

CMU 18-447 INTRODUCTION TO COMPUTER ARCHITECTURE, SPRING 2014
HW 4: SIMD, VLIW, GPU, AND CACHING

Instructor: Prof. Onur Mutlu
TAs: Rachata Ausavarungnirun, Varun Kohli, Xiao Bao Zhao, Paraj Tyle

Assigned: Wed., 2/26, 2014
Due: **Wed., 3/19, 2014 (Midnight)**
Handin: /afs/ece/class/ece447/handin/hw4

1 GPUs and SIMD [35 points]

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 < 1024768; 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 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.

B:

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

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

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.

The smallest SIMD utilization possible is the same as part (b), $67/256$, but this is greater than 25%.
--

2 Cache Enigma (Optional)

A processor has a 4-way set-associative L1 cache that can house 4 blocks in total. The access latency to this cache is 1 cycle. The replacement policy is true LRU. The processor is known to not employ any prefetching mechanism.

The processor also has a 16-way set-associative L2 cache that can house 128 blocks in total. The access latency to this cache is 20 cycles.

A programmer writes a test program that in a loop repeatedly accesses only the following data cache blocks (assume billions of iterations are run):

A, B, C, D, E, F

where A, ..., F are different cache block addresses.

In the steady state (i.e., after the loop has executed for a few iterations), the programmer finds out that the average memory access time is 1 cycle.

Then, the programmer writes another program that in a loop repeatedly accesses only the following data cache blocks:

A, B, C, D, E, F, G, H

In the steady state (i.e., after the loop has executed for a few iterations), the programmer finds out that the average memory access time is 20 cycles.

(a) What can you say about this processor? (I.e., what is going on?)

Please describe everything you can say, concretely, but be concise.

(b) Based on the above information, what do you expect the average memory access time of yet another program that in a loop repeatedly accesses only the following data cache blocks?

A, B, C, D, E

Explain:

(c) Again, based on the above information, what do you expect the average memory access time of yet another program that in a loop repeatedly accesses only the following data cache blocks?

A, B, C, D, E, F, G

Explain:

- (d) Finally, again, based on the above information, what do you expect the average memory access time of yet another program that in a loop repeatedly accesses only the following data cache blocks?

A, B, C, D, E, F, G, H, I

Explain:

3 Cache and Virtual Memory (Optional)

A four-way set-associative writeback cache has a 2^{11} · 89-bit tag store. The cache uses a custom replacement policy that requires 9 bits per set. The cache block size is 64 bytes. The cache is virtually-indexed and physically-tagged. Data from a given physical address can be present in up to eight different sets in the cache. The system uses hierarchical page tables with two levels. Each level of the page table contains 1024 entries. A page table may be larger or smaller than one page. The TLB contains 64 entries.

- (a) How many bits of the virtual address are used to choose a set in the cache?
- (b) What is the size of the cache data store?
- (c) How many bits in the Physical Frame Number must overlap with the set index bits in the virtual address?
- (d) On the following blank figure representing a virtual address, draw in bitfields and label bit positions for “cache block offset” and “set number.” Be complete, showing the beginning and ending bits of each field.

Virtual Address:

- (e) On the following blank figure representing a physical address, draw in bitfields and label bit positions for “physical frame number” and “page offset.” Be complete, showing the beginning and ending bits of each field.

Physical Address:

- (f) What is the page size?
- (g) What is the size of the virtual address space?
- (h) What is the size of the physical address space?

4 Vector Processing [40 points]

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, etc.
- Each memory bank has an 8KB 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. There is another solution if one interprets “never stall” to mean that a single load will never stall rather than the memory accesses in the program below: in that case, 32 banks suffices since each bank has two ports. For those who answered this way on the test, we gave full credit.

(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?

40 elements

```
VLD      |-----50-----|--- (VLEN-1) ----|
VLD      |1|-----50-----|
VADD     |-----50-----|--- (VLEN-1) ----|
VMUL     |-----50-----|--- (VLEN-1) ----|
VRSHF    |-----50-----|--- (VLEN-1) ----|
50+1+4+16+1 + (VLEN-1) = 71 + VLEN = 111 -> VLEN = 40
```

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

228 cycles

```
VLD      |-----50-----|--- (VLEN-1) ----|
VLD      |1|-----50-----|--- (VLEN-1) ----|
VADD     |-----50-----|--- (VLEN-1) ----|
VMUL     |-----50-----|--- (VLEN-1) ----|
VRSHF    |-----50-----|--- (VLEN-1) ----|
50 + 1 + 4 + 16 + 1 + 4*(VLEN-1) = 68 + 4*VLEN = 228
```

- (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)?

129 cycles

```

VLD [0]  |----50----|   bank 0 (takes port 0)
...
[31]  |--31--|----50----| bank 31
[32]           |---50---| bank 0 (takes port 0)
...
[39]           |--7--|   bank 7
VLD [0]  |1|----50----|   bank 0 (takes port 1)
...
[31]  |1|--31--|----50----| bank 31
[32]           |---50----| bank 0 (takes port 1)
...
[39]           |--7--|   bank 7
VADD                                |--4--| (tracking last elements)
VMUL                                |--16--|
VRSHF                                |1|
(B[39]: 1 + 50 + 50 + 7) + 4 + 16 + 1 = 129 cyc

```

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?

8 banks

```

VLD [0]  |---50---|
...
[8]           |---50---|
...
[16]           |--50--|
...
[24]           |--50--|
...
[32]           |--50--|
...
[39]           |--7--|
VLD [39]           |1|
VADD                                |--4--|
VMUL                                |--16--|
VRSHF                                |1|
5*50 + 7 + 1 + 4 + 16 + 1 = 279 cyc

```

- (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 (in less than 20 words)? Provide concrete reason(s).

Inter-application memory interference which leads to loss of row-buffer locality and bank-level parallelism. This occurs because all applications interleave their vectors across all banks in the system.

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 applications across channels, or use application-aware memory scheduling.

5 Programming a Systolic Array [35 points]

Figure 1 shows a systolic array processing element.

Each processing element takes in two inputs, M and N, and outputs P and Q. Each processing element also contains an “accumulator” R that can be read from and written to. The initial value of the “accumulator” is 0.

Figure 2 shows a systolic array composed of 9 processing elements. The smaller boxes are the inputs to the systolic array and the larger boxes are the processing elements. You will program this systolic array to perform the following calculation:

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

In each time cycle, each processing element will take in its two inputs, perform any necessary actions, and write on its outputs. The time cycle labels on the input boxes determine which time cycle the inputs will be fed into their corresponding processing elements. Any processing element input that is not driven will default to 0, and any processing element that has no output arrow will have its output ignored.

After all the calculations finish, each processing element’s “accumulator” will hold one element of the final result matrix, arranged in the correct order.

- (a) Please describe the operations that each individual processing element performs, using mathematical equations and the variables M, N, P, Q and R.

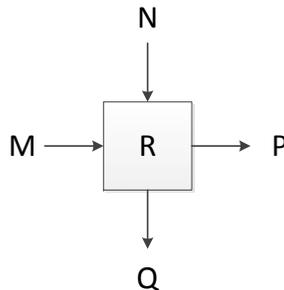


Figure 1: A systolic array processing element

P =	M
Q =	N
R =	$R + M \times N$

(b) Please fill in all 30 input boxes in Figure 2 so that the systolic array computes the correct matrix multiplication result described on the previous page. (Hint: Use a_{ij} and b_{ij} .)

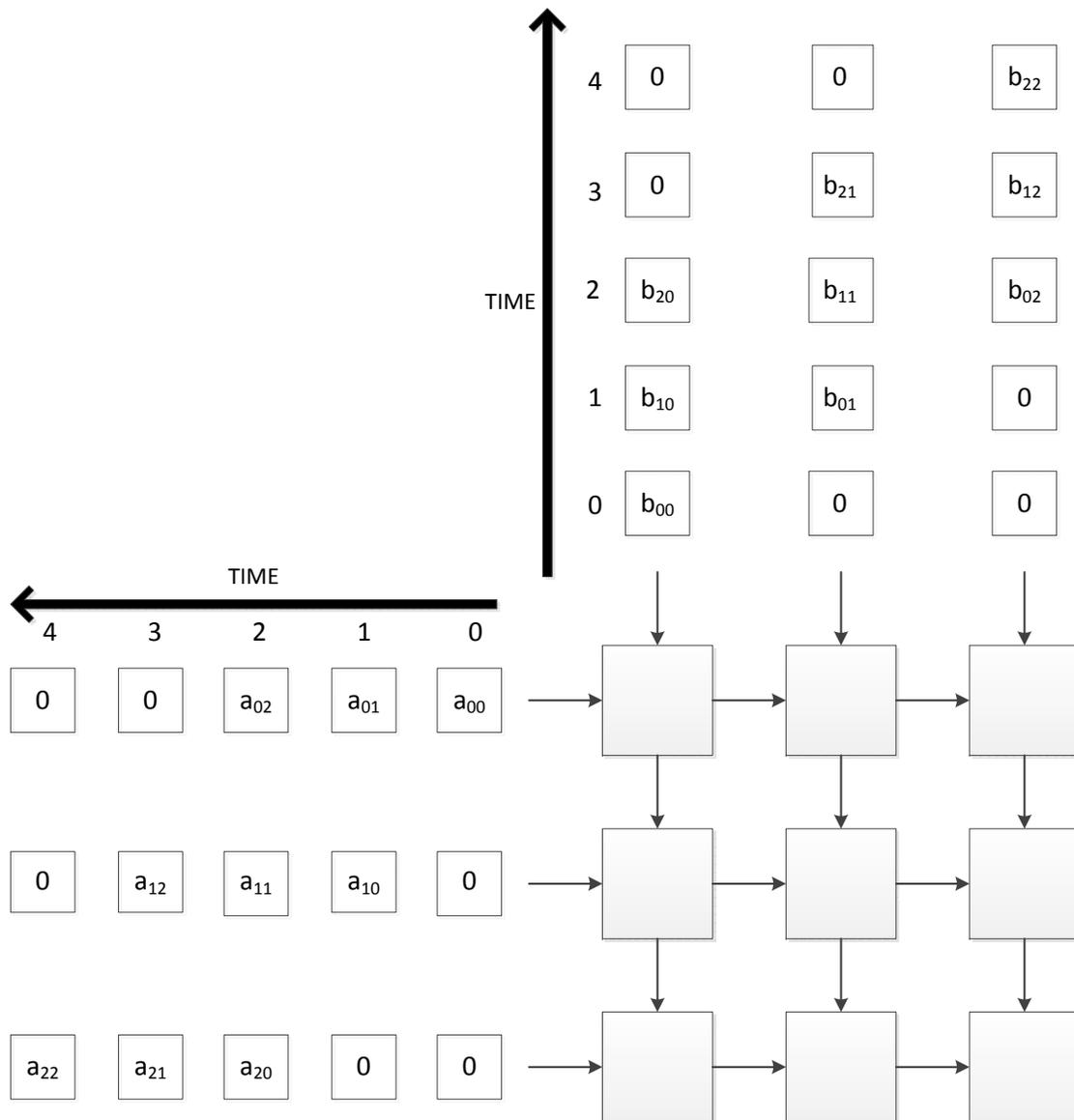


Figure 2: A systolic array