Parallel Computer Architectures: Where Next?

R.N. Ibbett
Edinburgh Parallel Computing Centre & Department of Computer Science,
University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, UK

Abstract. A historical perspective is presented on the development of parallel architectures through the exploration of a geneological taxonomy. Early Single Instruction Single Data Serial Execution system are seen to have evolved into Single Instruction Single Data Parallel Execution (SISDPE) systems and Single Instruction Multiple Data Serial Execution (SIMDSE) systems. The evolution of SISDPE systems is followed to SIMDPE Vector systems and Multiple SISDPE systems. SIMDSE systems are seen to have evolved into SIMDPE systems and these are now evolving into multiple SIMDPE systems. Finally some comment is presented on the current quest of users and manufacturers, a teraflop system.

1 Origins of Parallelism

Designers of early computers such as the Manchester University/Ferranti Mark 1 (first produced commercially in 1951) were constrained by the available technology (valves and Williams Tube storage, for example, with their inherent problems of heat dissipation and component reliability) to build (logically) small and relatively simple systems and to use serial arithmetic. Even so, the introduction of index registers and fast hardware multiplication in the Mark 1 were significant steps in the direction of cost-effective hardware enhancement of the basic design. At Manchester this trend was developed further in the Mercury computer, with the introduction, for the first time, of hardware to carry out floating-point addition and multiplication. This increased logical complexity was made possible by the use of semiconductor diodes and the availability of smaller and more reliable valves than those used in the Mark 1, both of which helped to reduce power consumption (and hence heat dissipation) and increase overall reliability.

The limitations on computer design imposed by the problems of heat dissipation and component reliability were eased dramatically in the late 1950s and early 1960s by the commercial availability of transistors. Transistors not only offered increases in the speed of logic circuits but also enabled designers to use many more of them. Thus the first generation of 'supercomputers' appeared with machines such as Atlas [1], which operated with parallel rather than serial arithmetic and had separate index and floating-point arithmetic units, and the CDC 6600 [2], which had ten separate parallel arithmetic/logic function units capable of concurrent operation.

This was parallelism below the instruction set level, however, and thus invisible to the programmer. Other groups were developing other forms of parallelism using a multiplicity of serial arithmetic units to produce quite different architectures *i.e.* array processors. As early as 1958 Unger published a paper entitled "A Computer Oriented Towards Spatial Problems" [3], from which the first array processor, SOLOMON, was developed [4]. The

SOLOMON design consisted of a two-dimensional array of 32×32 processing elements (PEs), each of which had 128 32-bit words of store and a bit-serial arithmetic unit. All PEs acted in unison, under the control of a single stream of broadcast instructions.

Further advances in technology led to the production of integrated circuits and it became possible to build systems of ever greater complexity, without increasing the component count, and without decreasing reliability or increasing the power requirements. The ideas from SOLOMON, for example, were carried forward into several important high-performance architectures including the ILLIAC IV [5], the Burroughs Scientific Processor [6], the Goodyear Aerospace MPP [7], and STARAN [8], and the ICL Distributed Array Processor (DAP) [9].

Today parallelism comes in essentially two varieties, *data parallelism* and *code parallelism*, and most available systems can be divided correspondingly into Flynn's SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data) systems [10]. SIMD systems such as the DAP are relatively easy to program, since the code is sequential, and what is needed are language extensions (typically to FORTRAN) to allow software data structures to be mapped on to the data parallelism in the hardware. Naturally the usefulness of these systems depends strongly on there being a significant amount of data parallelism in the application, and although they represent a fairly mature technology, their use has until recently been largely confined to niche areas.

MIMD systems are taking longer to mature since they offer an additional dimension of complexity *i.e.* multiple, concurrent threads of code execution, and this calls for a great deal more human ingenuity to make effective use of such systems. The interconnection problem is also more complex, since whereas SIMD systems depend for their performance on regularity in the data and consequently require, in most cases, only simple geometric communication patterns, MIMD systems frequently involve either full or random communication patterns which cannot easily be provided in hardware.

Clearly, if parallelism is a good thing, then the more of it we have the better, and the term "massively parallel" frequently appears in manufacturers' literature. However, the answer to the question "What constitutes a massively parallel system?" may be tens (or possibly hundreds) of thousands of processors in SIMD systems, but only hundreds (or possibly thousands) of processors in MIMD systems. To explore parallel systems further we need a taxonomy.

2 A Taxonomy

The taxonomy of parallel systems has been an area of academic endeavour for many years. Numerous authors have attempted to classify computers on the basis of the type of concurrency they exhibit. Flynn's taxonomy of Single Instruction Single Data (SISD) machines, Single Instruction Multiple Data (SIMD) machines (subsequently divided into two main classes: vector processors and array processors), and Multiple Instruction Multiple Data (MIMD) machines (virtually all of which are in fact Multiple Single Instruction Single Data (M[SISD]) systems), has been the subject of a host of refinements. The problem is to divide up a multi-dimensional design space in a meaningful way. Here we need to consider the number of instruction streams (single or multiple (*i.e.* code parallelism)), the number of data streams (single or multiple (*i.e.* data parallelism)), the type of arithmetic (serial or parallel execution), and the organisation of addresses (into single or multiple address spaces).

Figure 1 shows this taxonomy as a geneology chart. As observed in section 1, the earliest machines were Single Instruction, Single Data, Serial Arithmetic Execution (SISDSE) sys-

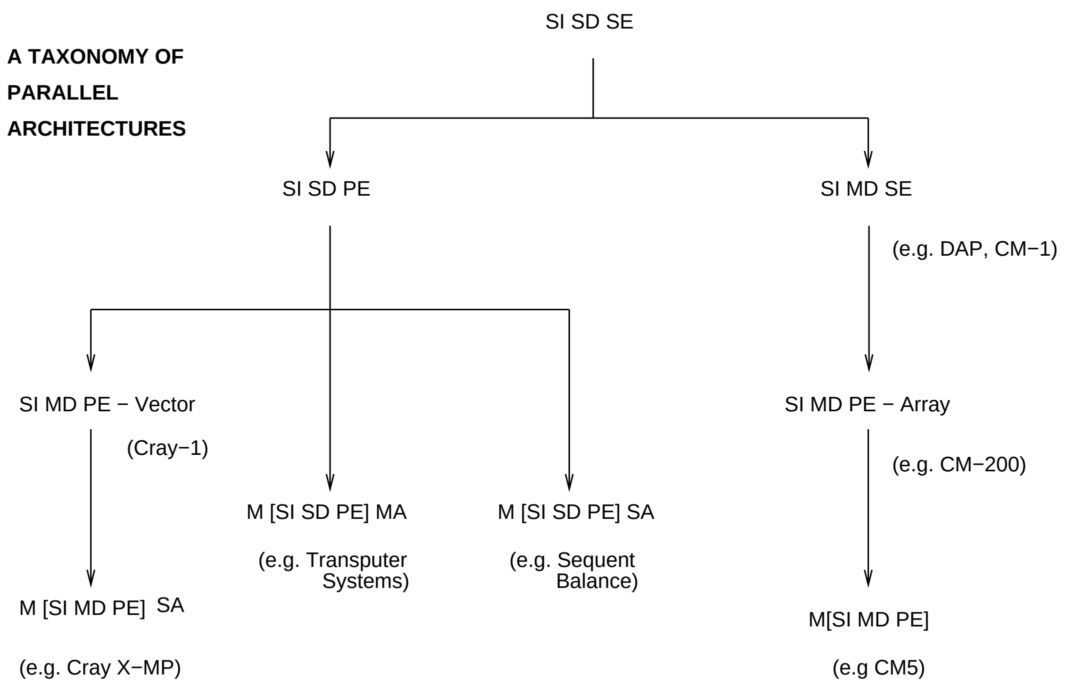

Figure 1: A Taxonomy of Parallel Architectures

tems, and these evolved down two paths, to Single Instruction, Single Data, Parallel Arithmetic Execution (SISDPE) systems such as the Atlas and CDC 6600, and Single Instruction, Multiple Data, Serial Arithmetic Execution (SIMDSE) systems such as SOLOMON.

SISDPE systems have evolved in several directions; into SIMDPE Vector machines, of which the Cray-1 is the earliest well-known example, Multiple SISDPE systems based on processors such as the Transputer, each of which has its own address space and communicates with other processors via message passing, and Multiple SISDPE systems based on "conventional" processors which share a common memory/address space. SIMDPE systems have themselves evolved into shared memory multiprocessor systems such as the Cray X-MP, here designated M[SIMDPE]SA systems.

SIMDSE systems have, as VLSI arithmetic units have become available, evolved into SIMDPE systems, examples of which are the CM-200 and the DAP/CP8. This line of evolution has progressed further with the CM5, which is a multiple SIMDPE system.

We now follow the evolution down the various paths in more detail.

3 SI MD PE Vector Systems

The need to provide facilities for processing sequences of vector elements was recognised in the very early days of digital computer design. The original von Neumann concept included the notion of allowing instructions to be treated as data, which meant that the address part of an instruction accessing a vector element, for example, could be incremented during the execution of a program loop and thus produce the effect of processing a vector. In practice, however, this technique allows so much scope for program error that the use of an index register was introduced as early as in the Manchester Mark 1, and this technique has been used almost universally ever since. Thus virtually any digital computer can be used to process vectors. The differences between machines lie in the addressing facilities which they provide to support accesses to data structures, and whether or not they include instructions which process a sequence of vector elements. There are essentially two types of vector machine, systems such as the Cray-1, in which vector instructions are register-to-register operations and systems such as the CDC Cyber 205, in which vector instructions are store-to-store operations. In both cases vector orders are provided as a means of improving performance.

The Cray-1 [11] is the logical successor to the CDC 7600, in which vector operations were introduced as a means of overcoming hardware bottlenecks in the 7600. The floating-point addition and multiplication units in the 7600 had a combined execution capacity of 1.5 FLOPS/CLOCK (floating-point operations per clock). However, instructions could only be issued at a rate of one per clock. Furthermore, results could only be returned to the floating-point operand registers (X registers) at a rate of one per clock. Thus the maximum floating-point execution rate was restricted by both of these constraints to 1 FLOP/CLOCK.

In the Cray-1 the vector orders cause streams of up to 64 data elements to be processed as a result of one instruction issue. Vectors are contained in a set of eight V registers (figure 2), each capable of holding 64 elements (each of 64 bits), and a typical vector instruction causes sets of operands to be taken from two V registers and the results to be returned to a third. In the following instruction sequence

$$\begin{aligned} V0 &\leftarrow V1 + V2 \\ V3 &\leftarrow V4 * V5 \end{aligned}$$

the second instruction uses different registers and a different functional unit from the first and can be issued one clock period after the first instruction. Subsequent to the pipeline start-up delays in the functional units (each of which can carry out operations at a rate of one per clock period), a floating-point result will appear from both the adder and the multiplier in each successive clock period. Thus if performance is estimated in terms only of floating-point addition and multiplication, the maximum floating-point execution rate is 2 FLOPS/CLOCK. Furthermore, around 60 other instructions can be issued before these units require further instructions to keep them busy.

The bottleneck on the entry of results into registers is overcome by providing each vector register with its own input multiplexer circuitry for selecting results from among the seven functional units which can produce vector results and, correspondingly, providing each vector functional unit with its own input multiplexers for selecting vector register operands.

Apart from overcoming the bottlenecks identified in the 7600 (themselves examples of the so called “von Neumann” bottleneck), vector processing also delegates to hardware the control of vector loops which in a scalar machine would require the execution of branch instructions (themselves notorious for reducing pipeline performance) and allows prefetching of streams of operands to be carried out in hardware, either for immediate use (as in systems such as the Cyber 205) or for filling registers (as in Cray systems). Both of these techniques help to improve performance.

3.1 M[SI MD PE] Systems

The first Cray-1 was delivered in 1976. In mid 1979 work started on the design of a system which would have an essentially identical instruction set to the Cray-1 but considerably more power. The possibility of obtaining this power by adding more vector units was considered, but although improved vector performance was obviously important, the effects of Amdahl’s Law dictated that the need for improved scalar performance was actually more important. A deliberate decision was therefore made to pursue a multiprocessor design rather than one involving multiple vector units. The prototype two-processor version of

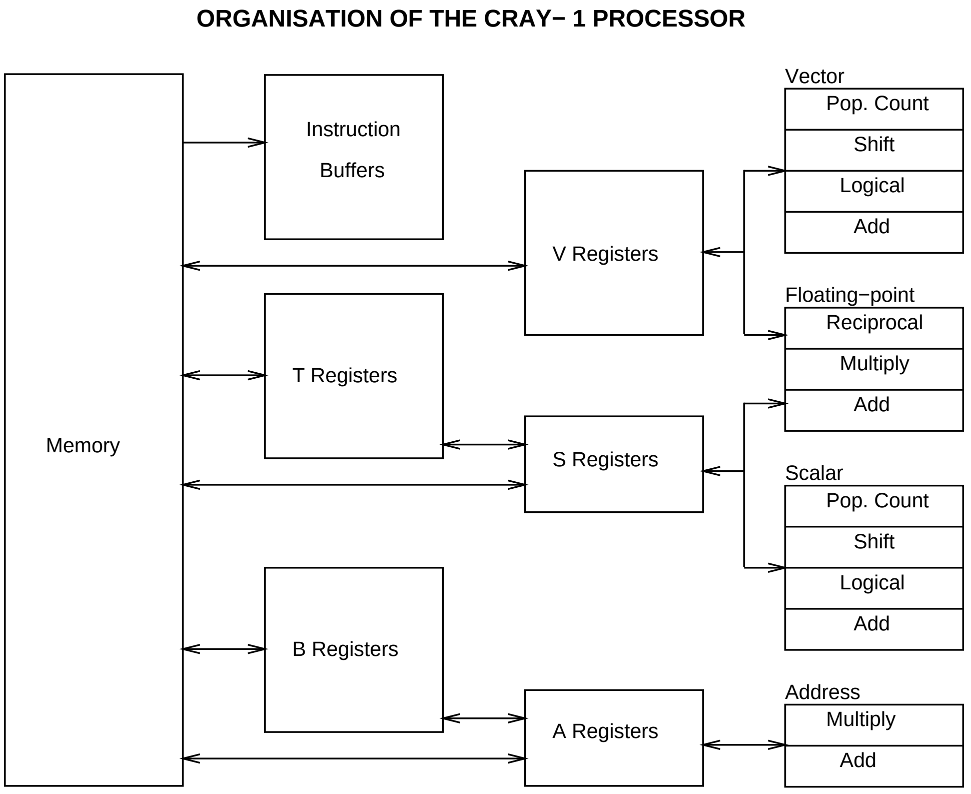

Figure 2: Cray-1 Processor Organisation

this system, the Cray X-MP/2, became operational in 1982. In 1988 Cray introduced the Y-MP. Architecturally the Y-MP is upwards compatible with the X-MP but has a 32-bit addressing capability, uses newer technology and offers greater performance. The first model in the range, the Y-MP/832, contained eight processors with a 6 ns clock, 32 Mwords of high speed central memory (in 256 banks) and a slower solid state memory containing 128, 256 or 512 Mwords.

All these systems have a common memory to which all the processors have multiport access. In the Cray-1 a single port is used for all data transfers between the vector registers and memory. Thus only one vector can be read or written at a time (at a rate of one per clock period). However, to sustain maximum floating-point performance the units require four input and produce two output operands per clock. This is not a problem in matrix multiplication, for example, where the output of the multiplication unit becomes an input to the adder and for full performance only two input operands per clock are required. One of these is a row of one matrix, held in a vector register and it is only necessary to fetch one operand (the columns of the second matrix) from memory at a time to sustain maximum performance. However, for some other algorithms, the Cray-1 is memory bandwidth limited.

The use of multiport memories in the X-MP and its successors overcomes this limitation. In the X-MP/4, for example (figure 3), the memory is organised as four sections each containing 16 banks, and each processor has an independent access path into each of the four sections. CPU Ports A and B are used by vector read instructions and Port C is used by vector store instructions and by scalar instructions. The CPU Ports contain the logic which generates sequential memory addresses for vector accessing. Memory addressing is arranged such that successive elements of a vector of stride 1 are accessed from successive sections of the memory, and within each section from successive banks. The bank cycle time is four clocks, but within each section each bank is connected via a separate data path to each port and the data path is only required in the last clock of each cycle. Thus the

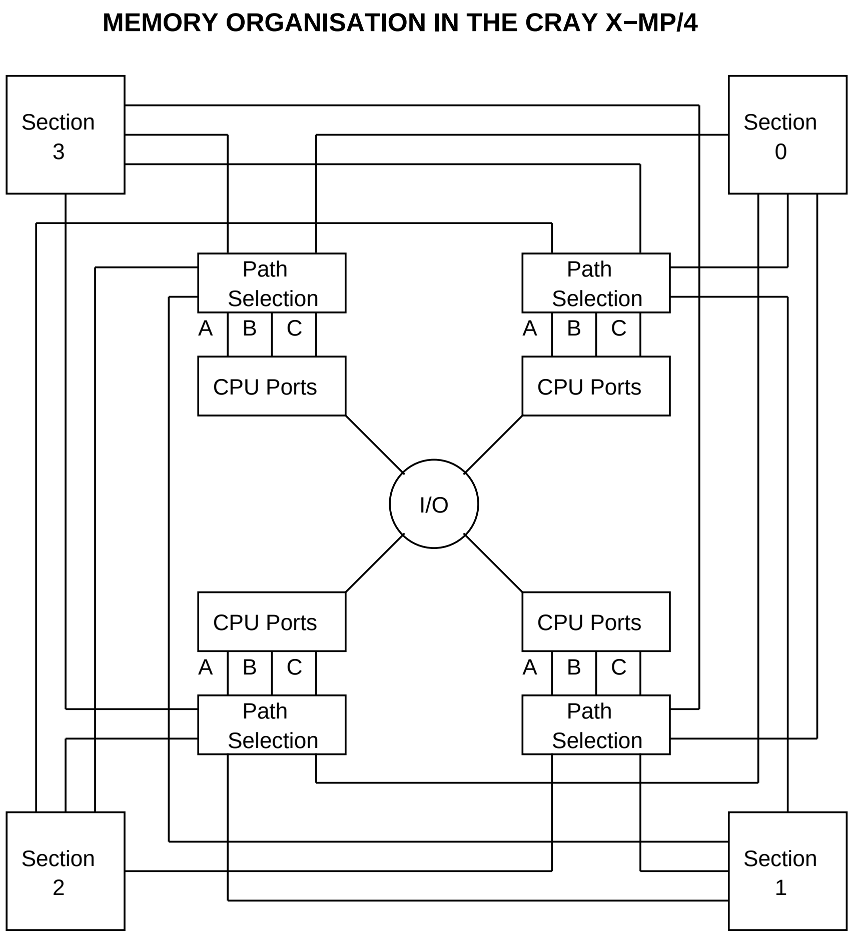

Figure 3: Memory Organisation in the CRAY X-MP/4

memory can sustain a transfer rate of one word per clock via each port.

Shared memory systems such as the Cray X-MP and Y-MP series have been successful as supercomputers primarily because each processor is so powerful; the scope for parallelism is strictly limited, simply because all the memories are shared by all the processors and there is a limit to the number of ports which can be built into a multi-port memory system. Indeed the 16-processor Y-MP C90 is probably the end of the line so far as this type of system is concerned. Cray's other line of development, the Cray-2 and beyond, has ground to a halt because of production difficulties with the Cray-3. Systems built using a multiplicity of SISD systems with simple memory interfaces offer greater scope for parallelism, using either shared memory or distributed memory with message passing.

4 M[SI SD PE]MA Systems

Message passing systems offer considerable scope for parallelism. Here the problems have primarily been in devising interconnection schemes which are bounded in time and space by something less than the square of the number of processors (*c.f.* the cross-bar switch in the 16-processor C.mmp [12], one of the very earliest multiprocessor systems. Hypercubes and butterfly networks have found favour in the USA; in Europe, where most parallel systems are based on the Transputer, more *ad hoc* solutions involving a multiplicity of small configurable cross-bar switches have been used. These allow the user to change the topology to suit the application. If the problem seems to require a tree of processors, a square or even a hypercube, the switches can be configured accordingly. These switching

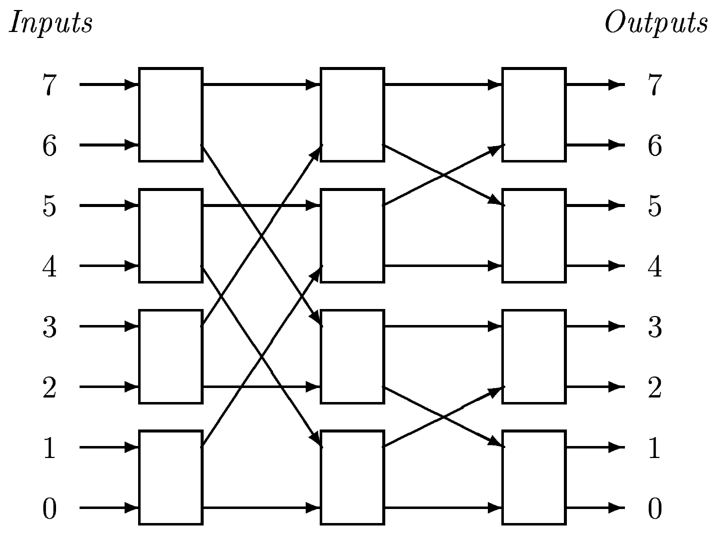

Figure 4: An 8-way 3 layer multi stage network using 2 × 2 cross-bar switches

mechanisms were developed as extensions to the limited functionality of the point-to-point communications protocols implemented in the Transputer. The development of efficient routing protocols, first in software [13], and hopefully soon in hardware in the T9000, is now obviating the need for switch configurability.

5 M[SI SD PE]SA Systems

Despite the current popularity of distributed memory message passing systems, it has become increasingly apparent that the relative ease of programming shared memory systems is a significant factor in their favour. As one moves from fully shared memory sytems (such as Cray machines) through hierarchical shared memory systems (such as the Sequent) to distributed shared memory systems (such as the Butterfly) and on to distributed memory systems with message passing, the problems of writing efficient programs and/or support environments increase. And as observed by Sietz in 1985 [14], although message passing machines are more economical, being more simple to design, they are not as versatile as tightly coupled shared memory machines. The latter can simulate the former, but it is hard for the former to simulate the latter. Indeed, such are the advantages of shared memory systems, which avoid the problems of process to processor mapping, data distribution, access time variability, etc., inherent in distributed memory systems, that considerable research effort is being put into ways of supporting the concept of virtual shared memory on distributed memory systems in such a way that the programmer is relieved of worrying about these problems.

Of course the easiest way is to use real shared memory, but then one is faced with the problem that in shared memory systems the interconnection problem becomes increasingly severe as the degree of parallelism increases. Networks such as the hypercube, in which messages are routed through the network from source to destination in a series of hops, are not appropriate for shared memory systems, since each processor requires direct access to all of the available memory, itself typically made up of a number of discrete units. The two most obvious techniques to use in this situation are a common bus or a cross-bar switch.

The shortcomings of the common bus are self evident; the number of processors which can be connected to a bus is restricted not only by the physical and electrical properties of the bus, but also by the finite bandwidth of the bus which constrains the data transfer capacity between the processors and memory. This latter problem can be ameliorated by the use of cache memories within each processor, though this then introduces the further problem of cache coherency [15]. Systems with a few tens of processors are nevertheless cost-effective, as witnessed by the success of commercial machines such as the Sequent Balance and the Encore Multimax [16]. More recently, the ability to accommodate potentially hundreds or possibly thousands of processing elements has been demonstrated in the KSR1 which uses distributed caches linked by ring technology, the overall effect being to give the illusion of fully shared memory at the hardware level.

The simplest way to implement full connectivity directly between m source units and n destination units is to use a cross-bar switch. The cross-bar switch is capable of realising any one-to-one, or one-to-many, set of connections. However, the hardware cost is proportional to $m.n$, and as m is normally similar in magnitude to n this equates to approximately n^2. This makes such interconnection structures impractical for highly parallel systems, where n and m are typically in the range 2^8 to 2^{16}.

However, an $N \times N$ cross-bar switch can be reduced to two $N/2 \times N/2$ cross-bar switches and two N-input exchange switches using a method devised by Beneš [17]. The resulting $N/2 \times N/2$ cross-bar switches can be similarly reduced, and through this recursive trade-off between complexity and network latency, a full connectivity network can be produced at a significantly lower cost than a full cross-bar switch. The network shown in figure 4, for example, is constructed entirely from 2-input 2-output switch-nodes, arranged in layers and suitably connected.

A crossbar based network may either be circuit switched or packet switched. Most interconnection networks used in shared memory multiprocessor systems have been circuit switched. The network used in the BBN Butterfly, for example [18], although described as being packet switched, is in fact circuit switched. When a processor makes a non-local memory request a circuit to memory is opened through the switch and held open until the request has been satisfied. The path through the switch is, in effect, an extension of the processor bus. At Edinburgh we have designed and fabricated a prototype VLSI device (Xbar [19]), intended for use in genuinely packet switched multistage interconnection networks.

The network shown in figure 4 provides a single bit interconnection path between each source and destination using 2*2 crossbar switches. By increasing the number of ways m for each crossbar-switch, both the number of layers and the number of devices per layer are reduced for the same number of inputs and outputs.

To transfer a w-bit packet through the network the plane must be duplicated w times thus giving low latency and high bandwidth but also increased cost, or the packet must be transferred serially through a single plane thus minimising cost but at reduced bandwidth and increased latency. Alternatively some intermediate solution may be adopted in which the w-bit packet is transferred across a D-bit wide network as a sequence of k w/D-bit parcels.

For efficient implementation in silicon an individual IC must take a data width d for a given pin count p and number of ways m. To provide support for a D-bit wide parcel the network must then be made up of D/d bit-slices. Xbar is based on a protocol optimised for bit-sliced multi-layer networks; it contains a 2-bit slice of a 4*4 cross bar, one slice of which is shown in figure 5. It uses on-chip queues to handle the situation where two input packets destined for the same output port arrive simultaneously, together with a handshaking mechanism which prevents queue overflow under extreme conditions. Each output port has four 32-entry queues, one per input port. Packets are taken from the queues according to a longest queue arbitration mechanism; in the case of equally filled queues, the mechanism degrades to a round robin arbitration scheme.

We are currently building a test-bed demonstration system to show that networks formed from Xbar devices will not only be suitable for replacing bus-based networks in multiprocessor systems but will also offer increased bandwidth and increased interconnectivity. We are also investigating the possibility of fabricating much larger Xbar devices to reduce latency in massively parallel systems.

6 SI MD SE Array Systems

Advances in VLSI technology have had a considerable impact on the design of SIMD array processors. The original DAP, for example, produced by ICL, was implemented as part of a large mainframe and occupied several cubic feet, whereas the more recent AMT DAP implemented in VLSI comes in the form of a plug-in accelerator for a deskside workstation. The availability of high-density gate-arrays and full-custom VLSI as a means of realising a particular implementation contributed towards the construction of the first Connection Machine [20] by Thinking Machines Corporation in 1985.

All the commercially available machines in this class (*e.g.* the AMT DAP, the MasPar MP-1 and the Connection Machines) have quite similar architectures. The main differences between them are in the interconnection mechanisms which link the individual processors.

6.1 The DAP

The DAP architecture is shown in figure 6. The processing elements (PEs) are arranged in a square 32×32 or 64×64 array. Each PE comprises a single-bit processor, together with a local memory which can range in size from 32 Kbits to 1 Mbits, and input connections from its four nearest neighbour processors in the North, East, West and South directions. The boundary connections at the perimeter of the array are determined by bits in the instruction. Either the boundary inputs are set to zero and boundary outputs are discarded, or else the boundary inputs are taken from the boundary outputs within the same dimension. Hence, East may be connected to West and North may be connected to South. This allows the array to be configured as an array, a vertical or horizontal cylinder, or a torus.

Program control is carried out by the Master Control Unit, which takes instructions from the Code Memory, interprets them and controls the PEs, the memory and data transfers.

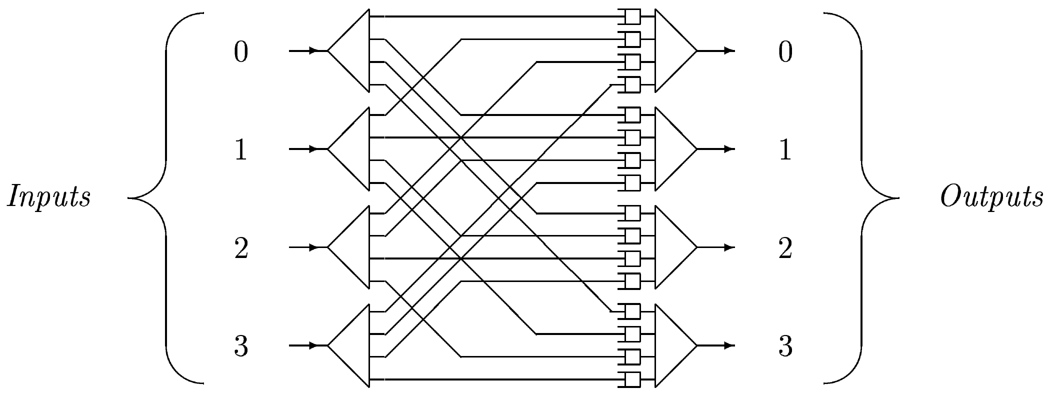

Figure 5: Organisation of Xbar

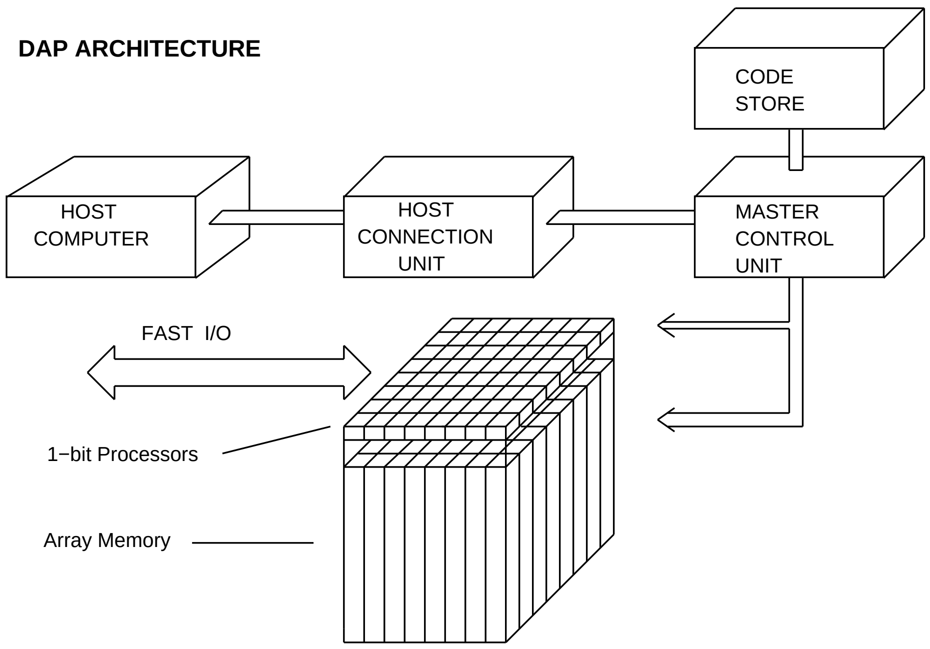

Figure 6: DAP architecture

Access to the array is via row and column data highways.

Interaction between the DAP and the host is controlled by the Host Connection Unit. Data can also be transferred to and from the memory via the D plane over a fast data channel (70 Mbytes/s). This is particularly useful for the attachment of high-speed graphics displays.

PE architecture

A simplified view of the internal architecture of a single PE is shown in figure 7. Each processing element consists of a single-bit adder, an input multiplexer, an output multiplexer and an $n \times 1$-bit store. The ALU consists of three one-bit registers, the accumulator Q, the carry register C and an *activity* bit A. The activity bit is used for local enabling or disabling of certain actions within the PEs, thus permitting a subset of the array to take part in whatever computation is in progress.

The input multiplexer selects data either from the output of one of the four nearest neighbours or from the local memory, depending on the instruction being executed. The output multiplexer selects which source of information (the output from the local adder or the row or column highway) is used when writing to the local memory.

The single-bit adder performs full addition of the accumulator and the selected input, with an optional carry input. The selected input may be complemented before addition, enabling subtraction and logical inversion operations to be implemented. The carry-in to the single-bit adder may come from the local carry register or the carry-out of the Eastern neighbour, depending on the operating mode of the array. This choice permits the DAP to perform word-arithmetic in two quite distinct ways, either *bit-serial/word-parallel* or *bit-parallel/word-serial.*

6.2 The MasPar MP-1

The basic architecture of the MasPar MP-1 family of SIMD processors [21] is similar to that of the DAP, but interprocessor communication is handled differently, by two separate

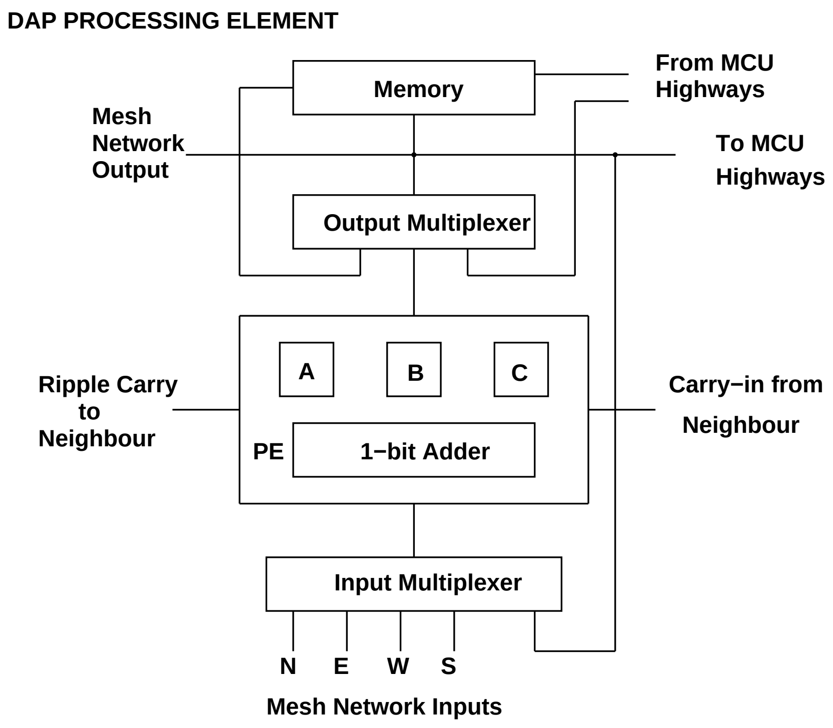

Figure 7: DAP processing element architecture

mechanisms. The choice of which mechanism is more appropriate for a given application is determined by the regularity of the data transfer. For situations in which an entire array of data is to be moved across the PE lattice then the X-Net communications mesh is used.

Conceptually, the X-Net mesh is a one-bit wide communications network which links each processor with its eight nearest neighbours. In fact each PE has only four interconnects, located at its diagonal corners, thus forming an X-shaped grid at 45 degrees to the PE lattice, rather than the square grid of the DAP. However, the X-Net mesh is more sophisticated than the DAP mechanism, since the diagonal links do not simply cross over each other, but are connected at nodes which allow communications to be routed to any of the PE's eight nearest neighbours (figure 8). The direction of the outgoing message through this tri-state node is set by the ACU at the same time as the PEs are instructed to transmit, so that there is no latency in the connection and hence the communications of a bit between neighbouring processors takes one clock cycle. At the edges of the PE array the interconnects are wrapped around to form a torus, though the user may select planar boundaries, in which case any differences between the topologies are handled by software.

Random communications between arbitrary processors are possible via a three-stage global router which emulates a crossbar switch. Each PE cluster has a connection to the router stage of the switch and another to the target stage. These ports are shared by all PEs within the cluster. The router and target units of the global switch are connected by an intermediate stage. The address of the target PE is calculated by the originating processor and, if all the links through the router between the start PE cluster and the finishing one are free, then connection is established. Clearly, this may necessitate some PEs waiting, perhaps for some time, for others within the same cluster to finish their data transfer. Once set, the link is bi-directional and on closing the target PE sends an acknowledgement. Data transfers through the links are bit-serial and clocked synchronously with the PE clock. Since the router ports are multiplexed for each PE cluster arbitrary communications patterns require a minimum of 16 router cycles to complete.

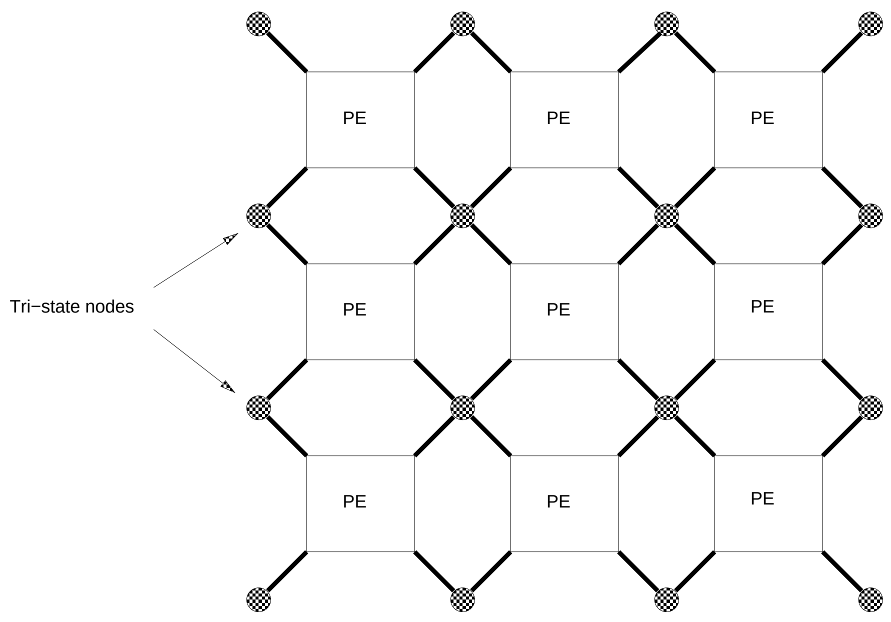

Figure 8: The MP-1 X-Net

6.3 The Connection Machine

The system level architecture of the CM-1 is shown in figure 9, in which the similarity with the DAP (and most other SIMD array processors) is clearly visible. The array of processing elements, comprising a simple boolean processor and some local memory, is seen by the host machine simply as an extended region of memory. The host computer directs the connection machine to implement parallel portions of code, and in this respect it differs from the DAP or MP-1 which have an instruction processor built into the array unit. The CM-1 host broadcasts a sequence of instructions to the array micro-controller, which interprets the instructions and broadcasts an appropriate sequence of micro-instructions to the array of PEs, for each received host instruction.

An important feature of a Connection Machine is its support for programmable links between PEs. In the DAP, when one processor communicates with its Northern neighbour all processors must communicate with their Northern neighbour, or not at all. This is because the DAP has a static square-mesh communication network, which only supports eight routing functions. Communication in the CM-1 is significantly more powerful than this, since a group of sixteen processing elements shares a link into a packet-switched binary 12-cube network, as well as having individual connections to a DAP-like NEWS grid. Essentially this means that all PEs can compute the address of a PE to which they want to send a message, and then use the 12-cube network to route the message in logarithmic time.

6.4 SI MD PE Systems

Although the CM-1 was aimed at the AI/symbolic processing market, TMC soon perceived the need to address the computationally intensive scientific market, and the CM-200 series machines have a floating-point capability based on the use of Weitek 3132 chips. The boolean processing elements are implemented as VLSI devices, each of which contains 16 PEs, and one floating-point unit is provided for each pair of processor chips (*i.e.* one per 32 processors). A memory interface unit is incorporated to carry out the necessary transpositions between data stored in serial form in the individual processor memories to

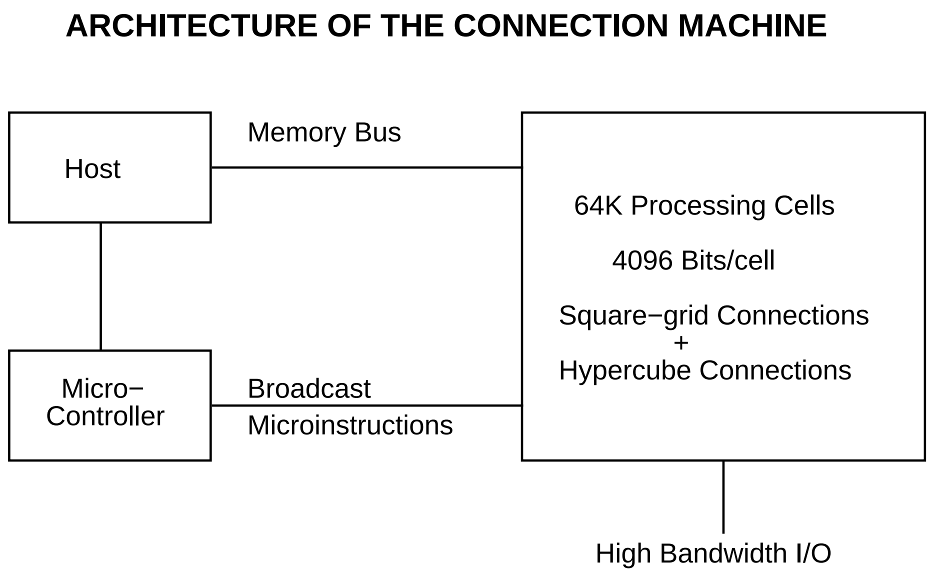

Figure 9: Architecture of the Connection Machine

the parallel form needed for the Weitek chip, but most floating-point users never use the 1-bit processors, so their data is stored in parallel form anyway.

This change represents an evolution from SIMDSE systems to SIMDPE systems. A similar evolution is seen too in the AMT DAP/CP8 range (figure 10) which includes an 8-bit coprocessor with each PE. Here the system is somewhat different, since there are as many 8-bit coprocessors as there are 1-bit processors. Each of the 8-bit coprocessors has 256 memory bits of its own, and data is exchanged with data in the main array through store-to-store transfers.

6.5 M[SI MD PE] Systems

One of the limitations of the SIMD model is the limited degree of control over the activity of the individual processsing elements: at each instruction a PE may be either active or inactive. To deal with boundary conditions in finite element programs, for example, parts of the code have essentially to be run twice; once to deal with the boundary elements and once to deal with the interior elements. What is needed is the ability to execute conditional branches at each node, as in an MIMD system running in Single Program Multiple Data mode (*i.e.* with all PEs running the same code, but taking different paths through it).

Until recently the complexity of PEs required to construct MIMD systems has precluded their use in massively (data) parallel systems. However, the complexity of modern VLSI circuits has allowed this possibility to be realised. The Thinking Machines CM5 system [22], for example, can be configured with hundreds or even thousands of parallel processing nodes, each with its own memory and capable of independent execution. These nodes can be operated in SIMD mode, all fetching instructions and operands from the same address in their own memories, or in MIMD mode, in which case they can operate independently and access individually computed addresses. Thinking Machines use the term *universal architecture* to describe this capability.

Groups of processing nodes are supervised by a control processor (running an enhanced Unix) which broadcasts blocks of instructions to the nodes and then initiates execution. When the nodes are operating in SIMD mode the processing nodes remain in synchronism and blocks are broadcast as needed. When operating in MIMD mode, nodes can take different branches, which may involve different execution times, and synchronisation is

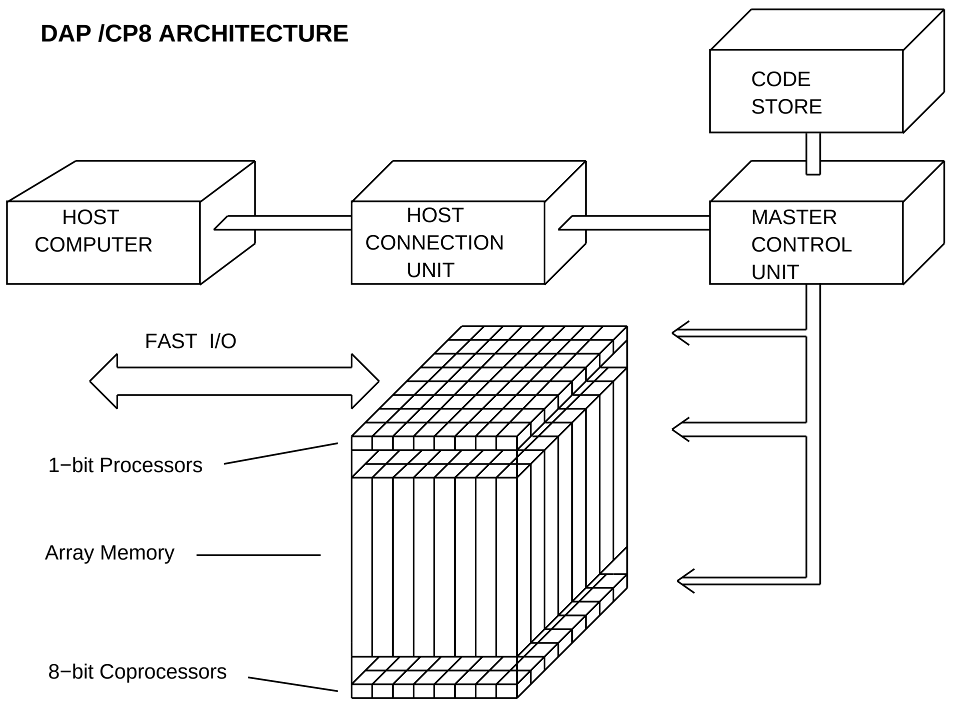

Figure 10: DAP/CP8 architecture

then effected under program control, as and when required by the algorithm.

The nodes are interconnected, and linked to the control processors, by both a Data Network and a Control Network. Each control processor is essentially a standard workstation to which CM5 network interfaces have been added. Also connected to the networks are I/O units supporting mass storage, graphic displays and VME and HIPPI peripherals.

Each processing node consists of a SPARC microprocessor (with cache memory) connected by a 64-bit bus to a network interface (itself connected to the Data and Control Networks) and a memory subsystem. The memory subsystem consists of a controller and from one to four 8 Mbyte DRAM memory units. As an option, a node may contain an 'arithmetic accelerator'. In this configuration (figure 11) the microprocessor acts as a scalar unit and is connected to four vector units. Thus not only can the nodes act in concert in SIMD Array mode, but the nodes themselves can operate in SIMD Vector mode. Each vector unit (figure 12) contains a pipelined floating-point arithmetic unit, a 64×64-bit register file and a memory controller linked via a dedicated path to its own 8 Mbyte memory unit.

The vector units operate under the control of the scalar unit via a memory-mapped control register interface. When a scalar unit read or write operation placed on the node bus addresses a vector unit, the high order bits of the address indicate the operation type: memory transaction, data register access, control register access or vector unit instruction. A vector instruction may be issued to a specific vector unit, to a pair of units, or to all four units simultaneously.

Vector instructions are essentially Cray-like with the register file being treated as a set of 32 or 64-bit vector registers. In 64-bit format the registers can be treated as sets of 16 4-element vector registers, eight 8-element registers or four 16-element registers, while in 32-bit format they can be treated either as 16 8-element registers or eight 16-element registers. In any of these configurations the number of elements per vector register is obviously smaller than in a Cray, but since four vector units can operate concurrently on the same instruction, the total number can be the same.

The time required to issue a vector instruction is also longer than in a Cray, but is

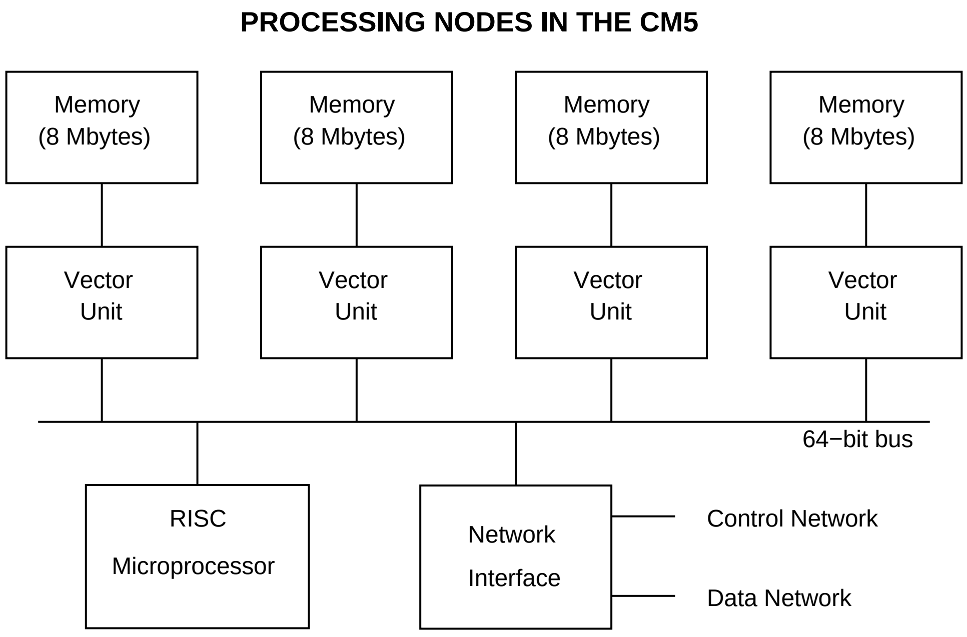

Figure 11: Processing Nodes in the CM5

typically less than the time required to execute a four-element CM5 vector instruction Furthermore, two successive vector instructions can be processed contiguously in the pipeline, thus avoiding any gaps and circumventing problems of pipeline latency on start-up and run-down. Thus in principle the combined peak computational power of the vector units in a node (128M 64-bit floating-point operations per second) is sustainable.

7 The Teraflop Quest

In the mid 1980s the holy grail for computational scientists was a machine with a performance of 1 Gflop/s [23]. By the late 1980s this had been achieved using a limited degree of (multiprocessor) parallelism (*e.g.* in the Cray-2). It is almost possible now to achieve 1 Gflop/s in a single processor, since clock periods as short as 1 ns have been achieved, at least in prototypes. However, the goal now is a machine with a performance in the region of 1 teraflop/s (*i.e.* 10^{12} flop/s). Technology alone, in the form of further significant reductions in clock periods, will not produce performance in this range within the foreseeable future, so the use of massive parallelism is the only option. But can a teraflops machine be built? Computational scientists are hopeful, as witnessed by the European Teraflops Initiative the American High Performance Computing and Communication Program, and the manufacturers are of course bullish. Cray, for example, plan to deliver their first MPP (Massively Parallel Processing) system with 100 Gflop/s performance in 1993, and to deliver a teraflop/s system in 1997.

Why should it not be possible? The history of architectural developments contains many examples of systems built to overcome the bottlenecks in previous systems. Once in use, further bottlenecks have been discovered, and new ideas and/or technologies have been used to overcome these. At one time it was fashionable to cite the “von Neumann” bottleneck as the cause of all the problems, and alternative architectures such as dataflow were proposed to overcome it. These architectures have come and gone.

So where are the bottlenecks now? The hardware of almost every new system has offered far higher computational power than the software has been able to exploit. Considerable effort has always been required to develop software capable of harnessing the hardare power

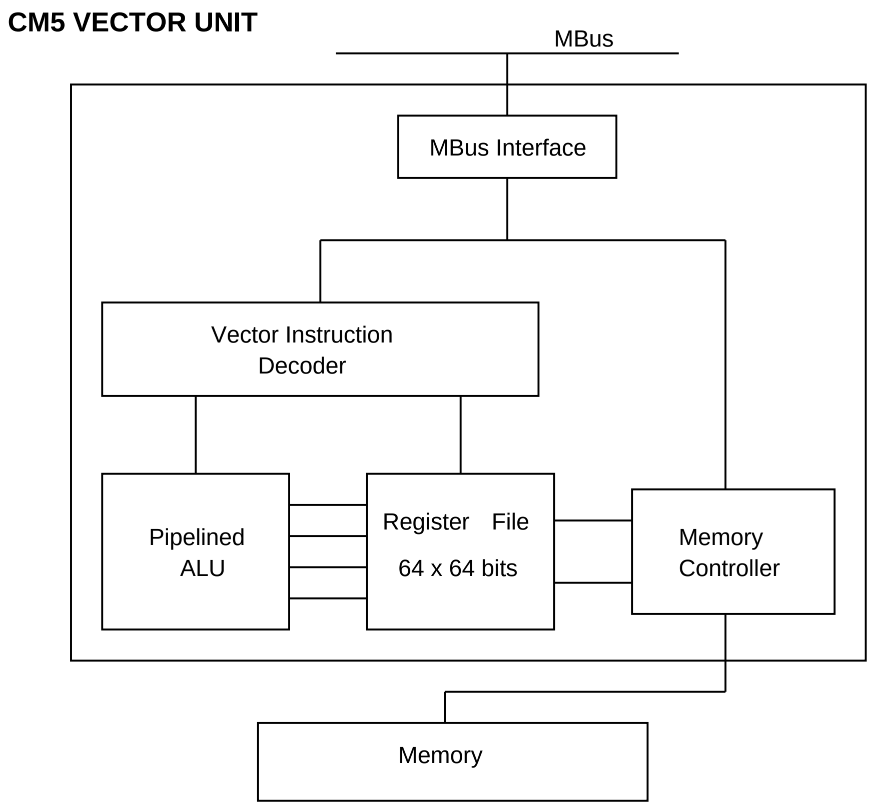

Figure 12: CM5 Vector Unit Architecture

on offer. Amdahl's Law has been seen at work in vector processing and parallel processing alike, and it is a problem which will not go away. There are inevitably parts of any program which cannot be vectorised or parallellised.

In parallel processing there is the further problem of the cost of communication. This involves two components; the time to deliver a message and the time spent by one processor waiting for another. Unless the latter cost can be effectively reduced to zero, then above a certain number of processors in a system, performance will decrease rather than increase as more processors are added. This is generally more of a problem in MIMD systems than in SIMD systems. The statement by Thinking Machines Corporation about the CM5 that "when operating in MIMD mode synchronisation is effected under program control" leads one to suppose that this synchronisation is likely to be the next bottleneck, and that techniques for combining both synchronisation signals and arithmetic results as they pass from a multiplicity of processors to a single point of control and/or accumulation will be the next requirement.

But the real problem in producing a teraflop machine is likely to be power dissipation. Cray machines have typically required 100 - 150 KWatt of electrical energy, and this amount of heat has had to be dissipated from them. Seymour Cray himself has claimed that the main problem in building supercomputers is power dissipation. In the CM5 ten nodes produce 1Gflop, so to produce 1 teraflop 10,000 nodes would be required. Currently a node requires of the order of 100 - 200 Watts of power. To deliver a teraflop of compute power using existing technology would thus require the supply and dissipation of some 1 - 2 MWatts. Clearly some further technological developments are yet required to reduce the power requirements per floating-point operation to well below their current level. Optimists in the industry believe an order of magnitude reduction is within sight, so a teraflop machine may also be within sight.

References

[1] T. Kilburn, D.B.G. Edwards, M.J. Lanigan, and F.H. Sumner. One-level Storage System. *IRE Transactions*, EC-11:223–234, 1962.

[2] J.E. Thornton. *Design of a Computer: The Control Data 6600.* Scott Foresman & Co, Glenview, Ill, 1970.

[3] S.H. Unger. A computer oriented towards spatial problems. In *Proc. Inst. Radio Eng.*, volume 46, pages 1744–50, 1958.

[4] D.L. Slotnick, W.C. Borck, and R.C. McReynolds. The SOLOMON computer. In *AFIPS Conf. Proc.*, volume 22, pages 97–107, 1962.

[5] G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick, and R.A. Stokes. The ILLIAC IV computer. *IEEE Transactions on Computers*, C-17:746–57, 1968.

[6] D.J. Kuck and R.A. Stokes. The Burroughs Scientific Processor (BSP). *IEEE Transactions on Computers*, C-31(5):363–376, 1982.

[7] K.E. Batcher. Design of a Massively Parallel Processor. *IEEE Transactions on Computers*, C-29:836–840, 1980.

[8] K.E. Batcher. STARAN Parallel Processor System Hardware. In *Proc. AFIPS-NCC*, volume 43, pages 405–410, 1974.

[9] S.F. Reddaway. DAP – a distributed array processor. In *1st Int. Symp. Comp. Architecture*, pages 61–65, 1973.

[10] M.J. Flynn. Some Computer Organisations and their Effectiveness. *IEEE Transactions on Computers*, C-21:948–960, 1972.

[11] R.M. Russell. The CRAY-1 Computer System. *Communications of the ACM*, 21:63–72, 1978.

[12] W.A. Wulf and C.G. Bell. C.mmp - A multi-mini-processor. *Proc. AFIPS Fall Joint Comp. Conf.*, 41:765–777, 1972.

[13] L.J. Clarke and G.V. Wilson. Tiny: An efficient routing harness for the inmos transputer. Technical Report EPCC-TR90-04, Edinburgh Parallel Computing Centre, 1990.

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

[15] P. Stenström. A Survey of Cache Coherence Schemes for Multiprocessors. *IEEE Computer*, 24:12–24, 1990.

[16] A. Trew and G.V. Wilson. *Past, Present, Parallel: A Survey of Available Parallel Computing Systems.* Springer-Verlag, London, 1991.

[17] V. Beneš. Optimal Rearrangeable Multistage Connecting Networks. *Bell System Technical Journal*, 43(4):1646–1656, 1964.

[18] R. Rettberg and R. Thomas. Contention is no Obstacle to Shared-Memory Multiprocessing. *Communications of the ACM*, Vol. 29(12):1202–12, December, 1986.

[19] D.D. Rogers and R.N. Ibbett. Xbar: a VLSI Circuit for Bit-sliced Packet Switching Networks. In *Information Processing 92*, volume 1, pages 562–570. Elsevier 1992, 1992.

[20] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[21] T. Blank. The MasPar MP-1 Architecture. In *Proceedings IEEE Compcon*, February 1990.

[22] Thinking Machines Corportion. *CM5 Technical Summary*, October 1991.

[23] R.N. Ibbett. The Gigaflop Quest. In *Supercomputers Conference, Paris & London.* DPMA Education Foundation, 1984.