**SF2568: Homework 1**

Yue Jiao

911024-7799

1. A processor is the physical device that do all the computation mean while a process is the computational activity assigned to a processor.
2. We assume there are totally P floating operations to compute. 10%P shall be computed sequentially and 90%P shall be computed parallelly on 100 processors. Then we assume is shall take t seconds to compute this. So . So, in average the performance measured in Gflops shall be .
3. The system efficiency is define as the kvot of the systems parallel speedup Sp and the number of processors. The systems parallel speedup is at most according to its definition. So , which means that is never bigger than 1.
4. We assume the elements that shall be sorted are totally 10p and labeled as . We shall do exactly the same thing like the original ranking sort algorithm except each node shall calculate rank of 10 elements instead of one and each node shall send back a map of 10 elements and their ranks.
5. We assume that the program can be parallelized perfectly which means that the parallel speedup is equal with P during the parallel phases.
   1. We assume that the serial running time is t. We know that the time to run the code on one processor is 64s and on 8 processors is 22s. Because of that the computational power of a processor is fixed, so

So, 72.7% of the parallel running time is spent in the serial section.

* 1. The parallel speedup is .
  2. We assume that the program runs in T second on one processor. Then is runned in the first phase, is runned in the second phase and in the third. The first phase runs optimal on 5 processors then the parallel speedup is at most 5 which means the processing time of first phase shall be shorted to . In the same way the second phase will be shorted to and the third phase will be shorted to . So, the totally parallel speedup is then .

1. The result with is showed below. I couldn’t get the exact same color but it looks fairly similar.

The code is listed below and I paste it here as well: <https://paste.ubuntu.com/26536377/>

/\*

Result can be plotted with the following matlab command:

\* load('color.txt')

\* image(color'\*12)

\*/

#include <stdio.h>

#include <string.h>

#include <complex.h>

#include <math.h>

#include <mpi.h>

double cabs(double complex z) {

return sqrt(creal(z) \* creal(z) + cimag(z) \* cimag(z));

}

int cal\_pixel(double complex d, double b, int N) {

int count = 0;

double complex z = 0;

while ((cabs(z) < b) && (count < N)) {

z = z \* z + d;

count++;

}

// printf("%d\n", count);

return count;

}

int main(int argc, char \*\*argv) {

int N = 256, w = 2048, h = 2048;

double b = 2;

int ierr, size, rank;

ierr = MPI\_Init(&argc, &argv);

ierr = MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

ierr = MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

//printf("Hello world! I'm process %i out of %i processes\n", rank, size);

int length = w / size \* h;

unsigned char color[length];

if (rank == 0) {

FILE \* fp;

fp = fopen("color.txt", "w");

double dx = 2\*b/(w - 1), dy = 2\*b/(h - 1);

int wp = w / size, hp = h;

for (int x = 0; x < wp; x++) {

double dreal = (x) \* dx - b;

for (int y = 0; y < hp; y++) {

double dimag = (y) \* dy - b;

double complex d = dreal + dimag \* I;

color[x \* h + y] = cal\_pixel(d, b, N);

}

}

for (int j = 0; j < length; j++) {

fprintf(fp, "%hhu ", color[j]);

if (!((j + 1) % h))

fprintf(fp, "\n");

}

for (int i = 1; i < size; i++) {

// if(i != 0) {

MPI\_Status \*status;

ierr = MPI\_Recv(color, length, MPI\_UNSIGNED\_CHAR, i, i, MPI\_COMM\_WORLD, status);

// printf("i: %d\n", i);

// }

for (int j = 0; j < length; j++) {

fprintf(fp, "%hhu ", color[j]);

if (!((j + 1) % h))

fprintf(fp, "\n");

}

}

fclose(fp);

} else {

double dx = 2\*b/(w - 1), dy = 2\*b/(h - 1);

int wp = w / size, hp = h;

double xoff = rank \* wp, yoff = 0;

for (int x = 0; x < wp; x++) {

double dreal = (x + xoff) \* dx - b;

for (int y = 0; y < hp; y++) {

double dimag = (y + yoff) \* dy - b;

double complex d = dreal + dimag \* I;

color[x \* h + y] = cal\_pixel(d, b, N);

}

}

ierr = MPI\_Send(color, length, MPI\_UNSIGNED\_CHAR, 0, rank, MPI\_COMM\_WORLD);

}

ierr = MPI\_Finalize();

return 0;

}

A magnified part of the result is showed below:

