CS 484 FA17  
Midterm 1, October 5th, 2017  
Total time: 75 minutes No outside materials or devices allowed

Name: ______________________________
NetID: ______________________________

<table>
<thead>
<tr>
<th>Points</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T/F, Multiple Choice</td>
<td>/ 10</td>
</tr>
<tr>
<td>Short Answer</td>
<td>/ 15</td>
</tr>
<tr>
<td>Caches and Loops</td>
<td>/ 20</td>
</tr>
<tr>
<td>OpenMP</td>
<td>/ 50</td>
</tr>
<tr>
<td>Challenge Question</td>
<td>/ 5</td>
</tr>
<tr>
<td>Total</td>
<td>/ 100</td>
</tr>
</tbody>
</table>
1) True or False / Multiple Choice (10 pts)

a) A sequence of reference is said to have temporal locality if accessed objects reside in nearby memory. (True / False)

b) If multiple processors frequently write to distinct variables with adjacent memory addresses, they are most likely to cause:
   Cache line corruption    Cache coherence    False sharing


c) If multiple processors frequently write to a same cache line, they are likely to avoid false sharing. (True / False)

d) Multiple copies of a same data may exist in a multiprocessor system. (True / False)

e) If a machine is using a write-through cache, the main memory always has an up-to-date copy of the data. (True / False)

f) In OpenMP, the number of loop iteration is required to be known before the loop is about to execute. (True / False)

g) In OpenMP, unless specified, the default loop scheduling is schedule(dynamic, 1). (True / False)

h) Given K cores in a system, the max number of pthreads that can be created in this system is K. (True / False)

i) Dynamic scheduling in OpenMP will always result in faster execution of a program than static scheduling. (True / False)

j) Create your own non-trivial true or false question that relates to our course material and provide an answer to it.
2) Short Answers (15 pts)

2.1) What impact does the size of a cache line have on performance of a program in terms of spatial locality? What would happen if a cache line is too big or too small?

2.2) OpenMP follows “Fork-Join Model” in its implementation. When `#pragma omp parallel` is used, when do the forked threads join?

2.3) Given a following for loop with dependencies in its code block, parallelize the for-loop by transforming the expression to a function of induction variable.

```c
int x = 0;
for (i = 0; i < N; i++) {
    x = x + 5;
    A[i] = f(B[i], x);
}
```
3) Cache and loop optimization (20 pts)
3.1) Consider the following nested loop. Assume that an Array A is a 2D array. Assume L1 cache is 32 KB, fully-associative and each cache line is 32 bytes.

double sum = 0;

for(int t = 0; t < 40; t++) {
    for(int i = 1; i <= 80; i++) {
        sum += A[i][t];
        if(i < 40)
            sum += A[i-1][t];
    }
    for(int j = 1; j <= 80; j++) {
        sum += A[j][t];
    }
}

a) Count the cache misses.

b) Propose an optimization.

c) Apply the optimization and recount the cache misses.
4) OpenMP (50 pts)
4.1) Consider the two code snippets below and answer the questions that follow. Provide a very brief reasoning for each of your answer. Assume C is an array initialized to 0’s. (15 pts)

<table>
<thead>
<tr>
<th>Code Snippet A</th>
<th>Code Snippet B</th>
</tr>
</thead>
<tbody>
<tr>
<td>int x = 0;</td>
<td>int x = 0;</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; N; i++) {</td>
<td>for (int i = 0; i &lt; N; i++) {</td>
</tr>
<tr>
<td>x = bar(A[i]);</td>
<td>C[i] = foo(i, x);</td>
</tr>
<tr>
<td>C[i] = foo(i, x);</td>
<td>x = g(A[i], x);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>// foo and bar do not modify i or x</td>
<td>// foo and g do not modify i or x</td>
</tr>
<tr>
<td>// x is not used after this point</td>
<td>// x is not used after this point</td>
</tr>
</tbody>
</table>

a) Can Snippet A be parallelized just by inserting “#pragma omp parallel for” before the loop and inserting declaration of x (int x; ) as the first line inside the loop?

b) Can Snippet A be parallelized just by inserting “#pragma omp parallel for private (x)” before the loop?

c) Can Snippet B be parallelized just by inserting “#pragma omp parallel for” before the loop and inserting declaration of x (int x; ) as the first line inside the loop?

d) Can Snippet B be parallelized just by inserting “#pragma omp parallel for private (x)” before the loop?
4.2) Parallelize the following loop efficiently. Make changes to the code as needed, but the outcome must be the same as that of sequential code. **(15 pts)**

The smallComputation function always takes less than a microsecond, whereas bigComputation may take anywhere between 30 microseconds to a millisecond, each time it is called.

```c
x = 0.0; y=0.0;
for (i=1; i<N; i++) {
    if ( A[i]*A[i] > x)
        B[i] = smallComputation(x,A[i]);
    else
        B[i] = bigComputation(x,A[i]);

    y += B[i];
x = 3.0 + x;
}
printf("x=%f, y=%f\n", x, y);
```
4.3) You are required to parallelize the following loop efficiently. Make changes to the code as needed, but the outcome must be the same as that of sequential code. (20 pts)

For this problem, array A is known to be a sorted array. N is very large (100s of millions). Dynamic scheduling has high overhead and should be avoided when possible. About 10% to 20% of elements of A are larger than c.

The smallComputation function always takes less than a microsecond, whereas bigComputation may take anywhere between 30 microseconds to a millisecond, each time it is called.

c = computeC();

for (i=1; i<N; i++) {
    if ( A[i] < c)
        B[i] = smallComputation(A[i]);
    else
        B[i] = bigComputation(A[i]);
}

a) Briefly explain how you would parallelize the code.
b) Write the parallel version of the code.
5) Challenge Question (5 pts)

Note: this question may take time disproportionate to the number of points assigned to it, intentionally. We suggest you attempt it only if you are done with other questions.

The following sequential code counts in the array inc_arr the number of elements that are less than the value of its preceding element in an array arr. Using pthreads, parallelize the seq_count_inc function. Assume array inc_arr is initialized to 0’s.

// Sequential Version
void seqCountInc(int *A, int N, int *B) {
    for (i =1; i < N; i++) {
        if (arr[i] < arr[i-1])
            B[i] = 1 + B[i-1]
        else
            B[i] = B[i-1]
    }
}

int main() {
    int *A, *B;
    int N = someNumber();     /* some number to N */
    A = (int *) malloc (sizeof (int) * N);
    B = (int *) malloc (sizeof (int) * N);
    init_array(A, N);          /* Initializes array to some numbers */
    parallel_count_inc(A, N, B);

    return 0;
}

Write your parallelized code on the next page
/ Parallel Version
void parallelCountInc(int *A, int N, int *B) {

}
OpenMP reference

#pragma omp

- **parallel**: “Forms a team of threads and starts parallel execution”
- **for**: “Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.”
- **single**: “Specifies that the associated structured block is executed by only one of the threads in the team.”
- **task**: “Defines an explicit task. The data environment of the task is created according to data-sharing attribute clauses on task construct and any defaults that apply.”
- **master**: “Specifies a structured block that is executed by the master thread of the team.”
- **critical**: “Restricts execution of the associated structured block to a single thread at a time.”
- **barrier**: “Specifies an explicit barrier at the point at which the construct appears.”
- **atomic**: “Ensures that a specific storage location is accessed atomically”
- **flush**: “Executes the OpenMP flush operation, which makes a thread’s temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.”
- **ordered**: “Specifies a structured block in a loop, simd, or loop SIMD region that will be executed in the order of the loop iterations.”
- **private**: “Declares one or more list items to be private to a task or a SIMD lane. Each task that references a list item that appears in a private clause in any statement in the construct receives a new list item.”
- **firstprivate**: “Declares list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.”
- **lastprivate**: “Declares one or more list items to be private to an implicit task or to a SIMD lane, and causes the corresponding original list item to be updated after the end of the region.”
- **reduction**: “Specifies a reduction-identifier and one or more list items. The reduction-identifier must match a previously declared reduction-identifier of the same name and type for each of the list items.”
- **section**: “A noniterative worksharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team.”
- **depend**: “Enforces additional constraints on the scheduling of tasks or loop iterations. These constraints establish dependences only between sibling tasks or between loop iterations.”
- **collapse**: “A constant positive integer expression that specifies how many loops are associated with the loop construct.
- **linear**: “Declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop”

Other:
- **omp_get_num_threads()**: “Returns the number of threads in the current team. The binding region for an omp_get_num_threads region is the innermost enclosing parallel region.”
- **omp_get_max_threads()**: “Returns an upper bound on the number of threads that could be used to form a new team if a parallel construct without a num_threads clause were encountered after execution returns from this routine.”
- **omp_get_thread_num()**: “Returns the thread number of the calling thread within the current team.”
Scratch Paper