C *omputing*
S *urface*

Vector Processing Element Overview

S1002–10M101.05

meiko

Meiko's address in the US is:

Meiko
130 Baker Avenue
Concord MA01742

508 371 0088
Fax: 508 371 7516

Meiko's full address in the UK is:

Meiko Limited
650 Aztec West
Bristol
BS12 4SD

Tel: 01454 616171
Fax: 01454 618188

Issue Status:

Draft	
Preliminary	
Release	x
Obsolete	

Circulation Control: *External*

Contents

General Description 1

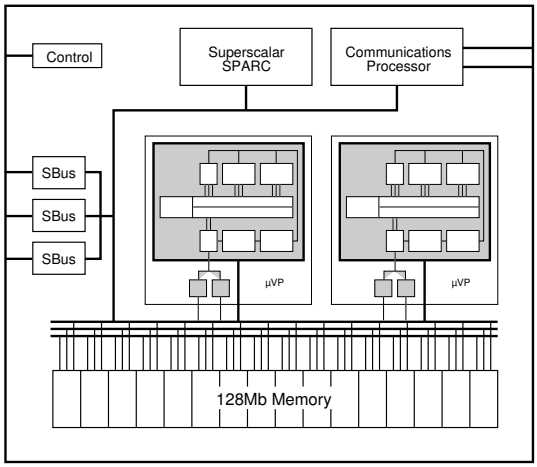
This document describes the architecture of the CS–2 vector element (MK403). It briefly describes the internal architecture of the Fujitsu µVP and the compilation strategy used to exploit the combined resources of the SPARC and multiple µVP processors.

For more details of the workings of the µVP see the µ*VP Programmers Reference Manual*.

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).

Figure 1-1 CS-2 Vector Processing Element.



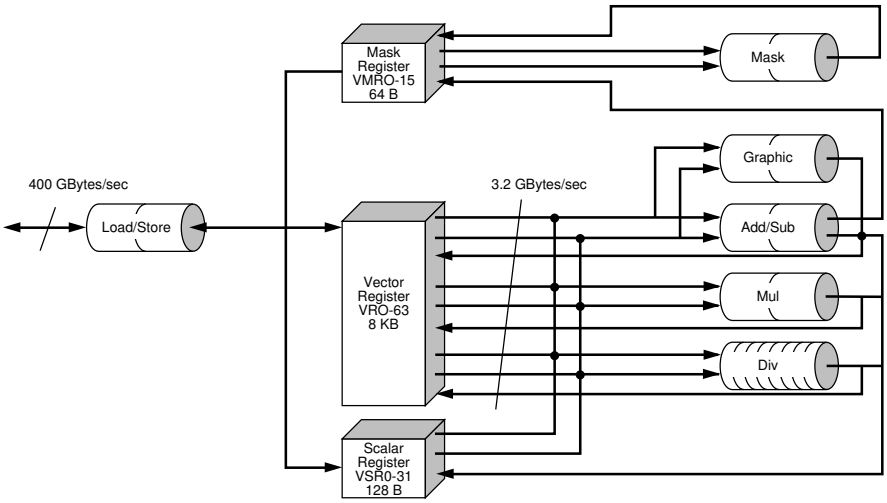
The memory system is implemented as 16 independent banks, with a (current) total capacity of 128 Mbytes. Memory bandwidth for each of the 3 ports is 1.2 Gbytes/s, with a total bandwidth of 3.2 Gbytes/s.

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.

µ*VP Vector Processor*

The µVP operates with a 50 MHz (20 ηs) 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.

Figure 1-2 μVP Vector Processor.



Configuration of the VP 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 separate 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 (64bit) or two single precision (32bit) IEEE format result(s) on every clock, giving a maximum theoretical performance of 100MFLOPS/s double precision and 200MFLOPS/s single precision; the divide pipe can simultaneously deliver an extra 6MFLOPS/s in either single or double precision. Both the add and multiply pipes have the low latency (pipe depth) of two cycles (40ηs), 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 64bit interface which can generate four addresses on consecutive clock cycles before stalling for the returned data. Once the data is present a 64bit word can be transferred on each clock cycle, giving a maximum bandwidth of 400Mbytes/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*.

Superscalar SPARC Processor

The MK403 uses SPARC MBus processor modules. It is generally populated with a 36 or 40MHz 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 16Kbytes data cache occur in a sin-

gle cycle. The floating point unit can execute multiply and add instructions simultaneously, though only one floating point instruction can be issued per cycle.

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 64bits of user data (78bits including error checking and correction, implemented using 20 by 4bit DRAMs with two bits unused). Error detection and correction is implemented on each half word (32bits), allowing write access to 32bit (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 8Kbytes, giving 128Kbytes 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 VP 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 (50MHz), and accesses from the 40MHz 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 (40ηs), and replies with the data four cycles (80ηs) later, giving a bandwidth of 8bytes every two cycles (40ηs), that is 200Mbytes/s. Since there are 16 banks, the total memory system bandwidth is thus 3.2Gbytes/s.

Each µVP can issue a memory request every cycle (20ηs), 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 8bytes every 20ηs, that is 400Mbytes/s.

Since each bank can accept a new address every two cycles (40ηs), but the μVP can generate an address every cycle (20ηs) 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 64bit double precision accesses). Such an access pattern would then see only one half of the normal bandwidth, that is 200Mbytes/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.

Compilers 2

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**, l**oop interchange**, **loop vectorisation**, **vectorisation of intrinsic functions**, **vector idiom recognition**, dead code removal,

as well as proprietary optimisations for the CS–2.

Languages

Fortran, C, High Performance Fortran (HPF) and Fortran–90 are supported.

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 vectorisation directives (for example, CDIR$IVDEP).

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

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. A HPF compiler for the CS–2 is currently under development.

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.

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.

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 vectoriser 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.

Conclusions *3*

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 20ηs, performance peaks at 200MFLOPS/s per processing element in 64bit arithmetic, or 400MFLOPS/s in 32bit.

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 1.2Gbytes/s on direct, strided, or indirect addressing. Memory capacity is currently 32 or 128Mbytes 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.