

CS4223 Tutorial 5: Data Level Parallelism

1. Consider the following vector code

```
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
```

Assume 4 cycle vector add latency, 6 cycle vector multiply latency, and 10 cycle vector load/store latency. Adder, multiplier, and load/store units are all pipelined. Also assume 8-lane vector processor, **that is, the processor contains 8 units of pipelined adder, multiplier, and load/store unit.** Further assume that there is no conflict among the memory bank accesses. How long will it take to execute this code if **vector length = 32, VLRMAX = 32 and there is no chaining?**

ANS: It will require $32/8 = 4$ **iterations per instructions** because there are 8 lanes and vector length is 32.

So LV/SV will take $10+3 = 13$ cycles; ADDV.D will take $4+3 = 7$ cycles, and MULTVS will take $6+3 = 9$ cycles.

In this case, MULV.S and LV can execute in parallel.

So total cycles = 13 (LV) + 13 (MULVS, LV) + 7 (ADDV) + 13 (SV) = 46 cycles

Note that as the units are pipelined, the next 8 elements can be fed into the pipeline while the first 8 elements are being processed.

2. Consider the following code fragment that produces 128 results.

```
for (i=0; i<64; i++) {
    a[i] = u * b[i];
    c[i] = b[i] + v;
}
```

Assume a vector pipeline with the following pipeline latencies and one lane per operation. Also assume there are no bank conflicts in the accesses for the above loop.

Operation	Start-up penalty
Vector load/store	12 cycles
Vector multiply	7 cycles
Vector add	6 cycles

- (a) Write the vector version of this code in assembly language.

ANS:

```
LD    ru, u
LD    rv, v
LV    Vb, rb
MULV  Va, Vb, ru
ADDV  Vc, Vb, rv
SV    Va, ra
SV    Vc, rc
```

- (b) Assuming no chaining and single memory pipeline, how many clock cycles will be required to execute this code fragment including start-up overheads.

ANS:

We ignore the scalar loads (first two instructions). The rest can be put into 4 chains.

```
LV    Vb, rb
MULV  Va, Vb, ru    ADDV  Vc, Vb, rv
SV    Va, ra
SV    Vc, rc
```

12 cycle for load + 63 cycles

7 cycle for the next chain startup + 63 cycles

12 cycle for store + 63 cycles

12 cycle for store + 63 cycles

- (c) If the vector sequence is chained but still has single memory pipeline, how many clock cycles will be required to execute this code fragment including start-up overheads?

ANS: 3 chains

```
LV    Vb, rb  MULV  Va, Vb, ru    ADDV  Vc, Vb, rv
SV    Va, ra
SV    Vc, rc
```

12 + 7 cycle for first chain startup + 63 cycles

12 cycle for store + 63 cycles

12 cycle for store + 63 cycles

- (d) Assuming three memory pipelines and chaining, how many clock cycles are required to execute this code fragment including start-up overheads?

ANS: 1 chain

12 + 7 + 12 cycle for chain startup + 63 cycles

2. Consider the following code, which multiplies two vectors that contain single-precision complex values:

```
for (i=0; i< 300; i++){
    c_re[i] = a_re[i] * b_re[i] - a_im[i] * b_im[i];
    c_im[i] = a_re[i] * b_im[i] + a_im[i] * b_re[i];
}
```

Assume that the processor runs at 700 MHz and has a maximum vector length of 64. The load/store unit has a startup overhead of 15 cycles; the multiply unit, 8 cycles; the add/subtract unit, 5 cycles.

- a) What is the arithmetic intensity (number of operations per byte transferred) of this kernel?
This code reads four floats and writes two floats, i.e., 6x4 byte = 24 byte for every six FLOP, so arithmetic intensity = $6/24 = 0.25$
- b) Convert this loop into VMIPS assembly code using strip mining.

This is the assembly code. For the C-code with strip mining, please see slide 37 of DLP lecture. The part in red is the vector code. The rest takes care of strip mining.

Assume MVL = 64:

```
li $VL,44          # perform the first 44 ops
li $r1,0           # initialize index
loop:  lv $v1,a_re+$r1  # load a_re
      lv $v3,b_re+$r1  # load b_re
      mulvv.s $v5,$v1,$v3  # a+re*b_re
      lv $v2,a_im+$r1  # load a_im
      lv $v4,b_im+$r1  # load b_im
      mulvv.s $v6,$v2,$v4  # a+im*b_im
      subvv.s $v5,$v5,$v6  # a+re*b_re - a+im*b_im
      sv $v5,c_re+$r1  # store c_re
      mulvv.s $v5,$v1,$v4  # a+re*b_im
      mulvv.s $v6,$v2,$v3  # a+im*b_re
      addvv.s $v5,$v5,$v6  # a+re*b_im + a+im*b_re
      sv $v5,c_im+$r1  # store c_im
```

```

        bne $r1,0,else      # check if first iteration
        addi $r1,$r1,#176   # first iteration, increment by 44x4
        li $VL, 64         # reset VL to 64
        j loop              # guaranteed next iteration
else:    addi $r1,$r1,#256   # not first iteration, increment by 64x4
skip:    blt $r1,1200,loop   # next iteration?

```

- c) **Assuming chaining and a single memory pipeline**, how many clock cycles are required per complex result value, including start-up overhead?

1. load a_re
2. load b_re → a_re * b_re (chaining)
3. load a_im
4. load b_im → a_im * b_im → sub (chaining)
5. a_re * b_im, store c_re
6. a_im * b_re → add → store c_im (chaining)

If you want accurate results, you should consider that the first outer loop processes only 44 vector elements. But for simplicity I will assume 64 vector elements.

1. Load 15 cycles + 63 cycles
2. Load 15 cycles + mult 8 cycles + 63 cycles
3. Load 15 cycles + 63 cycles
4. Load 15 cycles + mult 8 cycles + sub 5 cycles + 63 cycles
5. Store 15 cycles (worst path between mult and store) + 63 cycles
6. mult 8 cycles + add 5 cycles + store 15 cycles + 63 cycles

Calculate the total cycles = 502 cycles for 64 vector elements

So $502/64 = 7.84$ cycles per complex result

- d) Now assume that the processor **has three memory pipelines and chaining**. If there are no bank conflicts in the loop's accesses, how many clock cycles are required per result?

1. load a_re, load b_re → a_re * b_re
2. load a_im, load b_im → a_im * b_im → sub
3. store c_re, a_re * b_im
4. a_im * b_re → add → store c_im

1. Load 15 cycles + mult 8 cycles + 63 cycles (loads in parallel)
2. Load 15 cycles + mult 8 cycles + sub 5 cycles + 63 cycles (loads in parallel)
3. Store 15 cycles (worst path between store and mult) + 63 cycles
4. mult 8 cycles + add 5 cycles + store 15 cycles + 63 cycles

Total = 346 cycles for 64 vector elements. So $354/64 = 5.41$ cycles per complex result

3. The following kernel performs a portion of the finite-difference time-domain (FDTD) method for computing Maxwell's equations in a three-dimensional space, part of the SPEC06fp benchmarks:

```
for (int x = 0; x < NX-1; x++) {  
    for (int y = 0; y < NY-1; y++) {  
        for (int z = 0; z < NZ-1; z++) {  
            int index = x*NY*NZ + y*NZ + z;  
            if (y > 0 && x > 0) {  
                material = IDx[index];  
                dH1 = (Hz[index] - Hz[index-incrementY])/dy[y];  
                dH2 = (Hy[index] - Hy[index-incrementZ])/dz[z];  
                Ex[index] = Ca[material] * Ex[index] + Cb[material] * (dH2 - dH1);  
            }  
        }  
    }  
}
```

Assume that dH1, dH2, Hy, Hz, dy, dz, Ca, Cb, Ex are all single-precision floating-point arrays. Assume IDx is an array of unsigned int.

a) What is the arithmetic intensity of this kernel?

Reads 40 bytes and writes 4 bytes for every 8 FLOPs, thus $8/44$ FLOPs/byte = 0.18 FLOPs/byte

b) Assume this kernel is to be executed on a processor that has 30 GB/sec of memory bandwidth. Will this kernel be memory bound or compute bound?

Having an arithmetic intensity of 0.18, if the processor has a peak floating-point throughput $> (30 \text{ GB/s}) \times (0.18 \text{ FLOPs/byte}) = 5.4 \text{ GFLOPs/s}$, then this code is likely to be memory-bound, unless the working set fits well within the processor's cache.

c) Develop a roofline model for this processor, assuming it has a peak computational throughput of 85 GFLOP/sec. [We will discuss roofline model in class]

The single precision arithmetic intensity corresponding to the edge of the roof is $85/30 = 2.83$ FLOPs/byte.

At $1/8$ arithmetic intensity, the performance will be $30 \times 1/8 = 3.75$ GFLOP