Problem 1

Below are some plots of cache miss rates from four applications in the PARSEC benchmark suite. The cache modeled has 64B lines, is 4-way set associative, and its capacity is varied.

a) For each of the plots above, determine the application’s working set size.

b) Why might bodytrack’s miss rate have gone up with a bigger cache?

TA: Christopher Celio (most problems from TA Scott Beamer’s handout).
**Problem 2**

**Caches**

a) Cache A has a 90% hit rate and cache B has a 95% hit rate. Numerically, how much better is cache B?

b) Can you have a no-write allocate, write-back cache?

c) If you are concerned about memory bandwidth (to DRAM), should you have a write allocate cache?

**Problem 3**

**Write Buffers**

For this problem we are considering adding a write buffer to a single issue in-order CPU with one level of cache. The cache is write back, no write allocate, and for this problem you don’t have to worry about lines ever getting evicted. By adding a write buffer, if a write misses in the cache, it can go directly into the write buffer rather than stalling the system for the miss.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Time (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>Hit Time</td>
<td>2</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>200</td>
</tr>
<tr>
<td>Hit (after adding WB)</td>
<td>$3 + 2 \log_2(WB \text{ depth})$</td>
</tr>
<tr>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>Hit Time</td>
<td>2</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>240</td>
</tr>
<tr>
<td>Hit/miss time (after adding WB)</td>
<td>6</td>
</tr>
<tr>
<td>Cache read hit rate</td>
<td>98%</td>
</tr>
<tr>
<td>Cache write hit rate</td>
<td>70%</td>
</tr>
</tbody>
</table>

a) What is the average memory access time (AMAT) for reads before and after a write buffer of depth 4 is added?

b) What is the AMAT for writes before adding the write buffer?

c) Assume a program averages a write every 24 cycles and that they are decently uniformly distributed. What is the worst average write miss rate the program can have and expect a finite sized write buffer to not overflow?
typedef double real; /* 8 bytes! */

/* Column-major matrices */
#define A(row,column) _A[(row) + ldA*(column)]
#define B(row,column) _B[(row) + ldB*(column)]
#define C(row,column) _C[(row) + ldC*(column)]

void gemm(int M, int N, int K, real alpha, real* _A, int ldA,
          real* _B, int ldB, real beta, real* _C, int ldC) {
    int m, n, k;
    for(n = 0; n < N; n++) {
        for(m = 0; m < M; m++) {
            real C_mn = beta * C(m,n);
            for(k = 0; k < K; k++)
                C_mn += alpha*A(m,k)*B(k,n);
            C(m,n) = C_mn;
        }
    }
}

Above is code from Lab 2. The function gemm performs a matrix multiply such that
C=alpha*A*B+beta*C (where A, B, and C are matrices and alpha and beta are constants).

However, if you run this code you will notice it exhibits poor performance (say, the matrix
sizes denoted by M,N, and K are all 1024).

a) Why is the performance so bad?

b) As the programmer in charge of this piece of code, can you come up with two
   modifications to the software that will make this code run much, much faster? (hint: you
   may want to change or modify one or more of the matrices).
a) What is power? What is energy? How are they different?

b) Does an iPhone (or other mobile device) care more about power-efficiency, or energy-efficiency? What your desktop machine at home? What about a server chip at Google?