An Undergraduate Parallel Processing Laboratory

Chris Nevison
Colgate University
Hamilton, New York 13346

Abstract

We discuss possibilities for setting up an undergraduate laboratory for parallel processing and how such a laboratory, based on transputers, can be used in a course on parallel processing.

1. *Introduction*

We have recently set up an undergraduate parallel processing laboratory at Colgate University. This laboratory is being used for a new course on parallel processing and it will be incorporated into other courses such as Programming Languages and Analysis of Algorithms. In this paper I will describe some of the alternative approaches which may be taken to set up such a laboratory and what I see to be some advantages and disadvantages of each.

These observations are based on our experience here at Colgate while we investigated different possibilities for a parallel processing laboratory. I will give some estimates of costs for different kinds of equipment. These estimates are based on our explorations but are not by any means current prices. Anyone interested in precise cost estimates should contact the various manufacturers directly.

2. *Modes of Parallel Processing*

There are several kinds of computing devices which can be called "parallel." Going back to the earliest computers, the IAS (Institute for Advanced Study) computer was parallel at the arithmetic operation level--it worked with a 40 bit word. In contrast, the EDVAC was a bit-serial processor. The trade-offs at that time were the same as they are today--speed versus complexity of circuits.

Almost all of today's computers work in parallel at the arithmetic/logic level, with word lengths of 8, 16, 32, and 64 bits. However, parallel computers execute higher level instructions simultaneously, processing many items of data at one time. There are several types of parallel processing: pipelines, vector processing, more general SIMD processing, shared-memory machines, and message-passing multicomputers.

Pipelining refers to the breakdown of instruction execution into a series of stages which can effectively be overlapped. Different processors (or parts of the processor) execute these stages in parallel, thus effectively increasing the throughput of the machine. This method was the basis of the early "supercomputers" like the Cray-1. Many of today's computers use pipelining to one degree or another. However, the model of computation which the user sees in such a computer is the same von Neuman model of serial computation.

Vector processors, which may be implemented using pipelining, are designed to execute instructions efficiently on a linear array of values. These are also referred to as SIMD--single instruction stream, multiple data stream. The SIMD model can be extended to cover arrays of processors, where each processor executes the same instruction on its piece of data and may also reference the data located at nearby processors. In a d-dimensional mesh machine, nearby processors mean those which are nearest neighbors in the mesh connection (North, East, South, West for a 2-mesh). Other SIMD processors, such as the Connection Machine, may allow a greater variety of communications between processors.

In contrast to the SIMD model, an MIMD (multiple instruction, multiple data) computer has many processors, each of which may be running a different process. MIMD machines can generally be viewed as shared-memory or message-passing machines. In a shared-memory machine, all processors access the same memory and can thereby communicate via memory operations. If the memory is all equally accessible to all processors, either via a bus or a switching network, it is called a tightly-coupled multi-processor. If the memory is parceled out among the processors, so that memory accesses may take a variable amount of time according to whether the address is local to the processor or not, it is called loosely-coupled.

In a message-passing architecture or multicomputer, each processor has its own memory. Processors are linked in some communications network and communicate by passing messages from processor to processor. Thus the processing element is directly involved with the communication, whereas it is not in a shared-memory model.

3. *Commercially Available Machines*

It has recently become feasible to buy parallel machines for prices ranging from $20,000 to $200,000 and up. This means equipping an undergraduate laboratory for parallel processing is affordable. I will suggest some alternatives in this price range. My list is probably incomplete, but should serve to give some idea of what is available.

We investigated two machines of the tightly-coupled shared-memory type: the Sequent Balance 8000 and the Encore Multimax. Both of these are bus-based architectures where several processors communicate with shared memory on a common bus system (see Figure 1). Each uses a variety of techniques to enhance the bus band width. A Sequent Balance 8000 with six processors could cost under $100,000, depending on the configuration. Encore recently announced a version of the Multimax with eighteen processors for $150,000 under special university arrangements.

Both of these machines run multiprocessor versions of the Unix operating system and have versions of C, Pascal, and Fortran with parallel constructs added.

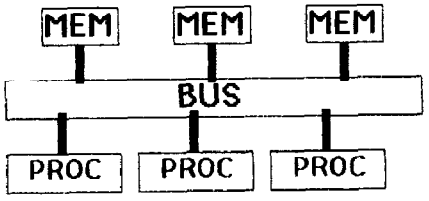

Figure 1

Bus-based Shared Memory Architecture

One advantage of this type of machine is that in addition to providing a parallel processing environment, it would also provide a general purpose Unix environment. In addition, other types of parallel machines are readily simulated on a shared-memory machine. For example, a message-passing MIMD computer can be simulated on a shared memory computer by partitioning the shared memory into local memories for each processor and small segments of memory which would serve as channels for inter-processor communication. Each processor would be restricted to access only its "local" memory and those "channels" to which it was "connected." On the negative side, the number of processors is limited by this architecture and, more realistically, by the budget.

A second type of machine which is available is a hypercube architecture. A hypercube multi-computer consists of 2^d processor nodes, each including a processor and a local memory for that processor. The nodes are connected in a pattern which forms the edges of a d-dimensional cube, with a host processor connected to one or more nodes (see Figure 2). The processors work independently and asynchronously and communicate by passing messages. There are two hypercube machines which have low-end versions at a reasonable cost. Intel has the SugarCube, a version of their iPSC cube machine, with up to sixteen nodes (their full cube can handle up to 128 nodes). NCube, which markets a machine which takes up to 1024 nodes, also has a board for an IBM PC or compatible which has four of these nodes on it. Both machines have operating systems based on Unix. The NCube board for a PC would cost about $10,000 and a SugarCube with four nodes $40,000 or eight nodes $80,000.

These machines provide the basis for experimenting with the message-passing hypercube model. They also provide an expansion path to larger machines and come with an operating system which should make their use easier. On the other hand, the cost per processor is high for an undergraduate laboratory.

A third alternative which might be possible, depending on local circumstances, would be for a department to

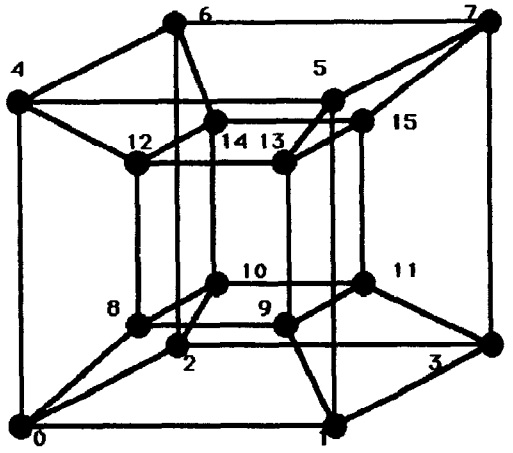

Figure 2

Four Dimensional Hypercube

arrange time-sharing access to a nearby parallel computing facility and whatever machines they have available. This has not been feasible for us at Colgate, so we have not investigated this closely. It seems likely, however, that this approach would not provide for the kind of experimentation desirable for a good laboratory experience.

The alternative which we have taken is to develop parallel machines based on the Transputer, a microprocessor manufactured by Inmos, Ltd. in Bristol, England. Inmos markets an experimental system whereby a host board in an IBM PC compatible machine can be connected to a network in a development system which will hold up to ten boards with four transputers each. Computer System Architects (CSA) of Provo, Utah, market a similar system, except their 4-transputer board goes into a PC expansion slot.

The advantage of a transputer system is that a variety of message-passing architectures can be easily wired, in either the Inmos or CSA systems. The transputers use the language Occam, based on Hoare's CSP (Hoare, 1978). On the other hand, there is nothing like the operating systems available on the other computers discussed. It is very much an experimental system. Finally, the cost is quite reasonable. A sixteen-transputer network, plus host transputer which will go into an IBM PC (5 slots needed) would cost $20-25,000 from CSA. The Inmos development system with 40 transputers, plus the host board for a PC, would cost $55-65,000.

Recently, another vendor, Microway, has announced a transputer-based add-in board for the IBM PC and compatibles. Their products are similar to the CSA boards.

4. *A Transputer-based Parallel Processing Laboratory*

At Colgate, we chose to develop a laboratory based on transputers. We were most interested in creating a lab where students could experiment as much as possible and confront some of the fundamental problems of parallel processing. The transputers enable us to experiment with different physical architectures, but using a common language--Occam.

With support from grants from the NSF CSIP program, the Culpeper Foundation, and Borland International, we developed a lab with three workstations. Each workstation comprises a Zenith-248 (PC AT equivalent) containing a transputer development board (one transputer, 2 megabytes

memory) and four 4-transputer boards, each transputer having 256K memory. The transputer boards were manufactured by CSA and the Transputer Development System for programming in Occam came from Inmos. We have the capability of wiring transputers in all three machines together to make one large network, but for our course in parallel processing we use them as three workstations. We use each transputer workstation as a four-dimensional hypercube (see Figure 2). The hypercube architecture combines a relatively low number of connections with a short maximum distance between processors and an easy addressing scheme. In a cube with $p = 2^d$ processors, the total number of links between processors is (p log p)/2 and the maximum distance is log p. Furthermore, every cube has an embedded mesh, such as the four by four mesh embedded in a four-dimensional cube (Figure 3) and a binary tree maps conveniently into a cube (Figure 4). If the nodes of a cube are numbered by having each bit represent a dimension, with nodes located at the corners of a d-dimensional unit cube, then the neighbors of a node have addresses that differ in exactly one bit (see Figure 2).

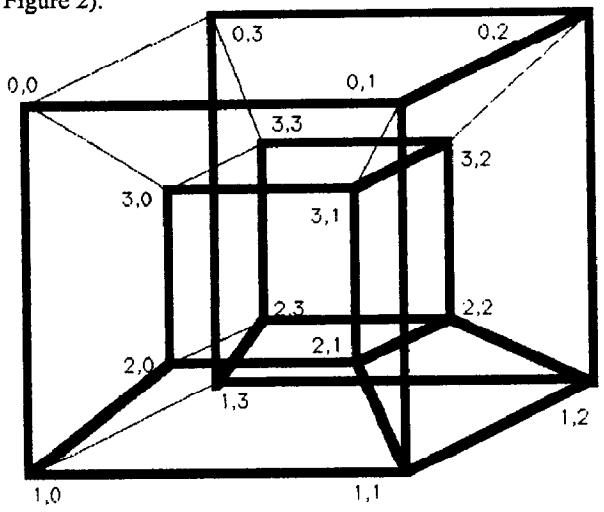

Figure 3

A 4x4 mesh embedded in a four dimensional hypercube. Adding the lighter lines makes a toroidal mesh.

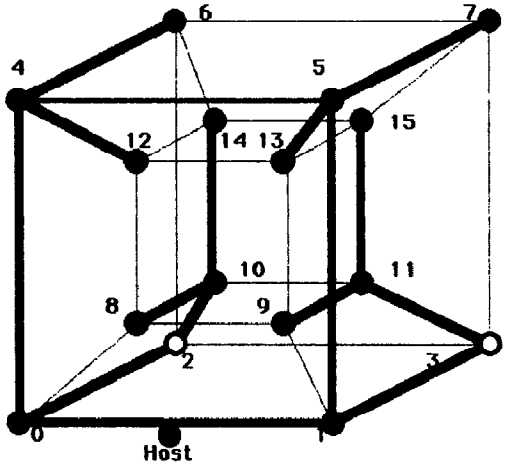

Figure 4

A Binary Tree Embedded in a Hypercube with Host between Nodes 0 and 1. White Nodes pass on Messages.

Another architecture which we can experiment with is the shuffle-exchange. In this interconnection scheme even to odd nodes are connected and connections corresponding to a perfect shuffle are included. We also include odd to even connections (see Figure 5). This architecture shares some of the advantages of the hypercube, such as a log p maximal distance between nodes, but has the advantage of a constant number of connections at each node. For example, a cube of dimension higher than four cannot easily be done with transputers since they have only four links per node, but a shuffle exchange of any size can be constructed.

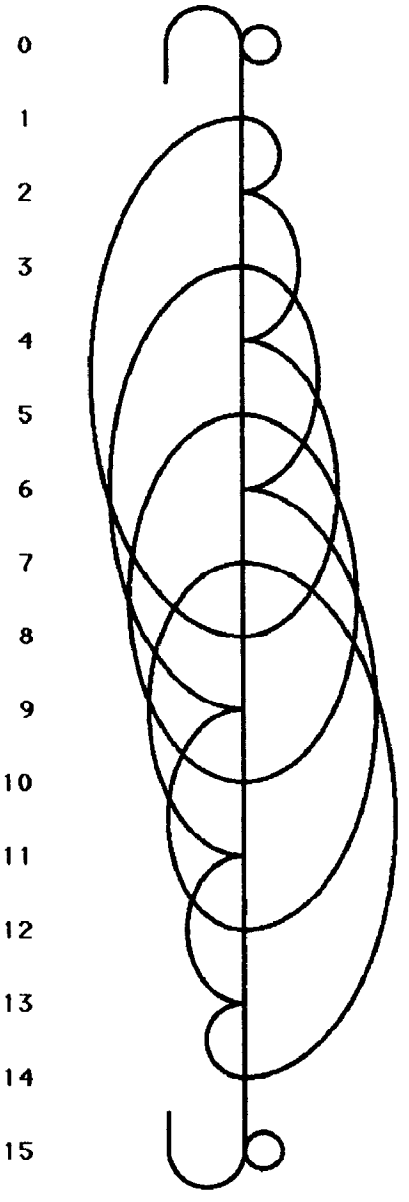

Figure 5

Sixteen Node Shuffle-Shift

One advantage of working with transputers is that we can experiment with these different architectures and several others, rather than being limited to a single configuration.

We are teaching an experimental course on parallel processing during the Fall semester, 1987. The course begins with the study of parallel architectures, based on the book Parallel Computing, Theory and Comparisons (Lipovski and Malek, 1987). The objective of this part of the course is to make students familiar with the variety of approaches to parallel computing and the strengths and weaknesses of each. Each student does a presentation and a paper on a particular machine or architecture. This takes just under half the course.

For the rest of the course we study parallel algorithms. Our objective is to understand some of the methods for developing parallel algorithms and to analyze the performance of these algorithms, especially their speedup over serial algorithms and their asymptotic behavior as either the number of processors grows or the size of the problem grows. Typically, we consider how each algorithm can be mapped onto several different parallel architectures and the advantages or disadvantages of each. We use the text Designing Efficient Algorithms for Parallel Computers (Quinn, 1987), for this part of the course. Each student is responsible for doing a presentation on a particular algorithm, including a parallel implementation on one of our transputer systems.

Because we were only able to set up our parallel machines a few weeks before this class began, the laboratory for this first time is experimental. Our objective for the laboratory is to enable students to experience some of the particular difficulties with parallel programming and to learn some of the techniques used in parallel programming.

During the first three weeks of the semester, the students are required to become familiar with the Transputer Development System, an environment produced by Inmos for writing programs in the language Occam. In addition, they must become familiar with the language Occam, in particular the idea of passing messages between processes over channels and the constructs for executing subprocesses in parallel. This can be done on a single transputer, so at this point students need not learn the details of loading a set of programs onto a network. The students are responsible for writing a simple program using a pipeline of communicating processes.

During the next two weeks, the students must learn the steps for setting up and configuring a set of programs for a hypercube. They experiment with variations on a simple communications program to become familiar with this process and the structure of the cube.

Finally, students must do a parallel implementation for the algorithm which they present in class and use this implementation to test the efficiency of the algorithm. They may use either the hypercube, 4x4 mesh, or shuffle exchange for their implementation. During the past semester students studied a variety of algorithms for their final projects. The development of a non-deadlocking communications system for a hypercube: since the hypercube runs asynchronously, it is possible for cyclic deadlock to occur when each processor in a cycle is trying to send to the next one. In this situation the message-passing in the cube will come to a halt and the computation stops. Although buffering reduces the probability of such deadlock, it cannot eliminate it. Consequently Scott Cost, who worked on this project, developed a scheme of token-passing to avoid deadlock.

A second project was the development of a systolic matrix multiplication algorithm. Although the implementation of this type of algorithm on an asynchronous machine involves a high overhead for synchronization, we expect to see significant speedup of computation for large arrays (experiments to determine actual speedups have not been completed at the time this is written).

Another student is adapting the Fast Fourier Transform algorithm to the hypercube architecture. This is an algorithm involving a binary divide and conquer using recursion. Algorithms of this type can be conveniently mapped to a hypercube by having an active node pass one recursive call to a neighboring node and continue itself with the other.

Other algorithms which students have explored include a maze routing algorithm for routing wires in VLSI work, image analysis on a hypercube, and dictionary look-up using a hypercube.

5. *Future Prospects*

After working through the seminar on parallel processing once, I have formulated some ideas on how to revise the course in the future. Some of these ideas could not be implemented the first time around, simply because we did not have the equipment in time to do the necessary development.

With equipment in place and some prepared lab assignments, I would plan to put less emphasis on architectures and more on algorithms than was possible the first time. Nevertheless, it is important for a course introducing parallel processing to survey the variety of approaches and architectures which are available. I would plan three weeks on this part of the course. The remaining nine weeks of the course would be spent studying parallel algorithms. I would organize the course around different types of algorithms so that some time would be spent discussing each of the following: low-communication/high-compute algorithms which can be done on a simple architecture such as a pipeline; local communications algorithms, such as low-level image processing (edge resolution algorithms); systolic algorithms such as matrix multiplication on a mesh; basic building blocks for higher level algorithms including parallel prefix operations, array packing, bucket sort; recursive algorithms which map naturally onto a cube such as the FFT example given above (several kinds of algorithm fall into this class); sorting algorithms such as bitonic sort. For each group of algorithms I would provide students with an example, and require them to develop a similar algorithm using the same approach, or experiment with the given algorithm by doing timing analysis on variation of the implementation. As a result of the experimental version of the course I have preliminary implementations of parallel prefix summation, systolic matrix multiplication, and FFT's. I hope to have a few more well worked examples ready as well as refinements of these, before I next teach this course.

I will organize the laboratory component of the course so that students will do something like the following sequence:

1. Fundamentals of Occam programming on a single processor.
2. A pipeline program on a single processor.

3. A pipeline program on a pipeline of processors.
4. Experimentation with systolic matrix multiplication -- an empirical study of speedup.
5. Building a larger program from some low-level operations such as parallel prefix summation.
6. A comparison of two different implementations of FFT's.
7. Individual projects.

6. *Summary*

A variety of machines are now available which make a parallel processing laboratory for undergraduates feasible for a reasonable cost. The selection of machines may depend on several criteria, including their use beyond parallel programming. At Colgate we selected a transputer-based system because it provides us with an experimental machine which we can configure in several ways. The students must get close to the operations of the parallel architecture and thereby learn about many of the issues fundamental to parallel processing. We expect that in the future we can add units involving parallel processing and making use of this equipment to several of our other upper-level courses.

References

Hoare, C. A. R., 1978. "Communicating Sequential Processes," Communications of the ACM, vol. 21, pp. 666-677.

Lipovski, G. J., and M. Malek, 1987. Parallel Computing, Theory and Comparisons. New York: John Wiley & Sons.

Quinn, M. J., 1987. Designing Efficient Algorithms for Parallel Computers. New York: McGraw-Hill.