Sample Mid-Term Exam 2

CS 4960-01, Fall 2008

Actual exam scheduled for November 24

Name: _____

Instructions: You have fifty minutes to complete this open-book, open-note, closed-computer exam. Please write all answers in the provided space plus the back of the exam. **Your programs do not have to be exactly right syntactically, but they should be reasonably close in the following sense: they should clearly indicate the language constructs or library functions to be used for parallelism, and they should clearly address all data distribution and synchronization concerns.**

1) One way to approximate π is to pick random points in an `N` by `N` square of the plane, and compute what percentage of the random points all within the circle of diameter `N` that has the same center as the square.

Write a program that computes a `double` approximation of π using $10,000,000$ random points, where a random point is a pair of random `int`s. Assume a function `random` that produces a random `int` between `MAX_INT` and `-MAX_INT`.

Your program should work with any number of threads P and obtain good speedup when $P << 10,000,000$ (assuming that `random` is independent for each thread, so calling `random` does not imply synchronization). See question 2 before writing your program.

2) Implement an MPI program that takes `N ints` and produces `N double` approximations of π, each using as many iterations as specified by the corresponding input integer`int`. Assume a function `read_input_data` that takes an array of `int`s and fills it with `N` numbers read from a file that is accessible only to process 0.

You can re-use any helper functions that you defined for the first question. Assume that integers in the input are evenly distributed (as opposed to having many more small numbers than large numbers in a given section of the input array), and assume that the number of processes P evenly divides `N`.

3) Imagine partitioning a `N` by `N` grid into `2` by `2` sub-grids, where `N` divides evenly by 2.

- A *rotation* of a sub-grid converts

$$\begin{matrix} n_{1,1} & n_{1,2} \\ n_{2,1} & n_{2,2} \end{matrix} \quad \rightarrow \quad \begin{matrix} n_{2,1} & n_{1,1} \\ n_{2,2} & n_{1,2} \end{matrix}$$

- A *swap* of columns i and $i+1$ for an off number i swaps the values at point `[`j`,`i`]` and `[`j`,`$i+1$`]` of the value at the former is larger than the value at the latter.

A *rotate+swap successor* of a grid is computed by rotating all sub-grids (no sub-grids overlap) and then swapping all pairs of columns (where the first of a pair is an odd column, so no pairs of columns overlap).

Write a Titanium program that computes the kth successor of an initial grid that is filled by a given function `readInputData`, which takes an `N` by `N` grid and fills it with initial values from a file that is accessible to process 0.

You can assume that the number of processes evenly divides $N/2$. Your program should perform reasonably if it is run in MPI mode. For maximum credit, you must minimize communication between processes.

Solutions

1)
```
int count(int iters) {
  int cnt = 0;
  while (iters--) {
    double x = random(), y = random();

    /* It would be enough here to say ‘‘check x,y’’ */
    if (sqrt(x * x + y * y) < MAX_INT)
      cnt++;
  }
  return cnt;
}

int main()
{
  int cnt, i;
#pragma omp parallel reduction(+:cnt)
  for (i = 0; i < 10000; i++) {
     cnt += count(10000);
  }

  /* It would be enough to say ‘‘get the result from cnt’’ */
  printf("%d\n", (double)(cnt * 4) / 100000000);
}
```

2)
```
int main() {
  int sz, me;
  int array[N], i;
  double results[N];

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &me);

  if (me == 0) {
    read_input_data(a);
    for (i = 1; i < sz; i++)
      MPI_Send(array + i * (N/sz), N/sz, MPI_INT, i, 0, MPI_COMM_WORLD);
  } else {
    MPI_Recv(array, N/sz, MPI_INT, 0, 0, MPI_COMM_WORLD);
  }

  for (i = 0; i < N/sz; i++) {
    /* It would be enough here to say ‘‘compute results[i] using array[i]’’ */
    results[i] = (double)(count(array[i]) * 4) / array[i]);
  }

  if (me == 0) {
    for (i = 1; i < sz; i++)
      MPI_Recv(results + i * (N/sz), N/sz, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
```

```
  } else {
    MPI_Send(result, N/sz, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
  }

  if (me == 0) {
    /* deliver ‘result’ array somehow... */
  }
}
```

3)
```
public void static main(String[] args)
{
   RectDomain<2d> all = [0:N-1,0:N-1];
   RectDomain<2d> sub = [0:1,0:1];
   int [2d] m = new int[all];

   if (Ti.thisProc() == 0) {
     readInputData(m);
   }
   m = broadcast m from 0;

   int my_count = N/2/Ti.numProcs();
   int my_offset = Ti.thisProc() * my_count;
   int rows = [0:my_count-1,0:N-1];

   int [2d] mine = new int[rows];
   int [2d] my_m = m.translate(m, [-my_offset, 0]);

   mine.copy(my_m, rows);

   for (int iter = 0; iter < k; i++) {
     /* It would be enough for this whole loop body to just say
        ‘‘compute a rotate and swap in ‘mine’ for rows 0
         through ‘my_count’ ’’ */
     for (int j = 0; j < my_count; j++) {
       for (int i = 0; i < N; i += 2) {
         int tl = mine[j, i];
         mine[j, i] = mine[j+1, i];
         mine[j+1, i] = mine[j+1, i+1];
         mine[j+1, i+1] = mine[j, i+1];
         mine[j, i+1] = tl;
       }
     }

     for (int j = 0; j < my_count; j++) {
       for (int i = 1; i < (N - 1); i += 2) {
         if (mine[j, i] > mine[j+1, i]) {
           int l = mine[j, i];
           mine[j, i] = mine[j+1, i];
           mine[j+1, i] = l;
         }
       }
```

```
      }
    }

    my_m.copy(mine, rows);

    if (Ti.thisProc() == 0) {
      /* report 'm' back somehow... */
    }
  }
```