Parallel Computing to Start the Millennium

Joel Adams
Computer Science
Calvin College
Grand Rapids, MI 49546
adams@calvin.edu

Chris Nevison
Computer Science
Colgate University
Hamilton, NY 13346
chris@cs.colgate.edu

Nan C. Schaller
Computer Science
R.I.T.
Rochester, NY
ncs@cs.rit.edu

Abstract
We describe the experience of three undergraduate computer science programs offering courses on parallel computing. In particular, we offer three different solutions to the problem of equipping a lab and discuss how those solutions may impact the content of the course.

1. Introduction

During the late 1980's and 90's the National Science Foundation sponsored several Undergraduate Faculty Enhancement (UFE) workshops that addressed how to teach parallel computing to undergraduates. These included 1989, 1991, 1992, 1997 and 1998 workshops at Colgate University; 1992-1996 workshops at Illinois State University; and 1995 workshops at the University of North Iowa and California State University, Fresno.

Most practitioners agree that programming a real parallel computer is an essential component to effectively teaching parallel computing concepts. While it is possible for students to gain insight into some parallel computing concepts through theory and the use of simulators, there are other concepts and practical issues that can only be appreciated through experimentation with truly parallel hardware. As many non-research focused universities cannot justify purchasing high-powered, state-of-the-art parallel machines, the UFE workshops tended to suggest low cost solutions, such as utilizing systems at supercomputing centers, utilizing clusters of workstations, or utilizing systems built using inexpensive components, such as Inmos Transputers, for the experiential portion of the course.

Two of us have utilized Transputer systems in the early versions of our parallel computing courses. However, as the 90's come to an end, these systems are no longer viable – the technology is no longer being developed, supported or available for sale. This has caused us to look for alternative hardware replacements and to reexamine the content of our courses.

In this paper, we present our solutions to the parallel hardware selection problem, and examine the effects that our hardware selections have had on the content of our parallel computing courses.

2. Our Parallel Courses

While our approaches to our courses are different there are common components and objectives. The objectives of our courses are to

(i) introduce students to the concepts, techniques, and architectures of parallel computing,

(ii) give students extensive practice designing and implementing parallel algorithms,

(iii) expose students to several different models of parallel computation, and

(iv) provide students with practical, hands-on experience of the benefits of parallel execution.

The common components include a history of parallel computing, parallel architecture, software and performance, parallel algorithms, and parallel languages.

2.1 Calvin College

Parallel computing is a relatively new course in the curriculum at Calvin College. In 1997, Adams attended Nevison and Schaller's UFE workshop. In January 1998, he drew upon that experience to implement the first *Parallel Computing* [1] course at Calvin College. The course is an upper-level elective whose prerequisites are courses in *Algorithms* and *Computer Architecture*.

Parallel computing concepts, techniques, and architectures are explored through the usual means of lectures and presentations. Students receive practice designing and implementing parallel algorithms through laboratory exercises and programming assignments, and are exposed to different parallel models by using different parallel software packages such as *Parallaxis* [2] and *MPI* [6].

Parallaxis is a simulator for programming in the data-parallel style appropriate to a Single Instruction stream, Multiple Data stream (SIMD) distributed memory, parallel architecture. *MPI*, the Message Passing Interface, is a standard library for developing programs for a Multiple Instruction Stream, Multiple Data stream (MIMD),

distributed memory parallel machine in C, C++ or FORTRAN. *MPI* can be used on networks of workstations and on most parallel computers available today. This means that *MPI* programs are highly portable.

At Calvin, students execute their parallel programs in *MPI* on a workstation cluster to experience firsthand the benefits of parallel execution.

2.2 Colgate University

The purpose of *Parallel Computing* [9] at Colgate is to provide an overview of the field of parallel computing and to develop some understanding of the design, implementation and analysis of parallel programs. The course description is quite similar to the Calvin College course. Two parallel programming platforms are explored in the laboratory, *Parallaxis* and *MPI*, both described in section 2.1. However, at Colgate *MPI* is executed on a dedicated parallel computer, a Parsytec PowerXplorer, described below.

Four laboratory exercises explore programming using *Parallaxis* as an example of programming a SIMD machine, while the *MPI* exercises include message timing, data parallel algorithms, processor farm algorithms, and irregular divide-and-conquer algorithms such as parallel branch-and-bound.

The Colgate course includes an overview of parallel architectures, including some of the history of the development of parallel computers and a discussion of parallel programming languages in addition to those used in the laboratory. Considerable time is spent studying techniques for developing parallel algorithms, analyzing the algorithms, implementing them as programs, and testing the performance of those programs.

2.3 Rochester Institute of Technology

Rochester Institute of Technology (RIT) offers a two course concentration in parallel computing, designed for students who have an interest in understanding the underlying principles and issues in parallel computing and in gaining expertise in the use of parallel systems. Both one quarter courses are offered to graduate and upper level undergraduate students.

The first course, *Parallel Computing I* [13], is similar to the courses described above and is designed to provide students with the flavor of the hardware and software issues in parallel computing. Topics include an introduction to the basic concepts, parallel architectures, parallel algorithms, parallel languages, network topology, coarse versus fine grained parallelism, applications, parallel programming design, and debugging. In addition to utilizing *Parallaxis* and *MPI*, students also experiment with languages designed for shared memory SIMD and MIMD computers, High Performance FORTRAN (*HPF*) [15] and *C-linda* [7], respectively. The prerequisite for this course is *Operating Systems*.

Parallel Computing II [10] is a study of the principal trends in parallel algorithm design, through the analysis of algorithms used in various areas of application. Specific techniques that have gained widespread acceptance are highlighted. The course investigates the interplay between architecture and algorithmic structure and discusses the effect that these issues have on the complexity and efficiency of parallel algorithms.

3. Our Parallel Hardware

Each of the authors has selected different hardware for their students to experiment on in their parallel computing courses. The sections that follow tell of the considerations that factored into their choices.

3.1 Calvin College

The high cost of a commercial multiprocessor makes it difficult for an undergraduate college like Calvin to acquire such a machine. This led to exploration of the available options for inexpensive parallel computing hardware.

Since 1994, NASA's *Beowulf* [2] and its off-spring (e.g., *Grendel* [11], *Loki* [17], et al.) have demonstrated how inexpensive, high-performance multiprocessors can be built from commodity, off-the-shelf components (i.e., PCs or workstations, ethernet, *Linux* [9], and *MPI* or *PVM* [6]). These multiprocessors have come to be known as *Beowulf clusters*, *workstation clusters*, or just *clusters*.

A dedicated cluster can achieve the performance of a supercomputer at a small fraction of a supercomputer's price. For example, *Loki* has achieved 1.2 GFLOPS (measured) and cost $67,000 to build in 1996; within a year the price to build it had dropped to $28,000. Building a cluster from new components typically costs from $20,000 to $200,000, depending on the kind and number of its CPUs, memory, disk, and communication infrastructure.

Converting an existing network of Unix/Linux workstations into a cluster is even less expensive, because it costs *nothing*. With free implementations of *MPI* (e.g., *mpich* [6]) and *PVM* available, all one must do is (i) download the software, (ii) install it on the servers/workstations in the network, and (iii) begin parallel processing!

This is the approach we have taken at Calvin College. Our "parallel hardware" consists of the *Unix Classroom* [2], a networked laboratory of 24 Sun workstations and a Sun Ultra-2 Enterprise Server. As the department's primary computing facility, the Unix Classroom is shared by *Parallel Computing* and other computer science courses, which can negatively impact parallel performance (see below).

With Linux, a laboratory of networked PCs can also be converted into a cluster. Ideally, the network should be 100 Mbps ethernet or faster to reduce communication latency, the bottleneck in many parallel computations.

As schools upgrade their PCs, another alternative is to build a dedicated cluster from cast-off PCs. A Calvin student did this for his senior project during the 1998-99 academic year, building a 3-dimensional hypercube from cast-off 486s. He upgraded the RAM on each machine to 16M, installed *Linux* and *MPI*, and had his own dedicated

cluster for parallel computing. The total cost (for fast ethernet cards and a switch) was roughly $6000.

The *MPI* vs. *PVM* debate can take on religious overtones. *MPI* was chosen because unlike *PVM*, *MPI* only consumes a workstation's resources when that workstation is involved in a parallel computation; this is especially important in a shared laboratory. *MPI* was configured to support parallel programming in Fortran-77, C, and C++, all using the GNU *gcc* compiler.

3.2 Colgate University

Colgate is fortunate to have a dedicated parallel computer, a 32 node Parsytec PowerXplorer. (This equipment, which cost about $125,000, was partially supported by an NSF ILI grant, DUE-9551105.) Each node in this system has a PowerPC-601 for computation and a transputer T805 for communications, with 8 MB of memory per node. The nodes are connected in a 4 x 8 mesh network.

The PowerXplorer runs both *PVM* and *MPI* message passing systems as well as its own proprietary message passing. Colgate University used *PVM* in the past and currently uses *MPI* for teaching purposes. Up to four users can access partitions of the machine at the same time, using 8, 16, 24, or 32 processors in a partition. The combination of multiple users and up to 32 processors to dedicate to one problem made this an excellent choice for our purposes.

The PowerXplorer has a SUN workstation host, which is tied into the Colgate laboratory network. Students can access our parallel system from any other computer in the lab, or, indeed, from any machine on the internet. In fact, this course has been offered to Hamilton College students simultaneously with Colgate University students using remote conferencing facilities at the two colleges, and the Hamilton College students have used the Colgate equipment over the internet. Students and faculty from Washington and Lee University have also used the Colgate equipment for research projects.

Because of the dedicated hardware, students are able to do routing, timing, and speedup studies without the problems inherent in a general purpose network.

3.3 Rochester Institute of Technology

The environment at RIT is a bit different than that at Calvin or Colgate. Not only is it a technical university rather than a liberal arts college, but the number of students in the computer science major is a factor of at least ten more.

The environment of the RIT computer science laboratories consists of clusters of networked, Unix workstations connected to servers that provide a distributed file system and printing support. Because its undergraduate computer science majors number around 450, it is important that its parallel computing hardware be accessible from this environment.

The department's first parallel computing equipment, an 88 node Transputer system, had required a sizable investment (more than $200,000) and were, for all practical purposes, used only for the courses in the parallel computing concentration. Given the number of students taking this concentration and the effective demise of NSF equipment grants meant that such a large amount of money was not likely to be available for such a specialized investment for replacement equipment.

Many options for replacement were explored, including using clusters of workstations but experimentation with PVM had indicated that parallel computing students would likely have a negative impact on the performance of the computer science computing environment if that mechanism were chosen. This impact could be avoided if the cluster were isolated, but then the parallel equipment would not be accessible from the general environment. Ultimately, it became desirable to purchase equipment that could be used both as a parallel processor and as a more general purpose computer.

Another consideration was to limit the maintenance impact for the department. The department's general purpose computing equipment consists mostly of Sun Microsystems workstations with some from Silicon Graphics, Inc. Therefore, options from both of these companies were explored: the SGI Origin 2000 and the Sun 450 fileserver. While the architecture of the Origin was enticing, negotiations provided quotes for more processors for substantially less money from Sun.

Ultimately, the department purchased two four node, symmetric multi-processor (SMP) Sun 450 fileservers for less than $70,000. (This price was due in part to being part of a much larger purchase.) Sun suggests that these servers be connected using Scalable Coherent Interface (SCI) interconnects for high-speed communications within the cluster, but due to price considerations, the two 450's are instead connected to each other using a dedicated ethernet connection. Sun's HPC (High Performance Computing) package offers a variety of software to explore, including *HPF*, *MPI*, *PVM*, and *LSF* (Load Sharing Facility).

4. How Equipment Affects Content

Choice of hardware can significantly affect the content of a course in parallel computing. For example, an instructor might tend to assign only projects and exercises that map naturally to their hardware configuration, or might tend to ignore issues that are only important on other hardware platforms. The following sections present the authors' reflections on this problem.

4.1 Calvin College

At Calvin College, we try not to let our use of a cluster affect what *concepts* we cover in our course. We like to think that we would teach the same concepts, regardless of the hardware that we use, because our course is intended to provide a general introduction to parallel computing.

However, it is our opinion that much (most?) of a student's learning in a course occurs *outside* of the lecture format, as the student works to solve problems. We also believe that our parallel hardware shapes the kinds of problems we give students to solve, as well as the forms of our students' parallel solutions (as do the programming languages in which they write those solutions). For example, because

ethernet is a broadcast medium, the educational value of message-timing exercises using a cluster is rather limited compared to a dedicated parallel machine. So our choice of a cluster directly affects what our students *learn*, because it colors their experience of parallel behavior. Our problem is to minimize this constriction of what our students learn.

One way we seek to solve this problem is by using *both Parallaxis* and *MPI*, because exclusive use of either results in an incomplete course. That is, *Parallaxis* lets students experiment with different connection topologies, and mapping parallel algorithms onto specific topologies, but they do not experience the speedup of parallelism. The opposite occurs with *MPI* on our cluster: students experience first-hand the speed-up of parallel execution; but its bus topology restricts their problem solutions. *Parallaxis* and *MPI* thus complement one another nicely.

We also try to solve the problem by seeking cost-effective ways to diversify our students' parallel experience. For example, recent hardware price drops let us upgrade our lab's server from a uniprocessor to a symmetric multiprocessor (SMP). Our students now receive hands-on experience in SMP multithreading, beyond their experience using our cluster.

The primary drawback to building a cluster from an existing laboratory of networked workstations is communication latency. If other users are in the lab, then a parallel computation competes with those users for network bandwidth, negatively impacting the parallel performance of our computation. To avoid this problem, our parallel computing students receive 24-hour access to the lab and are encouraged to measure a program's performance when the lab is empty.

This workaround is far from perfect, so we have applied to the NSF's Major Research Instrumentation (MRI) program for funding to build a cluster dedicated for parallel computation. If our proposal is successful, this cluster will be the subject of a future report.

Our hardware choices cannot help but affect what students take away from a course. In an introductory parallel computing course, our aim is to minimize the negative effects of those choices.

4.2 Colgate University

As at Calvin, we strive to teach the fundamental concepts of parallel computing independent of the hardware used. In order to do this it is important to expose students to different approaches to implementing parallel programs. We do this by using *Parallaxis* for simulated SIMD computing and *MPI* on a dedicated parallel message-passing architecture. We hope to add the opportunity to do computing on a shared memory multi-processor in the near future.

Using *Parallaxis* students learn about some of the fundamental ideas of SIMD computing, but without a chance to experience or experiment with the speedup of a true parallel machine. Using *MPI* on our PowerXplorer system, students not only can see true parallel speedups, but they can also experiment with some features such as routing, message-timing, and different mappings of a parallel algorithm to an architecture. For example in one lab, we measure the times for messages going around a ring first as mapped automatically to the machine, not a physical ring, and second mapped to a physical ring embedded in the mesh of the machine. In this lab students see the importance of the underlying machine topology and are also able to do accurate measures of message latency. Other labs include understanding of the mapping of the algorithm to the physical topology of the machine and speedup measurements on a dedicated machine. These labs would not be possible without a dedicated parallel machine. The combination provides an appropriate breadth of experience for an introductory course.

Because we use these two platforms and not, currently, a shared memory system, the content of our course focuses more on the message-passing paradigm and deals less with shared-memory algorithms and the associated control structures.

4.3 Rochester Institute of Technology

At RIT, we try as well to minimize the effect that the available hardware has on the concepts taught in the course. But, the purchase of our new equipment did affect the experiments our students were able to try.

On our Transputer system, students programmed in *Parallaxis* (SIMD, distributed memory), *C-linda* (MIMD, shared memory in simulator mode), and *Occam* [4] (MIMD, distributed memory); the use of FORTRAN (SIMD, shared memory) as a parallel language was presented in lecture. With the new equipment, we use *Parallaxis* and *MPI* for the same reasons discussed above. We also use *C-linda* in non-simulator mode and *HPF* to provide experience with shared memory paradigms. In both cases, the students had an opportunity to experience the programming paradigms used on most available parallel hardware. With the new machine, we have also encouraged students to experiment with multi-threading, *PVM*, *LSF*, and with a variety of Java parallel computing variants [15].

We have found that our (new) dual SMP's provide some unexpected challenges and real-world experiences for our students. For example, because our machine is accessible to everyone, we experience similar difficulties as at Calvin in timing the effects adding more processors to the solution of the problem. This has led to experimenting with batch queues both during day and night time. Also, our choice of interconnection network between the SMPs means that often when a fifth processor is added to the execution of a problem that it will often take more time to execute rather than less. Furthermore, finding the appropriate compiler options, and system and environment variables to optimize a particular solution to a problem on this system is a challenge. An effect of this is that both the teacher and students alike are encouraged to explore and share their results in the classroom and through e-mail: what worked, what did not, and an analysis of why. This has added another dynamic to the course. Challenges such as these are ones that students are likely to encounter as programmers once they leave school. Learning to overcome them and to

select an appropriate solution to a particular problem is an invaluable lesson.

5. Future Directions

RIT bases much of its parallel course on the use of two SMP machines; Calvin College has started to use multi-threading on an SMP as well as MPI on a workstation cluster; and Colgate has recently installed a 4 processor SMP which we plan to work into the parallel course in the future. With four (and soon eight) processor SMP machines becoming reasonably priced, many smaller schools will be able to afford them as means to introduce students to true parallel processing. Thus we see a combination of work with message-passing on dedicated machines or on networks of workstations combined with the use of small SMP machines as a likely combination for laboratories for parallel computing. (One caution however, the use of threads in C/C++ or Java on such machines currently involves the use of low level primitives for synchronization and control. Higher level constructs are needed and under development, such as versions of CSP for threads in Java [13].) The scientific and industrial use of SIMD architectures seems to be dwindling, so this will likely become a less important component of courses on parallel computing in the future.

6. Conclusions

There are many paths to select from to provide undergraduate students with real parallel computing experience. We offer three alternatives that have worked for us along with a flavor of our courses in hopes that others may benefit from our experience. Each alternative includes a different approach to equipping a laboratory for parallel computing and the ramifications that the equipment has for the content of the course.

References

[1] Adams, J., *Introduction to Parallel Computing*, Last update May 1999, http://cs.calvin.edu/CS/parallel/.

[2] Adams, J., "The Design and Implementation of a UNIX Classroom", *Twenty-fourth SIGCSE Technical Symposium on Computer Science Education*, Indianapolis, Indiana, March, 1992.

[3] Becker, D., Sterling, T., Savarese, D., Dorband, J., Ranawak, U., Packer, C., "Beowulf: A Parallel Workstation for Scientific Computation," *Proceedings, International Conference on Parallel Processing*, 1995. See also http://www.beowulf.org/.

[4] Bowen, J., *The Occam archive*, Last update May 1994, http://www.comlab.ox.ac.uk/archive/occam.html.

[5] Bräunl, T., *Parallaxis-III - A Structured Data-Parallel Programming Language*, Last update January 1995, http://www.ee.uwa.edu.au/~braunl/parallaxis/.

[6] Geist, Al, *PVM Parallel Virtual Machine*, Last update August 1999, http://www.epm.ornl.gov/pvm/.

[7] Gelernter, D., Carriero, N., *Linda Group (Yale University)*, http://www.cs.yale.edu/Linda/linda.html.

[8] Grop, B. and Lusk, R., *The Message Passing Interface (MPI) Standard*, Last update February 1999, http://www-unix.mcs.anl.gov/mpi/.

[9] Hekman, J.and Oram A. (eds.), *Linux In A Nutshell : A Desktop Quick Reference* (Nutshell Handbook), O'Reilly & Associates, 1997. See also http://www.linux.org/.

[10] Kitchen, A., *Parallel Computing II*, Last update May 1999, http://www.cs.rit.edu/~atk/736/PALG983.html.

[11] Ligon, W., *Grendel: The Clemson Beowulf Workstation*, Last update December 1997, http://ece.clemson.edu/parl/grendel.htm.

[12] Nevison, C., *Parallel Computing*, Last updated September 1999, http://149.43.80.141/CSLabWebPages/CS445_Web_pages/.

[13] Nevison, C., Seminar: Safe Concurrent Programming in Java with CSP, *Proceedings of 13th SIGCSE Symposium* (March, 1999), ACM Press, 367.

[14] Schaller, N., *Parallel Computing I*, Last update May 1999, http://www.cs.rit.edu/~ncs/Courses/531.shtml.

[15] Schaller, N., *Java for Parallel/High Performance Computing*, Last update June 1999, http://www.cs.rit.edu/~ncs/parallel.html#jav

[16] Tevis, P., *High Performance Fortran*, Last update June 1999, http://dacnet.rice.edu/Depts/CRPC/HPFF/.

[17] Warren, M., "Loki - Commodity Parallel Processing," Last update April 1998, http://loki-www.lanl.gov/.