CSCI-UA.0480-003

Parallel Computing

Homework Assignment 1

Total: 20 points

1.a [1 point]

```
quotient = n / p;
my_n_count = quotient;
my_first_i = my_rank * my_n_count;
my_last_i = my_first_i + my_n_count;
```

1.b [2 points]

One way of doing it is as follows:

```
quotient = n / p;
remainder = n % p;
if (my_rank < remainder) {
        my_n_count = quotient + 1;
        my_first_i = my_rank * my_n_count;
        }
else {
        my_n_count = quotient;
        my_first_i = my_rank * my_n_count + remainder;
        }
my_last_i = my_first_i + my_n_count;
```

2. (a: 1 point, b: 1 point, c: 4 points (-0.5 for each mistake), d:1 point)

(a) The number of receives is $p - 1$, and the number of additions is $p - 1$.

(b) The number of receives is $\log_2(p)$, and the number of additions is $\log_2(p)$.

(c)

p	Original	Tree-Structured
2	1	1
4	3	2
8	7	3
16	15	4
32	31	5
64	63	6
128	127	7
256	255	8
512	511	9
1024	1023	10

(d) The receive operation is more expensive because it requires communication through wires. So if we have 32 cores for example, core 0 may be very far from cores 31! To make things even worse, the cost of receive is variable depending on the position of the core that sends. If we do not take this into account, our parallel version of a program may perform worse that the serial version because of the high cost of communication!

3. [1 point]
This happens when those parts are seldom executed. If a part (a part can be a procedure, loop body, group of procedures, etc) is only 2%, for example, of the total execution time of the sequential version, then it is not worth the effort to parallelize it. This is why it is very important to profile the sequential application to see which parts are worth concentrating on.

4. [4 points: -0.5 for each mistake, no explanation is needed for 3rd column]

Type of Parallelism	**Definition**	**Needs Programmer Help**
Instruction Level parallelism	Executing several instructions in parallel, either through time (pipelining) or space (superscalar and VLIW)	No, hardware (superscalar and pipelining) and compiler (VLIW) exploit it.
Data Level Parallelism	Executing the same instruction (SIMD) or the same task/thread on different data items	Yes, GPUs are a clear example of that; although compiler can make use of vector instructions (vector processors).
Task Level Parallelism	Executing several threads/processes at the same time (MIMD)	Yes!

5. [1 point] When the number of cores/processors becomes very large, it will be very hard for all of them to share memory. The memory in that case will be a very severe bottleneck due to contention. A memory system can respond to only a limited number of requests in parallel.

6. [2 points] a. Yes, we can parallelize this algorithm because operations like:

a[0] = a[0] + a[N/2]
a[1] = a[1] + a[1 + N/2]
….
a[N/2 -1] = a[N/2 -1] + a[N-1]

are totally independent and can be executed in parallel.

[2 points] b. Based on the a. above, we can execute each of the N/2 operations above in parallel. So more than N/2 cores will not make any sense.