PRiW Zadanie 4

Kacper Małecki

November 17, 2025

Abstract

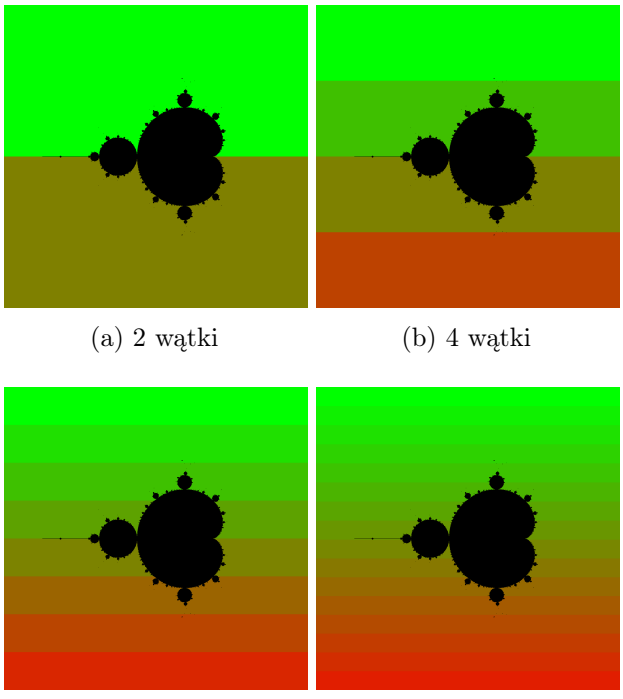
Sprawozdania, które omawia użycie OpenMP, wyniki czasowe oraz sposoby podziału zadania dla wątku przy wielowątkowym obliczaniu fraktala Mandelbrota.

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:

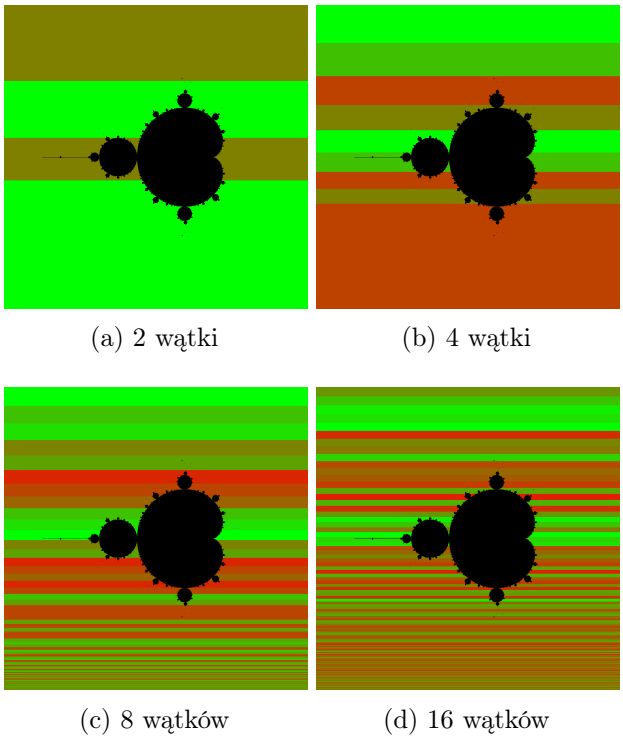
- Mały narzut (przydział raz).
- Dobra lokalność pamięci (ciągłe zakresy).

Wady:

- Słabe wyrównanie obciążenia, jeśli iteracje mają różny koszt.

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 chunku 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:

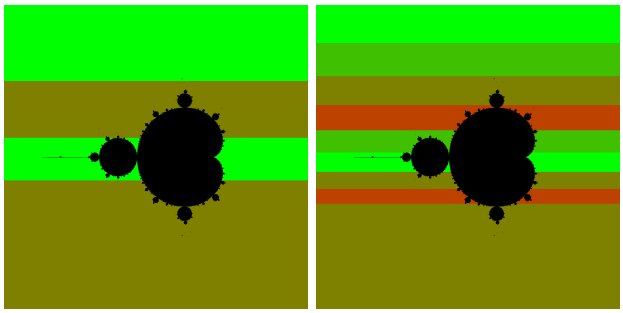
R - całkowita liczba pozostałych rekordów

T - liczba wątków / podziałów

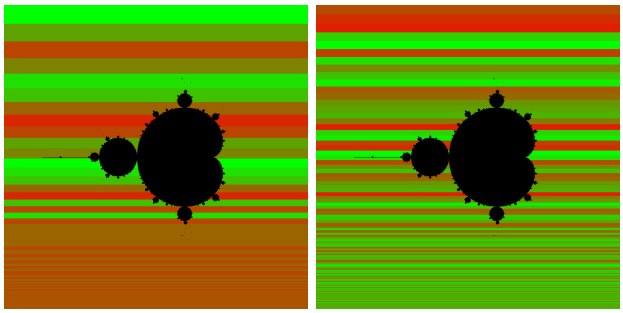
$C_{\min}$ - minimalny rozmiar bloku

Blok - wynikowy rozmiar chunku

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 overheadem (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:

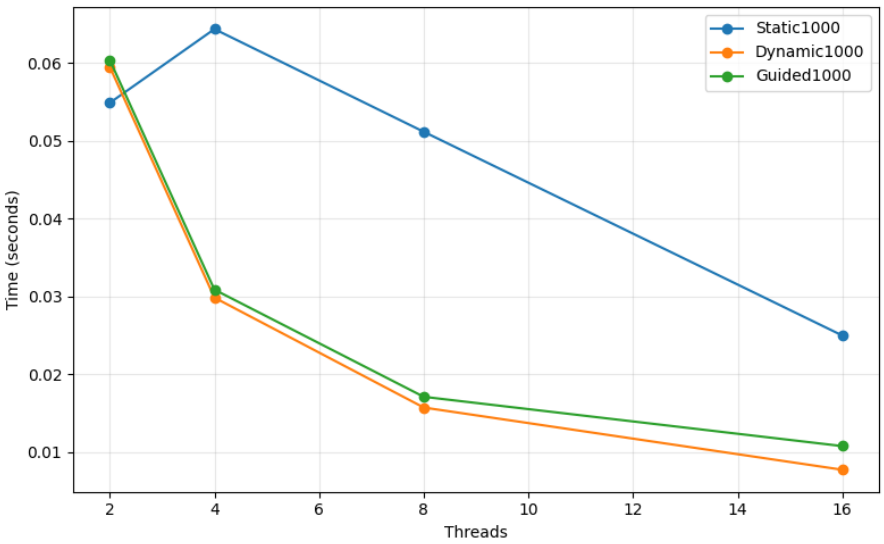
```
    // Deklaracja zmiennych
    double Cx, Cy;
    double PixelWidth = (CxMax - CxMin) / iXmax;
    double PixelHeight = (CyMax - CyMin) / iYmax;
    double Zx, Zy, Zx2, Zy2;
    double ER2 = EscapeRadius * EscapeRadius;

    /**
        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();
        auto start = omp_get_wtime();

        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)
        for (int iY = 0; iY < iYmax; ++iY) {
            ...
        }
```

Listing 1: Generalana struktura

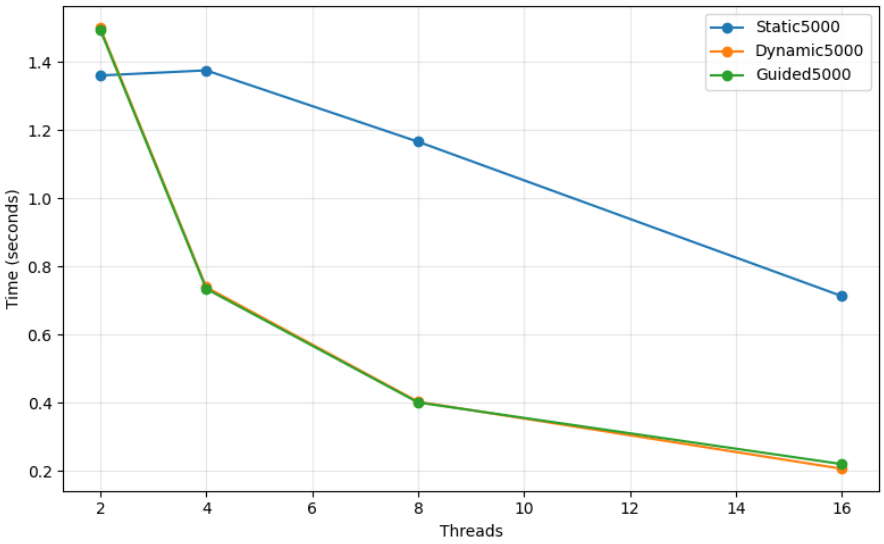
Różnica między tymi trzema implementacjami w kodzie, polega tylko i wyłącznie na zmianie HARMONOGRAM na static, dynamic, bądź guided.

3 Wyniki czasowe

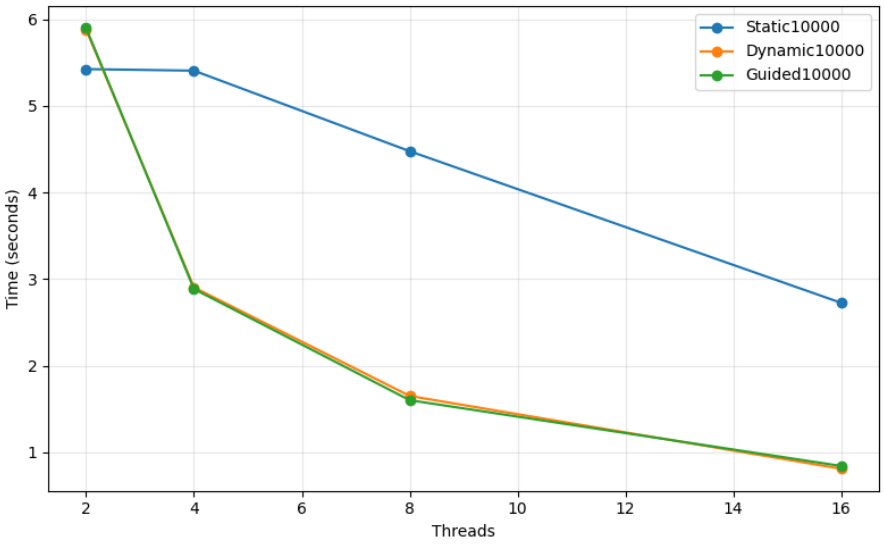
Na wykresach przedstawiono czasy obliczeń fraktala Mandelbrota dla trzech rozmiarów obrazu: 1000×1000, 5000×5000 oraz 10000×10000, z użyciem trzech harmonogramów OpenMP: `static`, `dynamic` oraz `guided`. Dla wszystkich testów analizowano wpływ liczby wątków (2, 4, 8, 16) na czas wykonania.



(a) 1000×1000



(b) 5000×5000



(c) 10000×10000

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

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.

Wynik z konsoli po uruchomieniu programu:

```
Guided: 1.59558 s
Thread 0: 1.59507 s, iterations executed: 584298201
Thread 1: 1.59508 s, iterations executed: 627917397
Thread 2: 1.59508 s, iterations executed: 581223939
Thread 3: 1.59506 s, iterations executed: 667332714
Thread 4: 1.59506 s, iterations executed: 691528148
Thread 5: 1.59508 s, iterations executed: 600463800
Thread 6: 1.59506 s, iterations executed: 711962679
Thread 7: 1.59506 s, iterations executed: 586498105

Static: 4.4613 s
Thread 0: 4.46128 s, iterations executed: 18163052
Thread 1: 4.46129 s, iterations executed: 26592758
Thread 2: 4.46129 s, iterations executed: 456477744
Thread 3: 4.46127 s, iterations executed: 2022970732
Thread 4: 4.46128 s, iterations executed: 2024707589
Thread 5: 4.46129 s, iterations executed: 457536751
Thread 6: 4.46129 s, iterations executed: 26606693
Thread 7: 4.46129 s, iterations executed: 18169664

Dynamic: 1.60001 s
Thread 0: 1.59999 s, iterations executed: 711962679
Thread 1: 1.6 s, iterations executed: 602700007
Thread 2: 1.59999 s, iterations executed: 586498105
Thread 3: 1.59999 s, iterations executed: 583139419
Thread 4: 1.59998 s, iterations executed: 627917397
Thread 5: 1.59998 s, iterations executed: 691528148
Thread 6: 1.6 s, iterations executed: 578987732
Thread 7: 1.59998 s, iterations executed: 668491496
```

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

Patrząc na ilość wykonanych iteracji dla każdego wątku można stwierdzić, że najlepsze rozłożenie pracy zapewnia Harmonogram guided oraz dynamic (przy założeniu domyślnych rozmiarów bloków).