Design of Digital Circuits (252-0028-00L), Spring 2018

Optional HW 5: Vector Processors and GPUs

SOLUTIONS

Instructor: Prof. Onur Mutlu

TAs: Juan Gomez Luna, Hasan Hassan, Arash Tavakkol, Minesh Patel, Jeremie Kim, Giray Yaglikci

Assigned: Thursday, May 24, 2018

1 Vector Processing

Consider the following piece of code:

```
for (i = 0; i < 100; i ++)
  A[i] = ((B[i] * C[i]) + D[i])/2;
```

(a) Translate this code into assembly language using the following instructions in the ISA (note the number of cycles each instruction takes is shown next to each instruction):

Opcode	Operands	Number of Cycles	Description
LEA	Ri, X	1	Ri ← address of X
LD	Ri, Rj, Rk	11	Ri ← MEM[Rj + Rk]
ST	Ri, Rj, Rk	11	MEM[Rj + Rk] ← Ri
MOVI	Ri, Imm	1	Ri ← Imm
MUL	Ri, Rj, Rk	6	Ri ← Rj x Rk
ADD	Ri, Rj, Rk	4	Ri ← Rj + Rk
ADD	Ri, Rj, Imm	4	Ri ← Rj + Imm
RSHFA	Ri, Rj, amount	1	Ri ← RSHFA (Rj, amount)
BRcc	X	1	Branch to X based on condition codes

Assume one memory location is required to store each element of the array. Also assume that there are 8 registers (R0 to R7).

Condition codes are set after the execution of an arithmetic instruction. You can assume typically available condition codes such as zero, positive, and negative.

```
MOVI    R1, 99       // 1 cycle
LEA     R0, A        // 1 cycle
LEA     R2, B        // 1 cycle
LEA     R3, C        // 1 cycle
LEA     R4, D        // 1 cycle
LOOP:
LD      R5, R2, R1   // 11 cycles
LD      R6, R3, R1   // 11 cycles
MUL     R7, R5, R6   // 6 cycles
LD      R5, R4, R1   // 11 cycles
ADD     R6, R7, R5   // 4 cycles
RSHFA   R7, R6, 1    // 1 cycle
ST      R7, R0, R1   // 11 cycles
ADD     R1, R1, -1   // 4 cycles
BRGEZ   R1  LOOP     // 1 cycle
```

How many cycles does it take to execute the program?

$5 + 100 \times 60 = 6005$

(b) Now write Cray-like vector assembly code to perform this operation in the shortest time possible. Assume that there are 8 vector registers and the length of each vector register is 64. Use the following instructions in the vector ISA:

Opcode	Operands	Number of Cycles	Description
LD	Vst, #n	1	Vst ← n (Vst = Vector Stride Register)
LD	Vln, #n	1	Vln ← n (Vln = Vector Length Register)
VLD	Vi, X	11, pipelined	
VST	Vi, X	11, pipelined	
Vmul	Vi, Vj, Vk	6, pipelined	
Vadd	Vi, Vj, Vk	4, pipelined	
Vrshfa	Vi, Vj, amount	1	

```
LD      Vln, 50
LD      Vst, 1
VLD     V1,  B
VLD     V2,  C
VMUL    V4,  V1,  V2
VLD     V3,  D
VADD    V6,  V4, V3
VRSHFA  V7,  V6, 1
VST     V7,  A

VLD     V1,  B + 50
VLD     V2,  C + 50
VMUL    V4,  V1,  V2
VLD     V3,  D + 50
VADD    V6,  V4, V3
VRSHFA  V7,  V6, 1
VST     V7,  A + 50
```

How many cycles does it take to execute the program on the following processors? Assume that memory is 16-way interleaved.

(i) Vector processor without chaining, 1 port to memory (1 load or store per cycle).

```
The third load (VLD) can be pipelined with the add (VADD). However as there is
just only one port to memory and no chaining, other operations cannot be
pipelined.

Processing the first 50 elements takes 346 cycles as below

| 1 | 1 | 11 | 49 | 11 | 49 | 6 | 49 |
                              | 11 | 49 | 4 | 49 | 1 | 49 | 11 | 49 |

Processing the next 50 elements takes 344 cycles as shown below (no need to
initialize Vln and Vst as they stay at the same value).

| 11 | 49 | 11 | 49 | 6 | 49 |
                      | 11 | 49 | 4 | 49 | 1 | 49 | 11 | 49 |

Therefore, the total number of cycles to execute the program is 690 cycles
```

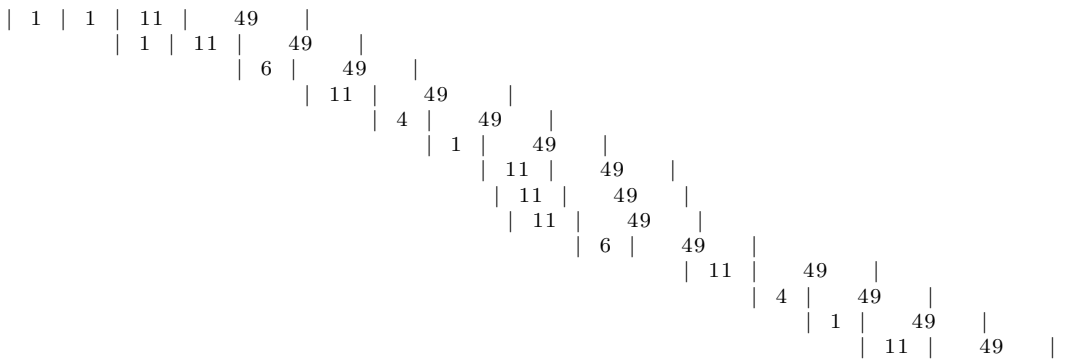
(ii) Vector processor with chaining, 1 port to memory.

```
In this case, the first two loads cannot be pipelined as there is only one port
to memory and the third load has to wait until the second load has completed.
However, the machine supports chaining, so all other operations can be
pipelined.

Processing the first 50 elements takes 242 cycles as below

| 1 | 1 | 11 |   49   | 11 |   49   |
                          | 6 |   49   |
                                | 11 |   49   |
                                     | 4 |   49   |
                                         | 1 |   49   |
                                             | 11 |   49   |


Processing the next 50 elements takes 240 cycles (same time line as above, but
without the first 2 instructions to initialize Vln and Vst).

Therefore, the total number of cycles to execute the program is 482 cycles
```

(iii) Vector processor with chaining, 2 read ports and 1 write port to memory.

Assuming an in-order pipeline.

The first two loads can also be pipelined as there are two ports to memory. The third load has to wait until the first two loads complete. However, the two loads for the second 50 elements can proceed in parallel with the store.



Therefore, the total number of cycles to execute the program is 215 cycles

2 More Vector Processing

You are studying a program that runs on a vector computer with the following latencies for various instructions:

- VLD and VST: 50 cycles for each vector element; fully interleaved and pipelined.
- VADD: 4 cycles for each vector element (fully pipelined).
- VMUL: 16 cycles for each vector element (fully pipelined).
- VDIV: 32 cycles for each vector element (fully pipelined).
- VRSHF (right shift): 1 cycle for each vector element (fully pipelined).

Assume that:

- The machine has an in-order pipeline.
- The machine supports chaining between vector functional units.
- In order to support 1-cycle memory access after the first element in a vector, the machine interleaves vector elements across memory banks. All vectors are stored in memory with the first element mapped to bank 0, the second element mapped to bank 1, and so on.
- Each memory bank has an 8 KB row buffer.
- Vector elements are 64 bits in size.
- Each memory bank has two ports (so that two loads/stores can be active simultaneously), and there are two load/store functional units available.

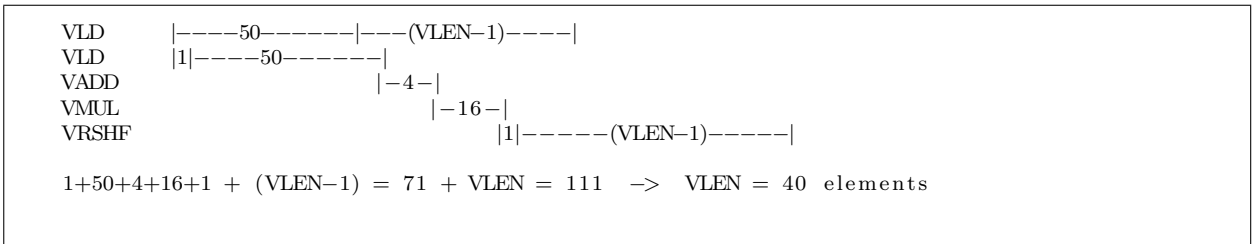
(a) What is the minimum power-of-two number of banks required in order for memory accesses to never stall? (Assume a vector stride of 1.)

64 banks (because memory latency is 50 cycles and the next power of two is 64)

(b) The machine (with as many banks as you found in part a) executes the following program (assume that the vector stride is set to 1):

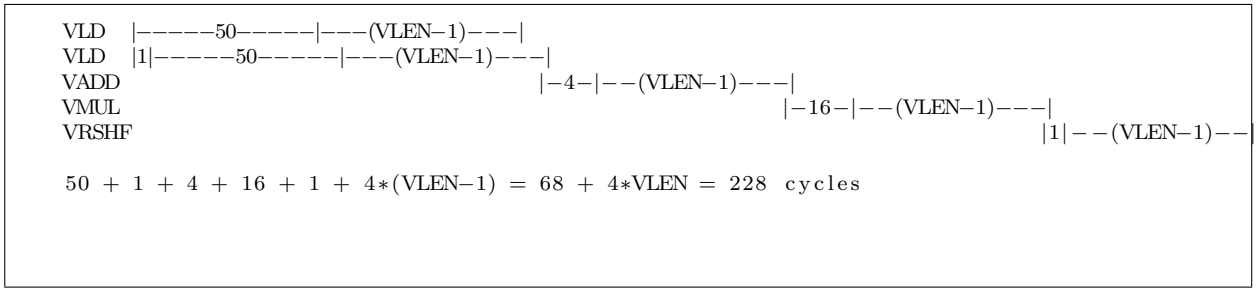
```
VLD V1 ← A
VLD V2 ← B
VADD V3 ← V1, V2
VMUL V4 ← V3, V1
VRSHF V5 ← V4, 2
```

It takes 111 cycles to execute this program. What is the vector length?

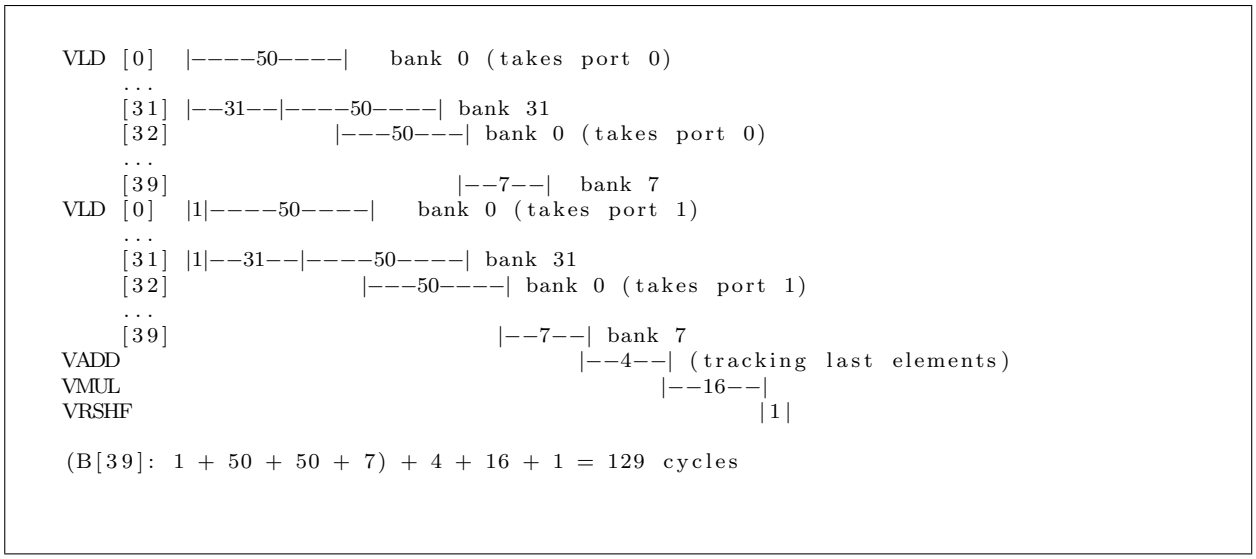


1+50+4+16+1 + (VLEN−1) = 71 + VLEN = 111 -> VLEN = 40 elements

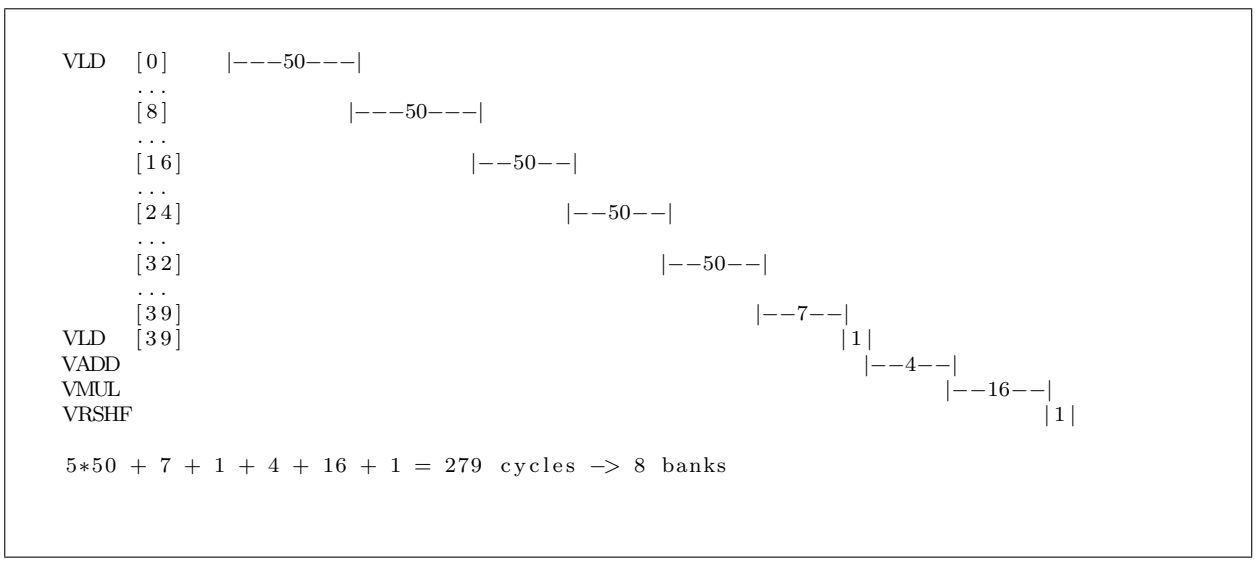
If the machine did not support chaining (but could still pipeline independent operations), how many cycles would be required to execute the same program?



(c) The architect of this machine decides that she needs to cut costs in the machine's memory system. She reduces the number of banks by a factor of 2 from the number of banks you found in part (a) above. Because loads and stores might stall due to bank contention, an *arbiter* is added to each bank so that pending loads from the oldest instruction are serviced first. How many cycles does the program take to execute on the machine with this reduced-cost memory system (but with chaining)?



Now, the architect reduces cost further by reducing the number of memory banks (to a lower power of 2). The program executes in 279 cycles. How many banks are in the system?



(d) Another architect is now designing the second generation of this vector computer. He wants to build a multicore machine in which 4 vector processors share the same memory system. He scales up the number of banks by 4 in order to match the memory system bandwidth to the new demand. However, when he simulates this new machine design with a separate vector program running on every core, he finds that the average execution time is longer than if each individual program ran on the original single-core system with 1/4 the banks. Why could this be? Provide concrete reason(s).

Row-buffer conflicts (all cores interleave their vectors across all banks).

What change could this architect make to the system in order to alleviate this problem (in less than 20 words), while *only* changing the shared memory hierarchy?

Partition the memory mappings, or using better memory scheduling.

3 SIMD Processing

Suppose we want to design a SIMD engine that can support a vector length of 16. We have two options: a traditional vector processor and a traditional array processor.

Which one is more costly in terms of chip area (circle one)?

`The traditional vector processor` `The traditional array processor` `Neither`

Explain:

An array processor requires 16 functional units for an operation whereas a vector processor requires only 1.

Assuming the latency of an addition operation is five cycles in both processors, how long will a VADD (vector add) instruction take in each of the processors (assume that the adder can be fully pipelined and is the same for both processors)?

For a vector length of 1:

The traditional vector processor:

5 cycles

The traditional array processor:

5 cycles

For a vector length of 4:

The traditional vector processor:

8 cycles (5 for the first element to complete, 3 for the remaining 3)

The traditional array processor:

5 cycles

For a vector length of 16:

The traditional vector processor:

20 cycles (5 for the first element to complete, 15 for the remaining 15)

The traditional array processor:

5 cycles

4 GPUs and SIMD I

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A and B are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 2 instructions in each thread.) A warp in the GPU consists of 32 threads, there are 32 SIMD lanes in the GPU. Assume that each instruction takes the same amount of time to execute.

```
for (i = 0; i < N; i++) {
    if (A[i] % 3 == 0) {     // Instruction 1
       A[i] = A[i] * B[i];   // Instruction 2
    }
}
```

(a) How many warps does it take to execute this program? Please leave the answer in terms of N.

$\lceil(\frac{N}{32})\rceil$

(b) Assume integer arrays A have a repetitive pattern which have 24 ones followed by 8 zeros repetitively and integer arrays B have a different repetitive pattern which have 48 zeros followed by 64 ones. What is the SIMD utilization of this program?

((24+8*2)/(32*2))*100% = 40/64*100 = 62.5%

(c) Is it possible for this program to yield a SIMD utilization of 100% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 100%?

Yes. If all of A's elements are divisible by 3, or if all are not divisible by 3.

What should be true about arrays B?

B can be any array of integers.

If NO, explain why not.

(d) Is it possible for this program to yield a SIMD utilization of 56.25% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 56.25%?

Yes, if 4 out of every 32 elements of A are divisible by 3.

What should be true about arrays B?

B can be any array of integers.

If NO, explain why not.

(e) Is it possible for this program to yield a SIMD utilization of 50% (circle one)?

YES NO

If YES, what should be true about arrays A for the SIMD utilization to be 50%?

What should be true about arrays B?

If NO, explain why not.

No. The minimum is when only 1 out of every 32 elements of A is divisible by 3. This yields a 51.5625% usage.

Now, we will look at the technique we learned in class that tries to improve SIMD utilization by merging divergent branches together. The key idea of the *dynamic warp formation* is that threads in one warp can be swapped with threads in another warp as long as the swapped threads have access to the associated registers (i.e., they are on the same SIMD lane).

Consider the following example of a program that consists of 3 warps X, Y and Z that are executing the same code segment specified at the top of this question. Assume that the vector below specifies the direction of the branch of each thread within the warp. 1 means the branch in Instruction 1 is resolved to taken and 0 means the branch in Instruction 1 is resolved to not taken.

```
X = {10000000000000000000000000000010}
Y = {10000000000000000000000000000001}
Z = {01000000000000000000000000000000}
```

(f) Given the example above. Suppose that you perform dynamic warp formation on these three warps. What is the resulting outcome of each branch for the newly formed warps X', Y' and Z'.

There are several answers for this questions but the key is that the taken branch in Z can be combined with either X or Y. However, the taken branch in the first thread of X and Y cannot be merged because they are on the same GPU lane.
X = 10000000000000000000000000000010
Y = 11000000000000000000000000000001
Z = 00000000000000000000000000000000

(g) Given the specification for arrays A and B, is it possible for this program to yield a better SIMD utilization if dynamic warp formation is used? Explain your reasoning.

No. Branch divergence happens on the same lane throughout the program resulting in the case where there is no dynamically formed warp.

5 GPUs and SIMD II

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 4 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU.

```
for (i = 0; i < 1048576; i++) {
    if (B[i] < 4444) {
        A[i] = A[i] * C[i];
        B[i] = A[i] + B[i];
        C[i] = B[i] + 1;
    }
}
```

(a) How many warps does it take to execute this program?

Warps = (Number of threads) / (Number of threads per warp)
Number of threads = 2^{20} (i.e., one thread per loop iteration).
Number of threads per warp = $64 = 2^6$ (given).
Warps = $2^{20}/2^6 = 2^{14}$

(b) When we measure the SIMD utilization for this program with one input set, we find that it is 67/256. What can you say about arrays A, B, and C? Be precise (Hint: Look at the "if" branch, what can you say about A, B and C?).

A: Nothing

B: 1 in every 64 of B's elements is less than 4444

C: Nothing

(c) Is it possible for this program to yield a SIMD utilization of 100% (circle one)?

YES NO

If YES, what should be true about arrays A, B, C for the SIMD utilization to be 100%? Be precise. If NO, explain why not.

Yes. Either:
(1) All of B's elements are greater than or equal to 4444, or
(2) All of B's element are less than 4444.

(d) Is it possible for this program to yield a SIMD utilization of 25% (circle one)?

YES NO

If YES, what should be true about arrays A, B, and C for the SIMD utilization to be 25%? Be precise.
If NO, explain why not.

No. The smallest SIMD utilization possible is the same as part (b), 67/256, but this is greater than 25%.

6 GPUs and SIMD III

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the *complete run* of the program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU. Please assume that all values in array B have magnitudes less than 10 (i.e., $|B[i]| < 10$, for all i).

```
for (i = 0; i < 1024; i++) {
    A[i] = B[i] * B[i];
    if (A[i] > 0) {
        C[i] = A[i] * B[i];
        if (C[i] < 0) {
            A[i] = A[i] + 1;
        }
        A[i] = A[i] - 2;
    }
}
```

Please answer the following five questions.

(a) How many warps does it take to execute this program?

Warps = (Number of threads) / (Number of threads per warp)
Number of threads = 2^{10} (i.e., one thread per loop iteration).
Number of threads per warp = $64 = 2^6$ (given).
Warps = $2^{10}/2^6 = 2^4$

(b) What is the maximum possible SIMD utilization of this program?

100%

(c) Please describe what needs to be true about array B to reach the maximum possible SIMD utilization asked in part (b). (Please cover all cases in your answer)

B: For every 64 consecutive elements: every value is 0, every value is positive, or every value is negative. Must give all three of these.

(d) What is the minimum possible SIMD utilization of this program?

Answer: 132/384
Explanation: The first two lines must be executed by every thread in a warp (64/64 utilization for each line). The minimum utilization results when a single thread from each warp passes both conditions on lines 2 and 4, and every other thread fails to meet the condition on line 2. The thread per warp that meets both conditions, executes lines 3-6 resulting in a SIMD utilization of 1/64 for each line. The minimum SIMD utilization sums to $(64 * 2 + 1 * 4)/(64 * 6) = 132/384$

(e) Please describe what needs to be true about array B to reach the minimum possible SIMD utilization asked in part (d). (Please cover all cases in your answer)

B: Exactly 1 of every 64 consecutive elements must be negative. The rest must be zero. This is the only case that this holds true.