Piotr Koronczok, 272955
WT-15

Wrocław, June 11, 2024

prowadzący: dr hab. inż. Tadeusz Tomczak

Laboratorium Wprowadzenia do Wysokowydajnych Komputerów

# Implementacja oraz badanie parametrów funkcji wykonującej operację splotu.

## Contents

# 1 Cele oraz zakres ćwiczenia

Cele ćwiczenia:

- Implementacja operacji splotu

$$M \circledast K = W$$

- Sprawdzenie poprawności
- Profilowanie dla skrajnych rozmiarów macierzy (obrazków)
- Pomiary wydajności w *GB/s*, *pikselach/sekundę*, *pikselach/cykl*
- Wprowadzenie optymalizacji i ponowne pomiary

# 2 Przebieg ćwiczenia

Zadanie bazuje na kodzie oraz zdjęciach udostępnionych poprzez *eportal* - archiwum *png.tar.bz2*. W jego skład wchodzą:

- program png.c
- folder zawierający wiele obrazków *png* w dwóch wersjach: podstawowej *<nazwa>.png* oraz przetworzonej *<nazwa>_ emboss.png*

Podstawą wykonania ćwiczenia był odpowiednia modyfikacja programu i wykonanie pomiarów, do których wykorzystana została użyta we wcześniejszych laboratoriach funkcja do odczytu licznika Time Stamp Counter (funkcja serializująca z pliku func.s).

# 3 Implementacja operacji splotu

Operacja splotu wykonywana jest przy użyciu macierzy $M$ - wejściowej, która jest podmacierzą reprezentacji obrazka oraz macierzy splotu - $K$.

Użyta macierz splotu:

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

Operacja polega na przemnożeniu wartości z macierzy $M$ przez odpowiadające im wartości z macierzy $K$ oraz zsumowaniu. Wynik zapisywany jest jako wartość piksela centralnego w macierzy wynikowej $W$. Opis implementowanej operacji jest na Wikipedii.

$$g(x,y) = \omega \circledast f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega(i,j) f(x-i, y-j)$$

Gdzie $g(x,y)$ to przefiltrowany obraz, $f(x,y)$ to oryginalny obraz, $\omega$ to jądro filtru. Każdy element jądra filtru jest rozważany w zakresie $-a \leq i \leq a$ oraz $-b \leq j \leq b$.

Uzyskanie poprawnych wartości typu *unsigned char* z zakresu $x \in < 0, 255 >$ wymagało przeskalowania otrzymanej sumy (w pierwotnej wersji $a = \frac{1}{6}$).

$$y = ax + b = \frac{1}{8}(x + 3 \cdot 255) = \frac{1}{8}(x + 765)$$

Listing 1: Funkcja odpowiedzialna za przetwarzanie obrazów

```
void filter (unsigned char * M, unsigned char * W, int width, int height)
{
    int size = width*height;

    char k[3][3] = {
        {-1, -1, 0},
        {-1, 0,  1},
        {0,  1,  1}
    };

    for (int i = 1; i < height - 1; i++)
    {
        for(int j = 1; j < width - 1; j++)
        {
            short int sum = 0;
            for (char x = 0; x < 3; x++)
            {
                for (char y = 0; y < 3; y++)
                {
                    sum += M[(i + x - 1)*width + j + y - 1] * k[x][y];
                }
            }
            unsigned char out = (unsigned char)((sum + 765) / 8);
            W[i*width + j] = out;
        }
    }
}
```

### 3.1 Użycie kompilatora gcc

- -o - zapewnienie pliku wyjściowego
- -S - wytworzenie pliku *.s*; jedynie kompilacja
- -O3 - ustawienie poziomu optymalizacji *gcc*
- -lpng - linkowanie razem w biblioteką *libpng*

Listing 2: Tworzenie pliku wykonywalnego wersji nieoptymalizowanej

```
gcc -o png png.c func.s -lpng
```

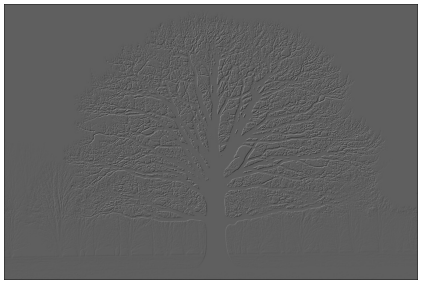
## 3.2 Sprawdzenie poprawności

Sprawdzenie poprawności wykonywanego kodu odbyło się poprzez wywołanie wywołanie programu dla różnych obrazków wejściowych oraz porównanie ich z referencyjnymi wynikami dostępnymi *eportalu*. Do tego celu wykorzystane zostało narzędzie *compare*, które pozwala na porównanie ze sobą plików *.png*.

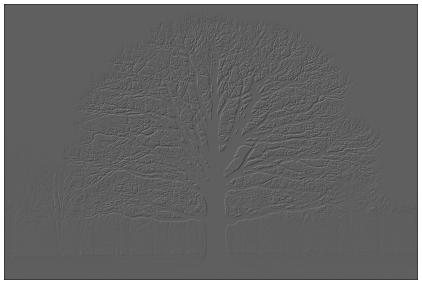
Listing 3: Wytworzenie pliku ukazującego różnicę pomiędzy obrazkiem wytworzonym a referencyjnym

```
compare out.png tree_emboss.png diff.png
```

(a) Obraz wejściowy - tree.png



(b) Wynik *png* - out.png



(c) Oczekiwany obraz wynikowy - tree_emboss.png



(d) Obraz ukazujący różnicę między out.png a tree_emboss.png - diff.png

Figure 1: Porównanie obrazów wejściowych i wynikowych oraz różnic między nimi.

Brak czerwonych pikseli w obrazie wynikowym 1d świadczy o poprawności działania programu.

### 3.3 Specyfikacja maszyny

Pomiary zostały wykonane na procesorze firmy *AMD* - AMD Ryzen™ 5 7640U. Pełna specyfikacja dostępna pod adresem strony producenta. System operacyjny to Ubuntu 22.04.4 LTS. Laptop posiada moduł pamięci DDR5 5600 SO-DIMM 16GB. Wykonanie *lscpu* pozwoliło na wyświetlenie najważniejszych informacji o maszynie.

```
(base) piotr@piotr-framework:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 5 7640U w/ Radeon 760M Graphics
    CPU family:          25
    Model:               116
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            1
    Frequency boost:     enabled
    CPU max MHz:         6466.7959
    CPU min MHz:         1600.0000
    BogoMIPS:            6987.03
```

Figure 2: Paramerty maszyny

### 3.4 Pomiary wydajności dla skrajnych rozmiarów macierzy

Macierzami wejściowymi były:

- *1.png* o rozmiarze

$$size = widht \cdot height = 800 \cdot 640 = 512000\ pixels$$

- *world_shaded_43k_mono.png* o rozmiarze

$$size = widht \cdot height = 43200 \cdot 21600 = 933120000\ pixels$$

Funkcja przetwarzająca obraz *1.png* wywołana została $n = 100$ razy w celu zwiększenia czasu wykonania i możliwości dokładniejszych pomiarów przy użyciu *perf*.

| odczyty rdtsc | czas [s] |
|---|---|
| 93855044 | 0.021331 |
| 87005439 | 0.019774 |
| 92268879 | 0.020970 |
| 99430334 | 0.022598 |
| 94007679 | 0.021365 |
| 89180759 | 0.020268 |
| 84897669 | 0.019295 |
| 98909954 | 0.022480 |
| 98844609 | 0.022465 |
| 92974304 | 0.021131 |
| 85168989 | 0.019357 |
| 85242699 | 0.019373 |
| 94229019 | 0.021416 |
| 87081599 | 0.019791 |
| $\overline{rdtsc}$ | $\overline{czas}$ |
| 91649784 | 0.020829 |

Table 1: Pomiary rdtsc dla funkcji filter wykonane przy dwukrotnym uruchomieniu programu dziesięciokrotnie dla częstotliwości taktowania 4.4 GHz dla najmniejszej macierzy.

Po weryfikacji wyników i odrzuceniu błędów grubych:

$$\overline{rdtsc} = 91649784$$

$$\overline{czas} = 0.020829s$$

Dla macierzy największej:

$$rdtsc = 2069176772$$

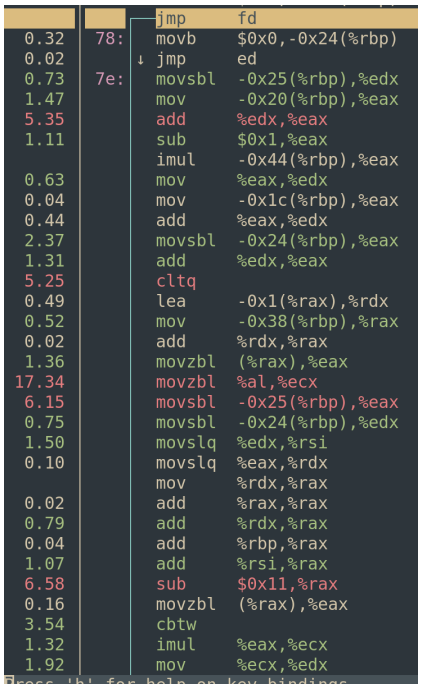
$$czas = 0.470s$$

```
Performance counter stats for 'taskset -c 1 ./png png_images/1.png':

        1,289.76 msec task-clock                       #    0.999 CPUs utilized
               6      context-switches                 #    4.652 /sec
               1      cpu-migrations                   #    0.775 /sec
             486      page-faults                      #  376.813 /sec
   5,676,771,655      cycles                           #    4.401 GHz                      (83.26%)
         768,775      stalled-cycles-frontend          #    0.01% frontend cycles idle     (83.25%)
      44,990,642      stalled-cycles-backend           #    0.79% backend cycles idle      (83.26%)
  20,409,093,580      instructions                     #    3.60  insn per cycle
                                                  #    0.00  stalled cycles per insn  (83.27%)
   1,107,282,144      branches                         #  858.516 M/sec                    (83.56%)
       2,348,463      branch-misses                    #    0.21% of all branches          (83.40%)

     1.290732379 seconds time elapsed

     1.290670000 seconds user
     0.000000000 seconds sys
```

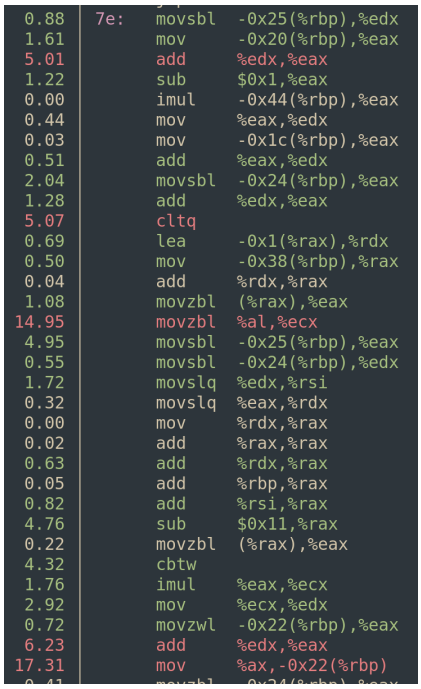
Figure 3: Wyniki perf stat dla najmniejszej macierzy z obrazu 1.png

```
(base) piotr@piotr-framework:~/Documents/Studia/Studia2023_24/semestr_4/WdWK/L/L12/lab6$ sudo perf stat -d -d -d -B taskset -c 1 ./png
tsc: 2069176772

Performance counter stats for 'taskset -c 1 ./png ../png_images/world_shaded_43k_mono.png':

       55,101.33 msec task-clock                       #    0.999 CPUs utilized
           1,744      context-switches                 #   31.651 /sec
               1      cpu-migrations                   #    0.018 /sec
         455,973      page-faults                      #    8.275 K/sec
 242,609,505,897      cycles                           #    4.403 GHz                      (33.32%)
     279,705,440      stalled-cycles-frontend          #    0.12% frontend cycles idle     (33.34%)
   1,017,509,751      stalled-cycles-backend           #    0.42% backend cycles idle      (33.36%)
 647,795,357,894      instructions                     #    2.67  insn per cycle
                                                  #    0.00  stalled cycles per insn  (33.37%)
  63,814,646,849      branches                         #    1.158 G/sec                    (33.38%)
   1,364,438,469      branch-misses                    #    2.14% of all branches          (33.36%)
 157,660,191,045      L1-dcache-loads                  #    2.861 G/sec                    (33.34%)
   4,409,586,260      L1-dcache-load-misses            #    2.80% of all L1-dcache accesses  (33.33%)
 <not supported>      LLC-loads
 <not supported>      LLC-load-misses
     375,235,990      L1-icache-loads                  #    6.810 M/sec                    (33.32%)
       4,725,644      L1-icache-load-misses            #    1.26% of all L1-icache accesses  (33.31%)
      12,955,837      dTLB-loads                       #  235.127 K/sec                    (33.32%)
       1,864,962      dTLB-load-misses                 #   14.39% of all dTLB cache accesses  (33.32%)
         231,384      iTLB-loads                       #    4.199 K/sec                    (33.31%)
       1,172,969      iTLB-load-misses                 #  506.94% of all iTLB cache accesses  (33.31%)
   1,502,655,585      L1-dcache-prefetches             #   27.271 M/sec                    (33.31%)
 <not supported>      L1-dcache-prefetch-misses

    55.183464725 seconds time elapsed

    54.460296000 seconds user
     0.640003000 seconds sys
```

Figure 4: Wynik perf stat dla największej macierzy z obrazu world_shaded_43k_mono.png




(a) Wnętrze pętli for w kodzie assemblera dla najmniejszej macierzy.




(b) Wnętrze pętli for w kodzie assemblera dla największej macierzy.

Figure 5: Porównanie wnętrza zagnieżdżonych pętli for w kodzie assemblera dla najmniejszej i największej macierzy. Fragment odpowiada fragmentowi *sum* += ... z funkcji *filter*1

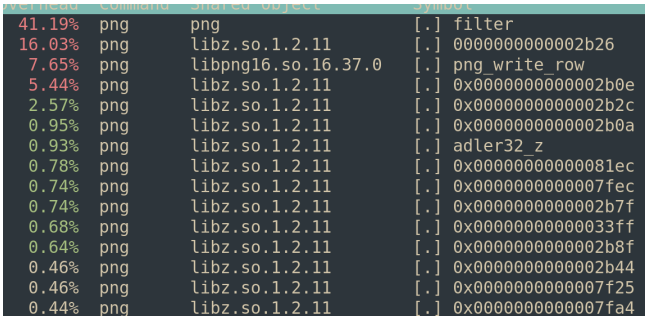



Figure 6: Pierwszy ekran perf dla największej macierzy.

Profilowanie z użyciem narzędzia *perf* wskazuje, iż najgorątszymi instrukcjami są te związane z przesyłem danych. Kluczową różnicą między wynikami *perf* dla najmniejszej oraz największej macierzy jest znaczna liczba próbek w funkcjach bibliotecznych *libz* oraz *libpng* w przypadku dużego obrazu.

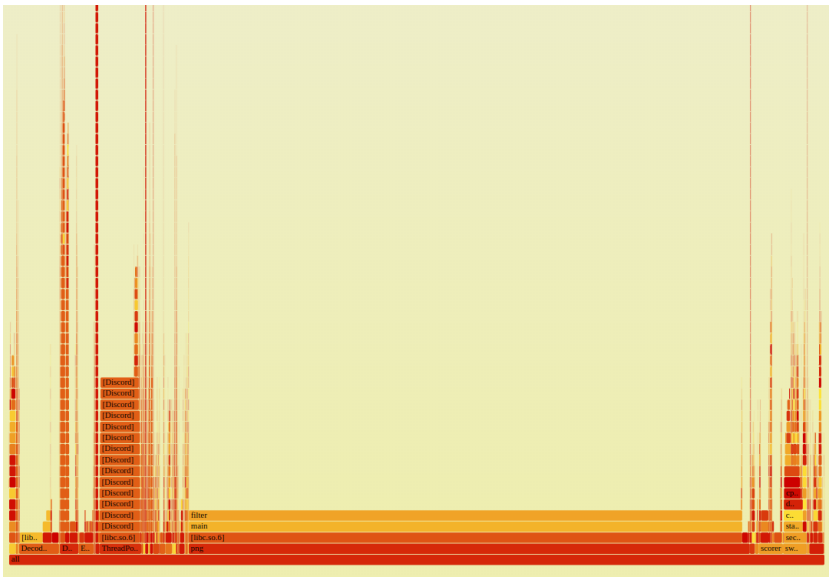



Figure 7: Flamegraph dla wywołania programu z macierzą najmniejszą jako wejście.

W ramach operacji splotu na pojedynczym pikselu wykonywana jest seria zapisów i odczytów z pamięci:

$$mem_{op}\frac{B}{iter} = 29\frac{B}{iter}$$

Dla macierzy najmniejszej:

$$\frac{pikseli}{sek} = \frac{512000 \cdot 100}{\overline{czas}} = 2.5 \cdot 10^9 \frac{pikseli}{sek}$$

$$\frac{pikseli}{cykl} = \frac{512000 \cdot 100}{\overline{rdtsc}} = 0.559\frac{piksela}{cykl}$$

$$throughput = \frac{29 \cdot 512000 \cdot 100}{\overline{czas}} = \frac{29 \cdot 512000 \cdot 100}{0.020829s} = 71\frac{GB}{s}$$

Dla macierzy największej 4:

$$\frac{pikseli}{sek} = \frac{933120000}{0.114328367} = 8.1 \cdot 10^9 \frac{pikseli}{sek}$$

$$\frac{pikseli}{cykl} = \frac{933120000}{503044816} = 1.85\frac{piksela}{cykl}$$

$$throughput = \frac{29 \cdot 933120000}{czas} = \frac{27060480000}{0.47s} = 57\frac{GB}{s}$$

### 3.5 Wprowadzenie optymalizacji

Podstawą optymalizacji były zwiększenie jej poziomu przy użyciu kompilatora *gcc* - użycie *-O3*. Pozwoliło na wytworzenie kodu zawierającego instrukcje wektorowe *SIMD* używające rejestrów *%xmm*

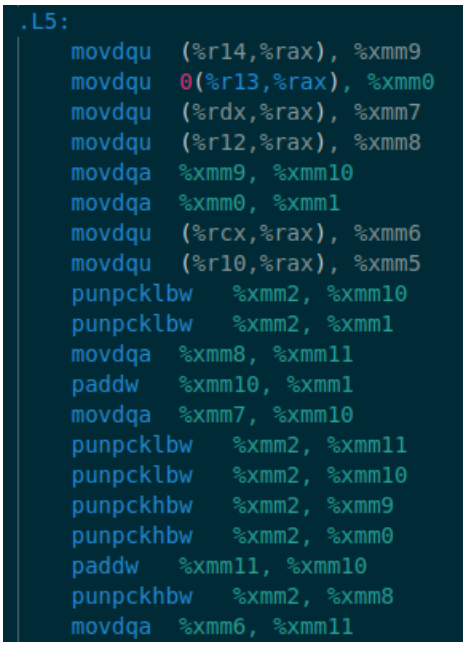



Figure 8: Fragment kodu wytworzonego przy użyciu flagi -O3

| odczyty rdtsc | czas [s] |
|---|---|
| 26602975 | 0.006047 |
| 25889045 | 0.005884 |
| 25324040 | 0.005755 |
| 27582240 | 0.006269 |
| 25692100 | 0.005839 |
| 25569320 | 0.005811 |
| 25565890 | 0.005811 |
| 25235070 | 0.005735 |
| 25701480 | 0.005843 |
| 25255930 | 0.005740 |
| 25589235 | 0.005816 |
| 25713065 | 0.005848 |
| 26874085 | 0.006108 |
| 26329905 | 0.005984 |
| $rdtsc_{avg}$ | $czas_{avg}$ |
| 25655583 | 0.005831 |

Table 2: Pomiary rdtsc dla funkcji filter (wywołanej 100-krotnie) wykonane przy dwukrotnym uruchomieniu programu dziesięciokrotnie dla częstotliwości taktowania 4.4 GHz dla najmniejszej macierzy. Odrzucono błędy grube.

Dla macierzy najmniejszej:

$$\frac{pikseli}{sek} = \frac{512000 \cdot 100}{0.005831} = 8.7 \cdot 10^9 \frac{pikseli}{sek}$$
$$\frac{pikseli}{cykl} = \frac{512000 \cdot 100}{25655583} = 2.0 \frac{piksela}{cykl}$$

Przyspieszenie:

$$S = \frac{0.020829}{0.005831} = 3.57$$

Dla macierzy największej:4:

$$rdtsc = 2563988732$$

$$czas = 0.58s$$

$$\frac{pikseli}{sek} = \frac{933120000}{0.58} = 1.6 \cdot 10^9 \frac{pikseli}{sek}$$
$$\frac{pikseli}{cykl} = \frac{933120000}{2563988732} = 0.36 \frac{piksela}{cykl}$$

Przyspieszenie:

$$S = \frac{0.470}{0.58} = 0.81$$

# 4 Podsumowanie

Wprowadzenie optymalizacji kompilatora przy przetwarzaniu mniejszej macierzy znacznie poprawiło czas wykonania programu [przyspieszenie 3.5], lecz w drugim badanym przypadku dla największego możliwego obrazu czas wykonania funkcji filter wydłużył się. Przyczyną tego stanu jest rozmiar przetwarzanych danych - 181MB, co dopowiada 933120000 pikselom.