2003－MPS－45 (7)
2003／6／24

メッセージ遅延の影響評価のための並列分散プログラムシミュレータの研究

天方貴久, 品野勇治, 中森眞理雄
東京農工大学 工学部 情報コミュニケーション工学科

概要:並列分散プログラムの一つであるメッセージパッシング方式のプログラムでは, メッセージの予測不可能な遅延がプログラムの性能に大きく影響し, 実際の並列計算機環境では性能の評価が困難である. 本論文では, メッセージパッシング方式の並列分散プログラムのメッセージ遅延による性能への影響を評価するためのシミュレータ開発について報告する. 開発したシミュレータを用いた数値実験により, 通信遅延が性能にあまり依存しないプログラムと, 通信遅延が性能に大きく依存するプログラムのシミュレーションの精度を調べ, シミュレータの有効性と問題点を述べる.

A Simulator for Message-Passing Based Parallel and Distributed Programs to Evaluate the Influence of Message Transfer Latency

Yoshihisa Amakata, Yuji Shinano and Mario Nakamori
Department of Computer Science,Tokyo A&T University

Abstract *Within the family of parallel and distributed programs, those that are based on message-passing are frequently characterized by unpredictable delays in message arrival times, which lead to significant differences in program performance. This paper reports the development of a simulator that measures program performance under parallel and distributed computing environments with variable parameters. Also, two types of numerical experiments are presented: one that was conducted on a program of matrix multiplication whose performance does not depend as much on communication delays and another experiment conducted on a program of branch and bound technique whose performance is more dependent on communication delays. Both experiments were performed in the simulator presented here, as well as in a real PC cluster environment. Through these numerical experiments, the effectiveness and problems of the simulator are discussed.*

1 Introduction

As parallel and distributed computing environments such as the PC cluster or the GRID become more popular, high performance parallel and distributed programs (hereafter, abbreviated simply as *parallel programs*) become increasingly more necessary. Among such parallel programs, those that are based on message passing frequently demonstrate unpredictable delays in message arrival time, which lead to significant differences in program performance. Therefore, performance evaluation of message passing parallel programs has become an important research topic, and a wide variety of research approaches have been proposed[2, 3, 4, 5]. In order to evaluate the performance of a parallel program, the authors have developed software that simulates the execution of programs using a message passing library in a parallel environment. In this paper, an overview of the simulator is given at first. Then, results are presented from numerical experiments on a PC cluster environment. These experiments were carried out in order to verify message passing parallel programs in the simulator. Finally, through comparing the above numerical experiments, the effectiveness and problems of the simulator are discussed.

2 Simulator Prototype

The following assumptions are made for the target parallel program whose performance is to be evaluated by the simulator:

- Each computer has one execution task.
- "Non-blocking send", "blocking receive", and "non-blocking receive" are all available message passing methods. More specifically in receiving, "selective receive" is available depending on the message tag or the sender process.
- New processes can be spawned to another computer by sending an invoking message.

The simulator is implemented in Java and an abridged Java interface of the message-passing library for PVM (Parallel Virtual Machine) was also developed.

2.1 The simulation model

A model presented for the execution and simulation of programs using a message-passing library in a parallel environment. The parallel computation environment is modeled by discrete event simulation with sequential direct execution. The model used is much simpler than that presented in [2, 3, 5], because the target machine for the first prototype simulator was a sequential computer.

The target computer is modeled as a *logical computing resource* in the simulator and the execution processes are modeled as *logical processes*(LPs) each of which has its own ID. There is a one-to-one correspondence between the logical computing resources and the LPs, which is based on the aforementioned assumption on the target parallel program.

Processes of message-passing parallel programs send, receive and interpret messages to each other. The path of such message-passing is modeled in the simulator as a *logical communication channel* through which messages are passed among LPs. The values of three parameters can be set in the simulator: a) the delay of signal transfer, b) the delay determined by the transfer rate, and c) the overhead of message processing. These parameter values can be set independently among target computers.

The delay of signal transfer t_i, which is the overhead in the beginning of the transfer, is calculated as $t_i = \alpha\frac{D}{C}$ where C is the signal transfer speed [m/sec], D is the distance of the transfer [m], and α is the coefficient of the delay of the signal transfer. The bandwidth delay of the communication channel is calculated as $t_d = \frac{S}{W}$, where S is the size of the messages [bit], and W is the transfer rate [bit/sec]. Denoting the constant of the overhead time for the preparation of the message transfer by t_c, we have for the time t_m of message transfer $t_m = t_i + t_d + t_c$ .Finally, the time t_g required for generating a process is expressed as $t_g = t_i + t_s + t_c$, where the constant t_s is the time required for starting a process.

As the parallel computation is simulated, each LP is executed sequentially on a single computer. There is, however, dependence among processes, which sometimes causes process interruption. Therefore, physical time of execution of a process is not continuous. The interruption time of a process includes the execution time of other processes and the waiting time of that process for messages from other processes. That is why the simulator has to manage the time of each LP, which is hereafter called *virtual time.*

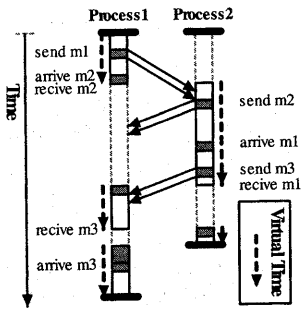

Figure 1: Simulation on a single sequential computer

2.2 An example of a simulation

Let us consider a parallel program, which consists of two execution processes. Suppose that messages are passed as shown in Figure 1, where all receives are "blocking receives", and the shaded area represents the waiting time for sending/receiving messages.

Figure 1 shows how the simulation is executed sequentially on a single computer in a physical time. Virtual time advances only along broken arrows.

2.3 The architecture of the simulator

Figure 2 depicts the simulator architecture. The simulator manages all LPs as an "execute" process and "wait" processes in a priority queue. "Wait" processes are executed according to their priority. The "execute" process is connected respectively to each "wait" process in the queue by a logical channel. Also, there is a one-to-one correspondence between LPs and logical computer resources. Each computer resource maintains a message queue in order to buffer messages from other LPs.

In order to make sharing the execution code and switching LPs easier, the LP is implemented as a light one. The virtual time of a LP is calculated by the last execution time and the time when a message is received. A LP having the oldest virtual time is executed

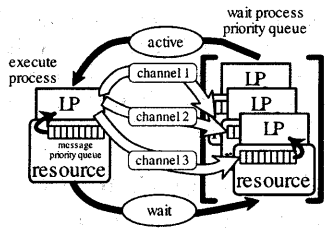

Figure 2: The architecture of the simulator

first, and then the virtual times of all LPs are renewed.

3 Computational Experiments

In order to evaluate the performance of the simulator, computational experiments were conducted using two typical parallel algorithms in two different environments and the results were compared: one environment is a PC cluster; the other is the simulator on the PC cluster where the number of PCs were limited to one, which is equivalent to a simulator on a single computer. The difference of each result is evaluated using the value E as follows: $E = (T_s - T_r)/T_r$,where T_s is the time measured by the simulator and T_r is running time in the PC cluster.

The PC cluster is composed of 15 computing resources, each resource has PentiumII 400MHz processor and 256MB main memory.

Table 1: Simulation parameters of the PC cluster

parameter	value
transfer rate W[Mbit/s]	30
overhead time of transfer t_c[μs]	300
task generating time t_g[s]	0.7
transfer distance D[m]	2
signal transfer speed C[m/s]	3×10^8
coefficient of the delay of signal transfer α	3

First, a preliminary experiment was conducted in order to measure the performance of communications and with that experiment the system parameters were determined as in Table 1 from the measured values and some theoretical characteristics. The value of the coefficient α of the delay of signal transfer was set to three because three TCP/IP packets(SYN, ACK/SYN and ACK) are necessary for establishing a connection.

3.1 Matrix multiplication

As a typical example in which the message transfer delay does not affects the performance of the parallel program so much, a parallel algorithm of square matrix multiplication was implemented. Each slave computes the divided part and returns the result and the master combines the individual results from the slaves.

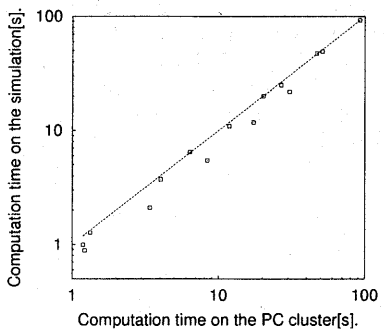

Figure 3: Average computational time of matrix multiplication by PC cluster and simulator

The computation was performed for a number of slaves p equal to 2, 6 and 11, and for a size of matrices n equal to 200, 400, 600 and 800. Each computation was performed on the simulator and on the PC cluster and the experiment was executed ten times for each environment. The average computational time is summarized in Figure 3, where the broken line along the diagonal is the theoretical computational time.

The correlation ratio of the average computational time on the PC cluster and on the simulator is 0.99, the mean value of E is -0.14 and the variance of E is 0.02. This shows that the simulator provides a high precision prediction when the delay of communication transfer has little effect on the characteristics of the algorithm.

3.2 Branch and bound method

As a typical example in which the delay of message transfer has a larger effect on the performance of the parallel program, a branch and bound method for the 0-1 knapsack problem was implemented and numerical experiments were conducted.

The algorithm of the slave process branches the subproblem received from the master process in depth first search and renews the incumbent value of all other slave processes, when its incumbent value is renewed. Also, the algorithm bounds each branched subproblem by the upper

bound obtained by a simple continuous relaxation. When the slave process completes all searches, it sends the incumbent value and incumbent solution to the master process and terminates.

Problems were generated by the advanced generator in Knapsack Problems Generation of test instances [1]. The number of items is 100, 250, 500, 750, 1000, 1250, 1500, 1750 and 2000, for each of which a random number generator generated ten problems, so that the utility and the weight of the items have no correlation.

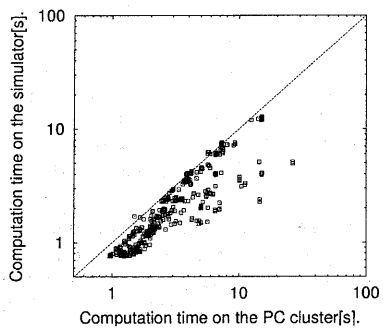

Figure 4: Computational time of branch and bound method by PC cluster and simulator

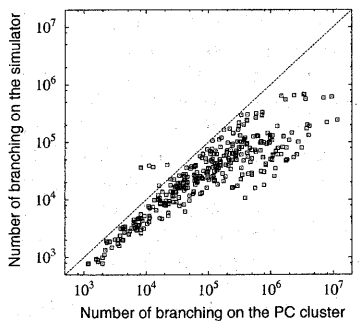

Figure 5: The number of branching of branch and bound method by PC cluster and simulator

The relation between the computational time by simulation and that by PC cluster is depicted in Figure 4. Also, the relation between the number of branching by simulation and that by PC cluster is shown in Figure 5. The mean value of E is -0.31, the variance of the E is 0.04 and the number of branching is almost proportional to the computational time. In the simulator, there is no high communication traffic and therefore, the incumbent values are transferred quickly, which result in earlier bounding and makes the number of branching less.

4 Concluding Remarks

A simulator has been implemented, which predicts the computational time almost accurately, if the target program is not heavily affected by message communication delays. The simulator can also predict theoretical computational time under an ideal communication environment, if the target program is largely affected by the delay. Hence, the developed simulator is effective especially for predicting the performance of a parallel program under an ideal communication environment. The simulator requires an execution time at least proportional to the number of processors of the target parallel environment, which makes future research necessary in order to accelerate the simulator(e.g., by parallelization).

References

[1] Knapsack problems generation of test instances. www.diku.dk/pisinger/codes.html.

[2] R. Bagrodia and R. Meyer. Parsec: A parallel simulation environment for complex system, 1998.

[3] Rajive Bagrodia, Ewa Deeljman, Steven Docy, and Thomas Phan. Performance prediction of large parallel applications using parallel simulations. In *Principles Practice of Parallel Programming*, pages 151–162, 1999.

[4] Adriana Iamnitchi and Ian T. Foster. A problem-specific fault-tolerance mechanism for asynchronous, distributed systems. In *International Conference on Parallel Processing*, pages 4–14, 2000.

[5] Sundeep Prakash, Ewa Deelman, and Rajive Bagrodia. Asynchronous parallel simulation of parallel programs. *Software Engineering*, 26(5):385–400, 2000.