Achievements and Prospects for Parallel Computing

Geoffrey Fox

CRPC-TR90083
June 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS–29

C^3P-927

CRPC-TR90083

Achievements and Prospects for Parallel Computing

Invited Talk at International Conference on

Parallel Computing:

Achievements, Problems and Prospects

Anacapri, Italy June 3–9, 1990

Geoffrey C. Fox

gcf@nova.npac.syr.edu

Achievements and Prospects for Parallel Computing

Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
111 College Place
Syracuse, New York 13244
gcf@nova.npac.syr.edu

January 15, 1991

Abstract

Parallel computing works for the majority of large scale computations. The development of parallel hardware designs has been largely transferred to industry, while universities continue major research efforts into better software environments. We describe a classification of problems and how different software models are needed for portable user friendly high performance implementations on parallel machines. The education of a new generation of computational scientists will be a major challenge to our universities.

1 Introduction

In 1981, I listened to a Physics colloquium given by Carver Mead at Caltech. Carver was a pioneer in the development of VLSI, and explained how developments in this technology would inevitably lead to parallel computers. At the time, we were using our relatively new VAX11/780 to perform crude Quantum Chromodynamics (QCD) simulations with week-long runs. That talk changed my career, as I realized that QCD would naturally run in parallel. While waiting for the initial 64 node 8086-8087 based hypercube to be completed, I realized that we (now a collaboration with Carver's colleague, Chuck Seitz, in the Caltech Computer Science Department) had a machine that appeared to be generally useful in science. Thus was born the Caltech Concurrent Computation Program (C^3P) which, from 1983–1990, investigated the question:

> "Is the hypercube (and later more general parallel architectures) effective in numerically intensive scientific and engineering calculations?"

I have described the history of C^3P elsewhere [Fox:87d], [Fox:88oo] with the culmination [Fox:89n] that in 1989 (eight years after believing Carver's dream), we finally obtained parallel computers that were truly Supercomputers. In a series of articles, I have described some highlights of the fifty significant applications we built for the hypercube [Fox:89b], [Fox:89i], [Fox:89n], [Fox:90o]. Here, I will concentrate on other areas—hardware, software and education—but I emphasize that these were each built on the experience of

"Using Real Hardware with Real Software to Solve Real Problems"

Indeed, this motto will continue to be a guiding principle in the new program that I am setting up at Syracuse University.

The research of C^3P taught me several lessons.

- Parallel Computing works for essentially all computationally intense problems.
- Computation will grow in importance—fed by high performance parallel computers—and will "change the world"—in academia, industry, day to day life and government applications.
- We need to develop new educational approaches to train the next generation of scientists who will have the interdisciplinary skills to exploit new computers for new problems.

My time at Caltech, and in particular with C^3P, were immensely rewarding and Caltech was one of the very few universities where a program like C^3P could have succeeded. Now, we know both that parallel computers work and how to exploit them for leading edge academic applications. The new challenges are different. We must make parallel computing available to a broad range of users both in academia and more importantly in the real world of industry—only in this way will it enter the mainstream of computing. This principle underlies my vision of the future of parallel computing. Correspondingly, at Syracuse University, we will have some similarities with C^3P. It will be interdisciplinary but with a different focus—the Syracuse Center for Computational Science will emphasize education and the integration of parallel computing into industry.

This paper has three major sections describing respectively, the status and prospects for hardware, software and education. We will see my focus on applications in the chapter on software, which we discuss from this point of view.

2 Hardware Prospects

Parallel Computers were surveyed by Paul Messina for this conference, and I will just make a few general remarks. University and industry research has developed the basic principles of parallel computer hardware, and this knowledge has been successfully transferred to industry which can be expected to produce future hardware. The time of large scale machines designed and built in universities has passed.

Existing machines can be classified into three broad architectural areas; MIMD distributed memory or multicomputers; SIMD distributed memory; MIMD shared memory or multiprocessors. My interest is in the first two—distributed memory—machines which can be expected to realize the highest and most cost effective performance. Shared memory machines will certainly be important in the near future, and this software (as opposed to hardware) structure will be of great importance even on distributed memory hardware.

There are several effective large-scale distributed memory machines now available; the Connection Machine CM-2, INTEL i860, and NCUBE2 custom hypercube all having multi-gigaflop peak performance. Smaller (in size and price) systems include those based on transputers, and the SIMD AMT DAP and Maspar machines. Although all these machines are very usable, many of them have rather clear design or implementations flows. These are in the areas of communication bandwidth and latency, input/output subsystems including the host or interface computer. More generally, system integration needs to be improved. The communication issues can be illustrated with the measurements shown in Table 1 [Dunigan:90a], [Hu:90a], [Williams:90e].

We would expect future systems to be able to achieve $t_{\text{comm}}/t_{\text{flop}}$ (typical node to node communication time per word/typical floating point node calculation time) in range $1 \rightarrow 10$ which is the value needed for general applicability of the machine. However, even the current i860 based INTEL systems with the high value of $t_{\text{comm}}/t_{\text{flop}} \sim 60$ perform well on many applications.

The issue of latency is less clear and cannot be separated from questions of network topology or node interconnection. Often these points, latency and

Machine	t_{flop} node Floating Point Time (μ s)	$t_{\text{comm}}/t_{\text{flop}}$ Communication Time/64 bit Word	$t_{\text{lat}}/t_{\text{flop}}$ Message Latency
		(measured in units of a typical node floating point time)	
Transputer System — Direct (no routing)	2	2.2	13
— Express (routing software)	2	2.8	300
INTEL i860 Express	0.1	60	1400
NCUBE 1 Express	7	2	100

Table 1: Communication and Calculation Parameters

topology, are treated separately with more attention being given to the interconnection scheme. We now find in current machines, interconnects using meshes, hypercubes, and switches with or without automatic routing hardware. These have all been successful and there is no convincing argument in favor of any one interconnect scheme. Most theoretical studies use unrealistic message passing models, such as random message traffic which is a poor representation of the highly correlated communication needs in scientific computation. Further, the current systems have too few nodes to allow decisive experiments which could distinguish different interconnect methods. Even less is known about latency which deserves greater study, and this one of the areas where further research is needed to refine the architecture of future machines.

There is no uniquely good machine, as is illustrated by the current commercial systems which are all broadly successful. We can expect architectural diversity to continue within the design space indicated by today's existing and planned machines. Such a range of machines implies that the software must be portable and insensitive to changes in topology, granularity (number of nodes, memory per node), latency and if possible the choice between SIMD and MIMD. To motivate our research in algorithms, applications and software, we can imagine what one of the year 2000 teraflop computers will look like. It could have perhaps $N \sim 10,000$ nodes, each realizing some 200 megaflops. Maybe the architecture would be hierarchical; perhaps a 1024 element hypercube, with each element consisting of around ten processors sharing the same memory system.

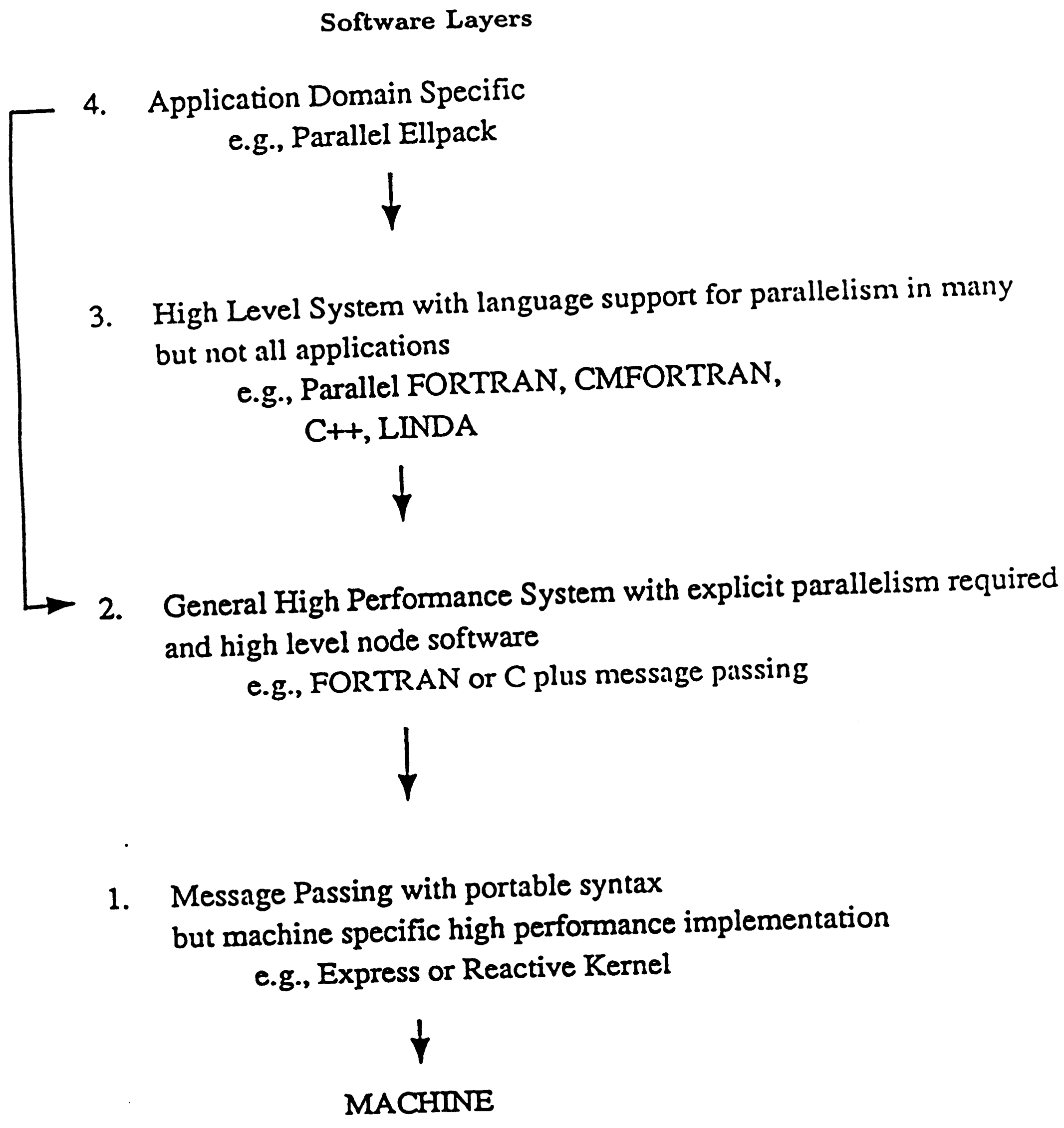
Such machines will exist before the year 2000 but at higher price and with technology trade-offs (e.g., are optical interconnects cost-effective?) which will affect the architecture and hence "balance" and "generality" of the machine. We can expect that some or all of today's parallel computer companies will grow as the field matures; new enterprises will be formed and traditional computer manufacturers will move into the field. In particular, Digital and IBM, with their systems integration experience, may be major players. This growth is currently limited by software tools and the need to increase the number of industrial applications that run on parallel machines.

In the above, we have discussed "conventional" parallel machines. These will be the top of the line supercomputers and get unbelievable performance on most problems. In particular, special purpose machines, such as those in high energy physics (QCD) and computational fluid dynamics, will no longer be cost effective. Major special purpose hardware will only be justified in areas like neural networks, which use a radically distinct computational methodology.

3 Software

The greatest challenge, and least agreement among practitioners, is found in software support for parallel computers. We will discuss this in the context of the layered structure shown in Figure 1, and introduced to me by Ken Kennedy for the Center for Research in Parallel Computation. Of the four layers, the uncertainty centers on the third; namely, on high level relatively general systems such as Parallel Fortran. It is not clear as to the nature of this software level and if it even works (exists)! Before returning to this, let us quickly discuss the other levels. We should first note that we will not discuss the support software that is critical for a reasonable environment; this includes debuggers, libraries, performance evaluation and visualization tools. These are important, but not the subject of this paper.

The top layer, with software systems such as NASTRAN (finite elements) and CHARMM (molecular dynamics), is particularly important in industry. These systems will certainly be at least as important for parallel systems, especially as there is an even greater need to shield the user from the immature parallel computers. However, it inevitably will take some time to produce these systems. This work will be accelerated when standards are established for the lower levels, two and three. Fortran and C, plus message passing (level two), are clearly effective in giving high performance implementations on MIMD distributed machines. Indeed, essentially all successful work, including the some fifty hypercube applications we did within C^3P at Caltech, has used this approach. As long as you parameterize the number of nodes, this software is formally portable; and the same program will run on the NCUBE and INTEL hypercubes, a network of workstations, and even multi-headed IBM 3090's and CRAY vector Supercomputers. This portability is supported by the commer-



Software Layers

4. Application Domain Specific
   e.g., Parallel Ellpack

3. High Level System with language support for parallelism in many but not all applications
   e.g., Parallel FORTRAN, CMFORTRAN, C++, LINDA

2. General High Performance System with explicit parallelism required and high level node software
   e.g., FORTRAN or C plus message passing

1. Message Passing with portable syntax but machine specific high performance implementation
   e.g., Express or Reactive Kernel

MACHINE

Figure 1: Software Layers for Parallel Computers

cial system, *Express*, for level-one software which is the outgrowth of our CrOS software developed at Caltech [Fox:88a]. However, this portability does not preserve performance; each machine involves different optimizations in the message passing and node programs to reflect tradeoffs for

- Granularity (number of nodes)
- Communication bandwidth and latency (1)
- Interconnect topology
- Overlap (or not) of communication and calculation
- Node architecture

We can illustrate this with our QCD (high energy physics) calculations on the hypercube. The original code developed for the Cosmic Cube in 1983 was run with little change on the later Mark II (JPL) and commercial NCUBE-1 hypercubes. However, substantial changes were needed when it was used on the more powerful (500 megaflops on 128 nodes) Mark III (JPL) hypercube. The relatively higher communication latency implied substantial rearrangement of the message passing and a reordering of the algorithm to allow communication to be blocked into a smaller number of larger messages. The poor compiler required assembly language coding of key parts of the node program to exploit the WEITEK XL vector floating point unit. We never did exploit the possible overlap of communication and calculation which was available on the Mark III. This would have increased performance by 25% and was possible in principle, but clumsy to implement. We believe that these optimizations in (1) above, can be performed by appropriate compilers, and this is a minimal goal for the higher level-three software systems.

There are many message passing systems including:

OCCAM	(Inmos)
Reactive Kernel/Cosmic Environment	(Caltech)
Express	(ParaSoft, Caltech)
Trollius	(Cornell, Ohio State)
Tiny	(Edinburgh)
Vertex	(NCUBE)
NX	(INTEL)

These offer comparable functionality with different tradeoffs in the area of portability, performance and collective (higher level) communication support. The latter includes broadcast and its generalizations, and the various primitives for data shuffling developed particularly by [Ho:86a] and [Ho:89f], and others [Fox:86c], [Fox:88g], [Fox:88h], [Stout:87a], [Valiant:88a]. Further work is needed to define the more sophisticated primitives, but it should soon be possible to

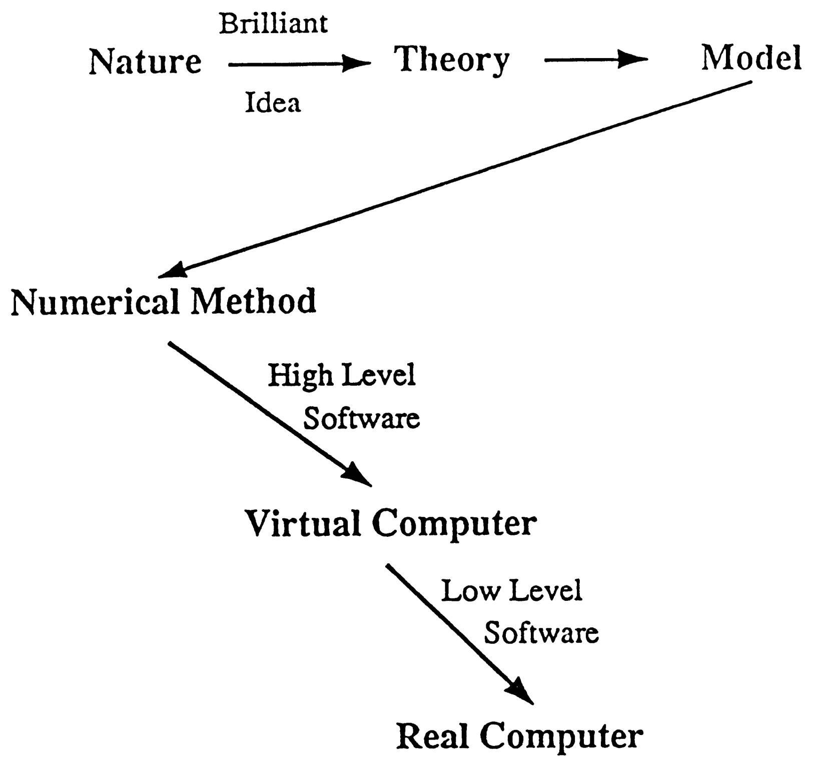


Figure 2: Computation and simulation as a series of maps.

agree on a standard message passing interface which would provide a portable base on which to build the higher level software.

As illustrated in Figure 2, we view software as a map of a problem onto a computer. As such, it is sensitive to the architecture (structure) of both problem and computer. My goal would be software specialized (optimized) for a broad class of problem architectures which is designed to run portably on all parallel architectures. We will illustrate this idea a little later.

We would like to develop a classification of problem architectures and present here a simple division of problems into three classes; each of which presents different software challenges and solutions (at level three). Previously, we have introduced the concept of the spatial and temporal aspects of a problem. We can, rather, generally view a computation as an algorithm performed on the many elements of a data set; the algorithm is repeated corresponding to an increasing iteration count or the simulation of the data set for increasing time. The structure (often called computational graph) of the data set and algorithm we will refer to as the spatial structure of the problem. We call the nature of iteration, recursion, or time simulation, as the temporal structure. We described in [Fox:88b] three distinct temporal structures; synchronous, loosely synchronous and asynchronous. Synchronous problems are characterized by identical al-

Class	Temporal Structure	Spatial Structure	Examples	Natural Parallel Software	Natural Parallel Machine
I	Synchronous	Regular Static	Finite Difference QCD Matrix Algebra	FORTRAN 90D (CMFORTRAN)	SIMD Distributed Memory
II	Loosely Synchronous	Irregular Dynamic	Finite elements with adaptive irregular [Williams:89b] mesh Cluster approach N-body Problem [Fox:89t], [Salmon:90a]	PARTI [Berryman:91a], [Saltz:87a],[Saltz:90a] Extensions of FORTRAN 90D?	MIMD Distributed Memory
III	Asynchronous	Loosely Coupled	Event Driven Simulation Computer Chess Real Time Control System Transaction Analysis	Functional Decompositions LINDA [Gelernter:89a] STRAND (PCN) [Foster:90a] Object Oriented Systems	Distributed Computing. MIMD Shared Memory

Table 2: Three Problem Architectures

gorithms being applied to all points in their data set. Loosely synchronous problems do not have a tight lockstep with microscopic synchronization, but rather macroscopic time synchronization. An example is a simulation evolving from t to $t + \delta t, t + 2\delta t$ with different algorithms at each data point, but natural synchronization at the beginning and end of each time step. Asynchronous problems have no natural synchronization in the time evolution of the different data points.

In table 2, we introduce three problem architectures which are imaginatively labelled Classes I, II and III. These classes are not clearly defined and merge into each other. Further, a given application may consist of several components which fall into different classes. From our point of view, Fortran 90 is not regarded as the natural language for SIMD computers. Rather, it is the natural language for synchronous (SIMD) Class I problems. One can, in fact, implement this language efficiently on SIMD and MIMD machines. This portability between SIMD and MIMD has been implemented for the analogous language C* [Quinn:90a], the functional language crystal [Chen:88b], and we are currently implementing a Fortran 90 to F77+MP (MP = message passing) preprocessor for MIMD machines [Wu:90c]. We expand on this later.

Class III problems consist of irregular problems with no time synchronization. These are, in principle, very hard to parallelize as is illustrated by the major efforts devoted to parallelizing event driven simulations [Wieland:89a]. This class can be parallelized when the parts are independent or loosely coupled and so synchronization overheads are small. The loose coupling makes these problems suitable for distributed systems (e.g., networks of workstations) which have lower communication bandwidth and higher latency than the dedicated parallel machines. The powerful (and hence higher software overhead) programming environments for distributed computers are natural here; an object oriented formalism is one obvious possibility.

The situation with Class II problems is less clear and a major research area for us. We have described in [Fox:89t] the astrophysical N-body simulation with the $N \log N$ Barnes-Hut clustering method. It is not easy either to implement on SIMD machines or to code in Fortran 90. We can understand this as stemming from the fact that the algorithm's data structure is an irregular dynamic tree. This cannot be represented in Fortran 90 which only has array and vector data structures. We are currently investigating if adding extra (e.g., tree) data structures to Fortran 90 will allow one to address the N-body problem and other applications (e.g., quicksort) with a recursive or divide and conquer architecture. Saltz and his colleagues at NASA ICASE have shown that Fortran can be used for irregular finite element (more generally sparse matrix) problems with extensions to handle pointers [Berryman:91a], [Saltz:90a]. Similarly, Fortran 90 with vector value subscripts can handle at least some of these problems [Johnsson:90a], [Mathur:89a]; correspondingly, the Connection Machine can run these problems successfully. We can understand these successes as reflecting that an array, albeit an array of pointers (addresses), is an appropriate data structure. Fortran 90D is Fortran augmented by those possible additional data structures and parallel decomposition constructs will have some general utility. We expect that most (scientific) problems in Class I and II will be elegantly expressed in Fortran 90D. With similar extensions implemented as "user hints", we should be able to parallelize F77 codes in these classes. Ken Kennedy's group at Rice is building this Fortran 77D compiler for distributed memory parallel machines [Fox:90n] [Wu:90b]; this will either produce directly F77+MP or possibly more usefully Fortran 90D.

CMFortran implements the essential parallel constructs in Fortran 90 italic and also the *forall* statement. We include the latter in Fortran 90D and can illustrate its importance with a typical chemical reaction [Wu:90a], [Hipes:90a] or potential calculation. This involves an "embarrassingly parallel" (Class III) calculation of the values of matrix elements combined with (Class I) matrix algebra (multiplication, inversion and eigenvalues). These two steps may be represented respectively by the *forall* and natural parallel array constructs in Fortran 90D. We need decomposition statements in Fortran 90D and these are extensions of the LAYOUT and ALIGN commands of CMFortran. These instruct the compiler how break up (either statically or dynamically) the arrays over the nodes [Fox:90n].

Above, we have described how Fortran 90 provides a possible high level language for Class I, and extensions may extend its utility into Class II. A major goal is to examine further applications to understand which features cannot be expressed in one of these ways and so provide a test suite of "skeletons in the programming language closet". These (like the irregular finite element and N-body problems) are the applications for which we need new languages. We do not really need new ways of expressing matrix multiplication and Laplace's equation; what we know now is sufficient. Nevertheless, many new language projects are tested and perhaps motivated by these unrealistic problems. I hope that our classification of problems and its refinement will focus the development of parallel languages and environments on the difficult problems that need to be solved.

We conclude by illustrating the parallelization of Fortran 90 for four small Class I problems.

4 Experiments with Fortran 90 on MIMD Multicomputers

These results come from the joint project [Wu:90b] between Rice University (Ken Kennedy's group) and my research group at Syracuse. We are building a library of problems designed to test languages and parallelizing compilers. For each problem, we build several versions; in particular,

P1) Fortran 77

P2) Fortran 90D (CMFortran)

P3) Hand coded Fortran 77 + Message Passing

P4) "What we think a parallelizing compiler could produce for a MIMD multicomputer from Fortran 77 (P1)"

P5) "What we think a parallelizing compiler could produce for a MIMD multicomputer from Fortran 90 (P2)"

We are using the results of these experiments to build the compilers that realize our expectations for P4) and P5). The Fortran 90D compiler is a collaboration with the ParaSoft Corporation.

Our results in Tables 3A, 3B, 3C, 3D are only preliminary but already interesting.

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
P3) Original Hand Coded F77+MP	85.4	58.1	31.1	16.0	8.42
P5) F77+MP from Fortran 90	80.0	50.2	26.6	13.8	7.72
P3) Revised Hand Coded F77+MP	73.4	50.1	26.9	13.8	7.53

Table 3A. Gaussian Elimination (256 x 256 matrix)

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
P3) Hand Coded F77+MP	71.7	35.9	17.9	8.98	4.83
P5) F77+MP from Fortran 90	139.6	69.1	35.5	18.1	9.40

Table 3B. N-body Simulation (1024 Particles)

	Number of Processors on iPSC2 Hypercube				
	1	2	4	8	16
P3) Original Hand Coded F77+MP	36.8	23.3	14.2	8.32	4.82
P3) Optimized Hand Coded F77+MP	13.0	6.67	3.42	1.75	0.91
P5) Optimized F77+MP from Fortran 90	18.8	10.1	5.36	2.84	1.50

Table 3C. Fast Fourier Transform (16384 Points)

Implementation	Size of Code (lines)	Machine	Performance (megaflops)
Original C Code	1500	CRAY X-MP (1 C.P.U.)	~ 1
P2) Fortran 90	600	CM-2	66
P5) F77 Produced by Hand from Fortran 90	1500	CRAY Y-MP (1 C.P.U.)	20
P5) F77+MP by Hand from Fortran 90	1650	NCUBE-1 16 node hypercube	3.3
		NCUBE-2 16 node hypercube	20
		INTEL i860 16 node hypercube	80

Table 3D. Climate Modelling Code [Keppenne:90a]

For a problem with a simple topology, LU decomposition in Table 3A, the Fortran 90 code produced essentially as good a code as the direct Fortran 77+ Message Passing. Indeed, the "automatic" Fortran 90 procedure pointed out a possible improvement in our handed codes F77+MP; this is the difference between lines one and three of Table 3A. In Table 3B, our current automatic approach for the N-body problem loses a factor of two compared to the best parallel implementation; this is due to inefficient communication and one may need to change the Fortran 90 implementation to allow the compiler to optimize this. In this sense, the user will need to understand some issues of parallelism, even when writing "explicitly parallel" code as with Fortran 90. Note the example in Table 3B is the simple $O(N^2)$ algorithm and not the more interesting and challenging $O(N(\log N))$ approach discussed earlier. In Table 3C, we find a 50% degradation in performance on the FFT for the Fortran 90 approach. This indicates that Fortran 90 does not optimally support the hierarchical data structures found in the FFT. As already discussed, we expect that the final Fortran 90D language will include new data structures—over and above the arrays and vectors in Fortran 90. We already mentioned a possible "tree" data structure needed for the Class II clustering approach for the N-body problem.

Finally in Table 3D, we come to a "real", albeit small in code size, problem. The original climate modelling code has been used in production [Keppenne:89a] on CRAY and SUN computers. We saw in this project, an interesting division of labor. The first rewriting from C to Fortran 90 was performed by the application expert. The further conversions into Fortran 77 and Fortran 77+ message passing were performed by "computer scientists" without deep knowledge of the application [Keppenne:90a]. In this case, we believe that no automatic method could have parallelized the original C code, but that our planned automatic approach would be able to perform the MIMD parallelization from Fortran 90. The result of this project is a portable code running well on the CRAY, Connection Machine and hypercubes. Note that we even improved the sequential performance (line one vs. line two of Table 3D) by an order of magnitude. The original C code made extensive use of pointers which had several repercussions. It made vectorization hard on the CRAY; it made the code impossible to automatically parallelize as the "structure of problem" has expressed in dynamic pointer values; it made the code hard to port except by the domain expert.

Our initial experiments are sufficiently encouraging that we believe that a language like Fortran 90 will become an efficient vehicle for Class I applications. We also hope that it can be extended with higher level data structures to accommodate the more complex "problem architectures" seen in Class II problems.

5 Computational Science Education

The emergence of computation as a fundamental methodology implies major changes in our educational system which will challenge our universities, community colleges, and schools in the next decade [Fox:90o]. Computers are being used in schools, even at the kindergarten level. This is necessary, but not sufficient. We need not only to teach students how to program, but instill the understanding that computing is fundamental and will change the nature of both their careers and their day to day life. Computing like the traditional reading, writing and arithmetic is a key enabling skill whose use and *understanding* permeates all other activities. Looking at applicants for graduate school (in Physics at Caltech), I saw few students who viewed computers as anything but a rather tiresome tool. Why is this when most of them come from prestigious undergraduate institutions where the very latest computers abounded? The reason may be that such students are typically not given any courses that treat computation as fundamental and exciting. They are taught by faculty whose vision of computing comes from a time when computers were a useful but unexciting tool. This will change.

The Federal High Performance Computing Initiative developed the concept of interdisciplinary teams to tackle the grand challenges and their implementations on parallel machines. This idea was part of my Caltech Concurrent

Computation Program but interestingly enough, something else happened as well. Namely, the majority of the research was performed by interdisciplinary individuals who combined the skills of a computer scientist with those of an application area such as Physics or Chemistry. It is this phenomenon that inspired much of my thoughts on computational science. You may argue that "jack of all grades, master of none" implies that interdisciplinary scientists are not competitive. I believe this is a valid concern, but the negative conclusion can be avoided. I propose that Computational Science should be an interdisciplinary program with degrees given in the traditional fields. Undergraduates, Masters or Ph.D. students will get degrees in the Computational Science Program such as

- "Undergraduate degree in Physics in the Computational Science Program" (in my point of view, this is a long way of saying an undergraduate degree in Computational Physics)
- "Masters in Chemistry with a minor in Computer Science"
- "Ph.D. in Computer Science with a Masters Degree in Economics"

Computational Science students will need new curricula, and this will require cooperation and understanding from the traditional disciplines. For instance, the requirements for a degree in Physics within the Computational Science Program may require fewer base Physics courses than the traditional Physics degree, with this reduction made up by a set of computational courses. There are typically electives in any set of course requirements and so these reductions in the base (in our example, Physics) academic area need not be significant. In some institutions and some departments, these changes may not be accepted, and a different implementation of Computational Science will be needed. We must gain experience as to what works and what doesn't. Students will enter a program in Computational Science with a variety of aspirations and skills. For instance, freshman undergraduates may initially be interested in Physics, but later decide to graduate in Computer Science. We need a good set of courses to accommodate the different initial skills and allow a student to switch from a focus on Computer Science to one on an application area—or vice versa. Not only are such curricula in their infancy but, as found while teaching at Caltech, the lack of relevant books is a severe handicap. We will develop the curricula, books and the needed educational software.

A Chemist graduating from the Computational Science program will be well grounded in Chemistry, but have a broad knowledge of Computer Science. The latter will enable the Chemist to use computers and new computer science techniques more effectively than a Chemist without the computational science background. Thus, he or she will be a better Chemist, and in this way the interdisciplinary scientist is not handicapped. A Computer Scientist graduating in our new program will have a sound basis in Computer Science and an understanding of the computational needs and algorithms of one or more application areas. The latter knowledge will enable our Computer Scientist to develop better compilers, debuggers and visualization aids because he or she understands the needs of the applications.

Thus, I believe that Computational Science is an exciting and viable area which can be implemented within the existing academic framework. I intend to pursue this vigorously at Syracuse. In particular, we will develop the new courses (definition, notes, books, software). We will work with the international community to establish common guidelines for the teaching of Computational Science and build a consensus that it is important.

6 Conclusions

Technology has opened several new opportunities, both in research and education. We are at the beginning of changes which will have profound implications for academia, industry and society. Parallel computing, discussed here, is only one part of this revolution.

I would like to thank Paul Messina and his colleagues, especially Almerico Murli from the University of Naples, for their warm hospitality and an excellent conference. This work was supported by the Department of Energy: Applied Mathematical Sciences—Grant: DE-FG03-85ER25009; Joint Tactical Fusion Program Office; and the National Science Foundation under Cooperative Agreement No. CCR-8809165 — the Government has certain rights in this material.

References

[Berryman:91a] Berryman, H., Saltz, J., and Scroggs, J. "Execution time support for adaptive scientific algorithms on distributed memory machines," *Concurrency: Practice and Experience*, 1991. Accepted for publication.

[Chen:88b] Chen, M., Li, J., and Choo, Y. "Compiling parallel programs by optimizing performance," *Journal of Supercomputing*, 2:171–207, 1988.

[Dunigan:90a] Dunigan, T. H. "Performance of the intel ipsc/860 hypercube." Technical Report ORNL/TM-11491, Oak Ridge National Laboratory, 1990.

[Foster:90a] Foster, I., and Taylor, S. *Strand: New concepts in Parallel Programming*. Prentice Hall, Englewood Clifs, New Jersey 07632, 1990.

[Fox:86c] Fox, G., and Furmanski, W. "Communication algorithms for regular convolutions and matrix problems on the hypercube," in M. T. Heath, editor, *Hypercube Multiprocessors*, pages 223–238. SIAM, Philadelphia, 1987. Caltech Report C3P-329.

[Fox:87d] Fox, G. C. "Questions and unexpected answers in concurrent computation," in J. J. Dongarra, editor, *Experimental Parallel Computing Architectures*, pages 97–121. Elsevier Science Publishers B.V., North-Holland, 1987. Caltech Report C3P-288.

[Fox:88a] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. *Solving Problems on Concurrent Processors*, volume 1. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1988.

[Fox:88b] Fox, G. C. "What have we learnt from using real parallel machines to solve real problems?," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 897–955. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-522.

[Fox:88g] Fox, G. C., and Furmanski, W. "Hypercube algorithms for neural network simulation the Crystal_Accumulator and the Crystal_Router," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 714–724. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-405b.

[Fox:88h] Fox, G. C., and Furmanski, W. "Optimal communication algorithms for regular decompositions on the hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 648–713. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-314b.

[Fox:88oo] Fox, G. C. "The hypercube and the Caltech Concurrent Computation Program: A microcosm of parallel computing," in B. J. Alder, editor, *Special Purpose Computers*, pages 1–40. Academic Press, Inc., 1988. Caltech Report C3P-422.

[Fox:89b] Fox, G. C. "Experience on the hypercube." Technical Report C3P-716, California Institute of Technology, February 1989.

[Fox:89i] Fox, G. C. "1989 — the first year of the parallel supercomputer." Technical Report C3P-769, California Institute of Technology, March 1989. Paper presented at the Fourth Conference on Hypercubes, Concurrent Computers and Applications.

[Fox:89n] Fox, G. C. "Parallel computing comes of age: Supercomputer level parallel computations at Caltech," *Concurrency: Practice and Experience*, 1(1):63–103, September 1989. Caltech Report C3P-795.

[Fox:89t] Fox, G. C., Hipes, P., and Salmon, J. "Practical parallel supercomputing: Examples from chemistry and physics," in *Proceedings of Supercomputing '89*, pages 58–70. ACM Press, November 1989. IEEE Computer Society and ACM SIGARCH, Reno, Nevada. Caltech Report C3P-818.

[Fox:90n] Fox, G. C., Hiranandani, S., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C.-W., and Wu, M.-Y. "Fortran D language specification." Technical Report SCCS-42; CRPC-TR90079, Rice Center for Research in Parallel Computation, December 1990.

[Fox:90o] Fox, G. C. "Applications of parallel supercomputers: Scientific results and computer science lessons," in M. A. Arbib and J. A. Robinson, editors, *Natural and Artificial Parallel Computation*, chapter 4, pages 47–90. MIT Press, Cambridge, Massachusetts, 1990. SCCS-23. Caltech Report C3P-806b.

[Gelernter:89a] Gelernter, D. "Multiple tuple spaces in Linda," in *Proceedings of Parallel Architectures and Languages Europes*, volume 2, page 366. Springer-Verlag, LNCS, June 1989.

[Hipes:90a] Hipes, P., Winstead, C., Lima, M., and McKoy, V. "Studies of electron-molecule collisions on the Mark IIIfp hypercube." Technical Report C3P-909, California Institute of Technology, April 1990. Published in Proceedings of the Fifth Distributed Memory Computing Conference, April 9–12, Charleston, South Carolina, IEEE.

[Ho:86a] Ho, C.-T., and Johnsson, S. L. "Distributed routing algorithms for broadcasting and personalized communication in hypercubes," in *Proceedings of IEEE 1986 International Conference on Parallel Processing*, pages 640–648, 1986.

[Ho:89f] Ho, C.-T. *Optimal Communication Primitives and Graph Embeddings.* PhD thesis, Yale University, May 1990.

[Hu:90a] Hu, L.-R., and Stiles, G. S. "Fluid dynamics on EXPRESS: an evaluation of a topology-independent parallel programming environment," in D. L. Fielding, editor, *Transputer Research and Applications 4.* IOS Press, 1990.

[Johnsson:90a] Johnsson, S. L., and Mathur, K. K. "Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer." Technical report, Thinking Machines Corporation, 1990.

[Keppenne:89a] Keppenne, C. L. *Bifurcations, Strange Attractors and Low-Frequency Atmospheric Dynamics.* PhD thesis, Universite Catholique de Louvain, 1989.

[Keppenne:90a] L., K. G., Ghil, M., Fox, G. C., Flower, J. W., Kolawa, A., Papaccio, P. N., Rosati, J. J., Shepanski, J. F., Spadaro, F. G., and Dickey, J. O. "Parallel processing applied to climate modeling." Technical Report SCCS-22, Syracuse University, November 1990.

[Mathur:89a] Mathur, K. K., and Johnsson, S. L. "The finite element method on a data parallel computing system," *Int. J. of High-Speed Computing,* 1(1):29–44, May 1989. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-742, Thinking Machines Corporation, Technical Report CS89-2.

[Quinn:90a] Quinn, M. J., and Hatcher, P. J. "Data-parallel programming on multicomputers," *IEEE Software,* pages 69–76, September 1990.

[Salmon:90a] Salmon, J. *Parallel Hierarchical N-Body Methods.* PhD thesis, California Institute of Technology, December 1990.

[Saltz:87a] Saltz, J., Mirchandaney, R., Smith, R., Nicol, D., and Crowley, K. "The PARTY parallel runtime system," in *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing.* Society for Industrial and Applied Mathematics, 1987. held in Los Angeles, CA.

[Saltz:90a] Saltz, J., Berryman, H., and Wu, J. "Multiprocessor and runtime compilation," *Concurrency: Practice and Experience,* 1990. Submitted for publication.

[Stout:87a] Stout, Q. F., and Wager, B. "Intensive hypercube communication I: prearranged communication in link bound machines." Technical Report CRL-TR-9-87, University of Michigan, 1987.

[Valiant:88a] Valiant, L. G. "General purpose parallel architectures." Technical report, Harvard University, January 1988.

[Wieland:89a] Wieland, F., Hawley, L., Feinberg, A., DiLoreto, M., Blume, L., Ruffles, J., Reiher, P., Beckman, B., Hontalas, P., Bellenot, S., and Jefferson, D. "The performance of a distributed combat simulation with the time warp operating system," *Concurrency: Practice and Experience*, 1(1):35–50, 1989. Caltech Report C3P-798.

[Williams:89b] Williams, R. D. "Supersonic flow in parallel with an unstructured mesh," *Concurrency: Practice and Experience*, 1(1):51–62, 1989. (See manual for this in code in C^3P-861 (1990)). Caltech Report C3P-636b.

[Williams:90e] Williams, R. D. "Express: Portable parallel programming." Technical Report C3P-944, California Institute of Technology, October 1990. Presented at Cray User Group, Austin, Texas.

[Wu:90a] Wu, Y.-S. M., Cuccaro, S. A., Hipes, P. G., and Kuppermann, A. "Quantum mechanical reactive scattering using a high-performance distributed-memory parallel computer." Technical Report C3P-860, California Institute of Technology, January 1990. Accepted for publication in Chemical Physics Letters.

[Wu:90b] Wu, M.-Y., and Fox, G. C. "Test suite and performance for Fortran90 compilers." Technical Report SCCS-40, Syracuse Center for Computational Science, 1990.

[Wu:90c] Wu, M.-Y., and Fox, G. C. "An outline of fortran90 compiler for distributed memory systems." Technical Report SCCS-41, Syracuse Center for Computational Science, 1990.