Computer Architecture (263-2210-00L), Fall 2018

HW 3: SIMD Processing, Branch Handling, Emerging Memory Technologies, and PIM

SOLUTIONS

Instructor: Prof. Onur Mutlu
TAs: Mohammed Alser, Can Firtina, Hasan Hassan, Jeremie Kim, Juan Gómez Luna, Geraldo Francisco de Oliveira, Minesh Patel, Giray Yaglikci

Assigned: Friday, Oct 26, 2018
Due: **Sunday, Nov 11, 2018**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture18/`. 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 (2-9).** 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 [150 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/fall2018/lib/exe/fetch.php?media=onur-comparch-f18-how-to-do-the-paper-reviews.pdf`
- Some sample reviews can be found here: `https://safari.ethz.ch/architecture/fall2018/doku.php?id=readings`

(a) Write a one-page critical review for the following paper:
B. C. Lee, E. Ipek, O. Mutlu and D. Burger. "Architecting phase change memory as a scalable DRAM alternative." ISCA 2009. `https://people.inf.ethz.ch/omutlu/pub/pcm_isca09.pdf`

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

- McFarling, Scott. "Combining branch predictors". Vol. 49. Technical Report TN-36, Digital Western Research Laboratory, 1993. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=combining.pdf`
- Yeh, Tse-Yu, and Yale N. Patt. "Two-level adaptive training branch prediction." Proceedings of the 24th annual international symposium on Microarchitecture. ACM, 1991. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=yeh_patt-adaptive-training-1991.pdf`
- Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., and Glasco, D. "GPUs and the future of parallel computing." IEEE Micro, 2011. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=ieee-micro-gpu.pdf`

2 Vector Processing I [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:

```
VLD     |-----50-----|---(VLEN-1)---|
VLD     |1|-----50-----|---(VLEN-1)---|
VADD                                 |-4-|--(VLEN-1)---|
```

```
VMUL                                                    |-16-|--(VLEN-1)---|
VRSHF                                                                       |1|--(VLEN-1)--|
```

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

(c) The architect of this machine decides that she needs to cut costs in the 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)?

Solution:

```
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]: $1 + 50 + 50 + 7) + 4 + 16 + 1 = 129$ 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:

```
VLD   [0]       |---50---|
      ...
      [8]               |---50---|
      ...
      [16]                      |--50--|
      ...
      [24]                            |--50--|
      ...
      [32]                                  |--50--|
      ...
      [39]                                        |--7--|
VLD   [39]                                              |1|
VADD                                                      |--4--|
VMUL                                                            |--16--|
VRSHF                                                                   |1|
```

$5{\times}50 + 7 + 1 + 4 + 16 + 1 = 279$ 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.

3 Vector Processing II [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.

Solution:

```
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     R8, R7, R5  // 4 cycles
RSHFA   R9, R8, R1  // 1 cycle
ST      R9, 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?

Solution:

$5 + 100 \times 60 = 6005$ cycles

(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	

Solution:

```
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).

Solution:

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 |
```

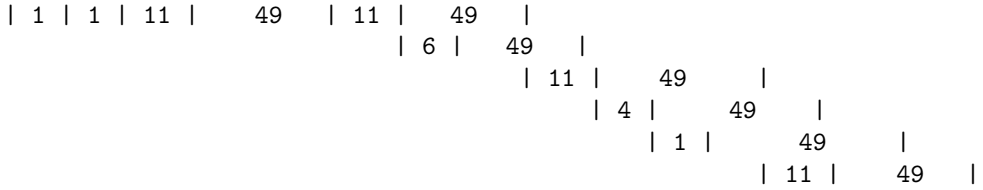
Processing the next 50 elements takes 344 cycles as shown below (no need to initialize Vln and Vst as they stay at the same value).

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

Therefore, the total number of cycles to execute the program = 690 cycles

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

Solution: 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



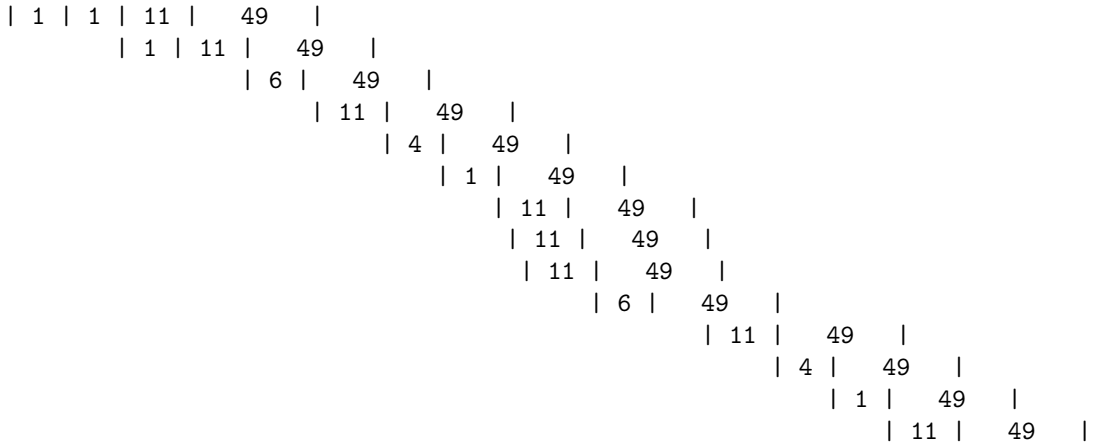
Processing the next 50 elements takes 240 cycles (same time line as above, but without the first 2 instructions to initialize Vln and Vst).

Therefore, the total number of cycles to execute the program = 482 cycles

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

Solution:

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 = 215 cycles

4 Branch Prediction I [100 points]

Assume the following piece of code that iterates through a large array populated with **completely (i.e., truly) random** positive integers. The code has four branches (labeled `B1`, `B2`, `B3`, and `B4`). When we say that a branch is *taken*, we mean that the code *inside* the curly brackets is executed.

```
for (int i=0; i<N; i++) { /* B1 */
  val = array[i];        /* TAKEN PATH for B1 */
  if (val % 2 == 0) {    /* B2 */
    sum += val;         /* TAKEN PATH for B2 */
  }
  if (val % 3 == 0) {    /* B3 */
    sum += val;         /* TAKEN PATH for B3 */
  }
  if (val % 6 == 0) {    /* B4 */
    sum += val;         /* TAKEN PATH for B4 */
  }
}
```

(a) Of the four branches, list all those that exhibit *local correlation*, if any.

Only `B1`.

`B2, B3, B4` are not locally correlated. Just like consecutive outcomes of a die, an element being a multiple of N (N is 2, 3, and 6, respectively for `B2`, `B3`, and `B4`) has no bearing on whether the next element is also a multiple of N.

(b) Which of the four branches are *globally correlated*, if any? Explain in less than 20 words.

`B4` is correlated with `B2` and `B3`. 6 is a common multiple of 2 and 3.

Now assume that the above piece of code is running on a processor that has a global branch predictor. The global branch predictor has the following characteristics.

- Global history register (GHR): 2 bits.
- Pattern history table (PHT): 4 entries.
- Pattern history table entry (PHTE): 11-bit signed saturating counter (possible values: -1024–1023)
- Before the code is run, all PHTEs are initially set to 0.
- As the code is being run, a PHTE is incremented (by one) whenever a branch that corresponds to that PHTE is taken, whereas a PHTE is decremented (by one) whenever a branch that corresponds to that PHTE is not taken.

(c) After 120 iterations of the loop, calculate the **expected** value for only the first PHTE and fill it in the shaded box below. (Please write it as a base-10 value, rounded to the nearest one's digit.)

Hint. For a given iteration of the loop, first consider, what is the probability that both B1 and B2 are taken? Given that they are, what is the probability that B3 will increment or decrement the PHTE? Then consider...

Show your work.

Without loss of generality, let's take a look at the numbers from 1 through 6. Given that a number is a multiple of two (i.e., 2, 4, 6), the probability that the number is also a multiple of three (i.e., 6) is equal to 1/3, let's call this value Q. Given that a number is a multiple of two and three (i.e., 6), the probability that the number is also a multiple of six (i.e., 6) is equal to 1, let's call this value R.

For a **single** iteration of the loop, the PHTE has four chances of being incremented/decremented, once at each branch.

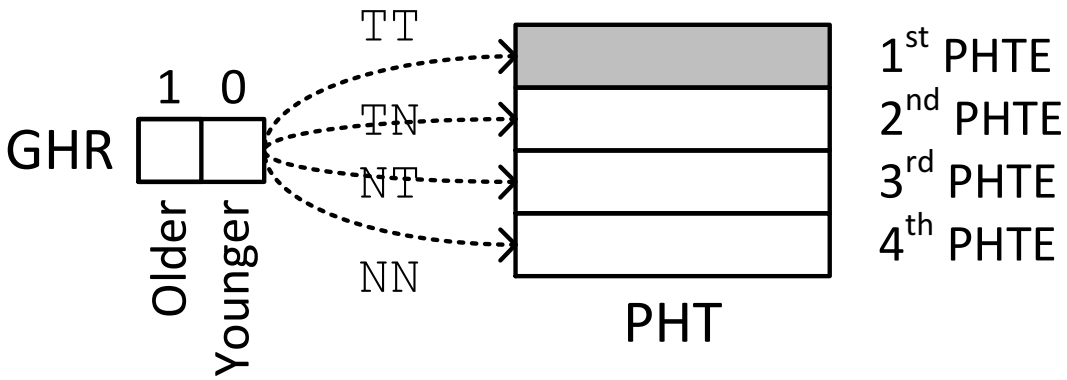
• B3's contribution to PHTE. The probability that both B1 and B2 are taken is denoted as P(B1_T && B2_T), which is equal to P(B1_T)*P(B2_T) = 1*1/2 = 1/2. Given that they are, the probability that B3 is taken, is equal to Q = 1/3. Therefore, the PHTE will be incremented with probability 1/2*1/3 = 1/6 and decremented with probability 1/2*(1-1/3) = 1/3. The net contribution of B3 to PHTE is 1/6-1/3 = -1/6.

• B4's contribution to PHTE. P(B2_T && B3_T) = 1/6. P(B4_T | B2_T && B3_T) = R = 1. B4's net contribution is 1/6*1 = 1/6.

• B1's contribution to PHTE. P(B3_T && B4_T) = 1/6. P(B1_T | B3_T && B4_T) = 1. B1's net contribution is 1/6*1 = 1/6.

• B2's contribution to PHTE. P(B4_T && B1_T) = 1/6*1 = 1/6. P(B2_T | B4_T && B1_T) = 1/2. B2's net contribution is 1/6*1/2 - 1/6*1/2 = 0.

For a single iteration, the net contribution to the PHTE, summed across all the four branches, is equal to 1/6. Since there are 120 iterations, the expected PHTE value is equal to 1/6*120=**20**.



5 Branch Prediction II [100 points]

Suppose we have the following loop executing on a pipelined MIPS machine.

```
DOIT  SW    R1, 0(R6)
      ADDI  R6, R6, 2
      AND   R3, R1, R2
      BEQ   R3, R0  EVEN
      ADDI  R1, R1, 3
      ADDI  R5, R5, -1
      BGTZ  R5  DOIT
EVEN  ADDI  R1, R1, 1
      ADDI  R7, R7, -1
      BGTZ  R7 DOIT
```

Assume that before the loop starts, the registers have the following decimal values stored in them:

Register	Value
R0	0
R1	0
R2	1
R3	0
R4	0
R5	5
R6	4000
R7	5

The fetch stage takes one cycle, the decode stage also takes one cycle, the execute stage takes a variable number of cycles depending on the type of instruction (see below), and the store stage takes one cycle.

All execution units (including the load/store unit) are fully pipelined and the following instructions that use these units take the indicated number of cycles:

Instruction	Number of Cycles
SW	3
ADDI	2
AND	3
BEQ/BGTZ	1

Data forwarding is used wherever possible. Instructions that are dependent on the previous instructions can make use of the results produced right after the previous instruction finishes the execute stage.

The target instruction after a branch can be fetched when the branch instruction is in ST stage. For example, the execution of an AND instruction followed by a BEQ would look like:

```
AND       F | D | E1 | E2 | E3 | ST
BEQ           F | D  | -  | -  | E1 | ST
TARGET                              F  | D
```

A scoreboarding mechanism is used.

Answer the following questions:

1. How many cycles does the above loop take to execute if no branch prediction is used (the pipeline stalls on fetching a branch instruction, until it is resolved)?

   Solution:
   The first iteration of the DOIT loop takes 15 cycles as shown below:

```
F | D | E1 | E2 | E3 | ST |
    F | D  | E1 | E2 | ST |
        F  | D  | E1 | E2 | E3 | ST |
             F  | D  | -  | -  | E1 | ST |
                                  F  | D  | E1 | E2 | ST |
                                       F  | D  | E1 | E2 | ST |
                                            F  | D  | -  | E1 | ST |
```

   The rest of the iterations each take 14 cycles, as the fetch cycle of the SW instruction can be overlapped with the ST stage of the BGTZ DOIT branch.
   There are 9 iterations in all as the loop execution ends when R7 is zero and R5 is one.

   Total number of cycles $= 15 + (14 \times 8) = 127$ cycles

2. How many cycles does the above loop take to execute if all branches are predicted with 100% accuracy?

   Solution:
   The first iteration of the DOIT loop takes 13 cycles as shown below:

```
F | D | E1 | E2 | E3 | ST |
    F | D  | E1 | E2 | ST |
        F  | D  | E1 | E2 | E3 | ST |
             F  | D  | -  | -  | E1 | ST |
                  F  | -  | -  | D  | E1 | E2 | ST |
                                  F  | D  | E1 | E2 | ST |
                                       F  | D  | -  | E1 | ST |
```

   The rest of the iterations each take 10 cycles, as the first three stages of the SW instruction can be overlapped with the execution of the BGTZ DOIT branch instruction.

   Total number of cycles $= 13 + (10 \times 8) = 93$ cycles

3. How many cycles does the above loop take to execute if a static BTFN (backward taken-forward not taken) branch prediction scheme is used to predict branch directions? What is the overall branch prediction accuracy? What is the prediction accuracy for each branch?

   Solution:
   The first iteration of the DOIT loop takes 15 cycles as the BEQ EVEN branch is predicted wrong the first time.

```
F | D | E1 | E2 | E3 | ST |
    F | D  | E1 | E2 | ST |
        F  | D  | E1 | E2 | E3 | ST |
             F  | D  | -  | -  | E1 | ST |
                                  F  | D | E1 | E2 | ST |
                                       F | D  | E1 | E2 | ST |
                                           F  | D  | -  | E1 | ST |
```

Of the remaining iterations, the BEQ EVEN branch is predicted right 4 times, while it is mispredicted the remaining four times.

The DOIT branch is predicted right all times.

Number of cycles taken by an iteration when the BEQ EVEN branch is predicted right = 10 cycles
Number of cycles taken by an iteration when the BEQ EVEN branch is not predicted right = 12 cycles

Total number of cycles = $15 + (10 \times 4) + (12 \times 4) = 103$ cycles

The BEQ EVEN branch is mispredicted 5 times out of 9. So, the prediction accuracy is 4/9.
The first BGTZ DOIT branch is predicted right 4 times out of 4. So, the prediction accuracy is 4/4.
The second BGTZ DOIT branch is predicted right 4 times out of 5. So, the prediction accuracy is 4/5.

Therefore the overall prediction accuracy is 12/18.

6 Interference in Two-Level Branch Predictors [50 points]

Assume a two-level global predictor with a global history register and a single pattern history table shared by all branches (call this "predictor A").

1. We call the notion of different branches mapping to the same locations in a branch predictor "branch interference". Where do different branches interfere with each other in these structures?

   Solution:
   Global history register (GHR), Pattern history table (PHT)

2. Compared to a two-level global predictor with a global history register and a separate pattern history table for each branch (call this "predictor B"),

   (a) When does predictor A yield lower prediction accuracy than predictor B? Explain. Give a concrete example. If you wish, you can write source code to demonstrate a case where predictor A has lower accuracy than predictor B.

   Solution:
   Predictor A yields lower prediction accuracy when two branches going in opposite directions are mapped to the same PHT entry. Consider the case of a branch B1 which is always-taken for a given global history. If branch B1 had its own PHT, it would always be predicted correctly. Now, consider a branch B2 which is always-not-taken for the same history. If branch B2 had its own PHT, it would also be predicted right always. However, if branches B1 and B2 shared a PHT, they would map to the same PHT entry and hence, interfere with each other and degrade each other's prediction accuracy.

   Consider a case when the global history register is 3 bits wide and indexes into a 8-entry pattern history table and the following code segment:

```
for (i = 0; i < 1000; i ++)
{
if (i % 2 == 0) //IF CONDITION 1
{
........
}


if (i % 3 == 0) // IF CONDITION 2
{
........
}
}
```

   For a global history of "NTN", IF CONDITION 1 is taken, while IF CONDITION 2 is not-taken. This causes destructive interference in the PHT.

   (b) Could predictor A yield higher prediction accuracy than predictor B? Explain how. Give a concrete example. If you wish, you can write source code to demonstrate this case.

   Solution:
   This can happen if the predictions for a branch B1 for a given history become more accurate when another branch B2 maps to the same PHT entry whereas the predictions would not have been accurate had the branch had its own PHT. Consider the case in which branch B1 is always mispredicted for a given global history (when it has its own PHT) because it happens to oscillate between taken and not taken for that history. Now consider an always-taken branch B2 mapping

to the same PHT entry. This could improve the prediction accuracy of branch B1 because now B1 could always be predicted taken since B2 is always taken. This may not degrade the prediction accuracy of B2 if B2 is more frequently executed than B1 for the same history. Hence, overall prediction accuracy would improve.

Consider a 2-bit global history register and the following code segment.

```
if (cond1) { }
if (cond2) { }
if ((a % 4) == 0) {}  //BRANCH 1
if (cond1) { }
if (cond2) { }
if ((a % 2) == 0) {}  //BRANCH 2
```

BRANCH 2 is strongly correlated with BRANCH 1, because when BRANCH 1 is taken BRANCH 2 is always taken. Furthermore, the two branches have the same history leading up to them. Therefore, BRANCH 2 can be predicted accurately based on the outcome of BRANCH 1, even if BRANCH 2 has not been seen before.

(c) Is there a case where branch interference in predictor structures does not impact prediction accuracy? Explain. Give a concrete example. If you wish, you can write source code to demonstrate this case as well.

Solution:
Predictor A and B yield the same prediction accuracy when two branches going in the same direction are mapped to the same PHT entry. In this case, the interference between the branches does not impact prediction accuracy. Consider two branches B1 and B2 which are always-taken for a certain global history. The prediction accuracy would be the same regardless of whether B1 and B2 have their own PHTs or share a PHT.

Consider a case when the global history register is 3 bits wide and indexes into a 8 entry pattern history table and the following code segment:

```
for (i = 0; i < 1000; i += 2) //LOOP BRANCH
{
if (i % 2 == 0) //IF CONDITION
{
........
}

}
```

LOOP BRANCH and IF CONDITION are both taken for a history of "TTT". Therefore, although these two branches map to the same location in the pattern history table, the interference between them does not impact prediction accuracy.

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.

8 In-DRAM Bitmap Indices [100 points]

Recall that in class we discussed Ambit, which is a DRAM design that can greatly accelerate Bulk Bitwise Operations by providing the ability to perform bitwise AND/OR of two rows in a subarray.

One real-world application that can benefit from Ambit's in-DRAM bulk bitwise operations is the database *bitmap index*, as we also discussed in the lecture. By using bitmap indices, we want to run the following query on a database that keeps track of user actions: "How many unique users were active every week for the past w weeks?" Every week, each user is represented by a single bit. If the user was active a given week, the corresponding bit is set to 1. The total number of users is u.

We assume the bits corresponding to one week are all in the same row. If u is greater than the total number of bits in one row (the row size is 8 kilobytes), more rows in different subarrays are used for the same week. We assume that all weeks corresponding to the users in one subarray fit in that subarray.

We would like to compare two possible implementations of the database query:

- *CPU-based implementation*: This implementation reads the bits of all u users for the w weeks. For each user, it `and`s the bits corresponding to the past w weeks. Then, it performs a bit-count operation to compute the final result.
  Since this operation is very memory-bound, we simplify the estimation of the execution time as the time needed to read all bits for the u users in the last w weeks. The memory bandwidth that the CPU can exploit is X bytes/s.
- *Ambit-based implementation*: This implementation takes advantage of bulk `and` operations of Ambit. In each subarray, we reserve one *Accumulation* row and one *Operand* row (besides the control rows that are needed for the regular operation of Ambit). Initially, all bits in the *Accumulation* row are set to 1. Any row can be moved to the *Operand* row by using RowClone (recall that RowClone is a mechanism that enables very fast copying of a row to another row in the same subarray). t_{rc} and t_{and} are the latencies (in seconds) of RowClone's `copy` and Ambit's `and` respectively.
  Since Ambit does *not* support bit-count operations inside DRAM, the final bit-count is still executed on the CPU. We consider that the execution time of the bit-count operation is negligible compared to the time needed to read all bits from the *Accumulation* rows by the CPU.

(a) What is the total number of DRAM rows that are occupied by u users and w weeks?

$TotalRows = \lceil \frac{u}{8 \times 8k} \rceil \times w.$

Explanation:
The u users are spread across a number of subarrays:
$NumSubarrays = \lceil \frac{u}{8 \times 8k} \rceil$.

Thus, the total number of rows is:
$TotalRows = \lceil \frac{u}{8 \times 8k} \rceil \times w.$

(b) What is the throughput in users/second of the Ambit-based implementation?

$Thr_{Ambit} = \frac{u}{\lceil \frac{u}{8\times 8k} \rceil \times w \times (t_{rc}+t_{and}) + \frac{u}{X\times 8}}$ users/second.

Explanation:
First, let us calculate the total time for all bulk `and` operations. We should add t_{rc} and t_{and} for all rows:
$t_{and-total} = \lceil \frac{u}{8\times 8k} \rceil \times w \times (t_{rc} + t_{and})$ seconds.

Then, we calculate the time needed to compute the bit count on CPU:
$t_{bitcount} = \frac{\frac{u}{8}}{X} = \frac{u}{X\times 8}$ seconds.

Thus, the throughput in users/s is:
$Thr_{Ambit} = \frac{u}{t_{and-total}+t_{bitcount}}$ users/second.

(c) What is the throughput in users/second of the CPU implementation?

$Thr_{CPU} = \frac{X\times 8}{w}$ users/second.

Explanation:
We calculate the time needed to bring all users and weeks to the CPU:
$t_{CPU} = \frac{\frac{u\times w}{8}}{X} = \frac{u\times w}{X\times 8}$ seconds.

Thus, the throughput in users/s is:
$Thr_{CPU} = \frac{u}{t_{CPU}} = \frac{X\times 8}{w}$ users/second.

(d) What is the maximum w for the CPU implementation to be faster than the Ambit-based implementation? Assume u is a multiple of the row size.

$w < \frac{1}{1-\frac{X}{8k}\times(t_{rc}+t_{and})}$.

Explanation:
We compare t_{CPU} with $t_{and-total} + t_{bitcount}$:
$t_{CPU} < t_{and-total} + t_{bitcount}$;

$\frac{u\times w}{X\times 8} < \frac{u}{8\times 8k} \times w \times (t_{rc} + t_{and}) + \frac{u}{X\times 8}$;

$w < \frac{1}{1-\frac{X}{8k}\times(t_{rc}+t_{and})}$.

9 Caching vs. Processing-in-Memory [100 points]

We are given the following piece of code that makes accesses to integer arrays A and B. The size of each element in both A and B is 4 bytes. The base address of array A is $0x00001000$, and the base address of B is $0x00008000$.

```
movi R1, #0x1000 // Store the base address of A in R1
movi R2, #0x8000 // Store the base address of B in R2
movi R3, #0

Outer_Loop:
    movi R4, #0
    movi R7, #0
    Inner_Loop:
        add R5, R3, R4  // R5 = R3 + R4
        // load 4 bytes from memory address R1+R5
        ld R5, [R1, R5] // R5 = Memory[R1 + R5],
        ld R6, [R2, R4] // R6 = Memory[R2 + R4]
        mul R5, R5, R6  // R5 = R5 * R6
        add R7, R7, R5  // R7 += R5
        inc R4          // R4++
        bne R4, #2, Inner_Loop // If R4 != 2, jump to Inner_Loop

    //store the data of R7 in memory address R1+R3
    st [R1, R3], R7     // Memory[R1 + R3] = R7,
    inc R3              // R3++
    bne R3, #16, Outer_Loop // If R3 != 16, jump to Outer_Loop
```

You are running the above code on a single-core processor. For now, assume that the processor *does not* have caches. Therefore, all load/store instructions access the main memory, which has a fixed 50-cycle latency, for both read and write operations. Assume that all load/store operations are serialized, i.e., the latency of multiple memory requests *cannot* be overlapped. Also assume that the execution time of a non-memory-access instruction is zero (i.e., we ignore its execution time).

(a) What is the execution time of the above piece of code in cycles?

> 4000 cycles.
>
> **Explanation:** There are 5 memory accesses for each outer loop iteration. The outer loop iterates 16 times, and each memory access takes 50 cycles.
> $16 * 5 * 50 = 4000$ cycles

(b) Assume that a 128-byte private cache is added to the processor core in the next-generation processor. The cache block size is 8-byte. The cache is direct-mapped. On a hit, the cache services both read and write requests in 5 cycles. On a miss, the main memory is accessed and the access fills an 8-byte cache line in 50 cycles. Assuming that the cache is initially empty, what is the new execution time on this processor with the described cache? Show your work.

> 900 cycles.
>
> **Explanation.**
>
> At the beginning A and B conflict in the first two cache lines. Then the elements of A and B go to

different cache lines. The total execution time is 1910 cycles.

Here is the access pattern for the first outer loop iteration:

$0 - A[0], B[0], A[1], B[1], A[0]$

The first 4 references are loads, the last (A[0]) is a store. The cache is initially empty. We have a cache miss for A[0]. A[0] and A[1] is fetched to 0th index in the cache. Then, B[0] is a miss, and it is conflicting with A[0]. So, A[0] and A[1] are evicted. Similarly, all cache blocks in the first iteration are conflicting with each other. Since we have only cache misses, the latency for those 5 references is $5 * 50 = 250$ cycles

The status of the cache after making those seven references is:

Cache Index	Cache Block
0	~~A(0,1)~~, ~~B(0,1)~~, ~~A(0,1)~~, ~~B(0,1)~~, A(0,1)

Second iteration on the outer loop:
$1 - A[1], B[0], A[2], B[1], A[1]$

Cache hits/misses in the order of the references:
H, M, M, H, M
$Latency = 2 * 5 + 3 * 50 = 165$ cycles

Cache Status:
- A(0,1) is in set 0
- A(2,3) is in set 1
- the rest of the cache is empty

$2 - A[2], B[0], A[3], B[1], A[2]$

Cache hits/misses:
H, M, H, H, H
$Latency : 4 * 5 + 1 * 50 = 70$ cycles

Cache Status:
- B(0,1) is in set 0
- A(2,3) is in set 1
- the rest of the cache is empty

$3 - A[3], B[0], A[4], B[1], A[3]$

Cache hits/misses:
H, H, M, H, H
$Latency : 4 * 5 + 1 * 50 = 70$ cycles

Cache Status:
- B(0,1) is in set 0
- A(2,3) is in set 1
- A(4,5) is in set 2
- the rest of the cache is empty

$4 - A[4], B[0], A[5], B[1], A[4]$

Cache hits/misses:
H, H, H, H, H
$Latency : 5 * 5 = 25$ cycles

Cache Status:
- B(0,1) is in set 0
- B(2,3) is in set 1
- A(4,5) is in set 2
- the rest of the cache is empty

After this point, single-miss and zero-miss (all hits) iterations are interleaved until the 16th iteration.

Overall Latency:
$165 + 70 + (70 + 25) * 7 = 900$ cycles

(c) You are not satisfied with the performance after implementing the described cache. To do better, you consider utilizing a processing unit that is available *close to the main memory*. This processing unit can directly interface to the main memory with a *10-cycle* latency, for both read and write operations. How many cycles does it take to execute the same program using the in-memory processing units? (Assume that the in-memory processing unit does not have a cache, and the memory accesses are serialized like in the processor core. The latency of the non-memory-access operations is ignored.)

800 cycles.

Explanation: Same as for the processor core without a cache, but the memory access latency is 10 cycles instead of 50. $16 * 5 * 10 = 800$

(d) You friend now suggests that, by changing the cache capacity of the single-core processor (in part (b)), she could provide as good performance as the system that utilizes the memory processing unit (in part (c)).

Is she correct? What is the minimum capacity required for the cache of the single-core processor to match the performance of the program running on the memory processing unit?

No, she is not correct.

Explanation: Increasing the cache capacity does not help because doing so cannot eliminate the conflicts to Set 0 in the first two iterations of the outer loop.

(e) What other changes could be made to the cache design to improve the performance of the single-core processor on this program?

Increasing the associativity of the cache.

Explanation: Although there is enough cache capacity to exploit the locality of the accesses, the fact that in the first two iterations the accessed data map to the same set causes conflicts. To improve the hit rate and the performance, we can change the address-to-set mapping policy. For example, we can change the cache design to be set-associative or fully-associative.