Równoległe generowanie spirali Ulama z użyciem OpenMP

1 Cel ćwiczenia

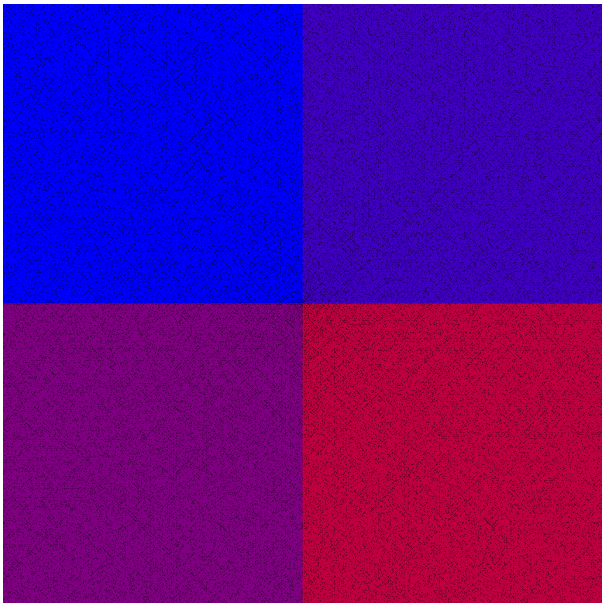
Celem projektu było zbadanie wydajności równoległego generowania spirali Ulama. W szczególności porównano dwie metody:

- **UlamBlocks** — równoległe przetwarzanie bloków macierzy z użyciem dyrektywy `collapse(2)`,
- **UlamBlocksNest** — równoległość zagnieżdżona, w której dla każdego bloku tworzony jest dodatkowy obszar równoległy.

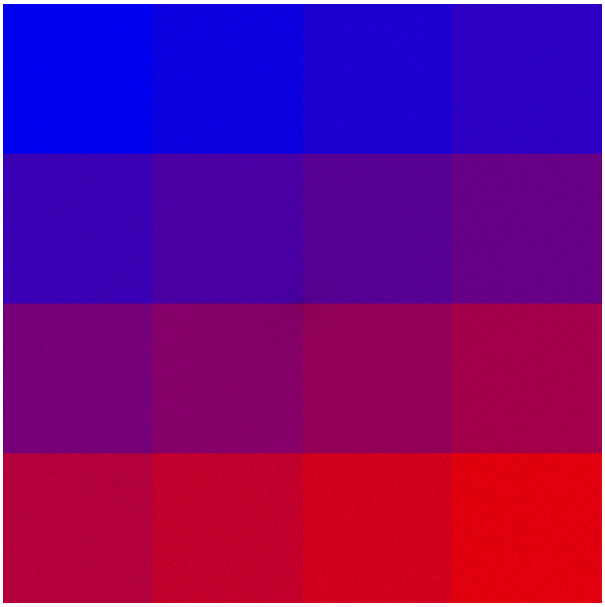
Analizowano wpływ liczby wątków oraz rozmiaru bloków na całkowity czas wykonania.

2 Techniki OpenMP

2.1 Wizualizajca



(a) Podział na bloki 2x2, 4 wątki



(b) Podział na bloki 4x4, 16 wątków

Figure 1: Obie metody dają to samo kolorowanie

2.2 Metoda UlamBlocks

Równoleglenie po blokach macierzy:

```
#pragma omp parallel
{
    #pragma omp for collapse(2) schedule(dynamic) nowait
    for (bi = 0; bi < Size; bi += block)
        for (bj = 0; bj < Size; bj += block)
            ...
}
```

Całość wykonywana jest w jednym obszarze równoległym.

2.3 Metoda UlamBlocksNest

Włączono pełne zagnieżdżanie:

```
omp_set_nested(1);
```

Każdy blok macierzy generuje *drugi* region równoległy:

```
#pragma omp parallel
{
    #pragma omp for collapse(2)
    for (bi ...)
        for (bj ...)
        {
            #pragma omp parallel
            {
                #pragma omp for collapse(2)
                for (x ...)
                    for (y ...)
                        ...
            }
        }
}
```

Pozwala to tworzyć hierarchiczne równoleglenie, lecz zwiększa narzut.

3 Konfiguracja eksperymentów

Dla rozmiaru macierzy:

$$Size = 1024$$

przeprowadzono pomiary dla:

- liczby wątków: $1, 2, 4, 8, 16$,
- wielkości bloków: od $Size/16$ do $Size/2$,
- liczby powtórzeń: 5.

4 Wyniki

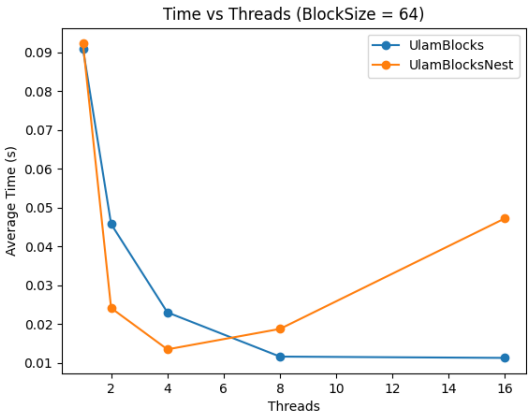
Wyniki zostały zebrane i przedstawione na wykresach zależnych od **BlockSize**, gdzie oś X to liczba wątków, a Y to średni czas wykonania.

Metoda **UlamBlocks** wykazała:

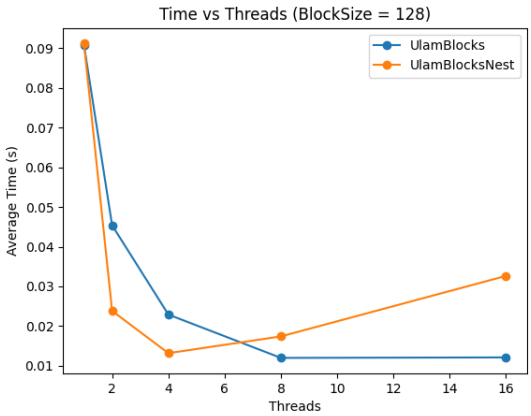
- niemal liniową skalowalność do 8 wątków,
- wzrost czasu przy zbyt dużych blokach,
- najlepsze wyniki dla $block = 128$.

Metoda **UlamBlocksNest** wykazała:

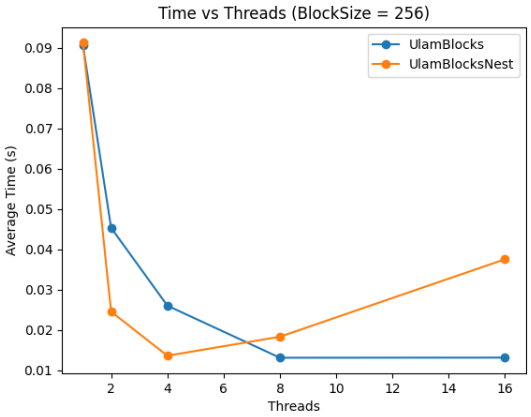
- wyraźnie większy narzut tworzenia zagnieżdżonych sekcji,
- spadek wydajności powyżej 4 wątków,



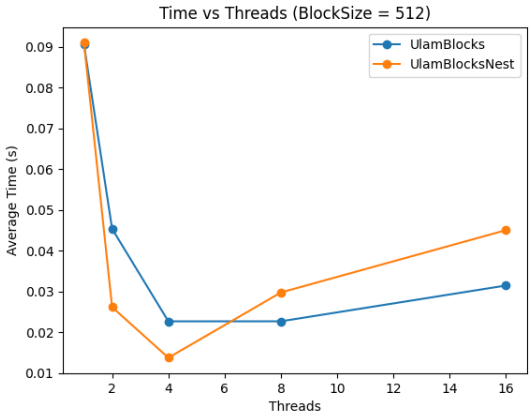
(a) Rozmiar bloku 64



(b) Rozmiar bloku 128



(c) Rozmiar bloku 256



(d) Rozmiar bloku 512

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

- silną zależność od kosztu zarządzania wątkami.

Listing 1: Czasy wykonania dla dużej liczby wątków oraz dużych bloków

```
UlamBlocksNest
BlockSize = 512, avg time = 0.0443896 s
Thread 0, time = 0.019412 s
Thread 1, time = 0.0145409 s
Thread 2, time = 0.0195141 s
Thread 3, time = 0.0171099 s
Thread 4, time = 0 s
Thread 5, time = 0 s
Thread 6, time = 0 s
Thread 7, time = 0 s
Thread 8, time = 0 s
Thread 9, time = 9.53674e-07 s
Thread 10, time = 9.53674e-07 s
Thread 11, time = 0 s
Thread 12, time = 0 s
Thread 13, time = 0 s
Thread 14, time = 0 s
Thread 15, time = 0 s
...
UlamBlocks
BlockSize = 512, avg time = 0.0394888 s
Thread 0, time = 0.0226781 s
Thread 1, time = 0.0436931 s
Thread 2, time = 0 s
Thread 3, time = 0 s
Thread 4, time = 0 s
Thread 5, time = 0 s
Thread 6, time = 0 s
Thread 7, time = 0 s
Thread 8, time = 0.0226719 s
Thread 9, time = 0 s
Thread 10, time = 0 s
Thread 11, time = 0 s
Thread 12, time = 0 s
Thread 13, time = 0.043663 s
Thread 14, time = 0 s
Thread 15, time = 0 s
```

Niektóre wątki siedzą bezczynnie z powodu dużych bloków, które dzielą zadanie np. tak jak w tym przykładzie na 4 bloki.

5 Analiza

5.1 Porównanie metod

- **UlamBlocks** jest efektywna dzięki jednemu regionowi równoległemu i małemu narzutowi.
- **UlamBlocksNest** generuje dużą liczbę dodatkowych wątków, co prowadzi do degradacji wydajności. Jest ona jednak lepsza dla niskiej liczby wątków.

Równoległość zagnieżdżona okazuje się nieopłacalna w przypadku tak drobnego zadania jak operacje na pojedynczych pikselach, jeśli chcemy, aby skolowało się one z liczbą wątków.

5.2 Wpływ rozmiaru bloków

- małe bloki → dużo iteracji pętli, spory narzut planisty OpenMP,
- duże bloki → słabe wyrównanie obciążenia,
- optimum: bloki średnie (ok. 64–128 dla Size=1024).

5.3 Skalowanie

- dobra skalowalność dla dobrze dobrego rozmiaru bloków,
- powyżej 8 (dla nested 4) wątków, szczególnie przy dużych blokach, w niektórych przypadkach możemy zauważyć, że tracimy na wydajności ze względu na to że pracują tylko niektóre wątki a my i tak powołujemy wszystkie do życia.