COL 380	Feb 20, 2019
Homework 1	
Instructor: Subodh Sharma	**Due: Feb 27, 23:55 hrs**

NOTE: Answer submissions must be made either in word document or pdfs. No hand-written assignments would be accepted. All assignment submissions will be checked for plagiarism.

Problem 1:

- Is the parallelization safe in the code above? Explain your answer. Replace, in the second loop, the size of the iteration space from N to M ($N \neq M$) and the schedule from `static` to `dynamic`. Is the parallelization safe? Explain your answer. (2 marks)

```
#pragma omp parallel
{
#pragma omp for schedule(static) nowait
  for(i=0;i<N;i++){
  a[i] = ....
  }
#pragma omp for schedule(static)
  for(i=0;i<N;i++){
  ... = a[i]
  }
}
```

- Efficiently parallelize the loop shown below. Explain why your solution is efficient. (4 marks)

```
for(int j=0; j<n; j++) {
    for(int i=1; i<n; i++) {
        a[i*n+j] += a[(i-1)*n+j];
    }
}
```

- Debug and optimize the code below. Explain your answer. (4 marks)

```
// located inside a parallel region in function foo
#pragma omp critical
{
  a++; // Modify a exclusively
}
// located inside a parallel region in function bar
#pragma omp critical
{
   b++; // Modify  b exclusively
}
#pragma omp atomic
{
  a = a+b; // Modify a exclusively
}
```

Problem 2:

Consider the code shown below. The sorting algorithm exhibits quadratic dependency in terms of the length of X and can be parallelized easily. (10 marks)

```
1  void seq_sort(std::vector<unsigned int> &X){
2    unsigned int i, j, count, N=X.size();
3    std::vector<unsigned int> tmp(N);
4    for(i=0; i<N; i++){
5      count=0;
6      for(j=0; j<N; j++)
7        if(X[j]<X[i] || X[j]==X[i] && j<i)
8          count++;
9      tmp[count]=X[i];
10   }
11   std::copy(tmp.begin(), tmp.end(), X.begin());
12 }
```

- Analyze the data dependencies of each loop. Which loop is ideally suited for parallelization with OpenMP pragmas?
- Implement a parallel version of seq_sort in OpenMP according to your former considerataions and discuss speedup and efficiency.

Problem 3:

- Consider the example GPU kernel shown in the figure below. Perform optimizations to reduce branch divergence, memory bank conflicts and overheads due to barriers in the code. Show the relevant code changes only for the sought optimizations. (10 marks)

```
1        __global__ void reduce0(int *g_idata, int *g_odata) {
2          extern __shared__ int sdata[];
3          // each thread loads one element from global to shared mem
4          unsigned int tid = threadIdx.x;
5          unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
6          sdata[tid] = g_idata[i];
7          __syncthreads();
8          // do reduction in shared mem
9          for (unsigned int s=1; s < blockDim.x; s *= 2) {
10           if (tid % (2*s) == 0) {
11           sdata[tid] += sdata[tid + s];
12         }
13         __syncthreads();
14       }
15       // write result for this block to global mem
16       if (tid == 0) g_odata[blockIdx.x] = sdata[0];
17       }
```

- What are the two ways to avoid a data race among threads of a block? Also explain the function qualifiers that are used in CUDA C. (4 marks)

Problem 4:

Explain *future, promises* and *asynchronous* function calls in C11 standard in the context of multi-threading. Choose a simple problem of your liking and showcase the use of the above primitives. (10 marks)