# Raport: Porównanie wydajności kerneli w generowaniu zbioru Mandelbrota

Heorhii Lopatin (hl456366)

6 kwietnia 2023

## 1 Wstęp

Zbiór Mandelbrota to zbiór punktów zespolonych w płaszczyźnie, które poddane iteracjom pewnej funkcji nie wykazują rozbieżności. Jest to fascynujący obiekt matematyczny o skomplikowanej strukturze.

W ramach badań nad tym zbiorem, skupiamy się na analizie wydajności obliczenia tego zbioru w CUDA.

## 2 Opis implementacji

Zaimplementowano dwa rodzaje kerneli - jednowymiarowy oraz dwówymiarowy. Ponadto, przetestowano różne konfiguracje kernela.

### Mandelbrot.cu

```
__global__ void computeMandelbrot(real X0, real Y0, real X1, real Y1, int POZ,
                                  int PION, int ITER, int* Mandel) {
  double dX = (X1 - X0) / (POZ - 1);
  double dY = (Y1 - Y0) / (PION - 1);
  double x, y, Zx, Zy, tZx;
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int SIZE = POZ * PION;
  int pion = i % POZ;
  int poz = i / POZ;
  int iter = 0;

  Zx = dX * poz + X0;
  Zy = dY * pion + Y0;
  x = 0;
  y = 0;
  while (x * x + y * y < 4 && iter < ITER) {
    double a = x * x - y * y + Zx;
    double b = x * y * 2 + Zy;
    x = a;
    y = b;
    iter++;
  }
  Mandel[pion * POZ + poz] = iter;
}

```

```
__global__ void computeMandelbrot2D(real X0, real Y0, real X1, real Y1, int POZ,
                                    int PION, int ITER, int* Mandel) {
```

```
  double dX = (X1 - X0) / (POZ - 1);
  double dY = (Y1 - Y0) / (PION - 1);
  double x, y, Zx, Zy, tZx;
  int SIZE = POZ * PION;

  int pion = blockIdx.x * blockDim.x + threadIdx.x;
  int poz = blockIdx.y * blockDim.y + threadIdx.y;
  int iter = 0;

  Zx = dX * poz + X0;
  Zy = dY * pion + Y0;
  x = 0;
  y = 0;
  if (pion * POZ + poz < POZ * PION) {
    while (x * x + y * y < 4 && iter < ITER) {
      double a = x * x - y * y + Zx;
      double b = x * y * 2 + Zy;
      x = a;
      y = b;
      iter++;
    }
    Mandel[pion * POZ + poz] = iter;
  }
}
```

# 3 Wyniki

Wyniki przeprowadzonego eksperymentu przedstawiono w tabelach:

## 3.1 Tabela 1. Podsumowanie wyników - wersja 1D

| Liczba wątków w bloku | Typowy czas wykonania (ms) | Minimalny czas wykonania (ms) | Średni czas wykonania (ms) | Przyspieszenie względem wersji referencyjnej |
|---|---|---|---|---|
| CPU | 68795.31 | 68761.44 | 68849.40 ± 35.89 | 1.00 |
| 32 | 173.29 | 167.86 | 179.94 ± 4.34 | 382.63 ± 9.67 |
| 64 | 137.89 | 128.55 | 137.30 ± 0.49 | 501.47 ± 2.06 |
| 128 | 131.88 | 129.86 | 132.12 ± 0.32 | 521.10 ± 1.53 |
| 256 | 131.35 | 130.35 | 131.50 ± 0.18 | 523.55 ± 1.01 |
| 512 | 132.48 | 132.40 | 132.51 ± 0.01 | 519.58 ± 0.32 |
| 1024 | 135.10 | 135.62 | 135.27 ± 0.11 | 508.98 ± 0.67 |

## 3.2 Tabela 2. Podsumowanie wyników - wersja 2D 256 wątków

| Liczba wątków w bloku | | Typowy czas wykonania (ms) | Minimalny czas wykonania (ms) | Średni czas wykonania (ms) | Przyspieszenie względem wersji referencyjnej |
|---|---|---|---|---|---|
| X | Y | | | | |
| CPU | | 68795.31 | 68761.44 | 68849.40 ± 35.89 | 1.00 |
| 256 | 1 | 131.77 | 129.49 | 132.01 ± 0.21 | 521.55 ± 1.12 |
| 128 | 2 | 126.21 | 126.22 | 126.38 ± 0.11 | 544.78 ± 0.77 |
| 64 | 4 | 121.93 | 121.95 | 121.92 ± 0.01 | 564.72 ± 0.32 |
| 32 | 8 | 119.09 | 117.06 | 118.27 ± 0.22 | 582.16 ± 1.39 |
| 16 | 16 | 114.01 | 113.98 | 114.00 ± 0.01 | 603.96 ± 0.33 |
| 8 | 32 | 112.55 | 112.55 | 112.55 ± 0.01 | 611.75 ± 0.33 |
| 4 | 64 | 112.05 | 112.03 | 112.03 ± 0.01 | 614.56 ± 0.34 |
| 2 | 128 | 112.26 | 112.25 | 112.28 ± 0.02 | 613.21 ± 0.45 |
| 1 | 256 | 113.23 | 113.21 | 113.21 ± 0.01 | 608.13 ± 0.33 |

### 3.3 Tabela 3. Podsumowanie wyników - wersja 2D 1024 wątki

| Liczba wątków w bloku | | Typowy czas wykonania (ms) | Minimalny czas wykonania (ms) | Średni czas wykonania (ms) | Przyspieszenie względem wersji referencyjnej |
|---|---|---|---|---|---|
| X | Y | | | | |
| CPU | | 68795.31 | 68761.44 | 68849.40 ± 35.89 | 1.00 |
| 1024 | 1 | 137.62 | 135.25 | 136.63 ± 0.22 | 503.90 ± 1.07 |
| 512 | 2 | 129.73 | 129.78 | 129.77 ± 0.01 | 530.55 ± 0.31 |
| 256 | 4 | 125.21 | 125.21 | 125.24 ± 0.01 | 549.76 ± 0.31 |
| 128 | 8 | 120.93 | 121.25 | 121.29 ± 0.20 | 567.64 ± 1.22 |
| 64 | 16 | 120.19 | 118.46 | 119.78 ± 0.18 | 574.82 ± 1.16 |
| 32 | 32 | 118.82 | 118.74 | 118.79 ± 0.01 | 579.60 ± 0.36 |
| 16 | 64 | 114.37 | 114.34 | 114.55 ± 0.12 | 601.05 ± 0.94 |
| 8 | 128 | 113.59 | 113.51 | 113.45 ± 0.01 | 606.89 ± 0.38 |
| 4 | 256 | 114.14 | 114.04 | 114.10 ± 0.01 | 603.41 ± 0.37 |
| 2 | 512 | 115.91 | 115.83 | 115.87 ± 0.01 | 594.22 ± 0.35 |
| 1 | 1024 | 120.02 | 118.74 | 119.20 ± 0.22 | 577.62 ± 1.35 |

### 3.4 Tabela 4. Podsumowanie wyników - wersja 2D: inne przypadki

| Liczba wątków w bloku | | Typowy czas wykonania (ms) | Minimalny czas wykonania (ms) | Średni czas wykonania (ms) | Przyspieszenie względem wersji referencyjnej |
|---|---|---|---|---|---|
| X | Y | | | | |
| CPU | | 68795.31 | 68761.44 | 68849.40 ± 35.89 | 1.00 |
| 32 | 32 | 119.62 | 115.10 | 119.27 ± 0.24 | 577.25 ± 1.48 |
| 16 | 16 | 114.16 | 114.16 | 114.17 ± 0.01 | 603.07 ± 0.32 |
| 8 | 8 | 116.61 | 116.60 | 116.61 ± 0.01 | 590.43 ± 0.32 |
| 32 | 16 | 118.20 | 118.20 | 118.22 ± 0.01 | 582.39 ± 0.33 |
| 64 | 8 | 119.37 | 119.34 | 119.36 ± 0.01 | 576.82 ± 0.34 |
| 8 | 64 | 113.42 | 113.37 | 113.42 ± 0.01 | 607.01 ± 0.36 |
| 16 | 32 | 114.97 | 114.95 | 114.96 ± 0.01 | 598.89 ± 0.33 |

## 4 Podsumowanie

Najpierw spójrzmy na każdą tabelę z osobna.

W pierwszej tabeli widzimy, że największa prędkość jest osiągana, gdy liczba bloków jest pomiędzy 128 a 256. Powodem tego jest to, że każdy blok ma ograniczone zasoby więc liczba wątków musi być ograniczona, ale jednocześnie nie chcemy tworzyć zbyt wielu bloków, ponieważ ich tworzenie oraz dostęp zajmuje czas.

W drugiej tabeli widzimy, że najbardziej optymalnym wyborem wymiarów dla bloku jest 4x64. Nie jestem pewien dlaczego, ale gdybym miał zgadywać, taki wybór wymiarów może zapewnić najszybszy dostęp do każdego wątku. Również interesującym faktem jest to, że jeśli A < B to układ AxB jest znacznie szybszy niż BxA.

Z trzeciej tabeli wynika, że 1024 wątków to za dużo dla pojedynczego bloku.

Generalnie można powiedzieć, że wybierając rozmiar bloku musimy mieć pewność, że on efektywnie wykorzystuje swoje zasoby. Także dowiedzieliśmy się, że wykorzystywanie wieloprocesorowości jest znacznie bardziej efektywne w przypadku obliczeń niezależnych.