Computing Surface 2

Overview Documentation Set

meiko

Issue status:

Draft	☐
Preliminary	☑
Release	☐
Obsolete	☐

Circulation control: External

CONTENTS

This documentation set contains the following three documents:

- *The CS-2 Communications Network.*

  Overview of the CS-2 data network. Compares the CS-2 network with other network types (logarithmic, ring etc.), and describes the benefits of Meiko's implementation.

- *The CS-2 Communications Processor.*

  Overview of the Meiko Elan communications processor, listing design objectives and implementation decisions.

- *The CS-2 Vector Processing Element.*

  Overview of the Meiko vector processing boards describing the hardware architecture, the Fujitsu μVP and SPARC processors, and compiler technology.

Computing Surface 2

Overview of the CS-2 Communications Network

meiko

Issue status:

Draft	☐
Preliminary	☑
Release	☐
Obsolete	☐

Circulation control: External

1 *GENERAL DESCRIPTION* . . . . . 1♦1

1.1 Network Characteristics . . . . . 1♦1

1.1.1 Full Connectivity . . . . . 1♦2

1.1.2 Low Latency . . . . . 1♦2

1.1.3 High Bandwidth . . . . . 1♦2

1.1.4 Fault Tolerance . . . . . 1♦3

1.1.5 Deadlock Freedom . . . . . 1♦3

1.1.6 Scalability . . . . . 1♦3

1.2 Logarithmic Networks . . . . . 1♦5

2 *THE CS-2 COMMUNICATIONS NETWORK* . . . . . 2♦1

2.1 Comparison With Fat-Tree Networks . . . . . 2♦2

2.2 Characterising a CS-2 Network . . . . . 2♦3

3 *NETWORK IMPLEMENTATION* . . . . . 3♦1

3.1 The Link Protocols . . . . . 3♦1

3.2 The Meiko Elite Network Switch . . . . . 3♦2

3.3 Routing Algorithms . . . . . 3♦3

4 *CONCLUSIONS* . . . . . 4♦1

1

GENERAL DESCRIPTION

Effective cooperation between processing elements (PEs) is a crucial factor in determining the overall sustained performance of a Massively Parallel Processing (MPP) system.

In designing the CS-2 architecture, Meiko has concentrated on minimising the impact of sharing work between processors. The effect of this is to increase the number of processors that can be effectively used to solve a problem, improving the performance of existing parallel programs, and making parallel processing effective for a significantly wider range of applications.

Every processing element in a CS-2 system has its own, dedicated interface to the communications network: a Meiko designed communications processor. The communications processor has a SPARC shared memory interface and two data links, these links connect the communications processors to Meiko designed cross-point switches.

This document provides an overview of the design of the communications network. For more information about the architecture of the communications processor see *Overview of the CS-2 Communications Processor.*

1.1 *Network Characteristics*

The design of the CS-2 data network builds on Meiko's considerable expertise in the field of MPP systems. From the outset the communications network was designed with several key characteristics in mind:

- Full connectivity.
- Low latency.
- High Bandwidth.
- Fault tolerance.
- Deadlock freedom.
- Scalability.

1.1.1 *Full Connectivity*

Every processing element (PE) has the ability to access memory on any other PE. Messages pass from the source to destination PEs via a dynamically switched network of active switch components. The network is fully connected, allowing a machine with n PEs to sustain n simultaneous transfers between arbitrarily selected pairs of PEs at full bandwidth.

The communication network does not use the PEs as part of the network, only as gateways on to it. This ensures that node resources (such as CPU and memory bandwidth) are not affected by unrelated network traffic.

1.1.2 *Low Latency*

Inter-process communications latency has two components, start-up latency which is coverered in "Overview of the CS-2 Communications Processor" and network latency. The CS-2 communication network is designed to minimise and hide network latency. Wormhole routing is used to reduce the latency through each switch stage, and the overall network topology is designed to minimise the number of stages through which a message passes. The low level communication protocols allow overlapped message acknowledgements, and the message packet size is dynamically adjusted so that it is always sufficient for full overlapping to occur.

CS-2 communications start-up latency are less than 10μs, network latencies are less than 200ns per switch.

1.1.3 *High Bandwidth*

The communication bandwidth in an MPP system should be chosen to give an appropriate compute communications ratio for current PE technology. The network design should ensure that additional bandwidth can be added to maintain the compute/communication ratio as the performance of the PEs improves with time. Although the actual required compute/communications ratio is application specific, the higher the network bandwidth the more generally applicable the MPP system will be.

CS-2 data links are byte wide in each direction and operate at 70MHz. Usable bandwidth (after protocol overheads) is 50 MBytes/s/link in each direction. Bi-sectional bandwidth of the CS-2 network increases linearly with the number of PEs. A 1024 PE machine has a bisectional bandwidth of over 50 GBytes/s.

1.1.4 *Fault Tolerance*

The network for a very large MPP system will of necessity consist of a very large number of components. Moreover for large systems a significant number of cables and connectors will be required. Under these circumstances reliability becomes a major issue. Tolerance to occasional failures by the provision of multiple routes through the network is desirable for small systems, and essential for very large systems.

CS-2 systems have two fully independent network layers and each PE is connected to both layers. In addition each layer provides multiple routes between each arbitrarily selected pair of PEs. The hardware link protocol uses Cyclic Redundancy Checks (CRCs) to detect errors on each link; failed transmissions are not committed to memory, but cause the data to be resent. All network errors are flagged to the system administrator; permanently defective links can be removed from service.

1.1.5 *Deadlock Freedom*

Routing through multistage networks is essentially a dynamic resource allocation problem and, because multiple PEs are attempting to acquire sets of route hops simultaneously, there is the potential for deadlock. The most common deadlock avoidance strategy is always to allocate resources in a fixed order. With wormhole routing, since the resources are allocated as the message wormholes through a network, this affects routing strategy for a given topology. For example in a hypercube or a grid, deadlock free routing is possible by ensuring that a PE routes by resolving the address one dimension at a time in ascending order. Note: that this actually removes the fault tolerance of the network; between PEs that differ by more than one dimension there are many possible routes, but only one can be used without risk of deadlock.

1.1.6 *Scalability*

The requirement for scalability within a network is one of the most difficult to achieve in actual systems. The three factors that need to be considered are, growth in network latency with scaling, growth in network cost, and growth in bisectional bandwidth.

The scalability properties of various network topologies are:

Type	Number of Switches	Number of Links	Latency	Bisectional bandwidth
Ring	N	N	$N-1$	2
d dimensional grid	N	dN	$d\sqrt[d]{N}$	$\sqrt[d]{N}$
Arity d Omega net	$Nlog_dN$	$\frac{dNlog_dN}{2}$	log_dN	N
Arity d Benes net	$2Nlog_dN$	$dNlog_dN$	$2log_dN$	N
Crosspoint	N^2	N^2	1	N

Where N is the number of processors in the machine, Number of links is the total number of connections between switches. Latency is the worst case number of switches which must be passed through. and Bisectional bandwidth is the worst case bandwidth between two halves of the machine.

For scalability it is essential that the bisectional bandwidth of the machine increases linearly with the number of processors. This is necessary because many important problems cannot be parallelised without requiring long distance communication (for example, FFT, and matrix transposition).

The cost (both in switches and wires) of a full crosspoint switch increases as the square of the number of processors. Adoption of this network therefore leads to a machine in which switch and wire costs rapidly dominate when significant numbers of processors are used. For the logarithmic networks the switch and wire costs increase only logarithmically faster than the number of processors. It is therefore possible to build machines which contain significantly more processors before the switch costs dominate and the machine ceases to be cost effective.

The crosspoint has the advantages of contention freedom and constant network latency for all routes. However, although the worst case latency in a logarithmic network increases slowly with the number of processors, they can be arranged so as to ensure that this increase only occurs when long distance communication is required

– performance is not dependent upon exploiting locality of reference, but doing so is beneficial.

The arity of the logarithmic network is the size of the crosspoint switch from which the network is built. So if the crosspoint is built from 2×2 switches it will have arity of 2. The choice of switch arity is highly influenced by the available packaging technology, since given a limited number of pins to connect into a switch there is a reciprocal relationship between the arity of the switch and the number of wires in each link. As the bandwidth of a link is directly related to the number of wires over which it is carried, this translates into a choice between a high arity switch which can switch many low bandwidth links, or a low arity switch for few high bandwidth links.

1.2 *Logarithmic Networks*

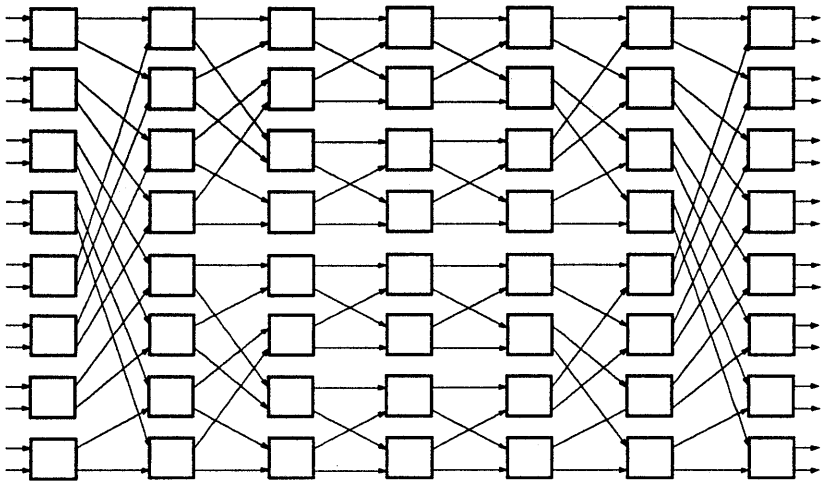
In order to analyse the CS-2 network it is useful to understand the characteristics of the Benes and Omega networks.

The main attraction of the Benes network is that it can be proved to have equivalent functionality to a full crosspoint (see Hockney and Jesshope† for a review) – any permutation of inputs can be connected to any permutation of outputs without contention. There are also multiple routes between any input-output pair. Calculating the routing to ensure that the routes are allocated without congestion for any given permutation is, however, a non-trivial problem.

This problem has been solved for a number of interesting special cases communication patterns: rings, grids, hypercubes etc. There has also been extensive simulation of these networks under a wide variety of loadings.

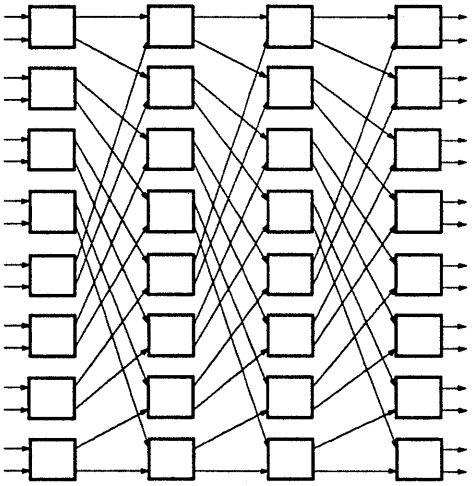
† *R.W.Hockney & C.R.Jesshope. Parallel Computers 2. Pub. Adam Hilger.*

Fig. 1.1 16 processor Benes network



In an Omega network there is only one possible route for each input-output pair. Not all possible permutations are possible without blocking, although common geometric patterns such as shifts and FFT butterflies can be shown to be contention free.

Fig. 1.2 16 processor Omega network



2

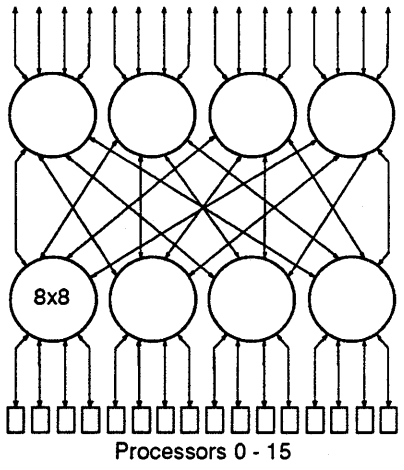
THE CS-2 COMMUNICATIONS NETWORK

CS-2 uses a logarithmic network constructed from 8 way crosspoint switches (see section 3.2 for details of their implementation) and bi-directional links.

For the purposes of this analysis it can be considered to be a Benes network folded about its centre line, with each switch chip rolling up the functionality of eight of the unidirectional two way switches.

Bandwidth is constant at each stage of the network, and there are as many links out (for expansion) as there are processors. Larger networks are constructed by taking four networks and connecting them with a higher stage of switches. A 16 processor network is illustrated in figure 2.1.

Fig. 2.1 One layer of a 2-stage CS-2 network. 16 processors are connected to stage 1, 16 links connect stage 1 to stage 2 and 16 links are available for expansion.



The scaling characteristics of the CS-2 network are shown in the table below; note that the latency is measured in switch stages for a route which has to go to the highest stage in the network.

Processors	Stages	Total Switches	Latency
4	1	1	1
16	2	8	3
64	3	48	5
256	4	256	7
1024	5	1280	9
4096	6	6168	11

One aspect of implementing the network using bidirectional switches is that routes which are relatively local do not need to go to the high stages of the switch hierarchy. So, for example, a communication to a PE which is in the same cluster of 16 processors only needs to pass through 3 switches irrespective of the total network size.

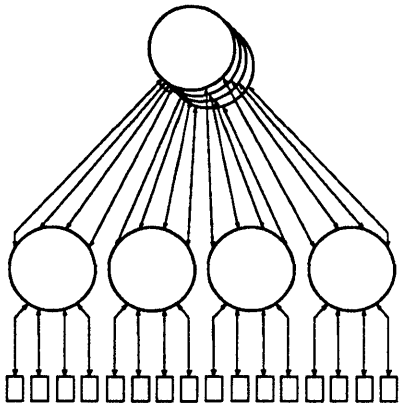
To broadcast to a range of outputs it is necessary to ascend the switch hierarchy to a point from which all the target PEs can be reached. From this point the broadcast then fans out to the target range of processors.

2.1 *Comparison With Fat-Tree Networks*

The multistage network used in the CS-2 machine can also be considered as a "fat tree". In figure 2.1 we see that for each of the higher layer switches has identical connections to the lower stages. If this is simply redrawn as shown in figure 2.2 we get the "fat tree" structure.

In fat trees packets do not always have to go to the top of the tree; packets are routed back down at the first node possible. This means that for problems which have locality of reference in communications, bandwidth at higher levels of the tree can be reduced. Exploiting the benefits of locality by reducing upper level network bandwidth has the effect of making process placement more significant. Although the CS-2 network permits this local packet routing, the bandwidth is not reduced in the higher level. This preserves the properties of Benes and Omega networks.

Fig. 2.2 One layer of a 16 processor CS-2 network drawn as a fat tree



Further properties of "fat trees" are described by Leiserson †.

2.2 *Characterising a CS-2 Network*

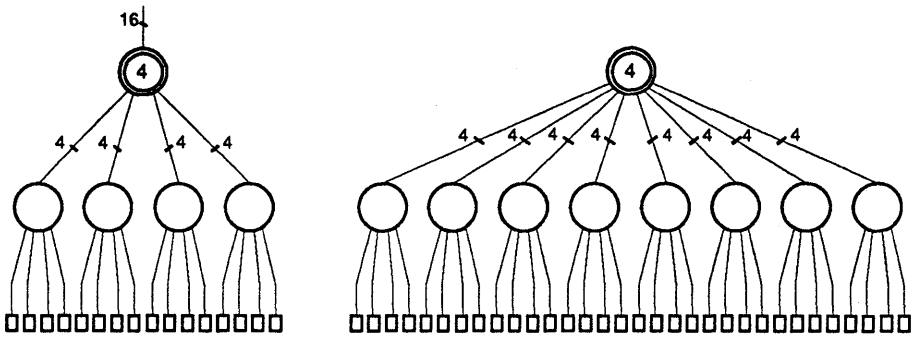
Logarithmic, or multi-stage, switch networks are described in a variety of ways by different people. The scheme used by Meiko is outlined below.

For a machine with N processors the size of its network is defined by one parameter: size. The position of a processing element is defined by two parameters: level and network id. The position of a switch in the network is defined by four parameters: layer, level, network id, and plane.

Every processor in a (complete) network is connected via a data link to a switch in the lowest stage, these switches are then connected to higher stages, etc and N links emerg from the top of the network. These links can be used to connect to further stages, or if we forgo the ability to expand they can be used to double the size of the network without introducing an extra stage (see figure 2.3).

† *C.E.Leiserson. Fat-Trees: Universal Networks for hardware-Efficient Supercomputing. IEEE Transactions on Computers, Volume C-34 number 10 (Oct. 1985). pp 892-901.*

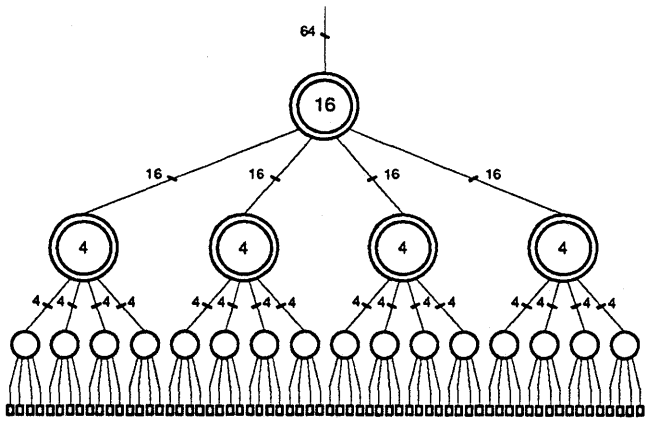
Fig. 2.3 Doubling the size of a CS-2 network



We use a binary form for network size, equal to the number of bits in the network id of the lowest processor in the network. This is used because the top stage of the network can use either 4 or 8 links.

A network has $\lfloor size/2 \rfloor$ stages, indexed by the parameter level. The top stage is 0. The deepest processors in the network have $level = size$. A network supports between $2^{size-2} + 1$ and 2^{size} processors. Note: it is not necessary for the switch network to be complete. Figure 2.4 illustrates a network of size 6.

Fig. 2.4 One layer of 64 processor (size 6) CS-2 network



There are a variety of ways of drawing these networks (see the "CS-2 Product Description" for two other examples). To draw (or manufacture !) them without crossing data links you need one more dimension than there are stages in the network.

A CS-2 machine has 2 completely independent identical switch networks. These networks are indexed by the parameter layer. Processors are connected to both layers, switches are in one layer or the other.

The position of each processing element is uniquely determined by its network id and level, which describe the route to it from all points at the top of the network ($level = 0$). Routes down are written <0-7>.<0-3>.<0-3> ... working down from the top of the network. Each digit represents the output link used on a network switch. For example, in figure 2.4 processor 0 has route 0.0.0, and processor 17 has route 1.0.1. Note that the route is the same for all starting points at the top of the network Network ids of communications processors (leaves of the network) are sometimes called elan ids.

Each stage of the switch network has 2^{size-2} switches, and 2^{level} distinct routes from the top of the network. The network id of a switch indexes the distinct routes within each level. Within each stage there are $2^{size-level-2}$ switches with the same route from the top of the network Plane indexes these points.

3

NETWORK IMPLEMENTATION

The CS-2 communications network is constructed from a VLSI packet switch ASIC — the Elite Network Switch. Interfacing between the network and the processors is performed by a second device, the Elan Communications Processor. Switches are connected to each other and to communications processors by byte wide bi-directional links.

3.1 *The Link Protocols*

The choice of a byte wide link protocol is dictated by a number of factors. The link must be wide enough to meet the bandwidth requirements of the processor, but must not be so large that the number of I/O pins on the devices becomes prohibitively large. The implementation that Meiko selected uses 20 wires for each bidirectional link, 10 in each direction. When clocked at 70 MHz this yields a bandwidth of 50 MBytes/s (after allowing for protocol overheads) in each direction. This level of performance and the underlying protocol format is appropriate for optic fibre communication over long distances (the link can be converted to a 630 MHz data stream).

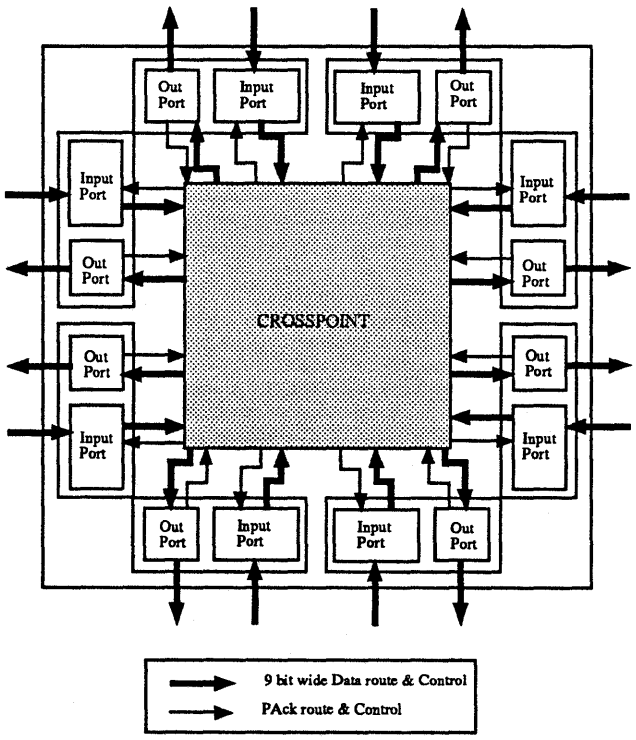
The use of bidirectional links permits flow control and acknowledge tokens to be multiplexed on to the return link. The low level flow control allows buffering of the data at the line level so that communications clock frequencies in excess of the round trip delay can be used. The interface is asynchronous and is tolerant to a 200ppm frequency difference between the ends. This means that each end can have its own clock, substantially simplifying construction of large systems.

3.2 *The Meiko Elite Network Switch*

The Elite switch is capable of switching eight independent links, each byte wide. The switch is a full crosspoint, allowing any permutation of inputs and outputs to be achieved without contention. For each data route through the switch a separate return route exists, ensuring that acknowledgements are never congested by data on the network.

The switch component contains a broadcast function that allows incoming data to be broadcast to any contiguous range of output links. The switch contains logic to recombine the acknowledge or not-acknowledge tokens from each of the broadcast destinations. To allow broadcasts to ranges of outputs over multiple switches the switch topology must be hierarchical.

Fig. 3.1 Meiko Elite network switch



The data passing through a switch is CRC checked at each switch. If a failure is detected, the message is aborted, an error count is incremented, and the packet is negatively acknowledged. This ensures that incorrect data is removed from the network as soon as possible.

Routing within the switch is byte steered. On entry into a switch the first byte of any packet is interpreted as the destination output or range of outputs. This byte is stripped off within the switch so that the next byte is used for routing in the following switch. The latency through each switch device is 7 clock cycles for outgoing data, and 5 cycles for returning acknowledge tokens. The switch contains no routing tables of any sort. The translation between destination processor and route information is performed entirely on the communications processor, where it can be more easily modified or updated.

Although the switch component is an 8×8 crosspoint, the use of bi-directional links means that for the purposes of constructing logarithmic networks the effective radix is 4.

Each switch has a performance monitoring and diagnostic interface connected to the CS-2 control network. This allows collection of statistics on error rates and network loading.

3.3 *Routing Algorithms*

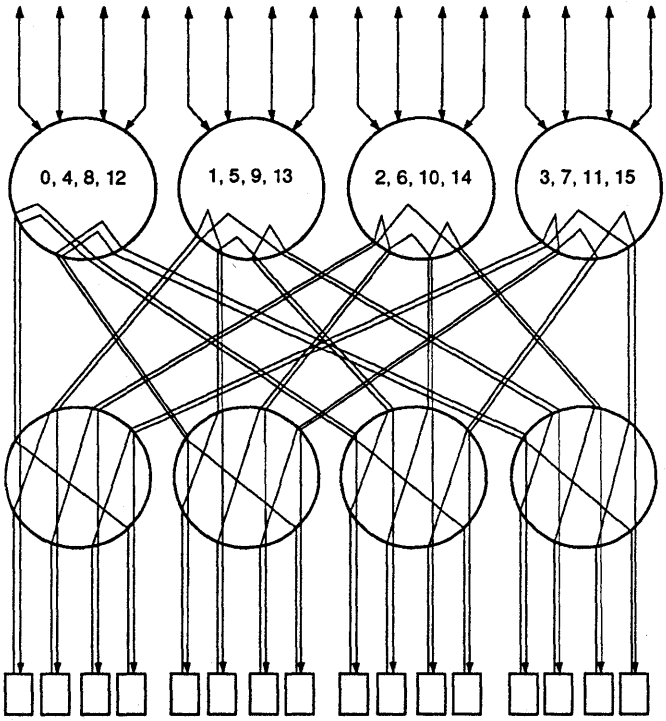
Although the CS-2 data network can have the congestion properties of a full crosspoint, but achieving this requires allocation of routes in a non-contending fashion. In the CS-2 network the route is pre-determined by the communications processor. By storing the route information in the Elan it becomes easier to change the routing algorithm, due to machine reconfiguration or link failure for example.

The translation from a processor address to network route is handled in the communications processor by a look-up, the table is stored in the memory of the PE and indexed by destination processor. Each table entry contains four alternative routes to the destination processor, one of which is selected. The specification of alternative routes allows the even distribution of traffic throughout the network, although all four routes may be identical when this is undesirable. Each PE maintains its own look-up table which may be different to the others, thus enabling any function of source/destination addressing to be used from.

One simple routing function is to direct all data for the same destination processor through a single switch node at the top of the hierarchy. This allows the network to perform two functions: data distribution, and distributed arbitration for use where

many senders wish to communicate with the same processor simultaneously. By adopting this strategy we ensure that if blocking does occur, it does so as soon as possible, and consumes little of the network resource. Using this simple algorithm has the effect of reducing the network to an Omega network — essentially the second, return part, of the network is guaranteed non blocking, and performs a simple data ordering operation. By virtue of its similarity to an Omega network, this network will be non-blocking for arbitrary shifts and FFT style permutations.

Fig. 3.2 Shift by 5 on a 16 processor CS-2 network



The programmable nature of the CS-2 communication network allows users (who are so inclined) to design their own routing algorithms. This permits optimisation of routing for specific traffic patterns or study of the effect of routing strategy on network performance.

4

CONCLUSIONS

The CS-2 network provides a flexible solution to the problem of connecting together large numbers of processing elements. The network can provide equivalent performance to a full crosspoint, but can be simplified where this level of interconnect is not required. The combination of Meiko Elan and Elite network technology allows considerable flexibility in the choice of routing algorithm.

The communications co-processor uses a lookup table to map abstract processor addresses to switch network routes. By maintaining the lookup tables within the PE memory they are easier to modify to reflect changing workload or network failures. By maintaining separate lookup tables on each communications processor, any function of address mapping may be implemented. The Elan communications processor acts as a gateway into the CS-2 switch network.

The Elite network switch is a full 8×8 crosspoint switch. It is the fundamental building block of the CS-2 communications network. The route through the switch is determined by the header byte of each incoming message. Headers are added by the communications processor and removed by the switch as the message passes through it. In addition to a direct mapping from input link to output link, the switch supports broadcast and combining operations by mapping a single input to a contiguous range of outputs.

Overview of the CS-2 Communications Processor

meiko

Issue status:

Draft	☐
Preliminary	☑
Release	☐
Obsolete	☐

Circulation control: External

1 *OVERVIEW* . . . . . 1♦1

2 *EFFICIENT INTER-PROCESSOR COMMUNICATIONS* . . . . . 2♦1

2.1 Latency and Bandwidth . . . . . 2♦2

2.2 Network Security . . . . . 2♦3

2.3 Virtual Addressing . . . . . 2♦3

3 *ELAN FUNCTIONALITY* . . . . . 3♦1

3.1 Checking . . . . . 3♦1

3.2 Translation . . . . . 3♦2

3.3 Copying . . . . . 3♦3

3.4 Device Control . . . . . 3♦3

3.5 Thread Processor . . . . . 3♦4

3.5.1 Thread code . . . . . 3♦4

3.5.2 Events . . . . . 3♦4

3.5.3 Other Forms of Remote Access . . . . . 3♦5

4 *ELAN IMPLEMENTATION* . . . . . 4♦1

4.1 Design Approach . . . . . 4♦1

4.2 Optimisation . . . . . 4♦2

4.3 Clocking . . . . . 4♦2

4.4 Re-hosting . . . . . 4♦2

5 *USING THE COMMUNICATIONS PROCESSOR* . 5♦1

5.1 DMA Transfers . . . . . 5♦1

6 *CONCLUSIONS* . . . . . 6♦1

1

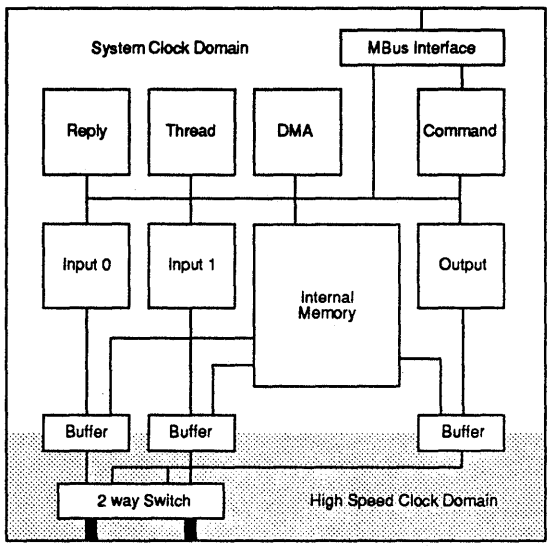
OVERVIEW

Effective cooperation between processing elements is a crucial factor in determining the overall performance of an MPP system. Maintaining effective inter-processor communication as a system scales in size is a vital aspect of preserving balance.

In designing the CS-2 architecture Meiko has concentrated on minimising the impact of sharing work between processors. The effect of this is to increase the number of processors that can be used effectively to solve a problem, improving the performance of existing parallel programs and making parallel processing efficient for a wider range of applications.

Every processing element in a CS-2 system has its own, dedicated interface to the communications network: a Meiko designed communications processor. The communications processor has a SPARC shared memory interface and two data links. Data links are connected by Meiko designed 8×8 cross-point switches. Data links are byte wide in each direction and operate at 70MHz, providing 50 Mbytes/s of user bandwidth in each direction.

Fig. 1.1 The Elan Communications Processor



The communications processor supports remote read, write and synchronisation operations specified by virtual processor number and virtual address – both are checked in hardware. Latency hiding is supported by non-blocking instructions, instruction sequences and completion tests.

This document provides an overview of the design of the communications processor and its usage. For more information about the architecture of the data network see the *CS-2 Communications Network Overview*.

2

EFFICIENT INTER-PROCESSOR COMMUNICATIONS

In a distributed memory system, work is shared between processors by exchanging data over a communications network. The efficiency of data exchange controls the effectiveness of work sharing and hence the number of processors that can be used on a given problem.

Rather than design a new processor with built in communications capability Meiko chose to separate the issues in the design of the CS-2. Processing elements consist of a high performance RISC CPU (with optional vector processing capabilities) and a dedicated communications processor.

The interface between the communications processor and the rest of the processing element is central to the efficiency of the CS-2 network. It provides the following essential features:

- Low communication start-up latency.
- High bandwidth inter-processor communication..
- Security against corruption.
- Operation in a network-wide virtual addressing, virtual process environment.

2.1 *Latency and Bandwidth*

Efficient inter-processor communication requires both low latency and high bandwidth. While solutions to the bandwidth problem can be addressed by ever improving hardware technology, these improvements only exacerbate underlying latency problems.

To show that this is the case consider a system with a communications start-up latency of 10μs. To transfer a 100 byte message via a 1 Mbyte/s network we will get an achieved bandwidth of 0.9 Mbytes/s (90% efficiency). For the same transfer over a 50 Mbytes/s network, the achieved bandwidth is just 8.3 Mbytes/s (16% efficiency). Clearly the improvements in bandwidth for this example system have been severely limited by the start-up latency and the size of the data transfer.

By using a dedicated communications processor Meiko have reduced start-up latency by implementing in hardware the communications code that would normally execute on the main processor.

The data links joining communications processors and network switches are byte wide in each direction. Links are clocked at 70MHz. Their bandwidth after protocol is 50MBytes/s in each direction. The CS-2 data network is a fat tree with constant bandwidth between stages. It is capable of supporting full bandwidth transfers between all pairs of processors (see the *CS-2 Communication Network Overview* for more details).

Moving communications code from the main processor to a communications engine does not in itself reduce latency. Performance improvements come from running the right code in the right places. In particular there are significant benefits to be had from moving the lightweight interrupt intensive operations associated with inter-process communication off a conventional microprocessor and onto a communications processor designed specifically for this purpose.

2.2 *Network Security*

The CS-2 communications network is shared by both user and system level communications so it is vital that a security mechanism is used to prevent unrelated communications from interacting. To relieve the burden of checking from the main processor and to reduce start up latency, the main processor issues unchecked communication instructions to the communications processor, the communications processor then implements the security strategy in hardware. This mechanism is preferable to the more conventional use of kernel mapped devices, which use checked system calls to access the device, often with a significant performance impact (a checked system call in a 40 MHz SPARC takes approximately 50μs).

The CS-2 network protects processes from communications errors that occur within other unrelated processes, but does not protect a process from errors within itself. This is the same model as that employed for memory protection by the UNIX operating system – processes are protected from each other, but not from themselves.

2.3 *Virtual Addressing*

The communications processor uses separate page tables from the main processor. This means that a user process need not make its entire address space visible when it communicates, only the portion that contains the data need be mapped for communication. Secondly, separate page tables may be used to reduce the amount of cache flushing in non cache-coherent systems; in a write through cache only those pages that are mapped with write permission need be flushed.

The two sets of page tables are kept in step by a modified page out daemon and new page in code in the operating system. The modified page out daemon modifies both sets of tables, whereas the new page in code handles the asynchronous page faults from the communications processor.

3

ELAN FUNCTIONALITY

The functionality of the communications processor was decided by drawing on experience from Meiko's CSTools/CSN communication software, used to create a programming environment over Transputer networks, and other message passing systems such as the Chorus Nucleus. This analysis showed that the start-up process consists of four components:

- Checking.
- Translation.
- Copying.
- Device control.

Each of which is important if start-up latency is to be minimised.

3.1 *Checking*

The CS-2 supports virtual memory addressing on each processing element, allowing it to implement a fully distributed store for operating system use, and permit it to implement the applications binary interface (ABI) for the base microprocessors. The communications processor therefore has two types of parameters to check: memory addresses and process addresses.

The communications processor receives unchecked virtual memory addresses from the main processor so it must incorporate a memory management unit (MMU). The MMU used within the Elan supports multiple simultaneous contexts allowing I/O to continue for suspended processes.

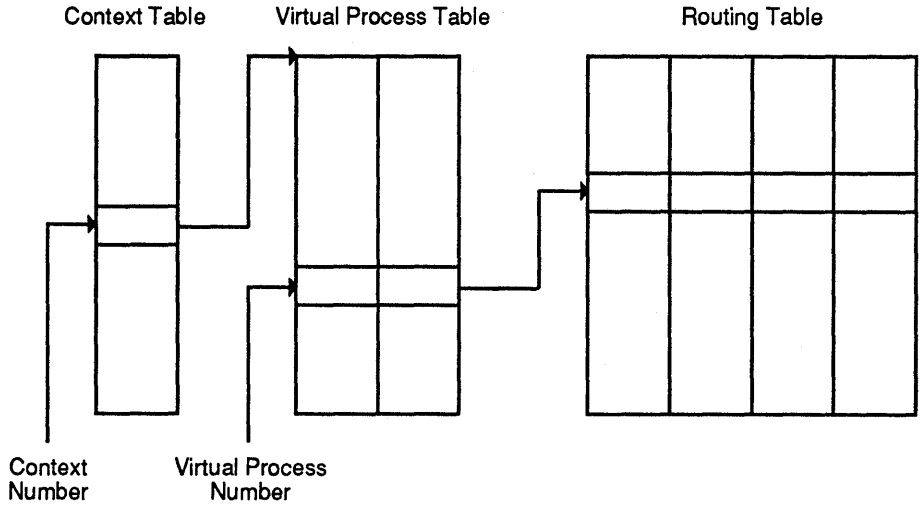
The checking of process addresses is analogous to the checking of memory addresses. It is implemented by a simple table look-up and exception mechanism. The communications processor is designed to handle the common case where a user is trying to communicate with other processes for which it has permission; an exception is generated whenever there is no permission. As checking is performed

independently on each of the communications processors, failed processing elements can be removed from service by removing them from each communications processor's list of valid destinations.

3.2 *Translation*

Process and memory translation within the communications processor is implemented through the same mechanism as the checking, that is, by table look-ups. Memory address translation yields the same results as the main processor's translation mechanism. Dynamic process translation yields two components: a destination processor and a destination context. There are no physical processor or memory addresses in user space.

Fig. 3.1 Elan Process Translation



Virtual process ids are translated through a per context virtual to physical processor translation which points at the route bytes needed to direct a message to this processor.

3.3 *Copying*

The communications processor supports a number of features to remove the requirement for copying of data. By using network wide virtual addressing there is no need to copy data into physically mapped output buffers, a common technique in distributed systems to overcome the problems of virtual address translation and page locking during communication. Furthermore, because the main processor and the communications processor share a common memory bus (a SPARC MBus) and the same cache coherency protocols, the problems associated with cache coherency are also avoided.

Clearly the avoidance of unnecessary copying contributes greatly to reduced start-up latency and efficient use of memory bandwidth. For messages that are copied once on sending, this adds `message size × 2 / memory bandwidth` to the start-up latency, and consumes three times as much store bandwidth.

3.4 *Device Control*

The final requirement of message start-up code is in device control. This is setting up the communications parameters in store, signaling to the communication device, and responding to interrupts returned by the communications processor.

Control of the communications processor is via a command port which is normally mapped into the user address space. The command port consists of a range of memory addresses. The communications processor command is determined by extracting 5 bits from the address that is used. The data that is used by the communications processor command corresponds to the 32 bits of data that are written to that memory address. Commands sent to the command port are written in a single read-modify-write cycle and are acknowledged with the value that is read back (which will be non-negative if the command is accepted). The kernel can prevent the user issuing certain commands by mapping limited portions of the command port address space in to the user address space.

Exceptions generated by the communications processor may be handled by the communications processor's own thread processor, without direct intervention by the main processor.

3.5 *Thread Processor*

One of the objectives of the Elan communications processor is to reduce the number of interrupts and system calls that must be executed to perform message passing. As we have seen the combination of the user mapped command port and the Elan communication processor's security mechanisms allows user level code to initiate remote memory accesses without making a system call. In many cases, however, message protocols require higher level functions than simply the transfer of data. Other common requirements are for synchronisation between processes executing on separate processors, and allocation of global resources. To support these requirements the Elan communications processor includes a RISC processor which can execute user level code independently of the main node processor, and also create additional network transactions.

The hardware and microcode of the thread processor support an extremely lightweight scheduling mechanism. This allows lightweight processes (*threads*) running on the thread processor to be suspended and then rapidly rescheduled by the hardware when the relevant event has occurred.

The user level code in the main node processor can directly request the execution of a thread process through access to the appropriate command port. The thread code has no more privileges than the user code which initiated it. The Elan communications processor uses its page tables for the relevant user context whenever it makes a store access from the thread.

3.5.1 *Thread code*

Thread code can be written in ANSI C. An inlined library provides access to the Elan communication processor I/O instructions without the overhead even of a subroutine call.

3.5.2 *Events*

Events provide a general mechanism by which synchronisation may be achieved between lightweight threads running either in the same, or different, Elan communication processors. In addition an event can be used to cause an interrupt to the main node processor. An event is represented by a double word in store.

A thread can perform the following operations on either local or remote events:

Wait If the event has already been set, then execution continues and the event is unset. Otherwise the thread is suspended on the event until the event is set, when it will be rescheduled.

Set	The event is set. If there was an action already present on the event then it is performed.
Clear	If the event was set it is cleared.
Test	Poll the status of an event without modifying or suspending on it.

There are various possible actions which can occur when an event is triggered, these depend on what has been suspended in the event structure:

A local thread	The thread is placed back on the thread run queue, so will resume execution.
A remote thread	The remote thread is rescheduled on its own processor.
A local interrupt	The main processor is interrupted.

Events also support queues of outstanding requests. When a queued event is set, the first action on the queue is executed, and the queue updated to point to the next action.

3.5.3 *Other Forms of Remote Access*

In addition to events, the Elan also supports other forms of remote store access. In particular thread code can generate network transactions to perform:

Atomic Swap	The word at the given remote address is returned, and overwritten with the word sent in the message.
Atomic Add	The word sent in the message is atomically added to the data at the remote address. The original remote data may optionally be returned.
Atomic test and store	The word at the remote address is compared with a test value sent in the message. If equal then a new value sent in the message is written to the remote store, otherwise the remote store is unchanged. The original remote value may optionally be returned.
Remote compares	The word at the remote address is compared with the given data using one of the operations ==, =, >= or <. The result of the comparison is returned as an acknowledge or negative acknowledge.

The broadcast capabilities of the Elite switch can be used to combine the results of a broadcast remote compare operation into a single result.

4

ELAN IMPLEMENTATION

The Elan communications processor and Elite network switch were designed by Meiko at its European subsidiary in Bristol, UK.

The two ASICs are implemented on 1.0 micron drawn 3 layer metal CMOS *sea of gates* gate arrays. Both components use a 110,000 gate base array which incorporates more than 440,000 transistors.

The communications processor achieves a utilisation of approximately 75% (representing around 83,000 gates), the network switch utilises approximately 55% (61,000 gates).

Both components are packaged in 208 pin PGAs.

4.1 *Design Approach*

The network design was subjected to extensive parallel simulation using models written in 'C'. The network components were designed using an iterative top-down approach.

A gross functional model was written in Verilog to simulate the behaviour of the Elan. This was rewritten and refined to produce a cycle by cycle functional model encompassing the full state required, together with a module hierarchy. These modules were converted into a stylised register transfer subset of the Verilog language, which aided their synthesis into gate level logic. This synthesis/conversion was done by hand for the majority of modules. The development for the Elite followed a similar design flow to the Elan.

The gate and functional models were automatically compared using a number of techniques, and were kept in step throughout the design. The two model types have different simulation properties. The functional model consumes considerable less memory however the gate model simulation executes faster if the entire data set is present in memory during the the simulation. Simulations of CS-2 data networks

(communications processors and network switches) were conducted using functional models.

The gate level logic was targeted at a vendor independent technology library. Vendor selection was based on the two qualities of cost and process speed requirement. Re-targeting of the implementation to a different vendor/process would be relatively easy to perform.

4.2 *Optimisation*

The 70MHz operating frequency of the Elan-Elite network required extensive optimisation of logic delays using a timing analyser to give guaranteed worst case temperature, process, and voltage operation. The layout associated with these high frequency circuits was carefully floor planned.

4.3 *Clocking*

The communications processor operates in two separate clock domains, one synchronised to the host processor's memory interface, the other to the 70MHz communications clock. Appropriate synchronisation occurs when data is transferred between the two domains.

The network switch also synchronises data from each link to its local 70MHz clock, the data is sampled and regenerated at each switch.

The communications processor and network switch therefore remove the requirement for global clock distribution throughout the machine.

4.4 *Re-hosting*

The interface between the communications processor and the processing element is through the processor's cache coherent memory protocol. The initial version of the communications processor implements the SPARC MBus protocol, however re-targeting of the communications processor to a different host memory bus would require only minor modifications to the communications processor, since the memory interface is a well contained module within the design consisting of around 5000 gates.

5

USING THE COMMUNICATIONS PROCESSOR

In this section we show in outline how the communications processor is used to communicate with other processes via the data network. The example shows how to initiate a DMA transfer to remote store.

5.1 *DMA Transfers*

In the previous sections we have seen that a key factor in the design of the communications processor is that it offers low communication start-up latencies, and that communication start-up requires minimal intervention by the main processor. For a typical DMA transfer of data to a remote processor, the actions required by the main processor are as follows:

- User program creates a DMA structure in store identifying the characteristics of the transfer (source and destination addresses, amount of data, etc). This could be done in advance if the same access is to be made repeatedly.
- User program issues DMA command with RmW to command port. The address of the DMA structure is written to the appropriate address in the command port.
- User program checks command accepted; a value of greater than or equal to 0 in the command port indicates that the command was accepted.

The main processor is now free to continue with its work leaving the communications processor to transfer the data, and to ensure its integrity. The actions now required by the communications processor are:

- Command processor reads the 32bit data from the command port and uses this to locate the DMA descriptor. The descriptor is read into the communications processors DMA queue.
- DMA processor reads the queue item in.

- DMA processor performs destination process translation.
- DMA processor reads route information.
- DMA processor reads source data in and starts to send. The route information is prepended to the data, and is stripped off as it passes through the switch network.

If the main processor wanted confirmation that a DMA had completed it would include a pointer to an event in the DMA description. Polling this event (when there is no more useful work to do) would confirm completion of the transfer.

CONCLUSIONS

Efficient inter-processor communications requires the right balance of latency and bandwidth. CS-2 uses Meiko's own communication hardware, developed from many years experience in the massively parallel processing field, to create a network with both high bandwidth and low start-up latency.

The Elan communications processor is key to minimising the network latency. It serves not just as a communications co-processor, but aims to minimise the amount of message start up code, and therefore minimise startup latency. For simple communications the overhead on the main processor can be reduced to a single read modify write. More complex protocols require small fragments of code to be run on the communications processor. The requirement for copying of messages is removed by the ability of the communications processor to operate in virtual store. Protection is implemented by hardware table look ups of translation tables which impose low overhead on valid operations, and generate exceptions in the much less frequent error cases.

Overview of the CS-2 Vector Processing Element

meiko

Issue status:

Draft	☐
Preliminary	☑
Release	☐
Obsolete	☐

Circulation control: External

1 *GENERAL DESCRIPTION* . . . . . 1♦1
1.1 MK403 Overview . . . . . 1♦1
1.1.1 μVP Vector Processor . . . . . 1♦2
1.1.2 Superscalar SPARC Processor . . . . . 1♦4
1.1.3 Memory System . . . . . 1♦4

2 *COMPILERS* . . . . . 2♦1
2.1 Overview . . . . . 2♦1
2.2 Languages . . . . . 2♦1
2.2.1 FORTRAN and C . . . . . 2♦1
2.2.2 High Performance Fortran (HPF) . . . . . 2♦2
2.2.2.1 Fortran-90 Binding . . . . . 2♦2
2.2.2.2 New Features in High Performance Fortran . . . . . 2♦2
2.3 Code Generation . . . . . 2♦3

3 *CONCLUSIONS* . . . . . 3♦1

GENERAL DESCRIPTION

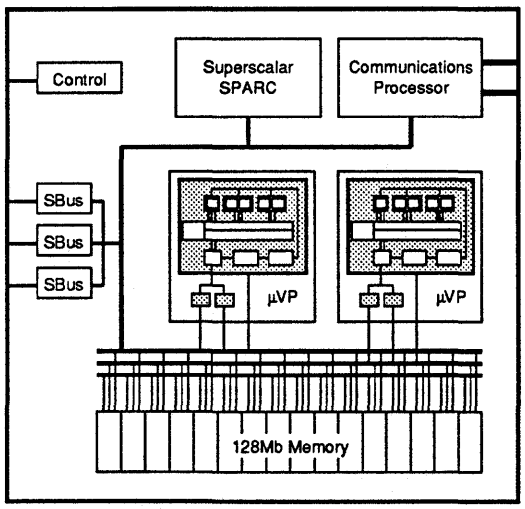
This document describes the architecture of the CS-2 vector element (MK403). It briefly describes the internal architecture of the Fujitsu μVP and the compilation strategy used to exploit the combined resources of the SPARC and multiple μVP processors.

For more details of the workings of the μVP see the "Programmers Reference Manual".

1.1 *MK403 Overview*

The CS-2 vector element incorporates a 40MHz Superscalar SPARC, a Meiko Elan Communications Processor and 2 Fujitsu μVP vector processors. All processors have access to the memory system via 3 memory ports, two of which are used by the vector processors and the third by the SPARC and Elan which share an MBus.

Fig. 1.1 CS-2 Vector Processing Element



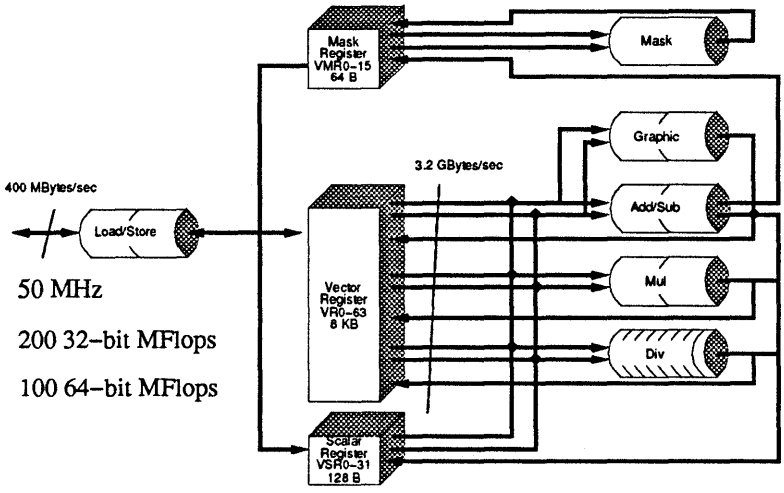
The memory system is implemented as 16 independent banks, with a (current) total capacity of 128 MBytes. Memory bandwidth for each of the 3 ports is 1.2 GBytes/s, with a total bandwidth of 3.2 GBytes/s.

External I/O support is provided through 3 SBus interface slots – primarily used for disk controllers, but capable of supporting network interfaces and graphics cards.

1.1.1 *μVP Vector Processor*

The μVP operates with a 50MHz (20ns) clock. It has a vector register architecture with 8 KBytes of vector registers, configurable as between 8 and 64 vectors each of 16-128 64-bit registers (see below). In addition there are 32 scalar registers and a set of vector mask registers whose format tracks that of the vector registers.

Fig. 1.2 μVP Vector Processor



Configuration of the μVP vector and mask registers:

Precision	Length	Number of registers
Single	32	64
Single	64	32
Single	128	16
Single	256	8
Double	16	64
Double	32	32
Double	64	16
Double	128	8

The μVP has seperate pipes for floating point multiply, floating point add, floating point divide, and integer operations. The floating multiply and add pipes can each deliver one double precision (64 bit) or two single precision (32 bit) IEEE format result(s) on every clock, giving a maximum theoretical performance of 100 MFLOPS/s double precision and 200 MFLOPS/s single precision; the divide pipe can simultaneously deliver an extra 6 MFLOPS/s in either single or double precision. Both the add and multiply pipes have the low latency (pipe depth) of two cycles (40ns), with one extra cycle being required to read and one to write the vector register file.

The vector register elements are scoreboarded, so that chaining between input and output operands occurs wherever possible without requiring explicit compiler or programmer intervention.

The μVP has a single load/store pipe which is used for accessing the memory system. This is a 64 bit interface which can generate four addresses on consecutive clock cycles before stalling for the returned data. Once the data is present a 64 bit word can be transferred on each clock cycle, giving a maximum bandwidth of 400 MBytes/s.

The instruction set includes masked vector operations, compressions (sum, maxval, maxindex, minval, minindex), vector compress under mask and expand under mask operations, as well as logical operations on integers and mask registers and conditional branches. Vector loads and stores can be performed with strides and under mask, as well as with an index vector ("indirect"). For further information about the μVP instruction set the *μVP Programmers reference Manual.*

1.1.2 *Superscalar SPARC Processor*

The MK403 uses SPARC MBus processor modules. It is generally populated with a 36 or 40HMz Viking SPARC, but other standard modules can be used.

The Superscalar SPARC has two independent integer ALUs which can execute separate arithmetic operations or can be cascaded so that the processor can execute two dependent instructions in the same cycle. It has instruction issue logic which can issue up to three instructions on the same cycle. Load and stores operations of all data types to the on chip 16 KBytes data cache occur in a single cycle. The floating point unit can execute multiply and add instructions simultaneously, though only one floating point instruction can be issued per cycle.

1.1.3 *Memory System*

The Superscalar SPARC processors and Elan communication processor are connected to a standard 40MHz MBus. The vector processors and MBus are connected to a 16 bank memory system, each bank providing 64 bits of user data (78 bits including error checking and correction, implemented using 20 by 4 bit DRAMs with two bits unused). Error detection and correction is implemented on each half word (32 bits), allowing write access to 32 bit (ANSI-IEEE 754-1985 `single`) values to be performed at full speed, without requiring a read modify write cycle.

Each bank of memory maintains a currently open DRAM page within which accesses may be performed at full speed. This corresponds to a size within the bank of 8 KBytes, giving 128 KBytes total for the 16 banks. When an access is required outside the currently open page a penalty of 6 cycles is incurred to close the previous page, and open the new one.

Refresh cycles are performed on all banks within a few clock cycles of each other, thus allowing the cost of re-opening the banks to be pipelined (since the μVP can issue four addresses before stalling for the data from the first), and reducing the overhead of refresh to a few percent of memory bandwidth.

The memory system is clocked at the same speed as the μVP processors (50 MHz), and accesses from the 40 MHz MBus are transferred into the higher speed clock domain. When accessing within an open page each memory bank can accept a new address every two cycles (40ns), and replies with the data four cycles (80ns) later, giving a bandwidth of 8 Bytes every two cycles (40ns), that is 200 MBytes/s. Since there are 16 banks, the total memory system bandwidth is thus 3.2 GBytes/s.

Each μVP can issue a memory request every cycle (20ns), and can issue 4 addresses before it requires data to be returned. In the absence of bank contention (which will be discussed below), after a start up latency of four cycles, these requests can be

satisfied as fast as they are issued, giving each μVP a steady state bandwidth of 8 Bytes every 20ns, that is 400 MBytes/s.

Since each bank can accept a new address every two cycles (40ns), but the μVP can generate an address every cycle (20ns) there is the possibility of bank contention if the μVP generated repeated accesses to the same bank. With a simple linear mapping of addresses to banks, this would occur for all strides which are multiples of 16 (for 64 bit double precision accesses). Such an access pattern would then see only one half of the normal bandwidth, that is 200 MBytes/s. All other strides achieve full bandwidth.

To ameliorate this problem as well as allowing the straightforward linear mapping of addresses to banks, Meiko also provide the option (through the choice of the physical addresses which are used to map the memory into user space) of scrambling the allocation of addresses to memory banks. The mapping function has been chosen to guarantee that accesses on "important" strides (1, 2, 4, 8, 16, 32) achieve full performance. Access on other strides may see reduced performance, but there are **no** strides within the open pages which see the pathological reduction to one half of the available bandwidth.

2

COMPILERS

2.1 *Overview*

The Fortran and C compilers for the vector processing element generate code for all three processors: using the scalar processor to execute scalar code, and the two μVPs to execute vector loops. They incorporate a wide range of standard optimisations:

> constant folding, constant propagation, common subexpression removal, **automatic function inlining,** instruction scheduling, loop invariant removal, induction variable detection, **software loop pipelining, loop splitting, loop interchange, loop vectorization, vectorization of intrinsic functions, vector idiom recognition,** dead code removal,

as well as proprietary optimisations for the CS-2.

2.2 *Languages*

2.2.1 *FORTRAN and C*

The FORTRAN language conforms to ANSI X3.9-1978, with the addition of many extensions including CRAY Pointers, `ALLOCATABLE` arrays and `COMMON` blocks, VMS structures, `END DO` statements, and `NAMELIST` I/O. The compiler also recognises the CRAY vectorization directives (e.g. `CDIR$IVDEP`).

The C compiler accepts the ANSI C language, and incorporates the same vectorizer and code generator as the FORTRAN compiler.

2.2.2 *High Performance Fortran (HPF)*

The High Performance Fortran Forum (HPFF) is a group of industrial and academic organisations which is open to all. The objective of the group is to standardise annotations and extensions to ISO 1539:1991 (Fortran-90) to allow a Fortran program to be efficiently executed under a data parallel execution model. HPFF have published the final draft specification for public comment. An HPF compiler for the CS-2 is currently under development.

2.2.2.1 Fortran-90 Binding

The HPFF has chosen Fortran-90 as the language for extension. The new dynamic storage allocation and array calculation features make it a natural base for HPF. The HPF language features fall into 3 categories with respect to Fortran-90:

- New directives.
- New language syntax.
- Language restrictions.

The new directives are structured comments which suggest implementation strategies or assert facts about a program to the compiler. They may affect the efficiency of the computation performed, but do not change the value computed by the program. The form of the HPF directives has been chosen so that a future Fortran standard may chose to include these features as full statements in the language.

A few new language features, namely the `FORALL` statement and certain intrinsics, are also defined. They were made first-class language constructs rather than comments because they can affect the interpretation of a program, for example by returning a value used in an expression. These are proposed as direct extensions to the Fortran-90 syntax and interpretation.

Full support of Fortran sequence and storage association is not compatible with the data distribution features of HPF. Some restrictions on use of sequence and storage association are defined. These restrictions may in turn require insertion of directives into standard Fortran programs in order to preserve correct semantics.

2.2.2.2 New Features in High Performance Fortran

High Performance Fortran extends Fortran in several areas. These areas include: data distribution features, parallel statements, extended intrinsic functions, foreign procedures and changes in sequence and storage association.

2.3 *Code Generation*

The compilers for the vector processing elements produce code that executes on the SPARC, and, dynamically if appropriate, on the two attached vector processors. Scalar code executes on the Superscalar SPARC processor, vector code is compiled to execute on either the SPARC processor or the μVPs, or both.

Where the vector length is not known at compile time, the compiler generates both vector code (for the μVPs) and scalar code: the choice of which code to execute being made at run time based on the actual vector length.

The vectorizer exploits the multiple μVPs in two different ways. Where there is a loop around a vector loop, as shown below, the compiler will generate code which executes alternative iterations of the outer loop on each of the μVPs; each instance of the inner loop (and its strip-mine loop) will execute entirely on a single μVP:

```
DO I = 1,N
    DO J = 1,M
        X(J,I) = A*X(J,I) + Y(J)
    END DO
END DO
```

The generated code is analogous to the following (pseudo) source code:

In parallel on μVP 1

```
DO I = 1,N,2
    DO J = 1,M
        X(J,I) = A*X(J,I) + Y(J)
    END DO
END DO
```

and on μVP 2

```
DO I'=2,N,2
    DO J' = 1,M
        X(J',I') = A*X(J',I') + Y(J')
    END DO
END DO
```

Where there is no outer level independent loop which can be exploited, then the compiler will split the individual strips of the inner loop across the two μVPs. Consider the following example:

```
      DO J = 1,M
          X(J) = A*X(J) + Y
      END DO
```

The generated code is analogous to the following (pseudo) source code:

In parallel on μVP 1

```
      IBASE = 1
      ILEN  = MIN( M-IBASE, stripLength)
C     Strip mine loop
      DO WHILE (ILEN .GT. 0)
C         Vector operation
          DO J = IBASE,IBASE+ILEN
              X(J) = A*X(J) + Y
          END DO
C         2 here is number of uVPs involved
          IBASE = IBASE + 2 * stripLength
          ILEN  = MIN( M-IBASE, stripLength)
      END DO
```

and on μVP 2

```
      IBASE' = stripLength
      ILEN'  = MIN( M-IBASE', stripLength)
C     Strip mine loop
      DO WHILE (ILEN' .GT. 0)
C         Vector operation
          DO J' = IBASE,IBASE+ILEN'
              X(J') = A*X(J') + Y
          END DO
C         2 here is number of uVPs involved
          IBASE' = IBASE' + 2 * stripLength
          ILEN'  = MIN( M'-IBASE', stripLength)
      END DO
```

All of this code executes on the μVP.

The code generator schedules vector instructions to ensure that chaining of vector operations happens as often as possible (by ensuring that there are no scalar operations scheduled between dependent vector operations).

If the operation is a vector sum, then each μVP will produce the sum of the elements it processes, and the final accumulation of the two partial sums will be performed by the scalar processor.

3

CONCLUSIONS

Each CS-2 vector processing element consists of a Superscalar SPARC, a Meiko Elan communications processor, and 2 Fujitsu μVP vector processors sharing a three ported memory system. Cycle time is 20ns, performance peaks at 200 MFLOPS per processing element in 64 bit arithmetic, or 400 MFLOPS in 32 bit.

To achieve high performance on real world problems you need the correct balance of CPU and memory system performance. The CS-2 vector memory system is organised as 16 independent banks, enabling it to sustain 1.2 GBytes/s on direct, strided, or indirect addressing. Memory capacity is currently 32 or 128 MBytes per processing element.

The CS-2 development environment for the vector processing elements includes compilers for FORTRAN-77, ANSI C, Fortran-90, and High Performance Fortran. The compilation system produces compiled code that executes on either the SPARC processor or, dynamically where appropriate, on the two attached vector processors.