Performance Prediction for MPI Programs Executing on Workstation Clusters

Phillip M. Dickens (presentor)
Department of Computer Science and Applied Mathematics
Illinois Institute of Technology, Stuart Building, Room 235C
10 West 31st. Street, Chicago, Illinois 60616
Phone: 312-567-3974, Fax: 312-567-5067, email: dickens@homer.mcs.anl.gov

George K. Thiruvathukal
Department of Computer Science and Applied Mathematics
Illinois Institute of Technology (same as above)
Phone: 312-567-5150, Fax: 312-567-5067, Email: gkt@delta.csam.iit.edu

March 1, 1998

Abstract

Performance prediction and/or scalability analysis of parallel programs is an important area of current research, especially as parallel computers have come to dominate the high performance computing arena. To date, most of the research in this area has concentrated on the performance of massively parallel machines such as the Intel Paragon and the IBM SP2. However, such machines are scarce, expensive, and unavailable to large segments of the research community, motivating the use of networks of workstations as large, distributed memory multicomputers.

Even though this approach to distributed computing is wide-spread, we still understand little about the behavior of codes executing on this computational platform. For instance, we would like to understand how an application scales as the size of the program and the number of workstations are simultaneously increased. Also, we would like to predict the behavior of codes executing under differing network mediums and under varying network loads.

The MPI (Message Passing Interface) message passing library is the emerging standard by which distributed computers communicate and synchronize, and we are therefore interested in performance prediction of codes executing on top of this library. In this paper, we investigate the use of direct-execution simulation to study the behavior of large codes, executing on networks of workstations, using the MPI message-passing library. We discuss the difficult issues encountered when building this kind of simulator, and the approach we will take to solve these problems.

Keywords: distributed simulation, direct-execution simulation, performance prediction.

1 Introduction

Networks of workstations are generally viewed as among the most promising computational platforms for the future of parallel and distributed computing. However, we do not yet understand issues such as how application codes scale as the number of workstations is increased, or how application codes behave under differing network loads. It is thus important to develop tools with which we can efficiently predict the performance and scalability of parallel codes executing on top of a workstation cluster.

One promising approach for this type of research is *direct-execution simulation*. In this approach, a *host* computer is used to simulate the behavior of an application code executing on top of some *virtual* machine of interest. If the host and target machine are of the same architecture, this similarity can be exploited by simulating only those details of the target machine not available on the host. For example, a new operating system, a new cache coherence protocol, or a new communication network would require simulation; the execution of the native instruction set would not. Thus the application code executes natively *until* it has an interaction with the simulated virtual machine, at which point the application traps out, and a discrete-event simulator models the impact of the particular call on the state of the virtual machine. From the point of view of the application, it *is* executing on the virtual machine. From the point of view of the discrete-event simulator, the application is a driver, describing activity to be simulated.

A detailed direct-execution simulator is computationally intensive, particularly when simulating a system with multiple processors. However, these costs can be reduced by parallelizing the application code *and* the discrete-event simulator. In previous research, we developed such a parallel direct execution simulator called LAPSE (Large Application Parallel Simulation Environment), where both the application code and the virtual machine simulator are parallelized. Currently, we are modifying LAPSE such that it can predict performance on networks of workstations.

1.1 Background

Direct-execution simulation is a mechanism to predict the behavior of a given application code executing on a *virtual machine* of interest. Given N application processes whose performance on N processors is sought, we use $n <= N$ processors to both execute the application code and simulate its timing behavior. Each workstation is assigned some number of application processes (termed *virtual processors* or *VPs*), and some number of simulator processes. The simulator processes control the execution of the applications processes, and together with the other simulator processes emulate the behavior of the assumed virtual machine. The execution behavior of the application processes executing on the virtual machine is obtained by actually running the application processes– all interactions of the application processes and the virtual machine are trapped and handled by the simulator processes. The simulator processes determine how simulation time advances as a function of the application process execution and simulated virtual machine behavior.

LAPSE has been implemented on the Intel Paragon, and provides excellent performance predictions for a wide range of application codes executing on top of a virtual Intel Paragon [4]. Also, we have ported LAPSE to a network of workstations ([5], [6]) to extend to this platform the performance prediction techniques developed for massively parallel machines.

It turns out that modeling the performance of application codes executing on top of a network of workstations poses at least two significant challenges not encountered when modeling a massively parallel machine. Most importantly, massively parallel systems generally have such powerful communication networks that it is often not necessary to model contention within the network to obtain excellent performance predictions. Both LAPSE

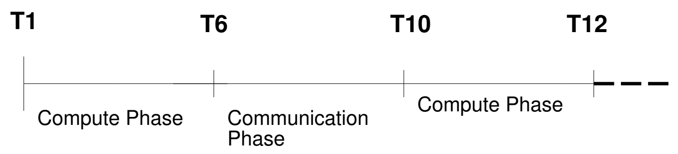

Figure 1: Model of the execution of a virtual processor of the simulated system.

and the WWT [9] project model the communication network of the target machine as a pure-delay network. However, communication networks between workstations are generally much less powerful, and contention within the network becomes a very important performance issue that must be considered by the simulator. This adds complexity to the simulation, and requires a more sophisticated synchronization mechanism than that required for a pure-delay network model.

Secondly, LAPSE and other simulators generally assume an SPMD model of computation, freeing the simulator from having to model the effects of multiprogramming on the behavior of the virtual machine. This assumption is quite reasonable for massively parallel machines with perhaps hundreds of processors. However, networks of workstations are generally much smaller, making it more likely that there will be more than one process executing on a given workstation. Thus it is important that the simulator be able to account for the effects of multiprogramming.

As can be seen, there are important issues that must be addressed when the target machine is a workstation cluster rather than a massively parallel machine. While we cannot discuss these issues fully in this extended abstract, we will do so in the full paper. Our purpose here is to describe the problem, discuss our approach to solving the problem, to briefly discuss related work and indicate the kinds of results we will present in the full paper.

2 Approach

There are two basic approaches to maintaining the fidelity of a distributed simulation. In the *conservative* approach, a process is not allowed to execute an event with a timestamp t if it is *possible* to receive an event with a timestamp less than t at some point in the future. Conservative simulations use blocking to guarantee this condition. In the *optimistic* approach, a process is allowed to immediately execute *any* message it receives, and any out-of-order processing is corrected through a state saving and rollback scheme.

LAPSE (as implemented on the Intel Paragon) uses a conservative *windowing* protocol. In this approach, the simulator processes synchronize frequently to share information about the future behavior of the virtual processes they control. (The ability to predict the future behavior of a virtual processor is termed *lookahead*, and all conservative synchronization protocols must have this feature to perform correctly.) Based on this global information, the simulator processes cooperatively define a window of simulation time such that all events with timestamps within this window can be executed concurrently without the possibility of out-of-order processing.

We model the execution of an application process as iterating between two phases: a *compute* phase and a *communication* phase. The compute phase consists of the code executed *between* calls to the MPI runtime library. The communication phase is the code executed *within* such a call. This model of program behavior is shown in Figure 1.

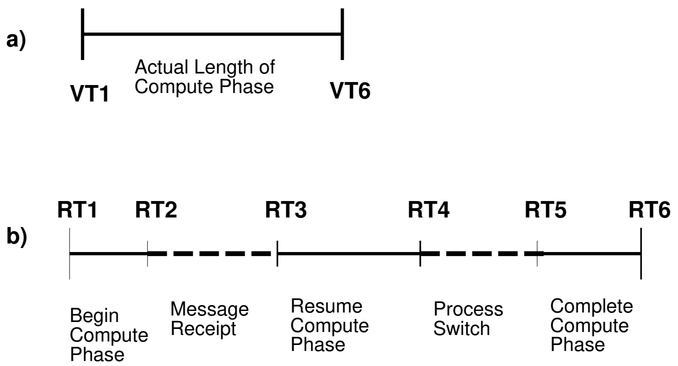

Figure 2: Example demonstrating how a process can be delayed in its execution burst

The simulator must accurately predict the time spent in each phase of execution to provide reasonable estimates of runtime behavior. Consider the estimation of the time a virtual processor spends in one of its compute phases. At first glance, it may appear as if the simulator could simply execute the virtual processors and use a hardware clock to directly measure this time. Unfortunately, this approach fails for two reasons. First, if the number of virtual processors is greater than the number of workstations, then each workstation will have to multiplex perhaps many virtual processors. In this case, the time returned by a hardware clock will include the time an application process is swapped out. Given that a process cannot determine when or for how long it is swapped out, it cannot adjust the value returned by the hardware clock to compensate for this time. Similarly, there is the impact of message traffic. The execution of a process will be interrupted whenever the workstation on which it is executing receives a message. Again there is no way to adjust the time returned by a hardware clock to account for the time such a process is interrupted. The more processes executing on a given workstation, the greater the inaccuracy of timing estimates based on a hardware clock.

As an example of these effects, consider the execution of some virtual processor P1 depicted in Figure 2. The time P1 spends in its compute phase in the *simulated* system spans from virtual time VT1 to virtual time VT6. In real time however, the compute phase of P1 is interrupted from time RT2 to RT3 due to receipt of a message bound for this workstation (and perhaps bound for another virtual processor executing on this same workstation). P1 then resumes its execution, and is swapped out from real time RT4 to RT5 due to a process switch. It regains control of the processor at time RT5 and completes the computation phase at time RT6. As can be seen, there is not necessarily a correlation between the real time and simulated virtual time behavior of an application process.

The question then is how to keep track of *virtual time* behavior when real-time events are affecting the execution of the virtual processors. Traditionally, the amount of virtual time spent in a compute phase is estimated based on the number of instructions executed during the phase. To accomplish this, the application code is instrumented, and the simulator retrieves the instruction count before and after each compute phase. It then uses some calibration to convert from the number of instructions executed to the time spent executing these instructions. As noted, a pure-delay model of the communication network is used to determine the impact of message traffic on the simulated system.

For the reasons discussed above, a simulator modeling a workstation cluster cannot make use of a simple communication network model. However, building a very detailed model of the communication network is undesirable due the additional complexity of the simulator and the increased execution time. We think a better approach is to use a combination of simulation and *analysis* to estimate the message traffic and its impact upon the execution of the virtual processors. The basic idea is as follows.

We model the time spent within a compute phase as having two components. The first component is the base time, representing the time a virtual processor would spend in the computation phase *assuming* no interruptions. The second component is the amount of time the virtual processor is interrupted due to message traffic and/or the effects of multiprogramming. We estimate the base time using the traditional instruction counting approach. We estimate the time a virtual processor is interrupted by message traffic by exploiting the *global knowledge* of the messaging activity during a given window of *virtual time*.

When the simulator processes synchronize to determine the width of the simulation window, they also exchange information about any message passing activity that will occur during the defined window of virtual time. After this synchronization, each simulator process knows all of the virtual processors that will send messages during this interval and the length of these messages, and all of the virtual processors that will post receives for messages. Given this information, and a simple analytical model of the behavior of ether-net under various network loads, the simulator processes can *estimate* most of the parameters in which they are interested. For example, if only one virtual processor will send a message during the current simulation window, then that process should find the full bandwidth of the ether-net available to it. Similarly, if many virtual processors will be sending messages during a particular simulation window, each virtual processor will have significantly less bandwidth available to it. Given an estimation of the available bandwidth, the simulator processes can estimate the number of packets a virtual processor will receive per unit time, and from this estimate derive the time a virtual processor will be interrupted due to the receipt of a given message.

This same approach can be utilized to predict the time spent in a communication phase. Consider for example a virtual processor that will send a message within the current simulation window. If the bandwidth available to this virtual processor at this point in virtual time is known, the amount of time required to complete the message transmission can be estimated.

We defer for now the discussion on including the costs of multiprogramming into the simulation model. We will do so in the final paper.

3 Related Work

Several other projects use direct execution of application processes to drive simulations of multiprocessor systems [1, 2, 3, 9]. The Wisconsin Wind Tunnel (WWT) [9] is, to our knowledge, the only multiprocessor simulator that uses a multiprocessor (the CM-5) to execute the simulation. It is worthwhile to note two important differences between LAPSE and WWT. First is the issue of purpose. The WWT is a tool for cache-coherency protocol researchers, being designed to simulate a different type of architecture than its host. LAPSE is designed primarily for performance and scalability analysis. LAPSE is implemented on an Intel Paragon and a cluster of Sun Sparc workstations, neither of which supports shared virtual memory. Coherency protocols complicate the simulation problem considerably, but are a facet we need not deal with currently. The second difference is the synchronization mechanism used to maintain the fidelity of the parallel simulation. WWT uses a special case of the YAWNS synchronization protocol [7] while the original version of LAPSE uses a synchronization mechanism combining YAWNS with appointments [8]. As noted, we are modifying the synchronization protocol of LAPSE to estimate performance for codes executing on workstation clusters.

We will present a more thorough literature review in the final paper.

4 Preliminary Results

We are still in the developmental stage of this research and do not yet have preliminary results to present. However, by the time the full paper is required for publication, we will have empirical data relating to the performance of our simulator.

5 Conclusions

In this paper, we have clearly defined the problems encountered when building a direct-execution simulator for codes executing on workstation clusters. To address these problems, we have proposed a new approach that incorporates simulation and *analysis* rather than simulation alone. This approach promises accurate performance predictions without the expense or complexity of a full-blown simulation of the communication network.

References

[1] Agrawal, D., Choy, M., Leong, H. and A. Singh. Maya: A simulation platform for distributed shared memories. In *Proceedings of the* 8^{th} *Workshop on Parallel and Distributed Simulation*, pages 151–155, July 1994.

[2] Brewer,E. Dellarocas, N., Colbrook, A. and W. Weihl. PROTEUS: A high-performance parallel-architecture simulator. Technical Report MIT/LCS/TR-516, Massachusetts Institute of Technology, September 1991.

[3] Chen,D., Su,H., and P. Yew. The impact of synchronization and granularity on parallel systems. In *Int'l. Symp. on Computer Architecture*, pages 239–248, May 1990.

[4] Dickens, P., Heidelberger, P. and D. Nicol. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. In IEEE Transactions on Parallel and Distributed Systems, Volume 7, Number 10, October 1996, pages 1090 - 1105.

[5] Dickens, P., Haines, M., Mehrotra, P. and D. Nicol. Towards a Thread-Based Parallel Direct Execution Simulator. In Proceedings of the 29th Hawaii International Conference on Computer Systems, Volume 1, pages 424 - 432, January 3 - 6 1996.

[6] Dickens, P. A Workstation Based Parallel Direct Execution Simulator. In Proceedings of the 11th Workshop on Paralle and Distributed Simulation, June, 1997.

[7] Nicol, D. The cost of conservative synchronization in parallel discrete-event simulations. *Journal of the ACM*, 40(2):304–333, April 1993.

[8] Nicol, D. Parallel discrete-event simulation of FCFS stochastic queueing networks.
In *Proceedings ACM/SIGPLAN PPEALS 1988: Experiences with Applications, Languages and Systems*, pages 124–137. ACM Press, 1988.

[9] Reinhardt,S., Hill, M., Larus, J., Lebeck,A., Lewis,J. and D. Wood. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM SIGMETRICS Conference*, pages 48–60, Santa Clara, CA., May 1993.