

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 przydzieliśmy wiersz co przyjętą ilość wątków.

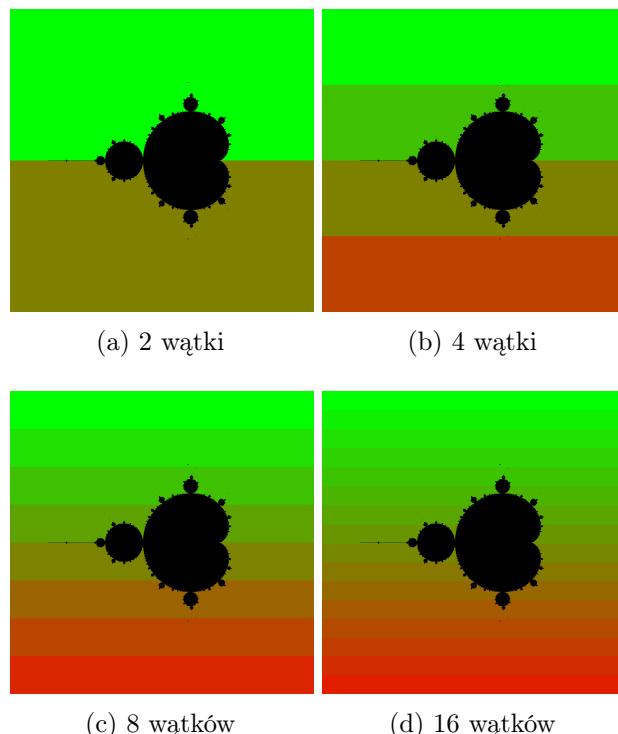


Figure 1: Harmonogramowanie static; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

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.

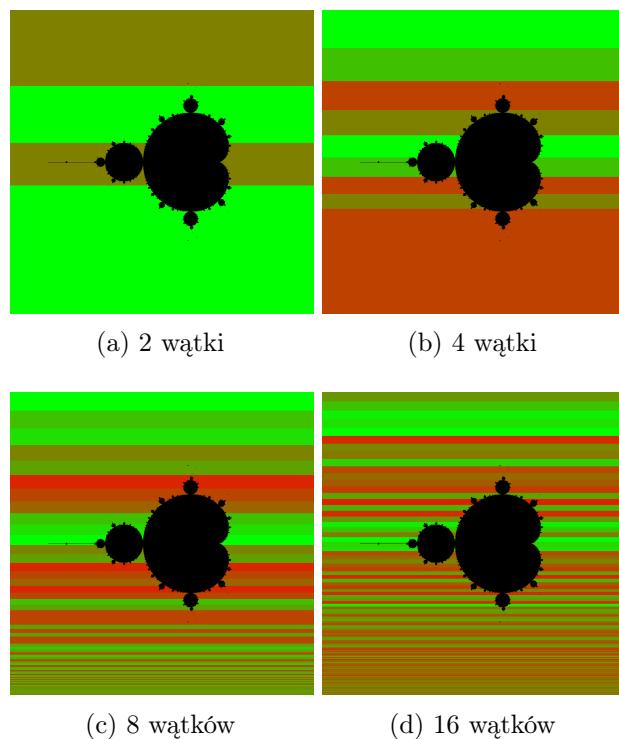


Figure 2: Harmonogramowanie Dynamic; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

Zalety:

- Świecone 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

Harmonogramowanie Guided to podejście hybrydowe pomiędzy static a dynamic; gdy wątek zarządza 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.

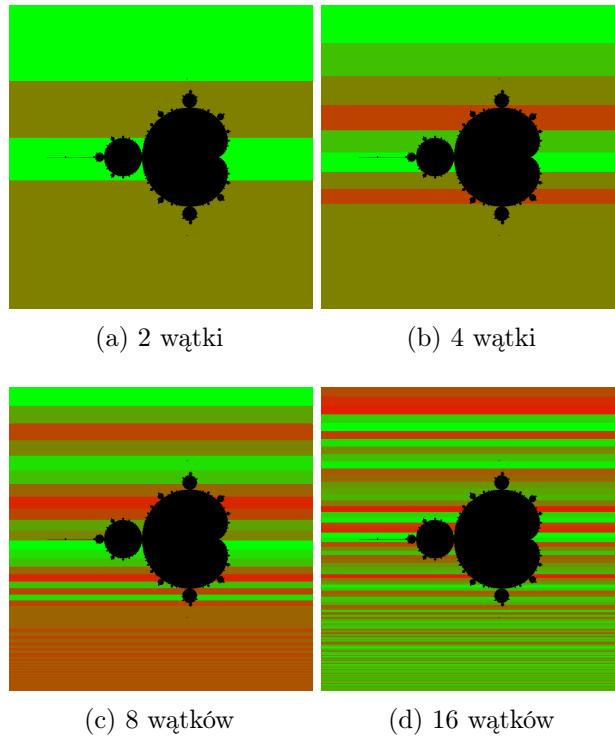


Figure 3: Harmonogramowanie Guided; podział bloków w zależności od ilości użytych wątków z domyślną wielkością bloku

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;

/**
    Rozpoczęcie pracy wątku, oraz wskazanie jakich zmiennych
    powinien uzyc "lokalnie", aby uniknąć współdzielenia pamięci
    z innymi wątkami
*/
#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;

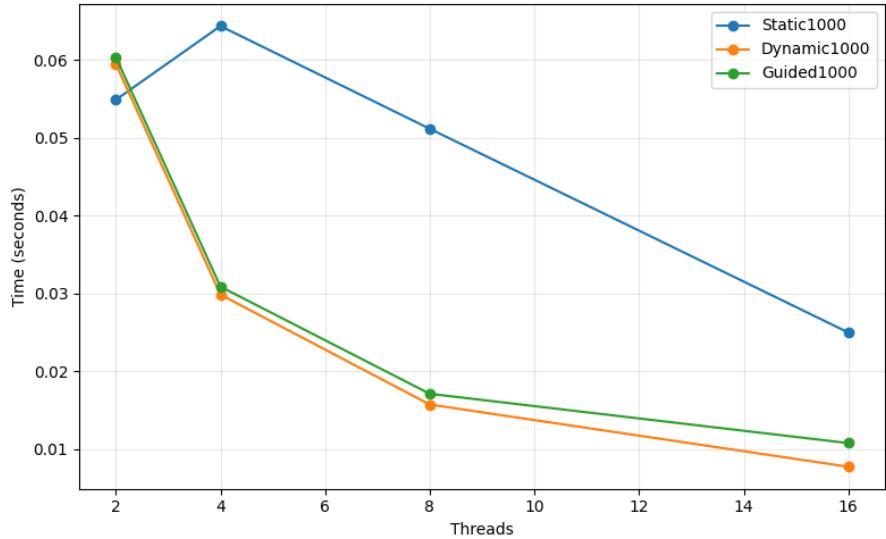
    // Wybor harmonogramowania oraz rozmiaru bloku
#pragma omp for schedule(HARMONOGGRAM, ROZMIAR_BLOKU)
    for (int iY = 0; iY < iYmax; ++iY) {
        ...
    }
    ...
}
```

Listing 1: Generalana struktura

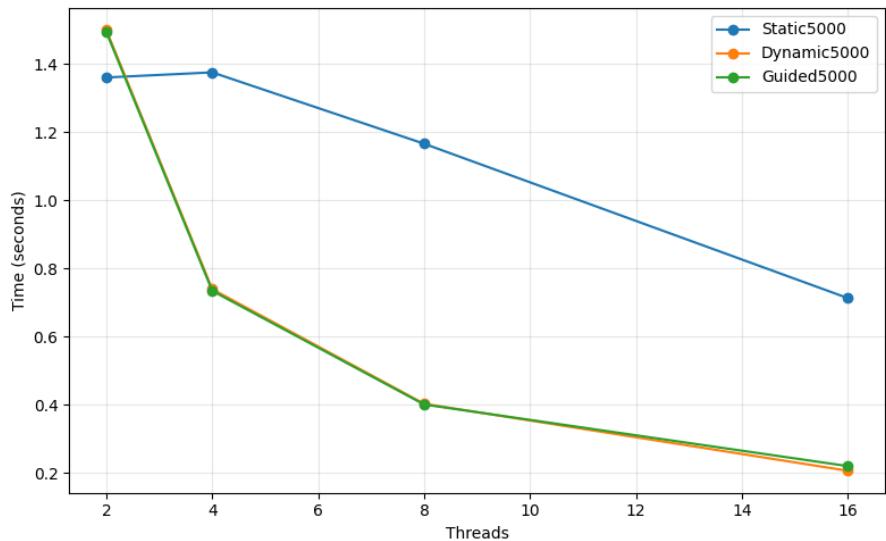
Różnica między tymi trzema implementacjami w kodzie, polega tylko i wyłącznie na zmianie HARMONOGRAM na static, dynamic, bądź guided.

3 Wyniki czasowe

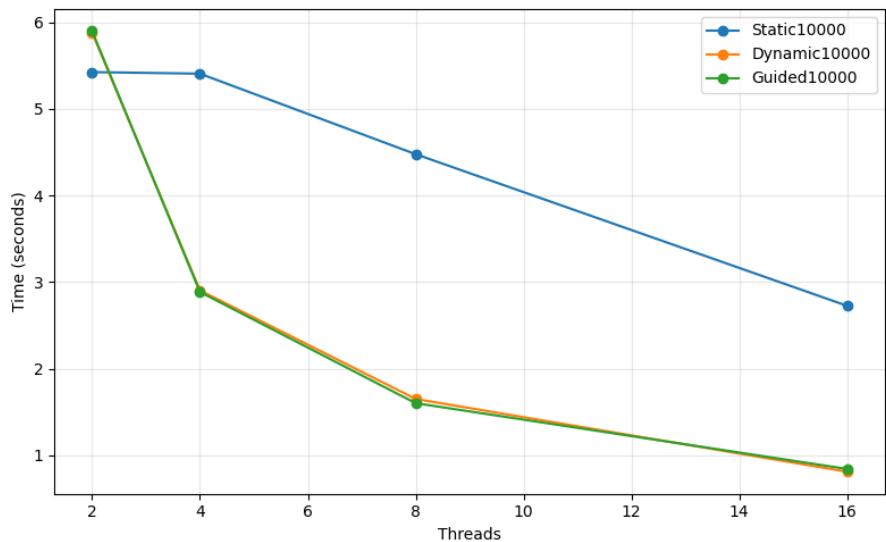
Na wykresach przedstawiono czasy obliczeń fraktala Mandelbrota dla trzech rozmiarów obrazu: 1000×1000 , 5000×5000 oraz 10000×10000 , z użyciem trzech harmonogramów OpenMP: static, dynamic oraz guided. Dla wszystkich testów analizowano wpływ liczby wątków (2, 4, 8, 16) na czas wykonania.



(a) 1000×1000



(b) 5000×5000



(c) 10000×10000

Figure 4: Wyniki czasowe w zależności od rozmiaru 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: 1.59759 s
Thread 0 iterations executed: 691528148
Thread 1 iterations executed: 711962679
Thread 2 iterations executed: 586498105
Thread 3 iterations executed: 580382681
Thread 4 iterations executed: 584298201
Thread 5 iterations executed: 627917397
Thread 6 iterations executed: 667332714
Thread 7 iterations executed: 601305058

Static: 4.4611 s
Thread 0 iterations executed: 18163052
Thread 1 iterations executed: 26592758
Thread 2 iterations executed: 456477744
Thread 3 iterations executed: 2022970732
Thread 4 iterations executed: 2024707589
Thread 5 iterations executed: 457536751
Thread 6 iterations executed: 26606693
Thread 7 iterations executed: 18169664

Dynamic: 1.59795 s
Thread 0 iterations executed: 583139419
Thread 1 iterations executed: 711962679
Thread 2 iterations executed: 670236686
Thread 3 iterations executed: 626172207
Thread 4 iterations executed: 697229612
Thread 5 iterations executed: 572770822
Thread 6 iterations executed: 586498105
Thread 7 iterations executed: 603215453
```

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).