CPE 631 Lecture 24: Vector Processing

Aleksandar Milenković, milenka@ece.uah.edu
Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

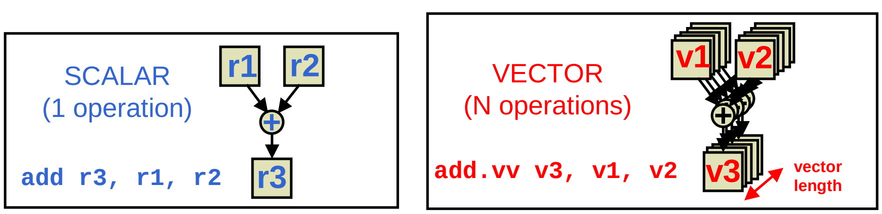
- Properties of Vector Processing
- Components of a Vector Processor
- Vector Execution Time
- Real-World Problems:
  Vector Length and Stride
- Vector Optimizations: Chaining,
  Conditional Execution, Sparse Matrices

Why Vector Processors?

- Instruction level parallelism (Ch 3&4)
  - Deeper pipeline and wider superscalar machines to extract more parallelism
    - more register file ports, more registers, more hazard interlock logic
  - In dynamically scheduled machines instruction window, reorder buffer, rename register files must grow to have enough capacity to keep relevant information about in-flight instructions
- Difficult to build machines supporting large number of in-flight instructions => limit the issue width and pipeline depths => limit the amount parallelism you can extract
- Commercial versions long before ILP machines

Vector Processing Definitions

- Vector - a set of scalar data items, all of the same type, stored in memory
- Vector processor - an ensemble of hardware resources, including vector registers, functional pipelines, processing elements, and register counters for performing vector operations
- Vector processing occurs when arithmetic or logical operations are applied to vectors



CPE 631 ©AM

Properties of Vector Processors

- 1) Single vector instruction specifies lots of work
  - equivalent to executing an entire loop
  - fewer instructions to fetch and decode
- 2) Computation of each result in the vector is independent of the computation of other results in the same vector
  - deep pipeline without data hazards; high clock rate
- 3) Hw checks for data hazards only between vector instructions (once per vector, not per vector element)
- 4) Access memory with known pattern
  - elements are all adjacent in memory => highly interleaved memory banks provides high bandwidth
  - access is initiated for entire vector => high memory latency is amortised (no data caches are needed)
- 5) Control hazards from the loop branches are reduced
  - nonexistent for one vector instruction

CPE 631 ©AM

Properties of Vector Processors (cont'd)

- Vector operations: arithmetic (add, sub, mul, div),memory accesses, effective address calculations
- Multiple vector instructions can be in progress at the same time => more parallelism
- Applications to benefit
  - Large scientific and engineering applications (car crash simulations, weather forecasting, ...)
  - Multimedia applications

CPE
631
©AM

Basic Vector Architectures

- Vector processor:
ordinary pipelined scalar unit + vector unit
- Types of vector processors
  - **Memory-memory** processors:
all vector operations are memory-to-memory (CDC)
  - **Vector-register** processors:
all vector operations except load and store
are among the vector registers
(CRAY-1, CRAY-2, X-MP, Y-MP, NEX SX/2(3), Fujitsu)
    - VMIPS – Vector processor as
an extension of the 5-stage MIPS processor

CPE
631
©AM

Components of a vector-register processor

- Vector Registers: each vector register
is a fixed length bank holding a single vector
  - has at least 2 read and 1 write ports
  - typically 8-32 vector registers, each holding 64-128 64 bit elements
  - VMIPS: 8 vector registers, each holding 64 elements
(16 Rd ports, 8 Wr ports)
- Vector Functional Units (FUs): fully pipelined,
start new operation every clock cycle
  - typically 4 to 8 FUs: FP add, FP mult, FP reciprocal (1/X), integer add, logical, shift;
  - may have multiple of same unit
  - VMIPS: 5 FUs (FP add/sub, FP mul, FP div, FP integer, FP logical)

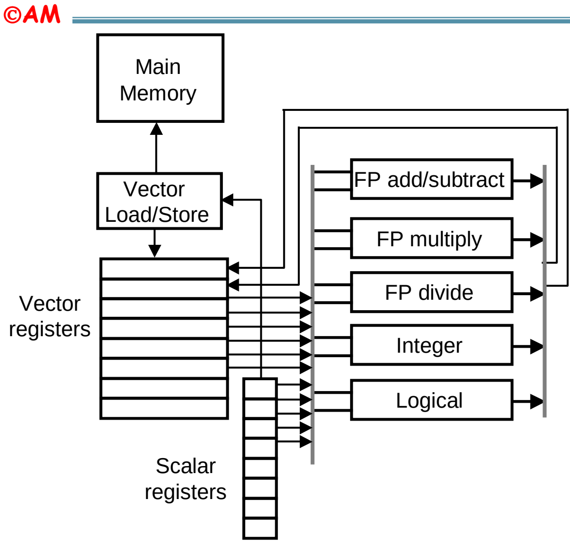
CPE 631 ©AM

Components of a vector-register processor (cont'd)

- Vector Load-Store Units (LSUs)
  – fully pipelined unit to load or store a vector; may have multiple LSUs
  – VMIPS: 1 VLSU, bandwidth is 1 word per cycle after initial delay
- Scalar registers
  – single element for FP scalar or address
  – VMIPS: 32 GPR, 32 FPRs they are read out and latched at one input of the FUs
- Cross-bar to connect FUs, LSUs, registers
  – cross-bar to connect Rd/Wr ports and FUs

CPE 631 ©AM

VMIPS: Basic Structure



- 8 64-element vector registers
- 5 FUs; each unit is fully pipelined, can start a new operation on every clock cycle
- Load/store unit - fully pipelined
- Scalar registers

CPE 631 ©AM

VMIPS Vector Instructions

Instr.	Operands	Operation	Comment
ADDV.D	V1,V2,V3	V1=V2+V3	vector + vector
ADDSV.D	V1,F0,V2	V1=F0+V2	scalar + vector
MULV.D	V1,V2,V3	V1=V2xV3	vector x vector
MULSV.D	V1,F0,V2	V1=F0xV2	scalar x vector
LV	V1,R1	V1=M[R1..R1+63]	load, stride=1
LVWS	V1,R1,R2	V1=M[R1..R1+63*R2]	load, stride=R2
LVI	V1,R1,V2	V1=M[R1+V2(i),i=0..63]	indir.("gather")
SeqV.D	VM,V1,V2	VMASKi = (V1i=V2i)?	comp. setmask
MTC1	VLR,R1	Vec. Len. Reg. = R1	set vector length
MFC1	VM,R1	R1 = Vec. Mask	set vector mask

See table G3 for the VMIPS vector instructions.

CPE 631 ©AM

VMIPS Vector Instructions (cont'd)

Instr.	Operands	Operation	Comment
SUBV.D	V1,V2,V3	V1=V2-V3	vector - vector
SUBSV.D	V1,F0,V2	V1=F0-V2	scalar – vector
SUBVS.D	V1,V2,F0	V1=V2- F0	vector - scalar
DIVV.D	V1,V2,V3	V1=V2/V3	vector / vector
DIVSV.D	V1,F0,V2	V1=F0/V2	scalar / vector
DIVVS.D	V1,V2,F0	V1=V2/F0	vector / scalar
..			
POP	R1, M	Count the 1s in the VM register	
CVM		Set the vector-mask register to all 1s	

See table G3 for the VMIPS vector instructions.

CPE 631 ©AM

DAXPY: Double a×X + Y

Assuming vectors X, Y are length 64

Scalar vs. Vector

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

```
      L.D     F0,a
      DADDIU  R4,Rx,#512  ;last address to load
loop: L.D     F2, 0(Rx)   ;load X(i)
      MULT.D  F2,F0,F2    ;a*X(i)
      L.D     F4, 0(Ry)   ;load Y(i)
      ADD.D   F4,F2,F4    ;a*X(i) + Y(i)
      S.D     F4,0(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
```

Operations: 578 (2+9*64) vs. 321 (1+5*64) (1.8X)

Instructions: 578 (2+9*64) vs. 6 instructions (96X)

Hazards: 64X fewer pipeline hazards

CPE 631 ©AM

Vector Execution Time

- Time = f(vector length, data dependencies, structural hazards)
- Initiation rate: rate at which a FU consumes vector elements (= number of lanes; usually 1 or 2)
- Convoy: set of vector instructions that can begin execution in same clock (no structural or data hazards)
- Chime: approximate time to execute a convoy
- m convoys take m chimes; if each vector length is n, then they take approx. m x n clock cycles (ignores overhead; good approximation for long vectors)

```
1: LV        V1,Rx     ;load vector X
2: MULVS.D   V2, V1,F0 ;vector-scalar mult.
   LV        V3,Ry     ;load vector Y
3: ADDV.D    V4,V2,V3  ;add
4: SV        Ry,V4     ;store the result
```

4 convoys, 1 lane, VL=64
=> 4 x 64 = 256 clocks
(or 4 clocks per result)

CPE 631 ©AM

VMIPS Start-up Time

- Start-up time: pipeline latency time (depth of FU pipeline); another sources of overhead

Operation	Start-up penalty (from CRAY-1)
Vector load/store	12
Vector multiply	7
Vector add	6

Assume convoys don't overlap; vector length = n:

Convoy	Start	1st result	last result	
1. LV	0	12	11+n (12-1+n)	
2. MULVS.D, LV	12+n	12+n+12	23+2n	load start-up
3. ADDV.D	24+2n	24+2n+6	29+3n	wait convoy 2
4. SV	30+3n	30+3n+12	41+4n	wait convoy 3

CPE 631 ©AM

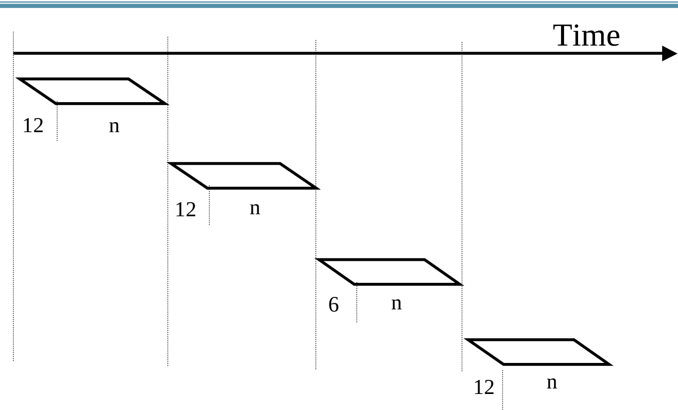
VMIPS Execution Time

1: LV V1,Rx

2: MULV V2,F0,V1
LV V3,Ry

3: ADDV V4,V2,V3

4: SV Ry,V4



CPE 631 ©AM

Vector Load/Store Units & Memories

- Start-up overheads usually longer for LSUs
- Memory system must sustain (# lanes x word) /clock cycle
- Many Vector Processors use banks (vs. simple interleaving):
  - support multiple loads/stores per cycle => multiple banks & address banks independently
  - support non-sequential accesses
- Note: No. memory banks > memory latency to avoid stalls
  - m banks => m words per memory latency l clocks
  - if m < l, then gap in memory pipeline
  - may have 1024 banks in SRAM

CPE 631 ©AM

Real-World Issues: Vector Length

- What to do when vector length is not exactly 64?

```
for(i=0; i<n, i++)
  {Y(i)=a*X(i)+Y(i)}
```

- Value of n can be unknown at compile time?
- Vector-Length Register (VLR): controls the length of any vector operation, including a vector load or store (cannot be > the length of vector registers)
- What if n > Maximum Vector Length (MVL)? => Strip mining

CPE
631
©AM

Strip Mining

- Strip mining: generation of code such that each vector operation is done for a size less than or equal to the MVL
- 1st loop: do short piece (n mod MVL), rest VL = MVL

```
i = 0;
VL = n mod MVL;
for (j=0; j<n/MVL; j++){
     for(i<VL; i++)
       {Y(i)=a*X(i)+Y(i)}
     VL = MVL;
}
```

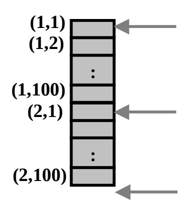
- Overhead of executing strip-mined loop?

CPE
631
©AM

Vector Stride

- Suppose adjacent elements not sequential in memory (e.g., matrix multiplication)

```
for(i=0; i<100; i++)
  for(j=0; j<100; j++) {
    A(i,j)=0.0;
    for(k=0; k<100; k++)
      A(i,j)=A(i,j)+B(i,k)*C(k,j);
  }
```



- Matrix C accesses are not adjacent (800 bytes between)
- Stride: distance separating elements that are to be merged into a single vector
  => LVWS (load vector with stride) instruction
- Strides can cause bank conflicts (e.g., stride=32 and 16 banks)

CPE 631 ©AM

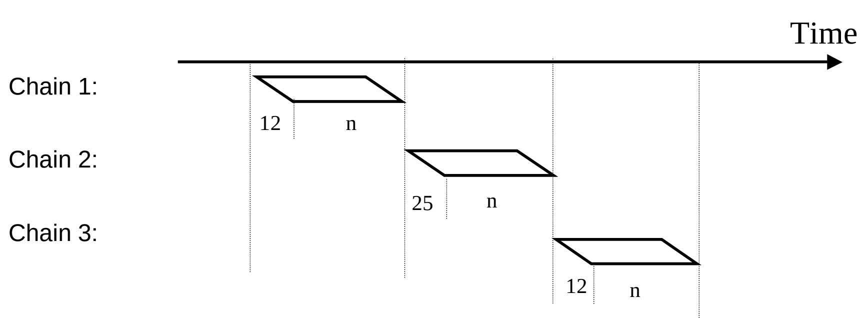
Vector Opt #1: Chaining

- Suppose:
  MULV.D V1,V2,V3
  ADDV.D V4,V1,V5 ; separate convoy?
- Chaining: if vector register (V1) is not treated as a single entity but as a group of individual registers, then pipeline forwarding can work on individual elements of a vector
- Flexible chaining: allow vector to chain to any other active vector operation => more read/write ports
- As long as enough HW, increases convoy size

CPE 631 ©AM

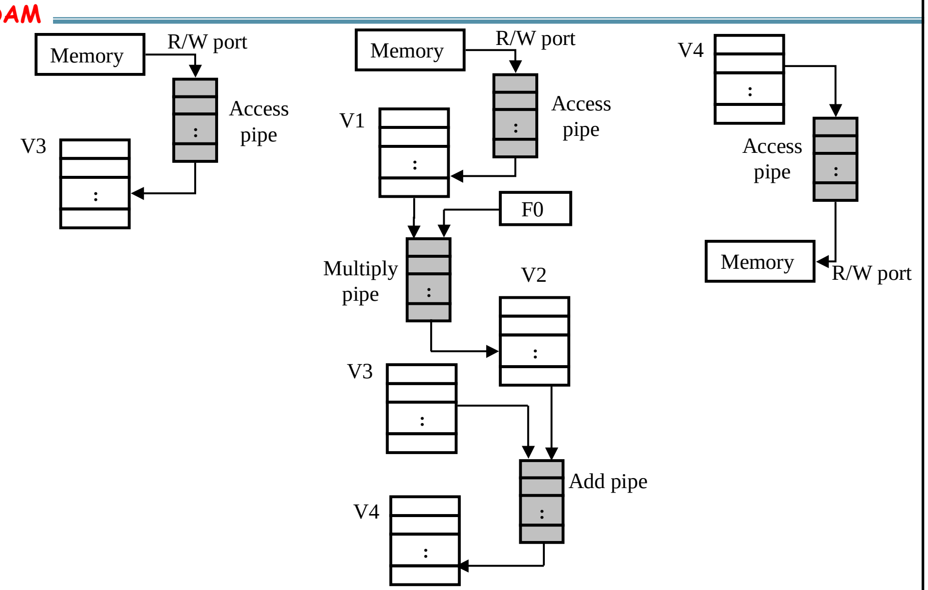
DAXPY Chaining: CRAY-1

- CRAY-1 has one memory access pipe either for load or store (not for both at the same time)
- 3 chains
  - Chain 1: LV V3
  - Chain 2: LV V1 + MULV V2,F0,V1 + ADDV V4,V2,V3
  - Chain 3: SV V4



CPE 631 ©AM

3 Chains DAXPY for CRAY-1

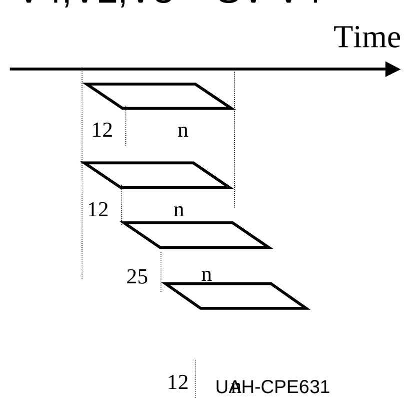


CPE 631 ©AM

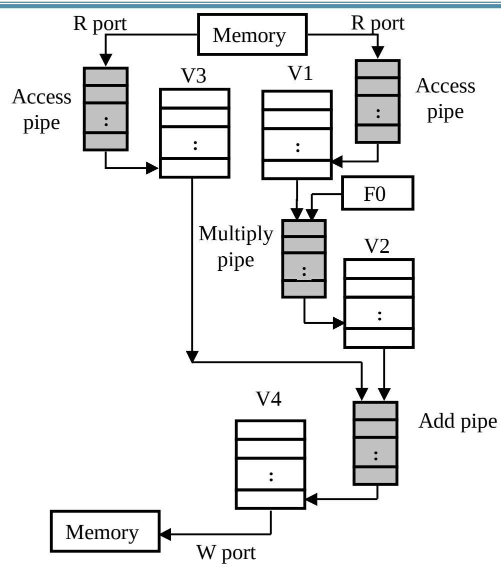
DAXPY Chaining: CRAY X-MP

- CRAY X-MP has 3 memory access pipes, two for vector load and one for vector store
- 1 chain: LV V3, LV V1 + MULV V2,F0,V1 + ADDV V4,V2,V3 + SV V4

Chain 1:



One Chain DAXPY for CRAY X-MP



Vector Opt #2: Conditional Execution

- Consider:

```
do 100 i = 1, 64
 if (A(i) .ne. 0) then
   A(i) = A(i) – B(i)
 endif
100 continue
```

- Vector-mask control takes a Boolean vector: when vector-mask register is loaded from vector test, vector instructions operate only on vector elements whose corresponding entries in the vector-mask register are 1
- Requires clock even for the elements where the mask is 0
- Some VP use vector mask only to disable the storing of the result and the operation still occurs;
zero division exception is possible? => mask operation

CPE 631 ©AM

Vector Mask Control

```
LV V1, Ra       ;load A into V1
LV V2, Rb       ;load B into V2
L.D F0, #0      ;load FP zero to F0
SNESV.D F0,V1 ;sets VM register if V1(i)<>0
SUBV.D V1,V1,V2 ;subtract under VM
CVM             ;set VM to all 1s
SV Ra,V1        ;store results in A
```


CPE 631 ©AM

Vector Opt #3: Sparse Matrices

- Sparse matrix: elements of a vector are usually stored in some compacted form and then accessed indirectly
- Suppose:

```
    do 100 i = 1, n
100 A(K(i))=A(K(i))+C(M(i))
```

- Mechanism to support sparse matrices: scatter-gather operations
- Gather (LVI) operation takes an index vector and fetches the vector whose elements are at the addresses given by adding a base address to the offsets given in the index vector => a nonsparse vector in a vector register
- After these elements are operated on in dense form, the sparse vector can be stored in expanded form by a scatter store (SVI), using the same index vector

CPE 631 ©AM

Sparse Matrices Example

```
    do 100 i = 1, n
100 A(K(i))=A(K(i))+C(M(i))
```

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

- Can't be done by compiler since can't know Ki elements distinct

CPE 631 ©AM

Sparse Matrices Example (cont'd)

```
LV V1,Ra    ;load A into V1
L.D F0,#0   ;load FP zero into F0
SNESV.D F0,V1 ;sets VM to 1 if V1(i)<>F0
CVI V2,#8   ;generates indices in V2
POP R1,VM   ;find the number of 1s
MTC1 VLR,R1 ;load vector-length reg.
CVM         ;clears the mask
LVI V3,(Ra+V2)   ;load the nonzero As
LVI V4,(Rb+V2)   ;load the nonzero Bs
SUBV.D V3,V3,V4  ;do the subtract
SVI (Ra+V2),V3   ;store A back
```

- Use CVI to create index 0, 1xm, ..., 63xm (compressed index vector whose entries correspond to the positions with a 1 in the mask register

Things to Remember

- Properties of vector processing
  - Each result independent of previous result
  - Vector instructions access memory with known pattern
  - Reduces branches and branch problems in pipelines
  - Single vector instruction implies lots of work (- loop)
- Components of a vector processor: vector registers, functional units, load/store, crossbar....
- Strip mining technique for long vectors
- Optimisation techniques: chaining, conditional execution, sparse matrices