Review:
Reg based vector architectures for certain class of problems
Runs multiple pipelined units efficiently
Vector reg size = 64
removes loops, data hazards between same operation
DLXV: vectorized DLX
Break:
Evening Midterm 7-9 in room ?
HW assignment: 7.3, 7.7 In class discussion: 7.15 and 7.16

Today:
Non 64 lenth vectors, Performance models, and compilers
DLXV Fig 7.5 on page 361

Vector length and stride
VLR: vectors shorter than 64
What if don't know the length?
Strip mining if greater; see p. 364

Vector stride:
What if 2D array?
2 Choices for layout in memory
7.10 p. 367 (Only Fortran uses column major order)

How get good prformance: banks
banks vs. interleaving: see page 363, Fig 7.6
(note access to bank 0 is not to 130; difference from interleaving)
=> latch/bank vs single latch
(latency is 6 clocks+per word)

stride: distance between vector elements in memory to be merged into a single vectgor

unit stride works best with memory;

DLXV supports nonunit stride:
LVWS, SVWS (With Stride); supplied as a GP register

non-unit stride can be a problem do to memory bank conflicts

Example: 32 bank memory and stride is 32 (double words), accesses from same bank

In general, the problem occurs if

$$\frac{\text{Least common multiple (Stride,Number of banks)}}{\text{Stride}} \geq \text{Memory-access latency}$$

ERROR: SHOULD BE <, not ≥

LCM(m,n) = p; m divides p; n divides p; p≤m*n
LCM(8,64) = 64
64 / 8 = 8
8 is not < 12, so conflict

Q: What is impact of increasing fast processor time for vector architecture?

=> memory latency => more banks

7.4 Model of performance

4 pieces

$T_{element}$. the time for each vector operation in the loop to process one element, ignoring the start-up costs. This time depends only on the execution of vector instructions.

The overhead for each strip-mined block of vector instructions. 2 parts:

T_{loop}, the cost of executing the scalar code for strip mining of each block

T_{start} the vector start-up cost for each block, .

T_{base} the overhead from computing the starting addresses and setting up the vector control. This time consists solely of scalar overhead instructions. This occurs once for the entire vector operation.

These components can be used to state the total running time for a vector sequence operating on a vector of length n, which we will call T_n:

$$T_n = T_{base} + \left\lfloor \frac{n}{MVL} + 1 \right\rfloor * \left(T_{loop} + T_{start}\right) + n * T_{element}$$

The values of T_{start} and T_{loop} are both compiler and machine dependent, while the value of $T_{element}$ depends mainly on the hardware. The exact vector sequence affects all three values; the effect on $T_{element}$ is probably the most pronounced, with T_{start} and T_{loop} less affected.

We use constants for T_{base} = 10 and T_{loop} = 15 on DLXV. Note T_{loop} is scalar and can overlap floating point

p. 370 shows code

top 6 instructions are T_{base}

6 scalar in loop are T_{loop}

3 vector operations set $T_{element}$ to 3

To calculate T_{start} is:

DLXV Fig 7.5 on page 361 says load latency is 12 clock cycles

4 clocks due to load/mult depdency

DLXV Fig 7.5 on page 361 says multiply latency is 7 clocks cycles

4 clocks due to mult/store depdency

12+4+7+4=27 clocks

p. 371, Fig 7.11: Why jump? Why time per element 50 less than time per element of 70, even though longer vector? (extra $T_{loop} + T_{start}$)

7.7 Evaluating Performance (N1/2, ...)

Traditionally use MFLOPS vs ET (leads to some troubles in field)
A few classic measures to categorize vector processors (includes overhead):

R_∞—The MFLOPS rate on an infinite-length vector.

(R_n is the MFLOPS rate for a vector of length *n*.)

$N_{1/2}$—The vector length needed to reach one-half of R_∞.

(This is a good measure of the impact of T_{base}.)

N_v—The vector length needed to make vector mode faster than scalar mode.

(This measures $T_{base}+T_{loop}+T_{start}$ and the speed of scalars relative to vectors.)

Keep this in mind; lets return to

7.6 Enhanced: Chaining

What about data hazards between vector instructions?

MULTV V1,V2,V3
ADDV V4,V1,V5

Time is

$T_{element}$ * Vector length + Start-up $time_{ADDV}$ + stall time + Start-up_{MULTV}

= 2 * Vector length + 6 + 4 + 7

= 2 * Vector length + 17

ADDV waits for MULTV to complete, just as if scalar isntruction

Suppose treat each vector register as 64 invididual elements vs monolith?

Chaining: vector starts as soon as first elements of vector avaialble vs last element of vector

internal forward from one FU to other FU

p. 378, Figure 7.14 shows with and without chaining (about 2X)

7.7 More Evaluating Performance

SAXPY inner loop bottom of page 384 takes:

at least 3 clocks for Telement (1 memory port, 3 memory accesses); chainining drops from 5

2 Fl. Pt. ops every 3 clocks (1.5 CPF)

2/3*Clock Rate = MFLOPS

But 80 MHz Cray1 runs at 1/6 (6 CPF). Why?

Ignores Tstart: sum of latencies for each peice: 12(L)+7(*)+12(L)+6(+)+12(S)=49

+Tloop=15 = 64 which is time of an operation=> 4*n vs 3*n

Why wait on store?

Also: average vector lenth = 66 vs infinity

Rest running in outer scalar loop

(60% of time in scalar loop)

N1/2 for SAXPY? Rinf = 40, so need to calcualte vector size for 20 MFLOPS

p. 387

8 clocks to achieve 20 MFLOPS

$$T_n = T_{base} + \left\lfloor \frac{n}{MVL} + 1 \right\rfloor * \left(T_{loop} + T_{start}\right) + n * T_{element}$$

Tn = 8*n

assume n< 64

(Tloop = 15, Tstart = 49,Tbase=10)

8*n = 10 + 1*64 +n*3

5*n = 74

n = 15

Nv for SAXPY?

Assume scalar time would be 12+12+6+7+scalar loop time (say 10)

47

Tn = 74+3*n

=> 74/44 or 2!

(let them read about multiple pipelines)

Later 7.6 and 7.5