Special Purpose Parallel Computing

W F McColl *
Programming Research Group
Oxford University

Abstract

A vast amount of work has been done in recent years on the design, analysis, implementation and verification of special purpose parallel computing systems. This paper presents a survey of various aspects of this work. A long, but by no means complete, bibliography is given.

1. Introduction

Turing [365] demonstrated that, in principle, a single general purpose sequential machine could be designed which would be capable of efficiently performing any computation which could be performed by a special purpose sequential machine. The importance of this universality result for subsequent practical developments in computing cannot be overstated. It showed that, for a given computational problem, the additional efficiency advantages which could be gained by designing a special purpose sequential machine for that problem would not be great.

Around 1944, von Neumann produced a proposal [66, 389] for a general purpose stored-program sequential computer which captured the fundamental principles of Turing's work in a practical design. The design, which has come to be known as the "von Neumann computer", has served as the basic model for almost all sequential computers produced from the late 1940s to the present time. As noted in [163], "The paper by Burks, Goldstine and von Neumann ([66]) was incredible for the period. Reading it today, one would never guess this landmark paper was written more than 40 years ago, as most of the architectural concepts seen in modern computers are described there." For an account of the principles of modern general purpose sequential (i.e. von Neumann) computer design, see e.g. [163]. For sequential computation, the stability of the von Neumann model has permitted the development, over the last three decades, of a variety of high level languages and compilers. These have, in turn, encouraged the development of a large and diverse software industry producing portable applications software for the wide range of von Neumann machines available, from personal computers to large mainframes. The stability of the underlying model has also allowed the development of a robust complexity theory for sequential computation and a set of algorithm design and software development techniques of wide applicability.

No single model of parallel computation has yet come to dominate developments in parallel computing in the way the von Neumann model has dominated sequential computing [135,

*Author's address: Programming Research Group, Oxford University, 11 Keble Road, Oxford OX1 3QD, UK. This paper was written while the author was a Visiting Scientist at NEC Research Institute, Princeton, USA.

375, 376]. Instead we have a variety of models such as VLSI circuits, systolic arrays and distributed memory multicomputers, in which the careful exploitation of network locality is crucial for algorithmic efficiency, and some models such as the PRAM in which there is no notion of network locality whatsoever. In this paper we will be concerned with the locality-based models. In a related paper [272] we consider locality-independent models and examine the prospects they offer for achieving the goal of a robust general purpose parallel computing framework, comparable to that provided for sequential computing by the von Neumann model.

2. Network Locality

In the 1970s it became clear [211, 280, 355] that developments in VLSI technology would permit, not only the design of cheaper and faster general purpose sequential computers, but also the design of cost-effective special purpose parallel systems for computationally intensive problems. In [279], it was shown that the basic principles of VLSI design could be understood at a level somewhat above that of device physics. The publication of [279] had an immediate and profound impact. A large number of computer scientists were encouraged to think about the design and analysis of algorithms for implementation, not as programs to be run on a general purpose computer, but as special purpose hardware, for example, as custom VLSI chips or as systolic arrays.

In the early 1980s the first distributed memory multicomputers began to appear, e.g. [170, 181, 339, 397]. These systems consisted of a set of general purpose microprocessors connected by a sparse communications network, e.g. an array, butterfly or hypercube. As each processor typically had its own private physical memory module, the most natural memory organisation in these early architectures was to have multiple private address spaces with explicit message passing by the programmer [168, 170, 187] for all non-local memory requests. In most of the early multicomputer systems, message passing was implemented using simple, store-and-forward communication, and hence was extremely slow for messages which had to pass through a number of intermediate processors before reaching the destination processor. The key to algorithmic efficiency in each of these types of system - VLSI chips, systolic arrays and multicomputers, is the careful exploitation of *network locality*. In the different types of system it manifests itself in different ways. In a VLSI system, a design with good network locality will have short wires, and hence will require less area. An efficient systolic algorithm will have a simple, regular structure and use only nearest neighbour communication. An efficient multicomputer algorithm will be one which minimises the distance that messages have to travel in the network by careful mapping of the virtual process structure onto the physical processor architecture. Of course, an efficient algorithm for, say, a hypercube multicomputer will not necessarily perform well when run on, for example, a 2D array multicomputer with the same number of processors.

The exploitation of network locality to achieve efficiency is the main idea which unites the various parallel algorithms which we describe in this paper. We will use the generic term "special purpose" to refer to this type of parallel computing. It can be contrasted with the development of algorithms for what we will refer to as a "general purpose" system [272], i.e. one in which the costs of communication between pairs of processors in the system is essentially uniform for all pairs. Examples of general purpose architectures, in this sense, would be the PRAM or the bulk-synchronous parallel computer [375, 376]. If architectures which efficiently support such a uniform memory model can be realised then they will offer major advantages in simultaneously providing both architecture-independence in software and scalable parallel performance. However, as noted in the next section, there are a number of application areas in parallel computing, e.g. signal processing, where the special purpose approach is likely to continue to be the method of choice.

Significant motivation for the special purpose approach is often drawn from the fact that computation is a physical process and must therefore be carried out in physical 3D space. Some, e.g. [232, 233, 383, 384] argue that this is such a fundamental constraint that it should play a major role in determining the model of parallel computation to be used in algorithm design. While accepting the validity of these physical 3D space arguments, the advocate of a more network-independent general purpose approach (such as the current author) would argue that, first and foremost, a model must be mathematically robust if it is to find widespread acceptance as a basis for the design and analysis of algorithms. Two examples of mathematically robust models are the unit cost RAM for sequential computation and the PRAM for idealised parallel computation. Both have proven to be very appropriate for encouraging the development of a wide variety of algorithms and design techniques, although neither reflects the true costs of physical computation in a highly accurate way. The major disadvantage of adopting a physically based model is, however, that it is likely to accurately reflect the costs of only one technology. There is strong evidence that the various costs associated with the emerging optical technologies are likely to be quite different from those which apply with VLSI semiconductor technologies. We will return to this issue later in the paper. Irrespective of one's opinion on the special purpose/general purpose issue, there can be no doubt that many of the special purpose designs produced in recent years are remarkable achievements in terms of the complex dynamic patterns of communication which are involved. The constraint of having to strive for network locality has, in many cases, resulted in algorithm designs of great elegance and ingenuity. The fact that many of the algorithms are indeed so intricate does, of course, indicate that it may be difficult for this style of "special purpose parallel programming" to become a mainstream activity in computing, as opposed to a niche.

3. Applications

Special purpose parallel systems are particularly appropriate in application areas where the goal of achieving the best possible performance is much more appropriate than that of achieving an architecture-independent design. In this section we briefly describe some of those areas.

The design of a high-performance special purpose system will often require the solution of a number of low level design problems such as efficient process mapping to minimise communication distances, or careful VLSI layout to minimise chip area. These extra difficulties in design can be justified in a number of situations. The most obvious one is where it is the only known way of achieving the required performance. Another is where the system is to form part of a high volume product, and therefore the extra design costs can be justified if they result in a system which is more cost-effective in performance when manufactured. Performance requirements come in a variety of forms. A robot control system may have strict real-time performance requirements which must be met, e.g. that the computation performed be completed in some given fraction of a second. Another example of a parallel computing application with a rather strict performance requirement is weather prediction. A prediction of the weather for a given day is only of value if it is produced before that day! For a third example of an area where achieving a given level of performance is crucial we need look no further than the design of computing systems themselves. Modern VLSI systems are amongst the most complex engineering designs ever produced. Even using the best algorithms known, the simulation of a VLSI circuit for the purpose of detecting errors in the design can take months on a supercomputer. In a highly competitive and fast moving industry such as the semiconductor industry, the time taken to complete such a simulation may seriously affect when the company can get the product onto the market, with potentially drastic consequences for sales. These three examples indicate the kinds of performance requirements which can justify the extra design difficulties associated with special purpose systems.

The "high volume" justification for special purpose design arises in application areas such as data compression and computer graphics where the market for a high-performance system is very large. It is likely that in the coming decade there will be major growth in the demand for "smart" consumer goods and user interface devices of all kinds. These are likely to be produced as application specific parallel architectures with similar characteristics to those described in this paper.

As a general rule, special purpose systems tend to be developed in areas where the basic algorithmic techniques involved are reasonably stable and well understood. The relative inflexibility of the design process tends to rule out application areas where a wide variety of high level algorithmic approaches are possible, and those where the algorithms and data structures involved are complex, dynamic and irregular. An area such as signal processing would appear to be ideally suited to the special purpose approach, whereas, at the present time at least, a complex, knowledge based system would not appear to be, although some computationally intensive component of it might be. In the remainder of this section we list a number of application areas for which special purpose systems are particularly appropriate at the present time. The list is not meant to be exhaustive, but merely to indicate the wide range of areas.

Digital Signal Processing

For a combination of technical, economic and performance reasons, a large number of digital signal processing applications are appropriate for special purpose implementation. Many such systems have been produced in recent years for various kinds of filtering and transform computations [214, 217]. Filtering: FIR, IIR, Wiener, Kalman, 1D and 2D convolution and correlation, 1D and 2D interpolation and resampling, template matching [214, 320]. Transforms: FFT, Walsh-Hadamard, discrete cosine, Hough, Karhunen-Loeve [40, 43, 89, 214, 358]. Another important application area is radar signal processing.

Image Processing and Pattern Recognition

As in the case of digital signal processing, many image processing applications require only relatively simple algorithmic structures, although they need to be performed very rapidly on large data sets [212, 214, 304, 312]. Some examples of computationally intensive problems in this area are: restoration, reconstruction, enhancement and smoothing, edge and line detection, template matching, texture analysis, basic geometric operations, syntactic pattern recognition, scene analysis, low level computer vision. For almost a decade, the NASA

Goddard Space Flight Center has been successfully using a 128×128 processor MPP system [23] for image processing applications [352]. This array architecture was specifically design to handle such problems.

Linear Algebra

For many fundamental linear algebra computations, if one is dealing with dense matrices or sparse regular matrices then one can obtain very efficient special purpose designs through the exploitation of network locality based on the matrix structure. Examples of such computations are: matrix multiplication, LU factorisation, Cholesky factorisation, solution of general, tridiagonal and triangular linear systems, matrix inversion, iterative solution of linear systems (e.g. using the conjugate gradient method), singular value decomposition, eigenvalue problems, QR factorisation, least squares problems, recursive least squares, [34, 85, 131, 132, 136, 137, 160, 162, 182, 185, 306, 311, 318, 331, 378]. The volume of published literature on this topic is huge. A bibliography on parallel numerical algorithms which contains over 2000 entries, many of them concerned with dense linear algebra computations, was produced in 1989 [296]. A useful entry point to the field is the survey paper [132].

Unfortunately, many linear algebra computations arising in practical applications involve very large sparse matrices with no regular structure [80, 86, 105, 138, 159, 191]. The efficient parallel solution of linear systems of this kind is generally much more complex than in the case of dense matrices. It often requires a number of very intricate algorithms and data structures to achieve acceptable efficiency. An excellent account of the difficulties involved in producing efficient parallel algorithms for the problem of sparse Cholesky factorisation can be found in [159]. Efficient Cholesky factorisation of large sparse matrices arises in a considerable number of computationally intensive applications. For example, it is at the heart of most of the modern interior point approaches to large scale linear programming [79, 173]. The careful exploitation of network locality seems less likely to deliver substantial additional performance gains in the case of large sparse irregular matrices. A random mapping of processes to processors, based simply on the goal of load balancing, may provide performance comparable to the best that could be achieved by careful design based on network locality. A similar situation exists with respect to many graph theoretic problems. The difficulty of designing efficient parallel algorithms for large sparse irregular matrix and graph problems provides a major practical impetus to much of the current work aimed at producing a more network-independent, general purpose approach to parallel computing. In [139] it is noted that some shared memory is desirable in parallel sparse matrix computations.

Scientific Computation

In industries such as the aerospace industry, the modelling of very complex scientific phenomena, e.g. turbulence, is an integral part of the design process. At present, such modelling problems account for a large percentage of the total time spent on all parallel supercomputers in use around the world. The problem areas include the following: computational fluid dynamics, direct and iterative solution of PDEs [266], 2D and 3D Navier-Stokes equations, lattice gas methods [46, 76, 100, 124, 125], finite element and finite volume methods [185, 186], multigrid methods [70, 299], climate modelling, weather prediction.

Simulation

In recent years, a number of new efficient algorithms have been developed for performing complex n-body particle simulations arising in astronomy, where the particles are interacting in a gravitational field, and in molecular modelling, where they are interacting in a Coulombic field [21, 147, 148, 257, 347, 410]. The implementation of these new methods on special purpose parallel architectures has enabled simulations to be performed on a scale which would have been unthinkable until recently. There is considerable scope for further improvements in this direction. Discrete-event simulation is another area where special purpose systems are appropriate, in view of the very specific techniques, such as the Time Warp mechanism of lookahead and rollback, which are currently being used to achieve concurrency in such simulations [128, 129, 130, 287]. Monte Carlo simulation was originally developed by Fermi, von Neumann and Ulam to solve certain problems related to neutron transport. Since then it has been widely used in applications where the goal is to accurately reproduce the microscopic dynamics of some statistical physical system [155]. For example, in the electronics industry it is used to model transport in semiconductor devices and to perform process modelling. The Monte Carlo method is also frequently used for the numerical evaluation of multidimensional integrals for which no analytical solution can be obtained. As in the case of the n-body simulations, special purpose parallel architectures will open up the possibility of performing Monte Carlo simulations with a greatly increased number of points, yielding results of much greater accuracy.

Communications

Although the capacity of communication networks is increasing steadily, the demands being placed upon such networks is probably increasing more rapidly. The development of special purpose parallel devices which quickly compress and expand text [356], images, audio and video [361] for more efficient use of communication networks is likely to be a major activity in the next few years. Other special purpose communications devices which give high performance encoding and decoding for error correction, and encryption and decryption for data security [345] are also likely to be in big demand.

Computer Graphics and Computational Geometry

Three-dimensional medical imaging [353], computer-aided design and solid modelling systems [406], spatial databases and computational geometry [74, 370], robot motion planning, scientific data visualisation, realistic image synthesis [97, 127, 303], ray tracing [144], animation, virtual reality systems, multimedia systems, high-definition television etc.

Molecular Modelling

Drug design has until recently been conducted in a rather haphazard manner. In an attempt to make it more rational and scientific, major pharmaceutical companies and large medical and scientific research organisations are increasingly using advanced computer systems to calculate the structure and dynamics of large biological macromolecules (proteins, carbohydrates and nucleic acids) using molecular mechanics [271, 349], molecular dynamics [267, 271, 349, 379, 380], free energy calculations [22] and quantum mechanical methods

[134, 161, 167]. At present, even very simple molecular modelling calculations of this kind can take months on the fastest available commercial supercomputers. Special purpose parallel architectures can provide improvements of orders of magnitude in performance. Other computationally demanding molecular modelling applications include structure prediction from NMR data using distance geometry methods [51, 92, 158] and modelling of new materials.

VLSI Design

The design and simulation of VLSI circuits requires a number of specialised tools for problems such as layout and wire routing, design rule checking, logic verification, switch-level simulation and circuit simulation [142, 279, 396]. Each of these is a major computationally intensive problem. The size of VLSI circuits continues to increase rapidly, creating a growing demand for very high performance special purpose parallel systems for these tasks. In performing layout and wire routing [239, 369], one is typically aiming to minimise the area required for the circuit, as the size of a chip can significantly affect the yield obtained when it is fabricated. Optimisation techniques such as simulated annealing can often be used to produce layouts which are close to optimal [1, 2, 196, 381]. The purpose of design rule checking is to ensure that all geometric features laid out on each mask meet the size, spacing and overlap rules. These design rules are specific to the technology and fabrication process to be employed. The geometric computation performed by a design rule checker [190] is similar to that performed by a hidden-line algorithm in computer graphics. The various types of VLSI verification and simulation are as follows. In logic verification, the circuit is modelled as a set of gates and flip-flops and the task is to verify that the design meets a specification given in some formalism such as higher-order logic [87] or temporal logic [63] For a survey of approaches to logic verification, see [275] and various other papers in [276]. In switch-level simulation, each transistor is modelled as a voltage-controlled switch which can be in a small fixed number of states [18, 64, 202]. A circuit simulation is the most detailed type of analysis and is usually only applied to small parts of a circuit design. Such simulations are often performed to get accurate information on timing delays. A well known example of a circuit simulator is the program SPICE [385] which models circuits containing transistors. A simulator such as SPICE solves the set of nonlinear equations corresponding to the electrical behaviour of the components in the circuit (gate capacitances, electron mobilities etc.) and the topology of those components. Standard algorithms such as Newton-Raphson are used.

Speech Processing

Continuous speech recognition, dynamic programming methods (time warps)[126, 405], speech synthesis.

Computational Physics

Lattice-gauge theories and quantum chromodynamics (QCD) [29, 32, 99].

Game Playing

Progress in computer chess in recent years has largely been based on the exploitation of special purpose parallel architectures [107, 174, 175]. It seems probable that the first non-human World Chess Champion will be a special purpose parallel computing system.

Control

Industrial control, mobile robot control etc. [61, 62]

Database Systems

Text retrieval, pattern matching in protein and nucleic acid sequence databases (Human Genome Project) [15, 246, 271, 329], relational databases (fast SQL servers) [210, 244], on-line transaction processing etc.

Financial Modelling

The valuation of complex financial instruments that appear in the banking and insurance industries requires simulation of their cashflow behaviour in a volatile interest rate environment. These simulations can be very computationally intensive and can benefit enormously from parallel computing. The use of parallelism in the simulation of multiple paths of interest rates is relatively straightforward. To exploit parallelism in performing a path dependent calculation along each of the paths one can use techniques such as parallel prefix computation [44, 45, 218]. Using advanced parallel systems, Hutchinson and Zenios [179] have shown that it is possible to perform the valuation of a single mortgage-backed security in real time (1-2 seconds) and that the analysis of a portfolio of such securities can be performed very rapidly. In the future, special purpose parallel systems are likely to offer significant advantages to those engaged in activities such as real-time trading and planning in the face of uncertain interest rates.

4. Technologies

Special purpose parallel systems can be realised using a wide variety of technologies. In this section we describe a number of those technologies.

4.1. VLSI Systems

The design and fabrication of a custom VLSI circuit [142, 279, 396] is one approach to implementing a special purpose system. Since the publication of [279], the principles of VLSI design have become not only a mainstream part of electronics but also of computer science. A large number of new and interesting VLSI designs for basic problems in computer science have been produced as a result [24, 75, 238, 240, 357, 369, 394]. At present, VLSI is essentially a two dimensional technology. A VLSI circuit is formed from a small (fixed) number of two dimensional layers, the exact number of layers depending on the particular type of circuit, e.g. CMOS [396]. More advanced fabrication processes in the future may permit full three dimensional VLSI designs to be realised. Although such a technological advance may yield more efficient devices, it is not likely to dramatically change the principles of efficient VLSI design which have been developed in recent years. We would instead see our current concern for area efficiency in the two dimensional model being replaced by a corresponding concern for volume efficiency in the three dimensional model. In fact, some theoretical work in this direction has already been done [229, 305, 322]. The impact of a technological change such as the replacement of VLSI by free-space optical systems [113, 252] would, on the other hand, be a fundamental change, with profound implications for the design and analysis of algorithms.

In deciding whether to implement a special purpose parallel system as a custom VLSI chip one has to weigh the advantages against the disadvantages. The arguments in favour of a custom VLSI implementation are typically that one can obtain the highest possible performance by that means and/or that one will be able to manufacture the custom VLSI system more cheaply. The latter may be very significant if it is to form part of a high volume product. The major disadvantages of custom VLSI design and implementation is that it is a very costly and time consuming process, and that it results in a product which cannot easily be changed. Although a large number of interesting and detailed VLSI algorithm designs have been developed over the last decade, remarkably few of them have actually been realised as custom chips. The reason for this is, of course, that few organisations can justify the prohibitive costs of producing such a chip.

4.1.1. Field-Programmable Gate Arrays

In the last five years, companies such as Xilinx [407] and Algotronix [7] have started to offer an attractive alternative to custom VLSI implementation. A field-programmable gate array (FPGA) [7, 33, 108, 253, 290, 298, 345, 407] consists of a matrix of logic cells with switched interconnections between them. The interconnections are configured afresh whenever the equipment is started up, by a program downloaded to the chip. By appropriate setting of the switching points, the programmer can group and interconnect the gates into useful functions: counters, flip-flops, shift registers, clocks, multiplexers, buffers, NOR gates, XOR gates etc.

The first gate array devices appeared in the 1970s. They were developed to provide a middle ground between off-the-shelf chips and full custom designs. They are essentially rows and columns of logic gates overlaid with rows and columns of metal paths. Customisation takes place in the last steps of the fabrication process using a design specific mask. These mask-programmed gate arrays offer a very large number of gates to work with. However, they suffer from the same problems as custom chips, they are expensive to produce and cannot be changed. Another approach is the programmable logic device. These are often structured as an array of AND gates with their outputs feeding into an array of OR gates. The earliest of these were programmed by electrically selecting and destroying fuses on the chip, leaving the desired interconnections intact. Later devices were electrically reprogrammable using ultraviolet-light-erasable memory technology. Programmable logic devices typically offer far fewer gates than FPGAs or mask-programmed arrays.

Compared to the other approaches, FPGAs would appear to offer an excellent compromise between flexibility and performance. As more powerful high-level design tools for FPGAs are developed [253, 298], the use of such arrays is likely to grow substantially in application areas where the volumes are insufficient to justify implementation either as a custom VLSI chip or as a mask-programmed gate array. FPGAs will also provide an unprecedented opportunity for computer scientists to pursue the development of high performance implementations of their parallel algorithms without requiring the services of a VLSI fabrication facility. We can look forward to a number of exciting new VLSI systems emerging as a consequence of this new found freedom.

4.1.2. Self-Timed and Delay-Insensitive Circuits

The complexity of custom VLSI systems has reached the point where the design time and design cost can greatly exceed the fabrication time and fabrication cost. Moreover, most VLSI systems designed today are monolithic and cannot easily be modified or improved [354]. A significant factor giving rise to this design difficulty and inflexibility is the synchronous nature of such systems. In recent years, a number of researchers have been investigating the possibilities of designing alternative forms of VLSI systems, such as self-timed and delay-insensitive circuits, which are not subject to the limitations imposed by the traditional clocked-logic framework [109, 262, 263, 338, 354, 368]. The use of delay-insensitive circuits, for example, can greatly simplify design at the system level as correct timing and control of operations can be achieved without explicit consideration by the system designer of circuit and interconnection delays, or of clock generation and distribution between components of the system. Only the sequences of operations to be performed need to be considered, not the times for performing each operation; operations are initiated after their predecessors are completed, not after some fixed and independently determined time. Such an approach offers the opportunity to build up complex systems by hierarchical composition from simpler ones, as in the case of modern software development methodologies. The resulting system designs are easier to modify and improve, in order to meet changing requirements or to take advantage of developments in VLSI technology. As in the case of FPGAs, the development of formal design methodologies and high-level design tools will result in self-timed and delay-insensitive circuits being used more and more as a means of implementing application specific VLSI designs.

4.2. Systolic Algorithms and Architectures

The concept of a "systolic array" was introduced by Kung and Leiserson in [211] where they describe the VLSI implementation of some matrix computations. In an outstanding paper, published slightly later [207], Kung discussed the key architectural issues involved in designing high performance special purpose parallel systems, and showed convincingly that the systolic approach offered an excellent means of producing such systems. A major problem in the design of special purpose parallel architectures is the input/output bottleneck. The systolic approach makes extensive use of pipelining, so that a large number of operations are performed on an input before it is output again. By increasing the ratio of computation to input/output in this way, a more balanced parallel design can be achieved which has no bottleneck at any point. Since those early papers, a whole subfield of parallel computing has developed, based around the concept of systolic computation, see e.g. [149, 206, 214, 231, 250, 258, 268, 277, 279, 291, 308, 309, 317]. Systolic array designs, and the related wavefront designs [213, 214], can be implemented in a variety of different ways. Depending on the application area and the performance requirements, they can be realised as custom VLSI systems, as FPGAs, or as programs to be run on a multicomputer. The wide range of applications for which the systolic approach is appropriate, has also motivated the development of programmable systolic machines.

4.2.1. Programmable Systolic Architectures

A group at Carnegie Mellon University, headed by H T Kung, started the development of a programmable systolic computer called Warp in 1984 [11]. Warp is a systolic array computer consisting of a linearly connected set of cells. Each cell contains a high performance programmable processor, a large local memory, and high bandwidth communications channels to adjacent cells in the linear array. The first ten-cell prototype was completed in 1986 and delivery of production machines started in 1987. A large number of computationally intensive application programs have been developed for the Warp architecture using the language W2 [11]. The applications include low-level vision for robot navigation, image processing, signal processing, scientific computing, magnetic resonance imagery, radar and sonar simulation, graph algorithms. The Warp project demonstrated the feasibility of programmable, high performance systolic array computers. In a successor project, Intel and Carnegie Mellon University have been developing the iWarp parallel architecture [27, 49, 50, 208]. Although based on the original Warp design, iWarp has added much more flexibility in the types of communication that can be efficiently supported. In particular, non-neighbour communication can be performed without involving programs at intervening cells. Also, iWarp architectures can handle intercell communication patterns which are quite different from the hardware interconnnect in the system, using techniques such as logical channels and wormhole routing. Unlike Warp, the iWarp design is probably best viewed as a modern general purpose multicomputer design with some special features to support the important class of systolic parallel computations.

4.3. Cellular Automata

The remarkable success of von Neumann's work on general purpose sequential computing has tended to link his name very strongly with the idea of sequential computation in the minds of many. However, he was also, in many ways, the major pioneer in parallel computing.

In the late 1940s, von Neumann began to develop a theory of automata. He envisaged a systematic theory which would be both mathematical and logical in form, and which would contribute in an essential way to our understanding of natural systems (natural automata) as well as to our understanding of both analog and digital computers (artificial automata). By the time of his premature death in February 1957, he had made major contributions in a variety of areas related to this general programme. He had developed the basic theory of self-reproducing cellular automata [388], demonstrated that reliable systems could be constructed from unreliable components [386], and had started to examine the structure and function of neural systems [387]. In this section we describe some of the achievements of von Neumann and others in the area of cellular automata, and discuss the significance of the model for the field of special purpose parallel computation. An up to date account of work on the development of reliable systems, stemming from his work in [386], can be found in [302]. The topic of neural systems will be briefly discussed at the end of the paper.

Influenced by the work of Turing [365] and others on the theory of automata and universal computing machines, von Neumann set himself the goal of producing a system which would be capable of self-reproduction. In his early studies of this problem he explored a continuous model of self-reproducing automaton based on a system of nonlinear partial differential equations. He also pursued the idea of a kinematic automaton which could, using a description of itself, proceed to mechanically assemble a duplicate from available parts. When he found it difficult to provide the rigorous and explicit rules and instructions needed to realise such an automaton, he redirected his efforts towards a model of self-reproducing automata based on an array of computing elements. [The idea of using such a parallel array had been suggested to him by S Ulam.] He eventually succeeded in designing a self-reproducing two dimensional cellular automaton [388], although the design he produced was extremely complex. It required around 200,000 computing elements, each of which could be in one of 29 states. [A *cellular automaton* consists of a set of identical cells arranged in a regular lattice structure with connections between each cell and its neighbours in the lattice. At a given time step $t+1$, the new state of each cell is computed in parallel as a function of the state of the cell and its neighbours at time t.] The complexity of von Neumann's design stemmed largely from the fact that he had tried to design not only a self-reproducing system but also one which would be a universal computing machine, equivalent in power to a Turing machine. Later work by Codd [83] produced a self-reproducing universal computing machine which required just 8 states per cell. Although simpler than von Neumann's, Codd's machine is still as complex as a modern digital computer. As noted by Langton [220], universality is a sufficient condition for self-reproduction, however it is not a necessary condition. In his paper, Langton [220] describes a very simple self-reproducing cellular automaton which is not based on the idea of a universal computing machine. Following the work of von Neumann, a great deal of work was done in the 1960s on the theoretical aspects of various types of cellular automata, see [65, 402].

In 1970, J H Conway produced a particularly interesting cellular automaton which he called "Life". Life is a two state cellular automaton on an infinite two dimensional grid. Each cell is either on or off, and the simple Life rule is the following. *If two of the eight nearest neighbours are on, then don't change. If three are on, then turn on. Otherwise, turn off.* Life may be thought of as describing a population of organisms, developing in time under the effect of counteracting propagation and extinction tendencies. A live cell will remain alive only when surrounded by two or three live neighbours, otherwise, it will feel either "overcrowded" or "too lonely" and will die. A dead cell will come to life when surrounded by exactly three live neighbours. Thus birth is induced by the meeting of three "parents". Conway's "Game of Life" was extensively reported by Martin Gardner in his widely read Scientific American column [133] and caught the attention of scientific amateurs and professionals all over the world. As a result, the incredible properties of various Life forms such as blinkers, gliders, glider guns, puffer trains, Catherine wheels, traffic lights and so on, became familiar to a large number of people, many of whom would otherwise have had little interest in cellular automata. A definitive account of these various forms can be found in [31] where it is also shown that Life can be used to construct a universal computing machine and, hence, can also be used to achieve self-replication. It is remarkable that a two state cellular automaton with such a simple rule can produce such complex phenomena. A number of three dimensional variants of Life have been studied by Bays [28]. A description of a one dimensional universal cellular automaton can be found in [245].

In the early 1980s, a number of researchers began to study cellular automata as models of complex dynamical systems and, more generally, as an alternative to differential equations in modelling physical phenomena [112, 259, 363, 382, 402]. Researchers such as Toffoli and Margolus have also produced various special purpose parallel machines for the simulation of cellular automata models [260, 261, 362, 364]. These machines provide performance at least several orders of magnitude greater than with a sequential computer, for a comparable cost. They have been used for a variety of physical modelling problems, including computing time correlations, self-diffusion, thermalisation, reflection and refraction, flow tracing, diffusion-limited aggregation, Ising spin systems, annealing, erosion, genetic drift, and spatial reactions. For details, see [261, 364]. As noted in [261], the advantages of using a special purpose parallel architecture optimised for cellular automata simulations are so great that for large scale cellular automata experiments, it would be absurd to use any other type of system. In terms of its impact on the development of special purpose parallel computing systems, one of the most important results of this research on "digital physics" has been the development of lattice gas methods for fluid flow problems.

4.3.1. Lattice Gas Computers

Partial differential equations have conventionally formed the basis of mathematical models of physical systems such as fluids. Only in rather simple circumstances can exact solutions to such equations be found. Most studies of fluid dynamics must therefore be based on digital calculations, which use discrete approximations to the original partial differential equations.

In recent years, an important class of cellular automata, called lattice gas automata, have been successfully used to model a variety of physical systems traditionally modelled by partial differential equations. Although the idea of using discrete methods such as cellular automata for modelling PDEs has been around for along time, the actual statement that cellular automata techniques could be used to approximate the solution of hydrodynamic PDEs was first discovered in August 1985 by Frisch, Hasslacher and Pomeau. For a description of the derivation, which assumes the validity of the Boltzmann equation, see [125]. The original demonstration by Frisch, Hasslacher and Pomeau of the validity of the lattice gas approach applied only to low-velocity incompressible flows near equilibrium. Since then, a huge amount of work has been carried out on the lattice gas method, and the range of applications for which the approach can be used is increasing steadily, see [46, 76, 100, 124, 364, 402].

One of the main reasons for the surge of interest in lattice gas methods has been the simple, regular and massively parallel nature of such computations and the prospects that offers for constructing special purpose lattice gas computers. Various designs for lattice gas architectures have been produced in the last few years [96, 204, 261]. Studies of related issues such as the problem of input/output complexity in lattice gas computations [203, 295] have also been carried out. Special purpose lattice gas architectures are likely to be the only way of achieving the performance required by researchers working on three dimensional fluid flow problems using these methods. A good indication of the requirements of such researchers can be seen in the target performance parameters in [96] for a dedicated lattice gas computer for the mid 1990s. They call for the design of a system capable of performing three dimensional incompressible hydrodynamics with flexible boundary conditions. It should have some flexibility in the possible rule sets for the lattice gas, have a high input/output rate,have around 10^{11} lattice sites (or cells), and should perform about 6×10^{15} site updates per second in total. It is unlikely that anything other than a very specialised system could come close to achieving these requirements.

Over the past few years, the technology for manipulating and observing material on an atomic scale has advanced at an astonishing pace. Although moving individual atoms is still essentially a laboratory game, laying down thin films of atoms has become serious business. By controlling precisely the structure and composition of layers of materials only an atom or two thick, scientists are demonstrating that they can program the electronic characteristics they want into a compound. Scientists have also succeeded in producing a variety of exotic quantum devices: two dimensional quantum wells, one dimensional quantum wires, and recently zero dimensional quantum dots [90, 143]. These developments open the way for a number of exciting new electronic and optical devices which would be much smaller, faster and more highly parallel than anything available today. Several researchers are currently working on techniques to produce two dimensional arrays of quantum dots. As each such dot can have multiple energy levels, it can, in principle, replace a number of conventional transistors. The physical structure of such an array should also allow adjacent quantum dots to communicate in certain ways. Arrays of this kind may, in the coming years, provide the means of developing quantum dot cellular automata machines of incredible computing power [90]. The exploration of array designs which permit quantum dot coupling of various kinds has only just begun. We can look forward some interesting possibilities for special purpose parallel computing systems from this work.

When Richard Feynman addressed the annual meeting of the American Physical Society on 29th December 1959, electrical engineers were struggling with the problems of fitting a few components on a chip. Nevertheless, Feynman, in a visionary lecture, described some of the opportunities for technology which would eventually emerge when single atoms and molecules could be manipulated. The text of his lecture was originally published as [114]. Feynman described techniques for developing computers with wires that were between 10 and 100 atoms in diameter, and circuits which were a few thousand angstroms across. He described the physics which would apply to circuits consisting of only a few atoms, and showed that such devices would present completely new opportunities for design. Feynman's remarkable ideas had little impact during the 1960s and 1970s, as the technology available in those years was far too primitive to permit the investigation of such structures. However, in the 1980s a number of researchers started to explore the possibilities of "nanotechnology", pursuing many of the ideas that he had originally proposed. Quantum dots are just one example of the kinds of interesting nanotechnologies that we can expect to emerge as a basis for the implementation of future special purpose parallel systems. In the next section we turn our attention away from exotic technologies and towards the mainstream.

4.4. Multicomputers

Most of the practical work in the last decade on the design of highly parallel algorithms has been carried out with a view to implementation on some particular distributed memory multicomputer architecture. By a multicomputer we mean a set of computing elements connected by a sparse communications network, e.g. an array or a hypercube. Each computing element consists of a general purpose microprocessors and its private physical memory module. A number of such multicomputer systems have been commercially available from companies in the United States and Europe since the mid 1980s. In such a system, memory is organised in terms of multiple private address spaces with explicit message passing by the programmer [168, 170, 187] for all non-local memory requests. In most of the early systems

[170, 181, 339, 397], message passing was implemented using simple, store and forward communication, and hence was extremely slow for messages which had to pass through a number of intermediate processors before reaching the destination processor. In some of the later ones, techniques such as logical channels and wormhole routing have been used to improve the performance of non-local communication [224, 340, 341]. However, in those systems there is still a substantial advantage to be gained by designing algorithms which require only local communication. The key to algorithmic efficiency in multicomputer systems is, therefore, the careful exploitation of network locality. An efficient multicomputer algorithm will be one which minimises the distance that messages have to travel in the interconnection network by careful design of the virtual process structure in the algorithm so that it maps efficiently onto the physical structure of the machine. Despite the difficulties of programming such systems, and the architecture dependence of the resulting software, a large number of interesting new high performance parallel algorithms have been developed for multicomputer architectures of various kinds. Although most of these algorithms have been oriented towards MIMD multicomputers, there has also been a significant amount of work on the more restricted class of SIMD machines [15, 23, 29, 116, 166, 304, 352, 399, 410]. Locality based algorithms have been developed for all kinds of interconnection network structures. For an introduction to this area, see e.g. [34, 123, 185, 224]. The following is a short list of some of the more common network structures for which such algorithms have been produced: arrays [224, 292, 335, 360], trees and Sneptrees [165, 242, 264], pyramids [5, 284, 351], the mesh of trees structure [178, 224], fat trees [232, 233], hypercubes [70, 80, 93, 152, 153, 162, 182, 224, 266, 312, 404], the cube connected cycles architecture [224, 307], butterflies [224] and various shuffle exchange and de Bruijn graphs [224, 350].

4.5. Optical Systems

In previous sections we have considered a number of technologies which are convenient for the development of parallel systems in which network locality must be exploited to achieve efficiency. These include VLSI systems, systolic architectures, cellular automata machines, and multicomputers based on message passing. In this section we turn our attention to optical systems, which, among other things, may provide a means of achieving highly efficient non-local communication in parallel systems.

Since the invention of the laser in 1960, optical computing has offered the prospect of systems with much greater parallelism and communications power than one could obtain through the use of electronics [30, 113]. It now seems likely that in the 1990s this prospect will finally become a reality. One can identify three distinct areas in which optical systems may have an important impact in the coming years. The first is simply as a more efficient communications technology, particularly for irregular and non-local communication. The second is as a means of achieving very high performance application specific systems by directly exploiting the capabilities of optical technology at the algorithmic level. The third is in the development of all optical general purpose digital computers, in which the logic gates, memory elements etc. are implemented using optical devices. We will briefly discuss these three uses of optical technology in the following sections. A good general introduction to optical systems, written from a computer science perspective, can be found in [113].

4.5.1. Optical Communication

Various kinds of communication systems are used in parallel architectures - buses [10, 101, 140, 272, 313, 375, 376], crossbars [113, 157, 334], networks [113, 300] etc. There are a number of reasons why optical technologies provide a more appropriate basis for the implementation of such systems than electronic technologies do. They include the following.

High bandwidth.

High speed of propagation. Electronic signals propagate approximately two orders of magnitude slower than light, because of the capacitance of electronic conductors.

Absence of interference. Electrons interact strongly with each other, through the Coulombic force. Therefore, electronic wires must be kept some distance apart, so as to prevent crosstalk. Photons, on the other hand, do not interact with each other. Beams of light carrying information can even cross each other without any problem. This absence of interference enables much greater density and parallelism to be achieved through the use of optics.

Dynamic reconfiguration. The interconnection pattern of an optical communication system can be dynamically changed in a way which would be difficult or impossible in electronics.

There are two basic approaches to the physical implementation of optical communication systems (i) the use of optical fibres as waveguides, and (ii) the use of free space. For communication over reasonably long distances it is necessary to conduct the light to wherever it should go. This can be done by using an optical fibre as a waveguide for the light. The propagation of light in (bent) optical fibres is controlled by ensuring the total internal reflection of the beam inside the fibre. The use of such fibre optic systems for long distance communication is already quite common in local area networks. In an application specific optical device or an all optical digital computer, the communication distances may be sufficiently short that it is more appropriate to simply allow the beams of light carrying information to cross in the free space between the various elements. [For engineering reasons it may be more appropriate to use a transparent material instead of free space. Nevertheless, we will refer to it as a free space interconnection.] The main advantage of this kind of free space interconnection is that one can more fully utilise the three dimensions of space. Free space interconnections can be implemented using various kinds of devices such as holographic elements and lenslet arrays. For details, see [113].

4.5.2. Special Purpose Parallel Computing with Optical Primitives

Special purpose optical systems can be divided into two classes (i) those that deal with image and signal processing, and (ii) those that deal with numerical computation. In image and signal processing systems we take an input beam of light, manipulate it, and produce an output beam of light. These systems exploit various optical phenomena, such as the ability to perform two-dimensional Fourier transforms. Optical numerical processors do not manipulate beams of light in that way, rather they deal with arrays of numbers, represented by multiple points of light. While many of these special purpose optical systems are analog in nature, this is not always the case; a number of systems which use digital encodings in order to improve accuracy have also been proposed.

The basic elements of optical computing systems are as follows. The light sources are semiconductor LEDs (light emitting diodes) or lasers. Light beams can be manipulated using various kinds of optical devices, e.g. beam splitters, gratings, lenses, and polarisers. In particular, lenses can be used to form a light distribution that is the Fourier transform of another light distribution. The intensity of a light beam can be detected by a photodiode, and the intensity distribution in an image can be captured by an array of such detectors.

Images are naturally thought of in terms of optical systems and light beams. Various image processing functions may be carried out directly on the image by application specific optical systems, with no need for sampling, quantisation etc. In such direct image processing systems, the input data is a two-dimensional analog light distribution. As the light beam traverses the various optical elements it is manipulated, and at the end of the process an output light distribution is produced. Some examples of "computations" which can be performed in this way include Fourier transforms, spatial filtering, cross correlation, deconvolution, Radon transforms, and synthetic aperture radar imaging. Direct signal processing systems are similar to these image processing systems, the main difference is that the data is a one-dimensional temporal signal, rather than a two-dimensional spatial image. Optical signal processors are useful for the kinds of computations performed in radar systems.

Light intensities (or amplitudes) may be used to represent numerical values in an analog manner, rather than the usual digital representation. With an analog optical representation, the basic arithmetic operations are trivial: addition is done by combining beams of light, and multiplication by the modulation of beams of light. Various important linear algebra and matrix operations can be simply and efficiently performed using these primitives [16, 20, 68, 113, 243]. An interesting example of the use of such systems is given in [111] where they show how an optical matrix-vector multiplier can be used iteratively to efficiently simulate the Hopfield model [172] of neural networks. Analog optical methods can also be used for the solution of certain types of partial differential equations [113].

In [19, 110, 315] several interesting new theoretical models of optical computation have been proposed, which capture, in a simple and mathematically tractable form, the costs involved in computing with electro-optical devices. A number of efficient algorithms for those models have been developed, and are described in the papers. These models provide an appropriate starting point for the development of a robust complexity theory which will help guide the future design of special purpose optical systems for important applications.

4.5.3. Digital Optical Computers

The goal of developing an all optical general purpose digital computer has been pursued for many years [53, 113, 176, 251, 252, 395]. Although much progress has been made in demonstrating that the various components required for such a system can indeed be realised as optical devices, it is still not clear whether such systems will ever outperform their electronic counterparts. A good discussion of the prospects for all optical systems is given in [113], which also describes the various techniques which can be used to design optical logic elements, arithmetic units, memory systems etc.

5. VLSI Systems

VLSI models of parallel computation have been extensively studied in the last decade. The two main characteristics of such models are *finite resolution* and *quasi planarity*. The finite resolution property stems from the fact that a component on a VLSI chip, i.e. a gate or a wire, takes a finite amount of space and cannot be made arbitrarily small. The fabrication technology used will determine the minimum feature size which can be achieved. The quasi planarity property follows from the fact that a VLSI chip is composed of a small fixed number of layers. The precise number depends on the technology used. It can be as small as two or as large as about thirty. However, it is always fixed, i.e. it does not grow with the size of the circuit.

The goal of VLSI algorithm design is to produce an algorithm which, when embedded in such a quasi planar structure, optimises certain complexity measures such as area and time. A large number of efficient VLSI algorithms have been obtained for important problems. The upper bounds on complexity obtained from those algorithms have also, in many cases, been closely matched by lower bounds. Most of the lower bounds have been established by simply considering the required communication of data between the various components on the chip, and to a large extent ignoring the details of the computation. This shows, in a very convincing way, that in the design of VLSI systems, it is the efficient communication of information within the chip which is the dominant concern.

Several different VLSI models have been proposed and studied. The following description is intended to capture the main features of those models. A VLSI chip is a synchronous Boolean circuit with feedback, i.e. a sequential machine. All gate actions take place in parallel each clock cycle. Gates and wire segments are laid out on ν layers, where ν is fixed and at least two. Each layer is a convex region of the plane, partitioned into a rectilinear arrangement of square cells of side $\lambda > 0$ (Manhattan geometry). Each gate, input port and output port occupies a single cell, and each gate computes some two-argument Boolean function. Each wire consists of a connected set of straight line segments of cells. The segments of two distinct wires cannot lie in adjacent cells. The time for a signal to propagate along a wire is independent of the length of the wire. (This assumption is perhaps the most controversial of those we will describe. If one takes account of the resistances of wires, then one is led to a model in which the time for a signal to propagate is proportional to the length of the wire. This model is considered in [75]. If one also takes into account the capacitive and inductive properties of wires, then one is led to a model in which the propagation time is quadratic in the length of the wire.) The input and output ports are on the boundary of the chip. The input/output behaviour of the chip is defined by a schedule giving the times and locations for inputs and outputs. The schedule is oblivious, i.e. it is the same for all possible values of the inputs. Inputs are supplied only once.

The main complexity measures considered are the chip *area* A and the computing *time* T. For A we measure the number of λ^2 units on a layer, for T we take the number of clock cycles spent in the computation. Since synchronous circuits can be pipelined, another important complexity measure in some contexts, is the *period* P. A pipelined circuit can receive input to a new problem instance to be solved before it has completed the solution of the previous one. P is the number of clock cycles between the first input cycle of two consecutive problem instances. Besides area, time and period, the *switching energy* E is another complexity measure of interest in some VLSI chip computations [240]. Every cell is said to consume one unit of switching energy whenever it changes its state from 0 to 1, or vice versa, in a clock cycle. The total switching energy E consumed by a chip in the course of a computation is closely related to the heat dissipation of the chip for that computation. The various complexity measures A, T, P and E can be combined in a number of ways to form composite complexity measures which capture the various tradeoffs which are possible in the design of VLSI algorithms. The most widely studied of such composite measures is AT^2, which captures an important tradeoff between area and time in VLSI computations.

In 1979, Thompson [357] introduced the above VLSI model of computation and demonstrated that, for the problem of computing the Fast Fourier Transform (FFT) on n inputs, the chip area A and time T must satisfy $AT^2 = \Omega(n^2)$. In the two or three years following the publication of [357], the goal of establishing AT^2 lower bounds for other important problems was pursued by a large number of researchers, and many such results were obtained using "crossing sequence" or information flow arguments. Following this work, Savage [332] observed that, in fact, a number of those results could have been obtained as a fairly direct consequence of earlier results on the existence of small separators in planar graphs [248, 249], and on the simulation of sequential machines by combinational circuits without feedback loops [330]. We will describe how AT^2 lower bounds can be established in this way. Before doing so, we need to define the notion of a Boolean circuit [106, 394], and of a planar circuit [106].

Definition 5.1 *Let $B = \{0,1\}$, $B_{n,m} = \{f : B^n \to B^m\}$, and $X_n = < x_1, x_2, \ldots, x_n >$ be a set of formal arguments. A* <u>*Boolean circuit*</u> *for $f \in B_{n,m}$ is a directed acyclic graph with n input nodes (in-degree 0) $I_1, I_2, \ldots, I_n$ where I_i is associated with x_i, and m output nodes $O_1, O_2, \ldots, O_m$ where O_i corresponds to the i^{th} output function f_i. Every node, other than the input nodes, corresponds to a gate. Gates have in-degree two and compute some function $g \in B_{2,1}$ of their left and right inputs. Let π be a finite region of the plane bounded by a simple closed curve γ. A* <u>*planar Boolean circuit*</u> *for $f \in B_{n,m}$ is a Boolean circuit embedded on the surface π with no crossing edges. The input nodes and output nodes must be on the boundary γ, and should appear in the cyclic order $< I_1, I_2, \ldots, I_n, O_m, O_{m-1}, \ldots, O_1 >$.*

Definition 5.2 *The* <u>*size*</u> *of a circuit or planar circuit is the total number of gates. For $f \in B_{n,m}$, let $C(f)$ denote minimum size of any circuit for f and $PC(f)$ denote minimum size of any planar circuit for f. For the various VLSI complexity measures A, T, AT, AT^2 etc., we will use the same convention, e.g. $AT^2(f)$ will denote the minimum AT^2 complexity of any VLSI chip for f.*

Theorem 5.3 (Savage [332]) *For all $f \in B_{n,m}$, $C(f) = O(AT(f))$*

Proof. This is a restatement of an earlier result by Savage [330]. It is obtained by constructing a Boolean circuit for f from T copies of a VLSI chip for f. A stack of T copies of the chip is formed, and where loops exist on the chip in flip-flops, the loops are broken and the signals fed to the corresponding position on the chip above. A Boolean circuit of size $O(AT)$ is thus created that computes the same function as the chip □

We can also relate circuit size and planar circuit size in the following way.

Theorem 5.4 *For all $f \in B_{n,m}$, $PC(f) = O((C(f))^2)$*

For a proof of Theorem 5.4, see [4] or [273]. By combining Theorems 5.3 and 5.4 we can show that $PC(f)$ is at most $A^2T^2(f)$. Savage [332] showed that the following stronger results could be obtained.

Theorem 5.5 (Savage [332]) *For all* $f \in B_{n,m}$, $PC(f) = O(AT^2(f))$ *and* $PC(f) = O(A^2T(f))$

The proof of these two upper bounds is again based on the idea of making T copies of the chip, breaking loops, and feeding signals between the copies. However, to achieve planarity, one must ensure that wires do not overlap and that at most one pair of wires cross at a point. In the AT^2 upper bound this is done by expanding the chip area by a factor of T in each dimension, and then stacking the copies with an incremental diagonal shift proportional to λ. Two-input two-output planar crossover circuits [249, 269] can then be used to produce the planar circuit. In the A^2T upper bound, the T copies of the chip are placed side by side in the plane, and crossings on the chip and those created by the wires connecting them are replaced by crossovers to produce a planar circuit. For more details, see [332].

For almost all Boolean functions, tight bounds on the various VLSI complexity measures can be easily obtained.

Theorem 5.6 *For all* $f \in B_{n,1}$, *the following upper bounds can be simultaneously achieved,* $A(f) = O(2^n)$, $T(f) = O(n)$ *and* $P(f) = O(1)$.

Proof. Let $f_{ij} = f(x_1, x_2, \ldots, x_{n-2}, i, j)$ and note that $f = (((f_{11} \wedge x_{n-1}) \vee (f_{01} \wedge \bar{x}_{n-1})) \wedge x_n) \vee (((f_{01} \wedge x_{n-1}) \vee (f_{00} \wedge \bar{x}_{n-1})) \wedge \bar{x}_n)$. This definition yields the recursive VLSI layout for f shown in Figure 1.

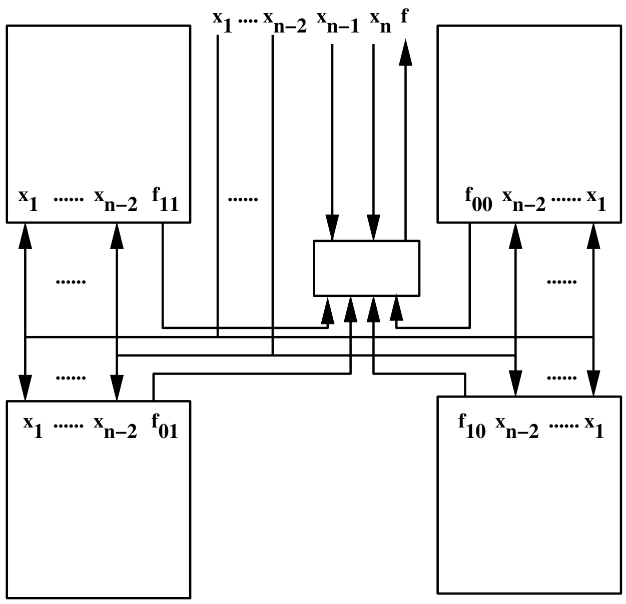

Figure 1. A universal VLSI chip

This layout establishes the bounds stated in the theorem. We have $A(f) \leq 4A(f_{ij}) + O(n^2)$ which yields $A(f) = O(2^n)$, and $T(f) \leq T(f_{ij}) + O(1)$ which gives $T(f) = O(n)$. The construction can also be used in a pipelined manner to give $P(f) = O(1)$ □

The bounds in Theorem 5.6 are tight for most Boolean functions.

Theorem 5.7 *For almost all $f \in B_{n,1}$, $A(f) = \Omega(2^n)$ and $T(f) = \Omega(n)$.*

The lower bound on area can be established by noting that the number of distinct chips with area a is at most c^a for some constant c. Since there are 2^{2^n} distinct functions in $B_{n,1}$, the lower bound follows [201]. A stronger result, showing that for almost all $f \in B_{n,1}$, $PC(f) = \Theta(2^n)$, is established in [270]. The lower bound on time in Theorem 5.7 follows from the well known lower bound on the circuit depth of almost all Boolean functions [106, 394].

Although we have tight bounds on the VLSI complexity of almost all Boolean functions, our real interest is, of course, in the complexity of specific functions of practical importance. Lower bounds on the VLSI complexity of specific Boolean functions can be established in a number of ways. We will now describe two simple lower bounds which can be obtained from a consideration of the input/output limitations and of the memory limitations of VLSI chips.

We begin by showing a simple input/output based lower bound. Let $COUNT^{(n)} \in B_{n,log\ n}$ denote the Boolean function corresponding to the problem of computing the $(log\ n)$-bit binary sum of n Boolean inputs.

Theorem 5.8 $AT^2(COUNT^{(n)}) = \Omega(n\ log\ n)$

Proof. Consider a VLSI chip which has area a and computes $COUNT^{(n)}$ in t clock cycles. If n/q is the maximum number of bits input in any one clock cycle, then we must have $a \geq n/q$ and $t \geq q + log\ (n/q)$, since a tree from n/q input ports to an output port must have area at least n/q and height at least $log\ (n/q)$. Therefore, at^2 must be at least $min\{nq + 2nlog(n/q) + (n/q)log^2(n/q)\}$, where q ranges from 1 to n. This gives $AT^2(COUNT^{(n)}) = \Omega(n\ log\ n)$ since the minimum is obtained for $q \approx log\ n$ □

A reasonably efficient VLSI implementation of $COUNT^{(n)}$ can be obtained in the following way. We divide the inputs into k blocks, each of size n/k. $COUNT^{(n/k)}$ is computed for each block using a serial counter (area $log(n/k)$, time n/k). The resulting k numbers, each $log(n/k)$ bits, are then added using a balanced binary tree of full adder chains (area $k\ log^2 n\ log(n/k)$, time $log\ k +$ log(n/k)). Choosing $k \approx (n/log\ n)$, we obtain a construction with area $O(n\ log\ n\ loglog\ n)$ and time $O(log\ n)$.

We now show how the memory limitations of VLSI chips can be used to prove lower bounds. Let $SORT^{(n,k)} \in B_{nk,nk}$ denote the Boolean function corresponding to the problem of sorting n k-bit binary numbers. Let X^i, $1 \leq i \leq n$, denote the i^{th} input number and x^i_p, $1 \leq p \leq k$, denote the p^{th} most significant bit of X^i, i.e. $X^i = \sum_{p=1}^{k} x^i_p * 2^{k-p}$. Let Y^i, $1 \leq i \leq n$, denote the i^{th} output number and define y^i_p in the same way as x^i_p.

Theorem 5.9 $A(SORT^{(n,k)}) = \Omega(n)$ *for $k > log\ n$.*

Proof. We first show that every x^i_k must be input before any y^j_k is output. If not, then there must be some r, s, such that y^r_k is output before, or when, x^s_k is input. Excluding X^s, let $r-1$ of the input numbers be 0, and $n-r$ of them be 1. Let $x^i_s = 0$ for all $i < k$. If $y^r_k = 1$, then let $x^s_k = 0$ and note that y^r_k is incorrect. Similarly for $y^r_k = 1$.

Let $\alpha_{a_1,a_2,\ldots,a_n}$, $a_i \in \{0,1\}$, denote the chip input where, for all $1 \leq i \leq n$, the $(k-1)$-bit number $< x^i_1, x^i_2, \ldots, x^i_{k-1} >$ has the value $i-1$, and $x^i_k = a_i$. Consider the behaviour of the chip on the family of 2^n inputs $\alpha_{a_1,a_2,\ldots,a_n}$. At the clock cycle just before the first least significant bit y^j_k is output, we know that every x^i_k must have been input, and that, henceforth, all inputs will be the same. We also know that for input $\alpha_{a_1,a_2,\ldots,a_n}$, $y_k^i = a_i$, since the k-bit input numbers X^i are already sorted. If the area of the chip is less than n then there must be two distinct inputs $\alpha_{a_1,a_2,\ldots,a_n}$ and $\alpha_{b_1,b_2,\ldots,b_n}$ in the family of 2^n, for which the state of the circuit is identical. Since all further inputs are the same, all y_k^i outputs will be the same. Contradiction. Hence, the area must be at least n □

The lower bounds in Theorems 5.8 and 5.9 are based on input/output and memory properties of VLSI computations. Stronger results can, in many cases, be obtained through an analysis of the information flow requirements of such computations [357]. In the remainder of this section we describe a number of lower bound results based on this approach. We begin by demonstrating a lower bound on the AT^2 VLSI complexity of $SORT^{(n,k)}$

Theorem 5.10 *$AT^2(SORT^{(n,k)}) = \Omega(n^2)$ for $k > log\ n$.*

Proof. (Note. In proving the theorem we do not make use of our assumption that all input and output ports are on the boundary of the chip.) Suppose the chip looks like Figure 2, a rectangle of height h and width w. We assume, without loss of generality, that $h \leq w$.

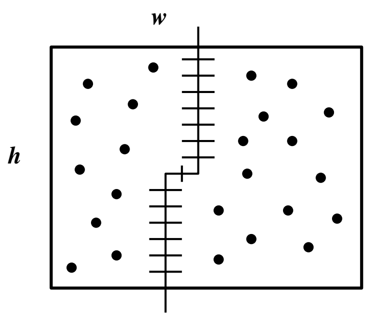

Figure 2. Bisection of VLSI chip

Let us focus, as we did in Theorem 5.9 on the least significant output bits. These bits are output at ports located at various points on the chip, as indicated in Figure 2. If as many as $n/3$ of them are output from the same port then $T \geq n/3$, and the theorem follows immediately. If no single port emits $n/3$ of the bits, then we can bisect the chip by a line, as in Figure 2, which runs between the cells and runs vertically, except possibly for a single jog of one cell width, and for which at least $n/3$ of the least significant output bits are emitted on each side of the line. To determine such a line we slide it from left to right, until the first point at which $n/3$ of the bits are output to the left of the line. If no more than $2n/3$ are output to the left we are done. If not, we start from the top and consider places to jog the line back one cell width to the left. Since no single cell outputs $n/3$ of the bits, we can always find a suitable place to jog the line. There, we will have between $n/3$ and $2n/3$ of the least significant output bits on each side. Now assume, without loss of generality, that at least half of the least significant input bits are read on the left of the line, and let us, by renumbering bits, if necessary, assume that these are $x_k^1, x_k^2, \ldots, x_k^{n/2}$. Suppose also that least significant output bits $y_k^{i_1}, y_k^{i_2}, \ldots, y_k^{i_{n/3}}$ are output on the right. Using an argument very similar to that used in the proof of Theorem 5.9 we can easily pick values for bits other than the least significant ones, so that for all $1 \leq j \leq n/3$, $y_k^{i_j} = x_k^j$. Therefore, at least $n/3$ bits must cross the line for the chip to correctly compute the outputs. The maximum number of bits which can cross the line in one clock cycle is $h+1$ (h horizontal and one vertical, at the jog). It follows that $(h+1)T \geq n/3$. Since we assumed $h \leq w$, we have both $hT = \Omega(n)$ and $wT = \Omega(n)$. Therefore, $hwT^2 = \Omega(n^2)$ which proves the theorem since $hw = A$ □

Theorem 5.10 is proved for the (geometric) VLSI model introduced at the start of the section. However, as noted earlier, AT^2 lower bounds can, in fact, be established in a graph theoretic setting as lower bounds on planar circuit size. An important result in the area of computational complexity is the following planar separator theorem.

Theorem 5.11 (Lipton and Tarjan [248]) *Let G be any n-vertex planar graph with non-negative vertex costs summing to no more than one. Then the vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B, neither A nor B has total vertex cost exceeding 2/3, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.*

Let $BC^{(n)} \in B_{2n,2n-1}$ denote the Boolean function corresponding to the problem of computing the convolution of two n-bit Boolean sequences, $IM^{(n)} \in B_{2n,2n}$ denote the Boolean function corresponding to the problem of computing the product of two n-bit integers, and $MM^{(n)} \in B_{2n^2,n^2}$ denote the function corresponding to the problem of computing the product of two $n \times n$ Boolean matrices. Using the planar separator theorem, Lipton and Tarjan [249] proved the following lower bounds on planar circuit size.

Theorem 5.12 *$PC(BC^{(n)}) = \Omega(n^2)$, $PC(IM^{(n)}) = \Omega(n^2)$ and $PC(MM^{(n)}) = \Omega(n^4)$*

These results are established by showing that in any planar circuit for Boolean convolution (integer multiplication, matrix multiplication) there must be an information flow of $\Omega(n)$ bits ($\Omega(n)$, $\Omega(n^2)$ bits respectively) from the input nodes to the output nodes. Any separator must therefore be of size $\Omega(n)$ ($\Omega(n)$, $\Omega(n^2)$ resp.) and thus by Theorem 5.11, any planar circuit must be of size $\Omega(n^2)$ ($\Omega(n^2)$, $\Omega(n^4)$ resp.). From Theorem 5.5 it follows that the lower bounds in Theorem 5.12 are also lower bounds on the AT^2 complexity and the A^2T complexity of those functions. We now describe two simple VLSI chip constructions which show that the lower bounds in Theorem 5.12 for Boolean convolution and matrix multiplication are quite close to the best possible.

Let $X =< x_{n-1}, x_{n-2}, \ldots, x_0 >$ and $Y =< y_{n-1}, y_{n-2}, \ldots, y_0 >$ be two n-bit sequences. The Boolean convolution $BC^{(n)}(X,Y)$ of X and Y is the sequence Z of functions $< z_{2n-2}, z_{2n-3}, \ldots, z_0 >$ where $z_i = \bigvee_{j \geq 0} x_j \wedge y_{i-j}$. A simple recursive construction based on divide and conquer can be used to compute $BC^{(n)}(X,Y)$. Let X_0 denote the subsequence of values x_i in X where $i < n/2$, and $X_1 = X - X_0$. Define Y_0 and Y_1 similarly. Let Z_k, $0 \leq k \leq 3$, denote the subsequence of values z_j in Z, where $kn/2 \leq i < (k+1)n/2$. If we compute $BC^{(n/2)}(X_i, Y_j)$ for all $i, j \in \{0,1\}$ then the required sequences Z_k for the computation of $BC^{(n)}(X,Y)$ can be obtained from those subcomputations, by computing various disjunctions, see Figure 3.

For the recursive geometric layout shown in Figure 3, let H_n denote the vertical height of the whole circuit for $BC^{(n)}(X,Y)$, and W_n denote the horizontal width. Then we have $H_n \leq 4W_{n/2}$ and $W_n \leq H_{n/2} + O(n)$, which gives $H_n = O(n\ log\ n)$ and $W_n = O(n\ log\ n)$. Therefore, the area A_n of the construction is $O(n^2 log^2 n)$. For time, we have $T_n \leq T_{n/2} + O(1)$ which yields $T_n = O(log\ n)$. Therefore, this simple construction shows that $AT^2(BC^{(n)}) = O(n^2 log^4 n)$.

Let A and B be two $n \times n$ Boolean matrices. The product $MM^{(n)}(A,B)$ of A and B is the $n \times n$ matrix C, where $c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$. To produce a VLSI chip for $MM^{(n)}$ we can again

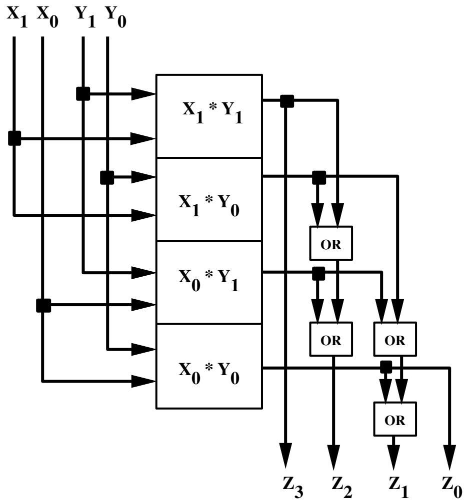

Figure 3. VLSI chip for convolution

use a recursive divide and conquer strategy, this time on partitioned matrices. To compute C, we partition each of the matrices A and B into four $(n/2) \times (n/2)$ submatrices, compute eight products on those submatrices, and then form disjunctions of the resulting matrices, see Figure 4.

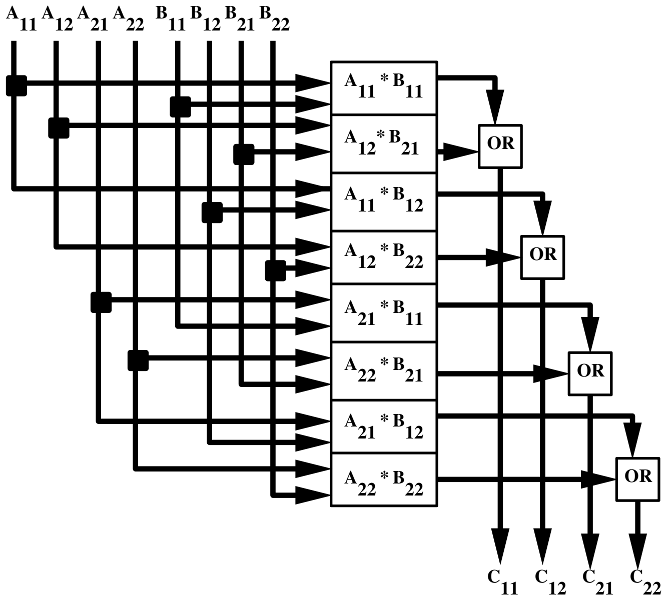

Figure 4. VLSI chip for matrix multiplication

For the layout shown in Figure 4, we can again let H_n denote the vertical height of the whole circuit for $MM^{(n)}(A, B)$, and W_n denote the horizontal width. This time we have $H_n \leq 8W_{n/2}$ and $W_n \leq H_{n/2} + O(n^2)$, which gives $H_n = O(n^2)$ and $W_n = O(n^2)$. Therefore, the area A_n of the construction is $O(n^4)$. For time, we again have $T_n \leq T_{n/2} + O(1)$ which gives $T_n = O(log\ n)$. Therefore, we have shown that $AT^2(MM^{(n)}) = O(n^4 log^2 n)$. In the next section we will describe a systolic algorithm for matrix multiplication which can be realised as a VLSI chip with area $O(n^2)$ and time $O(n)$. This systolic design shows that the lower bound in Theorem 5.12 for matrix multiplication is, in fact, optimal.

The AT^2 lower bounds which follow from Theorem 5.12 are for multiple output functions and assume that the inputs are read once only. Under this read once condition, lower bounds of the form $AT^2 = \Omega(n^2)$, where n is the number of inputs, have been obtained for a large number of important multiple output functions. They include Fourier transforms [357, 358], sorting [359], integer multiplication [3, 56, 249], matrix multiplication [249, 331], matrix inversion [331], transitive closure of a Boolean matrix [331], integer powers and reciprocals [332] and various transitive functions [390].

In [408], Yao introduced a general framework for the analysis of the communication complexity of problems. Informally, the communication complexity of a function f is the minimum amount of information that has to be exchanged between two cooperating processors that each have a partial knowledge of the input, and that want to aid each other in computing f [81, 184, 408]. In the VLSI context, one can think of the communicating processors as the two halves of the chip, when divided by a bisecting line as in Figure 2. The communication complexity corresponds to the minimum possible information flow between the two halves. Using the approach of communication complexity analysis, a number of lower bounds on the AT^2 complexity of single output Boolean functions have also been established [6, 227, 238, 240, 332, 333, 348]. The functions include graph isomorphism [409], evaluation of propositional formulae [54], recognition of a context-free language [54, 247], pattern matching [247], testing integer factorisation [247], testing matrix factorisation [332] and testing whether one matrix is the transitive closure of another [332]. In fact, Savage [332] shows that a lower bound can be obtained for every predicate (single output Boolean function) associated with a multiple output function for which such lower bound results have been derived. An excellent account of the methods used for establishing lower bounds on the VLSI complexity of predicates can be found in [238] and [240].

VLSI chips that can read inputs at multiple times and places have been considered in [194, 333, 409]. Yao [409] studied the function $x + y * z$, for integers over a finite field F. He showed that if a compact binary representation of F is used, then the lower bound $AT^2 = \Omega(n^2)$ holds for $n = log_2 \mid F \mid$. If no limit on the redundancy of the representation is prescribed, then he shows that we must have $AT^2 = \Omega(n^{4/3})$. Kedem and Zorat [194] show that, for the cyclic shift function on n inputs, $AT^2 = \Omega((n/\mu)^2)$ when inputs are read μn times. Savage [333] establishes lower bounds on the complexity of matrix multiplication and related problems such as LU decomposition, for the multiple read model.

The design of efficient VLSI algorithms for important problems has been widely studied over the last decade. A good general introduction to this area can be found in [369]. Some examples of the kinds of problems which have been investigated are: addition [55, 57], multiplication [3, 56, 282], division [281, 283], square root [281], convolution [26], Fourier transform [358], counting [84], sorting [41, 42, 88, 223, 359], merging [25], dictionaries [14, 297], graph computations [154, 183] and regular expressions [118]. Although many "paper designs" for VLSI systems have been produced, relatively few of these have actually been implemented as chips. The main reason has been the prohibitive time and cost of custom VLSI design and fabrication. As noted earlier in the paper, the availability of field-programmable gate arrays will, in the future, enable many such designs to be realised quickly and cheaply.

In our consideration of VLSI systems we have emphasised efficiency considerations, and have demonstrated that there is now a well developed theory to guide the design of efficient VLSI circuits. There has also been much work, in recent years, on the development of techniques for the formal verification of VLSI systems. For an appreciation of the various approaches in this area see, for example, [87, 275, 276, 346].

6. Systolic Algorithms and Architectures

In [207], Kung has persuasively argued the case for the design of parallel algorithms which carefully balance the input/output, communication and processing in parallel computations to avoid bottlenecks and hence increase throughput. Since the appearance of the early papers by Kung and his colleagues, a whole subfield of parallel computing has developed, based around the concept of systolic computation. In this section we will describe several simple systolic algorithms, and then briefly discuss some of the more general issues involved in the field of systolic computation. A systolic parallel algorithm typically has the following properties

- operations are pipelined to balance input/output, communication and processing
- there are only a few types of simple cells
- data and control flows are simple and regular
- only local communication is required
- computation proceeds synchronously

The synchronous property is not an essential feature and, indeed, many asynchronous systolic (or wavefront) algorithms have been developed [214, 215].

6.1. Convolution

In [207], Kung presents a family of systolic designs for the convolution problem. The *k-convolution problem* is defined as follows. Given a sequence of weights $W =< w_0, w_1, \ldots, w_{k-1} >$ and an input sequence $X =< x_0, x_1, x_2, \ldots >$, compute the output sequence $Y =< y_{k-1}, y_k, y_{k+1}, \ldots >$, where $y_i = \sum_{j=0}^{k-1} a_j * x_{i-j}$. The k-convolution problem is representative of a wide class of computations which are well suited to systolic designs. The problem can be viewed as a problem of combining two data streams, W and X, in a certain manner (for example, as in the above equation) to form a resultant data stream Y. This type of computation is common to a number of problems such as filtering, pattern matching, correlation, interpolation, polynomial evaluation, (including discrete Fourier transforms), and polynomial multiplication and division. For example, if multiplication and addition are interpreted as comparison and conjunction respectively, then the convolution problem becomes the pattern matching problem [121]. In computing the k-convolution, each input x_i must be multiplied by each of the k weights in W. If the x_i is input separately from memory for each multiplication then, for large k, memory bandwidth will become a bottleneck, preventing a high performance solution. A systolic algorithm resolves this input/output bottleneck by making multiple use of each x_i fetched from the memory.

Kung [207] describes two systolic designs for the convolution problem in which each of the inputs in X is simultaneously broadcast to all the cells in the systolic array. Such global data communication, although convenient, requires the use of a bus or some kind of tree network. As the number of cells increases, these bus or network structures will themselves become a bottleneck, limiting the performance which can be achieved. Noting this fact, Kung then describes four further systolic designs for the convolution problem, all of which require no global data communications whatsoever. Potentially, these four designs can be scaled to an arbitrarily large number of cells without loss of efficiency. We will now briefly describe two of these four designs. In both of them, each of the cells in the array is associated with a weight in W. In one, the inputs and results move in opposite directions. In the other, the inputs and results move in the same direction, but at different speeds.

Let $[m \ldots n]$ denote the sequence of integers $< i \mid m \leq i \leq n >$ and $[m \ldots]$ denote $< i \mid m \leq i >$. `par endpar` will be used to denote parallel iteration over a set, and `seq endseq` will be used to denote sequential iteration over a sequence. The k-convolution problem can be solved by the following parallel program.

`par` $i \in [(k-1)\ldots]$
$y_i := 0;$
`seq` $j \in [0\ldots(k-1)]$
$y_i := y_i + (w_{k-1-j} * x_{i+j-k+1})$
`endseq`
`endpar`

We will use this program as the basis for our systolic design. We begin by introducing an extra index for the y variables, which enables us to replace the assignments by equalities. This yields the following set of definitions (or single assignment program).

`For all` $i \geq k-1,\ 0 \leq j \leq k-1,$
$y_i^k = 0$
$y_i^j = y_i^{j-1\ (mod\ (k+1))} + (w_{k-1-j} * x_{i+j-k+1})$

The k-convolution problem then corresponds to the computation of y_i^{k-1}. From this set of definitions, we can obtain a linear systolic parallel algorithm for the computation of the y values, in which each y_i^j is computed in cell P_{k-1-j} during phase $i+j-k+1$ of the computation. In this way we can avoid having to broadcast the weights in W (they can be held at the corresponding cell), and the propagation of y values requires only local communication between each cell P_{k-1-j} and its neighbours P_{k-2-j} and P_{k-j}. However, the algorithm still requires broadcasting of $x_{i+j-k+1}$ to all cells during phase $i+j-k+1$. To eliminate this broadcasting we can redefine the values in the following way.

`For all` $i \geq k-1,\ 0 \leq j \leq k-1,\ l \geq 0,$
$x_l^l = x_l$
$y_i^k = 0$
$x_l^{l+j+1} = x_l^{l+j}$
$y_i^j = y_i^{j-1} + (w_{k-1-j} * x_{i+j-k+1}^{2i+j-2(k-1)})$

Now if we compute y_i^j and $x_{i+j-k+1}^{2i+j-2(k-1)}$ in cell P_{k-1-j} during phase $2i+j-2(k-1)$ then we have a fully localised algorithm with only nearest neighbour communication. This algorithm can now be directly implemented on a linear systolic array with bidirectional dataflow. The systolic array consists of a linear array of k computation cells P_i, $0 \leq i < k$, and a "host" cell P_k which supplies input and receives output. For each $0 \leq i \leq k$, we have a unidirectional channel t_i from P_i to P_{i-1}, and a unidirectional channel b_i from P_{i-1} to P_i. (Note. All cell and channel indices are taken modulo $(k+1)$.) At the start of the computation, the host cell inputs the values $x_0, x_1, x_2, \ldots$ to the array, on alternate clock cycles, using the channel b_0. These values are propagated through the array, on successive clock cycles, using the b channels. Once x_0 arrives at P_{k+1}, the host begins to input zero values to the array, corresponding to $y_{k-1}^k, y_k^k, y_{k+1}^k, \ldots$, on alternate clock cycles, using the channel t_k. The array of cells P_i, $0 \le i \le k-1$, now perform the computation given by the above definition. Each cell P_i inputs an x value on b_i, a y value on t_{i+1}, and computes the new y value. At the next clock cycle, it outputs the x value on b_{i+1} and the new y value on t_i. The set of even numbered cells and the set of odd numbered ones compute on alternate cycles. The resulting y_i^{k-1} values are returned to the host using channel t_0.

In our second systolic design for k-convolution, the inputs and results move in the same direction, but at different speeds. The design is based on the following parallel program.

```
par i ∈ [(k − 1)...]
y_i := 0;
seq j ∈ [0...(k − 1)]
y_i := y_i + (w_j * x_{i−j})
endseq
endpar
```

As before, we introduce an extra index for the y variables, and replace the assignments by equalities. This yields the following set of definitions.

For all $i \ge k-1$, $0 \le j \le k-1$,

$$y_i^k = 0$$

$$y_i^j = y_i^{j-1 \pmod{(k+1)}} + (w_j * x_{i-j})$$

The k-convolution problem then corresponds to the computation of y_i^{k-1}. From this set of definitions, we can obtain a linear systolic parallel algorithm for the computation of the y values, in which each y_i^j is computed in cell P_j during phase $i+j-k+1$ of the computation. We again avoid having to broadcast the weights in W by holding them at the corresponding cell and, as before, the propagation of y values requires only local communication between each cell P_j and its neighbours P_{j-1} and P_{j+1}. The movement of x values is simpler in this design. The x values need only be input to P_0 and then propagated from P_i to P_{i+1} during every alternate phase of the computation. This algorithm can be directly implemented on a linear systolic array with unidirectional dataflow. The systolic array again consists of a linear array of k computation cells P_i, $0 \le i < k$, and a "host" cell P_k which supplies input and receives output. For each $0 \le i \le k$, we have a unidirectional channel t_i from P_{i-1} to P_i, and a unidirectional channel b_i from P_{i-1} to P_i. (Again, all cell and channel indices are taken modulo $(k+1)$.) At the start of the computation, the host cell inputs the values $x_0, x_1, x_2, \ldots$ to the array, on successive clock cycles, using the channel b_0. These x values are then propagated from P_i to P_{i+1}, on every second clock cycle, using the b channels. When x_{k-1} is due to be input, the host begins to input input zero values to the array, corresponding to $y_{k-1}^k, y_k^k, y_{k+1}^k, \ldots$, on successive clock cycles, using the channel t_0. The array of cells P_i, $0 \le i \le k-1$, now perform the computation given by the above definition. At time step s, each cell P_i inputs an x value on b_i, a y value on t_i, and computes the new y value. At the next clock cycle, it outputs the new y value on t_{i+1}, and outputs on b_{i+1} the x value which it received at time step $s-1$. Therefore, the y values travel through the array at twice the speed of the x values. The resulting y_i^{k-1} values are returned to the host using channel t_k.

6.2. Sequence Comparison

Let $X_m = x_1 x_2 \ldots x_m$ and $Y_n = y_1 y_2 \ldots y_n$ be two sequences. The sequence comparison problem is the problem of determining the minimum number of insertions and deletions required to change X_m into Y_n. We assume that each insertion or deletion costs 1. Very large instances of this problem arise in the area of molecular biology where the strings correspond to nucleic acid sequences [329]. The edit distance is a measure of the similarity of the two sequences When $m = 0$ we have $Cost(X_m, Y_n) = n$. Similarly, $n = 0$ gives $Cost(X_m, Y_n) = m$. If $m > 0$, $n > 0$ then we have

$$
\begin{aligned}
Cost(X_m, Y_n) &\leq Cost(X_{m-1}, Y_{n-1}) && \text{if } \ x_m = y_n \\
Cost(X_m, Y_n) &\leq 1 + Cost(X_{m-1}, Y_n) && [\text{ delete } x_m \text{ }] \\
Cost(X_m, Y_n) &\leq 1 + Cost(X_m, Y_{n-1}) && [\text{ insert } y_n \text{ }]
\end{aligned}
$$

Using dynamic programming we can tabulate the values of $Cost$ in a straightforward way. As an example, consider the computation of $Cost(DEFINE, DESIGN)$.

	∅	D	E	F	I	N	E
∅	[0]	1	2	3	4	5	6
D	1	[0]	1	2	3	4	5
E	2	1	[0]	[1]	2	3	4
S	3	2	1	[2]	3	4	5
I	4	3	2	3	[2]	3	4
G	5	4	3	4	[3]	4	5
N	6	5	4	5	4	[3]	[4]

The number below x_i and to the right of y_j is $Cost(X_i, Y_j)$. The chain of bracketed numbers indicate that we should perform the following sequence of edits to change DEFINE into DESIGN: Delete F, insert S, insert G, delete E. The sequence comparison problem can, therefore, be solved by the following parallel program, where $c_{i,j}$ corresponds to $Cost(X_i, Y_j)$.

par $i \in [0 \ldots n]$
$c_{0,i} := i$
endpar
par $i \in [0 \ldots m]$
$c_{i,0} := i$
endpar
seq $d \in [0 \ldots (m+n)]$
par $i \in [min\{0, d-m\} \ldots min\{d, n\}]$
$c_{i,d-i} := min\{c_{i,d-i-1}, c_{i-1,d-i}\} + 1;$
if $x_i = y_{d-i}$ **then** $c_{i,d-i} := min\{c_{i,d-i}, c_{i-1,d-i-1}\}$
endpar
endseq

From this program we can easily obtain a linear systolic array algorithm with bidirectional dataflow for the sequence comparison problem. In this systolic array, $c_{i,d-i}$ is computed in cell P_{2i-d} during phase d of the computation. The algorithm requires at most $O(m+n)$ cells and runs in time $O(m+n)$. The following is an informal outline of the construction. The two sequences are compared as they pass through the array moving in opposite directions. Each processor P_k has input and output channels (In.L,Out.L) to the processor on the left, P_{k-1}, and similar channels (In.R,Out.R) to the processor on the right, P_{k+1}. We input the sequences $y_1\,1\,y_2\,2\,y_3\,3\,\cdots\,y_n\,n$ and $x_1\,1\,x_2\,2\,x_3\,3\,\cdots\,x_m\,m$ sequentially from the left and right respectively. Once the sequences have been input to the array, the computation proceeds with each processor repeatedly performing the following set of operations every two time steps.

Step 1: In.R(x_i);
In.L(y_j);
Bool:=($x_i = y_j$);
Out.R(y_j);
Out.L(x_i);

Step 2: In.R($c_{i,j-1}$);
In.L($c_{i-1,j}$);

$$c_{i,j} \; = \; minimum \begin{cases} c_{i,j-1}+1 \\ c_{i-1,j}+1 \\ c_{i-1,j-1} \;\; if \; Bool \end{cases}$$

Out.R($c_{i,j}$);
Out.L($c_{i,j}$);

A simple inductive argument shows that the matrix of edit distances has the property that horizontally and vertically adjacent pairs of values always differ by 1. Because of this property it is sufficient to keep the remainder modulo 4 of the matrix entry. If we implement the above systolic algorithm but keep only such remainders then we can still reconstruct the true value of the edit distance from either of the two sequences of values $c_{m,1}, c_{m,2}, \ldots, c_{m,n}$ and $c_{1,n}, c_{2,n}, \ldots, c_{m,n}$ which are output by the array ($c_{i,j} \; = \; Cost(X_i, Y_j) \; mod \; 4$) since

$$c_{m,0} = m \quad , \quad (c_{m,i} = c_{m,i-1} - 1) \vee (c_{m,i} = c_{m,i-1} + 1)$$

$$c_{0,n} = n \quad , \quad (c_{i,n} = c_{i-1,n} - 1) \vee (c_{i,n} = c_{i-1,n} + 1)$$

In matching nucleic acid sequences (four letter alphabet) we need only two bits per character and, as we have seen, two bits per value. This greatly reduces the communication and storage costs and significantly improves performance. Nucleic acid sequence databases are already growing in size at an astonishing rate. The Human Genome Project promises to further accelerate this growth. Special purpose parallel architectures for pattern matching and database searching, such as those described, will be crucial to future developments in this area.

6.3. Molecular Modelling

Molecular modelling is one of the most important practical applications of parallel architectures. Ab initio methods [134, 161, 167] are based on quantum-mechanical calculations of molecular structure. The computation times required for such calculations, even with highly parallel architectures, are at present prohibitive for all but the smallest molecules. However, as the power of parallel machines increases we can expect a much wider use of this approach.

At present, most calculations of the structure and dynamics of proteins, carbohydrates and nucleic acids are carried out using molecular mechanics [271, 349], molecular dynamics [267, 271, 349, 379, 380] and free energy [22] techniques. These methods adopt a force field approach to calculating the potential energy of a molecule in a given conformation. The *AMBER* force field [349] is quite typical. Let $i \Leftrightarrow j$ denote the (symmetric) relation that atom i is bonded to atom j. Then in *AMBER* the potential energy is calculated as a sum of the following types of term.

- Bond energies for all $i, j,\ i \Leftrightarrow j$
- Angle energies for all $i, j, k,\ (i \Leftrightarrow k) \wedge (k \Leftrightarrow j)$
- Torsional energies for all $i, j, k, l,\ (k \Leftrightarrow i) \wedge (i \Leftrightarrow j) \wedge (j \Leftrightarrow l)$
- Coulombic potential for all $i, j,\ (i \not\Leftrightarrow j)$
- Lennard-Jones potential for all $i, j,\ (i \not\Leftrightarrow j)$
- Hydrogen bonding term for all $i, j,\ (i \not\Leftrightarrow j)$

The problem of calculating the potential energy of a molecular conformation is similar in many ways to the sequence comparison problem. They are both instances of "all pairs" computations [347]. For any such computation, a linear systolic array provides a convenient pattern of dataflow. In the case of the potential energy calculation we simply pass one copy of the atomic coordinates and bond information of the structure past another in the array. When the coordinate and bond information for atom i meets that for atom j, we execute the following program.

```
if        i ⇔ j              then  calculate bond term;
                                   for all k,l, (k ⇔ i) ∧ (i ⇔ j) ∧ (j ⇔ l),
                                   calculate torsional terms;
elseif    (i ⇔ k) ∧ (k ⇔ j)  then  calculate angle energy;
                             else  calculate Coulombic energy;
                                   calculate Lennard-Jones potential;
                                   calculate hydrogen bonding term;
```

Let $d_{i,j}$ denote the distance between atoms i, j. The contributions from the Lennard-Jones and hydrogen bonding terms rapidly decline with increasing $d_{i,j}$ and can safely be neglected beyond a certain predefined cut-off distance. Unfortunately, the Coulombic potential does not decline quickly, and it is the calculation of these Coulombic terms for all non-bonded pairs which dominates the energy calculation for large molecules. For a molecule with n atoms, where n is very large, one can use various multipole methods [21, 147, 148, 410] to compute these Coulombic terms using only $o(n^2)$ operations, although the dataflow involved is rather more complex than for the simple systolic algorithm, which uses $O(n^2)$ operations.

6.4. Matrix Multiplication

Consider the problem of multiplying two $n \times n$ matrices A, B to produce C. The definition of matrix multiplication provides us with the following parallel program.

```
par < i,j > ∈ [1...n]^2
c_ij := 0;
seq k ∈ [1...n]
c_ij := c_ij + (a_ik * b_kj)
endseq
endpar
```

By adding an extra index to the c variables, we can replace the assignments by equalities, obtaining the following set of definitions.

`For all` $i, j, k \in [1 \ldots n]$
$c_{ij}^0 = 0$
$c_{ij}^k = c_{ij}^{k-1} + (a_{ik} * b_{kj})$

Implementing this definition as a systolic array would require the a and b values to be broadcast. To avoid this, we can add an extra index to those variables, and redefine the matrix product computation in the following way.

`For all` $i, j, k \in [1 \ldots n]$
$a_{ik}^0 = a_{ik}$
$b_{kj}^0 = b_{kj}$
$c_{ij}^0 = 0$
$a_{ik}^j = a_{ik}^{j-1}$
$b_{kj}^i = b_{kj}^{i-1}$
$c_{ij}^k = c_{ij}^{k-1} + (a_{ik}^j * b_{kj}^i)$

The graph of functional dependencies corresponding to this set of definitions is shown in Figure 5 for $n = 3$.

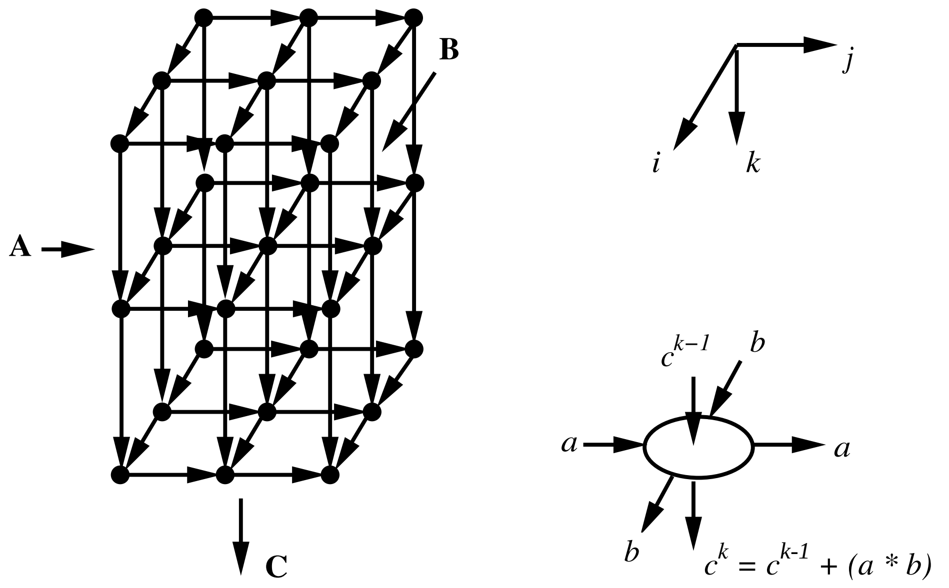

Figure 5. Dataflow for matrix multiplication

This graph shows the dataflow for a three dimensional parallel computation of matrix product. The algorithm requires volume $O(n^3)$ and runs in time $O(n)$. By projecting this graph in one of the three dimensions we can obtain a two dimensional systolic algorithm which requires area $O(n^2)$ and runs in time $O(n)$. For $n = 3$, the algorithm obtained by projecting in the k-direction is shown in Figure 6.

Each cell P_{ij} in the $n \times n$ array is associated with the computation of c_{ij}. The A matrix is input from the left and moves from left to right. The B matrix is input from the top and moves from top to bottom. (The D elements denote timing delays.) At each clock cycle, P_{ij} performs the computation $c_{ij} := c_{ij} + (a_{ik} * b_{kj})$. At the end of the computation, P_{ij} holds the final value of the inner product c_{ij}. The detailed timing and channel requirements

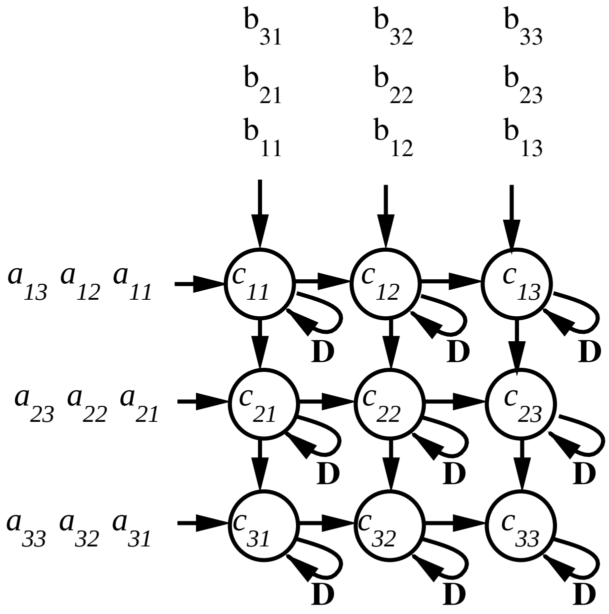

Figure 6. Projection in k-direction

are as follows. $Program_{ij}$ is implemented on processor $P_{i,j}$ in an $n \times n$ array of processors corresponding to the elements of C.

$\underline{Program_{ij}}$

Time step 0	:	$c_{ij} := 0;$
Time step $i+j+t-2,\ 0 < t \leq n$	:	Input(a_{it}, b_{tj}); $c_{ij} := c_{ij} + (a_{it} * b_{tj})$; Output$(a_{it}, b_{tj})$;

To ensure the correct synchronous dataflow we need to do the following. Provide a communication channel from $P_{i,j}$ to $P_{i,j+1}$ and from $P_{i,j}$ to $P_{i+1,j+1}$, supply a_{ij} as input to processor $P_{i,1}$ at time $i+j-1$, and supply b_{ij} as input to processor $P_{1,j}$ at time $i+j-1$. At time step $3n-1$ the product matrix C is stored in the $n \times n$ processor array and can be output using the communication channels. The algorithm obtained by projecting in the j-direction is shown in Figure 7.

Each cell P_{ij} is associated with element a_{ij} from matrix A. The B matrix is input from the right and moves from right to left. The inner product c_{ij} is computed by the cells in column i, and is output at the bottom of the column. The algorithm obtained by projecting in the i-direction is very similar to the one obtained by projecting in the j-direction although, this time, each cell P_{ij} is associated with element b_{ij} from matrix B and the A matrix is input from the left. These three systolic algorithms can be implemented as VLSI chips which compute the Boolean matrix multiplication function in area $O(n^2)$ and time $O(n)$. Combining this upper bound with the lower bound in Theorem 5.12 we obtain

Theorem 6.1 $AT^2(MM^{(n)}) = \Theta(n^4)$

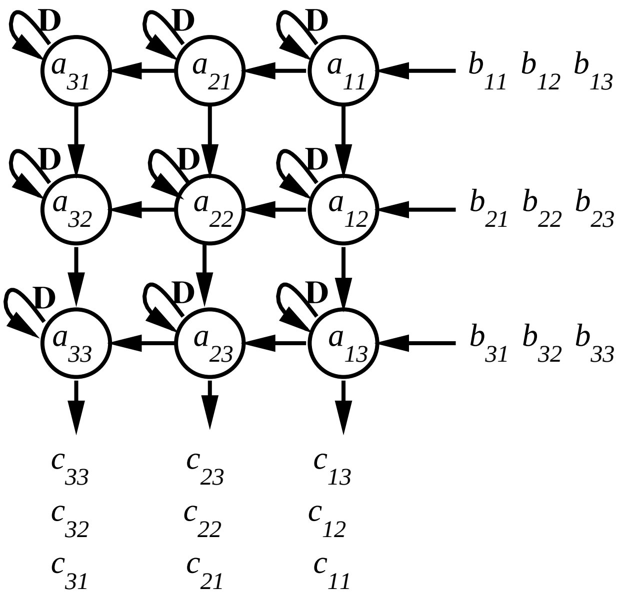

Figure 7. Projection in j-direction

6.5. LU Decomposition

Another very important problem in linear algebra is LU decomposition. We now show how to derive systolic algorithms for this problem. We begin by describing the standard recursive algorithm for the LU decomposition of an $n \times n$ nonsingular matrix A [91]. For $n = 1$ the problem is trivial since we can take $L = I$ and $U = A$. For $n > 1$, we break A into four parts:

$$
\begin{aligned}
A &= \left(\begin{array}{c|ccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right) \\
&= \begin{pmatrix} a_{11} & w^T \\ v & \hat{A} \end{pmatrix}
\end{aligned}
$$

where v is an $(n-1)$-column vector, w^T is an $(n-1)$-row vector, and $\hat{A}$ is an $(n-1)\times(n-1)$ matrix. Then, using elementary matrix algebra, we can factor A as

$$A = \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & \hat{A} - vw^T/a_{11} \end{pmatrix}$$

The 0's in the first and second matrices of the factorisation are row and column vectors, respectively, of size $n-1$. The term vw^T/a_{11}, formed by taking the outer product of v and w and dividing each element of the result by a_{11}, is an $(n-1)\times(n-1)$ matrix, which conforms in size to the matrix $\hat{A}$ from which it is subtracted. The resulting $(n-1)\times(n-1)$ matrix $\hat{A} - vw^T/a_{11}$ is called the *Schur complement* of A with respect to a_{11}. We can now apply the technique recursively to find the LU decomposition of the Schur complement $\hat{A} - vw^T/a_{11}$. Let $\hat{A} - vw^T/a_{11} = \hat{L}\hat{U}$ where $\hat{L}$ is lower triangular and $\hat{U}$ is upper triangular. Then we have

$$A = \begin{pmatrix} 1 & 0 \\ v/a_{11} & \hat{L} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & \hat{U} \end{pmatrix}$$

$$= \quad LU$$

thereby providing our LU decomposition. (Note that because $\hat{L}$ is lower triangular, so is L, and because $\hat{U}$ is lower triangular, so is U.) Transforming this tail recursion to an iteration we obtain the following parallel program.

```
seq k ∈ [1...n]
u_kk := a_kk ;
par
par i ∈ [(k+1)...n]
l_ik := a_ik/u_kk
endpar
par j ∈ [(k+1)...n]
u_kj := a_kj
endpar
endpar
par <i,j> ∈ [(k+1)...n]^2
a_ij := a_ij - (l_ik * u_kj)
endpar
endseq
```

From this program we can produce the following set of definitions.

For all $1 \leq k < i, j \leq n$,

$$\begin{aligned} u_{kk} &= a_{kk} \\ l_{ik} &= a_{ik}^{(k-1)}/u_{kk} \\ u_{kj} &= a_{kj}^{(k-1)} \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - (l_{ik} * u_{kj}) \end{aligned}$$

For $n = 4$, the horizontal sections of the corresponding dataflow graph are shown in Figure 8. (All vertical dataflow is downwards.)

By localising the broadcasts we obtain the following modified set of definitions.

For all $1 \leq k \leq i, j \leq n$,

$$\begin{aligned} u_{k,k,k} &= a_{k,k,k-1} \\ l_{i,j,k} &= a_{i,j,k-1}/u_{i,j,k} \text{ if } j = k, \; l_{i,j-1,k} \text{ otherwise.} \\ u_{i,j,k} &= a_{i,j,k-1} \text{ if } i = k, \; u_{i-1,j,k} \text{ otherwise.} \\ a_{i,j,k} &= a_{i,j,k-1} - (l_{i,j,k} * u_{i,j,k}) \end{aligned}$$

where $a_{i,j,0} = a_{ij}$.

The revised dataflow is shown in Figure 9.

As in the case of matrix multiplication, we can obtain two dimensional systolic algorithms by projecting this dataflow graph in various ways. For $n = 4$, the algorithm obtained by projecting in the i-direction is shown in Figure 10.

In this design, the cells correspond to the u_{kj} values. The matrix A is input from the top, the l values are output from the left, and the u values are held in the array at the end of the computation. An alternative symmetric projection is suggested by the pattern of dataflow in Figure 9. We sketch this symmetric projection for the case $n = 4$ in Figure 11.

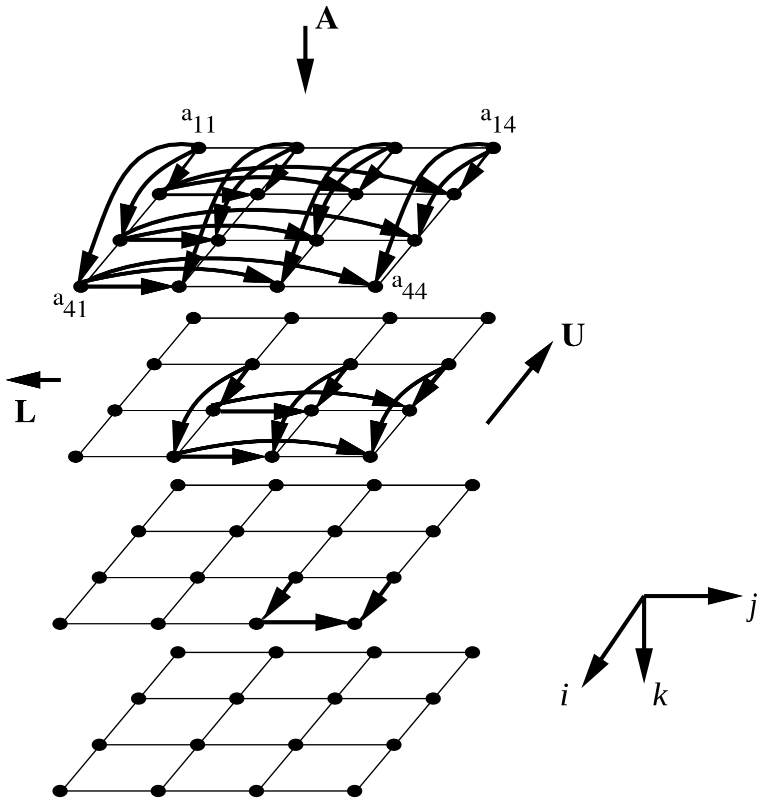

Figure 8. Horizontal dataflow for LU decomposition

The MA cells perform the multiply-and-add operation, those marked $*$ perform multiplication, the one marked / performs division, and those which are unmarked simply pass data. Each channel has an associated delay element for correct timing. For a detailed description of these systolic arrays, see [214].

6.6. Algebraic Path Problem

A *closed semiring* is an algebraic structure $(S, \oplus, \otimes, I_\oplus, I_\otimes)$ with the following properties:

$\oplus$ is a commutative monoid ($\oplus$ satisfies the closure, associative, commutative properties, and has identity element $I_\oplus$).

$\otimes$ is a monoid ($\otimes$ satisfies the closure, associative properties, and has identity element $I_\otimes$).

$\oplus$ is idempotent.

$\otimes$ is right and left distributive over $\oplus$.

For all $s \in S$, $s \otimes I_\oplus = I_\oplus$.

Let $G = (V, A)$ be a directed graph on $\mid V \mid = n$ vertices, in which each $< i, j > \in A$ has an associated weight $s_{ij} \in S$. Define an $n \times n$ matrix M of weights m_{ij} corresponding to the arcs of G: $m_{ij} = s_{ij}$ if $< i, j > \in A$, $I_\oplus$ otherwise. The *Algebraic Path Problem (APP)* is to compute $M^* = \bigoplus_{k=0}^{\infty} M^k$ where matrix product is defined in terms of the two operations $\oplus$ and $\otimes$. (M^0 is the identity matrix with diagonal elements $I_\otimes$). M^*_{ij} gives the "sum" of the

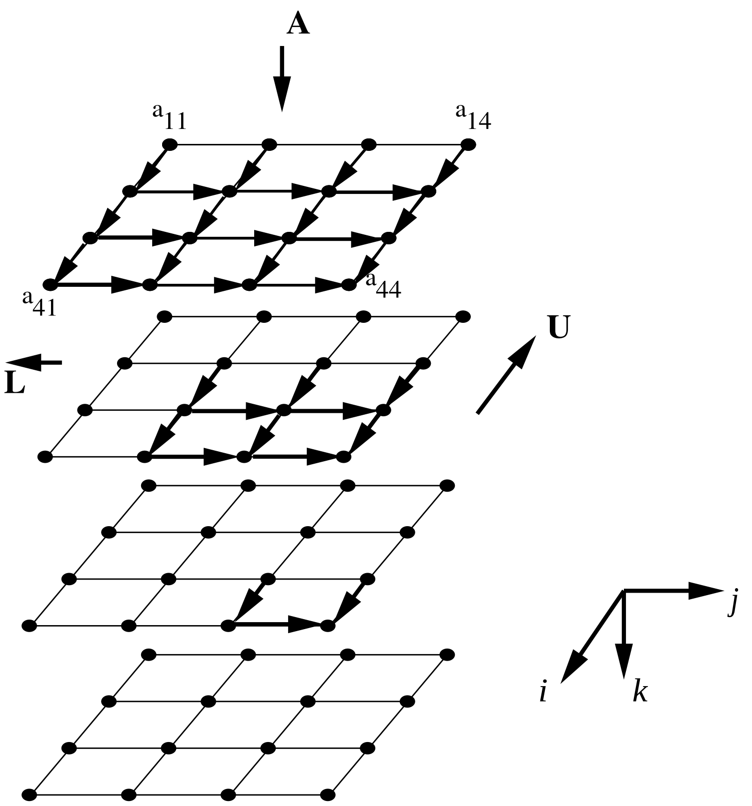

Figure 9. LU decomposition without broadcasting

weights of all directed paths from i to j where the weight of a path is the "product" of the weights of the arcs.

The APP is a problem of major importance in a wide variety of areas and has been extensively studied in recent years. A number of instances of the APP are described in [272]. They include a number of important problems on graphs such as transitive closure [393], shortest paths [117] and the minimum cost spanning tree problem [255]. Noting that node i is connected to node j by a directed path if and only if it is connected by a directed path of length $\leq n-1$, we have

$$\begin{aligned} M^* &= \bigoplus_{k=0}^{n-1} M^k \\ &= (M^0 \oplus M)^{n-1} \\ &= (M^0 \oplus M)^{2^l} \quad \text{for} \quad 2^l \geq n-1 \end{aligned}$$

Therefore, to obtain an efficient parallel algorithm for the computation of the APP on matrix M we need only set the main diagonal to $I_\otimes$ and repeatedly square the resulting matrix until we have a sufficiently large power. For an $n \times n$ matrix M, $O(log\ n)$ squarings will be sufficient. We have seen that matrix multiplication, and hence matrix squaring, can be performed in time $O(n)$ on a systolic array of area $O(n^2)$. Therefore, this approach gives us a way of solving the APP in time $O(n\ log\ n)$ on a systolic array of area $O(n^2)$. The systolic method which we will now describe, shows that time $O(n)$ and area $O(n^2)$ can be achieved.

The APP can be solved using the algorithm of Floyd [117]. Translated into our notation, Floyd's algorithm corresponds to the following parallel program.

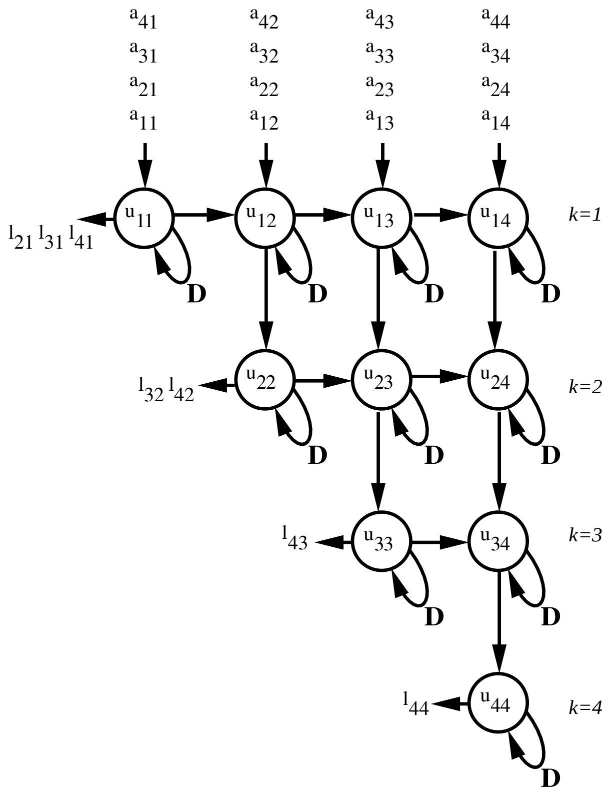

Figure 10. Projection in i-direction

```
seq k ∈ [1...n]
par < i, j > ∈ [1...n]^2
m_ij := m_ij ⊕ (m_ik ⊗ m_kj)
endpar
endseq
```

Replacing the assignments by equalities we obtain the following definitions.

For all $1 \leq i, j, k \leq n$,

$$\begin{aligned} m_{ij}^{(0)} &= m_{ij} \\ m_{ij}^{(k)} &= m_{ij}^{(k-1)} \oplus (m_{ik}^{(k-1)} \otimes m_{kj}^{(k-1)}) \end{aligned}$$

By localising the broadcasts we obtain the following modified set of definitions.

For all $1 \leq i, j, k \leq n$,

$$\begin{aligned} m_{i,j,0} &= m_{ij} \\ r_{i,j,k} &= m_{i,j,k-1} \text{ if } j = k, r_{i,j+1,k} \text{ if } j < k, r_{i,j+1,k} \text{ otherwise} \\ c_{i,j,k} &= m_{i,j,k-1} \text{ if } i = k, c_{i+1,j,k} \text{ if } i < k, c_{i-1,j,k} \text{ otherwise} \\ m_{i,j,k} &= m_{i,j,k-1} \oplus (r_{i,j,k} \otimes c_{i,j,k}) \end{aligned}$$

The c and r variables propagate values along columns and rows respectively. The dataflow graph corresponding to these definitions is shown in Figure 12.

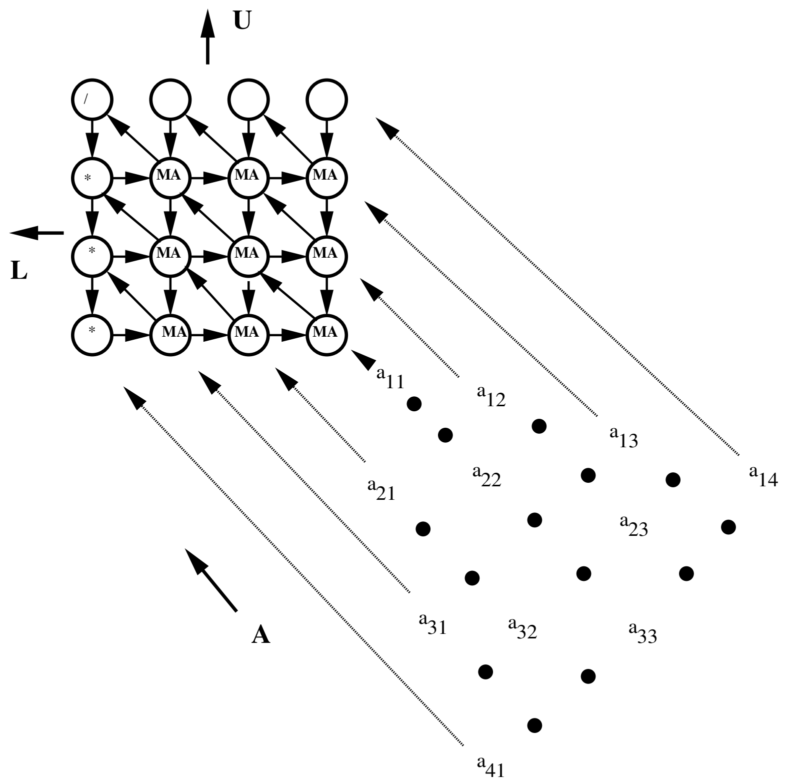

Figure 11. Symmetric i, j, k projection

The pattern of dataflow suggests that we should renumber the nodes, replacing each index (i,j,k) by $((i-k) \; mod \; n, (j-k) \; mod \; n, k)$. Having reindexed the nodes in this way, we can then project in the k-direction to obtain the two dimensional array shown in Figure 13.

All that now remains is to eliminate the spiral connections and the long connection between the m_{11} and m_{nn} nodes. Noting that $m_{i,k,k} = m_{i,k,k-1}$ and $m_{k,j,k} = m_{k,j,k-1}$ one can show that these connections can be eliminated to yield the systolic design shown in Figure 14.

A detailed account of this construction can be found in [216].

6.7. Other Applications

We have described some of the basic problems for which efficient systolic algorithms are known. Over the last decade, a large and diverse range of important applications have been shown to benefit from the systolic approach [120, 121, 150, 205, 207, 211, 213, 214, 258, 268, 276, 277, 279, 291, 309, 317]. The application areas which have been investigated include the following.

Integer and polynomial arithmetic [58, 59, 60]: Multiplication, division, GCD.

Signal and image processing [207, 212, 214, 217, 237, 320]: Filtering, convolution, Fourier transforms, image segmentation.

Linear algebra [137, 214, 311, 317]: Matrix multiplication and inverse, triangularisation, solution of linear systems, LU decomposition, QR decomposition, singular value decomposition.

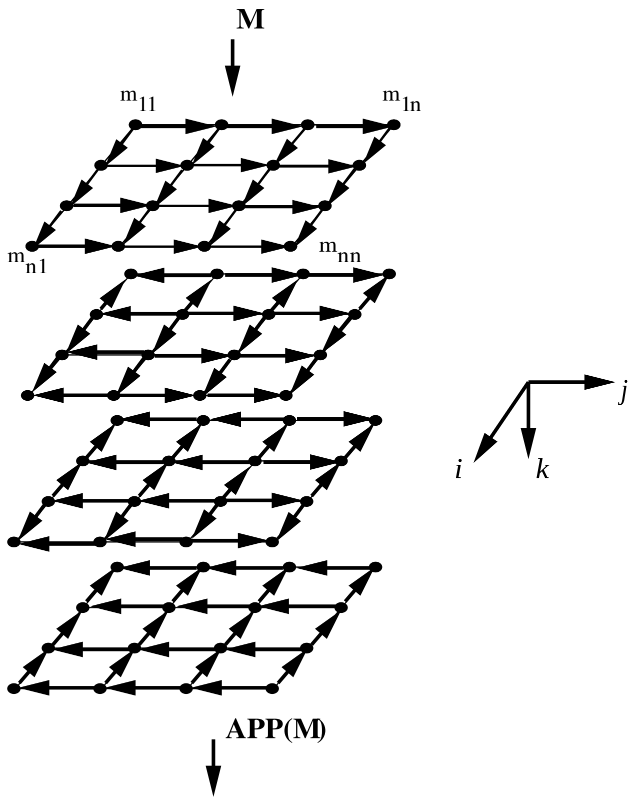

Figure 12. Horizontal dataflow for APP

Computational geometry [13, 74, 78, 95, 190, 370]: Convex hull, contour, visibility, triangulation, proximity, VLSI design rule checking.

Speech processing [126, 405]

Combinatorial algorithms [150, 216, 241, 316, 317, 321, 325]: Dynamic programming, transitive closure, shortest paths.

String processing [12, 73, 122, 180, 246, 271, 319, 356]: String matching, string comparison, longest common subsequence, regular expressions, parsing, data compression.

Data structures [151, 210, 219, 230, 244, 347]: Stacks, priority queues, counters, sorting, set operations, relational databases.

6.8. Systolic Design

Research work in the field of systolic computation over the last decade has not been exclusively concerned with the problem of designing efficient systolic algorithms. It has addressed a whole range of important issues, including

- design and analysis of new, efficient systolic algorithms [207, 258, 317]
- complexity issues in systolic computation [149]
- direct implementation of systolic arrays using custom VLSI, FPGAs, or wafer scale integration [225]

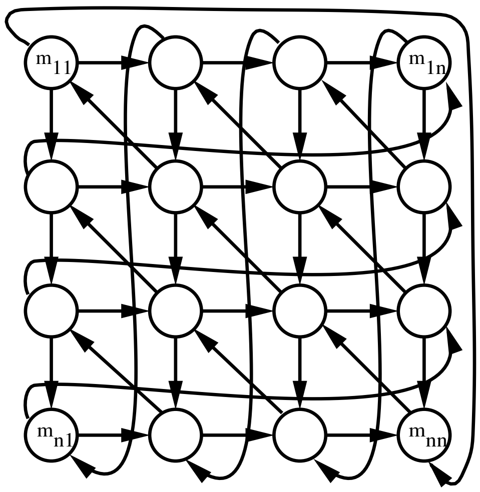

Figure 13. Projection in k-direction

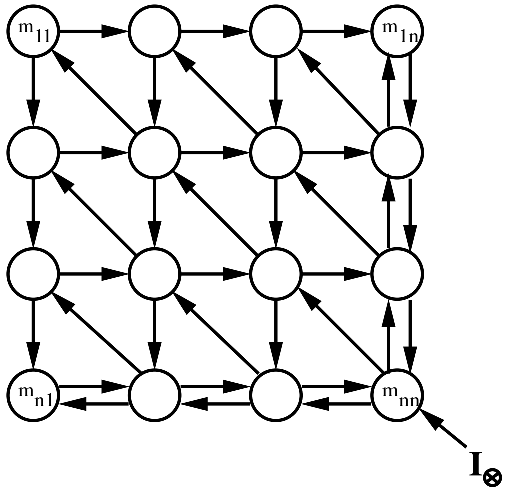

Figure 14. Systolic design for APP

- fault tolerance [209, 225]
- automatic synthesis of systolic designs from high level specifications [67, 72, 77, 177, 189, 214, 236, 250, 288, 308, 309, 310, 311, 314, 326, 346]
- automatic conversion of semi-systolic systems, involving broadcasting or unbounded fan-in, into pure systolic systems with no global data communication [234, 403]
- development of specification and programming languages for systolic computation [69, 77, 221, 346, 391]
- verification of systolic designs [71, 72, 77, 164, 346]
- investigation of asynchronous designs and wavefront arrays [213, 214, 215, 277]
- development of programmable systolic architectures [11, 27, 49, 50, 208]

A major emphasis has been placed on the formal synthesis of systolic designs. In this paper we have described most of the systolic designs informally, using pictures. In order to be able to specify and derive designs in a way that allows one to reason about their correctness, one requires a much more rigorous approach, with appropriate formal notations. This is particularly important in view of the very intricate nature of many of these designs in terms of communication, timing etc. A large number of researchers have investigated this formal synthesis and verification problem, and various design methodologies have been proposed for its solution. We now briefly describe the main techniques used in several of these approaches. A typical situation is where one starts with an initial parallel algorithm that computes a set of functions at all points in some *index space*, for example, the integer lattice points in a subset of Euclidean space. The examples described above are all of this form. If the algorithm is given as a simple loop program (set of recurrence equations) then the ranges of the loop (ranges of equation indices, respectively) define the index space. The body of the loop (set of equations, resp.) defines the functions to be computed by the cell. Given an initial specification in the form of a uniform or affine recurrence equation [192, 311, 314, 326], or in a form that can be easily transformed into one, the synthesis problem reduces to mapping the original index space to a *space-time* domain, i.e. assigning a place and time to each point in the original domain. The two parts of the mapping function are the allocation function (for the space component) and the timing function (for the time component). Such a mapping must satisfy certain constraints to be valid. The constraints are as follows.

- The data dependencies of the original algorithm must be spatially and temporally local, since systolic arrays have only nearest neighbour communication (spatial locality) and a finite memory in each cell. Note that this finite memory constraint mandates temporal locality, since the value used by any cell must have been produced by a neighbour only a finite number of clock cycles ago.

- The transformed dependencies must be uniform over the whole space-time domain, since the processors in a systolic array are identical and have similar interconnections independent of their physical location in the array.

- The mapping must be bijective, i.e. distinct points in the index space must be mapped to distinct points in the space-time domain. If not, the computations from two distinct points in the problem domain would be scheduled on the same cell at the same time, giving rise to a conflict.

- The time component must preserve the dependencies of the original index space, i.e. if a computation is scheduled at some point, then the computation of each of its arguments must be scheduled earlier.

- The space component of the transformed dependencies must correspond to nearest neighbour connections.

For a description of the various methods used to produce such space-time mappings see, for example, [67, 250, 311, 314, 326].

The design, analysis, implementation and verification of systolic systems is a major area of activity within the field of special purpose parallel computing, and rightly so. It is perhaps the most mature subfield in the area, having its own languages, compilers, formal design methodologies and verification methods. Researchers in systolic computation have also succeeded in producing an impressive collection of interesting and efficient algorithms for a wide range of important problems. The early arguments of Kung [207] in support of the systolic approach have undoubtedly been shown, by the experience of the last few years, to have been justified.

7. Cellular Automata

Differential equations form the mathematical basis for most current models of natural systems. Cellular automata may be considered as an alternative, and in some respects, complementary basis for mathematical models of nature. Whereas differential equations are suitable for systems with a small number of continuous degrees of freedom, evolving in a continuous manner, cellular automata describe the behaviour of systems with large numbers of discrete degrees of freedom. Although cellular automata are simple mathematical systems they can exhibit very complex behaviour. The book by Toffoli and Margolus [364] and the collections of papers in [112] and [402] provide a wealth of information on the theoretical and practical aspects of cellular automata.

In a simple case, a cellular automaton consists of a line of cells or sites, each with value 0 or 1. The sequence of site values is the "configuration" of the cellular automaton. These values are updated synchronously in parallel in a sequence of discrete time steps, according to a fixed rule. Denoting the value of a site at position i by a_i, a simple rule gives its new value as $a_i^{(t+1)} = f(a_{i-1}^{(t)}, a_i^{(t)}, a_{i+1}^{(t)})$. Here f is a Boolean function which specifies the rule. In general, the sites in a cellular automaton may have any finite number k of possible values. The rules for updating these sites may depend on values up to any finite distance r away. In addition, the cellular automaton sites may be arranged not on a line, but on a regular lattice in any number of dimensions. Many of the generic features of behaviour are however largely unaffected by such additional flexibility. Cellular automata have a number of basic defining characteristics.

Discrete in space. They consist of a discrete grid of spatial cells or sites

Discrete in time. The value of each cell is updated in a sequence of discrete time steps.

Discrete states. Each cell has a finite number of possible values.

Homogeneous. All cells are identical, and are arranged in a regular array.

Synchronous updating. All cell values are updated in synchrony, each depending on the previous values of neighbouring cells.

Deterministic rule. Each cell value is updated according to a fixed, deterministic rule.

Spatially local rule. The rule at each site depends only on the values of a local neighbourhood of sites around it.

Temporally local rule. The rule for the new value of a site depends only on values for a fixed number of preceding steps (usually just one step)

Each of these characteristics represents a simplifying feature in the basic construction of cellular automata. They make many kinds of analysis and simulation much easier. But they yield few, if any, restrictions on overall cellular automaton behaviour. While each cell is discrete, collections of large numbers of cells may, for example, show effectively continuous behaviour.

It is a common occurrence in nature for a large number of similar elements (e.g. molecules, animals) to interact in such a way as to exhibit large-scale phenomena not present in the individual elements. A simple example is a gas: molecules that individually obey rather simple collision laws give rise to collective behaviour that can be quite varied and complex, including sound waves, vortices, and turbulence. What is interesting from a modelling viewpoint is that substantially the same macroscopic behaviour can be obtained starting from "individuals" that are much simpler than real molecules, namely, the cells of a cellular automaton. Only certain features (such as, in this case, conservation of particles and of momentum) manage to make their effects felt all the way up to the macroscopic level; the others become irrelevant on a sufficiently large scale. This phenomenology was originally observed in physics, but analogous effects have been noted in other fields, such as biology and economics, where the interacting individuals may be much more complex than gas molecules. As was the case with gases, the essential aspects of these collective phenomena can generally be reproduced by cellular automata because they possess the essential feature of locality of interaction among a large number of similar individuals.

Cellular automata provide exactly computable models for various physical phenomena and large-scale correlations that result from very simple short-range interactions. In doing so, they provide a third alternative to the classical dichotomy between models that are solvable exactly (by analytical means) but are very stylised, and models that are more realistic but can only be solved approximately (by numerical means).

7.1. Lattice Gas Computers

In hydrodynamics we are chiefly interested in situations where different parts of a fluid move with respect to one another, and with respect to solid obstacles, at velocities that are much smaller than than that of sound. In this limit, and neglecting external forces such as gravity, any density differences equalise themselves in a negligible time and the fluid can be treated as incompressible. Even when these simplifications apply, the phenomenology of fluids can be enormously varied. Depending on the speed of the main flow, the size and shape of the obstacles, and the viscosity of the fluid, one can have laminar flow, vortices, turbulence etc. In this context the relevant variable is the *velocity* V (a vector) of flow at different points, and the relevant parameter is the *viscosity* ν of the fluid. The behaviour of the fluid is governed by the *Navier-Stokes* equation

$$\frac{\partial V}{\partial t} + (V\nabla)V = -\frac{1}{\rho}\nabla p + \nu\nabla^2 V$$

where p is the pressure and ρ is the (constant) density. This is a *nonlinear* differential equation, and, except for special cases, one must make recourse to numerical methods in order to find its solution for given initial and boundary conditions. Much computer time around the world is devoted to the numerical solution of such hydrodynamical problems.

In recent years various lattice gas cellular automata have been developed for the solution of fluid flow problems in hydrodynamics. We will briefly describe two of these automata. The first, which we will call HPP, was introduced in [156], and operates on a two dimensional orthogonal lattice. The second, FHP, was first described in [125], and operates on a two dimensional hexagonal lattice. It turns out that, on a macroscopic scale, the gases described by HPP obey the 2D Navier-Stokes equation approximately; and those described by FHP obey this equation exactly.

The HPP gas was originally formulated as follows. Consider an orthogonal lattice consisting of sites connected by north, south, west, and east links. There are four kinds of particles, one for each direction, and a site can be occupied by at most one particle of each kind (exclusion principle) Thus, there can be up to four particles per site. The updating is done with a two-step cycle. At step 0 each particle moves along a link from its current site to the adjacent site corresponding to its direction; at step 1 particles are shuffled within each site in the following way. If there are, at that site, exactly two particles which have come in from opposite directions, say north and south, then they are replaced by a west/east pair; otherwise nothing changes. As was mentioned, the behaviour of the HPP model departs from the Navier-Stokes equation. The various isotropy problems etc. associated with the HPP model are solved in the FHP model. It uses six kinds of particles; i.e. there are six directions of travel, with a 60° angle from one to the next. The lattice is a hexagonal one, and each site can contain up to six particles (one of each kind). As in the HPP model, collisions are defined so as to conserve energy (number of particles) and momentum. A simple version of the FHP rule has a collision occur whenever the momentum at a site is zero. If there is no collision, all particles go straight. If there is a collision, all particles at that site are deflected 60° clockwise (or counterclockwise) from the path they were following.

Three dimensional regular lattices do not have enough symmetry to ensure macroscopic isotropy [98, 125]. A suitable four dimensional structure was described in [98]. It is a face centered hypercubic (FCHC) lattice, defined as the set of signed integers (x_1, x_2, x_3, x_4) such that $x_1 + x_2 + x_3 + x_4$ is even. Each node is connected by links of length $\sqrt{2}$ to the 24 nearest neighbours, which have exactly two coordinates differing by ± 1. To model three dimensional fluids and maintain the required isotropies, we can define a pseudo-four-dimensional model [98] as the three dimensional projection of an FCHC model with unit periodicity in the x_4-direction.

Since the appearance of [125], lattice gas methods have been used to study 3D Navier-Stokes (NS) flow, 2D convective NS flow, 2D and 3D two-phase NS flow with interfacial surface tension, and various linear and nonlinear diffusion equations. The collection of papers in [100] provides an excellent account of the research work which has been carried out in recent years on lattice gas computation. See also [46, 76, 124, 364, 402].

By definition, cellular automata are highly parallel, highly regular, and require only local communication. This means that cellular automata machines can be easily scaled to arbitrarily large sizes with no loss of efficiency, as in the case of systolic arrays. In a previous section we described the kind of performance that a researcher working on three dimensional fluid flow using lattice gas methods is likely to require by the mid 1990s [96]. As noted there, it is unlikely that anything other than a very specialised system could come close to achieving those requirements. Therefore, we can expect to see the emergence of various new cellular automata and lattice gas architectures over the next few years. At one end of the spectrum, such systems might be based on standard VLSI devices such as FPGAs. At the other end, they may be based on exotic new technologies such as arrays of quantum dots.

The study of cellular automata and lattice gas computation is in many respects only just beginning. It will probably be many years before we have an understanding of the theory and applications of cellular automata and lattice gases comparable to that which we have today for differential equations. The availability of very powerful special purpose parallel systems for performing those computations will ensure that we continue to strive for such an understanding.

8. Multicomputers

A distributed memory multicomputer can be thought of as having p processor-memory pairs located at distinct vertices of a p-vertex graph. Each processor can send packets to, and receive packets from, processors at adjacent vertices in the graph. Each edge of the graph can transmit one packet of information in unit time, and has a queue for storing packets that have to be transmitted along it. A number of such multicomputer systems have been commercially available from companies in the United States and Europe since the mid 1980s. In such a system, memory is organised in terms of multiple private address spaces with explicit message passing by the programmer [168, 170, 187] for all non-local memory requests. Despite the difficulties of programming such systems, and the architecture dependence of the resulting software, a large number of interesting new high performance parallel algorithms have been developed for multicomputer architectures of various kinds. Detailed accounts of the theoretical aspects of multicomputers can be found in [185, 224, 226, 227]. For an account of some of the more practical issues involving multicomputers see, for example, [17, 34, 94, 185, 340, 341, 342]. In this paper we will limit ourselves to discussing some of the general theoretical issues concerning such systems.

A large number of graphs have been proposed as interconnection networks for such multicomputers. Two important parameters of any such graph are its *degree*, i.e. the maximum number of edges incident at any vertex, and its *diameter*, i.e. the maximum distance between any pair of vertices, where the distance between two vertices is the length of a shortest path between them. If implemented using conventional VLSI technology, a graph with low degree is likely to have advantages in terms of physical packaging. The advantage of using a graph with small diameter is, of course, that it will permit a packet to be sent quickly between any two vertices in the network. Another important property of any such graph is its bisection width. The *bisection width* of a graph is the minimum number of edges that have to be removed in order to partition the graph into two parts where the numbers of vertices differ by at most one. The bisection width of a network is often a critical factor in determining the speed with which a multicomputer can perform a computation. This is due to the fact that for many problems, the data contained in,or computed by, one half of the machine may be needed by the other half before the computation can be completed. A fourth property of any such graph, closely related to the bisection width, is the *area* required for its layout in the VLSI model of section 5.

Theorem 8.1 (Thompson [357]) *Any VLSI layout of a graph with bisection width B and degree at most four requires area $\Omega(B^2)$.*

Proof. Suppose the graph can be laid out in a rectangle of height h and width w, as in Figure 2 (section 5). By an argument very similar to that used in the proof of Theorem 5.10, we can show that there is a line which partitions the layout into two parts where the numbers of vertices differ by at most one, and which runs vertically between the cells, except possibly for a single jog of one cell width. By the definition of the bisection width B, the number of edges crossing the line is at least $B-1$ (at most one edge may cross the horizontal segment). Thus, the height of the layout must be at least $B-1$. Similarly, the width of the graph must be at least $B-1$. Therefore, the area must be at least $(B-1)^2$. (Note that Theorem 8.1 can be trivial, as in the case of a disconnected graph.) □

We have now listed four important properties of any graph proposed for the interconnection network of a multicomputer: degree, diameter, bisection width, and area. For our fifth, and final, property we note that ease of programming and avoidance of bottlenecks both suggest that a useful feature for any such graph is that it should look isomorphic from any vertex, i.e. it should be vertex transitive [39]. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. An automorphism of G is a permutation p of V such that $(v_1, v_2) \in E$ holds if and only if $(\pi(v_1), \pi(v_2)) \in E$. The set of all automorphisms with the operation of composition forms a group acting on V. For example, for the complete graph K_p on p vertices, the group of automorphisms is the symmetric group S_p since any permutation of vertices preserves adjacency. (The empty graph on p vertices also admits S_p.) A graph is *vertex transitive* if it admits a group of automorphisms acting transitively on the vertices [39]. (A permutation group (H, V) is *transitive* if there is only one orbit in the action of H on V. We can check this by picking an element $v \in V$ and looking for elements of H taking v to every other element in V.) There is also a related property of *edge transitivity* but, for simplicity, we will not pursue that notion here. Various graphs on p vertices which might be considered as interconnection networks, are listed in Table 1, together with information on their degree, diameter, bisection width, area, and vertex transitivity.

Detailed descriptions of these various graphs can be found in [38, 48, 145, 146, 222, 224, 227, 228, 231, 232, 233, 240, 289].

Name	Degree	Diameter	Bisection Width	Area	VT?
1D array (ring)	2	$p/2$	2	$\Theta(p)$	yes
Complete binary tree	3	$2\ log\ p$	1	$\Theta(p)$	no
Shuffle-exchange	3	$2\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	no
Cube-connected-cycles	3	$(5/2)\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	yes
2D mesh of trees	3	$2\ log\ p$	$\Theta(p^{1/2})$	$\Theta(p\ log^2 p)$	no
3D mesh of trees	3	$2\ log\ p$	$\Theta(p^{2/3})$	$\Theta(p^{4/3})$	no
1D multigrid	4	$4\ log\ p$	$O(log\ p)$	$\Theta(p)$	no
2D array (toroidal)	4	$\Theta(p^{1/2})$	$\Theta(p^{1/2})$	$\Theta(p)$	yes
Tree of meshes	4	$\Theta(p^{1/2}/log^{1/2} p)$	$\Theta(p^{1/2}/log^{1/2} p)$	$\Theta(p\ log\ p)$	no
Fat tree	4	$\Theta(log\ p)$	$O(p^{1/2}/log\ p)$	$\Theta(p)$	no
Butterfly (wrapped)	4	$2\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	yes
de Bruijn graph	4	$log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	no
Complete quadtree	5	$log\ p$	2	$\Theta(p)$	no
X-Tree	5	$2\ log\ p$	$\Theta(log\ p)$	$\Theta(p)$	no
2D multigrid	6	$3\ log\ p$	$\Theta(p^{1/2})$	$\Theta(p)$	no
3D array (toroidal)	6	$\Theta(p^{1/3})$	$\Theta(p^{2/3})$	$\Theta(p^{4/3})$	yes
Twin butterfly (wrapped)	8	$2\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	no
Pyramid	9	$log\ p$	$\Theta(p^{1/2})$	$\Theta(p)$	no
Complete 2^k-ary tree	$2^k + 1$	$2\ log_k p/k$	2^{k-1}	$O(4^k p)$	no
Hypercube	$log\ p$	$log\ p$	$p/2$	$\Theta(p^2)$	yes
$(p^{1/2})$-deg. random graph	$p^{1/2}$	2	$\Theta(p^{3/2})$	$\Theta(p^3)$	no
Complete graph (K_p)	$p - 1$	1	$\Theta(p^2)$	$\Theta(p^4)$	yes

Table 1.

If our interconnection network is to be implemented using VLSI technology then, ideally, we would like it to have low degree, low diameter, high bisection width and low area, and also to be vertex transitive. The reason so many networks have been proposed in the last decade is that, of course, these desirable features, to some extent, conflict with one another. The constant bisection width of rings and trees tends to rule them out of serious consideration. (Although multicomputers with more powerful connectivity are often used as rings to implement one dimensional systolic algorithms [271].) Graphs with degree higher than about $log\ p$ tend to be ruled out for multicomputer networks on grounds of packaging. (Although Valiant [374] has suggested ($p^{1/2}$)-degree random graphs as a biologically plausible structure for certain types of neural networks.) The multigrids, X-Tree and pyramid have been suggested as appropriate for applications in numerical computation, image processing and computer vision [5, 284, 351]. They do, indeed, appear to be well suited to such applications. However, for a broader class of computations, some of which require a large amount of simultaneous non-local communication, the exponentially decreasing bisection width as we move up from the base of the 2D multigrid or pyramid to the top level, is likely to be a severely limiting factor. Arrays have been widely proposed as the best general structure for multicomputers, and a number of practical designs are based on arrays. The main advantage of the 2D and 3D arrays are their simplicity, and the fact that they can be easily implemented as physical VLSI systems. 2D arrays have optimal $\Theta(p)$ VLSI area and 3D arrays have optimal $\Theta(p)$ VLSI *volume*. The main disadvantage of arrays, for computations requiring a large amount of non-local communication, is their high diameter. (Although, again, they are very appropriate for implementing systolic algorithms, and others which require only local communication [224, 335, 360].) We now briefly discuss some of the properties of the remaining graphs in the table. The shuffle-exchange, cube-connected-cycles, butterfly and de Bruijn graphs have very similar properties. We will refer only to the butterfly. Various fat tree designs have been proposed. They are all essentially improvements of Leighton's original tree of meshes graph [222]. For that reason, we will not explicitly consider the original tree of meshes graph further here. The 2D and 3D mesh of trees differ only in terms of their dimension. We will consider only the 2D mesh of trees. We thus have reduced our list of candidates to those shown in Table 2.

Name	Degree	Diameter	Bisection Width	Area	V.T. ?
2D mesh of trees	3	$2\ log\ p$	$\Theta(p^{1/2})$	$\Theta(p\ log^2 p)$	no
Fat tree	4	$\Theta(log\ p)$	$O(p^{1/2}/log\ p)$	$\Theta(p)$	no
Butterfly (wrapped)	4	$2\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	yes
Twin butterfly (wrapped)	8	$2\ log\ p$	$\Theta(p/log\ p)$	$\Theta(p^2/log^2 p)$	no
Hypercube	$log\ p$	$log\ p$	$p/2$	$\Theta(p^2)$	yes

Table 2.

Leiserson [232] has argued that, in the design of interconnection networks, a major goal should be to achieve *area universality* in the two dimensional VLSI model, or *volume universality* in the three dimensional VLSI model. An area universal network is one which, for a given area A, can simulate any other network of comparable area, with only $O(log\ A)$ slowdown (at least with high probability.) It is well known that a p-processor hypercube can efficiently simulate any fixed degree p-processor network. We need only realise an h-relation, where h is the degree of the network. This can be done in $O(log\ p)$ steps on a hypercube for any $h \leq log\ p$ [376]. If, however, we normalise by area instead of by number of processors, we see that an area A hypercube cannot simulate all area A networks efficiently. For example, since an area A hypercube (butterfly) has only $\Theta(A^{1/2})$ ($\Theta(A^{1/2}\ log\ A)$ respectively) processors (see above table), it cannot simulate a 2D array with area A, which has $\Theta(A)$ processors (see table) in any polylogarithmic time. Leiserson has established the following theorem.

Theorem 8.2 *The fat tree is an area universal network for the two dimensional VLSI model, and a volume universal network for the three dimensional VLSI model.*

For details of the proof of Theorem 8.2, see [145, 227, 232]. The design and analysis of efficient algorithms for the fat tree is considered in [233]. Theorem 8.2 suggests that if one plans to implement a p-processor interconnection network in VLSI, then, for large p, one should implement a fat tree rather than a butterfly or a hypercube, despite the fact that it lacks the attractive property of being vertex transitive. (Note. The interconnection network for the latest Connection Machine architecture, the CM5, is based on a fat tree.)

We have seen that butterflies and hypercubes cannot be efficiently implemented using VLSI technology, i.e. without having long wires for some of the edges in the network. However, optical communication systems [30, 113] offer the prospect of a dramatic improvement in the efficiency with which such networks can be implemented. In [313], Rao shows that a simple (and possibly cheap) optical interconnection architecture based on wavelength division multiplexing can be used to solve the VLSI wiring problem. He considers various routing problems on a $p^{1/2} \times p^{1/2}$ 2D array of processors, where the processors on each row of the array are connected by an optical bus, and the processors on each column of the array are also connected by such a bus. We thus have $2p^{1/2}$ buses, each of length $p^{1/2}$. Each bus uses *wavelength division multiplexing (WDM)* [101] to support simultaneous communication between many disjoint pairs of processors on the same bus. This communications architecture will be referred to as the p processor *mesh of buses (MOB)*. The idea of using a mesh of buses for interconnection was proposed by Wittie [401], and as a basis for optical interconnection networks by Dowd [101]. To solve the VLSI wiring problem we need only consider the simplest MOB in which the optical buses have fixed transmitters and receivers at each processor, i.e. where the transmitters and receivers are initially set (off-line) to achieve a certain communication pattern, e.g. a hypercube or a butterfly, and then remain fixed as that pattern is used. The first result shows that all networks can be emulated on the MOB with an efficiency related to the degree of the network.

Theorem 8.3 *Any p processor network N of degree d can be emulated on a p processor MOB so that (i) there are $O(d)$ transmitters/receivers per processor, and (ii) each edge in N is realised by a path of length at most three in the MOB.*

By an edge in the MOB we mean a channel on one of the buses, and by a path we mean a sequence of such edges. For networks such as the cube-connected-cycles, butterfly and hypercube we can do even better.

Theorem 8.4 *A p processor cube-connected-cycles or butterfly network can be emulated on a p processor MOB so that (i) there are $O(1)$ transmitters/receivers per processor, and (ii) each edge in the cube-connected-cycles is realised by a single edge in the MOB.*

Theorem 8.5 *A p processor hypercube network can be emulated on a p processor MOB so that (i) there are $O(log\ p)$ transmitters/receivers per processor, and (ii) each edge in the hypercube is realised by a single edge in the MOB.*

Theorems 8.4 and 8.5 follow directly from standard VLSI layouts of those networks.

The mesh of trees network has been extensively studied in recent years. In Chapter 2 of [224], Leighton describes a large number of parallel algorithms which can be efficiently implemented on the two dimensional mesh of trees architecture. The problems include packet routing, sorting, matrix-vector multiplication, Jacobi relaxation, pivoting, convolution, convex hull, integer multiplication, powering, division, root finding, minimum cost spanning tree, connected components, transitive closure, shortest paths, maximum matching, breadth-first search, and evaluation of straight line programs. He also shows that various matrix problems such as matrix multiplication, inversion, decomposition and powering can be efficiently solved on the three dimensional mesh of trees.

The hypercube family of graphs (hypercube, butterfly, shuffle-exchange, cube-connected-cycles, de Bruijn graph) have also been widely studied. In Chapter 3 of [224], Leighton describes various parallel algorithms which can be efficiently implemented on the hypercube family. The problems include packet routing, sorting, and computing the discrete Fourier transform. He also describes a number of more general results concerning the hypercube family, including the following two theorems.

Theorem 8.6 *A p-processor hypercube can simulate any $O(p)$-processor array, binary tree, mesh of trees, X-Tree, or pyramid, with only a small constant factor slowdown.*

A hypercube algorithm is said to be *normal* if only one dimension of hypercube edges is used at any step, and if consecutive dimensions are used in consecutive steps. There are many important examples of hypercube algorithms that are normal. For instance, all of the mesh of trees algorithms described in Chapter 2 of [224] can be implemented in a normal form on the hypercube.

Theorem 8.7 *For a normal hypercube algorithm, the fixed degree members of the hypercube family (p-processor butterfly, shuffle-exchange, cube-connected-cycles, de Bruijn graph) can simulate the computation of the p-processor ($(log\ p)$-degree) hypercube with only constant slowdown.*

Theorem 8.7 is a very powerful result since those fixed degree graphs have a factor of $log\ p$ fewer edges than the hypercube. It is, however, known that for the general problem of packet routing, the hypercube is more powerful than the fixed degree members of the family, since one can realise a $O(log\ p)$-relation on the hypercube in only $O(log\ p)$ steps [376]. Therefore, for the packet routing problem at least, the much greater communications bandwidth afforded by the higher degree is an advantage.

The multibutterfly family of graphs, of which the twin butterfly is a member, is another family of interconnection networks which has been receiving attention recently. In [228, 371] it is shown that a simple deterministic packet routing algorithm can be used to realise a 1-relation in $O(log\ p)$ steps on a multibutterfly. The multibutterfly is a randomly wired, bounded degree network based on the butterfly. An important feature of multibutterflies is that they have powerful expansion properties. In addition to permitting fast deterministic packet routing, such expander graphs also have very strong fault tolerance properties.

Finally, we mention two graphs of rather more specialised interest, the Sneptree and the polymorphic array. Many important computational problems have an inherent tree structure. For example, algorithms for game tree evaluation, functional expression evaluation, branch and bound search, and divide and conquer often give rise to tree structures, where the nodes of the tree represent computational tasks to be performed and the edges of the tree represent the required communication. For such computations, we would like to have an interconnection network onto which we could embed any tree in a way that balanced the computational load on each processor, and kept the communication local. In addition, we would like to be able to do the embedding dynamically. This is because, for many such applications (e.g. game tree evaluation, branch and bound search), the structure of the tree will change significantly during the course of the computation. In Chapter 3 of [224] it is shown that a binary tree can be efficiently embedded, in a dynamic manner, in a hypercube of comparable size. The Sneptree [165, 242, 264, 265] is another convenient network for dynamic tree embeddings. A Sneptree can be regarded as the degree-regular closure of a complete binary tree in much the same way as the ring is the degree-regular closure of a 1D array. A *k-level Sneptree* is a directed complete binary tree of $2^k - 1$ vertices, with arcs directed towards the leaves, augmented with 2^k arcs directed out of the leaves, such that each vertex has four incident arcs, two directed in and two directed out. The root and the leaves of the underlying binary tree of a Sneptree are called the root and leaves of the Sneptree as well. Of the two outgoing arcs of a vertex, one is called the left arc and the other the right arc, whilst the vertex pointed to by the left (right) arc of a vertex is said to be the left (right) child of that vertex. There are many possible ways to connect the 2^k augmented arcs. We can directly embed any complete binary tree T onto any Sneptree S by simply mapping the root of T onto the root of S, and mapping the left and right child of any vertex $t \in T$ onto the left and right child, respectively, of the vertex in S to which t is mapped. This embedding is efficient, i.e. achieves good load balancing and locality. For the problem of embedding a binary tree $\hat{T}$ which is not a complete binary tree, onto a Sneptree S, we can use exactly the same simple mapping technique. However, in this case, the distribution of vertices from $\hat{T}$ over the vertices of S will, of course, be strongly affected by the particular interconnection pattern of the Sneptree. A detailed treatment of Sneptrees can be found in [165]. A flexible network for 2D array computations is provided by the polymorphic array [115]. A *p-vertex polymorphic array* is a regular, degree four graph, which has a two dimensional VLSI layout in which all edges have constant length. It, therefore, requires only $O(p)$ area. Any 2D array (of any shape or size) can be directly mapped onto a polymorphic array, as in the case of the Sneptree. The resulting embedding preserves locality, and is guaranteed to balance the load on each processor to within an additive $O(log\ p)$ term of the best possible.

For more on the VLSI layout of networks, on the efficient embedding of one network in another, and on the demonstration of work-preserving emulations of one network by another, see [35, 36, 37, 38, 47, 52, 146, 165, 171, 185, 199, 200, 222, 224, 226, 227, 231, 240, 289, 323, 336, 372, 384].

9. Intelligent Systems

The remarkable phenomenon of human thought has interested philosophers since the time of Plato and Aristotle and a large number of physical and metaphysical explanations of thinking and, more generally, of the relationship between mind and body, have been proposed down through the centuries. In the past one hundred years, there have been major advances in neurobiology, psychology and computing. The developments in neurobiology and psychology have provided us with a detailed, but by no means complete, understanding of the structure and function of the various parts of the brain [343]. Progress in computing, and more particularly in the area of artificial intelligence [400], has shown that many of the "higher level" problem solving capabilities of the human brain can be performed by computing systems. In spite of these advances, a number of the central philosophical questions, such as "can a machine think?", continue to generate heated debates amongst both philosophers and scientists [82, 102, 119, 301, 337, 366, 367]. An examination of the arguments put forward in such debates would suggest that, as in the case of many interesting philosophical questions, the issues are unlikely to ever be completely resolved, and certainly not by progress in science and technology alone.

Soon after the first digital computers were developed, a number of researchers began to explore the possibility of designing computing systems which would be capable of "intelligent" behaviour, i.e. would exhibit behaviour which would hitherto have required human intelligence. From the time of the earliest papers on artificial intelligence, there have been two distinct approaches to achieving the general goal of intelligent computing systems. We will refer to these as the symbolic computation approach and the neural networks approach. In very general terms, those pursuing the symbolic computation approach, whether from the logicist [294] or the "knowledge is all there is" [235] schools, have been concerned with developing advanced computing systems which can perform "complex" problem solving tasks (e.g. chess playing, theorem proving, medical diagnosis, chemical structure analysis, constraint satisfaction, rule-based deduction) without particular regard to how those tasks might be carried out in the human brain. Those pursuing the neural networks approach, on the other hand, have traditionally been more concerned with directly modelling the kinds of complex physical processes that might be occurring in the brain when one is performing "simple" tasks such as vision or basic pattern recognition [8, 9, 172, 188, 195, 274, 285, 324, 327, 344, 398, 400]. In the neural networks approach there has been a much greater emphasis on the importance of training/learning. Over the last few years, in addition to the work which has been carried out on the engineering of learning systems (in the field of neural networks) and on the understanding of human learning (in computational neuroscience), a large amount of work has been done on the computational complexity of various aspects of machine learning [188, 193, 293, 373, 377].

The enthusiasm for each of these two approaches (symbolic computation, neural networks) has fluctuated over the years, as new computing technologies have been developed and as different types of applications have been explored [104, 286]. In the last few years, a third bottom-up perspective on the development of intelligent systems, which we might call the "embedded computation" approach, has been strongly promoted within the artificial intelligence community. In this approach there is much greater emphasis on the importance of situatedness and embodiment [61, 62, 197, 198, 254, 392], as in the philosophy of Heidegger [103]. These three approaches to the problem of developing intelligent systems emphasise the need for different types of special purpose parallel computing systems. With symbolic computation one requires high performance systems for combinatorial searching, mechanised reasoning etc. With neural networks one requires large collections of relatively simple automata. In the case of embedded computation, as exemplified by the robots developed by Brooks and his colleagues [61, 62], one requires small, powerful on-board distributed systems for various tasks such as sensing, navigation, planning, reaction and learning.

Much of the experimental work on neural networks so far has been concerned with the goal of emulating "high level" cognitive skills such as problem solving or language. Our current understanding of how such activities are performed in the human nervous system is not, however, particularly well developed. In contrast to this, there has been, in recent years, a major increase in our knowledge of the human visual and auditory systems, due to the emergence of new experimental techniques in neuroscience and psychology. Detailed physiological studies have given us a picture of the mapping of the visual field onto the visual cortex, and similar information is available for many important auditory areas. For simpler life forms, even more is known. For example, it is known that a fly eye contains about 24,000 visual receptors, and four layers of neurons between the photoreceptors at the input level and motoneurons at the output level (see reference in [343]). Over the last few years, Mead and his colleagues [256, 278, 328] have been developing a variety of new analog VLSI systems which aim to emulate the performance of living systems in certain well understood areas of neural computation. They have developed motion sensors [278], silicon models of the retina [256, 278] and inner ear [278], and models of pattern generators in simple invertebrate systems [328].

The development of various kinds of "intelligent" systems, based on special purpose parallel architectures, is likely to be a major area of activity in the coming decade.

Acknowledgements

I would like to thank Les Valiant and Bill Gear for providing the opportunity for me to spend my sabbatical leave from Oxford at NEC Research Institute, and to my colleagues at NECI for providing such a stimulating environment in which to work.

References

[1] E Aarts, F de Bont, E Habers, and P van Laarhoven. Parallel implementations of the statistical cooling algorithm. *Integration, The VLSI Journal*, 4:209–238, 1986.

[2] E H L Aarts and J H M Korst. Boltzmann machines as a model for parallel annealing. *Algorithmica*, 6(3):437–465, 1991.

[3] H Abelson and P Andreae. Information transfer and area-time tradeoffs for VLSI multiplication. *Communications of the ACM*, 23(1):20–23, January 1980.

[4] A Aggarwal. On simulation and interrelations between Boolean and VLSI circuits. In *Proc. 21st Annual Allerton Conference*, pages 258–265, 1983.

[5] A Aggarwal. A comparative study of x-tree, pyramid and related machines. In *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 89–99, 1984.

[6] A V Aho, J D Ullman, and M Yannakakis. On notions of information transfer in VLSI circuits. In *Proc. 15th Annual ACM Symposium on Theory of Computing*, pages 133–139, 1983.

[7] Algotronix Limited. *CAL 1024 Datasheet.* Algotronix Limited, 1990.

[8] J A Anderson, A Pellionisz, and E Rosenfeld, editors. *Neurocomputing 2: Directions for Research.* MIT Press, Cambridge, MA, 1990.

[9] J A Anderson and E Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, MA, 1988.

[10] R J Anderson and G L Miller. Optical communication for pointer based algorithms. Technical Report CRI 88-14, University of Southern California, 1988.

[11] M Annaratone, E Arnould, T Gross, H T Kung, M Lam, O Menzilcioglu, and J A Webb. The Warp computer: Architecture, implementation and performance. *IEEE Transactions on Computers*, C-36(12):1523–1538, December 1987.

[12] A Apostolico and A Negro. Systolic algorithms for string manipulations. *IEEE Transactions on Computers*, 33(4):361–363, April 1984.

[13] T Asano and H Umeo. Systolic algorithms for computing the visibility polygon and triangulation of a polygonal region. *Parallel Computing*, 6:209–216, 1988.

[14] M J Atallah and S R Kosaraju. A generalized dictionary machine for VLSI. *IEEE Transactions on Computers*, 34(2):151–155, 1985.

[15] M J Atallah and S McFaddin. Sequence comparison on the Connection Machine. *Concurrency: Practice and Experience*, 3(2):89–107, April 1991.

[16] R A Athale and J N Lee. Optical processing using outer product concepts. *Proceedings of the IEEE*, 72(7):931–941, July 1984.

[17] W C Athas and C L Seitz. Multicomputers: Message-passing concurrent computers. *IEEE Computer*, 12(8):9–24, August 1988.

[18] M L Bailey. How circuit size affects parallelism. *IEEE Transactions on Computer-Aided Design*, 11(2):208–215, February 1992.

[19] R Barakat and J H Reif. Lower bounds on the computational efficiency of optical computing systems. *Applied Optics*, 26:1015–1018, 1987.

[20] R Barakat and J H Reif. Polynomial convolution algorithm for matrix multiplication with application for optical computing. *Applied Optics*, 26(14):2707–2711, July 1987.

[21] J Barnes and P Hut. A hierarchical $O(n\ log\ n)$ force calculation algorithm. *Nature*, 324:446–449, 1986.

[22] P A Bash et al. Free energy calculations by computer simulation. *Science*, 236:564–568, 1987.

[23] K E Batcher. Design of a massively parallel processor. *IEEE Transactions on Computers*, C-29(9):836–840, September 1980.

[24] G M Baudet. Design and complexity of VLSI algorithms. In J W de Bakker and J van Leeuwen, editors, *Foundations of Computer Science IV, Part 1: Algorithms and Complexity, Mathematical Centre Tracts 158*, pages 49–74. Mathematisch Centrum, Amsterdam, 1983.

[25] G M Baudet and W-C Chen. Area-time tradeoffs for merging. In P Bertolazzi and F Luccio, editors, *VLSI: Algorithms and Architectures*, pages 61–68. North Holland, 1984.

[26] G M Baudet, F P Preparata, and J Vuillemin. Area-time optimal VLSI circuits for convolution. *IEEE Transactions on Computers*, 32(7):684–688, July 1983.

[27] B Baxter, G Cox, T Gross, H T Kung, D O'Hallaron, C Peterson, J Webb, and P Wiley. Building blocks for a new generation of application-specific computing systems. In J A B Fortes, S Y Kung, K W Przytula, and E E Swartzlander Jr, editors, *Proc. IEEE International Conference on Application Specific Array Processors, Princeton, New Jersey*, pages 190–201. IEEE Press, 1990.

[28] C Bays. Candidates for the game of life in three dimensions. *Complex Systems*, 1(3):373–400, June 1987.

[29] J Beetem, M Denneau, and D Weingarten. GF11 - a supercomputer for scientific applications. In *Proc. 12th Annual International Symposium on Computer Architecture*, 1985.

[30] T E Bell. Optical computing: A field in flux. *IEEE Spectrum*, 23(8):34–57, August 1986.

[31] E R Berlekamp, J H Conway, and R K Guy. *Winning Ways For Your Mathematical Plays. Volume 2: Games in Particular.* Academic Press, 1982.

[32] C Bermond, M C Ogilvie, T A DeGrand, C E DeTar, S A Gottlieb, A Krasnitz, R L Sugar, and D Toussaint. Studying quarks and gluons on MIMD parallel computers. *The International Journal of Supercomputer Applications*, 5(4):61–70, Winter 1991.

[33] P Bertin, D Roncin, and J Vuillemin. Introduction to programmable active memories. In J McCanny, J McWhirter, and E Swartzlander, editors, *Systolic Array Processors*, pages 301–309. Prentice Hall, 1989.

[34] D P Bertsekas and J N Tsitsiklis. *Parallel and Distributed Computation - Numerical Methods.* Prentice Hall, 1989.

[35] S N Bhatt, F Chung, F T Leighton, and A L Rosenberg. Optimal simulations of tree machines. In *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–282, 1986.

[36] S N Bhatt, F R K Chung, J W Hong, F T Leighton, and A L Rosenberg. Optimal simulations by butterfly networks. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 192–204, 1988.

[37] S N Bhatt, F R K Chung, F T Leighton, and A L Rosenberg. Efficient embeddings of trees in hypercubes. *SIAM Journal on Computing*, 21(1):151–162, February 1992.

[38] S N Bhatt and F T Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[39] N L Biggs. *Algebraic Graph Theory.* Cambridge University Press, Cambridge, UK, 1974.

[40] G Bilardi, S W Hornick, and M Sarrafzadeh. Optimal VLSI architectures for multidimensional DFT. In *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 265–272, 1989.

[41] G Bilardi and F P Preparata. A minimum area VLSI network for $O(log\, n)$ time sorting. *IEEE Transactions on Computers*, 34(4):336–343, April 1985.

[42] G Bilardi and F P Preparata. Area-time lower bound technique with application to sorting. *Algorithmica*, 1(1):65–91, 1986.

[43] G Bilardi and M Sarrafzadeh. Optimal discrete Fourier transform in VLSI. In P Bertolazzi and F Luccio, editors, *VLSI: Algorithms and Architectures*, pages 79–89. North Holland, 1984.

[44] G E Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.

[45] G E Blelloch. *Vector Models for Data-Parallel Computing.* MIT Press, Cambridge, MA, 1990.

[46] B M Boghosian. Lattice gases. In E Jen, editor, *1989 Lectures in Complex Systems. Proc. 1989 Complex Systems Summer School, Santa Fe, New Mexico*, pages 293–324. Addison-Wesley, 1990.

[47] S H Bokhari. On the mapping problem. *IEEE Transactions on Computers*, 30:207–212, 1981.

[48] B Bollobás. *Random Graphs*. Academic Press, 1985.

[49] S Borkar et al. iWarp: An integrated solution to high-speed parallel computing. In *Proc. Supercomputing '88*, pages 330–339, November 1988.

[50] S Borkar et al. Supporting systolic and memory communication in iWarp. In *Proc. 17th Annual International Symposium on Computer Architecture*, pages 70–81. IEEE Press, 1990.

[51] W Braun and N Go. Calculation of protein conformations by proton-proton distance constraints - A new efficient algorithm. *Journal of Molecular Biology*, 186:611–626, 1985.

[52] G Brebner. Relating routing graphs and two-dimensional grids. In P Bertolazzi and F Luccio, editors, *VLSI: Algorithms and Architectures*, pages 221–231. North Holland, 1984.

[53] K-H Brenner, A Huang, and N Streibl. Digital optical computing with symbolic substitution. *Applied Optics*, 25(18):3054–3060, September 1986.

[54] R P Brent and L M Goldschlager. Some area-time tradeoffs for VLSI. *SIAM Journal on Computing*, 11(4):737–747, November 1982.

[55] R P Brent and H T Kung. The chip complexity of binary arithmetic. In *Proc. 12th Annual ACM Symposium on Theory of Computing*, pages 190–200, 1980.

[56] R P Brent and H T Kung. The area-time complexity of binary multiplication. *Journal of the ACM*, 28(3):521–534, 1981.

[57] R P Brent and H T Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, C–31(3):260–264, March 1982.

[58] R P Brent and H T Kung. Systolic VLSI arrays for linear time GCD computation. In *VLSI83*. IFIP, 1983.

[59] R P Brent and H T Kung. Systolic VLSI arrays for polynomial GCD computation. *IEEE Transactions on Computers*, 33(8):731–736, August 1984.

[60] R P Brent and H T Kung. A systolic algorithm for integer GCD computation. In K. Hwang, editor, *Proc. 7th IEEE Symposium on Computer Arithmetic*, pages 118–125, 1985.

[61] R A Brooks. Intelligence without reason. In J Mylopoulos and R Reiter, editors, *Proc. 12th International Joint Conference on Artificial Intelligence*, pages 569–595. Morgan Kaufmann, San Mateo, CA, 1991.

[62] R A Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, January 1991.

[63] M C Browne, E M Clarke, D L Dill, and B Mishra. Automatic verification of sequential circuits using temporal logic. In C J Koomen and T Moto-oka, editors, *Computer Hardware Description Languages and Their Applications, Seventh International Conference, Tokyo*, pages 98–113. North Holland, 1985.

[64] R E Bryant. A switch-level model and simulator for MOS digital systems. *IEEE Transactions on Computers*, C-33(2):160–177, February 1984.

[65] A W Burks, editor. *Essays on Cellular Automata.* University of Illinois Press, 1970.

[66] A W Burks, H H Goldstine, and J von Neumann. *Preliminary discussion of the logical design of an electronic computing instrument. Part 1, Volume 1.* The Institute of Advanced Study, Princeton, First edition, 28 June 1946. Second edition, 2 September 1947. Report to the U.S. Army Ordnance Department. Also appears in *Papers of John von Neumann on Computing and Computer Theory*, W Aspray and A Burks, editors. Volume 12 in the Charles Babbage Institute Reprint Series for the History of Computing, MIT Press, 1987, 97-142.

[67] P R Cappello and K Steiglitz. Unifying VLSI array design with linear transformations of space-time. *Advances in Computing Research*, 2:23–65, 1984. Jai Press Inc.

[68] D Casasent. Acoustooptic linear algebra processors: Architectures, algorithms, and applications. *Proceedings of the IEEE*, 72(7):831–849, July 1984.

[69] P Caspi, D Pilaud, H Halbwachs, and J A Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Proc. 14th Annual ACM Symposium on Principles of Programming Languages*, pages 178–188, 1987.

[70] T F Chan and Y Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Transactions on Computers*, C-35:969–977, 1986.

[71] K M Chandy and J Misra. Systolic algorithms as programs. *Distributed Computing*, 1(3):177–183, 1986.

[72] K M Chandy and J Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[73] J Chang, O Ibarra, and M Palis. Parallel parsing on a one–way array of finite–state machines. *IEEE Transactions on Computers*, C–36(1):64–75, January 1987.

[74] B Chazelle. Computational geometry on a systolic chip. *IEEE Transactions on Computers*, 33(9):774–785, September 1984.

[75] B Chazelle and L Monier. A model of computation for VLSI with related complexity results. *Journal of the ACM*, 32(3):573–588, July 1985.

[76] H Chen, S Chen, G D Doolen, and W H Matthaeus. Lattice-gas models for multiphase flows and magnetohydrodynamics. In E Jen, editor, *1989 Lectures in Complex Systems. Proc. 1989 Complex Systems Summer School, Santa Fe, New Mexico*, pages 389–399. Addison-Wesley, 1990.

[77] M C Chen. A parallel language and its compilation to multiple processor machines or VLSI. In *Proc. 13th Annual ACM Symposium on Principles of Programming Languages*, pages 131–139, 1986.

[78] Y-A Chen, Y-L Lin, and L-W Chang. A systolic algorithm for the k-nearest neighbors problem. *IEEE Transactions on Computers*, 41(1):103–108, January 1992.

[79] Y-C Cheng, D J Houck Jr, J-M Liu, M S Meketon, L Slutsman, R J Vanderbei, and P Wang. The AT&T KORBX system. *AT&T Technical Journal*, 68(3):7–19, May/June 1989.

[80] E Chu and A George. Sparse orthogonal decomposition on a hypercube multiprocessor. *SIAM Journal on Matrix Analysis and Applications*, 11:453–465, 1990.

[81] J I Chu and G Schnitger. The communication complexity of several problems in matrix computation. *Journal of Complexity*, 7:395–407, 1991.

[82] P M Churchland and P S Churchland. Could a machine think? *Scientific American*, 262(1):32–37, January 1990.

[83] E F Codd. *Cellular Automata*. Academic Press, 1968.

[84] B Codenotti and G Lotti. A note on the VLSI counter. *Information Processing Letters*, 22:193–195, April 1986.

[85] B Codenotti, G Lotti, and P Romani. VLSI implementation of iterative methods for the solution of linear systems. *Integration, The VLSI Journal*, 3:211–221, 1985.

[86] B Codenotti and F Romani. A compact and modular VLSI design for the solution of general sparse linear systems. *Integration, The VLSI Journal*, 5:77–86, 1986.

[87] A Cohn and M Gordon. A mechanised proof of correctness of a simple counter. In K McEvoy and J V Tucker, editors, *Theoretical Foundations of VLSI Design*, volume 10 of *Cambridge Tracts in Theoretical Computer Science*, pages 65–96. Cambridge University Press, Cambridge, UK, 1990.

[88] R Cole and A Siegel. On information flow and sorting: New upper and lower bounds for VLSI circuits. In *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 208–221, 1985.

[89] J M Cooley and J W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[90] E Corcoran. Diminishing dimensions. *Scientific American*, 263(5):122–131, November 1990.

[91] T H Cormen, C E Leiserson, and R L Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[92] G M Crippen. *Distance Geometry and Conformational Calculations*. John Wiley and Sons, 1981.

[93] R Cypher and C G Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 193–203, 1990.

[94] W J Dally. Network and processor architecture for message-driven computers. In R Suaya and G Birtwistle, editors, *VLSI and Parallel Computaion*, pages 140–222. Morgan Kaufmann, San Mateo, CA, 1990.

[95] F Dehne, J-R Sack, and N Santoro. Computing on a systolic screen: Hulls, contours and applications. In J W de Bakker, A J Nijman, and P C Treleaven, editors, *PARLE - Parallel Architectures and Languages Europe, Vol.1: Parallel Architectures, LNCS Vol. 258*, pages 121–133. Springer-Verlag, 1987.

[96] A Despain, C E Max, G D Doolen, and B Hasslacher. Prospects for a lattice-gas computer. In Doolen et al. [100], pages 211–218.

[97] P M Dew, R A Earnshaw, and T R Heywood, editors. *Parallel Processing for Computer Vision and Display*. Addison-Wesley, 1989.

[98] D d'Humiéres, P Lallemand, and U Frisch. Lattice gas models for 3D hydrodynamics. *Europhysics Letters*, 4(2):291–297, 1986.

[99] H-Q Ding. Simulating lattice QCD on a Caltech/JPL hypercube. *The International Journal of Supercomputer Applications*, 5(2):73–80, Summer 1991.

[100] G D Doolen, U Frisch, B Hasslacher, S Orszag, and S Wolfram, editors. *Lattice Gas Methods for Partial Differential Equations. A Volume of Lattice Gas Reprints and Articles, Including Selected Papers from the Workshop on Large Nonlinear Systems, Los Alamos, August 1987*, volume IV of *Santa Fe Institute Studies in the Sciences of Complexity.* Addison-Wesley, 1990.

[101] P W Dowd. High performance interprocessor communication through optical wavelength division multiple access channels. In *Proc. 18th Annual International Symposium on Computer Architecture*, pages 96–105, 1991.

[102] H L Dreyfus. *What Computers Can't Do: A Critique of Artificial Reason.* Harper and Row, 1972.

[103] H L Dreyfus. *Being In The World: A commentary on Heidegger's Being and Time, Division I.* MIT Press, Cambridge, MA, 1991.

[104] H L Dreyfus and S E Dreyfus. Making a mind versus modeling a mind: Artificial intelligence back at a branch point. *Daedalus*, 117(1):15–43, 1988.

[105] I S Duff, A M Erisman, and J K Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, UK, 1986.

[106] P E Dunne. *The Complexity of Boolean Networks.* Academic Press, 1988.

[107] C Ebeling. Flexible hardware for computing complex evaluation functions. Technical Report 91-07-03, Department of Computer Science, University of Washington, July 1991.

[108] C Ebeling, G Borriello, S A Hauck, D Song, and E A Walkup. TRIPTYCH: A new FPGA architecture. Technical Report 91-09-05, Department of Computer Science, University of Washington, September 1991.

[109] J C Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3):107–119, 1991.

[110] M M Eshaghian, D K Panda, and V K Prasanna Kumar. Resource requirements for digital computations on electrooptical systems. *Applied Optics*, 30(8):928–935, March 1991.

[111] N H Farhat, D Psaltis, A Prata, and E Paek. Optical implementation of the Hopfield model. *Applied Optics*, 24(10):1469–1475, May 1985.

[112] D Farmer, T Toffoli, and S Wolfram, editors. *Cellular Automata. Proceedings of an Interdisciplinary Workshop, Los Alamos, 7-11 March 1983.* North Holland, 1984.

[113] D G Feitelson. *Optical Computing. A Survey for Computer Scientists.* MIT Press, Cambridge, MA, 1988.

[114] R P Feynman. There's plenty of room at the bottom. *Engineering and Science*, February 1960.

[115] A Fiat and A Shamir. Polymorphic arrays: A novel VLSI layout for systolic computers. In *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 37–45, 1984.

[116] P M Flanders, R L Hellier, H D Jenkins, C J Pavelin, and S van den Berghe. Efficient high-level programming on the AMT DAP. *Proceedings of the IEEE*, 79(4):524–536, April 1991.

[117] R W Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[118] R W Floyd and J D Ullman. The compilation of regular expressions into integrated circuits. *Journal of the ACM*, 29(2):603–622, 1982.

[119] J A Fodor and Z W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–72, 1988.

[120] F Fogelman Soulie, Y Robert, and M Tchuente. *Automata Networks in Computer Science: Theory and Applications.* Manchester University Press, 1987.

[121] M J Foster and H T Kung. The design of special purpose VLSI chips. *IEEE Computer*, 13:26–40, 1980.

[122] M J Foster and H T Kung. Recognizing regular languages with programmable building blocks. In J.P. Gray, editor, *VLSI 81*, pages 75–84. Academic Press, 1981.

[123] G C Fox, M A Johnson, G A Lyzenga, S W Otto, J K Salmon, and D W Walker. *Solving Problems on Concurrent Processors: Volume 1. General Techniques and Regular Problems.* Prentice Hall, 1988.

[124] U Frisch, D d'Humiéres, B Hasslacher, P Lallemand, Y Pomeau, and J P Rivet. Lattice gas hydrodynamics in two and three dimensions. *Complex Systems*, 1:649–707, 1987.

[125] U Frisch, B Hasslacher, and Y Pomeau. Lattice gas automata for the Navier-Stokes equation. *Physical Review Letters*, 56:1505–1508, 1986.

[126] P Frison and P Quinton. Systolic architectures for connected speech recognition. In R De Mori and C Y Suen, editors, *New Systems and Architectures for Automatic Speech Recognition and Synthesis, NATO ASI Series Vol. F16*, pages 145–167. Springer-Verlag, 1985.

[127] H Fuchs, J Poulton, J Eyles, T Greer, J Goldfeather, D Ellsworth, S Molnar, and G Turk. Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics*, 23(4):79–88, July 1989.

[128] R M Fujimoto. The virtual time machine. In *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 199–208, 1989.

[129] R M Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[130] R M Fujimoto, J-J Tsai, and G C Gopalakrishnan. Design and evaluation of the rollback chip: Special purpose hardware for Time Warp. *IEEE Transactions on Computers*, 41(1):68–82, January 1992.

[131] K A Gallivan, M T Heath, E Ng, J M Ortega, B W Peyton, R J Plemmons, C H Romine, A H Sameh, and R G Voigt. *Parallel algorithms for matrix computations.* SIAM Press, 1990.

[132] K A Gallivan, R J Plemmons, and A H Sameh. Parallel algorithms for dense linear algebra computations. In *Parallel algorithms for matrix computations,* K A Gallivan, M T Heath, E Ng, J M Ortega, B W Peyton, R J Plemmons, C H Romine, A H Sameh and R G Voigt, pages 1–82. SIAM Press, 1990.

[133] M Gardner. The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123, April 1970.

[134] Gaussian Inc. Documentation on Gaussian 86 and Gaussian 88. Technical report, Gaussian Inc., Gaussian Inc., 4415 Fifth Avenue, Pittsburgh, PA 15213, 1988.

[135] C W Gear, editor. *Computation and Cognition. Proceedings of the First NEC Research Symposium.* SIAM Press, 1991. Panel Session - The Future of Parallelism, pages 153-168.

[136] G A Geist and C H Romine. LU factorization algorithms on distributed memory multiprocessor architectures. *SIAM Journal on Scientific and Statistical Computing*, 9:639–649, 1988.

[137] W. Gentleman and H. Kung. Matrix triangularization by systolic arrays. In *Proc. SPIE 298, Real-time Signal Processing IV*, pages 19–26, 1981.

[138] A George, M T Heath, J Liu, and E Ng. Sparse Cholesky factorization on a local-memory multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 9(2):327–340, March 1988.

[139] A. George and E. Ng. Some shared memory is desirable in parallel sparse matrix computations. *SIGNUM Newsletter*, 23(2):9–13, 1988.

[140] M Gereb-Graus and T Tsantilas. Efficient optical communication in parallel computers. In *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[141] A M Gibbons and P Spirakis, editors. *Lectures on Parallel Computation. Proc. 1991 ALCOM Spring School on Parallel Computation.* Cambridge International Series on Parallel Computation. Cambridge University Press, Cambridge, UK, 1992.

[142] L A Glasser and D W Dobberpuhl. *The Design and Analysis of VLSI Circuits.* Addison-Wesley, 1985.

[143] G Graff. Materials scientists put the squeeze on electrons. *Science*, 254:1306–1307, 29th November 1991.

[144] S A Green and D J Paddon. A highly flexible multiprocessor solution for ray tracing. *The Visual Computer*, 6(2):62–73, March 1990.

[145] R I Greenberg and C E Leiserson. Randomized routing on fat-trees. In *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 241–249, 1985.

[146] R I Greenberg and C E Leiserson. A compact layout for the three-dimensional tree of meshes. *Applied Mathematics Letters*, 1(2):171–176, 1988.

[147] L Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* MIT Press, Cambridge, MA, 1988.

[148] L Greengard and W D Gropp. A parallel version of the fast multipole method. Research Report YALEU/DCS/RR-640, Department of Computer Science, Yale University, August 1988.

[149] J Gruska. Systolic architectures, systems and computations. In T Lepistö and A Salomaa, editors, *Proc. 15th International Colloquium on Automata, Languages and Programming. LNCS Vol.317*, pages 254–270. Springer-Verlag, 1988.

[150] L J Guibas, H T Kung, and C D Thompson. Direct VLSI implementation of combinatorial algorithms. In C L Seitz, editor, *Proc. Caltech Conference on VLSI*, pages 509–525, 1979.

[151] L J Guibas and F M Liang. Systolic stacks, queues, and counters. In P Penfield Jr, editor, *Proc. MIT Conference on Advanced Research in VLSI*, pages 155–164, January 1982.

[152] J L Gustafson. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal on Scientific and Statistical Computing*, 9(4):609–638, July 1988.

[153] J L Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, May 1988.

[154] S E Hambrusch and J Simon. Solving undirected graph problems on VLSI. *SIAM Journal on Computing*, 14:527–544, 1985.

[155] J M Hammersley and D C Handscomb. *Monte Carlo Methods*. Methuen, 1964.

[156] J Hardy, Y Pomeau, and O de Pazzis. Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions. *Journal of Mathematical Physics*, 14:1746, 1973.

[157] A Hartmann and S Redfield. Design sketches for optical crossbar switches intended for large scale parallel processing applications. *Optical Engineering*, 29(3):315–327, 1989.

[158] T F Havel, I D Kuntz, and G M Crippen. The theory and practice of distance geometry. *Bulletin of Mathematical Biology*, 45(5):665–720, 1983.

[159] M T Heath, E Ng, and B W Peyton. Parallel algorithms for sparse linear systems. In *Parallel algorithms for matrix computations*, K A Gallivan, M T Heath, E Ng, J M Ortega, B W Peyton, R J Plemmons, C H Romine, A H Sameh and R G Voigt, pages 83–124. SIAM Press, 1990.

[160] M T Heath and C H Romine. Parallel solution of triangular systems on distributed memory multiprocessors. *SIAM Journal on Scientific and Statistical Computing*, 9:558–588, 1988.

[161] W J Hehre, L Radom, P v.R Schleyer, and J A Pople. *Ab Initio Molecular Orbital Theory*. John Wiley and Sons, 1986.

[162] C S Henkel and R J Plemmons. Recursive least squares on a hypercube multiprocessor using the covariance factorization. *SIAM Journal on Scientific and Statistical Computing*, 12(1):95–106, January 1991.

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

[164] M Hennessy. Proving systolic systems correct. *ACM Transactions on Programming Languages and Systems*, 8(3):344–387, 1986.

[165] P Hilbers. *Processor Networks and Aspects of the Mapping Problem*, volume 2 of *Cambridge International Series on Parallel Computation*. Cambridge University Press, Cambridge, UK, 1991.

[166] W D Hillis. *The Connection Machine*. MIT Press, Cambridge, MA, 1985.

[167] A Hinchliffe. *Ab Initio Determination of Molecular Properties*. Adam Hilger, 1987.

[168] C A R Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[169] C A R Hoare, editor. *Developments in Concurrency and Communication.* University of Texas Year of Programming Institute on Concurrent Programming. Addison-Wesley, 1990.

[170] C A R Hoare. The transputer and occam: A personal story. *Concurrency: Practice and Experience*, 3(4):249–264, August 1991.

[171] J W Hong, K Mehlhorn, and A L Rosenberg. Cost trade-offs in graph embeddings, with applications. *Journal of the ACM*, 30:709–728, 1983.

[172] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79(8):2554–2558, April 1982.

[173] E Housos, C C Huang, and J M Liu. Parallel algorithms for the AT&T KORBX system. *AT&T Technical Journal*, 68(3):37–47, May/June 1989.

[174] F-H Hsu. Large-scale parallelization of alpha-beta search: An algorithmic and architectural study with computer chess. Ph.D. Thesis. Technical Report CMU-CS-90-108, School of Computer Science, Carnegie Mellon University, February 1990.

[175] F-H Hsu, T Anantharaman, M Campbell, and A Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, October 1990.

[176] A Huang. Architectural considerations involved in the design of an optical digital computer. *Proceedings of the IEEE*, 72(7):780–786, July 1984.

[177] C Huang and C Lengauer. The derivation of systolic implementations of programs. *Acta Informatica*, 24:595–632, 1987.

[178] M D A Huang. Solving some graph problems with optimal or near-optimal speedup on mesh-of-trees networks. In *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 232–240, 1985.

[179] J M Hutchinson and S A Zenios. Financial simulations on a massively parallel Connection Machine. *The International Journal of Supercomputer Applications*, 5(2):27–45, Summer 1991.

[180] O H Ibarra, T Jiang, and H Wang. String editing on a one-way linear array of finite-state machines. *IEEE Transactions on Computers*, 41(1):112–118, January 1992.

[181] INMOS Limited. *Transputer Reference Manual.* Prentice Hall, 1988.

[182] I C F Ipsen and E R Jessup. Solving the symmetric tridiagonal eigenvalue problem on the hypercube. *SIAM Journal on Scientific and Statistical Computing*, 11(2):203–229, March 1990.

[183] J Ja'Ja'. The VLSI complexity of selected graph problems. *Journal of the ACM*, 31:377–391, 1984.

[184] J Ja'Ja' and P Kumar. Information transfer in distributed computing with application to VLSI. *Journal of the ACM*, 31:150–162, 1984.

[185] S L Johnsson. Communication in network architectures. In R Suaya and G Birtwistle, editors, *VLSI and Parallel Computaion*, pages 223–389. Morgan Kaufmann, San Mateo, CA, 1990.

[186] S L Johnsson and K K Mathur. Data structures and algorithms for the finite element method on a data parallel supercomputer. *International Journal on Numerical Methods in Engineering*, 29(4):881–908, 1990.

[187] G Jones and M Goldsmith. *Programming in occam 2.* Prentice Hall, 1988.

[188] J S Judd. *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge, MA, 1990.

[189] A Kaldewaij and M Rem. The derivation of systolic computations. *Science of Computer Programming*, 14:229–242, 1990.

[190] R Kane and S Sahni. A systolic design rule checker. In *Proc. 21st Annual IEEE Design Automation Conference*, pages 243–250, 1984.

[191] N Karmarkar. A new parallel architecture for sparse matrix computation based on finite projective geometries. In *Proc. Supercomputing 91*, pages 358–369. IEEE Press, 1991.

[192] R M Karp, R E Miller, and S Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14:563–590, 1967.

[193] M Kearns. *The Computational Complexity of Machine Learning.* MIT Press, Cambridge, MA, 1990.

[194] Z M Kedem and A Zorat. On relations between input and communication/computation in VLSI. In *Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science*, pages 37–44, 1981.

[195] T Khanna. *Foundations of Neural Networks.* Addison-Wesley, 1990.

[196] S Kirkpatrick, C D Gelatt Jr, and M P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[197] D Kirsh. Foundations of AI: The big issues. *Artificial Intelligence*, 47(1-3):3–30, January 1991.

[198] D Kirsh. Today the earwig, tomorrow man? *Artificial Intelligence*, 47(1-3):161–184, January 1991.

[199] R Koch, F T Leighton, B M Maggs, S B Rao, and A L Rosenberg. Work-preserving emulations of fixed-connection networks. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 227–240, 1989.

[200] S R Kosaraju and M J Atallah. Optimal simulations between mesh-connected arrays of processors. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 264–272, 1986.

[201] M Kramer and J van Leeuwen. The VLSI complexity of Boolean functions. In E Börger et al., editors, *Logic and Machines: Decision Problems and Complexity, LNCS Vol.171*, pages 397–407, 1984.

[202] S A Kravitz, R E Bryant, and R A Rutenbar. Massively parallel switch-level simulation: A feasibility study. *IEEE Transactions on Computer-Aided Design*, 10(7):871–894, July 1991.

[203] S D Kugelmass, R Squier, and K Steiglitz. Performance of VLSI engines for lattice computations. *Complex Systems*, 1(5):939–965, October 1987.

[204] S D Kugelmass and K Steiglitz. A scalable architecture for lattice-gas simulations. *Journal of Computational Physics*, 84:311–325, 1989.

[205] H T Kung. Let's design algorithms for VLSI systems. In *Proc. Caltech Conference on Very Large Scale Integration*, pages 65–90, 1979.

[206] H T Kung. The structure of parallel algorithms. *Advances in Computers*, 19:65–112, 1980.

[207] H T Kung. Why systolic architectures? *IEEE Computer*, 15(1):37–46, January 1982.

[208] H T Kung. New opportunities in multicomputers. In C W Gear, editor, *Computation and Cognition. Proceedings of the First NEC Research Symposium*, pages 1–21. SIAM Press, 1991.

[209] H T Kung and M S Lam. Fault-tolerance and two-level pipelining in VLSI systolic arrays. *Journal of Parallel and Distributed Computing*, 1(1):32–63, 1984.

[210] H T Kung and P L Lehman. Systolic (VLSI) arrays for relational database operations. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 105–116, 1980.

[211] H T Kung and C E Leiserson. Systolic arrays (for VLSI). In I S Duff and G W Stewart, editors, *Sparse Matrix Proceedings*, pages 256–282. SIAM Press, 1978.

[212] H T Kung and J A Webb. Mapping image processing operations onto a linear systolic array. *Distributed Computing*, 1(4):246–257, 1986.

[213] S Y Kung. On supercomputing with systolic/wavefront array processors. *Proceedings of the IEEE*, pages 867–884, July 1984.

[214] S Y Kung. *VLSI Array Processors.* Prentice Hall, 1988.

[215] S Y Kung, K S Arun, R J Gal-Ezer, and D B Bhaskar Rao. Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers*, 31(11):1054–1065, November 1982.

[216] S Y Kung, S-C Lo, and P S Lewis. Optimal systolic design for the transitive closure and the shortest path problems. *IEEE Transactions on Computers*, C-36(5):603–614, May 1987.

[217] S Y Kung, H J Whitehouse, and T Kailath, editors. *VLSI and Modern Signal Processing.* Prentice Hall, 1985.

[218] R E Ladner and M J Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[219] H-W Lang, M Schimmler, H Schmeck, and H Schréder. Systolic sorting on a mesh-connected network. *IEEE Transactions on Computers*, 34(7):652–658, July 1985.

[220] C G Langton. Self-reproduction in cellular automata. In Farmer et al. [112], pages 135–144.

[221] H Le Verge, C Mauras, and P Quinton. The ALPHA language and its use for the design of systolic arrays. *Journal of VLSI Signal Processing*, 3(3):173–182, September 1991.

[222] F T Leighton. *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, MA, 1983.

[223] F T Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, 34(4):344–354, April 1985.

[224] F T Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[225] F T Leighton and C E Leiserson. Wafer-scale integration of systolic arrays. *IEEE Transactions on Computers*, C-34, No.5:448–461, May 1985.

[226] F T Leighton and C E Leiserson. Advanced parallel and VLSI computation. Research Seminar Series MIT/LCS/RSS 7, Laboratory for Computer Science, Massachusetts Institute of Technology, December 1989.

[227] F T Leighton, C E Leiserson, and M Klugerman. Theory of parallel and VLSI computation. Research Seminar Series MIT/LCS/RSS 10, Laboratory for Computer Science, Massachusetts Institute of Technology, January 1991.

[228] F T Leighton and B M Maggs. Expanders might be practical: Fast algorithms for routing around faults on multibutterflies. In *Proc. 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 384–389, 1989.

[229] F T Leighton and A L Rosenberg. Three-dimensional circuit layouts. *SIAM Journal on Computing*, 15:793–813, 1986.

[230] C E Leiserson. Systolic priority queues. In *Proc. Caltech Conference on Very Large Scale Integration*, pages 199–214, 1979.

[231] C E Leiserson. *Area-Efficient VLSI Computation.* MIT Press, Cambridge, MA, 1983.

[232] C E Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C–34(10):892–901, October 1985.

[233] C E Leiserson and B M Maggs. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica*, 3:53–77, 1988.

[234] C E Leiserson and J Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[235] D B Lenat and E A Feigenbaum. On the thresholds of knowledge. *Artificial Intelligence*, 47(1-3):185–250, January 1991.

[236] C Lengauer, M Barnett, and D G Hudson. Towards systolizing compilation. *Distributed Computing*, 5(1):7–24, 1991.

[237] C Lengauer and J Xue. A systolic array for pyramidal algorithms. *Journal of VLSI Signal Processing*, 3(3):237–257, September 1991.

[238] T Lengauer. The communication complexity of VLSI circuits. In *Proc. AMS Symposium in Applied Mathematics 31*, pages 59–82, 1985.

[239] T Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner-Wiley Series of Applicable Theory in Computer Science. Teubner/Wiley, 1990.

[240] T Lengauer. VLSI theory. In J van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity*, pages 835–868. North Holland, 1990.

[241] G-J Li and B W Wah. Systolic processing for dynamic programming problems. In D DeGroot, editor, *Proc. IEEE International Conference on Parallel Processing*, pages 434–441. IEEE Press, 1985.

[242] P P Li and A J Martin. The sneptree - a versatile interconnection network. In *Proc. 1986 IEEE International Conference on Parallel Processing*, pages 20–27, 1986.

[243] Y-Z Liang and H K Liu. Optical matrix-matrix multiplication method demonstrated by the use of a multifocus hololens. *Optics Letters*, 9(8):322–324, August 1984.

[244] Y-C Lin and F-C Lin. A family of systolic arrays for relational database operations. In W Moore, A McCabe, and R Urquhart, editors, *Systolc Arrays*, pages 191–200, 1986.

[245] K Lindgren and M G Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

[246] R J Lipton and D P Lopresti. A systolic array for rapid string comparison. In H. Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 363–376. Computer Science Press, 1985.

[247] R J Lipton and R Sedgewick. Lower bounds for VLSI. In *Proc. 13th Annual ACM Symposium on Theory of Computing*, pages 300–307, 1981.

[248] R J Lipton and R E Tarjan. A planar separator theorem. *SIAM Journal on Applied Mathematics*, 36(2):177–189, April 1979.

[249] R J Lipton and R E Tarjan. Applications of a separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[250] B Lisper. *Synthesis of Synchronous Systems by Static Scheduling in Space-Time.* Springer-Verlag, 1989. LNCS Vol.362.

[251] A W Lohmann. What classical optics can do for the digital optical computer. *Applied Optics*, 25(10):1543–1549, May 1986.

[252] A Louri. Three-dimensional optical architecture and data-parallel algorithms for massively parallel computing. *IEEE Micro*, 11(2):24–27,65–82, April 1991.

[253] W Luk and I Page. Parametrising designs for field-programmable gate arrays. In W Moore and W Luk, editors, *FPGAs. Proc. International Workshop on Field Programmable Logic and Applications, Oxford, 1991*, pages 284–295. Abingdon EE&CS Books, UK, 1991.

[254] P Maes, editor. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back.* MIT Press, Cambridge, MA, 1990.

[255] B M Maggs and S A Plotkin. Minimum-cost spanning tree as a path-finding problem. *Information Processing Letters*, 26:291–293, 1988.

[256] M A Mahowald and C A Mead. The silicon retina. *Scientific American*, 264(5):76–82, May 1991.

[257] J Makino, T Ito, T Ebisuzaki, and D Sugimoto. Grape: A special-purpose computer for n-body problems. In J A B Fortes, S Y Kung, K W Przytula, and E E Swartzlander Jr, editors, *Proc. IEEE International Conference on Application Specific Array Processors, Princeton, New Jersey*, pages 180–189. IEEE Press, 1990.

[258] S Manohar. Systolic architectures: A critical survey. Technical Report CS-86-05, Department of Computer Science, Brown University, March 1986.

[259] N Margolus. Physics-like models of computation. In Farmer et al. [112], pages 81–95.

[260] N Margolus and T Toffoli. Cellular automata machines. *Complex Systems*, 1(5):967–993, October 1987.

[261] N Margolus and T Toffoli. Cellular automata machines. In Doolen et al. [100], pages 219–249.

[262] A J Martin. Compiling communicating processes into delay insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[263] A J Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In Hoare [169], pages 1–64.

[264] A J Martin and J L A van de Snepscheut. Networks of machines for distributed recursive computations. Computing Science Note CS 8502, Department of Computing Science, University of Groningen, 1985.

[265] A J Martin and J L A van de Snepscheut. An interconnection network for distributed recursive computations. *IEEE Transactions on Computers*, 39(11):1393–1395, November 1990.

[266] O A McBryan and E F Van De Velde. Hypercube algorithms and implementations. *SIAM Journal on Scientific and Statistical Computing*, 8(2):s227–s287, March 1987.

[267] J A McCammon and S C Harvey. *Dynamics of Proteins and Nucleic Acids.* Cambridge University Press, Cambridge, UK, 1987.

[268] J McCanny, J McWhirter, and E Swartzlander, editors. *Systolic Array Processors.* Prentice Hall, 1989.

[269] W F McColl. Planar crossovers. *IEEE Transactions on Computers*, 30(3):223–225, March 1981.

[270] W F McColl. Planar circuits have short specifications. In K Mehlhorn, editor, *2nd Annual Symposium on Theoretical Aspects of Computer Science, LNCS Vol. 182*, pages 231–242, 1985.

[271] W F McColl. Parallel algorithms and architectures. In G A van Zee and J G G van de Vorst, editors, *Parallel Computing 1988, Shell Conference, LNCS Vol. 384*, pages 1–22. Springer-Verlag, 1989.

[272] W F McColl. General purpose parallel computing. This volume.

[273] W F McColl, M S Paterson, and B H Bowditch. Planar acyclic computation. *Information and Computation*, 90(2):178–193, February 1991.

[274] W S McCulloch and W H Pitts. A logical calculus for the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[275] K McEvoy and J V Tucker. Theoretical foundations of hardware design. In K McEvoy and J V Tucker, editors, *Theoretical Foundations of VLSI Design*, volume 10 of *Cambridge Tracts in Theoretical Computer Science*, pages 1–62. Cambridge University Press, Cambridge, UK, 1990.

[276] K McEvoy and J V Tucker, editors. *Theoretical Foundations of VLSI Design*, volume 10 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 1990.

[277] J G McWhirter and J V McCanny. Systolic and wavefront arrays. In J V McCanny and J C White, editors, *VLSI Technology and Design*, pages 253–299. Academic Press, 1987.

[278] C A Mead. *Analog VLSI and Neural Systems.* Addison-Wesley, 1989.

[279] C A Mead and L A Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[280] C A Mead and M Rem. Cost and performance of VLSI computing structures. *IEEE Journal on Solid State Circuits*, SC-14(2):455–462, 1979.

[281] K Mehlhorn. AT^2-optimal VLSI integer division and square rooting. *Integration, The VLSI Journal*, 2:163–167, 1984.

[282] K Mehlhorn and F P Preparata. Area-time optimal VLSI integer multiplier with minimum computation time. *Information and Computation*, 58(1-3):137–156, July/August/September 1983.

[283] K Mehlhorn and F P Preparata. Area-time optimal division for $T = \Omega((\log n)^{1+\epsilon})$. *Information and Computation*, 72(3):270–282, March 1987.

[284] R Miller and Q F Stout. Data movement techniques for the pyramid computer. *SIAM Journal on Computing*, 16(1):38–60, February 1987.

[285] M Minsky and S Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[286] M Minsky and S Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, third edition, 1988. Epilog: The New Connectionism, pages 247-280.

[287] J Misra. Distributed-discrete event simulation. *ACM Computing Surveys*, 18(1):39–65, March 1986.

[288] D I Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, 71(1):113–118, 1983.

[289] B Monien and I H Sudborough. Comparing interconnection networks. In M P Chytil, L Janiga, and V Koubek, editors, *Proc. 13th Symposium on Mathematical Foundations of Computer Science. LNCS Vol.324*, pages 138–153, 1988.

[290] W Moore and W Luk, editors. *FPGAs. Proc. International Workshop on Field Programmable Logic and Applications, Oxford, 1991.* Abingdon EE&CS Books, UK, 1991.

[291] W Moore, A McCabe, and R Urquhart, editors. *Systolic Arrays.* Adam Hilger, 1987.

[292] D Nassimi and S Sahni. Bitonic sort on a mesh-connected parallel computer. *IEEE Transactions on Computers*, 28(1):2–7, 1979.

[293] B K Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, San Mateo, CA, 1991.

[294] N J Nilsson. Logic and artificial intelligence. *Artificial Intelligence*, 47(1-3):31–56, January 1991.

[295] M H Nodine, D P Lopresti, and J S Vitter. I/O overhead and parallel VLSI architectures for lattice computations. *IEEE Transactions on Computers*, 40(7):843–852, July 1991.

[296] J M Ortega, R G Voigt, and C H Romine. A bibliography on parallel and vector numerical algorithms. In *Parallel algorithms for matrix computations,* K A Gallivan, M T Heath, E Ng, J M Ortega, B W Peyton, R J Plemmons, C H Romine, A H Sameh and R G Voigt, pages 125–197. SIAM Press, 1990.

[297] T Ottmann, A L Rosenberg, and L J Stockmeyer. A dictionary machine (for VLSI). *IEEE Transactions on Computers*, 31:892–897, 1982.

[298] I Page and W Luk. Compiling occam into field-programmable gate arrays. In W Moore and W Luk, editors, *FPGAs. Proc. International Workshop on Field Programmable Logic and Applications, Oxford, 1991*, pages 271–283. Abingdon EE&CS Books, UK, 1991.

[299] V Pan and J H Reif. Compact multigrid. *SIAM Journal on Scientific and Statistical Computing*, 13(1):119–127, January 1992.

[300] R Paturi, D-T Lu, J E Ford, S C Esener, and S H Lee. Parallel algorithms based on expander graphs for optical computing. *Applied Optics*, 30(8):917–927, March 1991.

[301] R Penrose. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics.* Oxford University Press, Oxford, UK, 1989.

[302] N J Pippenger. Developments in "The synthesis of reliable organisms from unreliable components". In J Glimm, J Impagliazzo, and I Singer, editors, *The Legacy of John von Neumann. Proc. of Symposia in Pure Mathematics*, volume 50, pages 311–324. American Mathematical Society, 1990. Proc. Summer Research Institute on the Legacy of John von Neumann, Hofstra University, May 29-June 4, 1988.

[303] M Potmesil and E M Hoffert. The Pixel Machine: A parallel image computer. *Computer Graphics*, 23(3):69–78, 1989.

[304] J L Potter. Image processing on the Massively Parallel Processor. *IEEE Computer*, 16(1):62–67, January 1983.

[305] F P Preparata. Optimal three-dimensional VLSI layouts. *Mathematical Systems Theory*, 16:1–8, 1983.

[306] F P Preparata and J Vuillemin. Area-time optimal VLSI networks for multiplying matrices. *Information Processing Letters*, 11(2):77–80, 1980.

[307] F P Preparata and J Vuillemin. The Cube-Connected Cycles: A versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, 1981.

[308] P Quinton. The systematic design of systolic arrays. In F Fogelman Soulie, Y robert, and M Tchuente, editors, *Automata Networks in Computer Science Theory and Applications*, pages 229–260. Manchester University Press, 1987.

[309] P Quinton and Y Robert. *Systolic Algorithms and Architectures.* Prentice Hall, 1991. Translated by Iain Craig.

[310] P Quinton and V Van Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1:95–113, 1989.

[311] S V Rajopadhye. Systolic arrays for LU-decomposition: An application of formal synthesis techniques. *International Journal of Computer Aided VLSI Design*, 3(1):69–90, 1991.

[312] S Ranka and S Sahni. *Hypercube Algorithms for Image Processing and Pattern Recognition.* Springer-Verlag, 1990.

[313] S B Rao. Properties of an interconnection architecture based on wavelength divsion multiplexing. Technical Report TR-92-009-3-0054-2, NEC Research Institute, Princeton, January 1992.

[314] S K Rao and T Kailath. Regular iterative algorithms and their implementations on processor arrays. *Proceedings of the IEEE*, 76(2):259–282, March 1988.

[315] J H Reif and A Tyagi. Efficient parallel algorithms for optical computing with the DFT primitive. In K V Nori and C E Veni Madhavan, editors, *Proc. 10th Conference on Foundations of Software Technology and Theoretical Computer Science. LNCS Vol.472*, pages 149–160. Springer-Verlag, 1990.

[316] T Risset and Y Robert. Synthesis of processor arrays for the algebraic path problem: Unifying old results and deriving new architectures. *Parallel Processing Letters*, 1(1):19–28, 1991.

[317] Y Robert. Systolic algorithms and architectures. In F Fogelman Soulié, Y Robert, and M Tchuente, editors, *Automata Networks in Computer Science: Theory and Applications*, pages 187–228. Manchester University Press, 1987.

[318] Y Robert. *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm.* Manchester University Press, 1990.

[319] Y Robert and M Tchuente. A systolic array for the longest common subsequence problem. *Information Processing Letters*, 21:191–198, October 1985.

[320] Y Robert and M Tchuente. An efficient systolic array for the 1D recursive convolution problem. *Journal of VLSI and Computer Systems*, 1(4):398–408, 1986.

[321] Y Robert and D Trystram. An orthogonal systolic array for the algebraic path problem. *Computing*, 39:187–199, 1987.

[322] A L Rosenberg. Three-dimensional VLSI: A case study. *Journal of the ACM*, 30:397–416, 1983.

[323] A L Rosenberg. On validating parallel architectures via graph embeddings. In S K Tewksbury, B W Dickinson, and S C Schwartz, editors, *Concurrent Computations - Algorithms, Architecture, and Technology*, pages 99–115. (Proc. 1987 Princeton Workshop on Algorithm, Architecture and Technology Issues for Models of Concurrent Computation), Plenum Press, 1988.

[324] F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[325] G Rote. A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion). *Computing*, 34:191–219, 1985.

[326] V P Roychowdhury and T Kailath. Study of parallelism in regular iterative algorithms. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 367–376, 1990.

[327] D E Rumelhart and J L McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volumes 1 and 2.* MIT Press, Cambridge, MA, 1986.

[328] S Ryckebusch, J M Bower, and C A Mead. Modeling small oscillating biological networks in analog VLSI. In D S Touretzky, editor, *Advances in Neural Information Processing I*, pages 384–393. Morgan Kaufmann, San Mateo, CA, 1989.

[329] D Sankoff and J B Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983.

[330] J E Savage. Computational work and time on finite machines. *Journal of the ACM*, 19(4):660–674, 1972.

[331] J E Savage. Area-time tradeoffs for matrix multiplication and related problems. *Journal of Computer and System Sciences*, 22(2):230–242, April 1981.

[332] J E Savage. Planar circuit complexity and the performance of VLSI algorithms. In H T Kung, B Sproull, and G Steele, editors, *VLSI Systems and Computations*, pages 61–68. Computer Science Press, 1981. (Expanded version appears as INRIA Report No.77 (1981).).

[333] J E Savage. The performance of multilective VLSI algorithms. *Journal of Computer and System Sciences*, 29:243–273, 1984.

[334] A A Sawchuk, B K Jenkins, C S Raghavendra, and A Varma. Optical crossbar networks. *IEEE Computer*, 20(6):50–60, 1987.

[335] C P Schnorr and A Shamir. An optimal sorting algorithm for mesh connected computers. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 255–263, 1986.

[336] E Schwabe. On the computational equivalence of hypercube-derived networks. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 388–397, 1990.

[337] J R Searle. Is the brain's mind a computer program? *Scientific American*, 262(1):26–31, January 1990.

[338] C L Seitz. System timing. In *Introduction to VLSI Systems,* C A Mead and L A Conway, chapter 7, pages 218–262. Addison-Wesley, 1980.

[339] C L Seitz. The Cosmic Cube. *Communications of the ACM*, 28(1):22–33, January 1985.

[340] C L Seitz. Concurrent architectures. In R Suaya and G Birtwistle, editors, *VLSI and Parallel Computaion*, pages 1–84. Morgan Kaufmann, San Mateo, CA, 1990.

[341] C L Seitz. Multicomputers. In Hoare [169], pages 131–200.

[342] C L Seitz, W C Athas, W J Dally, R Faucette, A S Martin, S Mattison, C D Steele, and W-K Su. *Message-Passing Concurrent Computers: Their Architecture & Programming.* Addison-Wesley, 1989.

[343] T J Sejnowski, C Koch, and P S Churchland. Computational neuroscience. *Science*, 241:1299–1306, 9th September 1988.

[344] T J Sejnowski and C R Rosenberg. NETtalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, 32pp, Electrical Engineering and Computer Science, Johns Hopkins University, 1986. Also appeared as *Parallel networks that learn to pronounce english text* in Complex Systems, Volume 1, Number 1, February 1987, pages 145-168.

[345] M Shand, P Bertin, and J Vuillemin. Hardware speedups in long integer multiplication. In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 138–145, 1990.

[346] M Sheeran. Describing and reasoning about circuits using relations. In K McEvoy and J V Tucker, editors, *Theoretical Foundations of VLSI Design*, volume 10 of *Cambridge Tracts in Theoretical Computer Science*, pages 263–298. Cambridge University Press, Cambridge, UK, 1990.

[347] Z-C Shih, G-H Chen, and R C T Lee. Systolic algorithms to examine all pairs of elements. *Communications of the ACM*, 30(2):161–167, February 1987.

[348] A Siegel. Aspects of information flow in VLSI circuits. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 448–459, 1986.

[349] U C Singh et al. AMBER Version 3.0 - Assisted Model Building with Energy Refinement. Manual, University of California, 1986.

[350] H S Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C-20(2):153–161, 1971.

[351] Q F Stout. Algorithm-guided design considerations for meshes and pyramids. In M.J.B. Duff, editor, *Intermediate-Level Image Processing*, pages 149–165. Academic Press, 1986.

[352] J P Strong. Computations on the Massively Parallel Processor at the Goddard Space Flight Center. *Proceedings of the IEEE*, 79(4):548–558, April 1991.

[353] M R Stytz, G Frieder, and O Frieder. Three-dimensional medical imaging: Algorithms and computer systems. *ACM Computing Surveys*, 23(4):421–499, December 1991.

[354] I E Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[355] I E Sutherland and C A Mead. Microelectronics and computer science. *Scientific American*, 237(3):210–228, September 1977.

[356] C D Thomborson and B W-Y Wei. Systolic implementation of a move-to-front text compressor. In *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 283–290, 1989.

[357] C D Thompson. Area-time complexity for VLSI. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, pages 81–88, 1979.

[358] C D Thompson. Fourier transforms in VLSI. *IEEE Transactions on Computers*, 32(11):1047–1057, November 1983.

[359] C D Thompson. The VLSI complexity of sorting. *IEEE Transactions on Computers*, 32(12):1171–1184, 1983.

[360] C D Thompson and H T Kung. Sorting on a mesh connected computer. *Communications of the ACM*, 20(4):263–271, 1977.

[361] M Tinker. DVI parallel image compression. *Communications of the ACM*, 32(7):844–851, July 1989.

[362] T Toffoli. CAM: A high-performance cellular-automaton machine. In Farmer et al. [112], pages 195–204.

[363] T Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modelling physics. *Physica*, 10D:117, 1984.

[364] T Toffoli and N Margolus. *Cellular Automata Machines*. MIT Press, Cambridge, MA, 1987.

[365] A M Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Series 2*, 42:230–265, 1936. Corrections, *ibid.*, 43 (1937), 544-546.

[366] A M Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[367] A M Turing. Intelligent machinery. In B Meltzer and D Michie, editors, *Machine Intelligence 5*, pages 3–23. Edinburgh University Press, 1970.

[368] J T Udding. A formal model for defining and classifying delay-insensitive circuits and systems. *Distributed Computing*, 1(4):197–204, 1986.

[369] J D Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[370] H Umeo and T Asano. Systolic algorithms for computational geometry problems: A survey. *Computing*, 41:19–40, 1989.

[371] E Upfal. An O(log N) deterministic packet-routing scheme. *Journal of the ACM*, 39(1):55–70, January 1992.

[372] L G Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, 30(2):135–140, 1981.

[373] L G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[374] L G Valiant. Functionality in neural nets. In *Proc. AAAI-88 Seventh American Association for Artificial Intelligence National Conference, St Paul, Minnesota*, volume 2, pages 629–634. Morgan Kaufmann, San Mateo, CA, August 1988.

[375] L G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[376] L G Valiant. General purpose parallel architectures. In J van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity*, pages 943–971. North Holland, 1990.

[377] L G Valiant. A view of computational learning theory. In C W Gear, editor, *Computation and Cognition. Proceedings of the First NEC Research Symposium*, pages 32–51. SIAM Press, 1991.

[378] E F Van de Velde. Experiments with multicomputer LU-decomposition. *Concurrency: Practice and Experience*, 2(1):1–26, March 1990.

[379] W F van Gunsteren. The role of computer simulation techniques in protein engineering. *Protein Engineering*, 2(1):5–13, 1988.

[380] W F van Gunsteren and H J C Berendsen. Algorithms for macromolecular dynamics and constraint dynamics. *Molecular Physics*, 34(5):1311–1327, 1977.

[381] P J M van Laarhoven and E H L Aarts. *Simulated Annealing: Theory and Applications.* D.Reidel, 1987.

[382] G Y Vichniac. Simulating physics with cellular automata. In Farmer et al. [112], pages 96–116.

[383] P M B Vitányi. Locality, communication, and interconnect length in multicomputers. *SIAM Journal on Computing*, 17(4):659–672, August 1988.

[384] P M B Vitanyi. A modest proposal for communication costs in multicomputers. In S K Tewksbury, B W Dickinson, and S C Schwartz, editors, *Concurrent Computations - Algorithms, Architecture, and Technology*, pages 203–216. (Proc. 1987 Princeton Workshop on Algorithm, Architecture and Technology Issues for Models of Concurrent Computation), Plenum Press, 1988.

[385] A Vladimirescu and S Liu. The simulation of MOS integrated circuits using SPICE2. Electronics Research Laboratory Memo UCB/ERL-M80/70, Electronics Research Laboratory, University of California at Berkeley, October 1980.

[386] J von Neumann. *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components.* California Institute of Technology, 1952. Lectures delivered at California Institute of Technology, 4-15 January 1952, with notes by R S Pierce.

[387] J von Neumann. *The Computer and the Brain.* Yale University Press, 1958. Silliman Lectures, Yale University, Spring Term 1956.

[388] J von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, 1966. Edited and completed by A W Burks.

[389] J von Neumann. *First draft of a report on the EDVAC.* Moore School of Electrical Engineering, University of Pennsylvania, 30 June 1945. Contract No. W-670-ORD-4926 between the United States Army Ordnance Department and the University of Pennsylvania. Reprinted in *Papers of John von Neumann on Computing and Computer Theory*, W Aspray and A Burks, editors. Volume 12 in the Charles Babbage Institute Reprint Series for the History of Computing, MIT Press, 1987, 17-82.

[390] J Vuillemin. A combinatorial limit to the computing power of VLSI circuits. *IEEE Transactions on Computers*, 32(3):294–300, March 1983.

[391] W W Wadge and E A Ashcroft. *Lucid, The Dataflow Programming Language.* Academic Press, 1985.

[392] P Wallich. Silicon babies. *Scientific American*, 265(6):124–134, December 1991.

[393] S Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[394] I Wegener. *The Complexity of Boolean Functions.* John Wiley and Sons, 1987.

[395] L C West. Picosecond integrated optical logic. *IEEE Computer*, 20(12):34–46, December 1987.

[396] N H E Weste and K Eshraghian. *Principles of CMOS VLSI Design - A Systems Perspective.* Addison-Wesley, 1985.

[397] C Whitby-Strevens. The transputer. In *Proc. 12th Annual International Symposium on Computer Architecture*, pages 292–300, 1985.

[398] B Widrow and M E Hoff. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part IV*, pages 96–104. IRE, New York, August 1960.

[399] N B Wilding, A S Trew, K A Hawick, and G S Pawley. Scientific modelling with massively parallel SIMD computers. *Proceedings of the IEEE*, 79(4):574–585, April 1991.

[400] P H Winston. *Artificial Intelligence.* Addison-Wesley, third edition, 1992.

[401] L D Wittie. Communication structures for large networks of microcomputers. *IEEE Transactions on Computers*, C-30(4):264–273, April 1981.

[402] S Wolfram. *Theory and Applications of Cellular Automata (including selected papers 1983-1986)*, volume 1 of *Advanced Series on Complex Systems*. World Scientific, 1986.

[403] Y Wong and J-M Delosme. Transformation of broadcasts into propagations in systolic algorithms. *Journal of Parallel and Distributed Computing*, 14(2):121–145, February 1992.

[404] J Woo and S Sahni. Hypercube computing: Connected components. *The Journal of Supercomputing*, 3(3):209–234, September 1989.

[405] D Wood. A survey of algorithms and architecture for connected speech recognition. In R De Mori and C Y Suen, editors, *New Systems and Architectures for Automatic Speech Recognition and Synthesis, NATO ASI Series Vol. F16*, pages 233–248. Springer-Verlag, 1985.

[406] J R Woodwark. A multiprocessor architecture for viewing solid models. *Displays*, 5(2):97–103, April 1984.

[407] Xilinx Inc. *The Programmable Gate Array Data Book*. Xilinx Inc., 1991.

[408] A C-C Yao. Some complexity questions related to distributive computing. In *Proc. 11th Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

[409] A C-C Yao. The entropic limitations on VLSI computations. In *Proc. 13th Annual ACM Symposium on Theory of Computing*, pages 308–311, 1981.

[410] F Zhao and S L Johnsson. The parallel multipole method on the Connection Machine. *SIAM Journal on Scientific and Statistical Computing*, 12(6):1420–1437, November 1991.