WYDZIAŁ FIZYKI I ASTRONOMII
UNIWERSYTET WROCŁAWSKI

Projekt MPI

Paweł Grabiński

24 stycznia 2015

1 OBLICZENIA RÓWNOLEGŁE PRZY UŻYCIU TECHNOLOGII MPI

1.1 TREŚĆ PROBLEMU

Wygeneruj zbiór Mandelbrota przy pomocy technologii MPI dzieląc obszar płaszczyzny zespolonej na którym wykonujesz obliczenia na równe części i przyporządkuj je różnym wątkom.

1.2 METODOLOGIA ROZWIĄZANIA

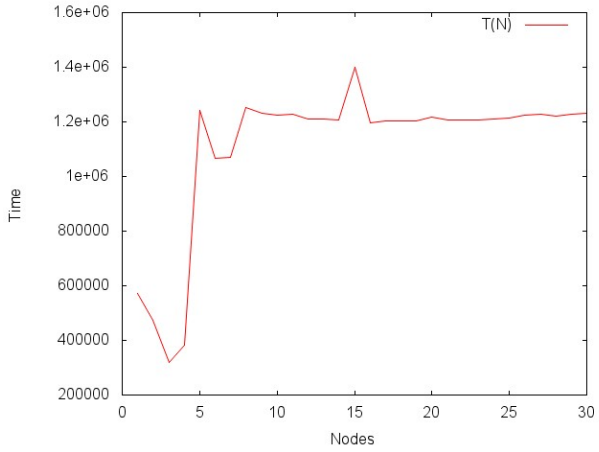
Podczas pracy programu zachodzą następujące działania:

1. Jako parametr wywołania programu jest podana żądana ilość wątków.
2. Dla każdego wątku wywoływana jest funkcja, do której przekazy jest zakres na jakim ma działać funkcja.
3. Po wykananiu funkcji następuje synchronizacja watków.

Czas pracy funkcji obliczeniowych mierzony jest przy pomocy funkcji $gettimeofday()$ z dokładnością do $1\,\mu s$.

1.3 WYNIKI

By zobaczyć różnicę w wydajności zbadaliśmy zależność czasu od ilości wątków. Zostało wykanane 10 pomiarów dla każdej liczby wątków. Obszar przestrzeni, na której wykonywaliśmy obliczenia była wielkości 1000×1000.



Zgodnie z prawem Ahmdala:

$$T_p = T_1 \times \left(1 - f + \frac{f}{N}\right)$$

Gdzie:

- T_p to czas równoległy
- T_1 to czas szeregowy
- N to liczba wątków
- f to część obliczeń, która została urównoleglona

Widzimy, że dla $N \in \{2,3,4\}$ następuje przyspieszenie. Jednak dla $N \geq 5$ koszty uruchamiania i zarządzania wątkami przewyższają zyski z urównoleglenia, które nie są uwzględnione w prawie Ahmdala.

1.4 Kod źródłowy rozwiązania

```
#include<stdio.h>
#include<mpi.h>
#include<sys/time.h>
#define W 1000
#define H 1000
#define T 300

/*pixel*/
typedef struct{
   int R;
   int G;
   int B;
}RGB;

RGB o[H][W];

void Mandelbrot(int Start,int Total){
    int i,j,t,WMax;
```

```
19      if((int)((Start+1)*W/(float)(Total))>W) WMax=W;
        else WMax=(int)((Start+1)*W/(float)(Total));
21      for(i=0;i<H;i++){
            for(j=(int)(Start*W/(float)(Total));j<WMax;j++){
23              float x0=j/(float)(W)*3.5-2.5,y0=i/(float)(H)*2-1,x=0,y=0,xt
    =0;
                t=0;
25          while(x*x+y*y<4 && t<T){
                    xt=x*x-y*y+x0;
27              y=2*x*y+y0;
                x=xt;
29                  t++;
                }
31              if(t/(float)(T)<1/(float)(3)){
                    o[i][j].R=(int)(t/(float)(T)*3*255);
33              }
                else{
35                  if(t/(float)(T)<2/(float)(3)){
                        o[i][j].R=255;
37                      o[i][j].G=(int)(t/(float)(T)*3/2*255);
                    }
39                  else{
                        o[i][j].R=255;
41                      o[i][j].G=255;
                        o[i][j].B=(int)(t/(float)(T)*255);
43                  }
                }
45          }
        }
47  }
49
    int main(int argc, char *argv[]){
51      struct timeval startCPU,stopCPU;
        int n_N;
53      gettimeofday(&startCPU,NULL);
        MPI_Init(&argc, &argv);
55      int rank,totproc;
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
57      MPI_Comm_size(MPI_COMM_WORLD, &totproc);
        Mandelbrot(rank,totproc);
59      if(rank==0) n_N=totproc;
        MPI_Finalize();
61      if(rank==0){
            gettimeofday(&stopCPU,NULL);
63          long int d_TotTimeCPU=(stopCPU.tv_sec-startCPU.tv_sec)*1000000+(
    stopCPU.tv_usec-startCPU.tv_usec);
            printf("%d %ld\n",n_N,d_TotTimeCPU);
65      }
        return 0;
67  }
```

pmb.c