Local Supercomputing Training in the Computational Sciences Using National Centers

Floyd B. Hanson, University of Illinois at Chicago

Abstract

Local training for high performance computing using remote national supercomputing centers is different from training at the centers themselves. The local site computing and communication resources are a fraction of those at the national centers. However, training at the local site has the potential of training more computational science and engineering students in high performance computing by including those who are unable to travel to the national center for training. The experience gained from supercomputing courses and workshops in the last seventeen years at the University of Illinois at Chicago is described. These courses serve as the kernel in the program for training computational science and engineering students. Many training techniques, such as the essential local user's guides and starter problems, that would be portable to other local sites are illustrated. Training techniques are continually evolving to keep up with rapid changes in supercomputing. An essential feature of this program is the use of real supercomputer time on several supercomputer platforms.

Keywords: *High performance computer training, Supercomputing education, Computational sciences.*

1 Introduction

High performance computing education is important for keeping up with the rapid changes in computing environments. The training of computational science and engineering students in the most advanced supercomputers makes it less likely that their training will become quickly obsolete as with conventional computer training. Rapid changes in supercomputing causes concern for some, yet supercomputing remains with us although changed in character, provided we accept the notion that the term supercomputing refers to the most powerful computing environments at the current time. Preparation in high performance computation is preparation for the future of computation.

The problem is that most universities and colleges do not have on-site supercomputer facilities and only a small fraction of the infrastructure. The computational environment for remote access at these institutions to the national or state supercomputer centers may not be up to the quality of the computing environment for access at the centers. In additional, while the the remote centers may have excellent training facilities, many students lack the mobility, both in finances and time, to travel to the remote center for training. Thus, local supercomputing training is important for making the educational benefits and computational power of supercomputers available to large numbers of students with only electronic access to remote national and state centers. This paper is designed to share practical supercomputer training advice gained from a long experience to other

local instructors in hope of reducing the burden of local training. The goal is to give the local students local supercomputing training comparable to the resource rich national centers using limited resources.

The University of Illinois at Chicago has pioneered local use of remote national supercomputer centers to train its computer science, applied science and engineering graduate students in the use of available highest performing computers to solve the large problems of computational science. The training has been at two levels, *Introduction to Supercomputing* [6] and *Workshop Program on Scientific Supercomputing* [5]. Starting in the 1985 academic year, the students of the introductory course have done group projects on massively parallel processors and vectorizing supercomputers.

For the 1987 academic year, N. Sabelli with the author conceived of the *UIC Workshop Program on Scientific Supercomputing* [5], the second level. Recently, due to program down-sizing at the University of Illinois, the workshop has been scaled down from full-time to multi-hour workshop course and is currently merged with the introductory course, maintaining a good part of the kernel of the original workshop. The workshop differed from the introductory course in that the graduate students had to bring a thesis or other computational science and engineering research problem along with code needing optimization, topics include a far wider range of computer topics than the course, and outside experts are brought in to lecture. The workshop course served as the advanced course to follow the introduction to supercomputing course. We have made significant changes for both introductory and workshop courses in course structure and contents since our earlier reports [6, 5], along with corresponding significant advances in supercomputing.

An objective is to train students in the use of supercomputers and other high performance computers, in order to keep their computer training at the leading edge. Other objectives are to give students background for solving large computational science research problems, and developing new parallel algorithms. The long range goal is to make the students better prepared for future advances in high performance computing.

A very early version of this paper was presented at the February 1994 *DOE High Performance Computing Education Conference* in Albuquerque on the panel *Laboratory and Center Educational Training Efforts.* The following sections describe the current *introductory supercomputing course*, the *supercomputing workshop* and related topics for the purpose of communicating the Chicago experience to others who are implementing a similar local supercomputing training courses.

2 Introductory Supercomputing Course Description

The introductory supercomputer course covers both theoretical developments and practical applications. For practical applications, real access to supercomputers and other advanced computers is obviously essential. Almost all all theoretical algorithms have to be modified for implementation on advanced architecture computers to gain optimal performance. Loops or statements may have to be reordered and data structures altered, so that data dependencies are minimized and load balancing of processors is maximized. Also, the selection of the best algorithm usually depends on the critical properties of the application and of the hardware.

Access to several advanced machine architectures greatly enhances the learning experience, by providing a comparison of architectures and performance. One of the best way to understand one supercomputer is to learn about another platform. Recent offerings have used Cray Y-MP, C90, T3D, T90 and T3E, as well as Connection Machine CM-5. The average enrollment has been about 16 students according the final grade count.

MCS 572 Introduction to Supercomputing is a one semester course that is intended to entry-

level give graduate students of the University of Illinois at Chicago a broad background in advanced computing, and to prepare them for the diversity of computing environments that now exist.

All offerings of this course, from the 1986 academic year to the present, were based on many journal articles, books, and the our own research experience in advanced computing. The students were mainly evaluated on their performance on theoretical homework, individual computer assignments, and major group project reports, as well as their presentation to the class. Over the years, the students completed advanced group computer projects on the Argonne National Laboratory Encore MULTIMAX, Alliant FX/8 and IBM SP2 parallel computers, on the NCSA Cray X-MP/48, Cray Y-MP4/64, Cray 2S, Connection Machine CM-2, and Connection Machine CM-5, on the PSC Cray C90 and Cray T3D, and on the SDSC Cray T90 and T3E.

Perhaps, the best way to describe the course contents is to list a recent course semester syllabus:

Introduction to Supercomputing Course Syllabus (Abbreviated)

Catalog description: Introduction to supercomputing on vector, parallel and massively parallel processors; architectural comparisons, parallel algorithms, vectorization techniques, parallelization techniques, actual implementation on real machines (Cray super vector and massively parallel processors).

Prerequisites: MCS 471 Numerical Analysis or MCS 571 Numerical Methods for Partial Differential Equations or consent of the instructor. Graduate standing.

Semester Credit hours: 4

List of Topics	**Hours.**
• Introduction to advanced scientific computing.	3 hours.
• Comparison of serial, parallel and vector architectures.	3 hours.
• Performance measures and models of performance.	3 hours.
• Pure parallel algorithms and data dependencies.	3 hours.
• Optimal code design.	3 hours.
• Loop optimization by reformulation.	6 hours.
• Code implementation on vectorizing supercomputers (eg, Cray T90).	5 hours.
• Code implementation on massively parallel processor (eg, T3E).	4 hours.
• Parallel programming interfaces (eg, MPI, PVM).	6 hours.
• Code implementation on hybrid and distributed parallel processors.	3 hours.
• Block decomposition and iteration methods.	6 hours.
• Total.	45 hours.

Required Texts:

- F. B. Hanson, A Real Introduction to Supercomputing, in "Proc. Supercomputing '90", pp. 376-385, Nov. 1990.
- F. B. Hanson, "MCS572 UIC Cray User's Local Guide to NPACI-SDSC Cray T90 Vector Multiprocessor and T3E Massively Parallel Processor, v. 14, http://www.math.uic.edu-/∼hanson/crayguide.html, Fall 2000.
- J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van der Vorst, Numerical Linear Algebra for High-Performance Computers, SIAM, 1998.

The following discussion will selectively focus on some of the topics covered in this course.

2.1 Texts

One difficulty in teaching *MCS572 Introduction to Supercomputing* is the choice of a text or texts, especially when real supercomputers are used. There are a *super* number of books available on high performance computing, parallel processing, and related issues. However, most of these references are either over-specialized, too theoretical, or out of date. The rapid changes in supercomputing technology cause many of the supercomputing references to quickly become out-of-date, especially if there is a strong emphasis on a small set of real machines. Although no single text was used in this course, if we had to choose one text for the *Introduction to Supercomputing* course, some possible choices that cover a broad range of supercomputing topics are Dongarra et al. [1], Levesque and Williamson [10], Golub and Ortega [3], Hwang [9], Ortega [11], and Quinn [13]. However, many of the other references have been used for particular topics. World Wide Web (WWW) links to many on-line web hypertexts are given on the class home page at the web address *http://www.math.uic.edu/~hanson/sylm572.html* and a much more extensive list of supplementary references is given at *http://www.math.uic.edu/~hanson/superrefs.html*.

2.2 Architecture

Our introductory course starts out with the basic architectural elements of serial, vector, shared and distributed memory parallel, and vector multiprocessor machines [9]. We believe it is important students have a good mental image of what is happening to data and instruction flow between memory, processing units and registers. Otherwise, students will have difficulty in understanding parallel and vector optimization techniques later in the course. We supplement the Flynn's simple computer classification (SISD, SIMD and MIMD) [9] by how real complications modify the classification, such as vector registers, vector operations, pipelining, bus communication networks, shared memory and distributed memory. Also, an early explanation that asymptotic pipeline speed-up is the number of pipeline stages [9] provides motivation for less than ideal vector speed-up found in practice.

2.3 Performance Models

The discussion of performance models is also helpful, because they give simply understood characterizations of the power of supercomputers. The simplest model is the classical Amdahl Law for a parallel processor,

$$T_p = [1 - \alpha + \alpha/p] \cdot T_1, \tag{1}$$

where the execution or CPU time on p parallel processors depends on the parallel fraction α and is proportional to the time on one processor T_1, which should be the time for the best serial algorithm. This model assumes that the parallel work can be ideally divided over the p parallel processors and leads to Amdahl's law for the saturation of the speed-up $S_p = T_1/T_p$ at the level $1/(1-\alpha)$ in the massively parallel processor limit $p \longrightarrow \infty$, i.e., that parallelization was limited. An analogous law holds for vectorization. The deficiency with this law is that it is too simplistic and is no match for either the complexity of supercomputers or the size of applications that are implemented on current supercomputers. Indeed, one principal flaw in Amdahl's law is that the parallel fraction is not constant, but also depends on the problem size, $\alpha = \alpha(N)$, and as computers become more super, computational users are apt to solve larger problems than they had been able to solve before.

A modification [7] of Amdahl's law for problem size, where the major parallel work is in loops of nest depth m with N iterations in each loop of the nest, leads to the formula

$$T_p = \tau \cdot [K_0 + K_m \cdot N^m/p], \tag{2}$$

where τ is some time scale, K_0 is a measure of scalar or non-loop work and K_m is a measure of loop nest work. Comparison of (2) and Amdahl's model (1) leads to the nearly ideal parallel fraction

$$\alpha(N) = 1 - \frac{K_0}{K_0 + K_m N^m} \longrightarrow 1, \tag{3}$$

in the large problem limit $N \longrightarrow \infty$, a limit not represented in the original Amdahl's law. Efficient use of supercomputers requires large problems.

Another useful modification of Amdahl's law is for linear parallel overhead

$$T_p = [1 - \alpha + \alpha/p] \cdot T_1 + \tau \cdot (p - 1), \tag{4}$$

developed as a model of the 20-processor Encore Multimax parallel computer performance to convince students that they needed to run larger problems to get the benefit of parallel processing. A Unix fork command was used to generate parallel processes and a performance evaluation measurement indicated that the cost was linear for each new process. The speedup for this model has a maximum at $p^* = \sqrt{\alpha T_1/\tau}$, so that as $p \to +\infty$, the speedup decays to zero. Hence, if the student's work load, T_1, is sufficiently small, a *slow-down* occurs for a sufficiently large number of processors, and the student sees no benefit in parallel processing, but the model demonstrates that *Supercomputers Need Super Problems.* The model and the story behind it always works for new students and prepares them to think beyond the toy problem homework assignments of regular classes.

Hockney's [8] asymptotic performance measures are another procedure for avoiding Amdahlian size dependence insensitivity and are used in the Top 500 Supercomputer Sites {http://www.top500.org/}. The question of size dependence is also related to *scaled speed-up* ideas used by Gustafson and co-workers [4] in the first Bell award paper. Scaled speed-up is based on the idea that is the inverse of Amdahl's, in that users do not keep their problem sizes constant (i.e., keep $T_1(N)$ fixed), but increase their problem size N to keep their turn around time, i.e., T_p, constant as computer power increases. Also, scaled speed-up implies that speed-up may be computed by replacing $T_1(N)$ by r times the time on a N/r fraction of the problem, since a single Massively Parallel Processor CPU cannot compute the whole problem.

2.4 Supercomputer Precision

Another feature that may have been overlooked in some supercomputing courses is numerical precision, but is not as important as it was in the past due to a near uniform adoption of IEEE floating point standards [12] (See also http://www.math.uic.edu/ hanson/mcs471/FloatingPointRep.html). At a more basic level is the difference in numerical representation was found in bit-oriented arithmetic such as on Cray or IEEE systems and byte-oriented arithmetic of IBM main-frames or similar systems. This lead to bad judgment and confusion for beginners, especially for such things as stopping criteria or the comparison of results and timings from different machines. The IBM 32-bit Fortran77 byte-oriented single-precision arithmetic uses 3 bytes (24 bits) for the decimal fraction and 1 byte for the exponent and sign in hexadecimal (base 16) representation. However, IBM 64-bit, byte-oriented, double precision arithmetic uses 7 bytes for the fraction and the same 1 byte for the exponent and signs. Byte-oriented double precision is more than double.

Bit-oriented arithmetic comes in several flavors. On its vector supercomputers, Cray uses 64 bits for its single precision, with 48 bits for the fraction and 16 bits for the exponent and sign, but uses 128 bits for its double precision with 96 bits for the fraction and 32 bits for the exponent and sign, i.e., authentic double precision. IEEE Precision is also bit-oriented and is used on many

recent machines including Cray massively parallel processors, but single and double precision are roughly half of the corresponding precision on Cray vector machines. The IEEE Precision [12] comes with several new concepts such as *Not a Number (NaN)*, *rounding to even*, *INFINITY* and *gradual underflow to zero* that may confuse new users. Most vendors have pledged to adopt the IEEE Precision standard.

These differences show up in the truncation errors. For internal arithmetic, the default truncation, contrary to popular opinion, has usually been *chopping or rounding down*, unless a different type of rounding is requested, such as in output. However, the IEEE Precision standard uses *rounding to even*. The features of these precision types are summarized in Table 1. The last

Table 1: SuperComputer Precision:

Floating Point Precision Type	Precision base b	Precision digits p	Machine Epsilon b^{1-p}	Equivalent Decimal Precision
IBM Single	16	6	0.95e-06	07.02
IBM Double	16	14	0.22e-15	16.65
IBM Quad	16	28	0.31e-32	33.51
Cray Single	2	48	0.71e-14	14.45
Cray Double	2	96	0.25e-28	29.59
CM & IEEE Single	2	24	0.12e-06	07.92
CM & IEEE Double	2	53	0.22e-15	16.65
Sun Sparc	2	53	0.22e-15	16.65
VAX D Float	2	56	0.28e-16	17.56

column gives the equivalent decimal precision which corresponds to the effective number of decimal digits (p_{10}) if the decimal machine epsilon ($10^{1-p_{10}}$) were equal to the machine epsilon (b^{1-d}) in the internal machine base (b). From this table, the different precision types can be summarized as

$$ibm\text{-}sgl < ieee\text{-}sgl \ll cray\text{-}sgl < ibm\text{-}dbl = ieee\text{-}dbl \ll cray\text{-}dbl < ibm\text{-}quad$$

2.5 Data Dependencies

The topic of data dependencies is of the utmost importance for parallel and vector optimization of code. Wolfe's [14] monograph gives a current description of dependencies, and a multitude of other optimization techniques. A useful observation about reducing data dependencies and anti-dependencies, in order to optimize code performance, is that the dependencies or anti-dependencies that inhibit vectorization are usually the same ones that inhibit parallelization. Thus a good instructional technique is to treat vectorization as a primitive kind of parallelization, avoiding most of the artificial historical distinctions.

2.6 Fortran Extensions

Many Fortran compilers for advanced computers, such as the Crays and the Connection Machine accept Fortran 90 array notation extensions. These extensions not only simplify the supercomputer programmer's coding tasks but, more importantly, facilitate the automatic optimizing compiler

recognition of optimizable array constructs. Making sure that each statement in a potentially vectorizable loop is a vector or array statement in the loop's index will maximize the performance in a powerful vectorizing compiler such as those on a Cray. Use of array notation and collecting like-size statements together is an effective optimization technique.

In the CM-5 Connection Machine multi-faceted, massively data parallel processor, only Fortran 90 statements (especially array statements) are computed on the CM-5 processing nodes; otherwise statements are computed on the single processor, control nodes.

While the principal programming language of most supercomputers was once Fortran, the C language is now more prevalent especially for computer science majors, although engineering majors not in computer science still prefer Fortran. During the fall of 1995, the number of *C* language projects became comparable to the number of Fortran projects. We have been making an effort to have a bilingual approach in our lectures in the past several years.

In addition, a short crash course in the Unix operating system may be necessary if the class is not Unix conversant, although most engineering majors are likely to be knowledgeable about Unix.

2.7 Memory Management

There are many other topics that depend on what other flavor the supercomputer course will take. Related to principle of memory reference locality [9] is the *principle of locality of reference for automatic code compiler optimizations.* The optimizing compiler will likely optimize only a relatively small segment of code, and the compiler will not be too aware of values initialized very far from the target segment. We find the special case of cache-based effects no longer seems very relevant since the disappearance of cache-based parallel machines like the Alliant.

If Cray vector computers are used, the chaining of pipelines together can be discussed as well as the overlapping of pipelines associated with different operations or multiple pipelines [9]. Also, the notion of division of memory into banks or other segments will be helpful in explaining memory conflicts [9] and extended dimension techniques. The discussion of gather and scatter in hardware is useful for explaining how the Cray Y-MP can efficiently handle data movement from and to array with subscripts of subscripts for arguments.

On the Connection Machines memory layout also corresponds to processor layout, so that performance will depend heavily upon how arrays are mapped to the processors which hold the distributed memory. On the CM-5, the local memory is actually held by the Vector Units attached to the local processing nodes and thus the Vector Units play a major role in memory management.

2.8 Message Passing Communication

A major change in the Fall 1995 semester was that we included message passing architectures and programming into the introductory course. The current prevalence of distributed memory processors meant we no longer could avoid message passing in supercomputing. We obtained access to the PSC Cray T3D and the CTC IBM SP2 massively parallel processors. We used PVM (Parallel Virtual Machine) for the T3D, since that was the best supported parallel message passing language on that machine at the time. Implementation was difficult and time consuming due to lack of adequate tutorial documentation and working, up to date examples for classes. Current massively parallel processing projects on the Cray T3E have used MPI (Message Passing Interface). We have learned to expect some failed plans when teaching an experimental supercomputing course. It should be noted that our supercomputing center proposals resulted in more machine power than we had anticipated: three 512 node massively parallel processors (T3D, CM-5 and SP2) in addition to

the 16 vector processors on the C90. Currently, we have our own adequate documentation examples online to make message passing work for class projects.

2.9 Loop Optimizations

There are many benefits in reformulating programming loops (cf., [10] for examples and original citations). One loop optimization technique is reordering nested loops for linear algebra constructs in a column-oriented Fortran environment. Another beneficial technique is the unrolling of loops in Fortran for pipelined machines where some data can be retained in vector registers. Block decomposition or strip-mining of loops can be helpful, but in many situations it can actually be less efficient due to greater efficiency in automatic optimization or to interference from memory conflicts. Benefits depend on the character and size of both problem and hardware and need to be tested for each situation.

For particular machines, knowledge of the compiler loop optimization model is important, because most of the potential optimizations are found in iterated DO-loops. Compilers will also write the contents of subroutines at the place of the subroutine call to save on subroutine overhead using a procedure called *in-lining*. Since parallelization or multi-tasking on the Cray vector multiprocessors tends to be costly, loop optimization on these machines is primarily the vectorization of the most inner loop, where most of the loop nest work should be concentrated. On the Cray and Connection Machine, compiler directives can, under appropriate circumstances, force optimization where automatic optimization does not work. Loop optimization became less important as smarter optimizing compilers did more of the optimization automatically.

One systemic problem in teaching optimization was that most students were trained to write modular code as in C and C^{++}, hindering optimization and a good amount effort was needed to make them understand the extra overhead in modular programming with automatic optimization. of effort

2.10 Code Tuning by the Compiler Model Method

Automatic optimizing compilers work on a principle of locality, in that the current program step depends on neighboring steps or references. Understanding the model under which the machine compiler optimizes code can help the user to enhance the performance by restructuring the code so that the optimizable code is transparent to the compiler. Tuning code to the machine optimization model might be called the *Compiler Model Method* and result in extra speed-ups over untuned code.

In the introductory supercomputing class, we use a starter problem consisting of about a dozen poorly optimizable (e.g., "dusty deck") loops is assigned on the current Cray supercomputer. The purpose of the problem is for the students to develop optimization skills by integrating class techniques, before they tackle larger group projects. Thus, concentrating the beginner's learning on a smaller, concrete problem and wasting less time on the more significant assignment. The grade on this problem is roughly inversely proportional to the CPU time the tuned, optimized loops take, provided that the final storage of the original, untuned, automatically optimized code remains unchanged and that no work is removed out of the main timing loop.

The results for the best user CPU timings on a single processor of the current Cray supercomputer over almost a decade and a half are given in Figure 1.

Results have varied considerably, but hand optimization still achieves several thousand times speed-up on top of automatic optimization of the untuned code, beyond the expected benefit from automatic parallelization or vectorization. However, there has been a significant change from about 3000-8000 times improvement on the Cray X-MP to about 2000 on the Cray Y-MP and successors,

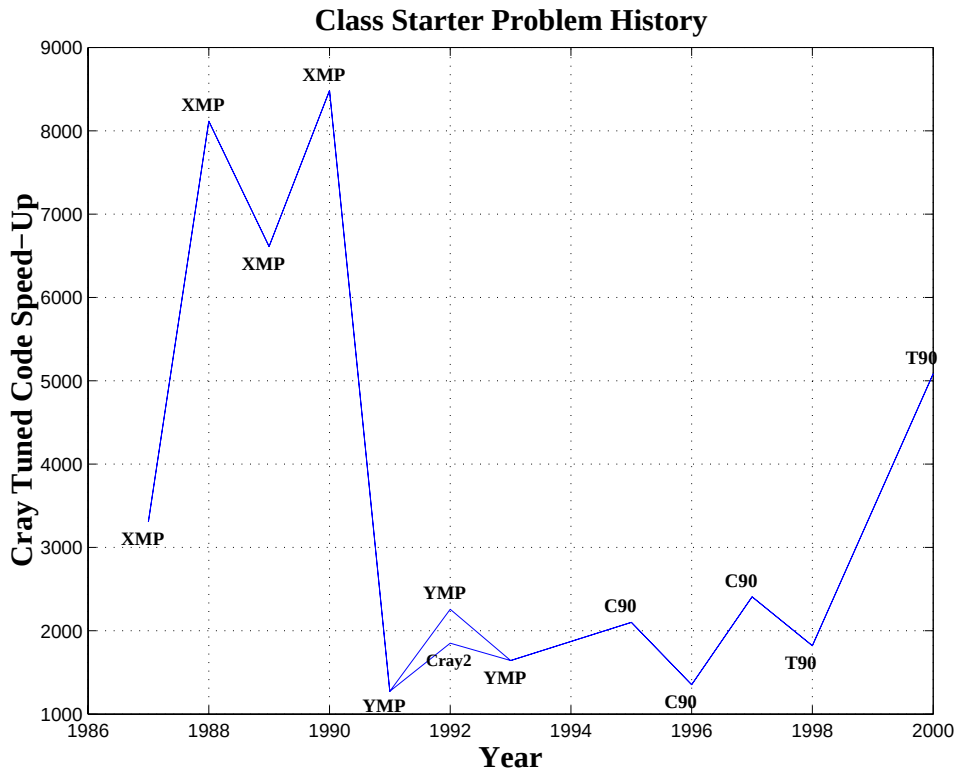

Figure 1: History of the best speed ups for the class Cray starter problem, with the nominal Cray vector model indicated.

the C90 and T90, likely due to improvement in automatic compiling that came with the Y-MP. The last jump in speed-up for the T90, was due to a computer science student who was extremely capable of unraveling recurrences. The full starter problem code is too long to present here in its entirety, but almost all the work load is in the loop nest ($n = 200$) given below:

```
      do  60  i=1,n
          do  60  j=1,n
              do  60  k=1,n
                  d1(i,j) = d1(i,j) + d2(i,j) + d3(i,j)
                  do  60  L = 1, 5
60                    d3(i,L) = d2(i,L)*d1(i,L)
```

The best times were obtained by students who were able to solve the complex recurrence in this loop nest, and these times were essentially reduced to the noise of the timer. Bear in mind that this is very extreme example, but means that significant gains can be made by hand tuning code on top of automatic optimization, just as new parallel algorithms may lead to advances comparable to advances in hardware.

2.11 Group Projects

We have observed that group projects appeared to reduce the amount of computer resources needed by the students and improves shared learning. We require group projects to be big enough to be

a realistic test of the target supercomputing systems, but small enough so as not to interfere with the professional research conducted on the systems.

In general, these group projects have been very successful. The majority of the projects have involved testing for optimal performance for numerical linear algebra algorithms, computational statistics, scientific visualization and scientific applications.

A list of the group student projects follows, with short descriptions, for this one semester and Connection Machine projects only. **MCS 572 Connection Machine Group Projects, Fall 1993**

1. C. Barnes and J. Goldman, *A Parallel Implementation of Progressive Radiosity on the CM-5* (This computer graphics project, concerned the computation of graphical light interaction using the radiosity method modified for progressive refinement, also comparing CM-5 performance with and without Vector Units to Y-MP performance).
2. S. Ginjupalli, *Solving a Set of Linear Equations using Gaussian Elimination* (Implementation of Forward Gaussian with scaling, pivoting and back substitution along with the LU decomposition in CM Fortran on the CM-5)
3. R. Jimenez, *Supercomputing Computation: A Statistical Application* (Investigates different data layouts for basic statistical analysis including χ^2 test of fitness, finding strong dependence on the number of rows used in the data decomposition, with performance results for both NCSA Y-MP/4 and CM-5)
4. R. Kandallu, *Cray Y-MP vs. CM-5 Using the Starter Problem* (Explored the class Cray starter problem on both the Y-MP and CM-5, but was unable to optimize the code so that the CM-5 was competitive).
5. M. E. Papka and T. M. Roy, *Marching Cubes Revisited* (Implementation of Lorensen and Cline's Marching Cubes algorithm for generating graphical surfaces of constant value using the CM-5 CMMD message-passing communication library, NCSA's Data Transfer Mechanism with comparisons to SGI Onyx, Cray Y-MP/464 and CM-5 with C* implementations; CMMD performs much better than the C* version on the CM-5 and other implementations).
6. J. J. Westman, *A Survey of Square Matrix Multiplication Methods on the CM-5* (Investigates many different implementations of square matrix multiplication on the CM-5; Cannon's algorithm was the fastest, followed closely by the built-in MATMUL function, while DOTPRODUCT and other loop reordering techniques performed poorly).
7. Q.-L. Zhang, J.-J. Chen and Z.-X. Zhao, *Testing the CMF Random Number Generator 'CMF-Random' on the NCSA Connection Machine CM-5* (Test of CMF random function using basic statistical moments up to the third).
8. Z.-H. Zhang, *Iterative Methods on the CM* (Comparison of CM-2 and CM-5 implementation of Fortran 90 shifting code for Laplace's equation generalized to a more general parabolic equation for the flow with drift as a simplified mode of 2-dimensional uniform flow over a circular cylinder and coupled with the corresponding parabolic equation for conduction of heat).

The graphical visualization projects were the highlight of the projects and some of the students were simultaneously preparing for exhibits at Supercomputing '93 and Supercomputing '95, while taking the introductory class.

3 Supercomputing Workshop Description

For more advanced and intensive instruction in supercomputing, we have a workshop course called *MSC573-EECS574 Workshop Program on Scientific Supercomputing*, which was formally a full time program called the *UIC Workshop Program on Scientific Supercomputing* [5]. In the former program graduate students spent full time for one term in the workshop and were rewarded with a research assistantship. Now, with state universities down-sizing programs, the workshop was

converted to a multi-hour course administered by the mathematics department and cross-listed the electrical engineering department. Students still must bring a computational research problem to the workshop course. Much of the computational science and engineering thrust of our local training comes from these student research problems, which have ranged from chemical reactors to NASA projects.

The Fall 1993 students of the workshop course had access to the Cray Y-MP4/464 at NCSA, the CM-2 as well as CM-5 at NCSA and the ES9000 at CTC. We had six students with about two consistent visitors during the this first term as a multi-hour class.

The high performance computing content remains basically the same in the workshop course as in the program, but some of the more general education efforts in computer science concepts no longer fit within the time constraints. The topical semester course syllabus gives the current contents:

Supercomputing Workshop Course Syllabus

Catalog description: Intensive laboratory immersion in supercomputing; working with existing computer programs to improve their performance by scalar, vector and parallel optimization; techniques of compilation, profiling, debugging under CMS and Unix. Required co-registration of 8 hours thesis research or special projects in participant's department.

Prerequisites: Graduate standing; MCS 572 Introduction to Supercomputing is encouraged as a prerequisite; Consent of instructor; Appropriate research project approved by instructor and student's advisor.

Semester Credit hours: 4

List of Topics	**Hours**
• What is Scientific Supercomputing?	1 hours.
• Introduction to Supercomputer Architectures & Operating Systems	4 hours.
• Special Supercomputer Architectures: Cray, IBM, CM, experimental	4 hours.
• Operating Environments: VM/CMS, Unix/UNICOS/AIX	3 hours.
• Software Design (including timing and debugging)	9 hours.
• Code Optimization (Scalar, Vector, Parallel, Input/Output)	18 hours.
• Numerical Considerations and Libraries	7 hours.
• Graphics	7 hours.
• Communications and Networks	2 hours.
• Distributed Computing	2 hours.
• Basic Procedures	13 hours.
• Controllers, Batch, and Background	4 hours.
• Introduction to NSFnet	1 hours.
• Introduction to Assembler	2 hours.
• Symbolic Computation	1 hours.
• *Total.*	*78 hours.*

Required Texts: There are no required texts. Many readings in the literature are recommended during the term. Vendor's manuals, local user guides and training material are provided.

Visiting lectures during Fall 1993 talked about computer memory management, Unix operating systems, Cray Supercomputers, Connection Machine CM-5 and automatic differentiation tools. Also very well received, master supercomputer designers like Seymour Cray, Danny Hillis and Guy Steele were brought to the workshop in video (*University Video Communications*).

Much of the material of the course *MCS572 Introduction of Supercomputing* was quickly reviewed using overhead transparencies, such as the material mentioned in the previous section. However, in the workshop, we spent more time on production oriented Unix tools like makefiles,

on optimization examples in both Cray and C, and on performance analysis tools like *Flowtrace*, *Perfview* and the X-Window oriented *ATexpert* (Autotasking Expert System) for the Cray. Cray training optimization manuals were extremely useful for these extra topics. One exceptional student even published a paper expanded from a workshop research project in *IEEE Computer Magazine*.

4 Local User's Guides to Overcome Limited Center Documentation and Communications Problems

We produced local user's guides that greatly expedited student access to the Crays, Connection Machines and several earlier parallel processors. National center and vendor guides or manuals are generally much too large, very operating system oriented, and too segmented for use in a single term of a supercomputing class. Most critically, they do not tell a beginning student the practical, basic things he or she needs to know to merely run a simple test program, while including much material that the ordinary user does not need to know. The concise local guide is essential for the efficient training of entry-level students. The need for a local guide illustrates the critical difference in local training versus national center training. The local guides expedite the process of getting students knowledgeable enough to begin running applications early. However, center documents serve as valuable, authoritative background resources.

Effective instruction in supercomputing requires that supercomputing operating systems and the communications system interfaces must be made as transparent as possible. The time spent on supercomputing optimization must be maximized, while that on system details must be minimized. A brief, self-contained and hands-on local user's guide is needed that goes from the local session to the remote supercomputer, illustrated by real examples and minimal command dictionaries. Our guides allow students to concentrate on code optimization with minimal systems problems, and prepares them for finding more advanced information if needed. For example, the current local guides along with their web addresses are *MCS572 UIC Cray User's Local Guide to NPACI Cray T90 Vector Multiprocessor and T3E Massively Parallel Processor, Version 14.00* {*http://www.math.uic.edu/~hanson/crayguide.html*} give basic details for the UIC user to directly access to the Cray machines at SDSC, using sample sessions via high-speed network links. This includes file transfer between NPACI/SDSC and UIC via FTP. Also, included are sample program execution steps. A contents list for the 2000 version of the Cray user's local guide is given below:

Abbreviated User-Guide Table of Contents

13. The ex Editor.
14. The vi Editor.
15. Interrupts Dictionaries for Telnet and Unix.
16. T90 Fortran90 and other Extensions.
17. MPI Message Passing Programming on Crays.
18. PVM Message Passing Programming on Crays.
19. Cray T90 f90 and cc Timing Utility Functions.

The Cray local guide and their contents will not be given here, but are available at the above web address. These guides have to be significantly revised for each offering, because of operating system, hardware and compiler changes have occurred in the previous year, e.g., the inclusion of message passing machines in 1995 or the non-Cray supercomputers that will be used in the next offering.

Communication difficulties make the use of local guides essential. Usually there are relatively few problems when communicating between similar computing systems, such as Unix to Unix. However, when there are communication incompatibilities, remote center documentation is usually of little help. Now many students are conversant with Unix, but in the earlier days introductory Unix tutorials were needed. Also communications problems were major problems due to early Unix and IBM incompatibilities, another significant difference between local and national center training. The local guide is thus essential for describing the empirical characteristics of the communication systems truncations that do not behave as expected. Since communications and other problems continually arise, we have found that keeping in close e-mail contact with students is essential to keep the amount of time students spend on system problems reasonable.

5 Other Supercomputing Courses and On-Line Web Documents

In recent years much documentation has become available on the World Wide Web accessible through a web browser. We have placed a large amount of our own documentation and a large number of links to useful external documentation on the class home page {http://www.math.uic.edu-/~hanson/sylm572.html}. These web links include related on-line web courses, hardware and software information, supplementary texts, additional supercomputing literature. Of course these links lead to even more supercomputing information. An example of software information linked are tutorials, guides and texts on MPI message passing package. Hardware information includes documentation on Crays, Connection Machines and other machines.

However, one of the most important kinds of information contained via web links is on-line material for supercomputing and parallel processing courses elsewhere. We will limit our discussion of related course to these on-line materials since they are usually updated annually and freely available to supercomputing students, whereas supercomputing texts quickly become stale and may be too expensive for many state university students. A selection of texts are given in the *References*, and many more are given on the class home page. The on-line course material allow instructors and students to find a wealth of information and to permit a free choice of material closer the interest of the class and individual student. This shared information certainly has the benefit of a global education project. In addition to our own introductory and workshop course notes available on the class home-page previously cited, there are several other courses with on-line material that merit mentioning. The most extensive endeavor is the *Computational Science Education Project (CSEP)* {http://csep1.phy.ornl.gov/csep.html} sponsored by the U. S. DOE and written by a multitude of authors. Some pertinent topics in supercomputing are Numerical Linear Algebra,

Fortran 90, Monte Carlo Methods, Scientific Visualization, Tutorials for Parallel Platforms, and Computer Architecture. Demmel of Berkeley has made material available for *cs267 Applications of Parallel Computers* {*http://HTTP.CS.Berkeley.EDU/~demmel/cs267/*}, covering similar topics in parallel processing, but with a strong integration of the numerical linear algebra packages of LAPACK and ScaLAPACK into the course. During Spring 1996, Demmel's course is being co-taught in part with Edelman of MIT via a coast to coast, high speed video link. The MIT course is *18.337 Parallel Scientific Computing* {*http://web.mit.edu/18.337/*} covers parallel processing, multipole methods, linear algebra including sparsity and PDE related applications, with extra emphasis on graph and geometric algorithms. Another available web course, CS4676 *http://www-ugrad.cs.colorado.edu/%7Ecsci4576/*}, is by Jessup and co-teachers at Colorado in Boulder. This course is aimed at undergraduates, but the material is usable at higher levels, containing many tutorials and lab manuals on parallel computing and associated computer tools such as scientific visualization and Unix, as well as computational science applications.

The computational science and engineering applications vary widely depending on the local instructional environment. There is a large amount of material in many flavors available on the World Wide Web and supercomputing time available at the cost of a short proposal to the national or state centers that an instructor can use to help create a local supercomputing class.

6 Conclusions

The experience we gained as graduate students and faculty from these offerings of the course and workshop have been invaluable. The courses have been essential in keeping the computational science and engineering students abreast of new advances in computer architecture. We find they help to greatly extend the time the students' training remains viable, because the courses are at the leading edge of technology. They also have help students to further their thesis research. The local training also extends the outreach of the national supercomputer centers at much less cost than visiting the center for training.

We find the principal disadvantage in teaching *Introduction to Supercomputing* and *Workshop Program on Scientific Supercomputing* is that it takes a super amount of effort on the instructor's part. A major problem is that supercomputing material has to be significantly updated each year for major system changes. Another major problem in not supercomputing itself, but the communications difficulties in accessing remote supercomputing sites from a large sample of student computing environments. The communications problem would be lessened greatly if a generous number of Unix workstations with good communications properties were available at the local site.

Our purpose for this paper is to pass on sufficient practical advice on supercomputing as an aide to others who wish to run similar local training courses. The great value is training future generations of students for future generations of computer systems, keeping both students and instructors on the leading edge.

Acknowledgments

Our project has been generously supported in part over the years by Argonne National Laboratory Advanced Computing Research Facility, University of Illinois National Center for Supercomputing Applications, Cornell National Supercomputing Center, Pittsburgh Supercomputer Center and the San Diego Supercomputing Center (SDSC/NPACI). The workshop has received direct support from IBM, Cray, Thinking Machines and other vendors. The workshop program has been locally supported Computer Center and more recently by the Department of Mathematics, Statistics, and

Computer Science as well as the UIC Laboratory for Advanced Computing. This project has been indirectly supported by the National Science Foundation under Grants DMS 88-0699, 91-02343, 93-01107, 96-26692 and 99-73231 via knowledge transfer from my sponsored supercomputing research experience to my classes.

References

[1] J. J. Dongarra, I. S. Duff, D. C. Sorensen and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM, Philadelphia, 1998.

[2] L. D. Fosdick, E. R. Jessup, C. J. C. Schauble, G. Domik, *An Introduction to High-Performance Scientific Computing*, MIT Press, 1996.

[3] G. Golub and J. M. Ortega, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, Orlando, FL, 1993.

[4] J. L. Gustafson, G. R. Montry and R. E. Benner, "Development of Parallel Methods for a 1024-processor hypercube," *SIAM J. Sci. Stat. Comp.*, vol. 9, no. 4, Jul. 1988, pp. 609-638.

[5] F. Hanson, T. Moher, N. Sabelli and A. Solem, "A Training Program for Scientific Supercomputing Users," in *Proceedings of Supercomputing '88*, Nov. 1988, pp. 342-349.

[6] F. B. Hanson, "A Real Introduction to Supercomputing: A User Training Course," in *Proceedings of Supercomputing '90*, Nov. 1990, pp. 376-385.

[7] F. B. Hanson, "Computational dynamic programming on a vector multiprocessor," *IEEE Trans. Automatic Control*, vol. 36(4), pp. 507-511, April 1991.

[8] R. W. Hockney and C. R. Jesshope, *Parallel Computers 2*, Taylor-Francis, Philadelphia, 1988.

[9] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.

[10] J. M. Levesque and J. W. Williamson, *A Guidebook to FORTRAN on Supercomputers*, Academic Press, New York, Dec. 1988.

[11] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, New York, 1988.

[12] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[13] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.

[14] M. Wolfe, *Optimizing Supercompilers for Supercomputers*, MIT Press, Cambridge, MA, 1989.