Parallel & Distributed Computing (Spring 2016): Midterm Exam

Show all work on this exam. Time: **1 hr 50 mins**

Make sure to write your full name and student ID on the exam! Only class notes are allowed and no other book or electronic device is allowed.

---

1. (42 points, 6 points each) Please give brief (1-2 lines) answers to each of the following questions.

   A. What is Amdahl's Law? List two possible reasons that one may not achieve the speedup predicted by the Amdahl's Law, even when parallelization is possible?

   B. When parallelizing a loop using OpenMP, under what condition shall one consider using dynamic scheduling? How can dynamic scheduling be specified?

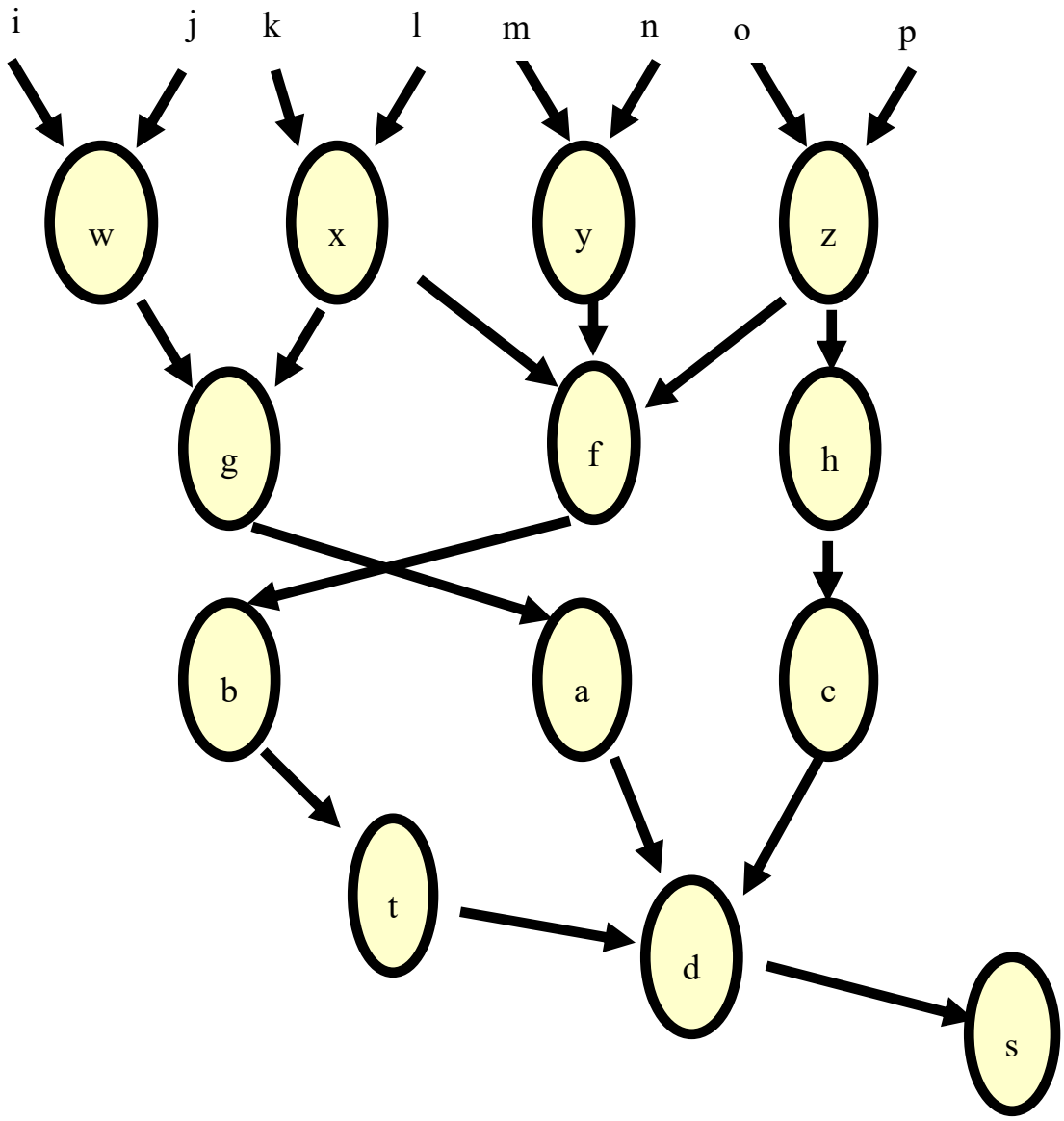
   C. In the 4-step Foster's methodology for developing parallel programs, why do we have both the partitioning and agglomeration steps? They seem to have opposite goals.

   D. In exploiting pipelining for parallelism, what are the important factors that may limit getting good speedup?

   E. Why loop exchange can help to improve performance of parallel program in some cases? When it is safe to exchange the order of two loops?

F. Please list two major differences between OpenMP and MPI?

G. Given the following task graph, what is the maximum number of threads (or cores) that one may use so that beyond that, there is no further speedup potential. Explain why.



2. (18 points) For the following sequential program, someone comes up with a parallel implementation using OpenMP on the right, is it correct? If not, please propose two different ways to fix the problem.

```
double a, b, t, output;
int i, n;
t = 0.0;
for (i = 0; i < n; i++) {
      a = (i + 1) / n;
      b =  i * i;
      t = t + b / (1.0 + a*a);
}
output = t / n;
```

```
double a, b, t, output;
int i, n;
t = 0.0;
#pragma omp parallel for private(a,b)
for (i = 0; i < n; i++) {
      a = (i + 1) / n;
      b =  i * i;
      t = t + b / (1.0 + a*a);
}
output = t / n;
```

3. (20 points) For the following pseudo-code for computing all-pair shortest paths, (a) please specify all data dependencies; (b) please discuss if each of these loop transformations can be applied. If yes, please explain to which loop(s) it can be applied, if it's going to be beneficial, and why. If not, please explain why as well.
    a. Loop permutation
    b. Loop unrolling
    c. Loop tiling
    d. Loop fusion

```
int main() {
    int i, j, k;
    float **a, **b;
    // initialize a[][], b[][] as the 1-hop distance matrix
    for (k = 0; k < N; k++) {
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++)
                a[i][j] = min(a[i][j], b[i][k] + b[k][j]);
        }
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++)
                b[i][j] = a[i][j];
        }
    }
}
```

4. (20 points) Parallelize N-body simulation.

N-body simulation finds the positions/velocities of a collection of interacting particles over a period of time. The input is

- The particle struct array *Q* of *N* particles, each particle contains
  - The masses *m*,
  - The positions *p (3-dimension)* at time 0,
  - The velocities *v (3-dimension)* at time 0.

The output is

- The same particle struct array *Q* of *N* particles with updated positions/velocities.

The sequential algorithm is shown below (pseudo code):

```
void nbody(Particle *Q) {
   struct Particle new_Q[N]; // N is an external defined constant.
   for timestamp t from 1 to T { // T is an external defined constant.
       for each particle x in Q {
           double force[3] = {0.0}; // Total force of particle x
           for each particle y in Q {
               calcForce(force, Q[x], Q[y]);
               // Compute and accumulate the force between particle x and y
               // in a constant time.
           }
           new_Q[x] = update(Q[x], force);
           // Update the position/velocity for particle x in a constant time.
       }
       swap(Q, new_Q); // Update all positions in a constant time.
   }
}
```

1) Assuming that the array *Q* is stored at processor 0 initially (and the results will be collected at processor 0 as well at the end), please provide MPI-like pseudo code to implement the algorithm using *p ($p = k^3$ for some integer k)* processors with distributed memory. Please specify

   a. What is your parallelization strategy?
   b. How the data will be distributed?
   c. What are the MPI communication functions that you are going to use at each step?
   d. Please analyze the communication complexity (assuming the network topology is a hypercube), and provide the scalability function of your implementation.

2) Suppose the N articles are all in an L*L*L box and their distribution is roughly uniform. Moreover, forces from particles of distance larger than R can be ignored (assuming that L is a multiple of R). Can you come up with a more efficient solution than the algorithm list above? Please also provide MPI-like pseudo-code.