Programowanie równoległe
Laboratorium 12

Sprawozdanie

Zadanie:

Wykorzystując przedstawiony sposób wczytania obrazu .pbm należy opracować sposób przesyłania informacji o pikselach i ich sąsiadach pomiędzy procesami.

- Należy dodać pomiar czasu, który powinien obejmować jedynie operację filtrowania.
- Można zastosować dowolny typ filtru, który należy krótko opisać w sprawozdaniu.
- Należy obliczyć przyspieszenie wyliczone zgodnie z prawem Amdahla.
- Do sprawozdania należy dołączyć wykres czasu i przy spieczenia.
- WAŻNE! Wyniki należy uzasadnić

Kod źródłowy rozwiązania:

```
#include <iostream>
#include <mpi.h>
#include "pbm.h"

#define FILTER_SIZE 3

int filter[FILTER_SIZE][FILTER_SIZE] = {
    { 1, 1, 1 },
    { 1, 1, 1 },
    { 1, 1, 1 }
};

int main(int argc, char *argv[]) {
    image in;
    image out;

    int npes;
    int myrank;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    if (myrank == 0) {
      readInput(argv[1], &in);

      out.height = in.height;
```

```
      out.width = in.width;
      out.maxValue = in.maxValue;

      memcpy(out.type, in.type, TYPE_LEN + 1);
    }

    MPI_Bcast(&in.width, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&in.height, 1, MPI_INT, 0, MPI_COMM_WORLD);

    uchar *inPixel = (uchar*) malloc(sizeof(uchar) * in.height *
in.width);
    if (myrank == 0) {
      memcpy(inPixel, in.pixel, sizeof(uchar) * in.height *
in.width);
    }
    MPI_Bcast(inPixel, in.height * in.width, MPI_UNSIGNED_CHAR,
0, MPI_COMM_WORLD);

    uchar *localPixel = (uchar*) malloc(sizeof(uchar) *
in.height * in.width);
    out.pixel = (uchar*) malloc(sizeof(uchar) * in.height *
in.width);

    double start, end;
    start = MPI_Wtime();

    if (npes == 1 || myrank != 0) {
      for (int i = npes == 1 ? 0 : myrank - 1; i < in.height; i
+= npes == 1 ? 1 : npes - 1) {
           for (int j = 0; j < in.width; ++j) {
                int sum = 0;
                int denominator = 0;

                for (int ii = 0; ii < FILTER_SIZE; ++ii) {
                     for (int jj = 0; jj < FILTER_SIZE; ++jj) {
                          int ri = i + ii - int(FILTER_SIZE) /
2;
                          int rj = j + jj - int(FILTER_SIZE) /
2;

                          if (ri >= 0 && rj >= 0 && ri <
in.height && rj < in.width) {
                               sum += *(inPixel + ri *
in.width + rj) * filter[ii][jj];
                               denominator += filter[ii][jj];
```

```
                        }
                    }
                }

                if (denominator) {
                    uchar value = sum / denominator;
                    memcpy(localPixel + i * in.width + j,
&value, sizeof(uchar));
                }
            }
        }
    }

    MPI_Reduce(localPixel, out.pixel, in.width * in.height,
MPI_UNSIGNED_CHAR, MPI_SUM, 0, MPI_COMM_WORLD);

    end = MPI_Wtime();

    MPI_Finalize();

    if (myrank == 0) {
      writeData(argv[2], &out);
      std::cout << "End of process " << end - start <<
std::endl;
    }

    return 0;
}
```

Opisanie filtru:

Jako filtr przykładowy zastosowany został filtr uśredniający. Ten filtr jest podstawowym filtrem dolnoprzepustowym, jego wynikiem jest uśrednienie każdego piksla razem ze swoimi ośmioma sąsiadami:

1	1	1
1	1	1
1	1	1

Wydruki programu dla różnych parametrów:

```
> mpirun -np 1 ./main.x86_64 casablanca.pgm
casablanca.pgm.filtered
< End of process 0.00900722
```

```
> mpirun -np 2 ./main.x86_64 casablanca.pgm
casablanca.pgm.filtered
< End of process 0.00949478
```

```
> mpirun -np 4 ./main.x86_64 casablanca.pgm
casablanca.pgm.filtered
< End of process 0.00360703
```

```
> mpirun -np 8 ./main.x86_64 casablanca.pgm
casablanca.pgm.filtered
< End of process 0.00313902
```

```
> mpirun -np 16 ./main.x86_64 casablanca.pgm
casablanca.pgm.filtered
< End of process 0.0423176
```

```
> mpirun -np 1 ./main.x86_64 boulevard.pgm
boulevard.pgm.filtered
< End of process 1.80417
```

```
> mpirun -np 2 ./main.x86_64 boulevard.pgm
boulevard.pgm.filtered
< End of process 1.8334
```

```
> mpirun -np 4 ./main.x86_64 boulevard.pgm
boulevard.pgm.filtered
< End of process 0.728327
```

```
> mpirun -np 8 ./main.x86_64 boulevard.pgm
boulevard.pgm.filtered
```

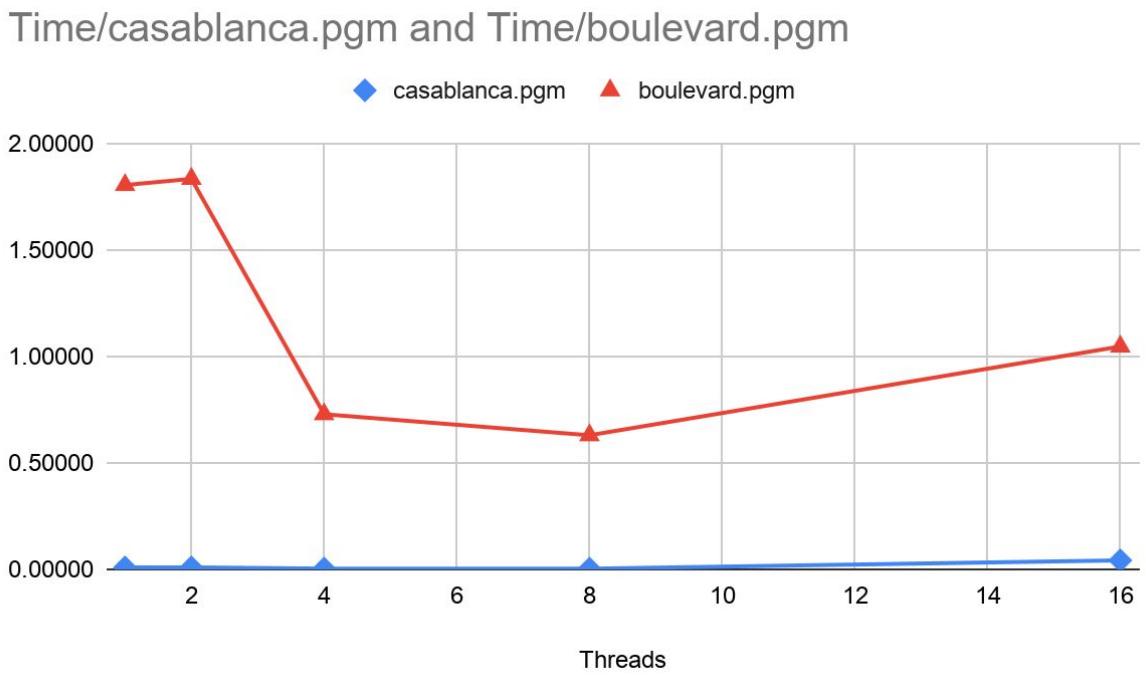
```
< End of process 0.629624
```

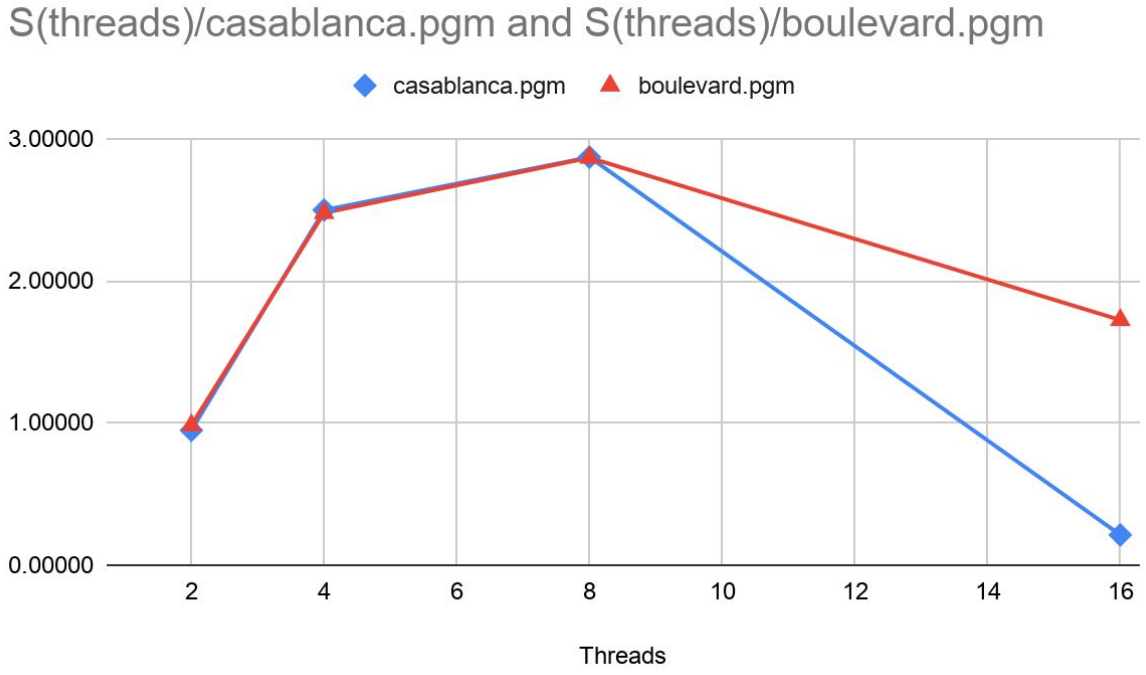
```
> mpirun -np 16 ./main.x86_64 boulevard.pgm
boulevard.pgm.filtered
< End of process 1.04636
```

Wyliczone przyspieszenie Amdahla:

	casablanca.pgm	boulevard.pgm
S(2)	0.94865	0.98406
S(4)	2.49713	2.47714
S(8)	2.86944	2.86547
S(16)	0.21285	1.72423

Wykresy czasu i przyspieszenia:





Uzasadnienie wyników:

- Przyspieszenie dla dwóch procesów nie jest istotne ponieważ wykorzystanie zasobów przez operacje synchronizacji procesów wzrasta.

- Największe przyspieszenie jest dla ilości procesów równej ilości jąder logicznych procesora, 8 w danym przypadku.
- Dalej przyspieszenie maleje dlatego że cena synchronizacji dalej wzrasta, ale program dalej wykonuje się na 8 jądrach.