Assignment #1

You can do this assignment in a Group of at most 2 students. Due: Oct. 12, 2022

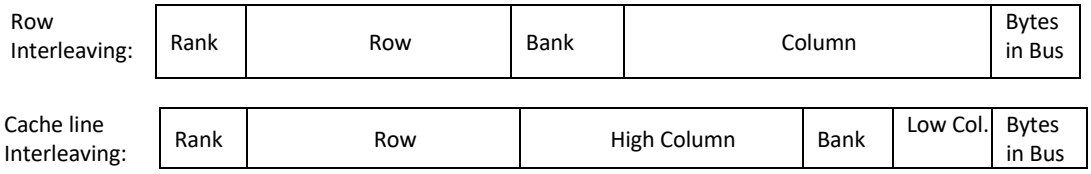
1. In this exercise you will evaluate six different versions of the matrix multiplication program. You are required to use the standard $O(n^3)$ matrix multiplication algorithm. The first variant uses `(i,j,k)` as the loop order. The next two variant use `(i,k,j)` and `(k,i,j)` loop orders. Then for each of these variants, create a corresponding blocked/tiled version with a tile size of 32. An example tiled version of the `(i,j,k)` loop is shown below.

```
for(i=0; i<N; i=i+B)
   for(j=0; j<N; j=j+B){
      A[i][j] = 0.0;
      for (k=0; k < N; k=k+B)
         for(ii=i; ii<i+B; ii++)
            for(jj=j; jj < j+B; jj++){
               for(kk=k; kk < k+B; kk++){
                  A[ii][jj] += B[ii][kk] * C[kk][jj];
   }
```

   You will measure the performance of these different versions on two different matrix sizes of (2048 x 2048) and (8192 x 8192). You will obtain the counts of various events such as the L1/L2/L3 cache misses, TLB misses, page faults, etc. from the Performance Monitoring Counters (PMCs). Performance counter values for running application can be obtained using performance monitoring tools such as PAPI [1] or perf [2].

   a) Correlate the execution time of different versions of the program with the memory hierarchy performance (misses at various levels of memory hierarchy).
   b) Based on your observation(s), explore different tile sizes (powers of 2) and experimentally obtain the one that achieves the best performance.

2. In this exercise you will evaluate five different memory access scheduling methods on a set of microbenchmark programs. You can run each of these benchmarks on your laptop/desktop and generate cache-filtered memory traces – memory accesses that miss in L1/L2/L3 caches and reach DRAM – of all loads and stores in the program. The traces can be generated using `perf` [2] or PIN [3] tool. Make sure you generate the traces with the timestamp (cycle at which the load/store request is generated). The traces are then run on the DRAMSim3 simulator. You will explore a memory configuration with a single memory controller, 1 DIMM, 2 ranks, 8 chips per rank (to fetch 8 bytes in parallel), 32 banks in each chip, 16K rows per chip and 2K columns per row. Consider two different addressing schemes one in which consecutive cache lines are served from the same row (row interleaving) and another where they are served from consecutive memory banks (cache line interleaving). The addressing schemes are as shown below:



For DRAM timing parameters (such as the t_{RCD}, t_{RAS}, t_{CL}, t_{RP}, etc), use the default values provided in the DRAMSimulator.

The memory access scheduling methods to be evaluated are: FCFS, Open, Closed, FR-FCFS, and Open-4 policies, where Open-4 policy is similar to Open policy but closes the page after at most 4 hits. More specifically, in the Open-4 policy, a bank is precharged if either (i) there are pending references to other rows in the bank and there are no pending references to the active row or (ii) 4 requests have been satisfied after the row was opened. While the first four policies would be available in the DRAM simulator, you are expected to implement the Open-4 policy.

Evaluate the performance of the given three microbenchmarks and analyse the performance of different memory scheduling schemes and the addressing schemes. In particular your analysis should include comparisons of row-buffer hit rates, bank-level parallelism, average memory access time, bandwidth exploited, etc.

[1] PAPI User's Guide. Available at: http://icl.cs.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE_23.htm

[2] Linux Kernel Profiling with perf. Available at: https://perf.wiki.kernel.org/index.php/Tutorial

[3] Pin - A Dynamic Binary Instrumentation Tool. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html

[4] S. Li, Z. Yang, D. Reddy, A. Srivastava and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," in IEEE Computer Architecture Letters, vol. 19, no. 2, pp. 106-109, 1 July-Dec. 2020, doi: 10.1109/LCA.2020.2973991.
Available at: https://github.com/umd-memsys/DRAMsim3