An Overview of Scientific Computing

Lloyd Fosdick
Elizabeth Jessup

September 28, 1995

High Performance Scientific Computing
University of Colorado at Boulder

The following are members of
the HPSC Group of the Department of Computer Science
at the University of Colorado at Boulder:

Lloyd D. Fosdick
Elizabeth R. Jessup
Carolyn J. C. Schauble
Gitta O. Domik

Contents

Trademark Notice

- Convex, Convex Exemplar, are trademarks of Convex Computer Corporation.
- Cray, Cray-1, Cray Y-MP, Cray C90, are trademarks of Cray Research, Inc.
- DEC, DECstation, DECstation 5000, DEC 5000/240, DEC 10000-660 AXP, DEC PXG 3D Accelerator, DEC VAX 11/780, PXG, PXG Turbo+, VAX are trademarks of Digital Equipment Corporation.
- HP 9000/735 is a trademark of Hewlett-Packard Company.
- Intel i860, Intel iPSC/2, Intel iPSC/860, Intel Delta, Intel Paragon are trademarks of Intel Corporation.
- IBM ES/9000 IBM Power2-990, IBM RS/6000-350, IBM 9076 SP1 IBM 9076 SP2 are trademarks of International Business Machines Corporation.
- KSR-2 is a trademark of Kendall Square Research.
- MasPar MP-1, MasPar MP-2 are trademarks of MasPar Computer Corporation.
- MATLAB is a trademark of The MathWorks, Inc.
- MIPS, MIPS R3000, MIPS R3010 are trademarks of MIPS Computer Systems, Inc.
- Mosaic is a trademark of the National Center for Supercomputing Applications.
- Netscape is a trademark of Netscape Communications Corporation.
- NEC SX-A is a trademark of Nippon Electric Company.
- SGI Indigo, SGI Indigo R4000, SGI Challenge, SGI Crimson are trademarks of Silicon Graphics, Inc.
- Sun SPARC, Sun SPARCstation 10/40, are trademarks of Sun Microsystems, Inc.
- CM-5, Connection Machine are trademarks of Thinking Machines Corporation.
- Illiac IV is a trademark of the University of Illinois.

An Overview of Scientific Computing*

Lloyd Fosdick
Elizabeth Jessup

September 28, 1995

1 Introduction

The computer has become at once the microscope and the telescope of science. It enables us to model molecules in exquisite detail to learn the secrets of chemical reactions, to look into the future to forecast the weather, and to look back to a distant time at a young universe. It has become a critically important filter for those tools of science like high-energy accelerators, telescopes, and CAT scanners that generate large volumes of data which must be reduced, transformed, and arranged into a picture we can understand. And it has become the key instrument for the design of new products of our technology: gas turbines, aircraft and space structures, high-energy accelerators, and computers themselves.

The story of modern scientific computing begins with the opening of the computer era in the 1940s during World War II. The demands of war provided the motivation and money for the first developments in computer technology. The Automatic Sequence Calculator built by H. H. Aiken at Harvard, the relay computers by George Stibitz at Bell Telephone Laboratories, the Eniac

*This work has been supported by the National Science Foundation under an Educational Infrastructure grant, CDA-9017953. It has been produced by the HPSC Group, Department of Computer Science, University of Colorado, Boulder, CO 80309. Please direct comments or queries to Elizabeth Jessup at this address or e-mail `jessup@cs.colorado.edu`.

by John Mauchly and J. Presper Eckert at the Moore School of the University of Pennsylvania, the Edvac growing from the Eniac effort and inspired by ideas of John von Neumann were products of this time. They were used almost exclusively for numerical computations, including the production of mathematical tables, the solution of equations for the motion of projectiles, firing and bombing tables, modeling nuclear fission, and so forth. But it was not all numerical computing, the deciphering of codes by Alan Turing on the Colossus computers at Bletchley Park in England is an important example of non-numerical computing activities in this period. The machines of this early period operated at speeds ranging from about one arithmetic operation per second to about one hundred operations per second.

Immediate problems of war did not provide the only motivation for computer development in this early period. Goldstine, von Neumann, and others recognized the importance of computers for the study of very fundamental problems in mathematics and science. They pointed to the importance of computers for studying nonlinear phenomena, for providing "heuristic hints" to break a deadlock in the advance of fluid dynamics, and for attacking the problem of meteorological forecasting. And computers themselves motivated new kinds of investigations, including Turing's work on fundamental questions in logic and the solvability of problems, and von Neumann's on self-reproducing automata. The possibility of solving systems of equations far larger than had ever been done before raised new questions about the numerical accuracy of solutions which were investigated by Turing, Goldstine, von Neumann, Wilkinson and others. While great advances have been made, these questions, these problems remain. This is not a failure of the promise of the computer but a testament to the fundamental nature of these questions.

In the following sections we take a broad look at scientific computing today. The aim is to capture your interest, to stimulate you to read further, to investigate, and to bring your own talents and energy to this field.

2 Large-scale scientific problems

In 1987, William Graham, who was then the director of the Office of Science and Technology Policy, presented a five-year strategy for federally supported research and development on high-performance computing. Subsequently,

as a part of this strategy, a detailed plan for a Federal High Performance Computing Program (HPCP) was developed.[1] It provided a list of "grand challenge" problems: fundamental problems in science and engineering with potentially broad economic, political, or scientific impact, which could be advanced by applying high-performance computing resources. The grand challenge problems are now often cited as prototypes of the kinds of problems that demand the power of a supercomputer.

Here is a slightly shortened list of the grand challenge problems as from the 1989 report on the HPCP by the Office of Science and Technology Policy.

Prediction of weather, climate, and global change *The aim is to understand the coupled atmosphere-ocean biosphere system in enough detail to be able to make long-range predictions about its behavior. Applications include understanding carbon dioxide dynamics in the atmosphere, ozone depletion, and climatological perturbations due to man-made releases of chemicals or energy into one of the component systems.*

Challenges in materials science *High-performance computing provides invaluable assistance in improving our understanding of the atomic nature of materials. Many of these have an enormous impact on our national economy. Examples include semiconductors, such as silicon and gallium arsenide, and high-temperature superconductors, such as copper oxide ceramics.*

Semiconductor design *As intrinsically faster materials, such as gallium arsenide, are used for electronic switches, a fundamental understanding is required of how they operate and how to change their characteristics. Currently, it is possible to simulate electronic properties for simple regular systems, however, materials with defects and mixed atomic constituents are beyond present computing capabilities.*

Superconductivity *The discovery of high temperature superconductivity in 1986 has provided the potential for spectacular energy-efficient power transmission technologies, ultra sensitive instrumentation, and new devices. Massive computational power is required for a deeper understanding of high tem-*

[1] Approximately $800M was proposed for this program in 1993.

perature superconductivity, especially of how to form, stabilize, and use the materials that support it.

Structural biology *The aim of this work is to understand the mechanism of enzymatic catalysis, the recognition of nucleic acids by proteins, antibody/antigen binding, and other phenomena central to cell biology. Computationally intensive molecular dynamics simulations, and three-dimensional visualization of the molecular motions are essential to this work.*

Design of drugs *Predictions of the folded conformation of proteins and of RNA molecules by computer simulation is a useful, and sometimes primary, tool for drug design.*

Human genome *Comparison of normal and pathological molecular sequences is our most powerful method for understanding genomes and the molecular basis for disease. The combinatorial complexity posed by the exceptionally long sequence in the human genome puts such comparisons beyond the power of current computers.*

Quantum chromodynamics (QCD) *In high energy theoretical physics, computer simulations of QCD yield computations of the properties of strongly interacting elementary particles. This has led to the prediction of a new phase of matter, and computation of properties in the cores of the largest stars. Computer simulations of grand unified "theories of everything" are beyond current computer capabilities.*

Astronomy *The volumes of data generated by radio telescopes currently overwhelm the available computational resources. Greater computational power will significantly enhance their usefulness.*

Transportation *Substantial contributions can be made to vehicle performance through improved computer simulations. Examples include modeling of fluid dynamical behavior for three-dimensional fluid flow about complete aircraft geometries, flow inside turbines, and flow about ship hulls.*

Turbulence *Turbulence in fluid flows affects the stability and control, thermal characteristics, and fuel needs of virtually all aerospace vehicles. Understanding the fundamental physics of turbulence is requisite to reliably modeling flow turbulence for the performance analysis of vehicle configurations.*

Efficiency of combustion systems *To attain significant improvements in combustion efficiencies requires understanding the interplay between the flows of the various substances involved and the quantum chemistry that causes those substances to react. In some complicated cases the quantum chemistry required to understand the reactions is beyond the reach of current supercomputers.*

Enhanced oil and gas recovery *This challenge has two parts: to locate as much of the estimated 300 billion barrels of oil reserves in the US as possible and to devise economic ways of extracting as much of this oil as possible. Thus both improved seismic analysis techniques and improved understanding of fluid flow through geological structures are required.*

Computational ocean sciences *The objective is to develop a global ocean prediction model incorporating temperature, chemical composition, circulation and coupling to the atmosphere and other oceanographic features. This will couple to models of the atmosphere in the effort on global weather as well as having specific implications for physical oceanography.*

Speech *Speech research is aimed at providing communication with a computer based on spoken language. Automatic speech understanding by computer is a large modeling and search problem in which billions of computations are required to evaluate the many possibilities of what a person might have said.*

Vision *The challenge is to develop human-level visual capabilities for computers and robots. Machine vision requires image signal processing, texture and color modeling, geometric processing and reasoning, as well as object modeling. A competent vision system will likely involve the integration of all of these processes.*

Thus there is no shortage of problems for today's supercomputers. At a future time, when we have petaflop computers (10^{15} floating-point operations per second), we can be sure there will be no shortage of problems for them either. The solution of old problems raises new problems. That is the nature of science, its challenge and its mystery.

Most of the grand challenge problems involve modeling a physical system in a computer and using this model to create a simulation of its behavior. Others involve reduction and analysis of experimental data on a very large scale. Even the modeling problems involve data analysis and reduction on a large scale because running the simulation generates large files of data for analysis. Frequently, data analysis requires representation of the data in the form of pictures, graphs, and movies — a fascinating and rapidly growing activity known as scientific visualization. To get a closer look at a modeling problem we focus briefly on an important problem from the atmospheric sciences.

2.1.1 Computer simulation of the greenhouse effect

Global warming has been the subject of growing international attention. This problem is being studied by computer simulations that help us understand how changing concentrations of carbon dioxide in the atmosphere contribute to global warming through the greenhouse effect. A study of this type requires modeling the climate over a period of time. Studies by Washington and Bettge at the National Center for Atmospheric Research provide a typical example. A climate model known as the general circulation model (GCM) is used to study the warming which would be caused by doubling the concentration of carbon dioxide over a period of 20 years. The computations they describe were done on a Cray-1, now a relatively old computer, with a peak speed of about 200 Mflops.[2] The scientists report:

> *"The model running time was 110 seconds per simulated day. For two 19-year simulations, over 400 computational hours were required to complete the experiment."*

The effects the GCM attempts to take into account are illustrated in figure 1.

[2]Mflop is an abbreviation for a "megaflop," 10^6 floating-point operations per second.

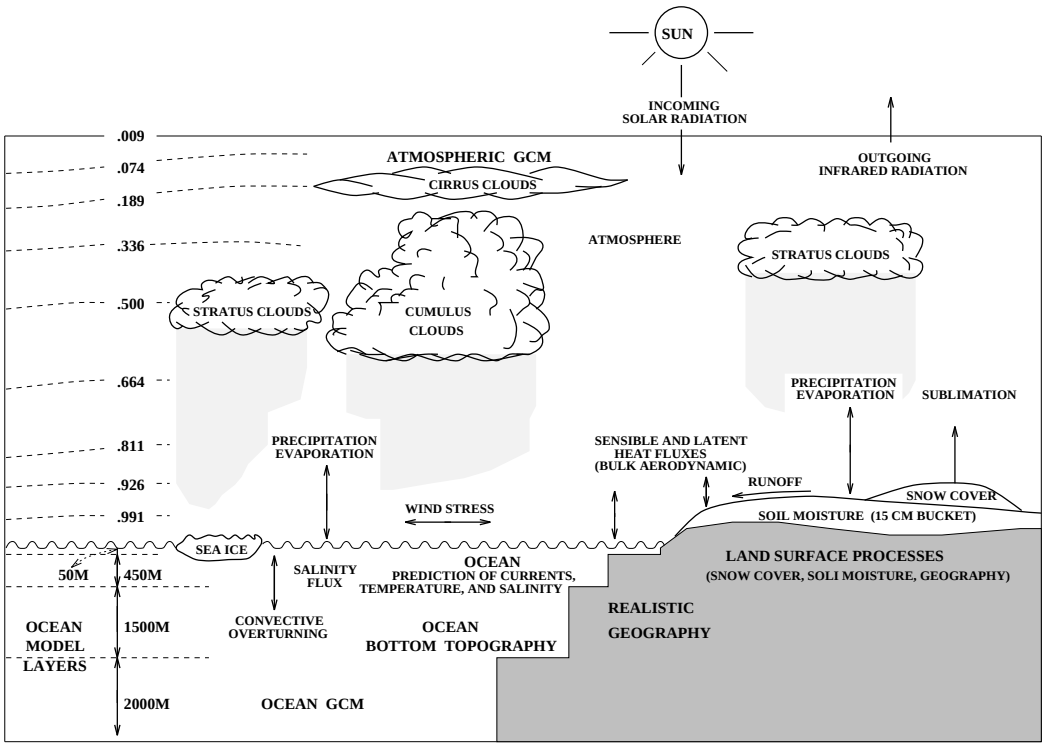

Figure 1: Schematic of the physical processes included in the GCM. This figure is adapted from the article by Washington and Bettge with their permission.

The atmosphere is a fluid and so the partial differential equations that govern the behavior of fluids are the mathematical basis of the GCM. Computer solution of these equations is done by a "finite difference" algorithm in which derivatives with respect to spatial coordinates and time are approximated by difference formulas in space and time. Thus a three-dimensional mesh in space is created, as illustrated in figure 2. Solution of the problem involves starting with some set of initial conditions, for which values are assigned to the variables at each mesh point, and stepping forward in time updating these variables at the end of each time step. There are some eight or nine variables at each mesh point that must be updated, including temperature, wind velocity, CO_2 concentration, and so forth.

The mesh is the key for understanding why speed is so important. The mesh used in the computations was three dimensional with about 2000 points

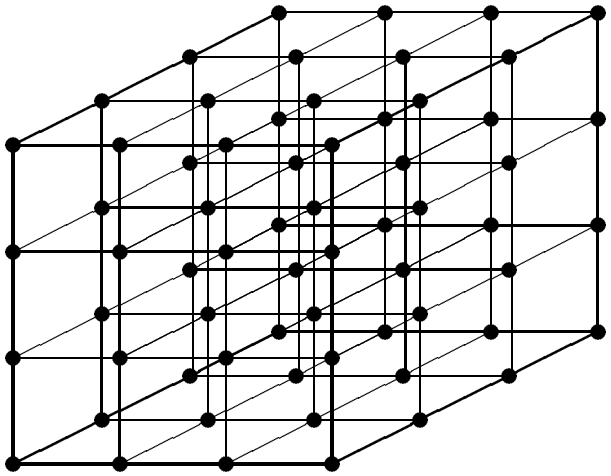

Figure 2: Illustration of a three-dimensional mesh.

to cover the surface of the earth and nine layers spaced at different altitudes: altogether about 18,000 mesh points. A moment's thought will make it apparent this is an extremely coarse mesh. The surface of the earth is 2.1×10^8 sq. mi.; i.e., $100,000$ sq. mi. per mesh point. Colorado with a land area of 103,595 sq. mi. rates one surface mesh point! Quite clearly we would like greater accuracy, and this means more mesh points. But if we double the density of points in each of the three directions we increase the number of mesh points by a factor of 8, essentially an order of magnitude increase in computational demand. And so this computation which took 400 hours for around 18,000 mesh points, would take over 3000 hours, and we would still have only 3 or 4 surface points for Colorado.

3 The scientific computing environment

The scientific computing environment consists of high-performance workstations, supercomputers, networks, a wide range of software, and technical literature. In this section you will find some pointers to this material.

High-performance workstations have peak speeds of 10 to over 100 Mflops;

Machine	**Manufacturer**	**LINPACK** (Mflops)
SUN SPARC10/40	SUN Microsystems	10.0
IBM RS/6000-350	IBM	19.0
IBM Power2-990	IBM	140.0
DEC 5000/240	Digital Equipment	5.3
HP 9000/735	Hewlett-Packard	41.0
SGI Indigo R4000	Silicon Graphics	12.0
SGI Crimson	Silicon Graphics	16.0

Table 1: A short list of high-performance workstations, and their performance on the LINPACK benchmark.

the supercomputers have peak speeds of 500 Mflops to over 100 Gflops.[3] High-resolution color monitors with over 10^6 pixels provide excellent tools for pictorially representing data from scientific simulations. In table 1 there is a short list of high-performance workstations with the name of the machine, followed by the name of the manufacturer, followed by the performance in Mflops on the LINPACK Benchmark. [Dongarra 94]. This benchmark is based on the speed of solving a system of 100 simultaneous linear equations using software from LINPACK [Dongarra et al 79]. The peak performance of workstations in this table could be as much as five to ten times the LINPACK performance number.

Table 2 provides a short list of supercomputers, with the name of the machine series, the name of the manufacturer, and the *Theoretical Peak Performance* [Dongarra 94].[4]

As with the workstations, the machines in this list come in various models and configurations; the performance data is for the largest system listed in Dongarra's report. Note here, in contrast with the previous table, a theoretical peak performance figure is given; in an actual computation the performance could fall to one-fifth or one-tenth the performance figure given

[3] Gflop is an abbreviation for a *gigaflop*, 10^9 floating-point operations per second (flops).

[4] Dongarra's definition of these values: "The theoretical peak performance is determined by counting the number of floating-point additions and multiplications (in full precision) which can be performed in a period of time, usually the cycle time of the machine."

Machine	**Manufacturer**	**Number of Processors**	**Theor Peak** (Gflops)
CM 5	Thinking Machines	16,384	2,000
Cray Y-MP	Cray Research	16	15
Cray T3D	Cray Research	2,048	307
IBM ES/9000	IBM	6	2.7
IBM 9076 SP2	IBM	128	32
Intel iPSC/860	Intel	128	7.7
Intel Paragon	Intel	4,000	300
KSR2	Kendall Square Research	5,000	400
MP-2	MasPar	16,384	2.4
NEC SX-A	Nippon Electric Company	4	22

Table 2: A short list of supercomputers and their theoretical peak performances.

here. Roughly speaking, the speed advantage of a supercomputer over a high-performance workstation is a factor of 1000. The price difference is roughly the same or slightly higher.

The National Science Foundation (NSF) supports five supercomputer centers available to scientists and students for research and education:

- Cornell Theory Center, Cornell University, Ithaca, NY.
- National Center for Atmospheric Research (NCAR), Boulder, CO.
- National Center for Supercomputer Applications (NCSA), University of Illinois, Champaign, IL.
- Pittsburgh Supercomputing Center, Carnegie Mellon University and the University of Pittsburgh, Pittsburgh, PA.
- San Diego Supercomputer Center, University of California at San Diego, San Diego, CA.

The facilities at these centers can be accessed via worldwide networks. Centers usually have facilities to accommodate on-site visitors, and they run workshops on a wide range of topics in scientific computing. In addition to

these NSF centers there are other supercomputer centers at universities and national laboratories which provide network access to their facilities. These include:

- Arctic Region Supercomputing Center, University of Alaska, Fairbanks, AK.
- Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN.
- Advanced Computing Research Facility, Argonne National Laboratory, Argonne, IL.
- Center for Computational Science, Oak Ridge National Laboratory, Oak Ridge, TN.
- Lawrence Livermore National Laboratories, Livermore, CA.
- Los Alamos National Laboratory, Los Alamos, NM.
- Massively Parallel Computing Research Center, Sandia National Laboratories, Albuquerque, NM.
- Maui High Performance Computing Center, Kihei, Maui, HI.
- Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA.

All of the supercomputers listed in table 2, except the NEC SX-A, are available at one of these centers.

Communication networks are a vital part of the supercomputing environment. The Internet is the worldwide system linking many smaller networks running the TCP/IP protocol. In May 1994 the Internet consisted of 31,000 networks, connecting over two million computers, and was growing at the phenomenal rate of one new network every 10 minutes [Leiner 94].

The National Science Foundation Network (NSFNET) is one of the most important components of the Internet, linking the the supercomputing centers in a network known as the backbone, which can be reached from other networks linking universities and other research organizations. Among these other networks are WESTNET in the Rocky Mountain states; NEARNET in the New England states; SURANET in the southern states; and MIDNET in the midwestern states.

The bandwidth of the NSFNET backbone has been 44.7 megabits/sec but a new backbone is under construction with a bandwidth of 155 megabits/sec. Communication networks with bandwidths in the gigabit/sec range are emerging. NSF and the Defense Advanced Research Projects Agency (DARPA) are supporting five testbed research projects on communication networks operating at gigabit/sec rates. Included in these research projects is a study of distributed computation on a very large scale: ocean and atmospheric climate models will simultaneously run on separate computers exchanging data across a network which includes the Los Alamos National Laboratory and the San Diego Supercomputer Center.

Software to support scientific computing is available on the Internet from the supercomputing centers and other sources. There is a particularly valuable resource for software known as *Netlib*. It is a library of numerical software available by e-mail or ftp from one of two centers in the U.S.A. A copy of machine performance information as shown in the two tables above, and short descriptions of these machines are also available from Netlib. Information about Netlib can be obtained via e-mail to

`netlib@research.att.com` or `netlib@ornl.gov`

The body of your mail message should contain just the line

```
help
```

Also, you can access Netlib directly with anonymous ftp. For Europe there is a duplicate collection of Netlib in Oslo, Norway with Internet address `netlib@nac.no`; and for the Pacific region there is a collection at the University of Wollongong, NSW, Australia with the following Internet address `netlib@draci.cs.uow.edu.au`.

There is a relatively new network tool called *Mosaic* now widely used for browsing and retrieving information on the Internet. With this tool the user can read documents located at another Internet site. These are hypertext documents so the user can navigate through them by clicking on highlighted keywords. High quality graphics images and animations can be included in these documents. Menus facilitate other operations including retrieving entire documents and programs. Mosaic was developed at NCSA and is available from them at no charge by anonymous ftp at `ftp.ncsa.uiuc.edu`. A commercial version known as *Netscape* was recently produced.

Another important resource is represented by technical publications: journals and conference proceedings. The following lists tends to focus on computer related publications, especially those concerning parallel computing and supercomputing, rather than applications. However an increasing number of articles concerning the application of computers to problems in physics, chemistry, and biology are finding there way into these publications. Some of the journals are:

- *ACM Transactions on Mathematical Software*
- *Computers in Physics*
- *Computer Physics Communications*
- *Computer Methods in Applied Mechanics and Engineering*
- *IEEE Transactions on Parallel and Distributed Systems*
- *International Journal of Parallel Programming*
- *International Journal of Supercomputer Applications*
- *Journal of Computational Physics*
- *The Journal of Supercomputing*
- *Methods in Computational Physics*
- *Parallel Computing*
- *SIAM Journal of Scientific Computing*
- *Supercomputing Review.*

Some of the regularly held conferences that issue proceedings are:

- Annual Symposium on Computer Architecture (IEEE Computer Society)
- Distributed Memory Computing Conference (IEEE Computer Society)
- Frontiers of Massively Parallel Computation (IEEE Computer Society)
- International Conference on Supercomputing (ACM)

Workstation	**Clock** MHz	**SPECmark**	**LINPACK** Mflops	**Theor Peak** Mflops
SUN SPARC10/40	40.0	60.2	10.0	40
IBM RS6000-350	41.6	74.2	19.0	84
IBM Power2-990	71.5	260.4	140.0	286
DEC 5000/240	40.0	35.8	5.3	40
HP 9000/735	99.0	167.9	41.0	200
SGI Indigo R4000	50.0	60.3	12.0	—
SGI Crimson	50.0	63.4	16.0	32

Table 3: Some typical workstations and performance data. The SPECmark value is the ratio of the average speed of floating-point operations for the given machine to the average speed of floating-point operations for a VAX 11/780 on a set of benchmark programs. The LINPACK value is the speed in solving a system of 100 linear equations in double precision arithmetic. Theor Peak stands for the theoretical peak performance.

- International Conference on Parallel Processing (ACM)
- International Parallel Processing Symposium (IEEE Computer Society)
- SIAM Conference on Parallel Processing for Scientific Computing (SIAM)
- Supercomputing (IEEE Computer Society)
- Supercomputing in Europe
- Visualization (IEEE Computer Society).

4 Workstations

The workstation is the desktop "supercomputer," small enough to fit on a desk but with peak speeds in the range of about 10 to 100 Mflops. Table 3 lists some popular workstations with performance figures. The actual speed of operation and the peak speed can differ substantially, as is illustrated by the data in this table: the speed on the LINPACK benchmark is substantially below the peak speed of these systems. Careful tuning of a program

is necessary to get close to peak speed, and for some computations this is simply an unreachable goal. In some cases, best performance requires programming parts of the computation in assembly language: the compilers for high-level programming languages cannot always produce the best results.

Normally, a workstation is connected to a network giving it access to additional computing resources which include other workstations, storage devices, printers, and still more powerful computing systems. Thus it serves the scientist as a primary computing resource and as a link to a wide array of other resources. The price of a workstation ranges from about \$15K to about \$100K. The high-end machines are faster and include high quality graphics, multiple processors, and large memories.

The architecture of scientific workstations has undergone a period of rapid development since the early 1980s when they first appeared. From that time to the present, clock rates have increased to about 200 MHz; pipelining of instructions and arithmetic, and parallel functional units have been introduced; and a variety of caching mechanisms have been developed to overcome memory access delays. In a ten year period the speed of these machines has increased by about two orders of magnitude. In the following paragraphs we discuss some of the architectural features now found in popular workstations.

4.1 RISC architecture

Many workstations use a RISC (Reduced Instruction Set Computer) architecture [Patterson 85]. It is characterized by a relatively small set of simple instructions, pipelined instruction execution, and cache memory. The principal goal of this architecture is an execution speed of one instruction per clock cycle: with a 40 MHz clock an execution speed of 40 mips[5] is the goal. In contrast, the acronym CISC is used for architectures with larger and more complex instruction sets: the DEC VAX 11/780 with about 256 instructions is a CISC system; the DEC 5000 with 64 instructions is a typical RISC system.

The move towards RISC systems was stimulated by recognition that better performance could be achieved with a simpler and smaller instruction set. Studies of programs executed on CISC machines showed that more complex

[5]mips is an abbreviation for "million instructions per second," 10^6 instructions per second.

Figure 3: Pipelined execution of instructions. The steps are: FI, fetch instruction; DI, decode instruction; FO, fetch operand; EX execute instruction; SR store result. Since this pipeline has five segments, it can be operating on five instructions simultaneously. While the FI segment is fetching an instruction, the DI segment is decoding the previous instruction, and so on down the pipe.

instructions were not heavily used. This was attributed to the observation that the more complex instructions were too specialized and not needed for many computations, also it was difficult for compilers to recognize when they could be used effectively. By reducing and simplifying the instruction set, instruction decoding time was reduced, and space on the chip was saved. This provided more space for cache and cache management. Particularly important was the fact that simple instructions, all taking about the same amount of time to execute, made it possible to execute instructions in a *pipeline*.

Using a pipeline makes it possible to execute one instruction per clock cycle. The idea of pipelined instruction execution is easy to understand by analogy to the automobile assembly line. In the automobile assembly line the work to be done is divided into a series of steps each requiring the same amount of time, say τ seconds, so the rate of production of autos is one auto per τ seconds. Similarly, in the instruction pipeline the work of executing an instruction is divided into steps as illustrated in figure 3. In this pipeline, which has five steps, we expect to gain a factor of five in speed over execution without pipelining.

The speedup promised by a pipeline cannot always be attained. For example, branch instructions can cause a problem because the next instruction after the branch is not known until the branch test is executed. Thus a branch interrupts the smooth flow of instructions through the pipe. Since branches occur frequently in code they could seriously degrade performance. To deal with this problem RISC systems use a *delayed branch* that delays fetching the next instruction after the branch by a fixed number of clock cycles. This causes a bubble in the pipe that can be filled by an instruction that would be executed regardless of the direction the branch takes. Whether or not the bubble can be filled depends on the program, and if it cannot, there will be a degradation of performance.

RISC systems usually have a separate floating-point pipelined coprocessor, and some RISC systems contain multiple functional units allowing overlap of operations: the IBM RS6000 series and the Intel i860 are examples. Therefore, with these systems it is possible to achieve a performance even higher than one instruction per clock cycle, often referred to as *superscalar* performance. For example, overlap of floating-point add and multiply in the IBM RS6000 system allows evaluation of $a \times b + c$ in one clock period, giving a peak speed of 50 Mflops with a 25 MHz clock. Moreover, in one clock cycle this system is capable of executing four instructions simultaneously: a branch, a condition-register instruction, a fixed-point instruction, and a floating-point instruction. Including the possibility of overlap of floating-point add and multiply, this system can execute five instructions per clock cycle.

Operation	Single	Double
add	0.75×10^{-7}	0.75×10^{-7}
subtract	0.75×10^{-7}	0.75×10^{-7}
multiply	1.01×10^{-7}	1.26×10^{-7}
divide	3.00×10^{-7}	4.79×10^{-7}

Table 4: Time for arithmetic on DEC 5000/240, units are seconds.

4.2 The DEC 5000 workstation

One system we use in the HPSC laboratory is a DEC 5000/240 series system with a 40 MHz clock for the CPU. The memory subsystem, I/O controller, etc. operate with a 25 MHz clock [DEC 91]. It uses a MIPS[6] processor and floating-point coprocessor. Measurements of the time for elementary arithmetic operations give the results shown in table 4. The system can have from 8 to 480 Mbytes of DRAM memory[7] with a bandwidth of 100 Mbytes/sec. Our systems are configured with 24 or 32 Mbytes.

[6] MIPS Computer Systems, Inc., Sunnyvale, CA.

[7] In this section, the following abbreviations are used: Mbytes for megabytes (10^6 bytes), Gbytes for gigabytes (10^9 bytes), and nsec for nanoseconds (10^{-9} seconds).

Operation	Single	Double
add	2	2
subtract	2	2
multiply	4	5
divide	12	19

Table 5: Cycle time for arithmetic in MIPS R3010 coprocessor

The processor subsystem has a 4 Gbytes virtual address space, 2 Gbytes of which are available to user processes. It has a 64 Kbyte cache memory for data and another 64 Kbyte cache memory for instructions. One word (32 bits) can be accessed from each cache in each processor cycle (25 nsec). A single memory read from a noncached address requires 690 nsec. Thus the time for a reading an operand from memory is about 28 times longer than reading it from cache.

The DEC 5000, as for most scientific workstations, can be augmented with 2D and 3D graphics options. The DEC PXG 3D Accelerator module includes an Intel i860 chip as a geometry engine and a scan converter chip to compute pixel values. The resolution of the display is 1280-by-1024 pixels. Double buffering and 24 image planes are provided with this module. The PXG system uses a 16 in. or 19 in. color Trinitron monitor. Peak speeds (for PXG Turbo+) are 436×10^3 vectors/sec and 106×10^3 polygons/sec: a "vector" is 10 pixels long; a "polygon" is a triangle, 100 pixels in area.

4.2.1 The MIPS R3000 and R3010 processors.

The CPU of the DEC 5000 is a MIPS R3000 processor and R3010 floating-point coprocessor. The user address space is 2 Gbytes. The coprocessor conforms to the ANSI/IEEE Standard for floating-point arithmetic.

The processor has 32 general purpose registers of 4 bytes each, and the coprocessor has 16 registers for floating-point numbers of 8 bytes each. Arithmetic instructions are register-register; e.g., add contents of register r1 to contents of register r2 and store result in register r3. Explicit move instructions move data between memory (cache) and registers in the processor or coprocessor, and between registers in the processor and coprocessor.

Times for arithmetic in the coprocessor are shown in table 5. Multiply and divide operations can be overlapped to some extent with other operations but not with each other; e.g., it is possible to execute a double precision floating-point addition and multiplication together in 5 cycles. For more information on these processors, see [Kane 88].

5 Supercomputers

The word "supercomputer" came into use in the late 1960s when radically new and powerful computers began to emerge from university and commercial laboratories. Since then it has symbolized the most powerful computers of the time.

Table 2 provides some information on the speeds of current supercomputers. The performance data given in the table provide only a rough estimate of performance. Performance depends strongly on the characteristics of the problem being solved, and hand tuning of the software can have a significant effect on performance. Because of the cost of supercomputers and the desire to solve increasingly large problems on them, the issue of squeezing the most out of these machines has received a lot of attention. New algorithms, tools, and techniques to improve their performance are constantly being developed, and research in programming languages to simplify writing effective programs is a continuing activity.

The high speed of supercomputers is the result of two factors: very fast logic elements and parallel architectures. Many people believe that the speed of logic elements is reaching the limit imposed by laws of physics and that parallel architectures are the key to supercomputers of the future. A parallel architecture allows many parts of a computation to be done simultaneously. This feature is what really distinguishes these computers from earlier machines of the computing era. It is also the reason why supercomputers pose such an interesting challenge to the algorithm designer and why their performance depends so strongly on the problem being solved.

5.1 Parallel architectures

The common notion of a *parallel computer* or *multiprocessor* is a collection of processors that are connected together in a manner that lets them work

together on a single problem, the idea is ten processors working together on a problem can get the job done ten times as fast as one of them working alone, and ten-thousand of them can get the job done ten-thousand times as fast, and so on. Of course, it doesn't happen quite that way.

Not all computational jobs can be divided up neatly into independent parcels in such a way as to keep all processors busy. Furthermore, one processor may need a result from another processor before it can proceed with its computation. Some jobs divide neatly while others do not; some require little interprocessor data communication while others require a lot. The following examples illustrate how hard it can be to keep the processor workloads balanced. In presenting these examples, we assume we are working with a *distributed-memory* multiprocessor. In this type of machine, a processor has direct access to its own memory only. Processors are required to send and receive messages to share data.

5.1.1 Evaluation of an integral

If the problem we have is to evaluate an integral, say

$$\int_a^b \int_c^d f(x,y)\,dx\,dy \quad ,$$

then an obvious way to proceed is to divide the domain of integration into n smaller rectangles, r_i, in such a way that the original problem is broken into subproblems, thus

$$\begin{aligned} \int_a^b \int_c^d f(x,y)\,dx\,dy \;=\; & \int\int_{r_1} f(x,y)\,dx\,dy \quad + & (1)\\ & \int\int_{r_2} f(x,y)\,dx\,dy \quad + & (2)\\ & \ldots + \quad \int\int_{r_n} f(x,y)\,dx\,dy \quad . & (3) \end{aligned}$$

The computation represented by each term on the right can be done by a different processor. After all of these computations have been done, the n results must be brought together to form the sum. At this point, communication is needed. Clearly we can reduce communication by reducing the number of rectangles covering the domain of integration, but then we reduce the parallelism in the computation.

If we divide the region of integration into equal subareas it might seem that we will keep all processors equally busy. Not necessarily so! The integrand may change its value very rapidly in some regions compared with others. To maintain accuracy we will need to use smaller integration steps in some regions than others. If we know this in advance, we can try to divide the work more or less evenly. If we don't know it, we can use an adaptive scheme to adjust the integration step during the computation. In the latter case, we could find that one processor is doing most of the work and a thousand others are idle. If we want to do some sharing of the work, we will need additional communication to find out which processors can accept work and to tell them what to do.

So in this relatively simple example we see in one circumstance, a relatively smooth integrand, the job divides up nicely into parcels and there is little communication. But when the integrand is not smooth there may be difficulty in dividing the work evenly, and there may be additional communication costs.

5.1.2 Molecular dynamics

Imagine we have n molecules interacting with each other as in a fluid. The forces between them are such that at very short distances of separation the molecules repel each other strongly. At intermediate distances, they attract each other. Beyond a certain distance, say several molecular diameters, they hardly interact at all. We are interested in tracing the motion of the molecules of the fluid. In a problem like this n may be in the hundreds of thousands or more.

The computation we must do can be described informally as follows. Newton's laws tell us the differential equations we must solve, numerical analysis gives us a choice of algorithms to use to solve these equations. We choose an algorithm and start its execution at time 0 with each molecule at a certain point in space, moving with a certain velocity. We compute the force on each molecule from all of the others within a specified distance, i.e., its neighbors. We then compute new values for the position and velocity of each molecule at time $t + \delta t$ and repeat the process to compute new values for the intermolecular forces, position, and velocity at time $t + 2\delta t$, and so on. In this way, we generate the solution stepwise in time until the final time of interest has been reached.

An obvious idea for doing the computation in parallel is to assign each processor to one or more molecules. The processor computes the new position and velocity of its molecules at each step and communicates them to the other processors. New positions must be computed for every molecule so the new forces between the neighboring molecules can be determined. Distributing the new values for all n molecules between all of the processors involves enormous amounts of communication. In message-passing multiprocessors, the time to communicate a floating-point number is much larger than the time to perform a floating-point operation, and the algorithm we've described will not be able to make efficient use of the machine without careful balancing of computation and communication.

To reduce the communication, we could have each processor communicate only with those processors holding molecules in the neighborhood of its own molecules. But determining what processors hold molecules in the neighborhood of a given molecule is itself a major problem. As the molecules move, the population of neighbors changes. To implement this new algorithm, each processor will have to keep track of its neighbors somehow or will have to institute a search at each step. Maintaining a list of neighbors would involve redundant, non-parallel computation, and a search would require extra interprocessor communication. Minimizing this overhead presents a difficult problem.

Yet another approach would be to assign processors to specific regions of space rather than to specific sets of molecules. However, it's not immediately obvious that this organization of the problem solves the difficulties encountered with the earlier ones. As each particle moves, we must keep track of the region and, hence, processor to which it is assigned. Again, we'll either need more computation or communication for bookkeeping purposes. Whatever the approach, organizing an efficient parallel computation for this problem is going to take some careful thought. It is a computation which can cause difficult load-balancing and communication.

5.1.3 Types of parallel computers

We can identify two distinct types of parallel computers according to whether they obey instructions asynchronously or synchronously. We can further classify them according how they communicate information: by sharing a common memory space or by sending messages to each other. And we can

classify them according to the physical nature of the interconnection network.

Parallel computers in which individual processors execute instructions asynchronously and send messages to each other are probably the easiest to understand at a logical level. These computers are labeled MIMD, which stands for *multiple instruction, multiple data* streams. The Intel computers iPSC/2, iPSC/860, and Paragon are of this type. Each processor executes its own private set of instructions. The Intel computers are also distributed-memory computers. Messages are passed between the processors by *send* and *receive* commands. There are mechanisms for causing the processor to enter a *wait state* in which it waits until receiving data from another processor. Programming these computers at a low level, that is, specifying the individual send and receive commands, is difficult and various languages have been developed attempting to simplify it: Linda, developed at Yale University by Gelernter [Carriero & Gelernter 89], and DINO, developed at the University of Colorado by Schnabel, Rosing, and Weaver [Rosing et al 91], are two examples. A group known as the High Performance Fortran Forum (HPFF) has been developing a version of Fortran, High Performance Fortran (HPF), which aims to support parallel programming (see [Koelbel et al 94]).

Parallel computers in which individual processors execute instructions asynchronously but share a common address space include machines like the Cray Y-MP and C-90, the IBM SP1 and SP2, the Silicon Graphics Challenge, and the Convex Exemplar. They are often referred to as *shared-memory computers*. These are also MIMD computers. They tend to be easier to program than the MIMD message passing computers referred to above, but they are not without their own set of difficulties. Clearly, if two processors are able to read and write into the same memory location some kind of synchronization is needed to insure the reads and writes are done in the correct sequence. Managing this synchronization adds to the complexity of programming these machines.

The other class of parallel computers in which processors operate synchronously is typified by the Maspar MP1 and MP2. In these computers there is a single sequence of instructions obeyed by all of the processors, each acting on its own data. They are labeled SIMD, standing for *single instruction, multiple data* streams. The first real parallel computer, the Illiac IV, finished in about 1970,[8] was in this class. A trivial example, matrix addition

[8]Now in the Computer Museum in Boston.

$C = A + B$, illustrates the nature of an SIMD computation. We store the matrices so that each element is on a separate processor. If we have n^2 processors connected to form a square mesh, we could map elements A_{ij} and B_{ij} to processor p_{ij} located in row i and column j of the mesh for $n \times n$ matrices A and B. With this distribution, a single add instruction performed in unison on all processors produces the sum. The result is left distributed one element per processor; in particular, processor p_{ij} holds element C_{ij}. Using this parallel algorithm, matrices of order 100 require only the time for one addition, not 10,000 additions.

SIMD machines do not share memory, rather they have a distributed memory, one memory module for each processor, and communication of data is by message passing. Like computation, message passing is generally done in parallel. For example, suppose we wish to compute the matrix product $C = AB$ on the square mesh with A_{ij} and B_{ij} initially in processor p_{ij}. As part of this computation, all processors in column j of the mesh must access all elements in column j of matrix B. Every processor in the square mesh has four neighbors (north, east, west, and south). One way to make sure all processors receive the proper elements of the matrix B is to have each processor send its element to its north neighbor. Each processor then receives an element from its south neighbor, performs the proper multiplication with its element of A, then passes the received element of B on to its north neighbor. During this data exchange, the processor at the top or northernmost position in the column passes the elements of B to the processor at the bottom or southernmost position in the column. In this way, the elements of column j of B can cycle through column j of the processor mesh, and all n columns of processors can cycle their elements in parallel.

This partial matrix multiplication algorithm description demonstrates one way in which the processors of a SIMD multiprocessor can cycle data in lock step. It is also possible to broadcast data from one processor to all others or to exchange data by other special algorithms which take advantage of the way processors are physically interconnected in a particular machine.

In addition to the square mesh mentioned above, interconnection patterns include buses, hypercube structures, two-dimensional square meshes, switches of various kinds, rings, and combinations of these mechanisms. Bus systems, where processors are attached to one central piece of hardware or *bus* for message passing, are limited to about thirty processors at most. The others can accommodate much larger numbers of processors.

As we approach the physical limits of how fast processors can operate, machine design has begun to concentrate on increasing the number of processors in a multiprocessor. Two conflicting factors limit this number. First, the performance of a message-passing computer is determined in part by the distance a message must travel from a given processor to any other processor in the machine. This suggests that, for efficient message passing, a multiprocessor should have each of its processors connected to every other processor in the machine. In such a machine, a message would always travel from the sending processor directly to the receiving processor without being transferred through intermediate processors. As we added more processors, however, this completely-connected machine would encounter the second limiting factor—the maximum number of processors possible is limited by the number of interprocessor connections (physical wires) required. A completely-connected machine with p processors requires $p-1$ connections per processor or a total of $p(p-1)/2$ connecting wires.

Other processor interconnection patterns are more amenable to increased processor number. In a ring interconnection, for instance, every processor is connected to only two others. But while the number of ring-connected processors can be almost arbitrarily large, processors diametrically opposite each other in an p-processor ring would have to pass messages through about $p/2$ processors to communicate. Some multiprocessors have used an interconnection scheme known as a hypercube: with p processors each has $\log_2 p$ interconnection wires and a message need pass through no more than $\log_2 p-1$ intermediate processors between its source and destination. This scheme has the advantage of a relatively short path length, measured in terms on the number of processor-processor connections that a message must traverse to reach its destination. However, it has the disadvantage that the number of wires connected to each processer increases with p. Some multiprocessors connect processors in a two-dimensional mesh: with p processors, assuming $p = m^2$, the mesh has m rows and m columns. In this scheme the number of wires connected to a processor is 4, assuming the edges at the top and bottom and left and right are joined, regardless of the value of p. Thus the number of wires connected to a processor does not grow with p as it does with a hypercube. On the other hand the path length grows as $m = \sqrt{p}$, thus messages in the mesh have longer paths than in a hypercube. However, the delay suggested by a longer path can be made almost insignificant by a technique called *wormhole routing*. In this technique a short header message

is sent first and it sets a switch in each processor along the path, enabling the body of the message to pass through without delay.

The notion of *scalability* is important in characterizing parallel computers. As we pointed out earlier we would like the speed of a computer to increase in proportion to the number of processors; that is, if we double the number of processors then we double the speed of the computer. In practice this isn't exactly true for a number of reasons, some of which have been described above. But if the computer is designed in such a way that its speed does increase, at least approximately, in proportion to the number of processors, and its complexity in terms of the number of interconnecting wires also increases in proportion to the number of processors, we say the computer is scalable. Our preceding discussion suggests that parallel computers in which the processors are organized in two-dimensional square meshes are scalable.

5.2 A virtual parallel computer

Over the years systems which allow the interconnection of a set of workstations in such a way as to enable them to act as a distributed memory multiprocessor have been developed. The most successful of these is a system developed by Dongarra and his coworkers, called PVM (Parallel Virtual Machine). In addition to the fact that this system enables anyone with workstations networked together, on an Ethernet say, to have a multiprocessor, it also enables them to make a single workstation look like a multiprocessor, insofar as programming is concerned.

PVM is not restricted to workstations, versions have been built for the Intel iPSC/860 and Paragon, Thinking Machine Corporation's CM-5, Cray computers, and Convex computers. Thus PVM can serve as a common language for a number of parallel computers and in this way improve the portability of parallel programs. Furthermore, it can serve as as a platform for a heterogeneous multiprocessor consisting of, say, a Paragon, a CM-5, and a group of workstations.

The PVM software and documentation can be obtained by anonymous ftp at `netlib.att.com`, and the PVM manual is now available as a book [Geist et al 95].

From the programmers point of view, PVM consists of a library of procedures for C programs or Fortran programs to support interprocessor communication by message passing. Experience with PVM and a number of related

works has led to an unofficial standard for message passing known as MPI (Message-Passing Interface), created by the Message Passing Interface Forum [Gropp et al 94]. It specifies the syntax and semantics for message-passing procedures callable from C or Fortran programs.

6 Further reading

This has been a very brief treatment of the important and rapidly developing field of scientific computing. The intent was, as we said earlier, to get you interested to study it further. Here are a few suggestions for further reading. Books by Hockney and Jesshope [Hockney & Jesshope 88], Almasi and Gottlieb [Almasi & Gottlieb 89], and Leighton [Leighton 92] discuss parallel computers, their architectures, and programming. For the broad history of computing see books by Williams [Williams 85] and Augarten [Augarten 84]. For the history of scientific computing see the book by Nash [Nash 90] The work on the greenhouse effect by Washington and Bettge is described in [Washington & Bettge 90]. The federal high-performance computing program is described in [HPCC 89]. A description of High Performance Fortran is in [Koelbel et al 94]. A simple introduction to the Internet is given in the book by Krol [Krol 92]; see also the August 1994 issue of the Communications of the ACM for a number of articles on the Internet and related topics.

More complete descriptions of the architecture, performance, and applications of supercomputers can be found in the following chapters of this book. There you will find further references to the topics touched on here.

References

[Almasi & Gottlieb 89] ALMASI, GEORGE S. AND ALLAN GOTTLIEB. [1989]. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1st edition.

[Augarten 84] AUGARTEN, STAN. [1984]. *Bit by Bit.* Ticknor & Fields, New York, NY.

[Carriero & Gelernter 89] CARRIERO, N. AND D. GELERNTER. [April 1989]. Linda in context. *Communications of the ACM*, 32(4):444–458.

[DEC 91] Digital Equipment Corporation, Palo Alto, CA 94301. [Dec 1991]. *DECstation 5000 Model 240: Technical Overview.*

[Dongarra 94] DONGARRA, J. J. [1994]. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Oak Ridge National Laboratory, Oak Ridge, TN 37831. `netlib` version as of November 1, 1994.

[Dongarra et al 79] DONGARRA, J. J., C. B. MOLER, J. R. BUNCH, AND G. W. STEWART. [1979]. *LINPACK User's Guide.* SIAM, Philadelphia, PA.

[Geist et al 95] GEIST, AL, ADAM BEGUELIN, JACK DONGARRA, WEICHING JIANG, ROBERT MANCHEK, AND VAIDY SUNDERAM. [1995]. *PVM – Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing.* Scientific and Engineering Computation. The MIT Press, Cambridge, MA.

[Gropp et al 94] GROPP, WILLIAM, EWING LUSK, AND ANTHONY SKJELLUM. [1994]. *Using MPI: Portable parallel programming with the message passing interface.* Scientific and engineering computation. The MIT Press, Cambridge, MA.

[Hockney & Jesshope 88] HOCKNEY, R. W. AND C. R. JESSHOPE. [1988]. *Parallel Computers 2.* Adam Hilger, Bristol.

[HPCC 89] Executive Office of the President, Office of Science and Technology, Washington, D.C. [Sep 1989]. *The Federal High Performance*

Computing Program. This is often referred to as the Bromley Report. The program is now called the High Performance Computing and Communications (HPCC) program.

[Kane 88] KANE, GERRY. [1988]. *MIPS RISC Architecture*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

[Koelbel et al 94] KOELBEL, CHARLES H., DAVID B. LOVEMAN, ROBERT S. SCHREIBER, JR. GUY L. STEELE, AND MARY E. ZOSEL. [1994]. *The High Performance Fortran Handbook*. Scientific and Engineering Computation. The MIT Press, Cambridge, MA.

[Krol 92] KROL, ED. [1992]. *The Whole Internet: User's Guide and Catalog*. O´Reilly & Associates, Inc., Sebastapol, CA.

[Leighton 92] LEIGHTON, F. THOMSON. [1992]. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Leiner 94] LEINER, BARRY M. [1994]. Internet technology. *Communications of the ACM*, 37(8):32.

[Nash 90] NASH, STEPHEN G., editor. [1990]. *A History of Scientific Computing*. ACM Press History Series. Addison-Wesley Publishing Company, Reading, MA.

[Patterson 85] PATTERSON, DAVID A. [Jan 1985]. Reduced instruction set computers. *Communications of the ACM*, 28(1):8–21.

[Rosing et al 91] ROSING, M., R. SCHNABEL, AND R. WEAVER. [1991]. The DINO parallel programming language. *Journal of Parallel and Distributed Computing*, 13:32–40.

[Washington & Bettge 90] WASHINGTON, WARREN M. AND THOMAS W. BETTGE. [May/Jun 1990]. Computer simulation of the greenhouse effect. *Computers in Physics*, pages 240–246.

[Williams 85] WILLIAMS, MICHAEL R. [1985]. *A History of Computing Technology*. Computational Mathematics. Prentice-Hall, Inc., Englewood Cliffs, NJ.