers to design a parallel program that can be run on a distributed memory multiprocessor efficiently (balanced load and low communication cost).

2. PPT

The outline of PPT is shown in Figure 1. PPT has six components: a program partitioning and DAG generator, a DAG analyzer, a scheduler, a communication analyzer, a code generator, and a performance evaluator.

The Program Partitioning and the DAG Generator : To run a program on a distributed memory multiprocessor, the program must be partitioned into tasks. The purpose of the program partitioning is to determine the grain size of tasks such that the best performance of the program on a distributed memory multiprocessor can be achieved. In general, the finer the grain size, the higher the parallelism. However, if a very fine grain partitioning is used, the communication overhead due to sending data from one processor to other processors may greatly increase the execution time of the program. If a coarse grain partitioning is used, a lot of parallelism available in the program

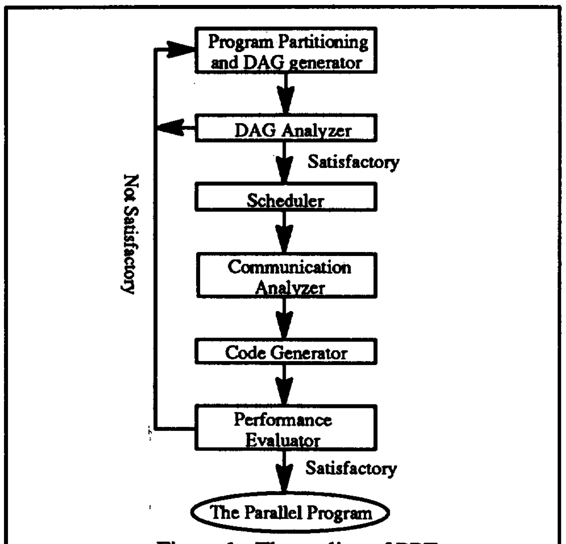

Figure 1 : The outline of PPT.

may be lost. This would result in a low speedup. Therefore, it is important to balance the trade-off between the parallelism and grain size so that a better partitioning can be obtained.

Since PPT is designed for SPMD model, we prefer to use the *modular programming style*, in which a program is composed of a set of procedures called by the main program. There are three advantages by using the modular programming style:

1. The program is easy to design, maintain, and debug.
2. The program partitioning is relatively easy to perform.
3. The DAG can easily be generated from the partitioned program (manually or automatically).

The programmer is required to partition a program into tasks. PPT provides a DAG generator for a programmer to generate the corresponding DAG of the partitioned program in a semi-automatic fashion. If the DAG has a regular structure, the programmer may write a DAG generator to generate the DAG. The programmer is also responsible for the attribute table generation. The attribute table stores the information about the corresponding procedure that a task is associated with, the computation cost of each task, the communication cost between tasks, and the data that must be sent to other processors.

The DAG Analyzer : The DAG analyzer is responsible for the property analysis of a DAG. The properties of a DAG, such as the *graph parallelism* (the ratio of the total computation time of a DAG to the total computation time of tasks on the critical path of a DAG) [5] and the ratio of the average communication cost to the average computational cost (CCR) [2], have great effect on the makespan of a scheduling algorithm and the

* The work of this author was supported in part by NSF under CCR-9110812 and DARPA under contract #DABT63-91-C-0028. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

number of processors that can be efficiently used for execution. For example, if the graph parallelism is equal to 4 and the CCR is less than 1, using the ***highest level first with estimated time*** (HLFET) scheduling algorithm [1] with 4 processors may produce a better makespan than using the HLFET scheduling algorithm with 8 processors. Therefore, it is important to study the relationship between the scheduling algorithms and the properties of DAGs and embed these properties in the PPT.

In [2], we have performed extensive simulation to study the relationship between the list scheduling algorithms and the properties of DAGs. The simulation results show that the graph parallelism and CCR of a DAG are the most important properties that have a great effect on the makespan of a scheduling algorithm. Therefore, the DAG analyzer designed in this tool is responsible for the analysis of the graph parallelism and CCR of the DAG. From the values of the graph parallelism and CCR, the programmer can check if the partitioned program meets the requirement. If it does not, the programmer needs to change the partitioning until the desired partitioning is obtained.

The Scheduler : The scheduler is responsible for selecting a scheduling algorithm and the number of processors for execution (the scheduling algorithm and the number of processors selection can also be make by the programmer), scheduling the DAG on the target machine, and producing a (task, processor) table and the earliest start time table of tasks on processors.

In the current development, PPT provides six list scheduling algorithms, the ***highest level first with estimated time*** (HLFET), HLFET–BTDH [2], HLFET/BTDH [2], the *earlier task first* (ETF) [4], ETF–BTDH [2], and ETF/BTDH [2]. HLFET is a list scheduling algorithm which does not consider the interprocessor communication overhead, while ETF is a list scheduling algorithm that takes the interprocessor communication overhead into account. HLFET–BTDH, HLFET/BTDH, ETF–BTDH, and ETF/BTDH are list scheduling algorithms that use a task duplication heuristic, BTDH, to optimize the makespan. For HLFET–BTDH (ETF–BTDH), BTDH is used as a pure optimization heuristic for HLFET (ETF). For HLFET/BTDH (ETF/BTDH), BTDH is used with HLFET (ETF) to form a new scheduling algorithm. In [2], we have performed extensive simulation to study the relationship between the efficiency of different list scheduling algorithms and the properties of DAGs. A ***relationship table*** is constructed to describe the relation between those scheduling algorithms and the properties of DAGs. According to the output from the DAG analyzer, the scheduler consults the relationship table to find the best candidate scheduling algorithm and decide the number of processors for execution.

PPT allows the addition of a new scheduling algorithm to be added as a member of the list of scheduling algorithms. The only restriction is that the scheduling algorithm should perform scheduling and mapping simultaneously. This is true for nearly all variants of list scheduling algorithms. However, the clustering algorithms proposed in [7] does not fit into this category. Before a scheduling algorithm can be added as a member of scheduling algorithms of the scheduler, the performance evaluation simulator, which is provided by PPT, must be executed for the scheduling algorithm in order to obtain the relationship between the scheduling algorithm and the properties of DAGs.

The Communication Analyzer : The communication analyzer is responsible for detecting the redundant communications, removing the redundant communication, and creating a *communication table*, which will be used by the code generator.

To detect and remove the redundant communications and create the communication table, the communication analyzer builds a *task block table* according to the (task, processor) table and the earliest start time table. In the task block table, some consecutive tasks on the same processor are labeled with the same block number if for those consecutive tasks only the first and the last tasks must send or receive date from other processors. According to the labeled communication table, a new communication table is created and redundant communication is removed.

The Code Generator : The code generator is responsible for generating the corresponding procedure call for each task on processors and inserting the communication primitives. According to the task block table, the corresponding procedure calls for tasks on processors are generated by consulting the attribute table. The communication primitives are inserted according to the communication table. The information about the data that must be transferred to tasks on other processors is provided by the attribute table.

Since the syntax of the basic communication primitives are machine dependent, there is a need for a communication primitives insertion routine. In the current development, we provide two communication primitives insertion routines for NCUBE–2 and CM–5, respectively. One can potentially use syntax of commercial packages like EXPRESS to achieve portability However, we do not use the communication primitives provided by these softwares in our tool. This is because that the communication primitives provided by these software are implemented by using the communication primitives provided by the machines. The overhead is significant and will usually reduce the performance of a parallel program. For example, on NCUBE–2, the time to execute the communication primitives provided by EXPRESS, in general, is 20% more than the time to execute the communication primitives provided by NCUBE–2.

The Performance Evaluator : The performance evaluator is responsible for executing the parallel program on the target machine and reporting the execution time of the parallel program. If the programmer is not satisfied the performance of the parallel program, it will provide information about the execution time of each processor; and the predicted (simulation) and real (experimental) speedups of the parallel program. It can also point out the bottleneck processors for the programmer.

Since many distributed memory multiprocessors, such as NCUBE–2 and CM–5, provide the execution profiler for the programmer to check the time spent in the various subroutines and functions on each processor, the programmer can use the information provided by the performance evaluator to make further refinement/modification of the program partitioning.

References:

[1] T.L. Adam, K.M. Chandy, and J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," *Communication of ACM*, Vol. 17, No. 12, pp. 685–690, 1974.

[2] Y.C. Chung and S. Ranka, "Applications and Performance Analysis of A Compiler–Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Proceedings of Supercomputing'92*.

[3] J. Hayes and T. Mudge, "Architecture of a Hypercube Supercomputer," *Proc. of International Conference on Parallel Processing*, pp. 653–660, 1986.

[4] J.J. Hwang *et al*, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM Journal of Computing*, Vol. 18, pp. 244–257, 1989.

[5] G.C. Sih and E.A. Lee, "Scheduling to Account for Interprocessor Communication within Interconnection–Constrained Processor Networks," *Proceedings of International Conference on Parallel Processing*, Vol. 1, pp. 9–16, 1990.

[6] L.W. Tucher and G.G. Robertson, "Architecture and Applications of the Connection Machine," *IEEE Computer Magazines*, pp. 26–38, August 1988.

[7] M.Y. Wu and D.D. Gajski, "Hypertool: A Programming Aid for Message Passing Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No.3, pp. 330–343, 1990.