review of the first six chapters

The titles of the first six chapters of our textbook [1] can be summarized in a couple of words as (1) analysis of speedup; (2) using MPI; (3) embarrassingly parallel algorithms; (4) data partitioning; (5) pipelined computations; and (6) synchronous iteration. In chapters 1 and 2 we learned respectively how to analyze the performance of a parallel algorithm and how to implement it using MPI. These two ingredients appear throughout the following four chapters, which address techniques to create parallel programs. In this review we will give four problems, which may be typical for the take home midterm on Friday. Please also review the homework problems corresponding to the first 21 lectures.

1. Numerical Integration

Consider a continuous function $f(x,y)$ defined over a square domain $(x,y) \in [0,1] \times [0,1]$. To compute the volume between the surface $z = f(x,y)$ and the square $[0,1] \times [0,1]$, we could sample f at a regular grid of points (x_i, y_j), $x_i = ih$, and $y_j = jh$, for some $h > 0$, and use these samples to construct a piecewise linear approximation of the surface $z = f(x,y)$. The integral $\int_0^1 \int_0^1 f(x,y)dxdy$ is then approximated by the sum of all volumes between the triangle in the plane (spanned by $\{(x_i,y_j),(x_{i+1},y_j),(x_i,y_{j+1})\}$ or $\{(x_{i+1},y_j),(x_i,y_{j+1}),(x_{i+1},y_{j+1})\}$) and the plane approximating the surface $z = f(x,y)$ ($z = a_i x + b_j y + c_{ij}$, with (a_i, b_j, c_{ij}) obtained via interpolation at points spanning the triangles).

1. Use $h = 1/n$ to give the pseudo code of this algorithm, clear enough to determine the cost of this computation as function of n.
2. Write a parallel version of this volume computation. Show why this is an embarrassingly parallel computation.
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. Planar Convex Hull

Suppose we are given a list of n points in the plane, given by their coordinates (x_i, y_i), $i = 1, 2, \ldots, n$. We assume the input is not degenerate, i.e.: not all points lie on the same line. We want to compute all edges of the convex hull of the point configuration, also represented by a list of points given by their coordinates, ordered so that two consecutive points span an edge. The last edge is spanned by the last and first point in the list.

Any good algorithm to solve this problem is based on sorting. Take the center of the point configuration as the origin of the coordinate system and sort the points according to the angle their position vector makes with the x-axis. Next we walk through the sorted list and consider the orientation of the triangles spanned by three consecutive points in the list. A point in the middle of an inward pointing triangle will not span an edge and is removed.

1. Write clear pseudo code to implement this algorithm and give convincing arguments why the computational cost to solve this problem is $O(n \log_2(n))$.
2. Give the pseudo code of a parallel algorithm.
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. Divided Differences

The Newton form of polynomial q interpolating a function f in one variable uses divided differences, defined recursively as follows:

$$f[x_i] = f(x_i), \quad f[x_i, x_{i+1}, \ldots, x_{j-1}, x_j] = \frac{f[x_i, x_{i+1}, \ldots, x_{j-1}] - f[x_{i+1}, \ldots, x_{j-1}, x_j]}{x_i - x_j}, \quad \text{for } j > i.$$

The divided differences we need occur in the interpolating polynomial q: $q(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1})$.

1. Give the pseudo code of the algorithm to compute all needed divided differences and count the number of arithmetical operations to show that the computational cost is $O(n^2)$.
2. Design a p-stage pipeline algorithm to compute the table of divided differences. Describe the algorithm in detail for $p = n$.
3. Analyze the speedup. Can a p-stage pipeline reduce the cost to $O(n)$?
4. Use MPI to implement a basic version of the parallel algorithm. Give the output to demonstrate the correctness of your implementation and to give examples of the analysis of its cost.

4. Cellular Automata for Predator Prey

Consider the predator-prey model described in [1, page 191], summarized (and with some slight modifications) here. The three dimensional ocean is divided into cells which may be empty or contain either one fish (prey) or one shark (predator).

Fishes live along the following 4 rules: (1) with no free adjacent cell, the fish stays; (2) if some adjacent cells are free, then the fish chooses one cell at random; (3) with at least one adjacent free cell and when the fish has reached breeding age (i.e.: survived b iterations), the fish gives birth to one new fish which stays in the current cell, while the fish moves on to an adjacent cell; (4) the fish dies after n iterations.

The life of a shark is ruled as follows: (1) if there is at least one fish in an adjacent cell, the shark moves to a randomly chosen adjacent cell with fish and and eats the fish; (2) with no fish in adjacent cells, sharks follow the same rules (1) and (2) as fish do; (3) baby sharks are born just like fishes (see rule (3), eventually at a different breeding age); (4) sharks die either of old age, or when they have not eaten in m iterations.

1. Describe the data structures and choices for the variables to implement this cellular automaton. Take n as the number of cells in each one of three directions, assuming our ocean is just a cubical fish tank.
2. Design a parallel version of this simulation. Compared to the heat distribution problem, what are the extra difficulties you encounter? (I can think of at least three extra problems.) What solutions do you propose for these extra difficulties?
3. Analyze the speedup, communication cost, computation/communication ratio, and overall speedup.
4. Write a computer program to simulate this model. Can you make it realistic?

References

[1] Barry Wilkinson and Michael Allen. *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers.* Second Edition, Prentice Hall 2005.