0907521 Parallel and Distributed Systems (Fall 2012)
Midterm Exam

الاسم: رقم التسلسل: رقم الشعبة: ١

==========

Instructions: Time **50** minutes. Open book and notes exam. No electronics. Please answer all problems in the space provided and limit your answer to the space provided. **No questions are allowed**.

<Good Luck>

Q1. Perform a comparison between the 2-dimensional mesh and hypercube interconnection networks. Assume that both networks have N nodes each. Use the table below to make this comparison.

<10 marks>

Criterion	2D Mesh	Hypercube
Node cost	**4 ports**	**lg N ports**
Links cost	**Cost of $2(\sqrt{N}-1)\sqrt{N} = 2N - 2\sqrt{N}$ links**	**Cost of (N/2) lg N links**
Bisection width	**$\sqrt{N}$ links**	**(N/2) links**
Max latency	**Latency to pass through $2(\sqrt{N}-1)$ links**	**Latency to pass through lg N links**
Routing algorithm	**Direct, x then y**	**Direct, fix one dimension at a time**

Q2. Write an MPI program that computes a tree-structured global sum. Processor 0 should display this sum. Assume that **`comm_sz`** is a power of two and each processor gets its partial sum through

```
my_sum = get_sum(my_rank);
```

<10 marks>

```
#include <stdio.h>
#include <mpi.h>

int main(void) {
  int comm_sz, int my_rank;
  int half;
  double my_sum, sub_to;

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  my_sum = get_sum(my_rank);

  for (half = comm_sz/2; half>=1; half = half/2)
    if (my_rank < 2*half) {
      if (my_rank >= half )
        MPI_Send(&my_sum, 1, MPI_DOUBLE, my_rank - half, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      else {
        MPI_Recv(&sub_to, 1, MPI_DOUBLE, my_rank + half, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        my_sum = my_sum + sub_to;
      }
    }

  if (my_rank == 0)
    printf("The global sum is %lf\n", my_sum);

  MPI_Finalize();
  return 0;
}
```

Q3. Write a Pthreads program that implements the trapezoidal rule. This program should accept one command line argument to specify the number of threads. Assume that calling **`init()`** would initialize the shared variables **`n`**, **`a`**, **`b`**, and **`h`**, assume that **`n`** is divisible by **`thread_count`**, and the **`Trap()`** function is available for you (you don't need to write it). Use a shared variable for the sum of all the threads' computations. Use busy-waiting to enforce mutual exclusion.

<10 marks>

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int thread_count;
int n, turn=0;
double a, b, h, integral=0.0;

void* Do_trap(void* rank);

int main(int argc, char* argv[]) {
  long thread;
  pthread_t* thread_handles;

  init();

  thread_count = strtol(argv[1], NULL, 10);
  thread_handles = malloc (thread_count * sizeof(pthread_t));
  for (thread = 0; thread < thread_count; thread++)
    pthread_create(&thread_handles[thread], NULL, Do_trap,
                   (void*) thread);

  for (thread = 0; thread < thread_count; thread++)
    pthread_join(thread_handles[thread], NULL);

  free(thread_handles);
  return 0;
}

void* Do_trap(void* rank) {
  long my_rank = (long) rank;
  int local_n;
  double local_a, local_b, local_integral;

  local_n = n/thread_count;
  local_a = a + my_rank * local_n * h;
  local_b = local_a + local_n * h;
  local_integral = Trap(local_a, local_b, local_n, h);
```

```
    while(my_rank != turn) ;    //busy wait

    integral += local integral;
    turn++;

    return NULL;
}
```