Politechnika Wrocławska

Wstęp do wysokowydajnych komputerów

Laboratorium nr. 6

Wykonawca:	
Imię i Nazwisko, nr indeksu, wydział	**Katarzyna Idasz** Wydział Informatyki i Telekomunikacji (W4N)
Termin zajęć: dzień tygodnia, godzina	Wtorek (11:15 – 13:00)
Numer grupy	1
Data wykonania ćwiczenia	11.06.2024 r.
Data oddania sprawozdania	17.06.2024 r.
Ocena końcowa	

Spis treści

Spis treści **1**
1. Wstęp **2**
1.1. Cel zadania 2
1.2. Splot 2
2. Kod w C **3**
2.1. Omówienie kodu 3
3. Pomiary **7**
3.1. Cykle 7
3.2. Czas 9
3.3. Perf 11
4. Wydajność **11**
4.1. Piksele/sekundę 11
4.2. GB/s 12
4.3. Piksele/cykl 12

1. Wstęp

1.1. Cel zadania

Należało zaimplementować operację splotu. Dodatkowo:

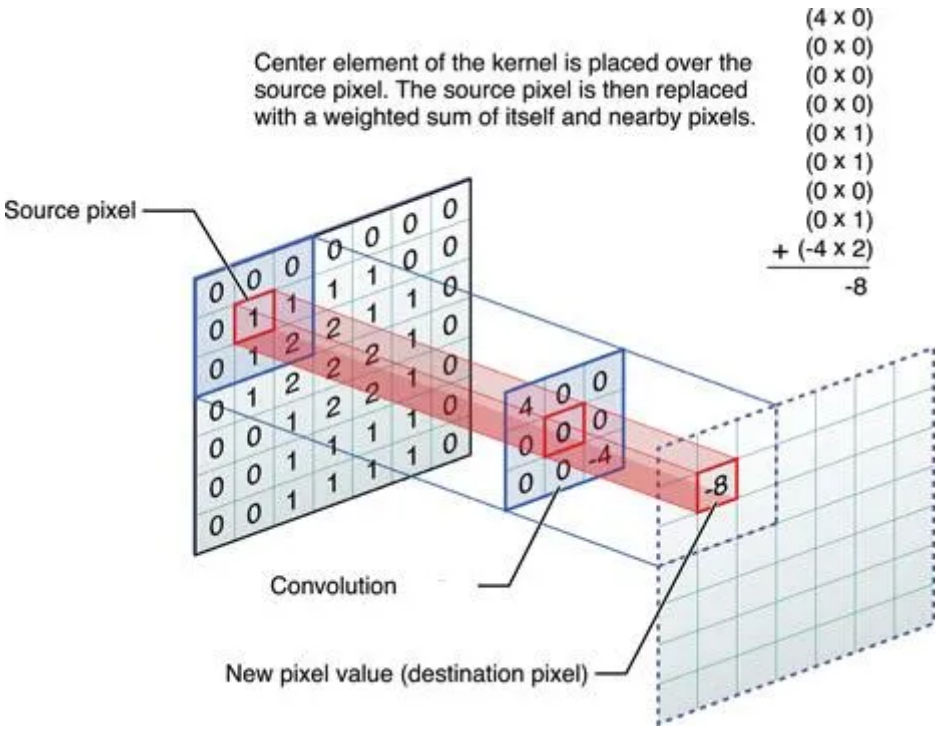
A. sprawdzić poprawność,
B. wykonać pomiary wydajności i profilowanie,
C. wprowadzić optymalizację i powtórzyć pomiary,
D. wydajność określić w GB/s, pikselach/sekundę, pikselach/cykl.

Podana macierz:

$$K = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

1.2. Splot

Aby obliczyć nową wartość piksela $p_{i,j}$ należy nałożyć macierz na obraz w taki sposób, aby środek macierzy znajdował się na tym pikselu. Następnie wartość pikseli zakrytych jest mnożona przez odpowiadającą wartość w macierzy i sumowana, aby otrzymać wartość piksela $p_{i,j}$. Operacja została zilustrowana na obrazie (1).



Obraz 1. Splot[1]

[1] https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Wartość piksela jest liczbą całkowitą z zakresu $[0,\ 255]$, gdzie 0 oznacza kolor czarny, a 255 biały. Ze względu na to, że po operacji splotu zakres nowej wartości wynosi $[-\ 3\ \cdot\ 255,\ 3\ \cdot\ 255]$, zastosowano skalowanie w postaci $p_{i,j} = (p_{i,j} + 3 \cdot 255) / 8$.

2. Kod w C

2.1. Omówienie kodu

Obraz wczytywany jest do tablicy M w kolejności *row-major order*, więc aby odwołać się do piksela $p_{i,j}$ należy skorzystać z wzoru $p_{i,j} = M[i \cdot width + j]$.

Ze względu na mały rozmiar podanej macierzy (3 x 3), funkcja `filter(unsigned char * M, unsigned char * W, int width, int height)` oblicza nową wartość piksela korzystając ze wzoru $p_{i,j} = (-1) * M[(i-1) * width + j - 1] + (-1) * M[(i-1) * width + j] + (-1) * M[i * width + j - 1] + M[i * width + j + 1] + M[(i+1) * width + j] + M[(i+1) * width + j + 1]$, a następnie wykonywane jest skalowanie. Wynik zapisywany jest w $W[i * width + j]$.

```
#include <png.h>
#include <stdio.h>
#include <stdlib.h>

#define ERROR                                                   \
    fprintf (stderr, "ERROR at %s:%d.\n", __FILE__, __LINE__) ;   \
    return -1 ;                                                 \

void
filter
(
    unsigned char * M, unsigned char * W,
    int width, int height
) ;


int main (int argc, char ** argv)
{
    if (2 != argc)
    {
       printf ("\nUsage:\n\n%s file_name.png\n\n", argv[0]) ;

       return 0 ;
    }

    const char * file_name = argv [1] ;
```

```
#define HEADER_SIZE (1)
unsigned char header [HEADER_SIZE] ;

FILE *fp = fopen (file_name, "rb");
if (NULL == fp)
{
   fprintf (stderr, "Can not open file \"%s\".\n", file_name) ;
   ERROR
}

if (fread (header, 1, HEADER_SIZE, fp) != HEADER_SIZE)
{
   ERROR
}

if (0 != png_sig_cmp (header, 0, HEADER_SIZE))
{
   ERROR
}

png_structp png_ptr =
   png_create_read_struct
         (PNG_LIBPNG_VER_STRING, NULL, NULL, NULL );
if (NULL == png_ptr)
{
   ERROR
}

png_infop info_ptr = png_create_info_struct (png_ptr);
if (NULL == info_ptr)
{
   png_destroy_read_struct (& png_ptr, NULL, NULL);

   ERROR
}

if (setjmp (png_jmpbuf (png_ptr)))
{
   png_destroy_read_struct (& png_ptr, & info_ptr, NULL);

   ERROR
}

png_init_io         (png_ptr, fp);
png_set_sig_bytes (png_ptr, HEADER_SIZE);
```

```
png_read_info       (png_ptr, info_ptr);

png_uint_32  width, height;
int  bit_depth, color_type;

png_get_IHDR
(
   png_ptr, info_ptr,
   & width, & height, & bit_depth, & color_type,
   NULL, NULL, NULL
);

if (8 != bit_depth)
{
   ERROR
}
if (0 != color_type)
{
   ERROR
}

size_t size = width ;
size *= height ;

unsigned char * M = malloc (size) ;

png_bytep ps [height] ;
ps [0] = M ;
for (unsigned i = 1 ; i < height ; i++)
{
   ps [i] = ps [i-1] + width ;
}
png_set_rows (png_ptr, info_ptr, ps);
png_read_image (png_ptr, ps) ;

printf
(
   "Image %s loaded:\n"
   "\twidth   = %u\n"
   "\theight  = %u\n"
   "\tbit_depth  = %u\n"
   "\tcolor_type = %u\n"
   , file_name, width, height, bit_depth, color_type
) ;

unsigned char * W = malloc (size) ;
```

```
    filter (M, W, width, height) ;

    png_structp write_png_ptr =
       png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (NULL == write_png_ptr)
    {
       ERROR
    }

    for (unsigned i = 0 ; i < height ; i++)
    {
       ps [i] += W - M ;
    }
    png_set_rows (write_png_ptr, info_ptr, ps);

    FILE *fwp = fopen ("out.png", "wb");
    if (NULL == fwp)
    {
       ERROR
    }

    png_init_io   (write_png_ptr, fwp);
    png_write_png (write_png_ptr, info_ptr, PNG_TRANSFORM_IDENTITY, NULL);
    fclose (fwp);

    return 0;
}

void filter(unsigned char * M, unsigned char * W, int width, int height) {

    for(int i = 1; i < height - 1; i++) {
       for(int j = 1; j < width - 1; j++) {
             int w =  0 + (-1)*M[(i-1)*width + j-1] + (-1)*M[(i-1)*width + j] +
                          (-1)*M[i*width + j-1] + M[i*width + j+1] +
                          M[(i+1)*width + j] + M[(i+1)*width + j+1];

             w = (w + 255*3)/8;
             W[i*width + j] = (char) w;
       }
    }
}
```

Kod 1. png.c

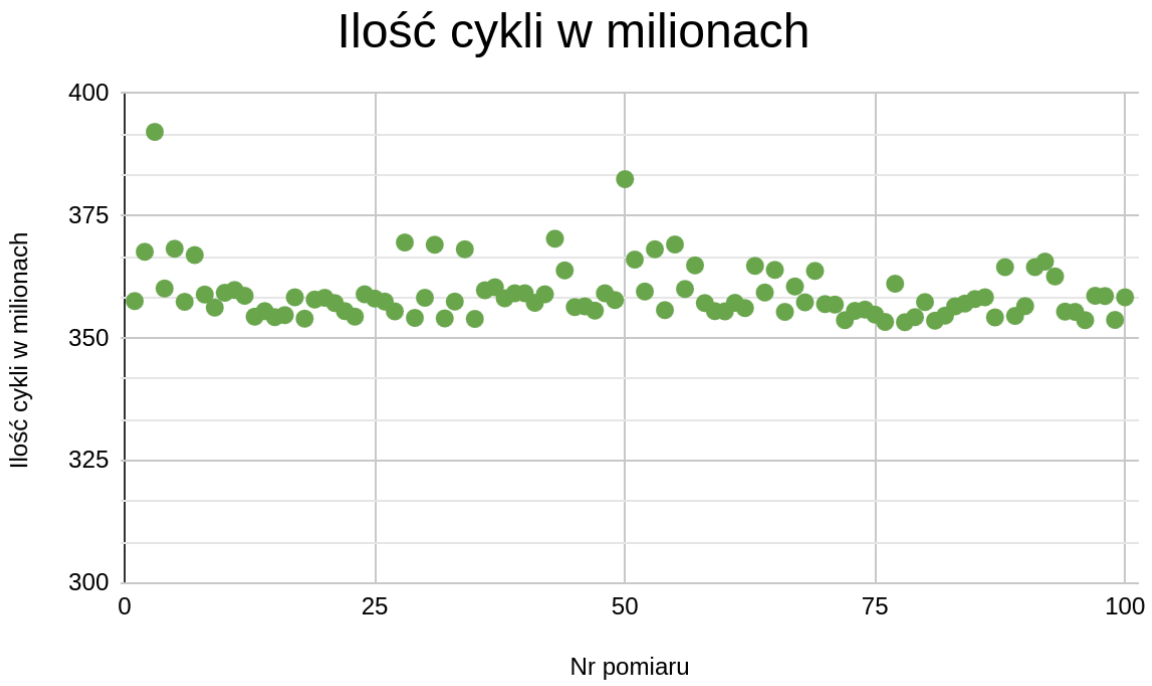
Sprawdzono poprawność zaimplementowanej operacji splotu korzystając z *compare p1.png, p2.png, diff.png*. Obraz wygenerowany był identyczny do obrazu wzorcowego.

3. Pomiary

Na laboratoriach wykonano pomiary dla obrazu 1.png o rozmiarze 800 x 640. Ze względu na mały rozmiar, rozrzut pomiarów był zbyt duży (zbyt mały rząd wielkości). Pomiary powtórzono dla obrazu tree2.png o rozmiarze 4896 x 3264 pikseli. Powtórzono je 100 razy. Wyniki umieszczono w tabelach oraz na wykresach.
Na szaro zaznaczono w tabelach (1) i (2) wartości odrzucone ze względu na błędy grube - pomiary odstające na wykresach (1) oraz (2).

3.1. Cykle

Zmierzono korzystając z *rdtsc()*.



Wykres 1. Ilość cykli

Tab 1. Ilość cykli przed optymalizacją

Nr pomiaru				Ilość cykli przed optymalizacją			
1	26	51	76	357605850	357513090	366075360	353363160
2	27	52	77	367669620	355538160	359545590	361154250
3	28	53	78	392188170	369557970	368199630	353333580
4	29	54	79	360194850	354195060	355747890	354330180
5	30	55	80	368300100	358272450	369153750	357415290
6	31	56	81	357447360	369095160	360077340	353600760
7	32	57	82	366990060	354092940	364923900	354651420
8	33	58	83	358962570	357532800	357232410	356555220
9	34	59	84	356281320	368170380	355559820	357093450
10	35	60	85	359293350	354001770	355509300	358036170
11	36	61	86	359868630	359802090	357279930	358386780
12	37	62	87	358672740	360439590	356208840	354270750
13	38	63	88	354423720	358167690	364808490	364566540
14	39	64	89	355553010	359204790	359384250	354580140
15	40	65	90	354313260	359236980	364003710	356591160
16	41	66	91	354744690	357261390	355414590	364548150
17	42	67	92	358402620	358983120	360585510	365691420
18	43	68	93	354035250	370343880	357347340	362633610
19	44	69	94	357950520	363867930	363764580	355469760
20	45	70	95	358293510	356378490	356992230	355416120
21	46	71	96	357188760	356542140	356880510	353699070
22	47	72	97	355556100	355687350	353740800	358684500
23	48	73	98	354417390	359208090	355607370	358641810
24	49	74	99	358998420	357833790	355852680	353793000
25	50	75	100	358131990	382517280	354853590	358402140

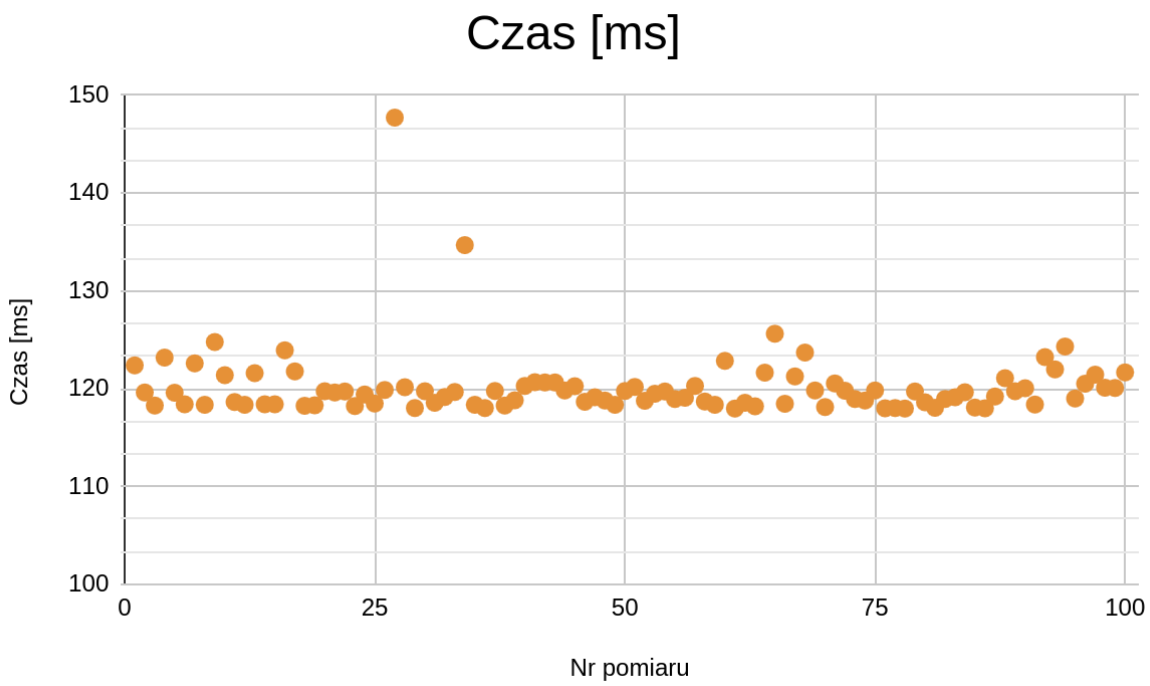
Po usunięciu odstających wartości wyniki pomiarów są zbliżone, dlatego podstawiono do wzorów na wydajność średnią = 358 713 067,7 cykli.

3.2. Czas

Zmierzono korzystając z *clock_t* z biblioteki <*time.h*>.

```
clock_t start = clock();
filter (M, W, width, height);
clock_t end = clock();
double miliseconds = (double)(end - start) / CLOCKS_PER_SEC * 1000;
printf("%lf\n", miliseconds);
```

Kod 2. Fragment odpowiadający za zmierzenie czasu wykonywania operacji splotu



Wykres 2. Czas [ms]

Tab 2. Czas w [ms]

Nr pomiaru				Czas [ms]			
1	26	51	76	122,393	119,872	120,176	118,017
2	27	52	77	119,633	147,722	118,782	118,039
3	28	53	78	118,295	120,165	119,493	117,979
4	29	54	79	123,184	118,042	119,72	119,726
5	30	55	80	119,585	119,748	118,965	118,618
6	31	56	81	118,427	118,565	119,079	118,061
7	32	57	82	122,611	119,156	120,287	118,966
8	33	58	83	118,367	119,666	118,702	119,141
9	34	59	84	124,783	134,697	118,353	119,64
10	35	60	85	121,405	118,366	122,868	118,075
11	36	61	86	118,648	118,028	117,971	118,011
12	37	62	87	118,353	119,767	118,581	119,209
13	38	63	88	121,607	118,277	118,219	121,094
14	39	64	89	118,408	118,821	121,654	119,746
15	40	65	90	118,405	120,298	125,626	120,071
16	41	66	91	123,936	120,683	118,478	118,399
17	42	67	92	121,777	120,647	121,27	123,231
18	43	68	93	118,252	120,659	123,711	121,976
19	44	69	94	118,326	119,82	119,825	124,317
20	45	70	95	119,749	120,253	118,136	118,995
21	46	71	96	119,635	118,684	120,549	120,519
22	47	72	97	119,73	119,125	119,806	121,45
23	48	73	98	118,237	118,809	118,944	120,119
24	49	74	99	119,417	118,365	118,791	120,096
25	50	75	100	118,491	119,778	119,822	121,664

Po usunięciu odstających wartości wyniki pomiarów są zbliżone, dlatego podstawiono do wzorów na wydajność średnią = 119,8 ms.

3.3. Perf

```
Performance counter stats for './program png_images/tree2.png':

        1 571,33 msec task-clock                       #    0,997 CPUs utilized
               5      context-switches                 #    3,182 /sec
               0      cpu-migrations                   #    0,000 /sec
           7 983      page-faults                      #    5,080 K/sec
   6 250 997 956      cycles                           #    3,978 GHz                      (83,13%)
     101 343 922      stalled-cycles-frontend          #    1,62% frontend cycles idle     (83,20%)
   2 036 756 611      stalled-cycles-backend           #   32,58% backend cycles idle      (83,45%)
   9 686 995 105      instructions                     #    1,55  insn per cycle
                                                  #    0,21  stalled cycles per insn     (83,45%)
   1 494 387 601      branches                         #  951,035 M/sec                    (83,45%)
      71 141 567      branch-misses                    #    4,76% of all branches          (83,31%)

     1,575638389 seconds time elapsed

     1,524677000 seconds user
     0,048021000 seconds sys
```

Obraz 2. perf stat

```
Samples: 5K of event 'cycles:P', Event count (approx.): 4404788896
Overhead  Command  Shared Object          Symbol
  10,47%  program  program                [.] filter
   8,59%  program  libpng16.so.16.37.0    [.] png_write_row
   5,30%  program  libz.so.1.2.11         [.] 0x0000000000002b13
   4,64%  program  libz.so.1.2.11         [.] 0x0000000000002b1a
   4,57%  program  libz.so.1.2.11         [.] 0x0000000000002b1d
   3,96%  program  libz.so.1.2.11         [.] 0x0000000000002b0a
   3,81%  program  libz.so.1.2.11         [.] 0x0000000000002b18
   3,68%  program  libz.so.1.2.11         [.] 0x0000000000002b16
   3,64%  program  libz.so.1.2.11         [.] 0x0000000000002b0e
   3,41%  program  libz.so.1.2.11         [.] 0x0000000000002b20
   3,32%  program  libz.so.1.2.11         [.] 0x0000000000002b08
   2,78%  program  libz.so.1.2.11         [.] 0x0000000000002b28
   2,66%  program  libz.so.1.2.11         [.] 0x0000000000002b2c
   0,87%  program  libz.so.1.2.11         [.] adler32_z
   0,86%  program  libz.so.1.2.11         [.] 0x000000000000002b7b
   0,71%  program  libz.so.1.2.11         [.] 0x0000000000002b40
   0,67%  program  libz.so.1.2.11         [.] 0x0000000000002b44
   0,63%  program  libz.so.1.2.11         [.] 0x0000000000002b75
   0,63%  program  libz.so.1.2.11         [.] 0x0000000000002b71
   0,55%  program  libz.so.1.2.11         [.] 0x0000000000002b91
   0,53%  program  libz.so.1.2.11         [.] 0x0000000000002bbf
   0,50%  program  libz.so.1.2.11         [.] 0x0000000000002b95
```

Obraz 3. perf report

Funkcję *filter* można byłoby zoptymalizować na przykład korzystając z równoległości.

4. Wydajność

4.1. Piksele/sekundę

$$N = width \cdot height = 4896 \cdot 3264 = 15\,980\,544$$

$$P_{p/s} = \frac{N}{t} = \frac{15\,980\,544}{119{,}8 \cdot 10^3} \approx 133,4\ pikseli/s$$

N – *ilość pikseli, każdy piksel zapisany na jednym bajcie*

t – *średni czas* $[s]$

4.2. GB/s

Ze względu na to, że każdy piksel zapisany jest na jednym bajcie, wystarczy wynik w pikselach/sekundę podzielić przez 1024^3, aby otrzymać GB/s.

$$P_{GB/s} = P_{p/s} \cdot \frac{1}{1024^3} = 133,4 \cdot \frac{1}{1024^3} \approx 0,00000012\ GB/s$$

4.3. Piksele/cykl

$$P_{p/c} = \frac{N}{C} = \frac{15\,980\,544}{358\,713\,067,7} \approx 0,04\ pikseli/cykl$$

$C - średnia\ ilość\ cykli$