Assignment 3
Intro to HPC

Wesley Liao
2019-10-13

1. p1.c:

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

typedef struct {
  int max_iter;
  double t0;
  double tf;
  double xmax[12];
  double xmin;
} Pars;

int main(int argc, char *argv[])
{
  int myid, numprocs, left, right;
  Pars buffer, buffer2;
  MPI_Request request;
  MPI_Status status;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  MPI_Datatype MPI_Pars;
  int count = 5;
  int blocklengths[5] = {1, 1, 1, 12, 1};
  MPI_Aint offsets[5];
    offsets[0] = offsetof(Pars, max_iter);
    offsets[1] = offsetof(Pars, t0);
    offsets[2] = offsetof(Pars, tf);
    offsets[3] = offsetof(Pars, xmax);
    offsets[4] = offsetof(Pars, xmin);
  MPI_Datatype types[5] = {MPI_INT, MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE, MPI_DOUBLE};

  MPI_Type_create_struct(
    count,
    blocklengths,
    offsets,
    types,
    &MPI_Pars);
  MPI_Type_commit(&MPI_Pars);

  right = (myid + 1) % numprocs;
  left = myid - 1;
  if (left < 0)
    left = numprocs - 1;

  // send myid to the left
  buffer.max_iter = myid;

  MPI_Sendrecv(&buffer, 1, MPI_Pars, left, 123,
     &buffer2, 1, MPI_Pars, right, 123, MPI_COMM_WORLD, &status);

  printf(" Process %d received %d\n",myid,buffer2.max_iter);

  MPI_Finalize();
  return 0;
}
```

2. p2.c:

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

int main(int argc,char **argv) {
  MPI_Init(&argc,&argv);
  int rank,p,i, root = 0;
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&p);

  // Make the global vector size constant
  int global_vector_size = 10000;

  int local_vector_size = (global_vector_size / p);

  double pi = 4.0*atan(1.0);

  // initialize the vectors
  double *a, *b;
  a = (double *) malloc(
       local_vector_size*sizeof(double));
  b = (double *) malloc(
       local_vector_size*sizeof(double));
  for (i=0;i<local_vector_size;i++) {
    a[i] = sqrt(i+(rank*local_vector_size));
    b[i] = sqrt(i+(rank*local_vector_size));
  }

  double mysum = 0.0;
  for(i = 0; i < local_vector_size; i++)
  {
    mysum += a[i]*b[i];
  }

  // compute the dot product
  double total = 0.0;
  MPI_Reduce(
    &mysum,
    &total,
    1,
    MPI_DOUBLE,
    MPI_SUM,
    0,
    MPI_COMM_WORLD);

  if ( rank == 0) {
    printf("The dot product is %g.  Answer should be: %g\n",
           total, 0.5*global_vector_size*(global_vector_size-1));
  }

  free(a);
  free(b);
  MPI_Finalize();
  return 0;
}
```

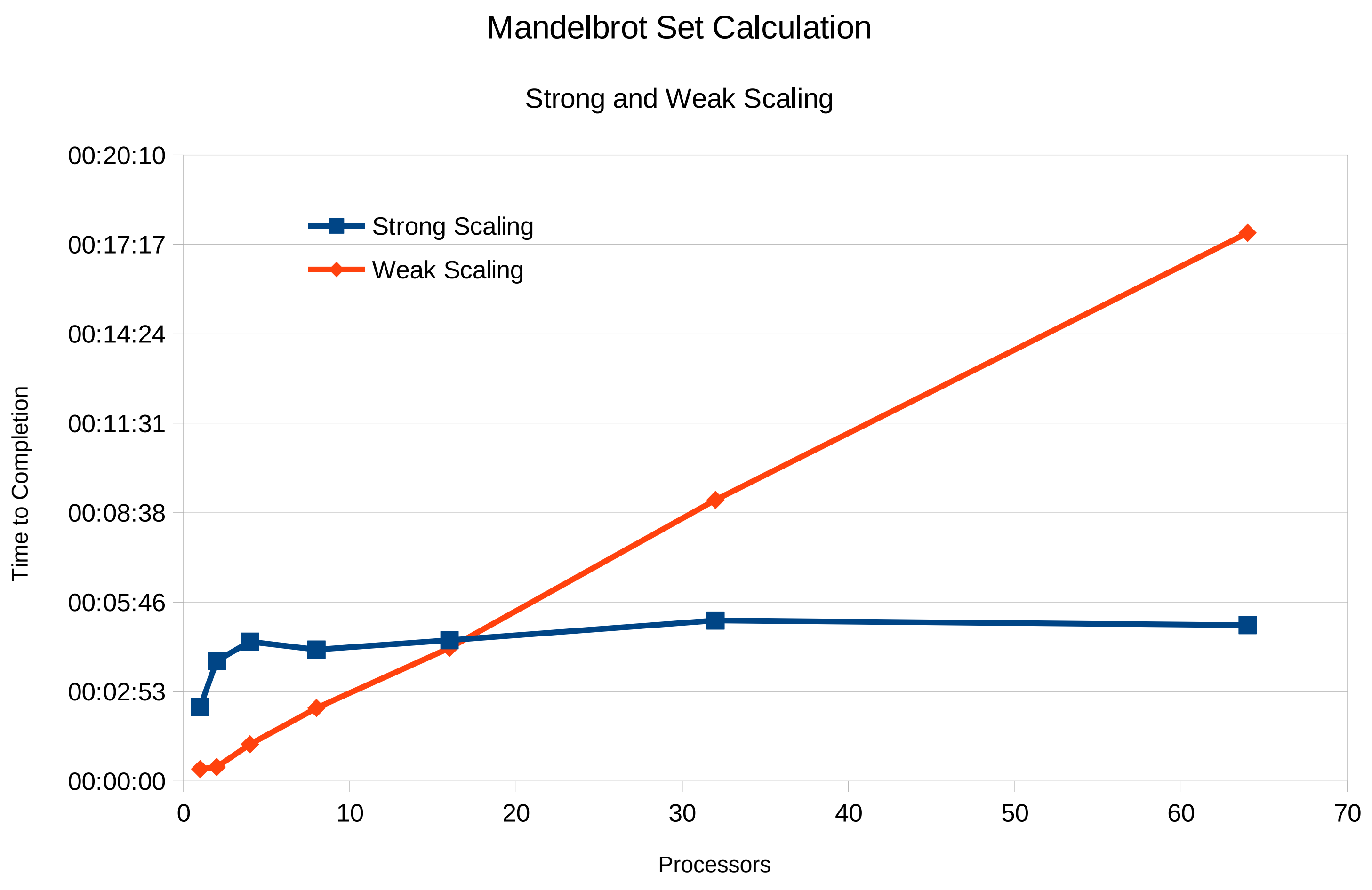
3.

Strong Scaling

Processors	Global Job Size	Individual Job Size	Start Time	End Time	Difference
1	268435456	268435456	08:16:46 PM	08:19:09 PM	00:02:23
2	268435456	134217728	08:19:09 PM	08:23:01 PM	00:03:52
4	268435456	67108864	08:23:01 PM	08:27:30 PM	00:04:29
8	268435456	33554432	08:27:30 PM	08:31:44 PM	00:04:14
16	268435456	16777216	08:31:44 PM	08:36:16 PM	00:04:32
32	268435456	8388608	08:36:16 PM	08:41:26 PM	00:05:10
64	268435456	4194304	08:41:26 PM	08:46:27 PM	00:05:01

Weak Scaling

Processors	Global Job Size	Individual Job Size	Start Time	End Time	Difference
1	16777216	16777216	08:46:27 PM	08:46:50 PM	00:00:23
2	33554432	16777216	08:46:50 PM	08:47:17 PM	00:00:27
4	67108864	16777216	08:47:17 PM	08:48:28 PM	00:01:11
8	134217728	16777216	08:48:28 PM	08:50:49 PM	00:02:21
16	268435456	16777216	08:50:49 PM	08:55:06 PM	00:04:17
32	536870912	16777216	08:55:06 PM	09:04:09 PM	00:09:03
64	1073741824	16777216	09:26:36 PM	09:44:15 PM	00:17:39



4. The CSP model uses multiple sequential processes running at the same time, periodically reaching barriers where partial results are communicated between the processes.