**Lab 5: Parallel Programming**
Spring 2015
**Due: Friday, April 24th 2015**

In this lab, I want you to experiment with a few different parallel programming environments. We will look at shared memory computation (such as OpenMP and CUDA) and message-passing (such as OpenMPI) frameworks.

We're are going to use all of these to perform a simple task: adding together the values in an array.

**OpenMP**

Let's start with OpenMP. Here is a simple program that adds 14 numbers together – in parallel. Create a file named "omp_sum.c" which contains the following code:

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

const int num_values = 14;
int A[] = {100, 45, 46, 233, 67, 1, 23, 45, 89, 6, 4, 101, 98, 56};

int main()
{
     int sum = 0;
     int tid;
     int i;

#pragma omp parallel private(tid, sum)
     {
          tid = omp_get_thread_num();
          #pragma omp parallel for reduction(+:sum)
          for (i=0; i < num_values; ++i) {
               sum += A[i];
          }

          if (tid == 0) {
               printf("The sum is: %d\n", sum);
          }
     }
}
```

This program adds together the 14 values in A. We are going to store the result in a variable named "num". We also keep track

To compile it, type:

gcc -fopenmp omp_sum.c -o Sum

To run it, type:
OMP_NUM_THREADS=8 ./Sum

This will run it using 8 cores. It should print “914” (which is the sum of the values in the array).

**OpenMPI**

Now create a file named mpi_sum.c which contains this code:

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

const int num_values = 14;
int A[] = {100, 45, 46, 233, 67, 1, 23, 45, 89, 6, 4, 101, 98, 56};

int main(int argc, char* argv[])
{
     int result;
     int overall_sum = 0;
     int sum = 0;
     int nthreads, tid;
     int i;

     //Set up MPI
     result = MPI_Init(&argc, &argv);
     if (result < 0) {
          printf("Init failed.\n");
          exit(-1);
     }

     //Retrieve the number of processes
     result = MPI_Comm_size(MPI_COMM_WORLD, &nthreads);
     if (result < 0) {
          printf("size failed.\n");
          exit(-1);
     }

     //Retrieve the id number of this thread (its rank)
     result = MPI_Comm_rank(MPI_COMM_WORLD, &tid);
     if (result < 0) {
          printf("rank failed.\n");
          exit(-1);
     }
```

*(Continued on next page)*

```
     //Add together the values -- but only the ones assigned to
     my thread.

     for (i=0; i < num_values; i += nthreads) {
          sum += A[i+tid];
     }

     //Use a "reduction" to add the partial sums with only a
          logarithmic communication cost
     result = MPI_Reduce(&sum, &overall_sum, 1, MPI_INT, MPI_SUM,
          0, MPI_COMM_WORLD);
     if (result < 0) {
          printf("Reduce failed.\n");
          exit(-1);
     }

     //Wait until everyone has finished
     result = MPI_Barrier(MPI_COMM_WORLD);
     if (result < 0) {
          printf("Barrier failed.\n");
          exit(-1);
     }

     //Only the master thread prints the result
     if (tid == 0) {
          printf("The sum is: %d\n", overall_sum);
     }

     //Clean up MPI
     result = MPI_Finalize();
     if (result < 0) {
          printf("Finalize failed.\n");
          exit(-1);
     }
}
```

In this code, each process starts with an array, A, which contains 14 values. Since the values are compiled into the program itself, we don't have to send them to each process. However, we do need to add them in parallel and we can do this using a special function called MPI_Reduce. Reduce uses a tree structure to perform an operation efficiently. MPI builds a tree of processes. Each process computes a partial sum and then sends its partial sum to its parent in the tree. The parent adds its own sum to the partial sums it gets from the children and then sends the new sum to its parent. At the end of this process, the root node has the complete sum of all the numbers.

To compile, type:

mpicc mpi_sum.c -o MPISum

To run, type:

mpirun ./MPISum -np 4

**Glitter**

As with other projects, extend this one in a unique and creative way.

**Submission**

Create a tarball named "Lab5.tar.gz" and submit through the course web site as usual.