```
/
*****************************************************************************
*
* Name: Roger Shore
* Date: 9-8-2016
* Class: CSC4310
* Location: ~/HPU/csc4310/mpi
*
* FILE: sum_array.c
* DESCRIPTION:
*   MPI Example - Array Assignment - C Version
*   This program demonstrates a simple data decomposition. The master task
*   first initializes an array and then distributes an equal portion that
*   array to the other tasks. After the other tasks receive their portion
*   of the array, they perform an addition operation to each array element.
*   They also maintain a sum for their portion of the array. The master task
*   does likewise with its portion of the array. As each of the non-master
*   tasks finish, they send their sum to the master which computes a total.
*
*   An MPI collective communication call is used to collect the sums
*   maintained by each task.  Finally, the master task displays selected
*   parts of the final array and the global sum of all array elements.
*   NOTE: the number of MPI tasks must be evenly divided by 4.
*
* NOTE: This is an example.  If it was code to be released, only
* the master node would output any values.  The output from the other
* nodes is to verify the data flow and my understanding of the events
* that take place when the program is executed.
*
* To Compile:
*
* mpicc sum_array.c -o sum_array
*
* To execute:
*
* mpiexec --mca btl_tcp_if_include <ethernet_ID> -n 4 -hostfile actHosts
sum_array
*   either ifconfig or ip a to determine the ethernet ID for the machine.
* mpiexec --mca btl_tcp_if_include enp3s -n 4 -hostfile actHosts sum_array
*
*   NOTE: the number of MPI tasks must be evenly divided by 4.
****************************************************************************/

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define  ARRAYSIZE  1600
#define  MASTER     0
#define  TAG1  0
#define  TAG2  1
#define  TAG3  2

void cluster_node_process(int chunksize, int taskid);
void master_node_process(int chunksize, int numtasks, int taskid);
double findSum(double data[], int myoffset, int chunk, int myid);
double initArray(double data[], int n);
void sampleOutput(double data[],int chunksize, int numtasks);

int main (int argc, char *argv[])
{
   int numtasks, taskid, rc=1;
   int chunksize;

   /***** Initializations *****/
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
   if (numtasks % 4 != 0) {
      printf("Quitting. Number of MPI tasks must be divisible by 4.\n");
      MPI_Abort(MPI_COMM_WORLD, rc);
      exit(0);
   }

   MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

   printf ("MPI task %d has started...\n", taskid);
   chunksize = (ARRAYSIZE / numtasks);

   /***** Master task only ******/
   if (taskid == MASTER){
      master_node_process(chunksize,numtasks, taskid);
   }


   /***** Non-master tasks only *****/
   if (taskid > MASTER) {
      cluster_node_process(chunksize, taskid);

   } /* end of non-master */


   MPI_Finalize();

}   /* end of main */


/* master_node_process - init an array of data to process
 *                       Send on portion of the array to
 *                       other cluster nodes.  Keep one chunk
 *                       for the master
 *                       Find the sum of the data
 *                       Receive the results from each node.
 * precondition: chunksize, numtasks, and taskid are established
 *                in the main for all processes
 * postcondition: nothing is returned from the function.
 */
void master_node_process(int chunksize, int numtasks, int taskid)
{
   double  data[ARRAYSIZE];
   int offset, source;
   struct timeval startTime, stopTime;
   double start, stop, diff;
   double mysum, sum;
   double t1,t2,result;
   MPI_Status status;
   double sum2,tmpsum;
   gettimeofday(&startTime,NULL);
   t1 = MPI_Wtime();

   sum = initArray(data,ARRAYSIZE);
   printf("What we should get - Initialized array sum = %e\n",sum);

   /* Send each task its portion of the array - master keeps 1st part */
   offset = chunksize;
   for (int dest=1; dest<numtasks; dest++) {
      MPI_Send(&offset, 1, MPI_INT, dest, TAG1, MPI_COMM_WORLD);
      MPI_Send(&data[offset], chunksize, MPI_DOUBLE, dest, TAG2,
MPI_COMM_WORLD);
      printf("Sent %d elements to task %d offset= %d\
n",chunksize,dest,offset);
      offset = offset + chunksize;
   }

   /* Master does its part of the work */
   offset = 0;
   mysum = findSum(data, offset, chunksize, taskid);
   sum2 = mysum;

   /* Wait to receive results from each task */
   /* Get final sum and print sample results */
   for (int i=1; i<numtasks; i++) {
      source = i;
      MPI_Recv(&offset, 1, MPI_INT, source, TAG1, MPI_COMM_WORLD, &status);
      MPI_Recv(&data[offset], chunksize, MPI_DOUBLE, source, TAG2,
               MPI_COMM_WORLD, &status);
      MPI_Recv(&tmpsum, 1, MPI_DOUBLE, source, TAG3,
               MPI_COMM_WORLD, &status);
      sum2 += tmpsum;
   }

   sampleOutput(data,chunksize,numtasks);
   printf("*** What we got - Final sum= %e ***\n",sum2);

   gettimeofday(&stopTime,NULL);
   t2 = MPI_Wtime();
   start = startTime.tv_sec + (startTime.tv_usec/1000000.0);
   stop = stopTime.tv_sec + (stopTime.tv_usec/1000000.0);

   diff = stop - start;
   result = t2-t1;

   printf("Time %f\n",diff);
   printf("MPI_Wtime %f\n",result);
}


/* cluster_node_process - receive a chunk of data from the master
 *                        find the sum of the data
 *                        send back the data and sum.
 * NOTE: there is no reason to send the data back, just demo
 *       how to send the array back.
 * precondition: chunksize and taskid are established in the main
 *               for all processes
 * postcondition: nothing is returned from the function.
 */
void cluster_node_process(int chunksize, int taskid)
{
   int source, dest, offset;
   MPI_Status status;
   double mysum;
   double  data[ARRAYSIZE];

   /* Receive my portion of array from the master task */
   source = MASTER;
   MPI_Recv(&offset, 1, MPI_INT, source, TAG1, MPI_COMM_WORLD, &status);
   MPI_Recv(&data[offset], chunksize, MPI_DOUBLE, source, TAG2,
MPI_COMM_WORLD, &status);

   mysum = findSum(data, offset, chunksize, taskid);

   /* Send my results back to the master task */
   dest = MASTER;
   MPI_Send(&offset, 1, MPI_INT, dest, TAG1, MPI_COMM_WORLD);
   MPI_Send(&data[offset], chunksize, MPI_DOUBLE, MASTER, TAG2,
MPI_COMM_WORLD);
   MPI_Send(&mysum, 1, MPI_DOUBLE, MASTER, TAG3, MPI_COMM_WORLD);

   printf("\t\t\t\t\t*** Final sum= %e ***\n",mysum);
}

/* initArray - initialize an array with increasing values of i starting at 0
 * precondition: The data array is empty and is large enough to hold n ints
 * postcondition: the data array is updated.
 */
double initArray(double data[], int n)
{
   /* Initialize the array */
   double sum = 0;
   for(int i=0; i<n; i++) {
      data[i] =  i * 1.0;
      sum = sum + data[i];
   }
   return sum;
}
/* findSum - find the sum of the elements within the chunk of data
 * precondition: The data array is loaded iio  a specific chunk
 * postcondition: the data array is updated.
 */
double findSum(double data[], int myoffset, int chunk, int myid)
{
   int i;
   double mysum;
   mysum = 0;
   for(i=myoffset; i < myoffset + chunk; i++) {
      mysum = mysum + data[i];
   }
   printf("\t\t\t\t\tTask %d mysum = %e\n",myid,mysum);
   return(mysum);
}


/* update - output the first 5 numbers each chunk of data
            passed to the non-master processes
 * precondition: The data array contains a collection of doubles
 *               chunksize is the number of doubles in each set
 *               of values.  numtasks is the number of tasks
 *               (or number of subsets) that process the data.
 * postcondition: no updates or changes in the data
 */
void sampleOutput(double data[],int chunksize, int numtasks)
{
   printf("Sample results: \n");
   int offset = 0;
   for (int i=0; i<numtasks; i++) {
      for (int j=0; j<5; j++)
         printf("  %e",data[offset+j]);
       printf("\n");
       offset = offset + chunksize;
   }
}
```