

# Parallel Processing SIMD, Vector and GPU's

EECS4201      Fall 2016      York University

{ 1 }

---

---

---

---

---

---

---

---

## Introduction

- Vector and array processors
- Chaining
- GPU

{ 2 }

---

---

---

---

---

---

---

---

## Flynn's taxonomy

- **SISD**: Single instruction operating on Single Data
- **SIMD**: Single instruction operating on Multiple Data
- **MISD**: Multiple instruction operating on Single Data
- **MIMD**: Multiple instructions operating on Multiple Data

{ 3 }

---

---

---

---

---

---

---

---

## SIMD

- SIMD architectures can exploit significant data-level parallelism for:
  - matrix-oriented scientific computing
  - media-oriented image and sound processors
- SIMD is more energy efficient than MIMD
  - Only needs to fetch one instruction per data operation
  - Makes SIMD attractive for personal mobile devices
- SIMD allows programmer to continue to think sequentially

{ 4 }

---

---

---

---

---

---

---

---

## Vector vs. Array Processors

- **Array processors** same instruction operating on many data elements at the same time (space)
- **Vector processors** Same instruction operating on many data in a pipeline fashion (what is the difference between this and regular pipelined processors?)

{ 5 }

---

---

---

---

---

---

---

---

## Vector Processors

- Cray-1 was the first commercially successful vector processor

{ 6 }

---

---

---

---

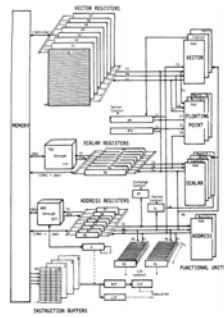
---

---

---

---

# Cray-1



{ 7 }

---

---

---

---

---

---

---

---

# VMIPS

- Example architecture: VMIPS
  - Loosely based on Cray-1
  - Vector registers
    - Each register holds a 64-element, 64 bits/element vector
    - Register file has 16 read ports and 8 write ports
  - Vector functional units
    - Fully pipelined
    - Data and control hazards are detected
  - Vector load-store unit
    - Fully pipelined
    - One word per clock cycle after initial latency
  - Scalar registers
    - 32 general-purpose registers
    - 32 floating-point registers

{ 8 }

---

---

---

---

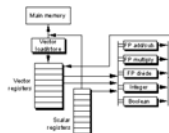
---

---

---

---

# VMIPS



{ 9 }

---

---

---

---

---

---

---

---

## VMIPS Instructions

- ADDVV.D V1,V2,V3 add two vectors
- ADDVS.D V1,V2,F0 add vector to a scalar
- LV V1,R1 vector load from address
- SV R1,V1 Vector store at R1
- MULVV.D V1,V2,V3 vector multiply
- DIVVV.D V1,V2,V3 Vector div (element by element)
- LVWS V1,(R1,R2) Load vector from R1, stride=R2
- LVI V1,(R1+V2) Load V1 with elements at R1+V2(i)
- CVI V1,R1 create an index vector in V1 (0, R1, 2R1,3R1,...)
- SEQVV.D V1,V2 Compare elements V1,V2 0 or 1 in VM EQ, NE, GT, ...
- MVTM VM,F0 Move contents of F0 to vec. mask reg.
- MTCI VLR,R1 Move r1 to vector length register

{ 10 }

---

---

---

---

---

---

---

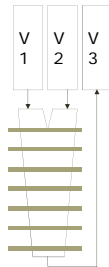
---

---

---

## Vector Processing

- ADDV V3, V1, V2
- After an initial latency (depth of pipeline) we get one result per cycle.
- We can do this with a simple loop, what is the difference?



{ 11 }

---

---

---

---

---

---

---

---

---

---

## Vector Execution Time

- Execution time depends on three factors:
  - Length of operand vectors
  - Structural hazards
  - Data dependencies
- VMIPS functional units consume one element per clock cycle
  - Execution time is approximately the vector length
- Convey
  - Set of vector instructions that could potentially execute together (could be more than one instruction)

{ 12 }

---

---

---

---

---

---

---

---

---

---

## Chimes

- Sequences with read-after-write dependency hazards can be in the same convey via *chaining*
- Chaining**
  - Allows a vector operation to start as soon as the individual elements of its vector source operand become available
- Chime**
  - Unit of time to execute one convey
  - $m$  conveys executes in  $m$  chimes
  - For vector length of  $n$ , requires  $m \times n$  clock cycles

13

---

---

---

---

---

---

---

---

---

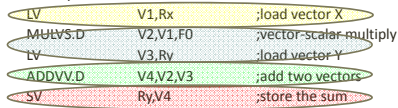
---

## Example

```

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 two vectors
SV      Ry,V4      ;store the sum
    
```

Conveys:



4 conveys => 4 x 64 » 256 clocks (or 4 clocks per result)

14

---

---

---

---

---

---

---

---

---

---

## Example

- Consider the following example:
- For (i=0;i<50.i++)
  - $c[i] = (a[i] + b[i])/2$
- Sequence of improvements from in order execution with one bank to chained vector processor with multiple banks

15

---

---

---

---

---

---

---

---

---

---

### Assembly Code

- Initialize registers R0, R1, R2, R3
- LOOP LD R4, 0(R1) 11
- LD R5, 0(R2) 11
- ADD R6, R4, R5 4
- SR R6, R6, 1 1
- ST R6, 0(R3) 11
- ADDI R1, R1, 4 1
- ADDI R2, R2, 4 1
- ADDI R3, R3, 4 1
- ADDI R0, R0, -1 1
- BEQZ R0, LOOP 2 = 44\*50

{ 16 }

---

---

---

---

---

---

---

---

---

---

### Vector Code

- The loop is vectorizable
- Initialize registers (including V\_length and stride) 5+5 dynamic instruction
- LV V1, R1 11+50-1
- LV V2, R2 11+50-1
- ADDV V3, V1, V2 4+50-1
- SLV V3, V3, 1 1+50-1
- SV V3, R4 11+50-1 =293

{ 17 }

---

---

---

---

---

---

---

---

---

---

### Vector Code

- Chaining: No need to wait until the vector register is loaded, you can start after the first element is ready.
- How long Does it takes for the previous case?
- 

{ 18 }

---

---

---

---

---

---

---

---

---

---

## Vector Code

- Chaining and 2 memory banks?

{ 19 }

---

---

---

---

---

---

---

---

## Vector length

- In the previous example, the vector length is less than the VREG length.
- What if more (operation on a vector of 1000 elements)
- Loops each load perform on a 64 element vector (need to adjust vector length in the last iteration)

{ 20 }

---

---

---

---

---

---

---

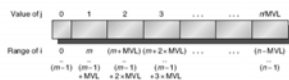
---

## Vector Stripmining

- Vector length not known at compile time?
- Use Vector Length Register (VLR)
- Use strip mining for vectors over the maximum length:

```

low = 0;
VL = (n % MVL); /*find odd-size piece using modulo op % */
for (j = 0; j <= (n/MVL); jj+=1) { /*outer loop*/
  for (i = low; i < (low+VL); ii+=1) /*runs for length VL*/
    Y[i] = a * X[i] + Y[i]; /*main operation*/
  low = low + VL; /*start of next vector*/
  VL = MVL; /*reset the length to maximum vector length*/
}
    
```



{ 21 }

---

---

---

---

---

---

---

---

## Effect of Memory

- Load/store unit is more complicated than FU's
- Start-up time, is the time for the first word into a register
- Memory system must be designed to support high bandwidth for vector loads and stores
- Spread accesses across multiple banks
  - Control bank addresses independently
  - Load or store non sequential words
  - Support multiple vector processors sharing the same memory
- Example:
  - 32 processors, each generating 4 loads and 2 stores/cycle
  - Processor cycle time is 2.167 ns, SRAM cycle time is 15 ns
  - How many memory banks needed?

[ 22 ]

---

---

---

---

---

---

---

---

---

---

## Example

- Cray T932 has 32 processors. Each processor is capable of generating 4 loads and 2 stores per clock cycle.
- Clock cycle is 2.167 ns, SRAM cycle time 15 ns. How many bank do we need to allow the system to run at a full memory bandwidth?

[ 23 ]

---

---

---

---

---

---

---

---

---

---

## Example

- 8 memory banks, bank busy time 6 cycles, total memory latency 12 cycles.
- What is the difference between a 64-element vector load with a stride of 1 and 32?

[ 24 ]

---

---

---

---

---

---

---

---

---

---



## Stride

- Consider:

```

for (i = 0; i < 100; i=i+1)
  for (j = 0; j < 100; j=j+1) {
    A[i][j] = 0.0;
    for (k = 0; k < 100; k=k+1)
      A[i][j] = A[i][j] + B[i][k] * D[k][j];
  }
    
```

- Must vectorize multiplication of rows of B with columns of D
- Use *non-unit stride*
- Bank conflict (stall) occurs when the same bank is hit faster than bank busy time:
  - #banks / LCM(stride,#banks) < bank busy time

25

---

---

---

---

---

---

---

---

---

---

## Strides

Add in a bank					SE	Q		M	O	D
	0	1	2	3	0	1	2	0	1	2
0	0	1	2	3	0	1	2	0	16	8
1	4	5	6	7	3	4	5	9	1	17
2	8	9	10	11	6	7	8	18	10	2
3	12	13	14	15	9	10	11	3	19	11
4	16	17	18	19	12	13	14	12	4	20
5	20	21	22	23	15	16	17	21	13	5
6	24	25	26	27	18	19	20	6	22	14
7	28	29	30	31	21	22	23	15	7	23

26

---

---

---

---

---

---

---

---

---

---

## Strides

- MOD can be calculated very efficiently if the prime number is 1 less than a power of 2.
- Division still a problem
- But if we change the mapping such that
- Address in a bank = address MOD number of words in a bank.
- Since the number of words in a bank is usually a power of 2, that will lead to a very efficient implementation.
- Consider the following example, the first case is the usual 4 banks, then 3 banks with sequential interleaving and modulo interleaving and notice the conflict free access to rows and columns of a 4 by 4 matrix

27

---

---

---

---

---

---

---

---

---

---

## Vector Mask Register

- What if we have a conditional IF statement inside the loop?
- Using scalar architecture, that introduces control dependence.
- The *vector-mask control*: A mask register is used to conditionally execute using a Boolean condition.
- When the *vector-mask register* is enabled, any vector instruction executed operate only on vector elements whose corresponding entries in the VMR are ones.
- The rest of the elements are unaffected.
- Clearing the vector mask register, sets to all 1's and operations are performed on all the elements.
- Does not save execution time for masked elements

28

---

---

---

---

---

---

---

---

---

---

## Vector Mask Register

- Consider:  

```
for (i = 0; i < 64; i=i+1)
  if (X[i] != 0)
    X[i] = X[i] - Y[i];
```
- Use vector mask register to “disable” elements:  

LV	V1,Rx	;load vector X into V1
LV	V2,Ry	;load vector Y
L.D	F0,#0	;load FP zero into F0
SNEVS.D	V1,F0	;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D	V1,V1,V2	;subtract under vector mask
SV	Rx,V1	;store the result in X

29

---

---

---

---

---

---

---

---

---

---

## Strides

- MOD can be calculated very efficiently if the prime number is 1 less than a power of 2.
- Division still a problem
- But if we change the mapping such that
- Address in a bank = address MOD number of words in a bank.
- Since the number of words in a bank is usually a power of 2, that will lead to a very efficient implementation.
- Consider the following example, the first case is the usual 4 banks, then 3 banks with sequential interleaving and modulo interleaving and notice the conflict free access to rows and columns of a 4 by 4 matrix

30

---

---

---

---

---

---

---

---

---

---

## Scatter-Gather

- Consider:  
for ( $i = 0; i < n; i=i+1$ )  
     $A[K[i]] = A[K[i]] + C[M[i]];$

- Use index vector:

```

LV      Vk, Rk      ;load K
LVI     Va, (Ra+Vk) ;load A[K[]]
LV      Vm, Rm      ;load M
LVI     Vc, (Rc+Vm) ;load C[M[]]
ADDVV.D Va, Va, Vc  ;add them
SVI     (Ra+Vk), Va ;store A[K[]]
    
```

31

---

---

---

---

---

---

---

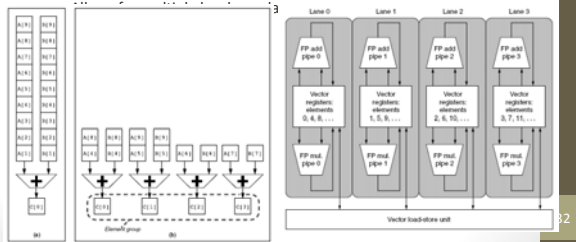
---

---

---

## Multiple lanes

- Element  $n$  of vector register  $A$  is "hardwired" to element  $n$  of vector register  $B$



32

---

---

---

---

---

---

---

---

---

---

## Not Quite SIMD

- Intel extension MMx, SSE, AVX, PowerPC AltiVec, ARM Advanced SIMD
- No vector length, just depends on the instruction, the register can be considered 16 8-bit numbers, 8 16-bit numbers, ...

33

---

---

---

---

---

---

---

---

---

---