CSCI-UA.0480-003

Parallel Computing

Final Exam

Spring 2015 - May 18th (90 minutes)

NAME: **ID:**

NetID:

- The exam is open book/notes.
- If you have to make assumptions to continue solving a problem, state your assumptions clearly.
- You answer on the question sheet. You can use extra white papers if you want.

1. [1 pt]What is the main reason we moved from single core to multicore processors?

Due to power constraint that became very severe because Dennard scaling stopped.

2. Suppose we have two MPI processes.
   a. [1 pt] Suggest one reason why they would execute slower on a system with two processors than on a system with one processor.

   When there is a lot of communication between them and one of them is I/O bound and the other computation bound.

   b. [1 pt] Suggest another reason why the reverse could be true.

   When they are totally independent, or with minimal communication.

3. Suppose we have the following piece of code:

```
for (i = 0; i < 100; i++)
    do_work(i);
```

   a. [2 pts] What are the characteristics of do_work() that makes the above code suitable for MPI?

   - **No dependencies among do_work(x) and do_work(y) where x != y**
   - **Each instance requires a lot of memory and computation resources**

   b. [2 pts] What are the characteristics of do_work() that makes the above code suitable for OpenMP with dynamic scheduling?

   - **Independent loop iterations**

- **The work at each iteration depends on i.**

4. [1 pt] Provide a scenario where you need to split the communicator in MPI (no need to write code)

   For example, when you need to use collective communication among subset of processes, and using one-to-one communication among all of the subgroup will incur a lot of overhead.

5. [2 pts] We have seen loop unrolling in CUDA. But obviously, it can also be used in OpenMP. When do you think it will be beneficial in OpenMP?

   When performance of each iteration is very predictable, unrolling loops can reduce the overhead of loop management code (condition checking and updating loop index if any).

6. [4 pts] For the following code:

```
for (i = 0; i < 100000; i++)
    a[i + 1000] = a[i] + 1;
```

Can we, somehow, parallelize the above code using OpenMP? If so, please re-write the parallel code. If no, explain why.

```
#pragma omp parallel for private(stride)
for (i=1; i<100; i++){
        stride = i*1000;
        for(j=0; j<1000; j++)
                a[stride+j] = a[j] + i;
}
```

7. [2 pts] How could the following code sequence be changed to expose more parallelism but still achieve the same final result (i.e. at the end: x, a, b, and c have the same value as the sequential code)?

```
x++ ;
a = x + 2;
b = a + 3;
c++;
```

Distribute original x to 4 threads, then: T1: x++ ; T2: a = x+3; T3: b: x+6; T4: c++

8. a. [1 pt] What is thread divergence?

If-else in a kernel, and some threads in a warp go with the if-part and the other with the else part. Since threads need to execute on lock-step, time for both the if-part and else-part is consumed.

b. [1 pt] Why is it bad for performance?

Because the time taken by the kernel is the time for both if-part and else-part.

c. [2 pts] Based on your answers in a and b above, does the following kernel suffer from thread divergence? Justify.

```
__global__ void do_work(int i){
        int result = 0;

        if( i < 5)
            for(j = 0; j < blockIdx.x; j++)      } Part 1
                result += j;

        a[threadIdx.x] = result;                 } Part 2
}
```

No branch divergence because all threads have the same i, and all thread are making the same comparison (i < 5). So, either all of them will be true or all of them will be false. Moreover, all of them have the same blockIdx.x