FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 18, No. 2, August 2005, 219-224

Teaching tools for parallel processing

Emina I. Milovanović and Natalija M. Stojanović

Abstract: Because many universities lack the funds to purchase expensive parallel computers, cost effective alternatives are needed to teach students about parallel processing. Free software is available to support the three major paradigms of parallel computing. Parallaxis is a sophisticated SIMD simulator which runs on a variety of platforms.jBACI shared memory simulator supports the MIMD model of computing with a common shared memory.PVM and MPI allow students to treat a network of workstations as a message passing MIMD multicomputer with distributed memory. Each of this software tools can be used in a variety of courses to give students experience with parallel algorithms.

Keywords: Parallel processing, simulator, PVM, MPI.

1 Introduction

Parallel computing is information processing that emphasizes the concurrent manipulation of data elements belonging to one or more processor solving a single problem. A parallel computer is a multiprocessor computer capable of parallel processing. Parallel computing has emerged as a key enabling technology in modern computing. The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing and for more ”general-purpose” applications. The trend was a result of the demand for higher performance, lower cost, and sustained productivity. The acceptance has been facilitated by two major developments: massively parallel processors and the widespread use of clusters of workstations. Because many universities lack the funds to purchase expensive parallel computers, cost effective alternatives are needed to teach students about parallel processing. The instuctors, who teach at a university with no parallel computer, regularly teach a parallel processing course

Manuscript received February 2, 2004

The authors are with Faculty of Electronic Engineering, A. Medvedeva 14, 18 000 Niš, Serbia and Montenegro (e-mail: ema@elfak.ni.ac.yu).

and units in other courses using software simulators, a network of workstations, and network access to a major research center that has parallel computers. Our experiences confirm that courses of parallel processing can be taught successfully with a limited budget. The purpose of this paper is to describe the models associated with parallel processing and how they can be introduced in a course.

2 Models of parallel computing

Since the processors depend on some type of communication network to share data, several models for connecting processors and memory modules exist, and each topology requires a different programming model. The three models that are most commonly used in building parallel computers include: (1) synchronous processors each with its own memory (a.k.a. distributed memory), where at each step, all processors execute the same instruction on a different data element. . These computers are often referred to as SIMD (Single Instruction Multiple Data) computers. Unlike a sequential processor, each processor has data elements in its own memory upon which the instructions are performed. Programs that are written for SIMD computers usually operate on single data elements, like array elements. Massively parallel computers typically use this model of computing. (2) asynchronous processors each with its own memory, where each processor operates under the control of an instruction stream issued by its control unit: therefore the processors are potentially all executing different programs on different data while solving different subproblems of a single problem. This means that the processors usually operate asynchronously. Communication between processors is performed by sending a message (data) from one processor to another. Because the time it takes to pass data is much longer than the time it takes to process it, programs written for these computers require that data elements be efficiently divided and stored in the local memories of the processors so that minimum message passing is required. Any needed synchronization of processors is also accomplished using the passing of messages. These computers are referred to as MIMD multicomputer with distributed memory. (3) asynchronous processors with a common, shared memory model, where multiple processors execute asynchronously, but all data is retained in a shared memory. Because updates to shared data must be synchronized to ensure correct results, support is needed for exclusive access to data in the sections of programs where shared data is updated. This requires the use of locks. These computers are referred to as MIMD multicomputer with shared memory. As we have already mentioned software that we use in our course to support these models of parallel processing are:

1) Parallaxis software simulator of SIMD computer

2) jBACI software simulator of MIMD multicomputer with shared memory.
3) PVM and MPI which allow that network of workstations can be treated as MIMD multicomputer with distributed memory.

3 Parallaxis

Parallaxis is a sophisticated SIMD simulator which runs on a variety of platforms. Parallaxis is a machine-independent language for data-parallel programming.[1] Sequential Modula-2 is used as the base language. Programming in Parallaxis is done on a level of abstraction with virtual processors and virtual connections, which may be defined by the application programmer. Each processing element has its own local memory, but all processing elements execute a common instruction stream synchronously. Parallaxis allows the programmer to specify the number of (simulated) processing elements and the topology which connects the elements. Parallaxis supports two types of variables, scalar and vector. Scalar variables reside on the central control unit, and vector variables reside on the processing elements. When writing programs for Parallaxis, the user needs to determine which parts of the algorithm can be executed in parallel and which must be executed sequentially. Vector variables must be specified for the data that will participate in the parallel portion of the algorithm, and corresponding scalar arrays need to be created to support I/O operations. Load and store statements copy data between a scalar array and a vector variable. Operations referred to vector variables are: propagate operations which copy data from one processing element to another and reduction operators, such as a global addition, summarize data across all processing elements. Parallaxis lets students experiment with different connection topologies, and mapping parallel algorithms onto specific topologies, but they do not have chance to experience or experiment with the speedup of a true parallel machine.

4 The Shared Memory Simulator-jBACI

Concurrency concepts and synchronization techniques are important issues in computer science. Due to the increasing emphasis on parallel and distributed computing, understanding concurrency and synchronization is more necessary than ever. To obtain a thorough understanding of these concepts, practical experience writing concurrent programs is needed. ”BACI” stands for Ben-Ari Concurrent Interpreter and is based on the work of M. Ben-Ari first published in his book ”Principles of Concurrent Programming”. It is designed to simulate the operation of concurrent processing and supports several synchronization techniques, such as general semaphores, binary semaphores and monitors.[2] The basis of the BACI system is a pre- emptive scheduler which randomly swaps between concurrent processes during execution. In our course we use jBACI simulator, an integrated development environment for learning concurrent programming by simulating concurrency. It was built from the BACI (Ben-Ari Concurrency Interpreter) compilers developed by Bill Bynum and Tracy Camp, and a modified version of the BACI GUI interpreter written by David Strite. jBACI implements extensions to the BACI source languages that allow graphics programs to be written. The BACI GUI program gives the student the capability to monitor all aspects of the execution of a program: The process table contains a line for each process, giving the process number and name and the concurrency status. The Console window displays the output from the program; this is in addition to the output from each process which is displayed in a pane of the process window. The Globals window displays the values of the global variables. The History window displays the last 150 source or PCode instructions that have been executed. Process window consists of three tabbed panes: the Code pane, the Console pane show the output from the process and the Details pane. shows the contents of the process stack as well as the concurrency status of the process When writing a program for the shared memory simulator, the user needs to identify the shared and process local variables in the program. The data to be manipulated by each processor needs to be as independent as possible. Any accesses, particularly updates, to shared data elements by multiple processors must be coordinated using locks, making that portion of the code sequential and slowing execution considerably. The primary use of the shared memory simulator has been to provide a programming environment for students who are studying the shared memory paradigm as part of a parallel processing course.This means that it can be used as the basis for a unit on parallel algorithms is several different courses. Finally, the simulator can also be used in a course, such as operating systems, where the students are studying the creation of processes and the management of semaphores. In this case, the instructor can provide working programs and a partial simulator.

5 PVM and MPI

PVM is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architecture.[3] The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation. The unit of parallelism in PVM is a task, an independent sequential thread of control that alternates between communication and computation. No process-to-processor mapping is implied or enforced by PVM; in particular, multiple tasks may execute on a single processor. The application's computational tasks execute on a set of machines that are selected by the user for a given run of the PVM program. Collections of computational tasks, each performing a part of an application's workload using data-, functional-, or hybrid decomposition, cooperate by explicitly sending and receiving messages to one another. Tasks access PVM resources through a library of standard interface routines. These routines allow the initiation and termination of tasks across the network as well as communication and synchronization between tasks. The PVM message-passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission. Communication constructs include those for sending and receiving data structures as well as high level primitives such as broadcast, barrier synchronization, and global sum. MPI is another message-passing standard for parallel programming. It has gained more popularity and acceptance in the parallel computing community recently. The goal of MPI is also widely used standard for writing message- passing programs. As such the interface attempts to establish a practical portable, efficient, and flexible standard for message passing.[4] MPI is not intended to be a complete and self-contained software infrastructure that can be used for distributed computing. MPI does not include necessities such as process management (the ability to start tasks), (virtual) machine configuration, and support for input and output.Like PVM, MPI include routines for communication between tasks, involving message-passing primitives for buffering and transmission. Also, MPI include high-level primitives for data transmission(like broadcast, scatter and gather), barrier synchronozation and global computations.[5] Also,implementations in MPI can be used in a heterogeneous environment. Unlike Parallaxis and the shared memory simulator, PVM and MPI are not simulators, they use multiple processors to perform the work. This means that execution times can be captured and analyzed. It can be concluded from observing many parallel algorithms run slower under PVM or MPI than their sequential counterparts. Overhead associated with forking and killing processes and the time it takes to send messages across the network are the main causes of the slowdown. Actual speedup only occurs in algorithms that perform large amounts of computation and require small amounts of data transfer. However, the slowdown opens the eyes of the students as to the limitations of parallel computations, and a discussion of the causes is very educational. Even though slowdowns,the ability to capture timing data does allow the students to make relative comparisons between alternative parallel algorithms.

6 Conclusion

Our experiences confirm that parallel processing course can be taught successfully with a limited budget.In our course we introduced two simulators,Parallaxis(simulator of SIMD computers) and jBACI simulator(simulator of shared mamory MIMD computers). Also, we introduce concept of MIMD computers with distributed memory using PVM and MPI software tools on interconnected computers of varied architecture. In this way, student gets hands-on expirience with models associated with parallel processing.

References

[1] Parallaxis-iii user manual. [Online]. Available: http://www.informatik.uni-stuttgart.de/

[2] jbaci user-guide. [Online]. Available: http://www.mines.edu/fs_home/tcamp/ baci/

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*. Massachusetts,London,England: MIT Press Cambridge.

[4] [Online]. Available: www.mcs.anl.gov/mpi/index.html.

[5] M. Snir, M. Snir, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI-The Complete Reference*. Massachusetts,London,England: The MIT Press Cambridge.