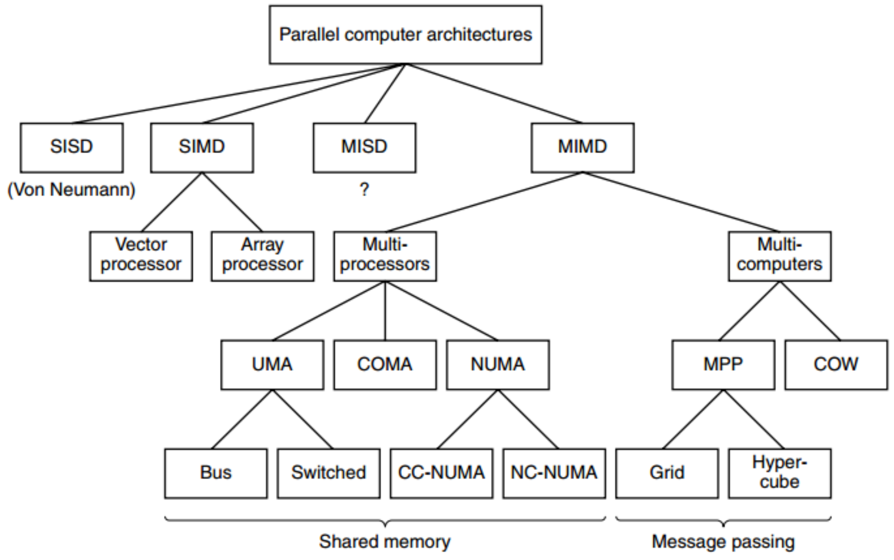
Architectures for Data-Level Parallelism

Programmers who care about performance must become parallel programmers, for sequential code now means slow code.

1

Flynn Taxonomy



2

Number of instruction streams and the number of data streams (Flynn)

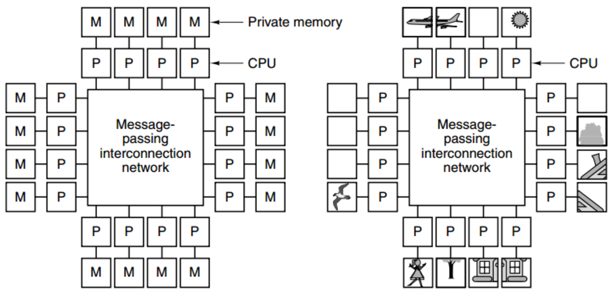
		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Core i7

MISD processor might be a “stream processor” that would perform a series of computations on a single data stream in a pipelined fashion.
SIMD computers operate on vectors of data with a single operation.

Single Program Multiple Data (SPMD)
The normal way to program a MIMD computer is to write a single program that runs on all processors of a MIMD computer, relying on conditional statements when different processors should execute distinct sections of code.

3

Multi-Computer (Distributed Shared Memory)


M
P
Private memory
CPU
Message-passing interconnection network
Message-passing interconnection network
CPU

- Private Memory
- SEND/RECEIVE instruction for communication
- More copy of OS
- More Page Table and Process Table

4

Consideration

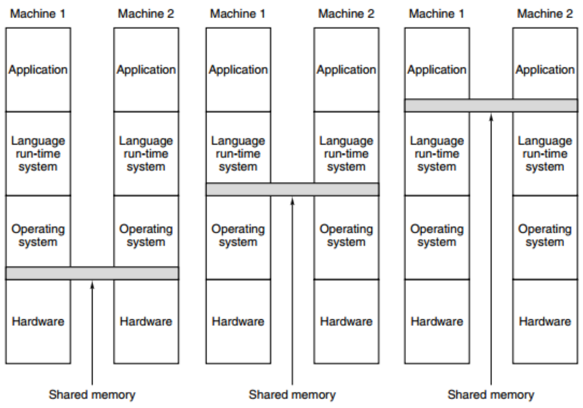
Multiprocess:

- PRO: easy to program
- CONS: hard to build

Multi-Computer:

- PRO: easy to build
- CONS: hard to program

Hybrid System



5

Vector Architecture

- Vector architectures grab sets of data elements scattered about memory, place them into large, sequential register files, operate on data in those register files, and then disperse the results back into memory.
- A single instruction operates on vectors of data,
  - dozens of register–register operations on independent data elements.
- Hide memory latency and leverage memory bandwidth.
  - Since vector loads and stores are deeply pipelined,
  - The processor pays the long memory latency only once per vector load or store versus once per element.

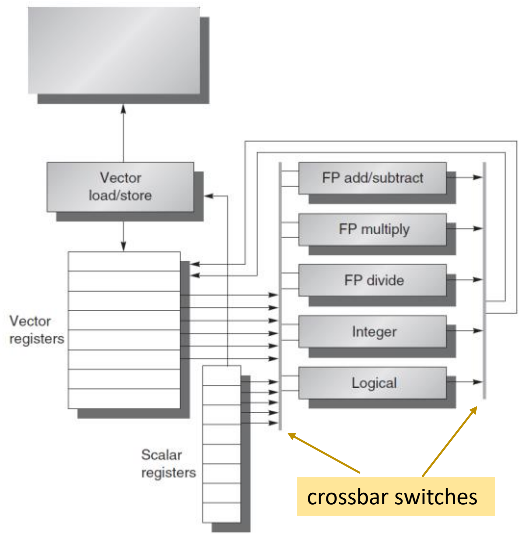
6

Vector architecture

There are also 32 vector registers, and all the functional units are vector functional units.

The vector and scalar registers have a significant number of read and write ports to allow multiple simultaneous vector operations.

The read and write ports, which total at least 16 read ports and 8 write ports, are connected to the functional unit inputs or outputs by a pair of crossbar switches.



7

Vector istructions

Instruction	Operands	Function
ADDVV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2, then put each result in V1.
SUBVV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULVV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.

8

How Vector Processors Work: An Example (I)

Let’s take a typical vector problem:

Y = a × X + Y

X and Y are vectors, initially resident in memory, and a is a scalar.

Scalar solution

```
       L.D F0,a          ;load scalar a
       DADDIU R4,Rx,#512 ;last address to load
Loop:  L.D F2,0(Rx)      ;load X[i]
       MUL.D F2,F2,F0 ;a × X[i]
       L.D F4,0(Ry)      ;load Y[i]
       ADD.D F4,F4,F2 ;a × X[i] + Y[i]
       S.D F4,9(Ry)      ;store into Y[i]
       DADDIU Rx,Rx,#8 ;increment index to X
       DADDIU Ry,Ry,#8 ;increment index to Y
       DSUBU R20,R4,Rx ;compute bound
       BNEZ R20,Loop ;check if done
```

The solution is based on the execution of 600 instructions for MIPS.

How Vector Processors Work: An Example (II)

Let’s take a typical vector problem:

Y = a × X + Y

X and Y are vectors, initially resident in memory, and a is a scalar.

Vector solution:

```
L.D F0,a          ;load scalar a
LV V1,Rx          ;load vector X
MULVS.D V2,V1,F0       ;vector-scalar multiply
LV V3,Ry          ;load vector Y
ADDVV.D V4,V2,V3 ;add
SV V4,Ry          ;store the result
```

The solution is based on the execution of 6 instructions for a vector processor.

How Vector Processors Work: An Example (III)

- Vector processor reduces the dynamic instruction bandwidth, executing only 6 instructions versus almost 600 for MIPS.
  - Vector operations work on 64 elements
  - Loop on MIPS is not present in vector code.

  Loops can be vectorized when they do not have dependences between iterations of a loop (*loop-carried dependences)*.

- Each vector instruction will only stall for the first element in each vector, and then subsequent elements will flow smoothly down the pipeline.

11

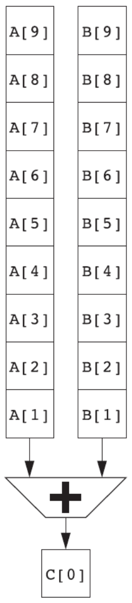
Architecture

Cray

12

Sequential solution (pipeline solution)

- The vector processor has a single add pipeline and can complete one addition per cycle:
  - Hardware cycle
  - All in an instruction



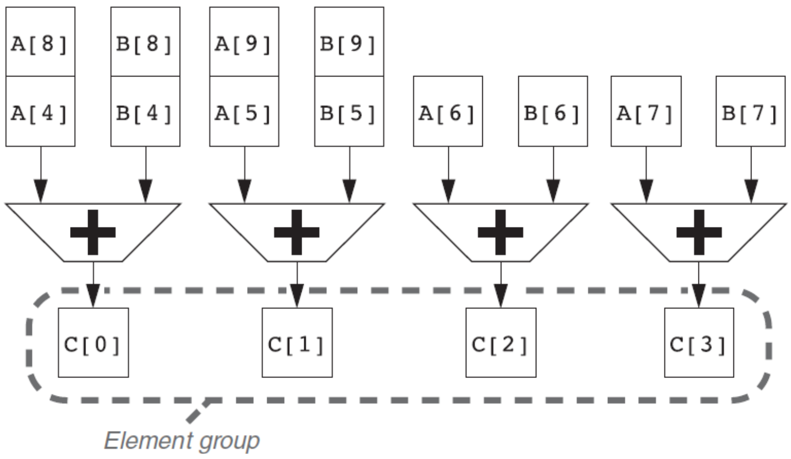
13

Partial parallel solution (parallel pipeline solution)

The vector has four add pipelines and can complete four additions per cycle.

The elements within a single vector add instruction are interleaved across the four pipelines.

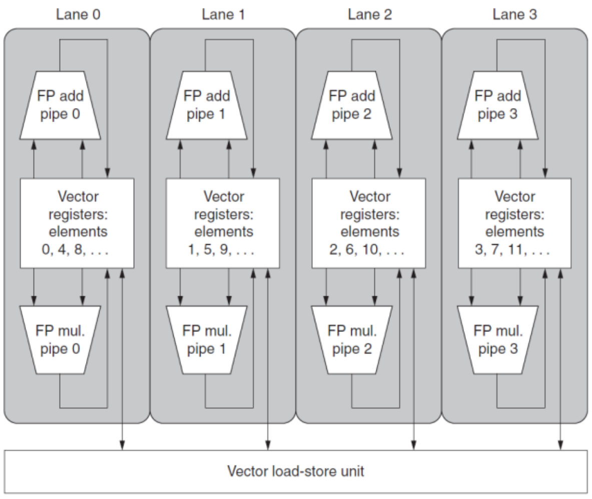
The set of elements that move through the pipelines together is termed an element group.



14

Structure of a vector unit containing four lanes

- The vector register storage is divided across the lanes, with each lane holding every fourth element of each vector register.
- The figure shows three vector functional units: an FP add, an FP multiply, and a load-store unit.
- Each of the vector arithmetic units contains four execution pipelines, one per lane, which act in concert to complete a single vector instruction.
- Note how each section of the vector register file only needs to provide enough ports for pipelines local to its lane.


Lane 0
Lane 1
Lane 2
Lane 3
FP add pipe 0
FP add pipe 1
FP add pipe 2
FP add pipe 3
Vector registers: elements 0, 4, 8, . . .
Vector registers: elements 1, 5, 9, . . .
Vector registers: elements 2, 6, 10, . . .
Vector registers: elements 3, 7, 11, . . .
FP mul. pipe 0
FP mul. pipe 1
FP mul. pipe 2
FP mul. pipe 3
Vector load-store unit

15

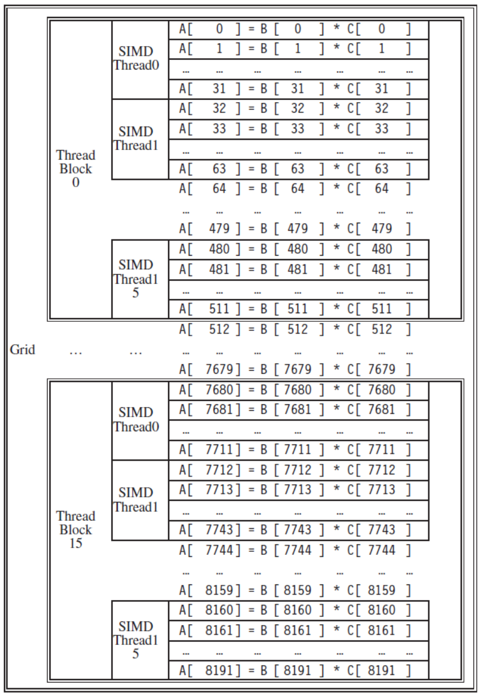
The mapping of a Grid (vectorizable loop)

Thread Blocks (SIMD basic blocks), and threads of SIMD instructions to a vector–vector multiply, with each vector being 8192 elements long.

Each thread of SIMD instructions calculates 32 elements per instruction, and in this example each Thread Block contains 16 threads of SIMD instructions and the Grid contains 16 Thread Blocks.

The hardware Thread Block Scheduler assigns Thread Blocks to
multithreaded SIMD Processors and the hardware Thread Scheduler picks which thread of SIMD instructions to run each clock cycle within a SIMD Processor.

Only SIMD Threads in the same Thread Block can communicate via Local Memory.



```
A[    0 ] = B [    0 ] * C[    0 ]
A[    1 ] = B [    1 ] * C[    1 ]
 …   …   …   …   …   …   …
A[   31 ] = B [   31 ] * C[   31 ]
A[   32 ] = B [   32 ] * C[   32 ]
A[   33 ] = B [   33 ] * C[   33 ]
 …   …   …   …   …   …   …
A[   63 ] = B [   63 ] * C[   63 ]
A[   64 ] = B [   64 ] * C[   64 ]
 …   …   …   …   …   …   …
A[  479 ] = B [  479 ] * C[  479 ]
A[  480 ] = B [  480 ] * C[  480 ]
A[  481 ] = B [  481 ] * C[  481 ]
 …   …   …   …   …   …   …
A[  511 ] = B [  511 ] * C[  511 ]
A[  512 ] = B [  512 ] * C[  512 ]
 …   …   …   …   …   …   …
A[ 7679 ] = B [ 7679 ] * C[ 7679 ]
A[ 7680 ] = B [ 7680 ] * C[ 7680 ]
A[ 7681 ] = B [ 7681 ] * C[ 7681 ]
 …   …   …   …   …   …   …
A[ 7711 ] = B [ 7711 ] * C[ 7711 ]
A[ 7712 ] = B [ 7712 ] * C[ 7712 ]
A[ 7713 ] = B [ 7713 ] * C[ 7713 ]
 …   …   …   …   …   …   …
A[ 7743 ] = B [ 7743 ] * C[ 7743 ]
A[ 7744 ] = B [ 7744 ] * C[ 7744 ]
 …   …   …   …   …   …   …
A[ 8159 ] = B [ 8159 ] * C[ 8159 ]
A[ 8160 ] = B [ 8160 ] * C[ 8160 ]
A[ 8161 ] = B [ 8161 ] * C[ 8161 ]
 …   …   …   …   …   …   …
A[ 8191 ] = B [ 8191 ] * C[ 8191 ]
```

16

Main performance problems

- Latency in Load/Store
  - Memory organized in banks
- Low frequency of instructions on vectors
  - Performance is due to the scalar part of the algorithm
- The presence of vector instructions changes the location of the programs
  - program size and loop reduction
  - location of access to vector data
- Many operations on large sparse matrices
  - The algorithms try to save memory

17