Supercomputers: Past and Future

S. Lennart Johnsson

YALEU/DCS/TR-778
March 1990

To appear in KOSMOS 1990.

Supercomputers: Past and Future

S. Lennart Johnsson

March 1990

Abstract

Progress in many fields of science and in engineering design is rapidly becomming critically dependent upon supercomputers. The management of very large data sets, including fast update and retrieval of information, is also becomming a very important function in many non-manufacturing businesses, such as the transportation, the securities, and financial industries, and in various parts of the government. The goal for the designers of the next generation supercomputers is a computer with a performance of a least a trillion operations per second, and a primary storage of a hundred Gbytes. Such computers will be massively parallel, and are expected to be commercially available by 1995.

1 A brief history

The term supercomputer refers to the most powerful computers available at any given time. It is a relative measure of performance in a field where a products lifetime based on price/performance is no more than three to five years. The performance of a Cray-1 computer, introduced in 1976 and often refered to as the first modern supercomputer, occupied a volume of about 6 - 7 m^3, required liquid nitrogen cooling with the cooling system occupying at least as much space as the computer itself, and carried a price tag of about $10 million in 1976. Today, a single chip[1] microprocessor that can be purchased for about $1,000 has half the processing capacity of the Cray-1, and a handfull of state-of-the-art memory chips also with a total price of about $1,000 can store as much information as the primary storage of the Cray-1 computer. Even though the comparison is unfair in terms of cost, since the quoted cost for the Cray-1 is a system cost including software, the reduction in volume, price, and power consumption over a 13 year period is more than a thousandfold.

Computers and communication systems are cornerstones of todays society. The banking, insurance, securities, transportation, real estate, and health care industries are examples of industries outside science and engineering that critically depend on modern computer and communciation systems. The success of the computer is its ability to store programs that allow the same piece of hardware to be used for a variety of tasks. Charles Babbage's Analytical Engine [36, 54] designed during 1833 – 1837 is considered the first *stored program* computer. It was designed to store 1000 words, with 50 digits per word. It was a mechanical computer, Figure 1. It was not until about 100 years later that new

[1] A chip is a piece of silicon less than 10×10 mm^2.

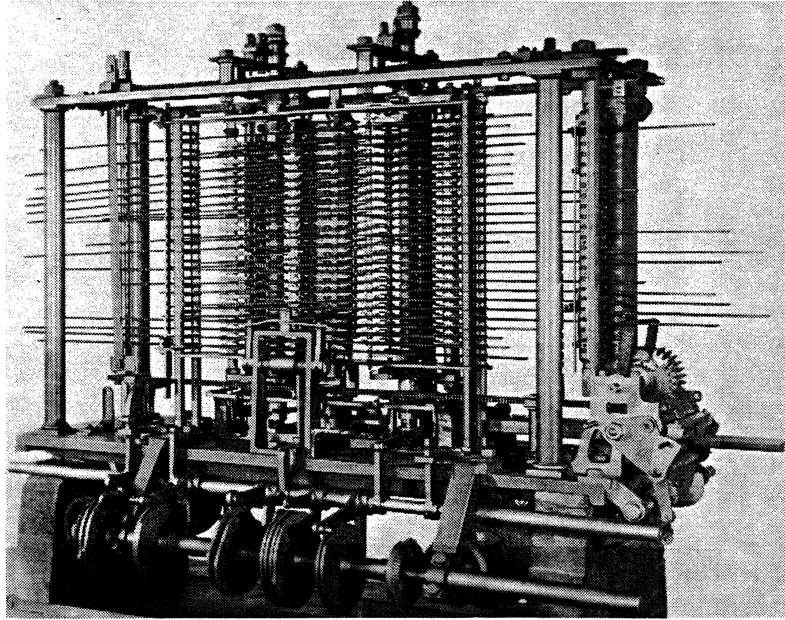

Figure 1: Charles Babbage's Analytical Engine.

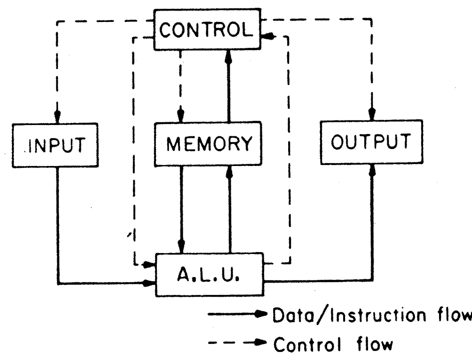

Figure 2: The von Neumann computer.

designs emerged. In the decade 1930 - 1940 electro-mechanical devices were used for the design of stored program computers at Bell Telephone Laboratories, Harvard University (MARK I), and a few other places (see for instance [4]). Non-programmable devices, such as calculators, existed long before the first stored program computer, and electro-mechanical devices for sorting were developed before the electro-mechanical computer. In 1946 Burks, Goldstine, and von Neumann [7] published a report on the design of an electronic stored program computer. The basic elements of the design was a memory unit for storage of data and instructions, a control unit, and an arithmetic unit as shown in Figure 2. Most modern computers are refinements of this architecture known as the von Neumann architecture. The path between the memory and the arithmetic and control units is often refered to as the "von Neumann bottleneck". Similar designs were being pursued by a few other groups in the USA and England.

The development of the stored program computer has been, and still is critically dependent upon the technology. It was not until electronic components became readily

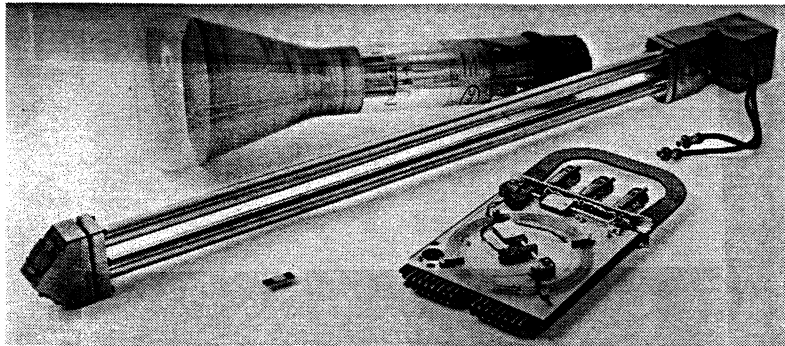

Figure 3: Memory devices in early computers.

available that the development of the modern computer started to accelerate. The three critical components in a digital computer are the logic devices used for the control unit and arithmetic/logic units, the storage, and the interconnections between the parts and the outside world. For the first 20 years of the electronic computer these elements were all manufactured in different technologies, and the task of the computer architect was to match the characteristics of these technologies with the needs. Vacuum tubes were used to create devices for logical operations for about a decade. The transistor, discovered at Bell Telephone Laboratories in 1947 became widely used in computers during 1955 – 1960. In 1966 the integrated circuit was introduced. Several devices and the interconnections between them could be manufactured in the same technology, and the space and power requirements were drastically reduced. Early storage devices were acoustic delay lines using mercury as the storage medium, or magneto-strictive effects in metals such as nickel, and electro-static storage using cathod ray tubes, Figure 3. Magnetic films in the form of drums were also used as early storage devices. Magnetic core store conceived independently by Rajchman at RCA and Forrester at MIT in 1949 [34] was introduced at about the same time as the transistor. By about 1970 core memory was replaced by storage based on transistors. Though core memory is obsolete, the term "core" is still often used for the primary storage of a computer. Today, MOS (Metal Oxide Semiconductor) technologies dominate in computer design. State-of-the art chips contain several million devices.

The increased level of integration has made possible a dramatic reduction in the volume of a sytem, its power consumption, and its cost, and equally dramatically increased the complexity measured in terms of gates, or elementary devices like transistors. The reliability is also vastly improved. The early computers consisted of a few thousand gates, had a storage of a few thousand words, occupied a room of significant size, and required an ample amount of power [28], Figure 4. A single chip of size say 5×5 mm^2 today can store about 100 times more information than the entire storage of the early computers, and costs about $50. Arithmetic operations such as addition and multiplication typically required a few milliseconds in early computers. A single chip, floating-point, processor designed in state-of-the-art MOS technologies can perform the same operation up to 100,000 times faster, and can be purchased for about $1,000. The Cray-1 introduced in 1976, Figure 5, had a primary storage of 8 Mbytes, and a peak floating-point capacity of about 160 Mflops/sec. Two state-of-the-art processor chips [40, 41] now have the same performance,

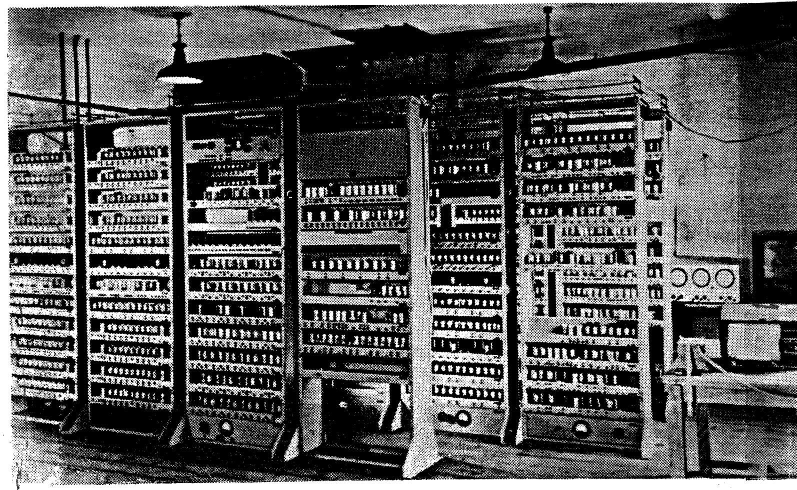

Figure 4: The EDSAC computer in 1949 with 3,000 valves and a 512 word storage.

Figure 5: The Cray-1 supercomputer with 8 Mbytes of storage.

and 16 state-of-the-art memory chips have the same storage capacity. The development is indeed dramatic. The supercomputer of a little more than a decade ago easily fits on a desk, and is affordable to almost any engineer and scientist. And it requires no liquid nitrogen cooling system in the basement.

Todays supercomputers have a peak performance of 5 – 10 billion floating-point operations per second (64-bit), or about 50 times the performance of a Cray-1. A hundredfold increase in performance is expected in the next several years. The trillion operations per second supercomputer is expected to be a commercial reality by 1995. Who needs a computer as powerful as all installed supercomputers today? Attempts to answer questions of this type has failed miserably in the computer field. In the early days of computing it was concluded that 10 to 20 machines, would satify all needs. A market study in the early days of electronic pocket calculators (about 1970) concluded that the entire market

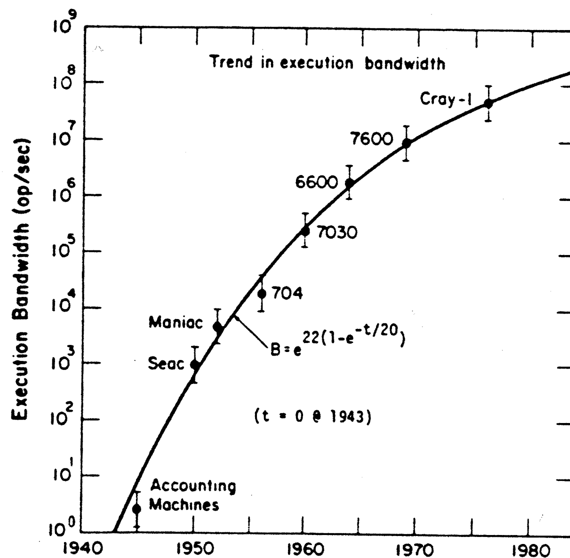

Figure 6: The evolution of peak computer performance.

would be a few thousand units. Both studies were useless at best. Our imagination to find useful applications for information processing devices once they are affordable to a large section of the population far exceeds our forecasting abilities.

2 Architecture - the past

Much of the increased performance in the past has come from technological innovations, like the transistor, the integrated circuit, and enhancements in the manufacturing technologies for such circuits. However, architectural innovations accounted for a significant portion of the performance enhancement during the first decade of the electronic computer, and is expected to account for most of the performance enhancement of supercomputers in the next decade. Figure 6 shows a plot of the growth in performance during the first 30 years of the electronic computer. During the first decade the performance increased by more than two orders of magnitude per decade, but during 1970 to 1980 the growth rate was less than one order of magnitude, and decreasing. Parallel architectures are now breaking the trend in Figure 6.

One of the most critical issues facing a designer of a high performance digital computer has been, and still very much is data motion. Different technological characteristics have accounted for this fact. In the first 5 - 10 years of the digital computer memory was in general faster than arithmetic units. Pipelining of logic operations was introduced to limit the number of logic stages a signal had to propagate through during a clock cycle. By diving an operation into several parts corresponding to logic stages, increased throughput is achieved when several identical operations are performed on different sets of data (operands). The rate at which operands can be accepted/delivered is determined by the slowest stage in the pipeline rather than the time through the entire pipeline. Pipelining of functional units is still widely used in high performance architectures. The

number of stages in arithmetic units is typically 5 ± 3. The performance enhancement is approximately proportional to the number of pipeline stages.

Pipelining was also introduced at a higher level. Memory operations, instruction decoding, and arithmetic logic operations were pipelined through prefetching techniques. Pipelining increases the required amount of logic somewhat, and was not commonplace in early designs where the number of components dominated the volume, the cost, and the power requirements. Today, pipelining is used at some level in most processor designs. The increased hardware complexity is minimal compared to the potential performance gain. The volume, cost and power requirements are dominated by the wires interconnecting the devices (transistors) [51, 32]. The main drawback of pipelining is in increased complexity of compilers, and/or programming.

Once the transistor, and in particular the integrated circuit, was used as elements of computers, it quickly became an easier task to design a fast processor than to design a fast memory unit. Today there exist several storage technologies, such as MOS circuits, magnetic discs and tapes, and optical storage. Each of these technologies differ quite significantly in price per bit, access time, and data transfer rate. In current high performance computers the memory system constitutes a hierarchy several levels deep, with registers being an integral part of a processor, and a fast but small memory called cache acting as a buffer between the registers and the primary storage of the computer. A small, fast memory, and a larger slow memory was used already in 1948 in the Mark I computer of Manchester University. The smaller storage was a random access electro-static storage (Cathod Ray Tube), and the larger storage electro-magnetic (drum). If most of the references during the execution of a program are made to the faster storage, then all of the storage appears to have the speed of the smaller storage. The Atlas computer, comissioned in 1962, was the first computer to make use of *paging* techniques to move blocks of instructions and data from a slow memory to a fast memory, thereby creating a *virtual storage* with the size of the larger storage operating effectively at the speed of the smaller storage for many applications. The success of this technique critically depends upon the placement of instructions and data on the slower memory device, and on the fetching mechanism being used.

Virtual memory techniques are indeed still key to the efficiency of modern computer systems. Slightly different techniques are used to manage the cache, compared to the virtual memory techniques used between primary and secondary storage. The differences are motivated by the differences in speed and sizes of the memories. Even though registers, cache, and primary storage may be manufactured in the same technology, they are designed to operate at different clock speeds. In MOS technologies it is feasible to design a small storage unit for a higher clock rate than a big storage unit. MOS technologies are charge transfer technologies [32]. Long wires require a large amount of power to drive the signals at a high speed from one end to the other. The on-chip wires are often determining the clock rate of many designs. Wire lengths are becoming more critical as the minimum feature sizes of the technology continue to decrease. Accounting only for capacitance,

Model	Opt. Delay	$L \rightarrow \frac{L}{\alpha}$		$L \rightarrow L$	
	τ_w	τ_w	$\frac{\tau_w}{\tau}$	τ_w	$\frac{\tau_w}{\tau}$
Capacitive	$const \cdot \tau \cdot log_e(const \cdot L)$	$\tau_w \rightarrow \tau_w/\alpha$	$\frac{\tau_w}{\tau} \rightarrow \frac{\tau_w}{\tau}$	$\tau_w \rightarrow \tau_w/\alpha(1 + \frac{log_e\alpha}{log_e(const \cdot L)})$	$\frac{\tau_w}{\tau} \rightarrow \frac{\tau_w}{\tau}(1 + \frac{log_e\alpha}{log_e(const \cdot L)})$
Resistive	$const \cdot \sqrt{\tau}L$	$\tau_w \rightarrow \tau_w/\sqrt{\alpha}$	$\frac{\tau_w}{\tau} \rightarrow \frac{\tau_w}{\tau}\sqrt{\alpha}$	$\tau_w \rightarrow \tau_w\sqrt{\alpha}$	$\frac{\tau_w}{\tau} \rightarrow \frac{\tau_w}{\tau}\alpha^{\frac{3}{2}}$

Table 1: Scaling of wire delays in MOS technology with optimized drivers.

the wire delay is reduced in proportion to the reduction of the device features when all dimensions are scaled, i.e., width, length, and thickness. However, if the length of the wire is not scaled, as for instance might be the case for a bus interconnecting more devices as the feature sizes are reduced, then the wire delay remains constant under scaling. Optimizing the driver for minimum delay, assuming minimum feature size logic on the input side, yields a delay τ_w proportional to the switching time τ of the technology and the logarithm of the length L of the wire [32]. The effect of scaling the design by a factor α is shown in Table 1. Accounting for wire resistance increases the wire delay, and degrades its behavior under scaling. The length of the clock cycle may have to be increased.

A characteristic often even more critical in the design of a high performance computer system in state-of-the-art technologies is the fact that the data transfer rate on a chip may be two orders of magnitude higher than the rate at which data can be transfered between a chip and its environment. Similarly, the rate at which information can be transfered between various units on a printed circuit board may be up to two orders of magnitude higher than the rate at which information can be transfered between boards. Locality of reference is critical also in state-of-the-art computer systems.

Another remedy for the relatively low speed of memory compared to processing units is to increase the number of storage units, or the width of the memory. This idea was used already on the Atlas computer, which had four memory banks. The CDC 6600 introduced in 1964 had 32 memory banks. For processors and memory units built in the same technology the difference in speed typically is less than a factor of 10, but if different technologies are used then the speed difference may be much higher. For instance, in the Cray-2 MOS technology is used for the primary memory, but the processors are designed in bi-polar technology operating at a clock rate of about 250 MHz. The computer has 256 memory units, or banks, for four processors. The processing capacity and the memory bandwidth are balanced with this degree of interleaving. The memory system was parallelised in a very early stage of the electronic computer.

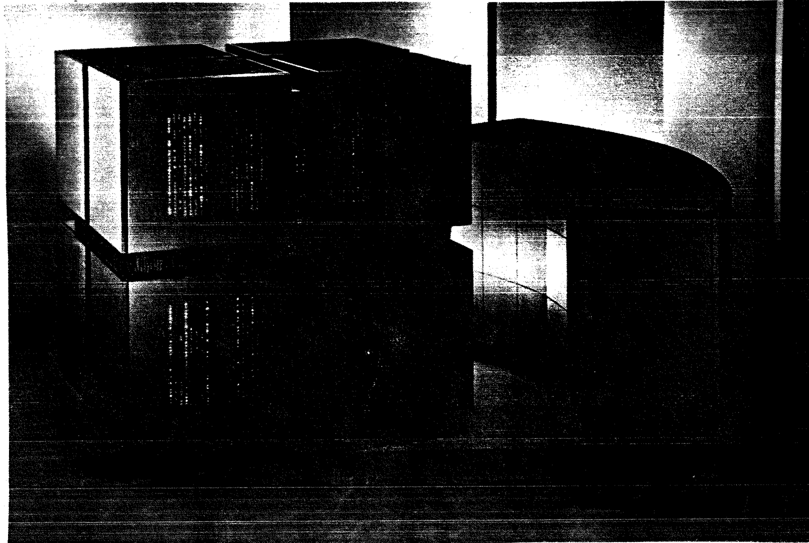

Figure 7: The Connection Machine with 2048 floating-point processors and 2 Gbytes of storage.

3 Supercomputers – parallel computers.

High performance computers have been designed to perform several operations concurrently through most of the history of computers. In general purpose computers parallelism was first accomplished by allowing different functional units to operate in parallel. For instance, the CDC 6600 had 10 functional units. Multiple units of the same type appeared in the Star-100 by CDC in 1973. Traditional supercomputer architectures, such as the Cray series of computers, are designed in the fastest (bi-polar) technology available at the time. Cray computers are probably the most carefully designed of any computer. In the next generation Cray computers (the Cray-3) light travels a about 4 *dm* during the targeted clock cycle. The signal propagation time has influenced all designs by Seymore Cray in a significant way from the CDC 7600 through the Cray-3. Data paths are very carefully layed out. But, as the bi-polar technology has been approaching fundamental speed limits parallelism has been employed to increase the performance. The most recent Cray computers have up to 8 processors, and 64 processor systems are being designed. Today, supercomputers are also constructed out of high volume MOS technologies using up to several thousand processors, such as the Connection Machine with 2048 floating-point processors and 2 Gbytes of primary storage, Figure 7.

The idea of large scale parallelism for general purpose computing was explored in the Illiac IV [20], the MPP [43], the Staran and the ICL DAP [18]. However, it is not until the last five or so years that the integrated circuit technologies have evolved to a level of integration where massively parallel, or data parallel architectures became a cost effective alternative for supercomputers. Integrated circuit technologies are replication technologies. The manufacturing cost per chip is very small, but the design cost is substantial. With complete processors on a single chip the cost per processor has fallen dramatically.

By 1995 the Tflop/s supercomputer is expected to be a commercial reality. At a

computational rate of a trillion operations per second, and a memory size of say 128 Gbytes, the operation code, the operand addresses, and the operands require 300 – 400 bits for a single instruction. The storage system at the register level must deliver 300–400 trillion bits per second, or about 8 million bits per cycle at a 50 MHz clock rate. Even with a memory hierarchy in the form of registers and cache the memory will have to be hundreds of thousands of bits wide. Assuming each processor can deliver two operations per clock cycle, or 100 Mflops/s, a system with a total of 40,000 processors will have a nominal capacity of four trillion floating-point instructions per second. With the required storage bandwidth, and with tens of thousands of processing units, a network is the only feasible alternative for passing data between processors and storage units in currently used technologies. A bus would have to be a hundred thousand wires wide, or more.

The feasibility of the Tflop/s supercomputer with respect to nominal performance and number of components is illustrated by the following calculations. With 2λ denoting the minimum feature size of the technology [32] a 64-bit RISC-like processor requires an estimated chip area of 30 $M\lambda^2$ [15, 16, 27, 31, 48, 47]. A floating-point unit requires at most 100 $M\lambda^2$. A preliminary design of routing circuitry for the Fluent supercomputer [45] suggests that 30 $M\lambda^2$ is a realistic estimate for circuitry that supports arbitrary communication in a network of processors. With 16 Mbits of memory per chip 40,000 chips contain 80 Gbytes of storage. Assuming 100 λ^2 per bit [47], the estimate for the total area for 16 Mbits of memory is 1600 $M\lambda^2$. With the floating-point processor, the communication circuitry, and 16 Mbits of memory integrated on the same chip a total area of 1800 $M\lambda^2$ is required per chip. In 0.5 μm technology the chip size is 10×10 mm^2. The nominal processing and memory capacity can be furnished by a number of chips that by experience can be made to work reliably in a system.

The data motion requirements for the Tflop/s computer mandates a massively parallel memory system, and a network for data motion between units. The speed of the technology (whether MOS or gallium arsenide) also forces massive parallelism to be employed for the processing subsystem. With 10^3 - 10^4 channels per side of a chip, the total data motion capacity of 40,000 chips is 100-1,000 TBytes/sec without sharing of on-chip channels between different data paths. But, assuming current standard packaging technologies of 100-300 pins per chip the data motion capacity at the chip boundary is about 10 TBytes/sec. At the board boundary, assuming about 500 pins, the data motion capacity for a 200 board system is about 0.16 TBytes/sec. The data transfer rate at the chip boundary is at least two orders of magnitude less than on the chip, and the transfer rate at the board boundary about two orders of magnitude less than on the board. The data delivery rate at the functional units required to sustain a Tflop/sec processing rate is about 40 TBytes/sec. Hence, with all references being on-chip the technology has the capacity to support the processing rate, but with no locality of reference the data motion capacity falls short by about three orders of magnitude. A substantial local memory is required for each processor, and the program must exhibit locality of reference to achieve a sustained performance close to the peak.

Computation	Registers only	4 Mbit chips	256 4 Mbit chips (board)	256 boards
Mtx mpy	0.5	104	1600	26000
3-d Relaxation	0.17	4.27	26.7	170.7
FFT	1	18.8	28.8	38.8

Table 2: Number of operations per remote reference of a single variable.

4 Locality of reference.

The reduction in required bandwidth is a function of the computation, the data allocation, and the size of the local storage. We demonstrate the potential reduction by considering three often used functions in scientific applications: matrix multiplication, nearest neighbor communication in three dimensional grids, as in 3-D relaxation, and butterfly based computations, as in the Fast Fourier Transform [9], and bitonic sort [3]. In 3-D relaxation on a regular lattice with k variables per lattice point, and two operations per variable the number of operations per remote reference is $\frac{1}{2d}(\frac{M}{k})^{\frac{1}{d}}$, where M is the size of the "local" storage. For $d = 3$ the number of operations per remote reference is $\frac{1}{6}(\frac{M}{k})^{\frac{1}{3}}$. In Tables 2 and 3 $k = 8$. For a Navier-Stokes code a more realistic value of k is 100-150 [39]. If the variables form matrices, then the number of arithmetic operations per variable is higher. Several linear algebra operations, including finite difference operators, and iterative equation solvers for partial differential equations, have a ratio of operations to remote references that follow the rule $\frac{1}{\alpha}(\frac{M}{\beta})^{\frac{1}{\gamma}}$ for suitable values of α, β and γ. For butterfly based algorithms, such as FFT and sorting, the dependence is of the form $\alpha \log(\frac{M}{\beta})$. For the FFT the ratio is $1.25log_2(M/2)$ real operations per remote reference using a radix-M algorithm, which is optimum [19].

Table 2 gives the ratio of local references to remote references as a function of local memory (columns one and two), the number of references local or remote with respect to a board, or the entire system. Optimum data allocation is assumed. Each chip has one processing unit and a board has 256 processing units. Table 3 gives the number of bits that have to cross the chip, board, and system boundaries during a single cycle, assuming optimum locality, or no locality of reference. The estimates are based on single precision variables.

Exploiting locality in the sample computations reduces the required communication bandwidth by a factor of up to 300 at the chip boundary, a factor of up to 7,500 at the board level, and by a factor of 160 – 10^5 at the I/O interface. To the extent these sample computations captures the essence of real applications a sustained performance of a Tflop/s is possible if locality is exploited, but only if that is the case. The value of exploiting locality is apparent, but the techniques for accomplishing this task are not.

Computation	4 Mbit, 1 proc. 1 chip	256 procs. Board	256 boards Machine
Mtx mpy	1	10	160
3-d relaxation	32	480	24600
FFT	3	1140	160000
no locality	300	76800	19660800

Table 3: Number of bits across the chip/board/system boundary per cycle.

5 Applications

Designing a computer for maximum performance requires careful attention both to technological realities and the computations to be performed. The demand for ultimate performance has traditionally first ocurred in the sciences, and in engineering. The evolution of technology is at the core of these fields, and the management of technological risks is part of everyday life, and of being a leader. The history of high performance computing is closley related to that of the computational sciences and engineering.

In several areas of science and engineering the cost and time involved in carrying out experiments have become limiting factors to rapid progress. In fundamental physics the cost of experimental facilities are now several billion dollars (the Supercollider). The cost of windtunnels for entire aircraft is prohibitive. Many experiments have been replaced by computer simulations even when experiments can be carried out. The experiments that still are carried out have the nature of final verification. Below we consider four applications: fluid dynamics, stress analysis, underwater acoustics, and lattice gauge physics. The prupose of the examples is to illustrate the data interaction that takes place in some typical large scale computations.

The success of pipelining depends heavily upon the ability to create a stream of similar operations, *vectorization*, and the success of storage hierarchies depend on locality of reference. A considerable improvement in software technology during the last 15 - 20 years has made the automatic vectorization of codes quite effective, and so called pageing algorithms and cache replacement algorithms [50] are so successful that programmers even for computationally very demanding applications rarely do their own memory management. These techniques are a necessity in future supercomputers, but locality of reference takes on new dimensions in a system consisting of a large number of processing units with their own memories interconnected by a network. Locality is a function of the network topology, and the data placement. In this section we review the nature of the locality of reference inherent in a few computationaly demanding applications.

5.1 Navier-Stokes compressible flow

One of the main application areas for supercomputers today is fluid dynamics, where computer simulations are replacing windtunnel experiments both for aircraft and automobile body design, as well as the design of jet and combustion engines. The accurate modeling of an entire aircraft and simulation of its aerodynamic properties would require at least 10^{15} floating-point operations. Todays supercomputers have contributed to a much shorter design cycle by replacing many windtunnel experiments with simulations, as well as to an improved design by allowing designers to explore more alternatives than would otherwise be possible. Another area of fluid dynamics with great computational demands is weather forecasting, and areas of growing concern such as ocean and atmospheric modeling for the study of pollution and global warming. These areas require fairly high geometrical resolution as well as accurate models of the chemistry involved. These problems are barely tractable for todays supercomputers.

The prototypical fluid dynamics problem is the solution of Navier-Stokes equations, which describe the balance of mass, linear momentum and energy. It models the turbulent phenomena that occur in viscous flow. In three dimensions the equations are of the form

$$\frac{\partial \mathbf{q}}{\partial \tau} = \frac{\partial \mathbf{F} + \mathbf{F}_\nu}{\partial \xi} + \frac{\partial \mathbf{G} + \mathbf{G}_\nu}{\partial \eta} + \frac{\partial \mathbf{H} + \mathbf{H}_\nu}{\partial \zeta}, \tag{1}$$

where the variable vector $\mathbf{q}(\xi, \eta, \zeta, \tau)$ has five components: one for density, three for the linear momentum in the three coordinate directions x, y and z, and one component for the total energy. The coordinates of the physical domain is x, y and z, whereas ξ, η and ζ are coordinates in the computational domain. $\mathbf{F}, \mathbf{G}$ and $\mathbf{H}$ are the flux vectors and $\mathbf{F}_\nu, \mathbf{G}_\nu$ and $\mathbf{H}_\nu$ are the viscous flux vectors. The exact form of these functions is beyond the scope of this article. Suffice it to mention here that they are functions of the vector $\mathbf{q}$, the transformation between the physical and computational domains, and the derivatives of this transformation (for details see for instance [39]).

For regular domains the solution to the Navier-Stokes equations can be approximated by computing the solution in points of the domain forming a three-dimensional lattice. The spacing between these points along the normal to solid walls often need to be much smaller closer to the boundary than in the interior, in order to compute the flow in the boundary layer with sufficient accuracy. A stretched grid is one way to accomplish this task. Such a grid is topologically equivalent to a regular grid. In solving the Navier-Stokes equations by a finite difference method, the partial derivatives are approximated by differences between computed values in neighboring lattice points. For a first order accurate approximation, values from two neighboring lattice points suffice. The higher the accuracy of the approximation the larger the number of points involved in the approximation. A few typical stencils in three dimensions are shown in Figure 8. Figure 9 shows a stretched grid for the computation of the channel flow illustrated in Figure 10.

For many solution methods and flows the stencils may need to be dependent on location, upon the variable subject to differentiation, and time. In the case of the Navier-

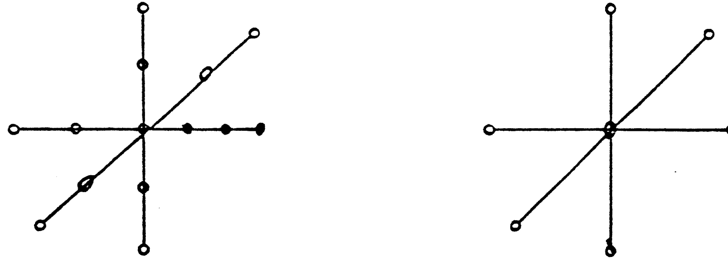

Figure 8: Samples of difference stencils.

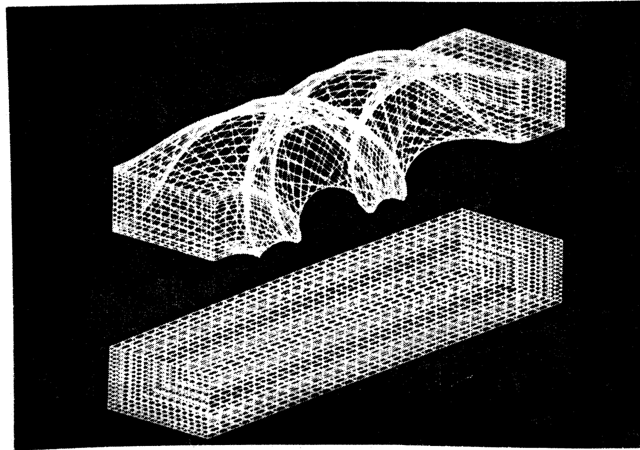

Figure 9: A stretched grid for channel flow calculations.

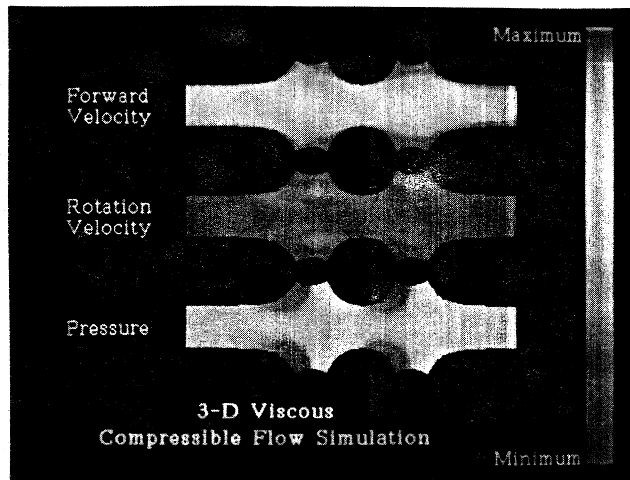

Figure 10: Forward and rotation velocities, and pressure for viscous flow in a channel.

Stokes equations artificial viscosity needs to be introduced to stabilize explicit numeric methods. The artificial viscosity is introduced through a fourth order derivative, and the difference stencil for this term includes five points in each dimension, centered at the interior point for which the evaluation is desired. Hence, in the case of the Navier-Stokes equations under very simple conditions there are two difference stencils being used in each lattice point, and the stencils vary for interior points, points on or close to a boundary surface, edge, and corner. Each difference stencil defines a combining operation on a set of points. All stencils for all points can be applied concurrently.

5.2 The Finite Element Method

In the finite element method [56] the solution to a set of partial differential equations in a domain is approximated by polynomial solutions over subdomains called elements. The most typical shape of the elements in two dimensions are triangles and rectangles. In three dimensions brick, prism, and tetrahedral elements are common. The order of the element determines the order of the polynomial approximation. Each polynomial is specified by the value of the polynomial itself, or its derivatives, in a number of points consistent with its order. These points are the nodal points of the elements. The solution is computed for the nodal points, each of which has a polynomial associated with it. The desired solution is expressed as a linear combination of the polynomials. The displacements of the nodal points as a function of the applied forces are obtained as the solution to a set of equations defined by a *stiffness matrix*. Evaluation of the elements of the stiffness matrix requires that products of the polynomials be integrated over the element. The numerical integration, or quadrature, is performed by evaluating the product at a number of locations on the element, and computing a weighted sum of these values.

There is a very high degree of concurrency in the computation of the elemental stiffness matrices. Large problems contain millions of nodal points. Much of the computations can be performed for each nodal point with a limited amount of communication [25]. For the solution of the equilibrium equations the elemental stiffness matrices are often assembled into a global stiffness matrix by introducing a global node ordering. For the elemental stiffness matrices an ordering local to an element will suffice. If the matrix is assembled, then every node couples to every node on every element that shares the node. For second order elements in the form of bricks four different stencils describe the interaction between nodal points. The number of points in the stencils are 27, 45, 75, and 125 respectively. The stencils are considerably more complex than in a finite difference method of the same order. Figure 11 illustrates the stencils in two dimensions, and Figure 12 the stresses in a wrench subject to bending.

Most of the data interaction in the finite element method occurs in the solution of the equilibrium equations. For an iterative method the communication requirements are defined by the stencils at the nodal points. A direct method requires a global partial ordering of all nodal points. Unless the union of the stencils for all nodal points define a

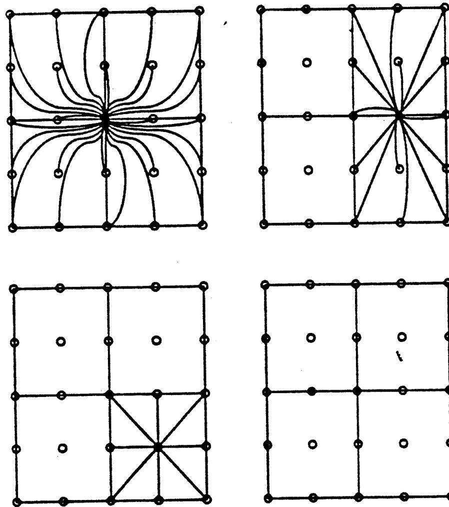

Figure 11: Second order stencils for rectangular finite elements.

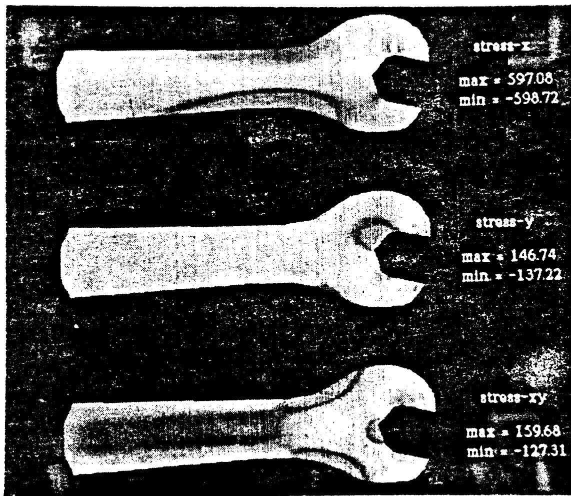

Figure 12: Stresses in a wrench subject to bending.

perfect elimination graph fill-in will occur in the elimination. The communication pattern is in general quite complex.

5.3 Acoustic Field computation by an Alternating Direction Method

A problem of considerable interest to the Navy is the modeling of sound propagation in the ocean. With the desired resolution this problem requires in the order of 10^{16} floating-point operations. The forward propagation of acoustic waves by the so called Wide Angle Wave Equation [29] implies the solution of an equation of the form

$$(1+\frac{1}{4}(1-\delta)X)(1-\frac{1}{4}Y)u(r+\Delta r)=(1+\frac{1}{4}(1+\delta)X)(1+\frac{1}{4}Y)u(r) \qquad (2)$$

where k_0 is a reference wave number, and $n(r,\theta,z)=k(r,\theta,z)/k_0$ $\delta=ik_0\Delta r$, and

$$X=\frac{1}{k_0^2}\frac{\partial^2}{\partial z^2}+(n^2(r,\theta,z)-1), \quad \text{and} \quad Y=\frac{1}{k_0^2r^2}\frac{\partial^2}{\partial\theta^2}.$$

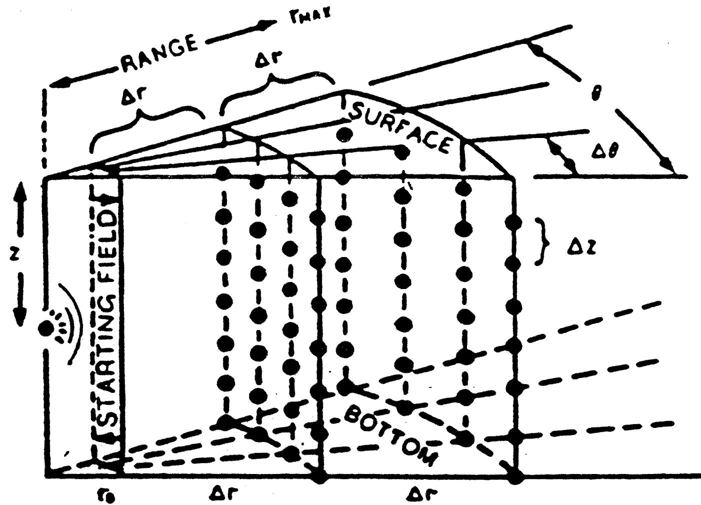

Figure 13: The grid obtained through the marching algorithm for underwater acoustics.

This equation is a parabolic approximation of the Helmholtz equation. The solution to the equation above can be marched out in the range direction r with an Alternating Direction Method [46, 26] Figure 13. Tridiagonal matrix-vector multiplications are performed in the θ and z directions, followed by the solution of tridiagonal systems in the same directions. Both operations consist of a number of one-dimensional problems that can be solved independently, and concurrently. Each system can be solved concurrently by substructuring, pipelined Gaussian elimination, partial or complete transposition of equations, and odd-even cyclic reduction, or any combination thereof [23] (which for multiple systems may be performed as balanced cyclic reduction). The communication pattern (in one dimension) of odd-even cyclic reduction is given in Figure 14, and of balanced cyclic reduction in Figure 15. The communication topology of balanced cyclic reduction is known as a *data manipulator* network, or a *PM2I* [49] network.

Hence, in the case of the underwater acoustics problem communication as defined by the difference stencil is required for matrix-vector multiplication, but for the solution of the systems of tridiagonal systems of equations the communication depends on the selected algorithm: for pipelined Gaussian elimination communication in the form of a Hamiltonian path is required, for equation transposition every processor communicates with every other processor, for balanced cyclic reduction communication is required in the form of a data manipulator network.

5.4 Lattice Gauge Physics

In the study of the fundamentals of matter the interaction between the elements of an atom, such as fermions, gluons, and quarks, is intensely studied in a formulation known as quantum chromodynamics. This problem is computationally very demanding. One research group consisting of collaborators from Los Alamos National Laboratories, Caltech, Argonne National Laboratories, and Thinking Machines Corp. are just beginning to see some new results after having performed computations equivalent to 100 trillion (10^{14}) floating-point operations. It is estimated that several orders of magnitude more compu-

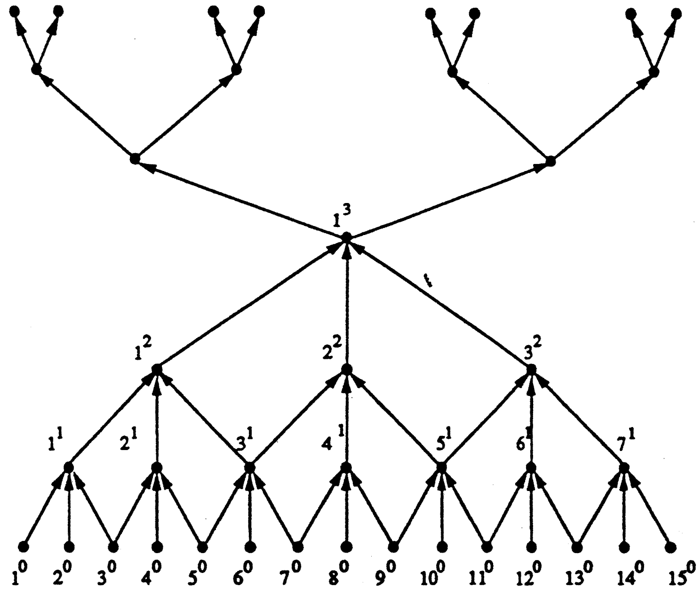

Figure 14: The communication topology of odd-even cyclic reduction.

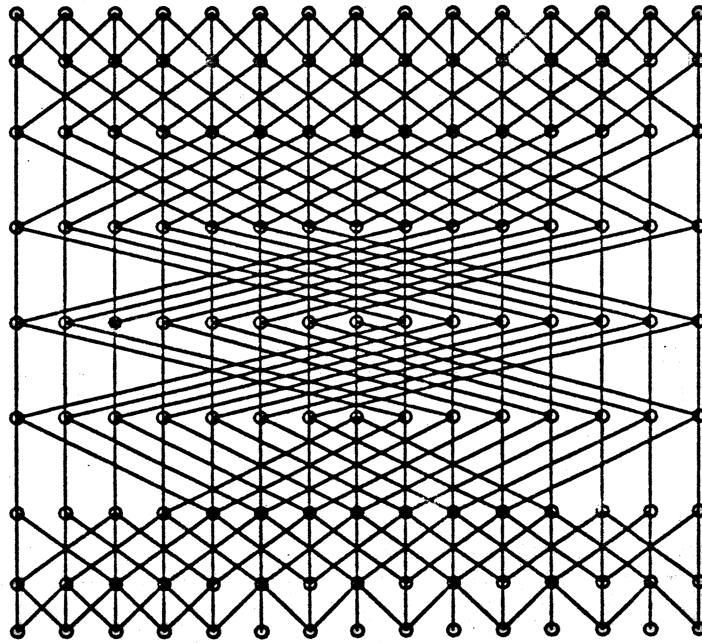

Figure 15: The communication topology of balanced cyclic reduction.

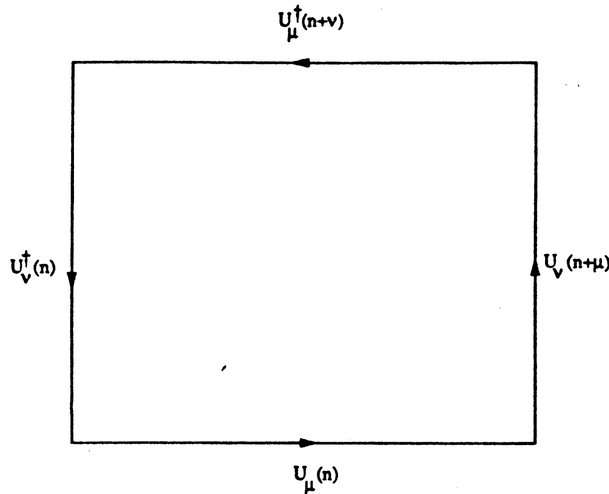

Figure 16: The plaquette calculations

tations need to be expended for a thorough understanding of this particular problem.

Quantum chromo-dynamics computations are based on four-dimensional lattices. The simplest formulation is the original due to Wilson [55], in which the action S for the gauge fields U is local, involving only the product of the gauge fields around elementary squares, called plaquettes, on the lattice. The action S is expressed as

$$S(U) = \beta E(U) = \beta \sum_{p} \left(1 - \frac{1}{N} ReTrU_p\right)$$

with

$$U_p = U_\mu(n) U_\nu(n+\hat{\mu}) U^{\dagger}_{\mu}(n+\hat{\nu}) U^{\dagger}_{\nu}(n),$$

and is illustrated in Figure 16.

The gauge fields $U_\mu(n)$ are represented by 3×3 complex matrices known as $SU(3)$. A matrix is associated with each link in the four dimensional lattice. $U_\mu(n)$ represents a link in the direction $\hat{\mu}$, i.e., from n to $n+\hat{\mu}$. $U^{\dagger}_{\mu}(n)$ is used if the link is traversed in the opposite direction. The parameter β determines the interaction strength, or "temperature" of the theory. The constant N is the dimensionality of the group (3 for QCD).

There are two classes of algorithms dominating the simulation of lattice gauge theories: stochastic and deterministic. The most popular stochastic algorithms are based on the Metropolis algorithm [35]. Deterministic algorithms are usually of the microcanonical type [8, 42]. The Monte Carlo algorithm changes the energy of a system while keeping its temperature constant, whereas the microcanonical algorithm conserves the total energy while allowing the temperature to vary. The two approaches may be combined as in [2], where a Monte Carlo method is used to bring the lattice gauge theory into equilibrium at a specified temperature (coupling), then a microcanonical algorithm is used to evolve the system for measurements of its properties. The microcanonical algorithm is computationally less demanding. During the latter phase it may be required to periodically switch back to a Monte Carlo algorithm in order to obtain ergodicity [11].

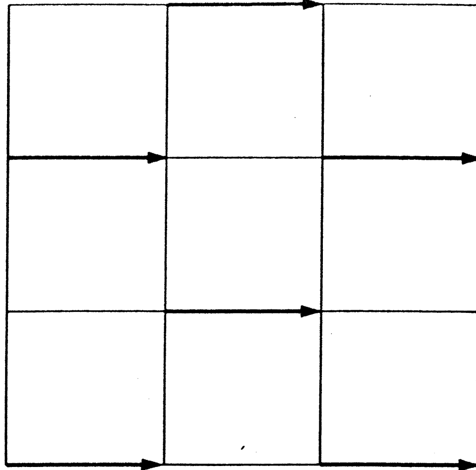

Figure 17: Link updates in parallel Monte Carlo lattice gauge theories.

Monte Carlo algorithms [35] cycle through all the gauge field links of the lattice changing their values by a random procedure until they settle down into physically correct configurations, C. These are such, that when statistical equilibrium is reached, the probability of finding any one of them is proportional to its Boltzmann factor $e^{-S(C)}$, where S is the action of the gauge theory. A sufficient condition for the statistical equilibrium to be attained is that, at each step of the Monte Carlo algorithm, the probability of changing a configuration C into a new one C' is the same as the probability of changing C' back to C. This state is called "detailed balance". In order to preserve detailed balance one cannot simultaneously update gauge field links which interact with one another. As the action involves interactions around plaquettes, one can therefore update only half the links in any one dimension simultaneously and preserve detailed balance, Figure 17. On a parallel computer full processor utilization is obtained by observing that there are two plaquettes to be calculated for each dimension and link update, and scheduling half of the processors to calculate the "positive plaquettes" and half to calculate the "negative plaquettes".

The edges of the four-dimensional lattice are directed, and the values associated with the edges, or bonds, can be stored at for instance the node at the tail end of the edge. The main quantity being computed for each bond in a step of the computation is its contribution to the total action, which involves all bond values of the plaquettes of which the given bond is a part. A bond is part of six plaquettes in four dimensions. With the bond values stored at the tail end of the directed edges, the stencil defining the communication in any plane is given in Figure 18.

5.5 Linear Algebra

We have already mentioned several techniques in numerical linear algebra, such as Gaussian elimination, odd-even cyclic reduction, balanced cyclic reduction, and iterative techniques like the conjugate gradient method. The multigrid method [5] is a technique for solving partial differential equations based on a sequence of grid refinements. Interpola-

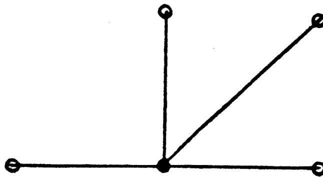

Figure 18: Communication stencil in a plane for quantum electro- and chromo-dynamics computations.

tion is performed in the transition from a grid to the points of the one level refined grid, and a smoothing operation is performed in going from a grid to the next level coarser grid. The idea is that slow variations are resolved on coarse grids and fast variations on fine grids. The communication in each dimension is similar to the case of odd-even cyclic reduction in that every other grid point is excluded in moving to the next coarser grid (and the communication distance doubles in the physical domain). With smoothing by relaxation and a five-point stencil, the communication required for this operation in the multigrid computation is indeed the same as for odd-even cyclic reduction. It is also possible to formulate the multigrid algorithm such that instead of a single coarser grid multiple coarse grids are used. All grid points are considered at every level [13]. In this case, the communication in one dimension is similar to the communication of balanced cyclic reduction.

Dongarra and Sorensen [10] have suggested a parallel algorithm for computing eigenvalues of tridiagonal systems by a divide-and-conquer method that is fully parallel. The algorithm proceeds by tearing the tridiagonal system into two smaller tridiagonal systems of approximately equal sizes, recursively. The computations start from the bottom level of the recursion by computing eigenvalues for a large number of small tridiagonal systems. Then, the tridiagonal systems are joined pairwise in the next step, such that eigenvalues are computed on half as many systems of approximately twice the size. The eigenvalues are computed as the roots of equations of the type

$$1 + \rho \sum_{j=1}^{n} \frac{\zeta_j^2}{\delta_j - \lambda} = 0$$

where n is the number of equations in a system. With the components of ζ and δ distributed the computations can either be organized with reduction and copy operations within segments representing the independent systems, or by all-to-all broadcasting within segments and concurrent computation in every processor at every step of the computation. The data structure for the problem is preferably organized as a dynamic two-dimensional array of elementary objects. During the course of computation the array is reshaped from an array with few rows and many columns, to an array with a single column [6].

6 Data motion in distributed memory systems

In the above examples for the computation of approximations to solutions of partial differential equations describing the fluid flow, the stresses and displacements in a solid under pressure, the acoustic field, or the internals of matter, the discretization of the domain was regular. A two-dimensional lattice was used for the acoustics problem, a three-dimensional stretched grid for the Navier-Stokes equations, a deformed three-dimensional lattice for the stress analysis problem, and a four dimensional lattice for lattice gauge physics. The required communication for each processor was defined by difference stencils in the Navier-Stokes equations solved by an explicit method, and for the right-hand side (matrix-vector multiplication) in the acoustics problem. Solving the equilibrium equations by a finite element method and an iterative solver also implies communication according to a local stencil (defined by the elements). The lattice gauge physics example used only local communication that could be modeled by a fairly simple stencil (11-points). In this example a single stencil suffices for the entire domain, in part due to periodic boundary conditions. Other boundary conditions cause a proliferation in the number of stencils that are required, as was the case in the Navier-Stokes example, and not all interior points may be equivalent, as is the case in the finite element method with higher order elements. The stencils define combining operations in the form of a +-reduction on weighted variables.

Direct equation solvers require long range communication in the physical space. If the problem is of full rank, then global communication is required for the solution [14]. The iterative solvers accomplish this task in the iterative process. The balanced cyclic reduction algorithm require communication corresponding to a *PM2I* network, and odd-even cyclic reduction requires commuication according to a subtree of this network. Conventional multigrid algorithms requires communication similar to odd-even cyclic reduction, and the super-convergent multigrid algorithm requires communication similar to that for balanced cyclic reduction.

Algorithms making use of transposition of systems of tridiagonal equations, as might be used in the acoustics problem, may use communication in the form of butterfly networks, the ideal communications network for the Fast Fourier Transform. Many other functions, such as sorting and permutations can also be performed well on a butterfly network, Figure 19.

In addition to data motion as determined by computational algorithms there are also a few well defined permutations that occur often, either in the context of the above mentioned algorithms, or as data rearrangements between computations to improve load balance, or in order to minimize communication during a particular computation. Familiar permutations are matrix transposition and bit-reversal. These permutations with the rows and columns being powers of two are examples of a class of permutations called dimension permutations [37, 38, 12, 52, 17]. Such permutations are defined on the bits of the address field. Conversion between the *cyclic* and *consecutive* [21] storage schemes, and many other storage schemes are dimension permutations, and so are shuffle permutations.

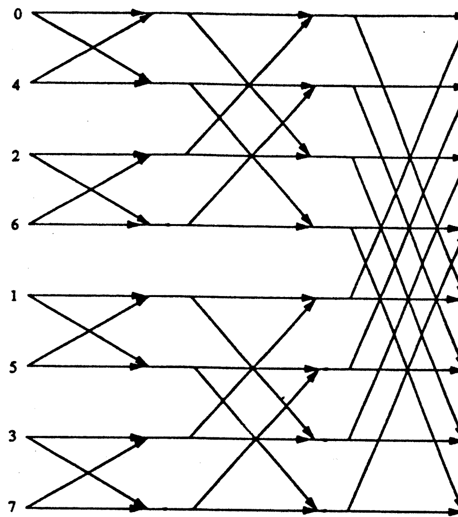

Figure 19: A butterfly network on 8 nodes.

In summary, computations on domains with a regular discretization often have a data interaction that can be modeled by

- Lattices of arbitrary dimensionality and shape.
- Butterfly networks.
- *PM2I* networks.
- Pyramid networks.
- Spanning trees, spanning graphs, and multiple instances thereof.
- Dimension permutations.

For complex data structures, and many dynamic data structures the data interaction is often difficult to characterize in a simple manner. A *shared memory* model of computation is often the only feasible solution, with an accompanying performance penalty. Recently, a technique for optimal emulation of a shared memory programming model in a distributed memory architecture has been proposed [44].

The challenge for the computer architect is to select a communications network such that the potential for exploring locality is an option for the programmer, and the compiler writer. The challenge for the programmer and the compiler writer is to determine a data allocation that preserves locality of reference to the extent possible in the network. The challenge for the communications software, or hardware is to route messsages such that the time to cary out a given communication task is minimized given the constraints defined by the network.

In order to further the understanding of these issues extensive research is currently in progress. One direction of this research is to find networks that can emulate any other

network with a moderate slowdown, so called universal networks. Another direction is to find the optimal address map and routing algorithms for emulation of a given network, correpsonding to the communication needs, on another network, corresponding to the physical network interconnecting processors. This approach is favored for relatively simple data structures with well defined communication patterns. A third approach, appropriate for complex and dynamic data structures, is to search for address maps and routing algorithms that minimizes the routing time in the worst possible case. Rapid progress is being made in all three areas.

7 Programming

Architectures in which tens of thousands of operations can be performed concurrently are often referred to as *data parallel* to emphasize massive parallelism, and to distinguish them from *control parallel* architectures, which usually offer a considerably lower degree of concurrency. In a data parallel programming model, algorithms are designed based on the structure and representation of the problem domain. It is considered to consist of sets of *elementary objects*, where objects in the same set are subject to the same transformations (at least most of the time). The objects in the same set can be operated upon concurrently. Different sets of elementary objects are subject to different transformations, but may be operated upon concurrently. An elementary object contains the object description, as well as a description of the object state. An algorithm is expressed as a sequence of transformations of the state of an elementary object, and interactions between elementary objects. For instance, in the finite element method, the physical domain is discretized by a set of finite elements. Apparent choices of elementary objects are finite elements, and nodal points [25].

In data parallel programming languages, sets of elementary objects are represented by a higher level data type, such as the vector data type in some Fortran dialects, the array extensions of Fortran 8X [33], arrays in APL, the type poly in C* [1], and parallel variables in *Lisp [53]. In Fortran 8X the computation defined by a 7-point stencil at every point in a three dimensional grid can be expressed using the function CSHIFT, which defines a circular shift. No explicit loops are required for the array axes.

```
      subroutine psolve(phi, omega, inside, n, iter)
      real phi(n, n, n), omega(n, n, n), factor
      logical inside(n, n, n)
      factor = 1.0/6.0
      do 100 i=1,iter,1
        where(inside)
        phi = factor * (
1         CSHIFT(phi, dim=1, shift=-1) +
2         CSHIFT(phi, dim=2, shift=-1) +
3         CSHIFT(phi, dim=3, shift=-1) +
4         CSHIFT(phi, dim=1, shift=+1) +
5         CSHIFT(phi, dim=2, shift=+1) +
6         CSHIFT(phi, dim=3, shift=+1) ) +
7         omega
        endwhere
 100  continue
      return
      end
```

The first argument for CSHIFT is the variable to which the shift is applied, the second defines the axis along which the shift takes place, and the third argument defines the length and direction of the shift. If the **where** statement is omitted, then the code would correspond to periodic boundary conditions.

Another example in which a large number of concurrent matrix-vector products are performed is given next. The operation dominates the computation in the iterative solver used in a three dimensional finite element computation for a domain discretized by first order brick elements [24]. The state is represented by three displacements, $x = (u, v, w)$. The local interaction matrix, the elemental stiffness matrix, is a 1×8 vector of 3×3 matrices, i.e., a 3×24 matrix. In the particular finite element code from which the code segment is extracted, the elemental stiffness matrices are not assembled into a global stiffness matrix. Instead, a matrix vector product is performed for each element, and a total product vector assembled.

```
CMF$LAYOUT K(:SERIAL, :SERIAL, , , ), R(:SERIAL, , , ), X(:SERIAL, , , )
REAL K(3,24, 32, 32, 32), R(3,32,32,32), U(3,32,32,32), V(3,32,32,32), W(3,32,32,32), X(24,32,32,32)
CALL ALL-TO-ALL-ELEMENT-BROADCAST(U,V,W,X)
R = 0.0
DO I=1,24
  DO J=1,3
    R(J,:,:,:)=R(J,:,:,:)+K(J,I, :, :, :) * X(I, :, :, :)
  END DO J
END DO I
```

```
(WHERE (.NOT. I-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 1, 1)
(WHERE (.NOT. I-LEFT-BOUNDARY)) R= EOSHIFT(R, 1, -1)
(WHERE (.NOT. J-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 2, 1)
(WHERE (.NOT. J-LEFT-BOUNDARY)) R= EOSHIFT(R, 2, -1)
(WHERE (.NOT. K-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 3, 1)
(WHERE (.NOT. K-LEFT-BOUNDARY)) R= EOSHIFT(R, 3, -1)
```

I–RIGHT–BOUNDARY, **I–LEFT–BOUNDARY**, etc. are boolean arrays that define the right–hand and left–hand boundaries of each finite element in the three dimensions respectively. The code segment contains one compiler directive, **SERIAL**, which is not part of the proposed Fortran 8X language. It is used in the version of the language implemented on the Connection Machine system as a tool to control the data layout.

With higher-level data types present in data parallel languages operations on sets are also natural parts of the language. Reduction, copy, parallel prefix operations, and certain permutations are examples of such operations. In the proposed Fortran 8X **SUM** and **SPREAD** are examples of the first two operations. Scans in APL and *Lisp are examples of parallel prefix operations, and reshape operations in APL and Fortran 8X are examples of dimension permutations under certain restrictions on the array. The multi-prefix instruction in the Fluent architecture [45] is a generalization of the parallel prefix instruction.

8 Summary

The current generation supercomputers are all parallel computers, but the degree of concurrency in the systems vary from a few processors to thousands of processors. The next generation of supercomputers with performance of a trillion operations per second will all be massively parallel. Many of the algorithms in use today for scientific and engineering applications can be modified to exploit this degree of parallelism effectively. Programming these massively parallel systems requires languages with a higher level of abstraction than is typical in the traditional programming languages. The detailed management of each variable is not feasible. Critical to the success of the trillion operations per second architecture built in state-of-the-art technologies is the preservation of locality inherent in many applications.

Key issues in determining the data allocation with the data structure distributed across tens of thousands of storage modules interconnected by a network are communication and load balance [22]. Much of the research in computer architecture and compiler technology during the last decade has been aimed at developing an understanding and techniques for determining optimal address maps, and path selection and scheduling for data routing for a variety of networks and communication patterns. Significant progress has been made, and some of these techniques are being implemented in state-of-the-art supercomputer

systems, like the Connection Machine system. Another research direction is the search for universal networks, i.e., networks that can emulate all other networks with a loss in performance that is independent of the network size. This type of research has led to the invention of networks such as the fat-tree [30].

For applications with dynamic data structures, or complex access patterns, a shared memory programming model may be preferable, even though such a model must be implemented on a system with a physically distributed memory. A general routing algorithm that provides a provably optimal shared memory emulation in a worst case sense was recently proposed in [44]. This routing offers combining at a very low hardware expense, and shows that prefix operations on multiple, arbitrary sets can be performed as a single instruction, a multi-prefix instruction. The routing algorithm allows for single instruction set operations, such as insertion, deletion, union etc., also at a very low hardware expense. The routing algorithm indeed forms the basis for a very powerful programming model.

The understanding of how to effectively use massively parallel, network architectures is progressing rapidly. The level of innovation in computer architecture and software technology is higher than in the past two decades, and perhaps as high as in the first decade of computer design. The impact of massively parallel architectures with a peak performance of several trillion operations per second, a primary storage of a hundred Gbytes, and Tbytes of secondary storage will profoundly change how computational sciences, engineering, and information intensive real-time applications are carried out in the next decade. The availability of workstations with a performance of hundreds of millions of operations per second will have an equally profound impact on what tasks are carried out as routine engineering work, and high quality interactive color graphics will dramatically change the human interface. This decade will be exciting for both designers and users of computer systems at all levels.

References

[1] Programming in C*. Thinking Machines Corp., 1987.

[2] Clive Baillie, S. Lennart Johnsson, Luis Ortiz, and G. Stewart Pawley. QED on the Connection Machine. In *The Third Conference on Hypercube Concurrent Computers and Applications*, pages 1288–1295. ACM Press, January 1988.

[3] Kenneth E. Batcher. Sorting networks and their applications. In *Spring Joint Computer Conference*, pages 307–314. IEEE, 1968.

[4] C. Gordon Bell and Allen Newell. *Computer Structures: Readings and Examples.* McGraw-Hill, 1971.

[5] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31:333–390, 1977.

[6] Jean-Philippe Brunet, Danny C. Sorensen, and S. Lennart Johnsson. A data parallel implementation of the divide-and-conquer algorithm for computing eigenvalues of tridiagonal systems. Technical report, Thinking Machines Corp., 1989. in preparation.

[7] Arthur W. Burks, Herman H. Goldstine, and John von Neumann. Preliminary discussion of the logic design of an electronic computing instrument. Technical report, U.S. Army Ordnance Department, 1946. Reprinted in Bell and Newell, Computer Structures: Readings and examples, McGraw-Hill, 1971.

[8] D.J.E. Callaway and A. Rahman. *Phys. Rev. Lett.*, 49:613, 1982.

[9] Jim C. Cooley, P.A.W. Tukey, and P.D. Welch. *J. Sound Vibrations*, 12(3):315–337, 1970.

[10] J.J. Dongarra and D.C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Scientific and Statistical Computing*, 8(2):s139–s153, 1987.

[11] S. Duane. *Nucl Phys.*, B257:652, 1985.

[12] Peter M. Flanders. A unified approach to a class of data movements on an array processor. *IEEE Trans. Computers*, 31(9):809–819, September 1982.

[13] Paul Frederickson and Oliver McBryan. *Parallel Superconvergent Multigrid.* Marcel Dekker, 1988.

[14] W. Morven Gentleman. Some complexity results for matrix computations on parallel processors. *J. ACM*, 25(1):112–115, January 1978.

[15] John L. Hennessey, N. Jouppi, Forrest Baskett, and J. Gill. Mips: A VLSI processor architecture. In *VLSI Systems and Computations*, pages 337–346. Computer Sciences Press, 1981.

[16] John L. Hennessey, N. Jouppi, S. Przybylski, and C. Rowen. Design of a high performance VLSI processor. In *Proc. of the Third Caltech Conference on VLSI*, pages 33–54. Computer Sciences Press, 1983.

[17] Ching-Tien Ho and S. Lennart Johnsson. Optimal algorithms for stable dimension permutations on Boolean cubes. In *The Third Conference on Hypercube Concurrent Computers and Applications*, pages 725–736. ACM, 1988.

[18] Roger W. Hockney and C.R. Jesshope. *Parallel Computers.* Adam Hilger, 1981.

[19] J.W. Hong and H.T. Kung. I/O complexity: The red-blue pebble game. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 326–333. ACM, 1981.

[20] R. Michael Hord. *The ILLIAC IV: The first supercomputer.* Computer Sciences Press, 1982.

[21] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[22] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures.* Morgan Kaufman, 1989.

[23] S. Lennart Johnsson and Ching-Tien Ho. Optimizing tridiagonal solvers for alternating direction methods on Boolean cube multiprocessors. *SIAM J. on Scientific and Statistical Computing*, 11(3), 1990.

[24] S. Lennart Johnsson and Kapil K. Mathur. Experience with the conjugate gradient method for stress analysis on a data parallel supercomputer. *International Journal on Numerical Methods in Engineering*, 27(3):523–546, 1989.

[25] S. Lennart Johnsson and Kapil K. Mathur. Data structures and algorithms for the finite element method on a data parallel supercomputer. *International Journal of Numerical Methods in Engineering*, 29(4):881–908, 1990. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-743, Technical Report CS89-1, Thinking Machines Corp., December, 1988.

[26] S. Lennart Johnsson, Yousef Saad, and Martin H. Schultz. Alternating direction methods on multiprocessors. *SIAM J. Sci. Statist. Comput.*, 8(5):686–700, 1987.

[27] M.G.H. Katevenis. *Reduced Instruction Set Computer Architectures for VLSI.* MIT Press, Cambridge, MA, 1985.

[28] Simon Lavington. *Early British Computers.* Digital Press, 1980.

[29] Ding Lee, Yousef Saad, and Martin H. Schultz. An efficient method for solving the three-dimensional wide angle wave equation. Technical Report YALEU/DCS/RR-463, Department of Computer Science, Yale University, October 1986.

[30] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, 34:892–901, October 1985.

[31] Christoffer Lutz, Steve Rabin, Charles L. Seitz, and Donald Speck. Design of the mosaic element. In *Proceedings, Conf. on Advanced research in VLSI*, pages 1–10. Artech House, 1984.

[32] Carver A. Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[33] Michael Metcalf and John Reid. *Fortran 8X Explained.* Oxford Scientific Publications, 1987.

[34] N Metropolis, J Howlett, and Gian-Carlo Rota, editors. *A History of Computing in the Twentieth Century.* Academic Press, 1980.

[35] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. *J. Chem Phys*, 21:1087, 1953.

[36] P. Morison and E. Morrison, editors. *Charles Babbage and his Calculating Engine.* Dover, 1961.

[37] David Nassimi and Sartaj Sahni. An optimal routing algorithm for mesh-connected parallel computers. *JACM*, 27(1):6–29, January 1980.

[38] David Nassimi and Sartaj Sahni. Optimal bpc permutations on a cube connected simd computer. *IEEE Trans. Computers*, C-31(4):338–341, April 1982.

[39] Pelle Olsson and S. Lennart Johnsson. A dataparallel implementation of explicit methods for the three-dimensional compressible Navier-Stokes equations. *Parallel Computing.* Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-747, October 1989, Technical Report CS89-4, Thinking Machines Corp., February 1989.

[40] Carlton M. Osburn and Arnold Reisman. Challenges in advanced semiconductor technology for high-performance and supercomputer applications. *The Journal of Supercomputing*, 1(2):149–189, 1987.

[41] Tekla S. Perry. Intel's secret is out. *IEEE Spectrum*, 26(4):22–28, 1989.

[42] J. Polonyi and H.W. Wyld. *Phys. Rev. Lett.*, 51:2257, 1983.

[43] J.L Potter, editor. *The Massively Parallel Processor.* MIT Press, Cambridge, MA, 1985.

[44] Abhiram G. Ranade. *Fluent Parallel Computation.* PhD thesis, Yale University, 1988.

[45] Abhiram G. Ranade, Sandeep N. Bhatt, and S. Lennart Johnsson. The Fluent abstract machine. In *Advanced Research in VLSI, Proceedings of the fifth MIT VLSI Conference*, pages 71–93. MIT Press, 1988.

[46] R. Richtmyer and K.W. Morton. *Difference Methods for Initial-Value Problems.* Wiley-Interscience, 1967.

[47] Charles L. Seitz. Concurrent VLSI architectures. *IEEE Trans. Comp.*, 33(12):1247–1265, 1984.

[48] Charles L. Seitz. Experiments with VLSI ensemble machines. *J. VLSI Comput. Syst.*, 1(4):311–334, 1986.

[49] Howard J. Siegel. *Interconnection Networks for Large Scale Parallel Processing.* Lexington Books, 1985.

[50] Harold S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 1987.

[51] Ivan E. Sutherland and Carver A. Mead. Microelectronics and computer science. *Scientific American*, pages 210–228, September 1977.

[52] Paul N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[53] Thinking Machines Corp. **Lisp Release Notes*, 1987.

[54] Maurice V. Wilkes. Babbage as a computer pioneer. In *Proceedings of the Babbage Memorial Meeting, 1971.* British Computer Society, 1972. Reprinted in Historia Mathematica, vol. 4, pp. 415-440, 1977.

[55] Kenneth G Wilson. *Phys. Rev*, D10:2445, 1974.

[56] O.C. Zienkiewicz. *The Finite Element Method.* McGraw-Hill, 1967.

[49] Howard J. Siegel. *Interconnection Networks for Large Scale Parallel Processing.* Lexington Books, 1985.

[50] Harold S. Stone. *High-Performance Computer Architecture.* Addison-Wesley, 1987.

[51] Ivan E. Sutherland and Carver A. Mead. Microelectronics and computer science. *Scientific American*, pages 210–228, September 1977.

[52] Paul N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[53] Thinking Machines Corp. **Lisp Release Notes*, 1987.

[54] Maurice V. Wilkes. Babbage as a computer pioneer. In *Proceedings of the Babbage Memorial Meeting, 1971.* British Computer Society, 1972. Reprinted in Historia Mathematica, vol. 4, pp. 415-440, 1977.

[55] Kenneth G Wilson. *Phys. Rev*, D10:2445, 1974.

[56] O.C. Zienkiewicz. *The Finite Element Method.* McGraw-Hill, 1967.