COMP4211 :
Advance Computer Architecture

Vector Processor

Overview

- Introduction: What and Why?
- Basic Vector Architecture
- Example: MIPS Vs VMIPS
- Parallelism using convoys
- Vector Memory Systems
- Real World Issues:
  - Vector Length
  - Stride
- Introduction into Cray-1

Introduction

What is a Vector Processor?

- Consider an operation D = A +C
- Vector processor provides high-level operations that work on vectors.
- A typical instruction might add two 64 element FP vectors.
- Commercialized long before ILP machines.

Introduction cont.

Why Vector Processors?

- It is equivalent to executing an entire loop
  - Reducing instruction fetch and decode bandwidth.
- Each instruction guarantees each result is independent on other results in same vector
  - No data hazard check needed in an instruction.
  - Executed using array of paralleled functional units, or deep pipeline.

Introduction cont.

- Hardware need only check for data hazards between two instructions, once per operand.
  - More instructions per data check.
- Memory access for entire vector, not a single word.
  - Reduced Latency
- Multiple vector instructions in progress.
  - Further parallelism

Basic Vector Architecture

- Ordinary scalar pipeline unit + Vector unit.
- Two Types –
  - Vector-register -> all operations except load and store based on registers.
  - Memory-memory -> all operations are memory to memory.
- Concentrate on Vector-register, particularly VMIPS architecture.

BVA ñ the components

Vector register
- Fixed length, holds a single vector
- In VMIPS
  - 2 read and 1 write port.
  - 8 vector registers, 64 elements each

Vector functional units
- Fully pipelined, start new operations every cycle.
- Might contain scalar function unit.

Control unit
- Detect structural and data hazards.

BVA ñ the components cont.

- Vector load-store unit
  - Loads and stores vector to and from memory.
- Special-purpose registers
  - Vector length
  - Vector mask registers
- Set of Scalar registers
  - Provide data as input to the vector functional units.
  - Compute addresses to pass to the Load-Store unit.
  - In VMIPS
    - 32 general purpose and 32 floating-point registers.

Example: MIPS Vs VMIPS

Example Show the code for MIPS and VMIPS for the DAXPY loop. Assume that the starting addresses of X and Y are in Rx and Ry, respectively.

Answer Here is the MIPS code.

```
        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     0(Ry),F4    ;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
```

Here is the VMIPS code for DAXPY.

```
        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
        ADDV.D  V4,V2,V3    ;add
        SV      Ry,V4       ;store the result
```

- Greatly reduced instruction bandwidth
  - Six instructions instead of 600.

Parallelism using convoys

Convoys

- A set of instructions that could begin execution together.
- Consider this sequence of code.

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

ï Using Convoys, results in

```
1. LV
2. MULVS.D    LV
3. ADDV.D
4. SV
```


Vector Memory Systems

- Problem
  - Memory system needs to be able to produce and accept large amounts of data.
  - But how do we achieve this when there is poor access time?
- Solution
  - Creating multiple memory banks.
    - Useful for fragmented accesses.
    - Support multiple loads per clock cycle.
    - Allows for multi-processor sharing.

Vector Memory System

Example

Suppose we want to fetch a vector of 64 elements starting at byte address 136, and a memory access takes 6 clocks. How many memory banks must we have to support one fetch per clock cycle? With what addresses are the banks accessed? When will the various elements arrive at the CPU?

Answer Six clocks per access require at least six banks, but because we want the number of banks to be a power of two, we choose to have eight banks. Figure G.7 shows the timing for the first few sets of accesses for an eight-bank system with a 6-clock-cycle access latency.

	Bank							
Cycle no.	0	1	2	3	4	5	6	7
0		136						
1		busy	144					
2		busy	busy	152				
3		busy	busy	busy	160			
4		busy	busy	busy	busy	168		
5		busy	busy	busy	busy	busy	176	
6			busy	busy	busy	busy	busy	184
7	192			busy	busy	busy	busy	busy
8	busy	200			busy	busy	busy	busy
9	busy	busy	208			busy	busy	busy
10	busy	busy	busy	216			busy	busy
11	busy	busy	busy	busy	224			busy
12	busy	busy	busy	busy	busy	232		
13		busy	busy	busy	busy	busy	240	
14			busy	busy	busy	busy	busy	248
15	256			busy	busy	busy	busy	busy
16	busy	264			busy	busy	busy	busy

Real World Issues (1)

Vector – Length Control

- Problem
  - How do we support operations where the length is unknown or not the vector length?
- Solution
  - Provide a vector-length register, solves problem only if real length is less than Maximum Vector Length.
  - Use Technique Called strip mining.

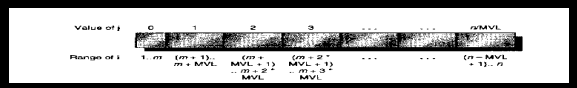
Strip mining

- Generating code where vector operations are done for a size no greater than MVL.
- Create 2 loops
  - One that handles any number of iterations multiple of MVL.
  - Another that handles the remaining iterations.
- Code becomes vectorizable.
- Careful handling of VLR needed.

Example: Strip Mining

- **For the DAXPY loop, a we can generate a C code as below.**

```
low=1;  /*Assume start element at 1*/
vL = n % mvL; /*find the odd – size piece */
for(j=0; j<=n / mvL; j++){  /*Outer Loop*/
        for(i=low; i<=low+vL-1;i++){ /*Inner loop-runs for length vL*/
        y[i] = a*x[i] + y[i];   /*Start of next vector*/
        }
        low = low + vL;    /*Find start of next vector*/
        vL = mvL;          /* reset length to max */
```



Real World Issues (2)

Vector Stride

- *Problem*
  - Position in memory of adjacent elements in may not be sequential. Set up time could be enormous.
  - E.g. Matrix Multiplication.
- *Solution*
  - Distance seperating elements is called *the Stride.*
  - Store the stride in a register, so only a single load or store is required.

Vector Stride

Access time

- Vector processors use interleave memory banks. Non-unit Strides can cause stalls.
- Stall will occur if

No. of banks /LCM (Stride, No. of Banks)

<

Bank Busy time

- No conflicts if Stride and no. of banks are relatively prime.
- Increasing the no. of banks to greater than minimum.
- Most vector supercomputers have at least 64, with some having up to 1024.

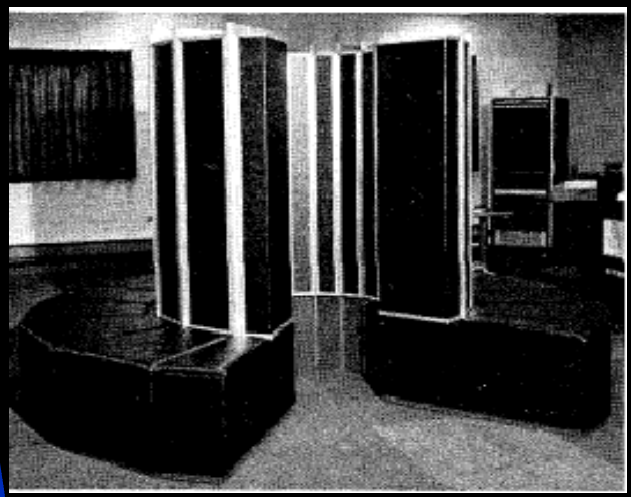
Example-Vector Stride

Example Suppose we have 8 memory banks with a bank busy time of 6 clocks and a total memory latency of 12 cycles. How long will it take to complete a 64-element vector load with a stride of 1? With a stride of 32?

Answer Since the number of banks is larger than the bank busy time, for a stride of 1, the load will take 12 + 64 = 76 clock cycles, or 1.2 clocks per element. The worst possible stride is a value that is a multiple of the number of memory banks, as in this case with a stride of 32 and 8 memory banks. Every access to memory (after the first one) will collide with the previous access and will have to wait for the 6-clock-cycle bank busy time. The total time will be 12 + 1 + 6 * 63 = 391 clock cycles, or 6.1 clocks per element.

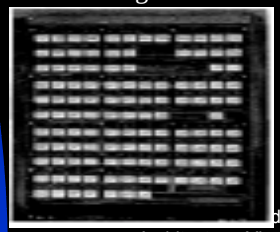
Cray - 1

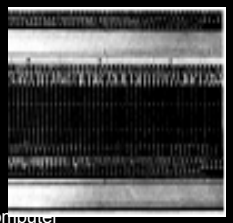
- Most well-known vector processor, released in 1976.
- Fastest super-computer in the late 70s.
- 32 bit instruction length.
- Architecture Consists of 3 sections:
  - The Main Memory
  - The Scalar Subsystem
  - The Vector Subsystem



Cray-1: Main Memory

- 16 banks, each consisting of 72 64K, 64-bit words.
- Cycle time of 50 nSec, which is equivalent to 4 cycles.
- Can transfer 1-4 words per clock period depending on the register or buffer.
- 4 words per clock cycle for instruction buffer, resulting in a bandwidth of 1280mB/sec.





Cray-1: Scalar subsystem

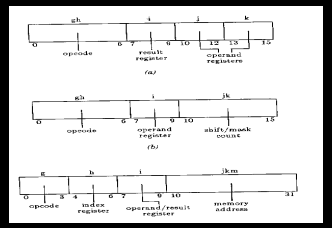
- Consists of
  - Instruction buffers
  - 2 file scalar registers
  - 2 address functional registers
  - Scalar functional unit
  - Shared floating point functional unit

Cray-1: Vector subsystem

- Consist of
  - 8 vector registers
  - Set of 3 vector functional units
  - Shared set of 3 floating point functional units

Cray-1: Instruction Format

- Binary arithmetic and logic instructions (a)
- Unary shift and mask instructions (b)
- Memory read and store instructions (c)
- Branch instructions use lower 24 bit for branch address.



References

- Computer Architecture: A quantitative Approach, Patterson and Hennessy, Appendix G, section 1-3.
- Computer Architecture: A modern Synthesis, Subrata Dasgupta, Chapter 7, P246 – P249.
- http://www.crhc.uiuc.edu/IMPACT/ece412/public_html/Notes/412_lec20/
- The Cray-1 Computer System, Richard M Russell, Cray Research Inc.
- http://csep1.phy.ornl.gov/ca/node24.html