CS484 Practice Midterm

Name: _____ NetID: _____

Please read the instructions carefully.

- Please check that this booklet has **16** different pages, including this cover sheet.
- This exam is 75 minutes long.
- Please write your name and NetID in the spaces above.
- This is a **closed-book** exam. No calculators, notes, books, slides, electronic devices, cheat sheets, or other aids allowed.
- **No cheating**. Do not share anything with other students. Do not talk to other students. Do not look at another student's exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will be counted as cheating.
- Read every problem carefully.
- There are **7** problems on this exam. The exam will be scored out of **100** points.
- All the sheets in this exam including all scratch paper will be taken back at the end of the exam.
- *Good luck!*

Problem	Maximum Points	Points obtained
1	10	
2	18	
3	6	
4	14	
5	12	
6	12	
7	28	
Total	**100**	

Problem 1 (10 Points)

Circle whether the following statements are true or false. Each question is worth 2 points. No explanations are necessary.

1. Performing floating point operations is more computationally intensive than integer operations and is of particular interest when measuring performance.

   a. True b. False

2. Bandwidth of a system is the time taken to transfer one cache line of data from memory to the processor.

   a. True b. False

3. Is Moore's Law a Law of Nature?

   a. Yes b. No

4. When the amount of work per thread is too small, the overhead of generating multiple threads can make the performance be worse than running the program sequentially.

   a. True b. False

5. The implicit barrier of a **`#pragma omp parallel for`** loop makes all threads in the team wait for each other when they finish a single iteration before they move on to the next iteration assigned.

   a. True b. False

Problem 2 (18 Points)

For the following questions, circle the ONE answer that applies. Each question is worth 3 points. No explanations are necessary.

1. A program consisting of 200 instructions is run on a 10-stage pipelined machine, where each stage takes 1 clock cycle. The total cycles to execute the program is:
    a. 20 cycles
    b. 21 cycles
    c. 209 cycles
    d. 210 cycles

2. Which of the following statements is false for a 4-way set associative cache of size 64 words and cache line size of 2 words.
    a. The cache has 32 cache frames.
    b. Each cache line has exactly 4 cache frames it can be mapped to.
    c. There are 4 different sets of cache frames.
    d. To access 6 continuous words of data, at least 3 cache lines need to be brought to cache.

3. Which of the following statements are true about reuse ratio.
    a. Reuse ratio is always greater than or equal to the hit ratio.
    b. Reuse ratio is independent of the system the code is run on.
    c. Reuse ratio depends on the cache line size.
    d. All the above statements are false.

4. OpenMP **nowait** clause can be used with **#pragma omp parallel for**. Which of the following statements best describes the effect of OpenMP **nowait** clause?
    a. It indicates that a thread does not have to wait for other threads in the same team to reach the entry of the loop before starting execution.
    b. It indicates that a thread does not have to wait for other threads in the same team when it leaves the loop that has an implicit barrier.

c. It indicates that a thread does not have to wait for any synchronization construct within the loop.
d. It indicates that all threads ignore **#pragma omp wait** within the loop.

5. OpenMP clause **lastprivate** is used when:
   a. The value assigned by the last thread to finish is copied out to the master thread.
   b. Only the thread with the largest index has a private copy, and all other threads share the same copy of the variable.
   c. The value assigned by the iteration which is the last if the loop is executed sequentially is copied out to the master thread.
   d. None of the above.

6. A miss caused by the first access to a piece of memory is:
   A. Compulsory Miss*
   B. Capacity Miss
   C. Conflict Miss
   D. None of the above

.

Problem 3 (6 Points)

Answer the following questions concerning the loop in the following code segment. Assume the size of each array element is 1 word and N is an integer variable passed in as a function argument. For array writes, assume each write incurs a single memory store (there is no load of A).

```
int i;
float x = 3.2;
float y = -1.6;

for(i = 0; i < N; i++) {
    A[i] = x * B[i] + y;
}
```

a. (2 points) What is the data traffic per iteration in words?

b. (2 points) What is the number of FLOPs per iteration?

c. (2 points) Define code balance in terms of data traffic per iteration and FLOPs. What is the code balance B_c here?

Problem 4 (14 Points)

Consider the following code:

```
for (int i = 0; i < N-1; i++)
  for (int j = 0; j < N; j++)
    A[i][j] += A[i+1][j];
```

1. Compute the number of misses in terms of N and w (words in a cache line, where a word is 8 bytes) if the cache can only store 1 entire row of matrix A (cache size = N*sizeof(double)). You can assume that A starts at the beginning of a cache line, that A is an array of doubles, and that the cache is fully associative and is an LRU cache. How many compulsory, capacity, and conflict misses are there?
2. Compute the number of misses in terms of N and w if the cache can store 2 entire rows of matrix A (cache size >= 2*N*sizeof(double)). You can make the same assumptions as part (1). How many compulsory, capacity, and conflict misses are there?

Problem 5 (12 Points)

A primitive machine has a small cache of 64 bytes, with a cache line size of 8 bytes. An array A of size 128 bytes starts from memory address 0. Each element of array A is 1 byte. The cache is 2-way set associative where memory address M is mapped to frame 2*(floor(M/8)%4) or 2*(floor(M/8)%4) + 1. For a sequence of data accesses given in the table below, fill in the following columns.

1. Write the indices of array A which share the cache line with the data accessed.
2. Write Hit, Compulsory Miss, Capacity Miss or Conflict Miss in the column Hit/Miss.
3. Write the cache frame the data is mapped to. Not the possible frames it can be mapped to, but the actual frame in which the cache line is present when this access is done.
4. Mention Yes/No if an eviction has occurred. If yes, give data indices of the array that has been evicted.

Access to a particular element brings the entire cache line. If more than one possible empty cache frame is available, the frame with the lowest index is selected. Assume an LRU eviction strategy if all possible frames are full.

Initially, the **cache line with elements A[0] to A[7] is in frame 0** and its state is valid. The rest of the cache is empty. The first row has been done as an example for you.

S.No.	Data accessed	Cache line	Hit/Miss	Cache frame	Eviction?
1	A[34]	32 - 39	Compulsory miss	1	No.
2	A[6]				
3	A[67]				
4	A[38]				
5	A[4]				

Problem 6 (12 Points)

Suppose you write an OpenMP program which uses 3 threads to parallelize a *for* loop containing 12 iterations, each of which has a different execution time specified below. For each of the scheduling clauses before the *for* loop, fill in the following table with the indices of loop iterations that a thread will be working on at the specific time period.

Assumptions:

(1) When a thread is assigned multiple iterations, it works on the one with lowest iteration index first.

(2) In dynamic scheduling when there are more than one thread is available, the thread with the smallest thread ID gets assigned first.

(3) Ignore the overhead of dynamic scheduling.

The following table specifies the units of time each iteration takes to execute.

Loop iteration	0	1	2	3	4	5	6	7	8	9	10	11
Time units spent	3	1	4	3	2	1	1	5	2	3	1	2

a. (6 points) **`#pragma omp parallel for schedule(dynamic, 1)`**

Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Thread 0												
Thread 1												
Thread 2												

b. (6 points) **`#pragma omp parallel for schedule(static)`**

Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Thread 0												
Thread 1												
Thread 2												

Problem 7 (28 Points)

The following program is an attempt to parallelize matrix-matrix multiplication with OpenMP. However, the code does not parallelize matrix-matrix multiplication correctly and optimally.

```
int i, j, k;

for(i=0; i<N; i++)
    for(j=0; j<N; j++)
#pragma omp parallel for
        for(k=0; k<N; k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

Answer the following questions.

a. (5 points) The parallel code does not produce the same result as the sequential code. Why?

b. (5 points) Why is it not a good parallelization strategy in terms of performance?

c. (8 points) Now try to fix the correctness issue AND optimize the parallel execution performance by **changing the placement of the OpenMP pragma** (you are allowed to add clause to it). Your program must produce the correct results and optimize the performance at the same time. Write down the resulting program and explain why your solution is correct and the performance is better.

d. (10 points) If changing the placement of the OpenMP pragma is **not allowed**, what can you do to make the program produce correct result? Make the necessary changes to parallelize the innermost loop to produce correct result without granting horrible performance.

Appendix A: OpenMP Syntax Reference

Constructs	Clauses	API
#pragma omp parallel #pragma omp parallel for #pragma omp for #pragma omp single #pragma omp master #pragma omp critical #pragma omp atomic #pragma omp ordered #pragma omp flush #pragma omp task	default shared private firstprivate lastprivate reduction nowait schedule if num_threads	omp_set_num_threads omp_get_num_threads omp_get_thread_num omp_get_ancestor_thread_num omp_get_num_procs omp_set_dynamic omp_set_nested omp_get_nested omp_set_lock omp_unset_lock omp_test_lock omp_init_lock omp_destroy_lock

Appendix B: Power of 2 Values

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096

Scratch Space