

# **CS422**

# **Computer Architecture**

Spring 2004

**Lecture 33, 22 Apr 2004**

Bhaskaran Raman  
Department of CSE  
IIT Kanpur

<http://web.cse.iitk.ac.in/~cs422/index.html>

# Lecture Outline

- Vector Processors
- 
- *Scribe for today?*

# Why Vector Processing

- Deep pipeline ==> more parallelism
  - But more dependences
  - Need to fetch and issue many instructions (Flynn bottleneck)
- Same issues with multiple-issue processor
- Operations on vectors:
  - No data dependences
  - No control hazards
  - Single instn. ==> instn. bandwidth reduced
  - Well defined memory access pattern

# Basic Architecture

- Vector-register processors vs. memory-memory vector processor
- DLXV: vector extn. of DLX (vector-register)
- Components:
  - Vector registers (V0..V7), 64-element
  - Vector functional units:
    - ADD/SUB, MUL, DIV, Integer, Logical
    - Each is pipelined, can start a new opn. every cycle
  - Vector load/store unit: also pipelined
  - Scalar registers and scalar unit (like in DLX)

# Some Vector Instructions

- ADDV      V1, V2, V3
- ADDSV    V1, F0, V2
- SUBV      V1, V2, V3
- SUBVS    V1, V2, F0
- SUBSV    V1, F0, V2
- Similar for MUL and DIV
- LV            V1, R1
- SV            R1, V1

# SAXPY/DAXPY Loop

- $Y = aX + Y$  (caps ==> vector)

|                 |                    |               |                   |
|-----------------|--------------------|---------------|-------------------|
| <b>LD</b>       | <b>F0, a</b>       | <b>LD</b>     | <b>F0, a</b>      |
| <b>ADDI</b>     | <b>R4, Rx, 512</b> | <b>LV</b>     | <b>V1, Rx</b>     |
| <b>Loop: LD</b> | <b>F2, 0(Rx)</b>   | <b>MULTSV</b> | <b>V2, F0, V1</b> |
| <b>MULTD</b>    | <b>F2, F0, F2</b>  | <b>LV</b>     | <b>V3, Ry</b>     |
| <b>LD</b>       | <b>F4, 0(Ry)</b>   | <b>ADDV</b>   | <b>V4, V2, V3</b> |
| <b>ADDD</b>     | <b>F4, F2, F4</b>  | <b>SV</b>     | <b>Ry, V4</b>     |
| <b>SD</b>       | <b>0(Ry), F4</b>   |               |                   |
| <b>ADDI</b>     | <b>Rx, Rx, 8</b>   |               |                   |
| <b>ADDI</b>     | <b>Ry, Ry, 8</b>   |               |                   |
| <b>SUB</b>      | <b>R20, R4, Rx</b> |               |                   |

Reduction in instn. bandwidth  
Lesser pipeline interlocks

# Estimating Execution Time

- *Convoy*: set of vector instructions which can begin execution in same cycle
  - Check for structural, data hazards
- For simplicity: convoy must complete before initiating next convoy
- *Chime*: time taken to execute one vector opn.
- Approximations:
  - Only one instn. can be initiated per cycle
  - Pipeline setup latency

# Adding Flexibility

- Vector-length register (VLR), Maximum vector length (MVL)
  - MOV12S VLR, R1
  - MOVS2I R1, VLR
- Vector longer than MVL ==> use *strip-mining*
- Vector stride:
  - LVWS V1, (R1, R2)
  - SVWS (R1, R2), V1
- Memory-bank conflicts?

# Enhancing Vector Performance

- Chaining: data-forwarding
- Conditional execution:
  - Vector Mask Register
  - Some related instructions
    - SNEV    V1, V2
    - SGTSV   F0, V1
    - CVM
- Sparse matrices: scatter-gather
  - LVI    V1, (R1+V2)
  - SVI    (R1+V2), V1