CS 484 Midterm 1
March 2nd 2016

Total time: 1 hour 15 mins

Name: ________________________________

NetID: _________________________________

<table>
<thead>
<tr>
<th>Question Number</th>
<th>Total Points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>
1. Answer the following questions: [5 points]
   a. GProf is a tool for parallelizing sequential code. True / False
   b. In order to use omp parallel for, the number of times the loop body is to be executed must be available at runtime before the loop is entered. True / False
   c. Arrange the following components from the slowest to the fastest (in terms of access time): HDD, register, main memory.
   d. Fully associative caches cannot have conflict misses. True / False
   e. If a fully parallelizable sequential program executes in T seconds, the fastest parallel program with N threads takes __________ seconds.
2. For this question, assume that \( N \) is a power of 2.

\[
\text{for}(\text{int } d = 1; \ d <= N/2; \ d *= 2) \\
\quad \text{for}(\text{int } i = 0; \ i < N; \ i += 2*d) \\
\qquad \text{a}[i] = \text{a}[i] + \text{a}[i + d];
\]

a. If the array is initialized as \( \text{int } \text{a[]} = \{3, 1, 4, 7\} \), what will it contain after running the above algorithm?  

\textbf{Hint:} Try to understand what the algorithm computes in \( \text{a}[0] \). [6 points]

b. If the above can be parallelized, show how, by inserting the appropriate \texttt{openmp} pragma(s). If not, then explain why. [4 points]

\[
\text{for}(\text{int } d = 1; \ d <= N/2; \ d *= 2) \\
\quad \text{for}(\text{int } i = 0; \ i < N; \ i += 2*d) \\
\qquad \text{a}[i] = \text{a}[i] + \text{a}[i + d];
\]
3. Below are two versions of the same program:

```c
struct node
{
    char *name; // Assume no more than 120 bytes.
    double *age;
    node *next;
};

for (node* iter = head; iter != NULL; iter = iter->next)
{
    // access iter->name;
    // access iter->age;
}
```

```c
struct node
{
    char name[120];
    double age;
};

node array[N];

for (int i = 0; i < N; i++)
{
    // access array[i].name;
    // access array[i].age;
}
```

Assuming
- A fully associative cache.
- Cache line size = 256 bytes.
- sizeof(double) = 8 bytes
- sizeof(array) >> cache size

Write down an expression for the number of cache misses for each version in terms of N. [5 + 5 points]

Part A
--------
Assuming the nodes are arranged in a linked list, each node = 3 pointers
The entries within a node are in contiguous memory locations, i.e. name, age and next pointers fit in a cache line
For each iteration of the inner loop will result in one cache miss – to access the node.
There will be two additional misses to dereference the pointers name and age.
For the second code, each structure is 128 bytes which are contiguous. The data structure is an array of structures, which is again contiguous.

128 + 128 = 256 bytes fit in a cache line
For N iterations of the loop, there will be one miss for every even value of N, the odd iterations will be hits since they belong to the same cache line.
4. Consider the program given below:

```c
#define N ...
#define NUM_THREADS ...

double A[N], local_sum[NUM_THREADS];
double sum = 0, result=0;

#pragma omp parallel num_threads(NUM_THREADS)
{
    int me = omp_get_thread_num();
    int load_per_th = ceil((double) N / num_threads);
    int start_index = me * load_per_th;
    int end_index = (me + 1) * load_per_th;

    int local_val = 0;  // fix false sharing

    for(int i=start_index;i<end_index;i++)
        local_sum[me] += A[i];

    for(int i = 0; i < NUM_THREADS; i++)
        sum += local_sum[i];

    result = sum/N;
}
```

a) Is there a problem with the correctness of this code? It computes the average of N numbers and stores the result in N. [4 points]

b) Identify the most serious performance bottleneck in the above code and fix it. [6 points]
5. Consider the following program and answer related questions:

Assume that there are a small number of zeros and ones in A. Also, don’t worry about efficiency, only worry about correctness.

```c
int ones = 0, zeros = 0;
int A[N];

#pragma omp parallel num_threads(NUM_THREADS)
{
    #pragma omp for
    for(int i=0; i < N; i++)
        if(A[i] == 0)
            Zeros++;
        else if(A[i]==1)
            Ones++;
        else
            continue;
}
```

a) What does this program compute sequentially, i.e. if you ignore the pragmas? [2 points]

b) Will the code provided execute correctly? If not, fix it in the simplest way possible, without removing the pragma. [5 points]
6. Look at the table below, and circle the appropriate answer:

Pair 1: codes A and B are equivalent (true / false) [3 points]
Pair 2: codes A and B are equivalent (true / false) [3 points]
Pair 3: codes A and B are equivalent (true / false) [3 points]

// Pair 1 -- Code A
#pragma omp parallel for
for (int i = 0; i < n; i++)
    sum += array[i];

// Pair 1 -- Code B
for (int i = 0; i < n; i++)
    sum += array[i];

// Pair 2 -- Code A
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++)
    sum += array[i];

// Pair 2 -- Code B
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++)
    sum += array[i];

// Pair 3 -- Code A
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++)
    sum += array[i];

// Pair 3 -- Code B
pthread_t threads[num_threads];
int local_sums[num_threads];
int sum = 0;
for (int i = 0; i < num_threads; i++)
    pthread_create(&threads[i], NULL, add, &i);
for (int i = 0; i < num_threads; i++)
    pthread_join(&threads[i], NULL);
for (int i=0; i < 10; i++)
    sum += local_sums[i];

void* add(void *args) {
    int id = (int) *args;
    for (int i = id; i < n; i += num_threads)
        local_sums[id] += array[i];
}
7. Take a look at the following code ($f$ and $g$ are simple functions that depend only on their arguments):

```c
#pragma omp parallel
{
1. for(i = 0; i < N; i++)
2. {
3. A[i] = f(B[i], i);
5. D[i] = g(C[i]);
6. }
}
```

a. What are the true dependencies in this program? [3 points]

b. Add openmp pragma/s to the code to parallelize it. (only partial credit if you end up modifying the sequential code or the loop structure). You are not allowed to create any additional data structures. [10 points]

c. Assuming
   - N threads on N cores.
   - f() takes 100ns
   - line 5 takes 1 ns
   - g() takes 200 ns

Find the total execution time as a function of N. [6 points]