PRiW Zadanie 4

Kacper Małecki

November 22, 2025

Abstract

Sprawozdania, które omawia użycie OpenMP, wyniki czasowe oraz sposoby podziału zadania dla wątku przy wielowątkowym obliczaniu fraktala Mandelbrota.

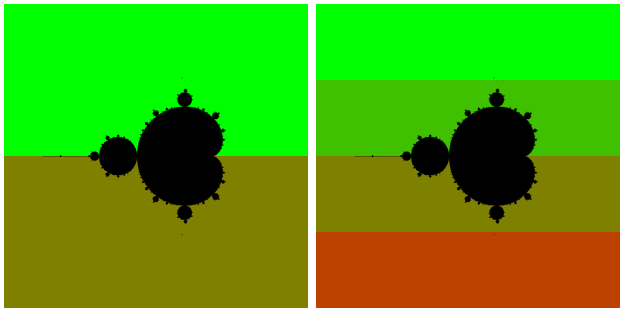
Specyfikacje procesora:
Nazwa: Ryzen 7 7800X3D (Desktop)
Ilość procesorów: 8
Ilość wątków: 16
Base Clock: 4.2 GHz
Boost Clock: 5.0 GHz
L1 Cache: 512 KB
L2 Cache: 8 MB
L3 Cache: 96 MB
Default TDP: 120W

1 Sposoby podziału zadań

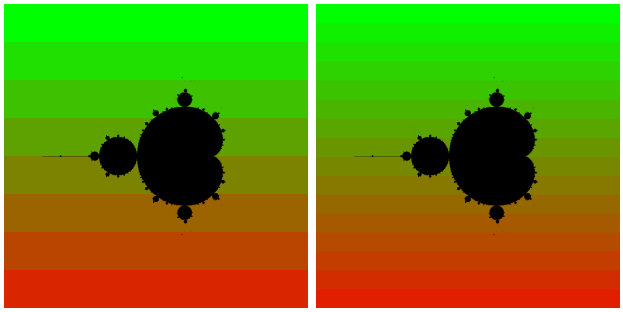
Przy wykonaniu zadania posłużyłem się 3 metodami (harmonogramami) podziału zadań dla wątków.

1.1 Static

Harmonogram static, który dzieli zadanie na prawie równe bloki zgodnie z ilością, którą mu podamy; gdy tego nie zrobimy każdy wątek dostanie jedną spójną część bloku. Harmonogram ten zachowuje się tak samo jak pierwsze nasze podejście z poprzedniego labolatorium. W zależności od podanej wielkości bloku (w tym przypadku 1) może zachowywać się też jak nasze drugie podejście, gdzie przydzielaliśmy wiersz co przyjętą ilość wątków.



(a) 2 wątki (b) 4 wątki



(c) 8 wątków (d) 16 wątków

Figure 1: Harmonagramowanie static; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

Zalety:

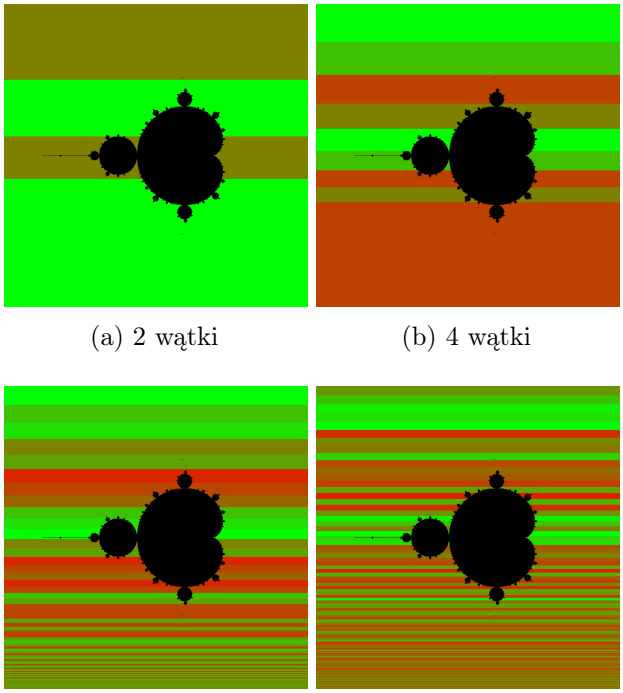
- Mały narzut (przydział raz).
- Dobra lokalność pamięci (ciągłe zakresy).

Wady:

- Słabe wyrównanie obciążenia, jeśli iteracje mają różny koszt (przy założeniu domyślnego rozmiaru bloku).

1.2 Dynamic

Iteracje są podzielone na kawałki o podanej wielkości. domyślnie jest to 1. Każdy wątek pobiera kolejny dostępny kawałek „na żądanie” (runtime utrzymuje licznik kolejnego bloku i atomowo go inkrementuje). Gdy wątek skończy swój kawałek, bierze następny. Jest to odpowiednik naszego podejścia z mutexem.



(a) 2 wątki (b) 4 wątki

(c) 8 wątków (d) 16 wątków

Figure 2: Harmonagramowanie Dynamic; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

Zalety:

- Świetne wyrównanie obciążenia przy dużych różnicach w czasie iteracji.
- Większy narzut (atomowe/licznikowe operacje, większe przełączanie).

Wady:

- Gorsza lokalność pamięci (wątki mogą wykonywać odległe zakresy).

1.3 Guided

Harmonagramowanie Guided to podejście hybrydowe pomiędzy static a dynamic; gdy wątek zarząda kolejnego bloku środowisko wykonawcze przydzieli mu blok o następującym rozmiarze:

$$\text{Blok} = \max\left(\left\lceil \frac{R}{T} \right\rceil, C_{\min}\right)$$

gdzie:

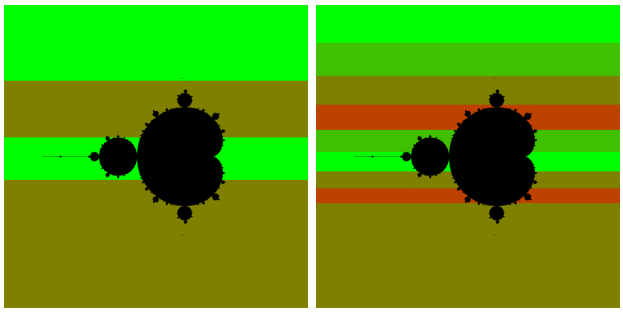
R - całkowita liczba pozostałych rekordów

T - liczba podziałów

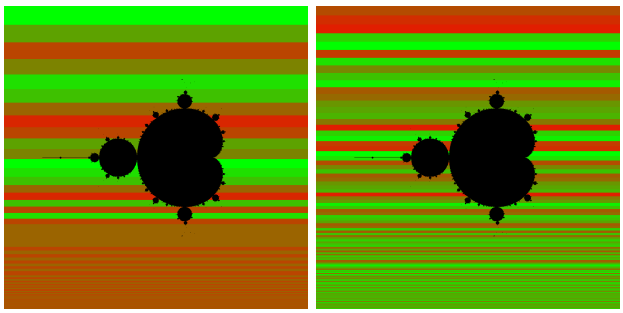
$C_{\min}$ - minimalny rozmiar bloku

Blok - wynikowy rozmiar bloku

W rezultacie przydzielane bloki stają się coraz mniejsze.



(a) 2 wątki (b) 4 wątki



(c) 8 wątków (d) 16 wątków

Figure 3: Harmonagramowanie Guided; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

Zalety:

- Kompromis między niskim overhead (duże początkowe bloki) a dobrym wyrównaniem (małe końcowe bloki).

Wady:

- Implementacje i zachowanie zależą nieco od runtime'u.

2 Struktura kodu

Struktura wszystkich programów wygląda w ten sposób:

```
    // Deklaracja zmiennych
    double Cx, Cy;
    double PixelWidth = (CxMax - CxMin) / iXmax;
    double PixelHeight = (CyMax - CyMin) / iYmax;
    double Zx, Zy, Zx2, Zy2;
    double ER2 = EscapeRadius * EscapeRadius;

    /**
        Rozpoczecie pracy watku, oraz wskazanie jakich zmiennych
           powinien uzyc "lokalnie", aby uniknac wspoldzielenia pamieci
           z innymi watkami
    **/
#pragma omp parallel private(Cx, Cy, Zx, Zy, Zx2, Zy2)
    {
        int tid = omp_get_thread_num();

        long int localSum = 0;

        unsigned char threadColor[3];
        threadColor[0] = (255 / nr_threads) * tid;
        threadColor[1] = 255 - threadColor[0];
        threadColor[2] = 0;

    // Wybor harmonogramowania oraz rozmiaru bloku
#pragma omp for schedule(HARMONOGRAM, ROZMIAR BLOKU) nowait //Nie
   synchronizujemy w tk w po wykonaniu p tli
        for (int iY = 0; iY < iYmax; ++iY) {
            ...
        }
        ...
    }
```

Listing 1: Generalana struktura

Różnica między tymi trzema implementacjami w kodzie, polega tylko i wyłącznie na zmianie HARMONOGRAM na static, dynamic, bądź guided.

3 Wyniki czasowe

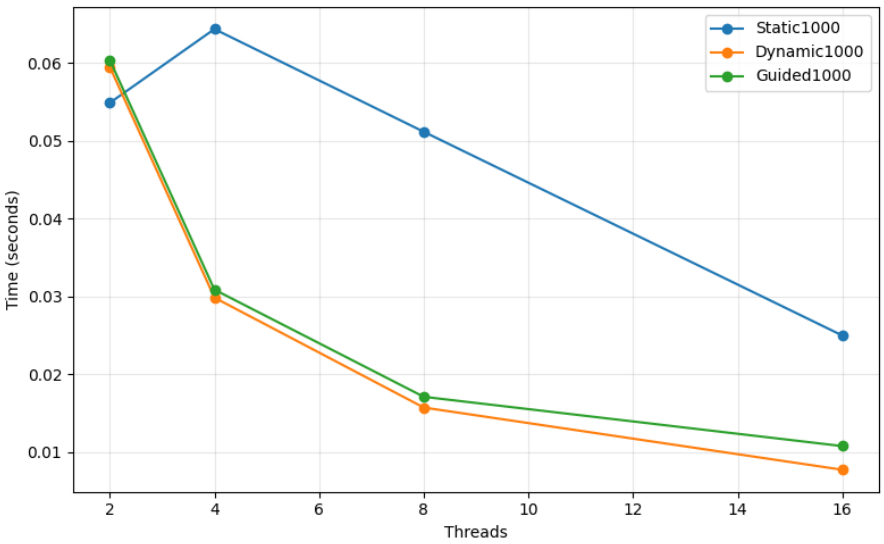
3.1 Jak liczba wątków wpływa na czas wykonaia

Na wykresach przedstawiono czasy obliczeń fraktala Mandelbrota dla trzech rozmiarów obrazu: 1000×1000, 5000×5000 oraz 10000×10000, z użyciem trzech harmonogramów OpenMP: `static`, `dynamic` oraz `guided`. Dla wszystkich testów analizowano wpływ liczby wątków (2, 4, 8, 16) na czas wykonania. Wykresy na następnej stronie.

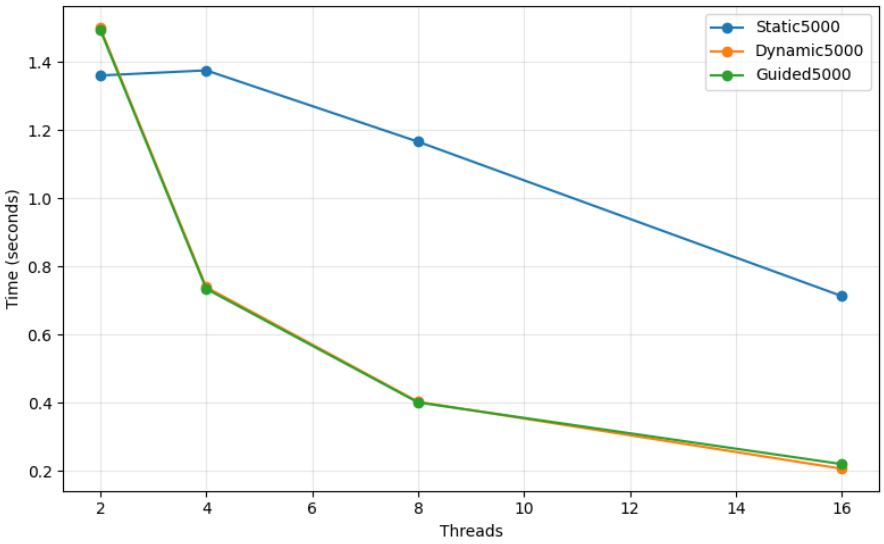
Zwiększanie liczby wątków powoduje wyraźne skracanie czasu obliczeń dla wszystkich harmonogramów. Największa poprawa widoczna jest przy przejściu z 2 do 4 wątków. Przy tym zakresie narzut synchronizacji jest niewielki, a dodatkowe wątki są w pełni wykorzystane. Dalsze zwiększanie liczby wątków — do 8 oraz 16 — również zmniejsza czas wykonania, choć przyrost szybkości staje się stopniowo mniejszy.

Harmonogram `static` skaluje się najsłabiej. Różnice w liczbie iteracji pomiędzy poszczególnymi obszarami obrazu powodują nierówne obciążenie wątków, przez co część z nich kończy pracę wcześniej i pozostaje bezczynna.

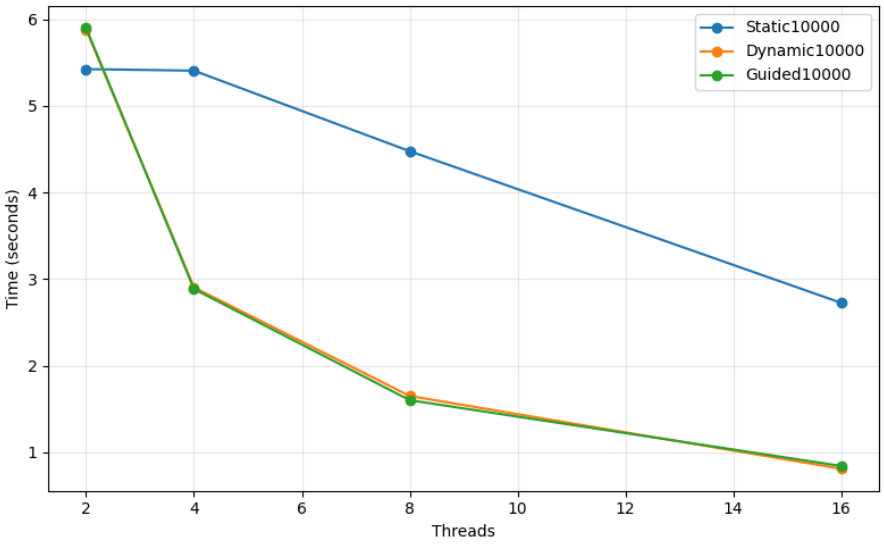
Harmonogramy `dynamic` oraz `guided` lepiej wykorzystują większą liczbę wątków. Przy 8 i 16 wątkach oba z nich zapewniają wyraźne skrócenie czasu, ponieważ zadania są pobierane na bieżąco i obciążenie pozostaje wyrównane nawet przy dużej liczbie iteracji.



(a) 1000×1000



(b) 5000×5000



(c) 10000×10000

Figure 4: Wyniki czasowe w zależności od razmiaru oraz ilości wątków

3.2 Wpływ doboru rozmiaru bloku

Jednak, gdy postanowimy dynamicznie dobierać rozmiary bloków okazuję się, że najlepsze rozłożenie problemu dostajemy od harmonogramu static i rozmiaru bloku 1. Podejście te ma duży overhead, jednak dostajemy prawie idealny podział bloków, co zdecydowanie przyspiesza program. Dynamic i Guided pokrywają się na wykresie. W przypadku dynamic, gdy zwiększamy rozmair bloku środowisko uruchomieniowe ma mniej pola do popisu przy dynamicznym rozdzielaniu .

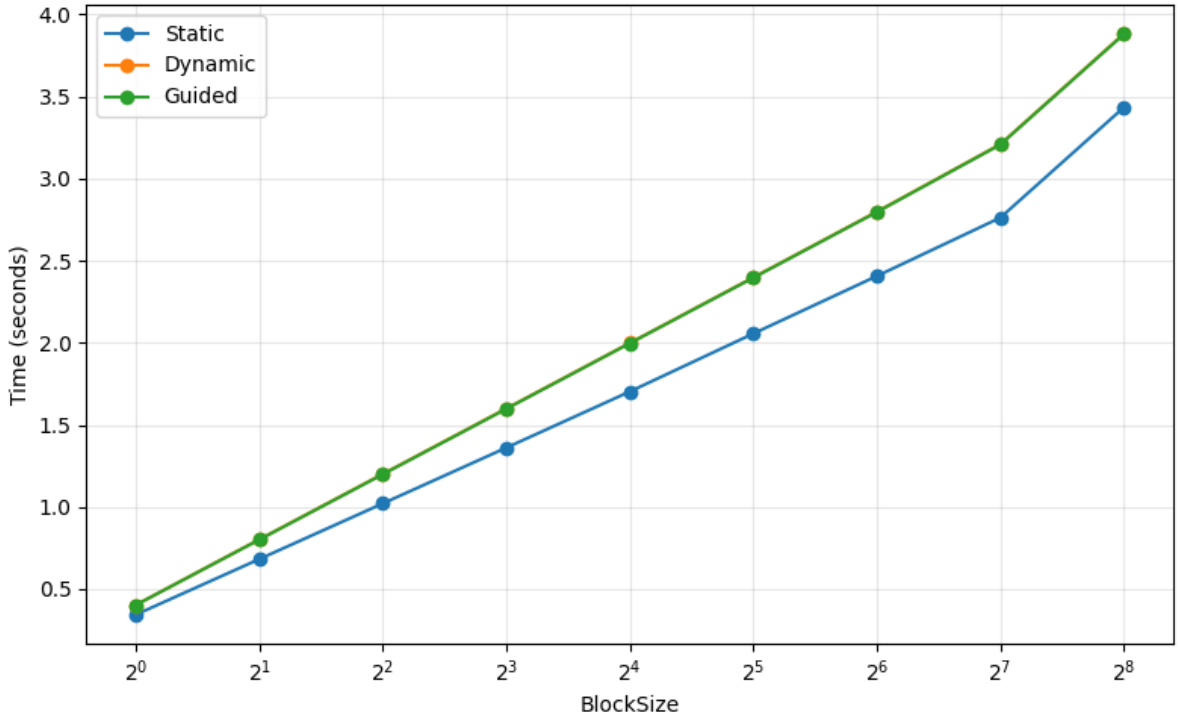


Figure 5: Wyniki czasowe w zależności od razmiaru bloku

3.3 Analiza wyniku z konsoli

Wynik z konsoli po uruchomieniu programu:

```
Size 1
Guided: 1.59909 s
Thread 0 iterations executed: 584298201, execution time: 1.26596
Thread 1 iterations executed: 627917397, execution time: 1.38379
Thread 2 iterations executed: 603838298, execution time: 1.24668
Thread 3 iterations executed: 667332714, execution time: 1.50119
Thread 4 iterations executed: 691528148, execution time: 1.5564
Thread 5 iterations executed: 711962679, execution time: 1.59857
Thread 6 iterations executed: 577849441, execution time: 1.2467
Thread 7 iterations executed: 586498105, execution time: 1.28396

Size 1
Static: 1.35927 s
Thread 0 iterations executed: 631928595, execution time: 1.35926
Thread 1 iterations executed: 631334006, execution time: 1.35056
Thread 2 iterations executed: 631335406, execution time: 1.35662
Thread 3 iterations executed: 631301965, execution time: 1.34826
Thread 4 iterations executed: 631353325, execution time: 1.3506
Thread 5 iterations executed: 631301581, execution time: 1.34994
Thread 6 iterations executed: 631335410, execution time: 1.35085
Thread 7 iterations executed: 631334695, execution time: 1.35058

Size 1
Dynamic: 1.59631 s
Thread 0 iterations executed: 583139419, execution time: 1.26261
Thread 1 iterations executed: 668250326, execution time: 1.49149
Thread 2 iterations executed: 691769318, execution time: 1.55271
Thread 3 iterations executed: 586498105, execution time: 1.28034
Thread 4 iterations executed: 578539745, execution time: 1.23971
Thread 5 iterations executed: 603147994, execution time: 1.2397
Thread 6 iterations executed: 711962679, execution time: 1.59629
Thread 7 iterations executed: 627917397, execution time: 1.3826
```

Listing 2: Czasy wykonania oraz ilość wykonanych iteracji dla każdego wątku

Patrząc na ilość wykonanych iteracji dla każdego wątku oraz czasy wykonania można stwierdzić, że najlepsze rozłożenie pracy zapewnia Harmonogram static, gdy dobierzemy rozmiar bloku 1.