Computer Architecture (263-2210-00L), Fall 2017
HW 4: Vector Processing, GPU, Memory Scheduling, Cache Partitioning

Instructor: Prof. Onur Mutlu
TAs: Hasan Hassan, Arash Tavakkol, Mohammad Sadr, Lois Orosa, Juan Gomez Luna

Assigned: Thursday, Nov 9, 2017
Due: **Thursday, Nov 23, 2017**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture/`. Please check your inbox. You should have received an email with the password you can use to login to the paper review system. If you have not received any email, please contact comparch@lists.ethz.ch. In the first page after login, you should click in "Architecture - Fall 2017 Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-7).** Please upload your solution to the Moodle (`https://moodle-app2.let.ethz.ch/`) as a single PDF file. **Please use a typesetting software (e.g., LaTeX) or a word processor (e.g., MS Word, LibreOfficeWriter) to generate your PDF file. Feel free to draw your diagrams either using an appropriate software or by hand, and include the diagrams into your solutions PDF.**

1 Critical Paper Reviews [200 points]

Please read the following handout on how to write critical reviews. We will give out extra credit that is worth 0.5% of your total grade for each good review.

- Lecture slides on guidelines for reviewing papers. Please follow this format. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=onur-comparch-f17-how-to-do-the-paper-reviews.pdf`
- Some sample reviews can be found here: `https://safari.ethz.ch/architecture/fall2017/doku.php?id=readings`

(a) Write a one-page critical review for at least **two** of the following papers:

- E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems," ASPLOS 2010. `https://people.inf.ethz.ch/omutlu/pub/fst_asplos10.pdf`
- Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," HPCA 2010. `https://people.inf.ethz.ch/omutlu/pub/atlas_hpca10.pdf`
- S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning," MICRO 2011. `https://people.inf.ethz.ch/omutlu/pub/memory-channel-partitioning-micro11.pdf`
- L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," TPDS 2016. `https://people.inf.ethz.ch/omutlu/pub/bliss-memory-scheduler_ieee-tpds16.pdf`

2 Vector Processing [200 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, 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.)

(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?

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 machines 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).

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?

3 Vector Processing [100 points]

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 × 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.

How many cycles does it take to execute the program?

(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	

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).

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

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

4 GPUs [150 points]

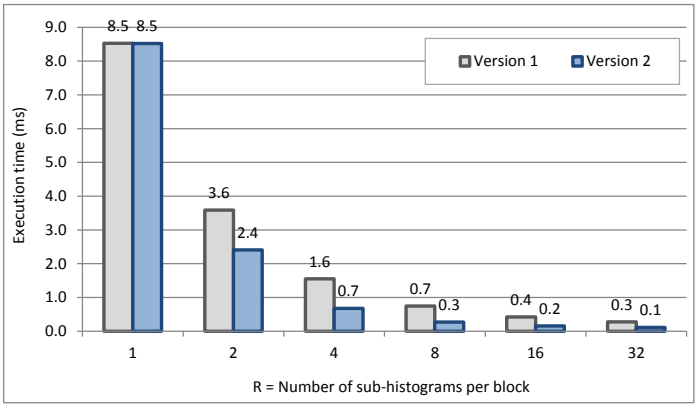
Histograms are a powerful tool in many fields, such as image processing. Their implementation on GPU is challenging because of the need for atomic operations. One way to accelerate their computation is using privatization in the fast shared memory. The following code calculates the histogram of an image "img" using privatization:

```
1 extern __shared__ unsigned int Hs[];// Dynamic shared memory allocation
2 __global__ void histogram_kernel(
3         unsigned int* histo, unsigned int* img, int size, int BINS){
4    // Block and thread index
5    const int bx = blockIdx.x;
6    const int tx = threadIdx.x;
7    // Constants for read access
8    const int begin = bx * blockDim.x + tx;
9    const int end = size;
10   const int step = blockDim.x * gridDim.x;
11   // Sub-histogram initialization
12   for(int pos = tx; pos < BINS; pos += blockDim.x) Hs[pos]=0;
13   __syncthreads();  // Intra-block synchronization
14   // Main loop
15   for(int i = begin; i < end; i += step){
16     // Global memory read
17     unsigned int d = img[i];
18     // Atomic vote in shared memory
19     atomicAdd(&Hs[d], 1);
20   }
21    __syncthreads();  // Intra-block synchronization
22    // Merge in global memory
23   for(int pos = tx; pos < BINS; pos += blockDim.x){
24     unsigned int sum = 0;
25     sum = Hs[pos];
26     // Atomic addition in global memory
27     atomicAdd(histo + pos, sum);
28   }
29 }
```

(a) As natural images are smooth (that is, they present spatial correlation), it is very likely that neighboring pixels fall into the same bin. To avoid atomic conflicts, R sub-histograms per block can be used (and later merged). Lets analyze two different ways of accessing the sub-histograms (to replace line 19):

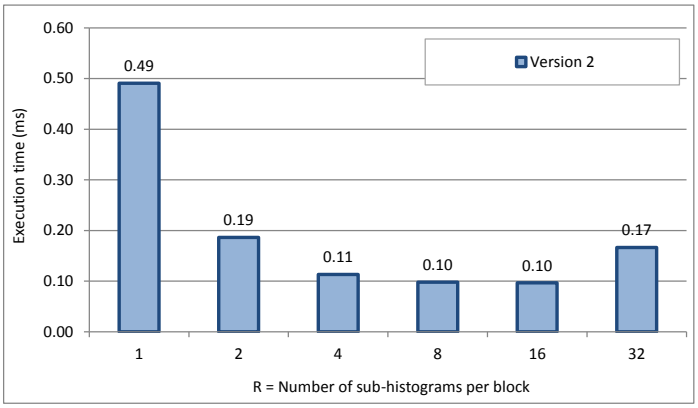
```
atomicAdd(&Hs[BINS * (tx % R) + d], 1); // Version 1
atomicAdd(&Hs[tx % R + d * R], 1); // Version 2
```

This graph shows the execution time for a 32-bins image histogram:



Why does version 2 obtain better results? What would happen for an odd-number-sized histogram?

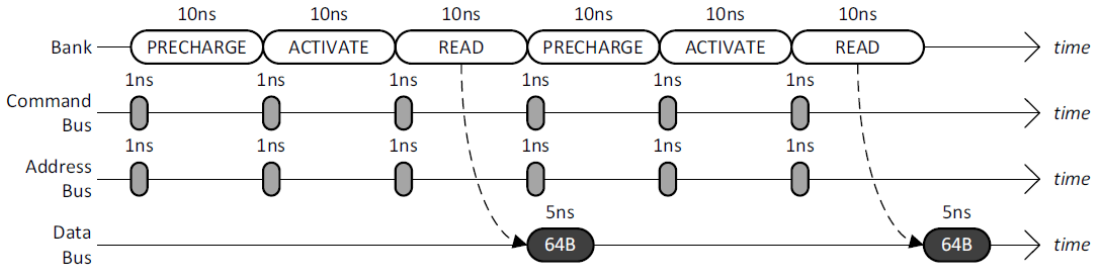
(b) As can be seen in the above graph, increasing the number R of sub-histogram tends to reduce the number of atomic conflicts, and consequently the execution time. Could you then explain the following graph? (Note: Histograms of 256 bins are calculated. Tests have been carried out on a Kepler GPU with a maximum of 64 warps per multiprocessor, and 48 KB of shared memory. Blocks of 256 threads are used).



(c) For very large histograms, privatization in shared memory is not possible, unless multiple passes are carried out. Assume that, given the limited shared memory availability, N passes are needed. Atomic operations in shared memory take 2 ns to complete. For each pass, 10% of the input data loads hit the L2 cache. Compare this multi-pass approach to an approach where the histogram resides in global memory. Assume a GPU with global memory atomic operations in L2. Each atomic operation takes 10 ns to complete in L2, and 200 ns to complete in DRAM. 95% of the atomic operations hit the L2 cache. Find the value of N that makes worthwhile each of the approaches. (Note: the global memory bandwidth is 100 GB/s, and the L2 is 10 times faster).

5 Memory Scheduling [200 points]

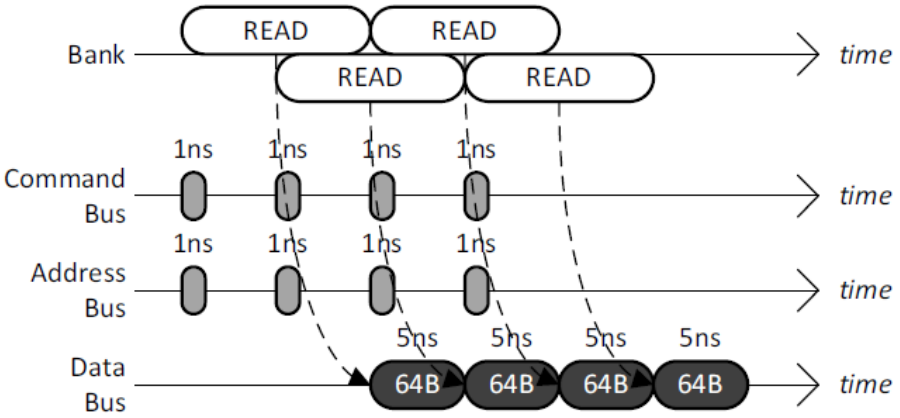
Row-Buffer Conflicts. The following timing diagram shows the operation of a single DRAM channel and a single DRAM bank for two back-to-back reads that conflict in the row-buffer. Immediately after the bank has been busy for 10ns with a READ, data starts to be transferred over the data bus for 5ns.



(a) Given a long sequence of back-to-back reads that always conflict in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

(b) To increase the data throughput, the main memory designer is considering adding more DRAM banks to the single DRAM channel. Given a long sequence of back-to-back reads to all banks that always conflict in the row-buffers, what is the minimum number of banks that is required to achieve the maximum data throughput of the main memory system?

Row-Buffer Hits. The following timing diagram shows the operation of the single DRAM channel and the single DRAM bank for four back-to-back reads that hit in the row-buffer. It is important to note that rowbuffer hits to the same DRAM bank are pipelined: while each READ keeps the DRAM bank busy for 10ns, up to at most **half** of this latency (5ns) can be overlapped with another read that hits in the row-buffer. (Note that this is different from Lab 6 where we unrealistically assumed that row-buffer hits are non-pipelined.)



(c) Given a long sequence of back-to-back reads that always hits in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

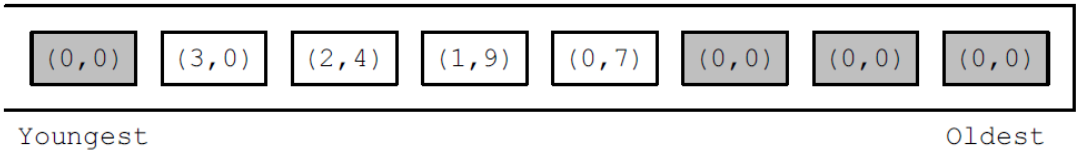
(d) When the maximum data throughput is achieved for a main memory system that has a single DRAM channel and a single DRAM bank, what is the bottleneck that prevents the data throughput from becoming even larger? **Circle** all that apply.

BANK COMMAND BUS ADDRESS BUS DATA BUS

Memory Scheduling Policies. The diagram below shows the memory controllers request queue at time 0. The shaded rectangles are read requests generated by thread T0, whereas the unshaded rectangles are

read requests generated by thread T1. Within each rectangle, there is a pair of numbers that denotes the requests (BankAddress, RowAddress). Assume that the memory system has a single DRAM channel and four DRAM banks. Further assume the following.

- All the row-buffers are closed at time 0.
- Both threads start to stall at time 0 because of memory.
- A thread continues to stall until it receives the data for all of its requests.
- Neither thread generates more requests.



For extra credits (50 points), please make sure that you model contention in the banks as well as in all of the buses (address/command/data).

(e) For the FCFS scheduling policy, calculate the memory stall time of T0 and T1.

(f) For the FR-FCFS scheduling policy, calculate the memory stall time of T0 and T1

(g) For the PAR-BS scheduling policy, calculate the memory stall time of T0 and T1. Assume that all eight requests are included in the same batch.

6 Memory Scheduling [50 points]

In class, we covered "parallelism-aware batch scheduling," which is a memory scheduling algorithm that aims to reduce interference between threads in a multi-core system.

(a) What benefit does request batching provide in this algorithm?

(b) How does the algorithm preserve intra-thread bank parallelism?

(c) If thread ranking was formed in a "random manner" (i.e., threads were assigned a random rank), would each thread's parallelism be preserved? Why or why not? Explain.

7 Utility-Based Cache Partitioning [100 points]

(a) Does utility-based cache partitioning guarantee a minimum amount of cache space to each thread/core sharing the cache? Why or why not? Explain.

(b) If yes, describe (and analyze) the minimum level of guarantee provided by utility based cache partitioning to each thread. If no, describe how the basic utility-based cache partitioning mechanism can be modified to provide a minimum amount of cache space to each thread.

(c) Describe how you would perform utility based cache partitioning if each core has an identical prefetcher that prefetches into the shared cache. What needs to be modified in the utility based cache partitioning mechanism described by Qureshi and Patt (MICRO 2006) to take into account prefetches? Explain the new hardware design.