CS 654 Homework 4

Due date: Monday, Nov 14

In the implementation problems below (3–5), run the program many times to make sure it always produces the same (correct) answer. Parallelism bugs and race conditions may appear infrequently.

1. Solve problem 5.29 from your book.

2. Solve problem 5.32 from your book, but with following while loop instead:
   `P2: while (flag == 0) {;}`

3. We want to compute $\pi = 3.14159265358979$ through the following series:

$$\pi = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1}.$$

   A straightforward code for this is:

```
double sign = 1.0;
double piold = 1.0;
double pi = 0.0;
int i = 0;
while ( abs(pi-piold)>1D-10 ) {
  piold = pi;
  pi += sign/(2*i+1);
  sign=-sign;
  i++;
}
pi = 4*pi;
```

   (a) Can you parallelize the above code with appropriate OpenMP directives? What are the difficulties? What solutions can you provide to these difficulties so that this or a modified code is parallelizable? Make sure that the summation is still correct, regardless of the number of threads running.

   (b) Run experiments on a machine with 4 cores (bg4 or bg5), and vary the number of threads from 1 to 16. Report times and speedup. Discuss your results.

4. This is the basic algorithm for matrix-vector multiplication. If A is an $n \times m$ matrix, and x is an $m \times 1$ vector, $y = Ax$ is a vector of size n such that: $y_i = \sum_{j=1}^{m} A_{i,j}x_j, \;\; i = 1, 2, \ldots, n$. Write a short program to implement this (two nested loops), where A is a 2 dimensional C double array, and x, y are one dimensional double arrays. Consider two cases, First, A is of size (10000×1000) and second A is of size (1000×10000). Both can be populated with arbitrary numbers.

   (a) Parallelize with OpenMP the inner loop. Make sure the resulting parallel code is correct. Run experiments for both matrix sizes on a machine with 4 cores, and vary the number of threads from 1 to 16. Report times and speedup.

(b) Parallelize with OpenMP the outer loop. Make sure the resulting parallel code is correct. Run experiments for both matrix sizes on a machine with 4 cores, and vary the number of threads from 1 to 16. Report times and speedup.

(c) Comment on the performance observed above.

5. The cumulative sum is the simple loop:

```
for i=2,N
  A(i)=A(i-1)+A(i),
```

which produces $A'_i = \sum_{k=1}^{i} A_k$. Obviously data dependencies across loops does not allow a straightforward parallelization. Assume that we have p threads, and that $N \gg p$.

(a) Modify the above loop (breaking it in more loops and possibly using temporary arrays of dimension p) to parallelize it.

(b) Implement this algorithm in C, and run it on bg4 or bg5 with 8 threads. Beyond what N does the parallel algorithm become faster? What is the maximum speedup achieved with sufficiently large N?