CSCI 596: HW 3

M. Oneeb H. Khan

September 21, 2021

1 Source Codes

1.1 global_pi.c

The text below is the entire code for the *global_pi.c* file, with the requisite modifications to the *global_sum* function from Assignment 2.

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

#define NBIN 1000000000
int nprocs;  /* Number of processes */
int myid;    /* My rank */

double global_sum(double partial) {
  /* Write your hypercube algorithm here */
  double mydone, hisdone;
  int bitvalue, partner;
  MPI_Status status;
  mydone = partial;
  for(bitvalue = 1; bitvalue < nprocs; bitvalue *= 2) {
    partner = myid ^ bitvalue;
    MPI_Send(&mydone, 1, MPI_DOUBLE, partner, bitvalue, MPI_COMM_WORLD);
    MPI_Recv(&hisdone, 1, MPI_DOUBLE, partner, bitvalue, MPI_COMM_WORLD, &status);
    mydone = mydone + hisdone;
  }
}

int main(int argc, char *argv[]) {
  double partial, sum, pi, step, x;
  double cpu1, cpu2;
  long long i;

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

  cpu1 = MPI_Wtime();
  step = 1.0/NBIN;
  for (i=myid; i<NBIN; i+=nprocs) {
    x = (i+0.5)*step;
    sum += 4.0/(1.0+x*x);
  }
  partial = sum*step;
  pi = global_sum(partial);
  cpu2 = MPI_Wtime();
```

```
40
41   if (myid == 0) {
42     printf("Pi = %le\n", pi);
43     printf("NProcs & Execution time (s) = %d %le\n",nprocs, cpu2-cpu1);
44   }
45   MPI_Finalize();
46   return 0;
47 }
```

1.2 global_pi_iso.c

The text below is the entire code for the *global_pi_iso.c* file, with the requisite modifications to the *global_pi.c* file.

```
1  #include "mpi.h"
2  #include <stdio.h>
3
4  #define NPERP 1000000000 /* Number of quadrature points per processor */
5  int nprocs;  /* Number of processes */
6  int myid;    /* My rank */
7
8  double global_sum(double partial) {
9    /* Write your hypercube algorithm here */
10   double mydone, hisdone;
11   int bitvalue, partner;
12   MPI_Status status;
13   mydone = partial;
14   for(bitvalue = 1; bitvalue < nprocs; bitvalue *= 2) {
15     partner = myid ^ bitvalue;
16     MPI_Send(&mydone, 1, MPI_DOUBLE, partner, bitvalue, MPI_COMM_WORLD);
17     MPI_Recv(&hisdone, 1, MPI_DOUBLE, partner, bitvalue, MPI_COMM_WORLD, &status);
18     mydone = mydone + hisdone;
19   }
20 }
21
22 int main(int argc, char *argv[]) {
23   double partial, sum, pi, step, x;
24   double cpu1, cpu2;
25   long long i;
26
27   MPI_Init(&argc, &argv);
28   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
29   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
30
31   long long NBIN;
32   NBIN = (long long) NPERP * nprocs;
33
34   cpu1 = MPI_Wtime();
35   step = 1.0/NBIN;
36   for (i=myid; i<NBIN; i+=nprocs) {
37     x = (i+0.5)*step;
38     sum += 4.0/(1.0+x*x);
39   }
40   partial = sum*step;
41   pi = global_sum(partial);
42   cpu2 = MPI_Wtime();
43   if (myid == 0) {
44     // avg = sum/nprocs;
45     // printf("Global average = %le\n", avg);
```

```
    printf("Pi = %le\n", pi);
    printf("NProcs & Execution time (s) = %d %le\n",nprocs, cpu2-cpu1);
  }
  MPI_Finalize();
  return 0;
}
```

2 global_pi.out Printout

The figure below shows the printout of *global_pi.c* and *global_pi_iso.c* running on 4, 2 and 1 processors.

```
========================================
SLURM_JOB_ID = 5979381
SLURM_JOB_NODELIST = d05-[15,26-28]
TMPDIR = /tmp/SLURM_5979381
========================================
##### Strong scaling #####
Pi = 3.141593e+00
NProcs & Execution time (s) = 4 9.875478e-01
Pi = 3.141593e+00
NProcs & Execution time (s) = 2 1.954865e+00
Pi = 3.141593e+00
NProcs & Execution time (s) = 1 3.741783e+00
##### Weak scaling   #####
Pi = 3.141593e+00
NProcs & Execution time (s) = 4 3.830135e+00
Pi = 3.141593e+00
NProcs & Execution time (s) = 2 3.830326e+00
Pi = 3.141593e+00
NProcs & Execution time (s) = 1 3.748947e+00
```

Figure 1: Printout of global_pi.c and global_pi_iso.c

3 Efficiency Plots

3.1 Fixed Problem Size Parallel Efficiency

The figure below shows the plot of the parallel efficiency for the strong scaling instance.

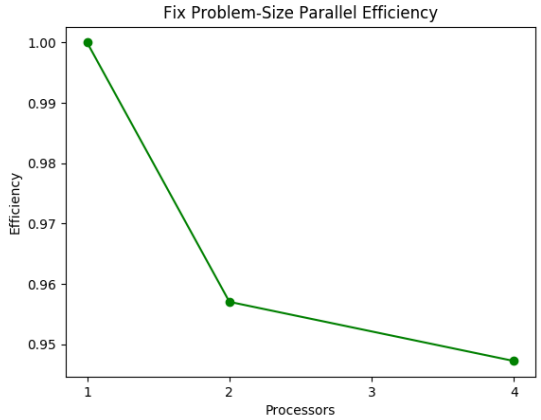

Figure 2: Fixed Problem Size Parallel Efficiency

3.2 Isogranular Parallel Efficiency

The figure below shows the plot of the parallel efficiency for the weak scaling instance.

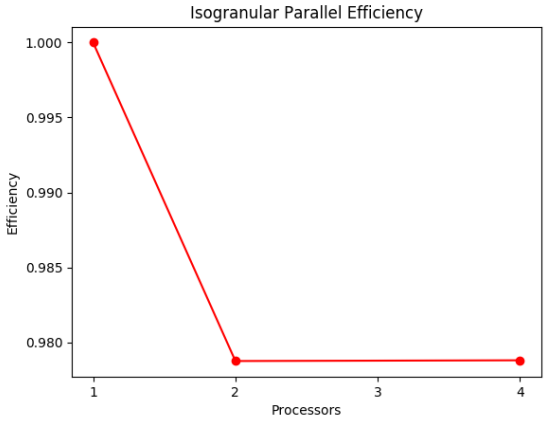

Figure 3: Isogranular Parallel Efficiency