The questions on page 1 are for the in-class exam and must be solved within 50 minutes without any computing device. Text books, copies of slides, and notes are allowed. The decision to do the take-home version of this exam can be postponed till 10:49AM, if you do not hand in at 10:50AM the answers to the questions below. In grading, all questions will receive the same amount of points.

1. Consider the application of the composite Trapezoidal rule to compute the integral of f over $[a, b]$ by a distributed parallel program.

   Let m be the number of floating point operations needed to evaluate f at one point. The cost of the composite Trapezoidal rule over n subintervals of $[a, b]$ requires $n + 1$ function evaluations.

   (a) Describe a high level parallel distributed memory with just enough detail to define the communication and computation cost, in terms of m and n as defined above, and p the number of processors.

   (b) Define the efficiency and develop a model to analyze the scalability.

2. Let f be a continuous function over an interval $[a, b]$, with $f(a)f(b) < 0$.

   One step of the bisection method to find one root of f in $[a, b]$ computes the midpoint $m = (a + b)/2$ and continues to bisect $[a, m]$ if $f(a)f(m) < 0$ or to bisect $[m, b]$ if $f(m)f(b) < 0$.

   Consider the application of the bisection method to the problem of approximating **all** roots of f over an interval $[a, b]$. Given a tolerance tol, the output consists in a list of subintervals of $[a, b]$ which contain one root of f, where the width of each subinterval is no larger than the value of tol.

   (a) Describe a shared memory parallel implementation of such a bisection root finder. Name the techniques we covered to write parallel algorithms that apply to this problem. Use a high level task based description of the algorithm.

   (b) How does work stealing apply?

3. The inverse power method is an iterative method to approximate the eigenvectors of a matrix.

   Starting with an approximation `x[0]` for the eigenvector with corresponding eigenvalue `mu` for which `mu[0]` is an approximation for its absolute value `|mu|`, consider

   ```
   for k = 0, 1, ... do
      x[k] = x[k]/norm(x[k])
      (A - mu[k])*x[k+1] = x[k]
      m[k+1] = norm(A*x[k+1])/norm(x[k+1])
   ```

   where `x[k+1]` is computed as the solution of a linear system with right hand side vector `x[k]` and matrix `(A - mu[k])`.

   The method is expected to converge to the the eigenvector `x` with corresponding eigenvalue `mu` of `A`.

   (a) Describe a parallel implementation for this method.

   (b) For distributed memory, describe the synchronization techniques, referring the proper terminology we covered in class.

4. Consider the tiled version of the LU factorization.

   For a matrix divided as a 4-by-4 matrix of tiles, answer the following questions.

   (a) Lists all operations on the tiles to compute the LU factorization.

   (b) Draw the directed acyclic graph which has as its nodes the tasks to compute the tiled LU factorization. The directions in this graph indicate the sequence of tasks. Parallel branches in the graph may run simultaneously.

The questions on page 2 are for the take-home exam and must be solved individually. Answers are due on Monday 24 October, at 10AM.

1. Consider the application of the composite Trapezoidal rule to compute the integral of f over $[a, b]$ by a distributed parallel program.

   Let m be the number of floating point operations needed to evaluate f at one point. The cost of the composite Trapezoidal rule over n subintervals of $[a, b]$ requires $n + 1$ function evaluations.

   (a) Write a basic MPI implementation and test your implementation with polynomials of increasingly higher degrees. Can you extend your basic version with double double and quad double arithmetic?

   (b) Examine speedups and scalability. Take into account that for higher degree polynomials, double double and quad double precision may be needed.

2. Let f be a continuous function over an interval $[a, b]$, with $f(a)f(b) < 0$.

   One step of the bisection method to find one root of f in $[a, b]$ computes the midpoint $m = (a + b)/2$ and continues to bisect $[a, m]$ if $f(a)f(m) < 0$ or to bisect $[m, b]$ if $f(m)f(b) < 0$.

   Consider the application of the bisection method to the problem of approximating **all** roots of f over an interval $[a, b]$. Given a tolerance tol, the output consists in a list of subintervals of $[a, b]$ which contain one root of f, where the width of each subinterval is no larger than the value of tol.

   (a) Use OpenMP, Pthreads, or the Intel TBB to write a correct shared memory implementation. Correctness is the main criterion. Test your code on polynomials with known roots.

   (b) Count the number of function evaluations. Compare the number of function evaluations in the serial implementation and the parallel implementation.

3. The inverse power method is an iterative method to approximate the eigenvectors of a matrix.

   Starting with an approximation `x[0]` for the eigenvector with corresponding eigenvalue `mu` for which `mu[0]` is an approximation for its absolute value `|mu|`, consider

   ```
   for k = 0, 1, ... do
      x[k] = x[k]/norm(x[k])
      (A - mu[k])*x[k+1] = x[k]
      m[k+1] = norm(A*x[k+1])/norm(x[k+1])
   ```

   where `x[k+1]` is computed as the solution of a linear system with right hand side vector `x[k]` and matrix `(A - mu[k])`.

   The method is expected to converge to the the eigenvector `x` with corresponding eigenvalue `mu` of `A`.

   (a) Provide a correct distributed memory implementation. Use an iterative method to solve the linear system to compute the update. You may recycle code of the lectures.

   (b) Measure the computation cost with a flop count, that is: report the number of floating point operations for each processor. To examine the computation cost, compute how many bytes are transferred in each synchronization state. Compare the computation and communication cost.

4. Consider the tiled LU factorization.

   (a) Generate a random dense matrix and apply the proper routine of the PLASMA software package to examine the speedup of the parallel LU factorization. How large should the matrix A be to obtain a good speedup?

   (b) Explore the numerical stability of the tiled LU factorization in the PLASMA library routine, for increasing dimensions. For random matrices, compare the product of the computed L with U to the values in the generated matrix A.