Computer Architecture (263-2210-00L), Fall 2019

HW 4: Memory Interference and QoS - **Solutions**

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, Rahul Bera, Geraldo Francisco De Oliveira Junior, Can Firtina, Juan Gomez Luna, Jawad Haj-Yahya, Hasan Hassan, Konstantinos Kanellopoulos, Jeremie Kim, Nika Mansouri Ghiasi, Lois Orosa Nogueira, Jisung Park, Minesh Hamenbhai Patel, Abdullah Giray Yaglikci

Assigned: Monday, Nov 18, 2019
Due: **Sunday, Dec 1, 2019**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture19/`. 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 2019 Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-5).** 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 [400 points]

Please read the guidelines for reviewing papers and check the sample reviews. You may access them by *simply clicking on the QR codes below or scanning them.* We will give out extra credit that is worth 0.5% of your total grade for each good review.

Guidelines

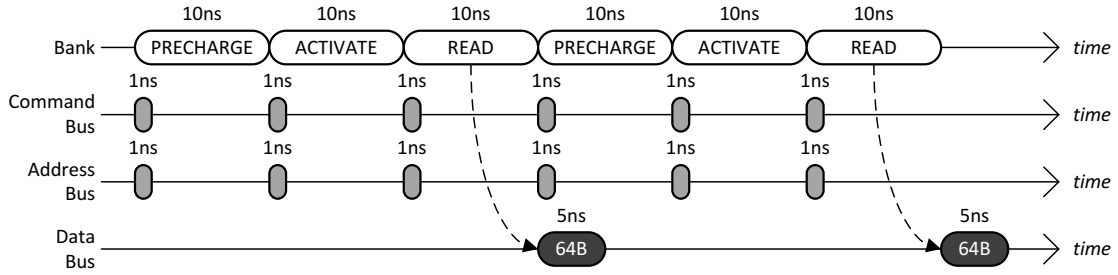
Sample reviews

Write an approximately one-page critical review for each of the following papers. A review with bullet point style is more appreciated. Try not to use very long sentences and paragraphs. Keep your writing and sentences simple. Make your points bullet by bullet, as much as possible.

- Veynu Narasiman, Chang Joo Lee, Michael Shebanow, Rustam Miftakhutdinov, Onur Mutlu, and Yale N. Patt, "Improving GPU Performance via Large Warps and Two-Level Warp Scheduling" MICRO 2011. `https://people.inf.ethz.ch/omutlu/pub/large-gpu-warps_micro11.pdf`
- O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," ISCA 2008. `https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf`
- Ebrahimi E, Lee CJ, Mutlu O, Patt YN. "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`
- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative" ISCA 2009. `https://people.inf.ethz.ch/omutlu/pub/pcm_isca09.pdf`

2 Memory Interference and QoS [100 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.



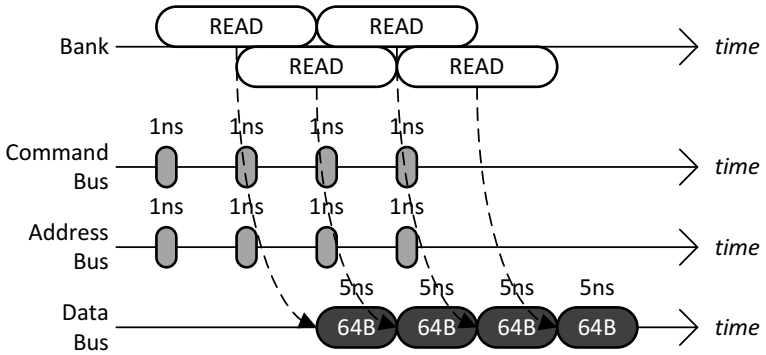
(a) Given a long sequence of back-to-back reads that always conflict in the row-buffer, what is the data throughput of the main memory system? Please state your answer in **gigabytes/second**.

64B/30ns = 32B/15ns = 32GB/15s = 2.13 GB/s

(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?

30ns/5ns = 6

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 row-buffer 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.



(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**.

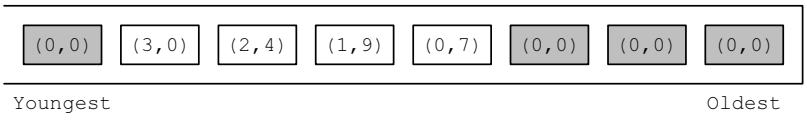
64B/5ns = 64GB/5s = 12.8GB/s

(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 through- put from becoming even larger? **Circle** all that apply.

BANK **COMMAND BUS** **ADDRESS BUS** **DATA BUS**

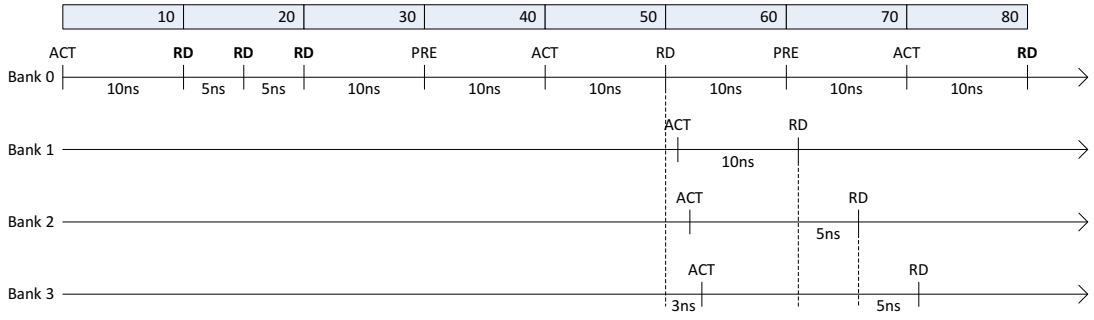
Memory Scheduling Policies. The diagram below shows the memory controller's *request queue* at time 0. The shaded rectangles are read requests generated by thread *T0*, whereas the unshaded rectangles are read requests generated by thread *T1*. Within each rectangle, there is a pair of numbers that denotes the request's (*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.



We provide two sets of answers. The correct way to solve the problem is to model contention in the banks as well as in all of the buses (address/command/data). The answer that is given in the answer boxes is for the case you modeled contention in only the banks.

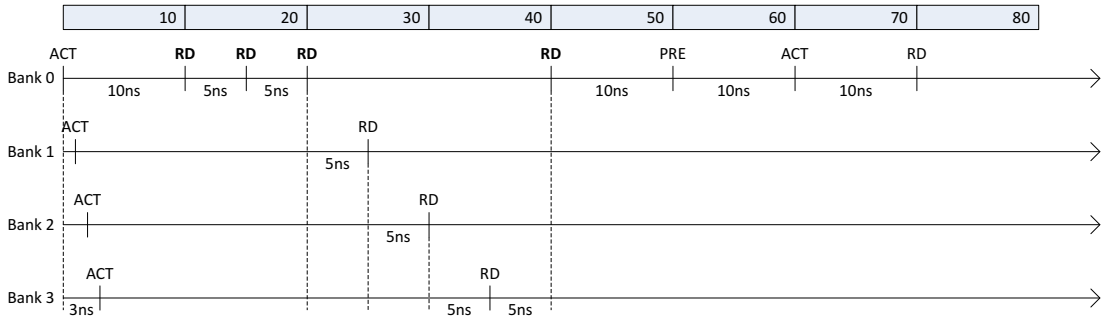
(f) For the $FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 10 + 10 + 10) + 10 + 10 + 10 + 10 + 5 = 95ns

T1: (10 + 5 + 5 + 10 + 10 + 10) + 1 + 10 + 5 + 5 + 10 + 5 = 86ns

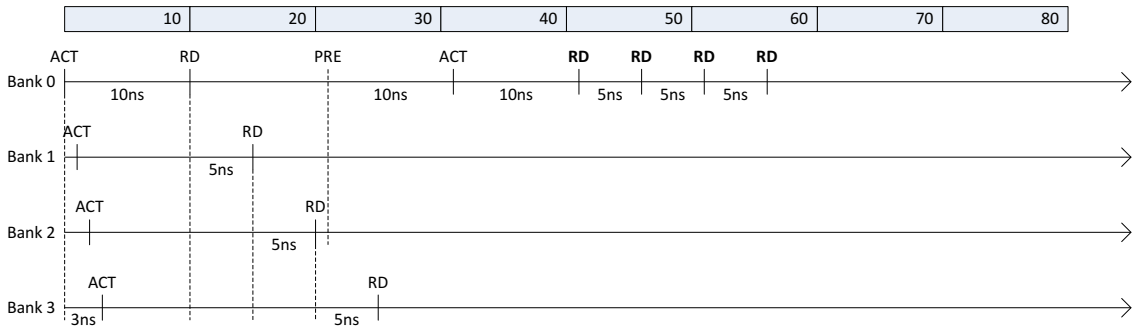
(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.



T0: (10 + 5 + 5 + 5 + 5 + 5 + 5) + 10 + 5 = 55ns

T1: (10 + 5 + 5 + 5 + 5 + 5 + 5) + 10 + 10 + 10 + 10 + 5 = 85ns

(h) 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.



T0: (10 + 5 + 5) + 1 + 10 + 10 + 5 + 5 + 5 + 10 + 5 = 71ns

T1: (10 + 5 + 5) + 5 + 10 + 5 = 40ns

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

T0:

Bank 0 is the critical path for both threads.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 30ns + 5ns
= 95ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 30ns + 5ns
= 65ns

(g) For the $FR-FCFS$ scheduling policy, calculate the memory stall time of *T0* and *T1*.

T0:

Bank 0 is the critical path for both threads. First, we serve all four shaded requests since they are row-buffer hits. Lastly, we serve the unshaded request.

T0 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 5ns + 5ns + 5ns + 5ns
= 40ns

T1:

T1 = Closed + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Conflict + Data
= (ACT+RD)+(RD/2)+(RD/2)+(RD/2)+(PRE+ACT+RD)+DATA
= 20ns + 5ns + 5ns + 5ns + 30ns + 5ns
= 70ns

(h) 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.

First, we serve all four unshaded requests in parallel across the four banks. Then, we serve all four shaded requests in serial.

T0: T0 = Closed + Conflict + Pipelined-Hit + Pipelined-Hit + Pipelined-Hit + Data
= (ACT+RD)+(PRE+ACT+RD)+(RD/2)+(RD/2)+(RD/2)+DATA
= 20ns + 30ns + 5ns + 5ns + 5ns + 5ns
= 70ns

T1: T1 = Closed + Data
= (ACT+RD)+DATA
= 20ns + 5ns
= 25ns

3 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?

Request batching allows PAR-BS to avoid starvation as requests of older batches are always prioritized over requests of younger batches.

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

Threads are ranked based on the number of requests they have at all the banks. All banks service requests based on this ranking. Hence, requests from the same thread will likely be serviced in parallel in different banks, which preserves the threads bank-level 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.

It depends on the applications. Assume there are two banks and there is one application which has a lot of requests to one bank and no requests to the other. A random ranking can prioritize this application over others and thereby preventing the other applications from exploiting their bank-level parallelism.

4 Memory System [80 points]

A machine with a 4 GB DRAM main memory system has 4 channels, 1 rank per channel and 4 banks per rank. The cache block size is 64 bytes.

(a) You are given the following byte addresses and the channel and bank to which they are mapped:

Byte:	$0x0000$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0100$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0200$	$\Rightarrow$	Channel	0,	Bank	0
Byte:	$0x0400$	$\Rightarrow$	Channel	1,	Bank	0
Byte:	$0x0800$	$\Rightarrow$	Channel	2,	Bank	0
Byte:	$0x0C00$	$\Rightarrow$	Channel	3,	Bank	0
Byte:	$0x1000$	$\Rightarrow$	Channel	0,	Bank	1
Byte:	$0x2000$	$\Rightarrow$	Channel	0,	Bank	2
Byte:	$0x3000$	$\Rightarrow$	Channel	0,	Bank	3

Determine which bits of the address are used for each of the following address components. Assume row bits are higher order than column bits:

- Byte on bus
  Addr [2 : 0]
- Channel bits (channel bits are contiguous)
  Addr [11 : 10]
- Bank bits (bank bits are contiguous)
  Addr [13 : 12]
- Column bits (column bits are contiguous)
  Addr [9 : 3]
- Row bits (row bits are contiguous)
  Addr [31 : 14]

(b) Two applications App 1 and App 2 share this memory system (using the address mapping scheme you determined in part (a)). The memory scheduling policy employed is FR-FCFS. The following requests are queued at the memory controller request buffer at time t. Assume the first request (A) is the oldest and the last one (A + 15) is the youngest.

A B A + 1 A + 2 A + 3 B + 10 A + 4 B + 12 A + 5 A + 6 A + 7
A + 8 A + 9 A + 10 A + 11 A + 12 A + 13 A + 14 A + 15

These are cache block addresses, not byte addresses. Note that requests to A + x are from App 1, while requests to B + x are from App 2. Addresses A and B are row-aligned (i.e., they are at the start of a row) and are at the same bank but are in different rows.

Assuming row-buffer hits take T time units to service and row-buffer conflicts/misses take 2T time units to service, what is the slowdown (compared to when run alone on the same system) of

- App 1?

1.

All requests A + x map to one row, and all requests B + x map to another row (both rows are in the same bank), because there are 16 cache blocks/row and in all requests above, x $<$ 16. Since the request for cache block address A comes first, the row containing all requested cache blocks A+x will be opened and all of these requests, which are row-buffer hits come first (and will complete in time 1 2T + 15 1T = 17T). Thus, none of App 1's requests are ever delayed by requests from App 2, and so they all execute at exactly the same time as they would if App 2 were not running. This results in a slowdown of 1.

- App 2?

21/4.

When running alone, App 2's three requests are a row buffer-closed access (time 2T) and two row buffer hits (each taking time T); thus they complete in 1 2T + 2 1T = 4T time. When running with App 1, all of App 1's requests come first, in time 17T (see above). Then App 2's requests execute as in the alone case (row conflict, row hit, row hit) in 4T time. Hence App 2's requests are completed at time 21T. Slowdown is thus 21T /4T = 21/4.

(c) Which application slows down more?

App 2.

Why?

The high row-buffer locality of App 1 causes its requests to occupy the bank for a long period of time with the FR-FCFS scheduling policy, denying App 2 of service during that period.

(d) In class, we discussed memory channel partitioning and memory request scheduling as two solutions to mitigate interference and application slowdowns in multicore systems. Propose another solution to reduce the slowdown of the more-slowed-down application, without increasing the slowdown of the other application? Be concrete.

Interleaving data at a sub-row or cache line granularity could reduce the slowdown of App 2 by reducing the row-buffer locality of App 1 which causes the interference.

One possible interleaving scheme that achieves this is shown below:

- Byte on bus Addr [2 : 0]
- Lower Column bits Addr [7 : 3]
- Channel bits Addr [9 : 8]
- Bank bits Addr [11 : 10]
- Higher Column bits Addr [13 : 12]
- Row bits Addr [31 : 14]

This address interleaving scheme interleaves 256 KB chunks across channels. Thus, the longest row hit streak would be 4, as compared to 16 in the original interleaving scheme in part (a), preventing App 2's requests from being queued behind 16 of App 1's requests.

5 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.)

Solution:

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?

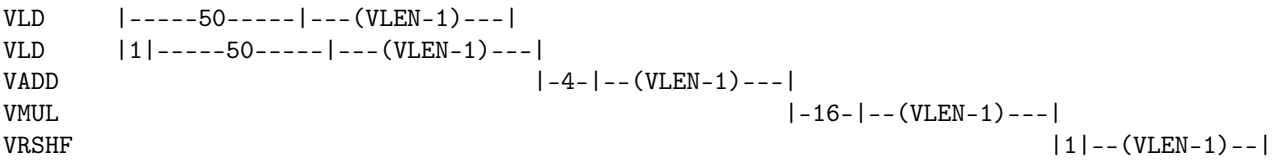
Solution:

```
VLD     |----50------|---(VLEN-1)----|
VLD     |1|----50------|
VADD                  |-4-|
VMUL                      |-16-|
VRSHF                         |1|-----(VLEN-1)-----|
```

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?

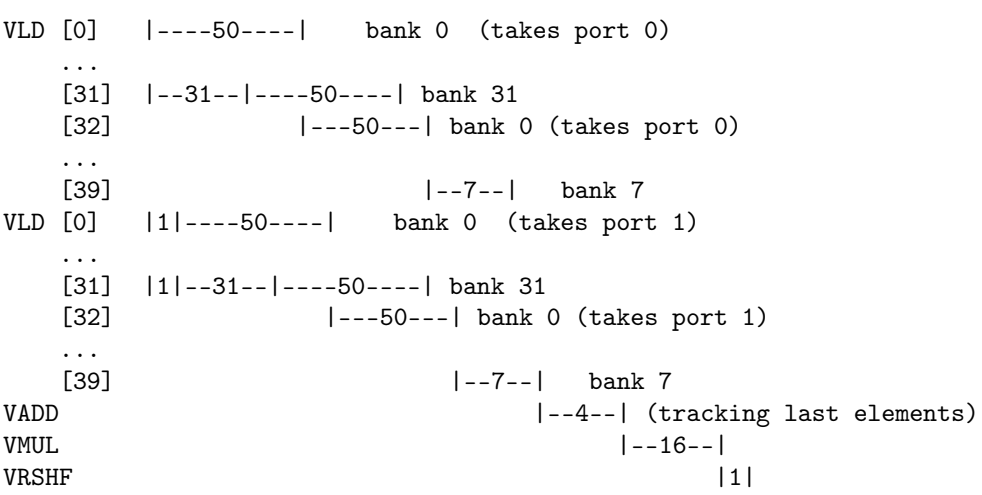
Solution:



$50 + 1 + 4 + 16 + 1 + 4 \times (\text{VLEN - } 1) = 68 + 4 \times \text{VLEN} = 228$ cycles

(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)?

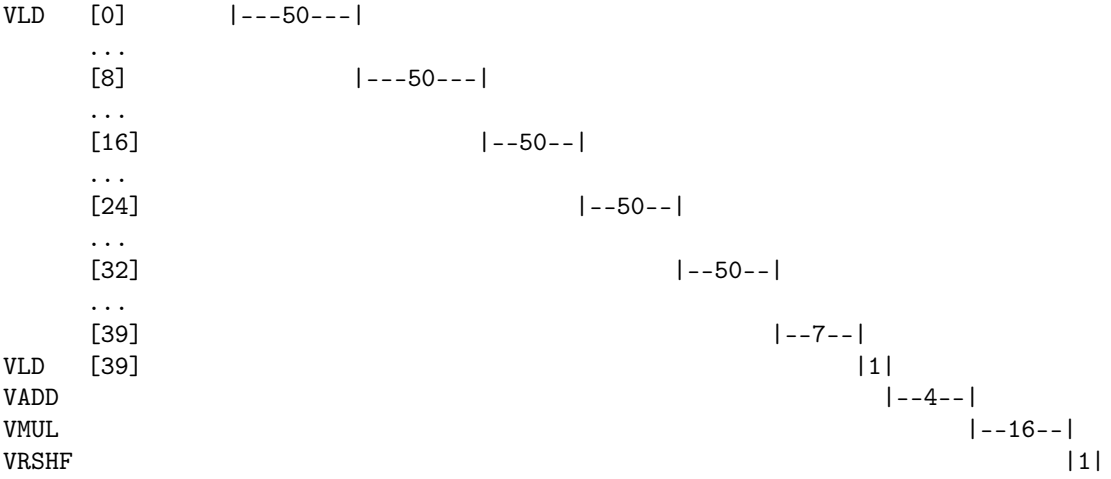
Solution:



(B[39]: $1 + 50 + 50 + 7) + 4 + 16 + 1 = 129$ cycles

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?

Solution:



$5{\times}50 + 7 + 1 + 4 + 16 + 1 = 279$ cycles $\rightarrow$ 8 banks

(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).

Solution:

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?

Solution:

Partition the memory mappings, or using better memory scheduling.

6 GPUs and SIMD [100 points]

We define the *SIMD utilization* of a program that runs 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, 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` and `B` are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 3 instructions in each iteration.) A warp in the GPU consists of 32 threads, and there are 32 SIMD lanes in the GPU.

```
for (i = 0; i < 1025; i++) {
    if (A[i] < 33) {        // Instruction 1
        B[i] = A[i] << 1;   // Instruction 2
    }
    else {
        B[i] = A[i] >> 1;   // Instruction 3
    }
}
```

Please answer the following six questions.

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

33 warps.

Explanation:
The number of warps is calculated as:
$\#Warps = \lceil \frac{\#Total\_threads}{\#Warp\_size} \rceil$,

where
$\#Total\_threads = 1025 = 2^{10} + 1$ (i.e., one thread per loop iteration),

and
$\#Warp\_size = 32 = 2^5$ (given).

Thus, the number of warps needed to run this program is:
$\#Warps = \lceil \frac{2^{10}+1}{2^5} \rceil = 2^5 + 1 = 33$.

(b) What is the maximum possible SIMD utilization of this program? (Hint: The warp scheduler does not issue instructions where no threads are active).

$\frac{1025}{1056}$.

Explanation:
Even though all active threads in a warp follow the same execution path, the last warp will only have one active thread.

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

For every 32 consecutive elements of `A`, every element should be lower than 33 (`if`), or greater than or equal to 33 (`else`). (NOTE: The solution is correct if both cases are given.)

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

$\frac{1025}{1568}$.

Explanation:
`Instruction 1` is executed by every active thread ($\frac{1025}{1056}$ utilization).
Then, part of the threads in each warp executes `Instruction 2` and the other part executes `Instruction 3`. We consider that `Instruction 2` is executed by α threads in each warp (except the last warp), where $0 < \alpha \leq 32$, and `Instruction 3` is executed by the remaining $32 - \alpha$ threads.
The only active thread in the last warp executes either `Instruction 2` or `Instruction 3`. The other instruction is not issued for this warp.

The minimum SIMD utilization sums to $\frac{1025+\alpha\times32+(32-\alpha)\times32+1}{1056+1024+1024+32} = \frac{1025}{1568}$.

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

For every 32 consecutive elements of `A`, part of the elements should be lower than 33 (`if`), and the other part should be greater than or equal to 33 (`else`).

(f) What is the SIMD utilization of this program if `A[i] = i`? Show your work.

$\frac{1025}{1072}$.

Explanation:
`Instruction 1` is executed by every active thread ($\frac{1025}{1056}$ utilization).
`Instruction 2` is executed by the first 33 threads, i.e., all threads in the first warp and one thread in the second warp.
`Instruction 3` is executed by the remaining active threads.

The SIMD utilization sums to $\frac{1025+32+1+31+960+1}{1056+32+32+32+960+32} = \frac{2050}{2144} = \frac{1025}{1072}$.

7 BossMem [50 points]

A researcher has developed a new type of nonvolatile memory, BossMem. He is considering BossMem as a replacement for DRAM. BossMem is 10x faster (all memory timings are 10x faster) than DRAM, but since BossMem is so fast, it has to frequently power-off to cool down. Overheating is only a function of time, not a function of activity. An idle stick of BossMem has to power-off just as frequently as an active stick. When powered-off, BossMem retains its data, but cannot service requests. Both DRAM and BossMem are banked and otherwise architecturally similar. To the researcher's dismay, he finds that a system with 1GB of DRAM performs considerably better than the same system with 1GB of BossMem.

(i) What can the researcher change or improve in the core (he can't change BossMem or anything beyond the memory controller) that will make his BossMem perform more favorably compared to DRAM, realizing that he will have to be fair and evaluate DRAM with his enhanced core as well? (15 words or less)

Solution: Prefetcher degree or other speculation techniques so that misses can be serviced before memory powered off.

(ii) A colleague proposes he builds a hybrid memory system, with both DRAM and BossMem. He decides to place data that exhibits low row buffer locality in DRAM and data that exhibits high row buffer locality in BossMem. Assume 50% of requests are row buffer hits. Is this a good or bad idea? Show your work.

Solution: No, it may be better idea to place data with high row buffer locality in DRAM and low row buffer locality data in BossMem since row buffer misses are less costly.

(iii) Now a colleague suggests trying to improve the last-level cache replacement policy in the system with the hybrid memory system. Like before, he wants to improve the performance of this system relative to one that uses just DRAM and he will have to be fair in his evaluation. Can he design a cache replacement policy that makes the hybrid memory system look more favorable? In 15 words or less, justify NO or describe a cache replacement policy that would improve the performance of the hybrid memory system more than it would DRAM.

Solution: Yes, this is possible. Cost-based replacement where cost to replace is dependent on data allocation between DRAM and BossMem.

(iv) In class we talked about another nonvolatile memory technology, phase-change memory (PCM). Which technology, PCM, BossMem, or DRAM requires the greatest attention to security? What is the vulnerability?

Solution: PCM is nonvolatile and has potential endurance attacks.

(v) Which is likely of least concern to a security researcher?

Solution: DRAM is likely least vulnerable, as BossMem also has nonvolatility concerns.