**CSCI-UA.0480-003**

# Parallel Computing

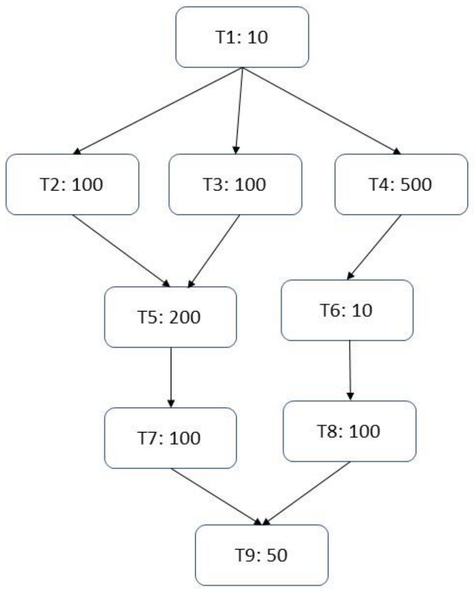
**Midterm Exam**

**Spring 2018 (60 minutes)**

**NAME:** NetID:

- This exam contains **5 questions** with a total of 20 points in **4 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. Do not make any assumptions regarding cache misses, coherence, etc. Just assume that the given runtimes took all these factors into account.



a. [2 points] What is the smallest number of cores needed to get the best performance? Explain.

**We just need two cores. Note that T2+T3+T5+T7 = T4. So a core is needed to execute T2,T3, T5, and T7. Another core will execute, in parallel, T4. After that, one of these two cores will execute T6, T8, and T9 sequentially due to the dependence.**

b. [1 point] Based on the number of cores you picked in part (a) above, what is the total run time?

**The total runtime will be T1(10) + T4 (500) + T6(10) + T8(100) + T9(50) = 670**

c. [2 points] If we use 9 cores, what will be the efficiency? You don't need to write the final number if you don't have a calculator.

**Efficiency = speedup/cores = speedup/9**
**To get speedup we need to get the sequential time = sum of all tasks = 1170**
**Time with 9 cores = 670**
**Efficiency = (1170/670)/9 = 0.19 which is bad.**

d. [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 = $T_\infty$ = 670**

**Work is the total amount of execution spent on all instructions**
**Work = $T_1$ = 1170**

d. [1 point] What is the parallelism?

**Parallelism $T_1/T_\infty$ = 1170/670 = 1.7**

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

**It tells us that increasing the number of processes beyond ceil(1.7) = 2 does not yield any advantages.**

2. [2 points] We have studied Amdahl's law in class. Suppose that the fraction of the sequential part of your program if 50% and we have two cores. What is the best action to get better performance: trying to reduce the sequential part to 25%? or keeping it at 50% but use 4 cores instead of 2? Show clearly all the steps.

**$S = 1/(F + (1-F)/p)$**
**if $F = 0.25$ and $p = 2 \rightarrow S = 1.6$**
**If $F = 0.5$ and $p = 4 \rightarrow S = 1.6$**
**So, both strategies yielded the same result.**

3. [2 points] Suppose you have an MPI program and a sequential program. State two reasons that can make the MPI running with one process slower than the sequential code.

- **The overhead of MPI related function calls (e.g. MPI_Init, …).**
- **An algorithm with more computations, but with better parallelization opportunities, could have been used in the MPI parallel program.**

4. Answer the following questions about coherence.
a. [2 points] State two overheads of coherence?

- **Extra bandwidth due to messages sent to other caches to invalidate their copies of a block when a core wants to write to that block.**
- **Extra cache misses when caches invalidate their block.**
- **Extra delay because a core must wait for the other caches to invalidate their copies before it can write to its own copy.**

b. [1 point] If two parallel threads (i.e sharing memory) try to update two different variables, can this cause coherence overhead? Justify.

**Yes, if these two variables are stored in the same cache block. This is called false sharing**

c. [1 point] As the number of processes in MPI increases, does this make coherence overhead more severe? Explain.

**It has no effect on coherence because processes are not sharing any data in MPI.**

5. [3 points] Suppose we have the following code snippet to be executed by three processes.

```
switch(rank) {
  case 0:
          MPI_Bcast(x, count, MPI_INT, 0, MPI_COMM_WORLD);
          MPI_Send(y, count, MPI_INT, 1, 0, MPI_COMM_WORLD);
          break;
  case 1:
          MPI_Recv(x, count, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                      status);
          MPI_Bcast(y, count, MPI_INT, 0, MPI_COMM_WORLD);
          MPI_Recv(x, count, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                      status);
          break;
  case 2:
          MPI_Send(x, count, MPI_INT, 1, 0, MPI_COMM_WORLD);
          MPI_Bcast(y, count, MPI_INT, 0, MPI_COMM_WORLD);
          break;
}
```

Fill in the blanks in the following table with the values of the variables x and y for each process.

| | Process 0 | Process 1 | Process 2 |
|---|---|---|---|
| x | 7 | **8** | 9 |
| y | 8 | **7** | **7** |