CMU 15-418/618: Parallel Computer Architecture and Programming
Practice Exercise 6

Problem 1: Bringing Locality Back

Justin Timberlake and Kanye West hear that Spark is all the rage and decide they are going to code up their own implementation to compete against that of the Apache project. Justin's first test runs the following Spark program, which creates four RDDs. The program takes Justin's lengthy (1 TB!) list of dancing tips and finds all misspelled words.

```
var lines = spark.textFile("hdfs://mydancetips.txt'');     // 1 TB file
var lower = lines.map( x => x.toLower() );                 // convert lines to lower case
var words = lower.flatmap( x => x.split(`` '') );          // convert RDD of lines to RDD of
                                                           // individual words
var misspelled = words.filter( x => !x.isInDictionary() ); // filter to find misspellings

print misspelled.count();   // print number of misspelled words
```

A. (3 pts) Understanding that the Spark RDD abstraction affords many possible implementations, Justin decides to keep things simple and implements his Spark runtime such that each RDD is implemented by a fully allocated array. This array is stored either in memory or on disk depending on the size of the RDD and available RAM. **The array is allocated and populated at the time the RDD is created — as a result of executing the appropriate operator (map, flatmap, filter, etc.) on the input RDD.**

   Justin runs his program on a cluster with 10 computers, each of which has 100 GB of memory. The program gets correct results, but Justin is devastated because the program runs *incredibly slow*. He calls his friend Taylor Swift, ready to give up on the venture. Encouragingly, Taylor says, "shake it off Justin", just run your code on 40 computers. Justin does this and observes a speedup much greater than $4\times$ his original performance. Why is this the case?

B. (3 pts) With things looking good, Kanye runs off to write a new single "All of the Nodes" to use in the marketing for their product. At that moment, Taylor calls back, and says "Actually, Justin, I think you can schedule the computations much more efficiently and get very good performance with less memory and far fewer nodes." Describe how you would change how Justin schedules his Spark computations to improve memory efficiency and performance.

C. (4 pts) After hacking all night, the next day, Justin, Kanye, and Taylor run the optimized program on 10 nodes. The program runs for 1 hour, and then right before `misspelled.count()` returns, node 6 crashes. Kanye is irate! He runs onto the machine room floor, pushing Taylor aside and says, "Taylor, I have a single to release, and I don't have time to deal with rerunning your programs from scratch. Geez, I already made you famous." Taylor gives Kayne a stink eye and says, "Don't worry, it will be complete in just a few minutes." Approximately how long will it take after the crash for the program to complete? You should assume the `.count()` operation is essentially free. But please **clearly state any assumptions about how the computation is scheduled in justifying your answer.**

Problem 2: Controlling DRAM

Consider a computer with a single DIMM containing eight 1 MB DRAM chips (8 MB total capacity). Each DRAM chip has one bank and row size 2 kilobits (256 bytes). As discussed in class, the DIMM is connected to the memory controller via a 64-bit bus, with 8-bits per cycle transferred from each chip.

Assume that:

- Contiguous 1 MB regions of the physical address space are mapped to a single DRAM chip. 256 consecutive physical address space bytes are in a row. 1048576 consecutive bytes fill a DRAM chip.
- Physical address 0 maps to chip 0, row 0, column 0. Physical address 1048576 maps to chip 1, row 0, column 0, etc.

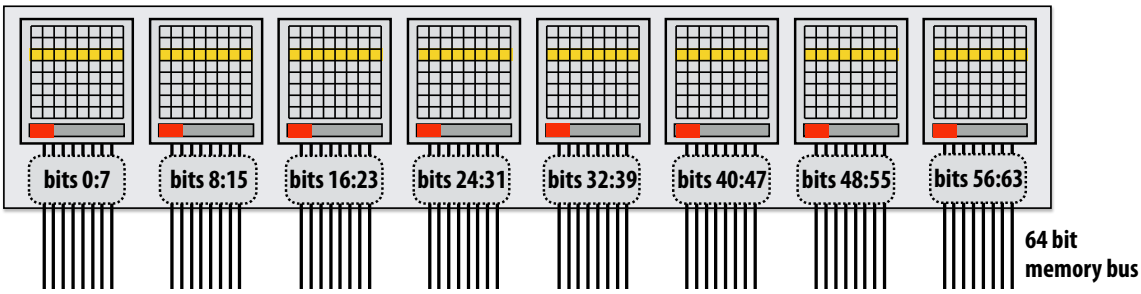
Given these assumptions, reading a 64-byte cache line beginning at address X requires the following memory-controller logic, presented in C code below: (the data ends up in `cache_line`)

```
char cache_line[64];

// compute DRAM clip, row, col for address X
int chip = X / 1048576;
int row =  (X % 1048576) / 256;
int col =  (X % 1048576) % 256

for (int i=0; i<64; i++) {

   // Read one byte from each DRAM chip at given row and column (eight in total)
   // so that the byte from chip j ends up in 'from_dram[j]'. Assume necessary
   // DRAM row and column activations are performed inside DIMM_READ_FROM_CHIPS.

   char from_dram[8];
   DIMM_READ_FROM_CHIPS(row, column, from_dram);

   cache_line[i] = from_dram[chip];

   column++;  // move to next byte in column
}
```

Questions are on next page...

A. (2 pts) Explain why 64 iterations (64 reads from the DRAM chips) are required to populate the buffer `cache_line`.

B. (3 pts) Now assume the address space is **byte-interleaved across the DRAM chips** as discussed in class and shown in the Figure below. (Byte X in the address space is stored on chip X % 8.) Please provide C-like pseudocode for reading the 64-byte cache line at address X from DRAM into `cache_line`. Your code should make a series of calls to `DIMM_READ_FROM_CHIPS`.



Hint: Recall that each DRAM chip row is 256 bytes.

C. (1 pts) How much higher "effective bandwidth" is achieved using the interleaved mapping from part B than the original blocked mapping from part A?

D. (2 pts) Imagine the byte interleaved memory system from part B is connected to a dual-core CPU. The memory controller uses a naive **round-robin policy** to schedule incoming memory requests from the cores (it services a request from core 0, then core 1, then core 0, etc.) All requests from the same core are processed in FIFO order.

Both cores execute the following C code **on different 4MB arrays.** Simply put, each thread is linearly scanning through different regions of memory.

```
int A[N];           // let N = 1M, so this array is 4MB
int sum = 0;        // assume 'sum' is register allocated
for (int i=0; i<N; i++)
   sum += A[i];
```

Assume that the cores request data from memory at granularity of 8 bytes. On this system, you observe that when running two threads, the overall aggregate bandwidth from the memory system *is lower* than when one thread is executing the same code. Why might this be the case? (Hint: we are looking for an answer the pertains to DRAM chip behavior: consider locality, but which kind?)

E. (2 pts) Why might performance improve if the granularity of each memory access was increased from 8 to 64 bytes?