CSCI-UA.0480-003

Parallel Computing

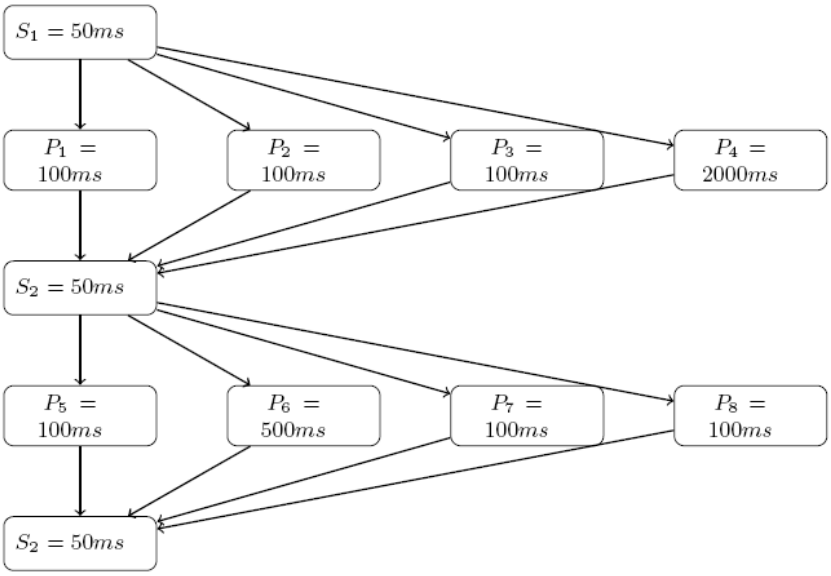
Midterm Exam

Spring 2017 (60 minutes)

NAME: NetID:

- This exam contains **9 questions** with a total of 40 points in **5 pages.**
- If you have to make assumptions to continue solving a problem, state your assumptions clearly.

1. Assume we have the following task graph. A task can be thought of as function/procedure. Every task is labeled with its run time on a core. An arrow from a task to another means that the first task generates data needed by the second one. Assume all data are of the same size. For all the questions of this problem, assume we have enough cores to run each process on a separate core.



a. [3 points] What is the best number of processes to parallelize that program? Justify.

Although you may think we need 4 processes, but on a deeper thinking, you will find that 2 processes are enough to get as much speedup as 4 processes but better efficiency.
Process 1 will work on S1, P1, P2, and P3; while process 2 will work on P4. This will take 2050ms.
Then process 1 will work on S2, P5, P7, and P8; while process 2 works on P6. The total time for that part is 550ms.

Finally, process 1 will do S2. This makes the total execution time = 2050 + 550 + 50 = 2650ms

b. [3 points] What will be the speedup if we have 2 processes? 4 processes? 8 processes?

Speedup(p) = T_1/T_p where P is the number of processes.
T_1 = 3250 (the sum of all tasks)
$T_2 = T_4 = T_8 = 2650$ (as indicated in the previous question)
So
S(2) = S(4) = S(8) = 3250/2650 = 1.23

c. [2 points] What is the span for the above graph? what is the work?

Span is the longest path of dependence in the graph
Span = T_∞ = 2650

Work is the total amount of execution spent on all instructions
Work = T_1 = 3250

d. [1 point] What is the parallelism?

Parallelism $T_1/ T_\infty = 3250/2650 = 1.23$

e. [1 point] What does the number you calculated in d mean?

It tells us that increasing the number of processes beyond ceil(1.23) = 2 does not yield any advantages.

2. [6 points] State 3 shortcomings of Amdahl's law.

- **Assumes we can have infinite number of cores/processors**
- **Does not take overheads into account (communications, synchronization …).**
- **It is not easy to calculate the sequential part.**

3. [6 points] State three factors that can negatively affect the scalability (as we keep adding processes) of a parallel program.

- **problem size not big enough**
- **high communication overhead**
- **synchronization overhead**
- **load imbalance**
- **overhead of creating and managing processes by the runtime and OS**

4. [3 points] Can a process in MPI send a pointer to another process? If yes, write the lines for the send and receive between the two processes. It no, justify.

No, it cannot because each process has its own address a space. So address X for process 0 is different than address X for process 1.

5. [3 points] Does coherence protocols cause extra communication overhead in MPI? Justify.

No, because there is no coherence needed with MPI as the memory is not shared among processes.

6. [3 points] Suppose that you want to synchronize (i.e. put a barrier) among several processes but *without using MPI_Barrier()*. What can you do?

Simply use a collective call for a small piece of data.

7. [2 points] How does MPI programs deal with race conditions?

Race conditions are bad when there are critical sections because the correctness of the program is affected. But in MPI, it does not deal with critical sections at all because there is no shared memory with MPI and hence no *harmful* race conditions.

8. [4 points] State 2 reasons why two processes assigned to two different cores (and no other processes share the cores with them) and executing the same piece of code may not finish at the same time.

- **One can get cache misses while the other does not.**
- **Non-uniform memory access**
- **Non-uniform cache access**
- **One process can suffer page faults while the other does not.**

9. [3 points] Suppose a process wants to send data to a subset of processes in MPI. There are two options: (a) several send-receive pairs between this process and each one of the destinations. (b) Split the communicator then use a collective call to send the data to the destinations. Which option do you think is faster? and why?

Splitting the communicator is a better strategy because we will have two overhead: one for splitting for the communicator (runtime overhead) and one for sending the data. On the other hand the first strategy will suffer overheads for each send-receive pair.