MCS 572 Take Home Midterm on Friday 3 March 2006, at 1PM

0. The due date is Monday 6 March, at 1PM

Your answers to the questions will be collected before the class starts on Monday 6 March, at 1PM. If you cannot make it to class, you may submit earlier, eventually electronically.

All four problems will be graded for the same number of 25 points each, summing up to a total of 100.

The problems may still be handed in later to count as homework on our regular Friday collection. As homework questions, each problem is worth 4 points.

You are allowed and encouraged to consult all literature and computing resources available to you. However, this exam is an individual effort. Collaborations are not permitted.

1. Monte Carlo methods to estimate permanents

The permanent per(A) of an n-by-n matrix $A = (a_{ij})$ is per$(A) = \sum_{\sigma} \prod_{i=1}^{n} a_{i\sigma(i)}$, where σ in the sum runs over all permutations of n elements. While the definition seems easier than a determinant, to compute per(A) exactly requires the evaluation of $n!$ sums. As the exact computation of per(A) is impossible for large n, we will apply Monte Carlo methods to estimate per(A).

To keep our methods clean, we consider only zero-one matrices for A, i.e.: $a_{ij} \in \{0, 1\}$. The problem of computing per(A) is then equivalent to estimating how many of the $n!$ permutations will contribute to per(A), i.e.: for how many σ will $\prod_{i=1}^{n} a_{i\sigma(i)} \neq 0$? The Monte Carlo method generates N random permutations σ, computes for each σ the corresponding product, and updates the sum S. The estimate for per(A) is then $n!S/N$. As N grows closer to $n!$, our estimate will get more accurate, In the extreme case: if $N = n!$ and if all generated σ's are different, then we have computed per(A).

1. Use pseudo code to define the algorithm to generate the permutations σ.
2. Write a parallel version of this method to estimate per(A). Explain why this is an embarrassingly parallel computation, despite the variation in the cost of evaluating the contribution of one permutation.
3. Analyze the speedup and communication cost. What is the optimal way to collect the results?
4. Implement a basic version using MPI and give the output of some runs which support your analysis.

2. Octtrees to store three dimensional data sets

To distribute n data points (x_i, y_i), $i = 1, 2, \ldots, n$ evently among p processors, we sketched in class the creation of Quadtrees to create clusters of data. In this assignment we will study the generalization to Octtrees to divide n points in space (x_i, y_i, z_i), $i = 1, 2, \ldots, n$, into clusters. Points belonging to different clusters are separated by planes, perpendicular to the coordinate axes.

We assume that n is so large that it is not possible to store all points into one memory module. The algorithm to distribute the n points into k clusters reads a file, beginning with n and k, followed by n lines containing three doubles on every line with the values of the x, y, and z coordinates of the points. The output of the algorithm is a sequence of numbers, the ith number gives the location of the cluster for the ith point. The end of the output contains k numbers, counting the number of points that went into each cluster.

1. Write pseudo code to define an algorithm to distribute n points into k clusters which runs in $O(n)$, using a fixed amount of storage. Assuming that the points are given in random order, think about how you might achieve an even distribution based on the relative locations of the first 1000 points.

2. Give pseudo code of a parallel algorithm to distribute the clusters among p processors, i.e.: $k = p$.

3. Analyze the computational speedup, communication cost, computation/communication ratio, and overall speedup.

4. Provide a basic implementation of the parallel algorithm and give the output of some runs on large sets of random data points to illustrate your analysis.

3. The modified Gram-Schmidt orthogonalization method

Denote the n columns of an n-by-n matrix A by $[\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n]$. Let $Q = [\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k]$ be an orthogonal matrix ($Q^{-1} = Q^T$, its inverse is its transpose) with the same column span as A. To compute Q we may use the modified Gram-Schmidt orthogonalization method, applying the formulas

$$\mathbf{p}_j = \mathbf{a}_j - \sum_{i=1}^{j-1} \mathbf{q}_i(\mathbf{q}_i^T \mathbf{p}_j), \quad \mathbf{q}_j = \frac{\mathbf{p}_j}{||\mathbf{p}_j||_2}, \quad \text{for } j = 1, 2, \ldots, n.$$

The expression $\mathbf{q}_i^T \mathbf{p}_j$ is the inner product of two vectors of length n. The cost of this algorithm is $O(n^3)$. Collecting the coefficients $\mathbf{q}_i^T \mathbf{a}_j$ into r_{ij} leads to an upper triangular matrix R such that $A = QR$.

1. Translate the formulas into pseudo code for an algorithm, leaving the inner products as one operation. Note that $||\mathbf{p}||_2 = \sqrt{\mathbf{p}^T \mathbf{p}}$.

2. Design a p-stage pipeline algorithm to compute one inner product, and illustrate it with a time space diagram for $p = 4$. Choose an appropriate value for n and show how to achieve a speedup arranging the multiple instances of the inner products appropriately.

3. Analyze the speedup. Can an n-stage pipeline give an $O(n^2)$ algorithm to compute Q?

4. Use MPI to simulate the pipelined computations. Give the output to demonstrate the correctness of your implementation and to give examples of the cost analysis.

4. The heat equation to diffuse images

As we did in our first project, we consider an image as given by an n-by-m matrix A of gray scales, where each gray scale is an integer between 0 and 255. For this question, we consider the gray scales as real numbers. To diffuse the image (like creating an impressionistic painting), we apply the equation of diffusion to the image. The heat equation is given by $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, for $u = u(x, y, t)$ the temperature at the point (x, y) at time t.

1. Describe the derivation of the iterative method, interpreting the operations in terms of the particulars of the application. For example, what could be the criteria for choosing the number of timesteps?

2. Describe a parallel version of the algorithm, taking into account that a double requires much more space than a single byte.

3. Analyze the speedup, communication cost, computation/communication ratio, and overall speedup.

4. Use MPI to implement a parallel version of this image diffusion. As an illustration you could diffuse the Mandelbrot set, or take any picture. With MATLAB it is relatively simple to create a matrix of gray scales from a .jpg file.