Alicja Kraska-Kapiszka, Informatyka N1, sem. 7, gr. 41A

Zadanie 1. - Zbiór Mandelbrota

Cel zadania:

- Celem zadania była modyfikacja kodu napisanego w języku C, którego zadaniem było wygenerowanie zbioru Mandelbrota w formacie .ppm. Zbiór Mandelbrota jest zbiorem liczb zespolonych c takich, że ciąg zdefiniowany wzorem:

  $z_0 = 0$

  $z_{n+1} = z_n^2 + c$

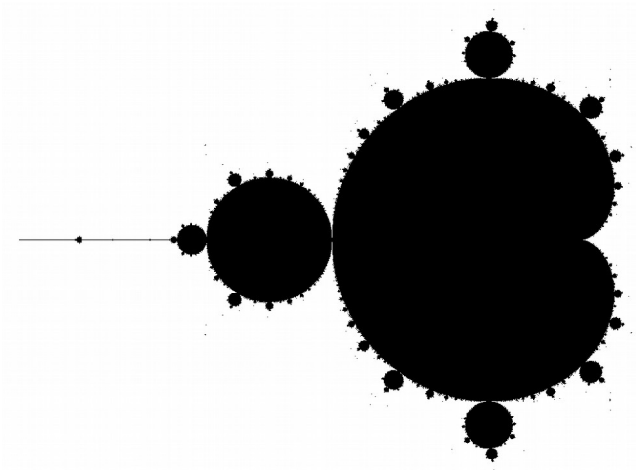
  nie dąży do nieskończoności. Brzeg tego zbioru jest fraktalem.
- Podany kod należało zmodyfikować tak, aby umożliwić równoległe wykonywanie obliczeń przez kilka wątków.

Przebieg zadania:

- W pierwszej kolejności należało usunąć zapis do pliku poza pętle. W tym celu zastąpiłam tablicę jednowymiarową color[3] tablicą dwuwymiarową color[iXmax*iYmax][3]. Poprzednio w każdej iteracji pętli ustalany był kolor piksela, a następnie był on zapisywany do pliku .ppm. W kolejnych iteracjach wartości pikseli były nadpisywane. W mojej wersji, w pętli kolor danego piksela jest zapisywany w tablicy (wspólnej dla wszystkich pikseli), natomiast zapis do pliku następuje na samym końcu, poza pętlami:

  fwrite(color, 1, iXmax * iYmax * 3, fp);
- Następnie należało zrównoleglić pętle przy użyciu OpenMP. W tym celu przed pętlą dodałam dyrektywę:

  #pragma omp parallel for shared(color) private(counter, Zx2, Zy2, Zx, Zy, Cx, Cy, iY, iX, Iteration)
  - dyrektywa „#pragma omp parallel for” służy do stworzenia grupy wątków, a następnie podzielenia pomiędzy nie iteracji pętli;
  - „shared” określa zmienne, które są wspólne dla wszystkich wątków – tutaj jest to tablica color, która jest uzupełniana kolorami pikseli, a więc każdy wątek musi mieć do niej dostęp;
  - „private” określa zmienne, które są prywatne – każdy wątek posiada ich własną kopię; są to zmienne używane do iteracji pętli oraz obliczeń fraktala.
- Za pomocą funkcji omp_get_wtime() zmierzyłam czasy wykonania programu dla różnej liczby wątków, ustawianej za pomocą funkcji omp_set_num_threads() oraz dla różnych rozmiarów fraktala.
- Wynik działania programu:



Wyniki:

- Dla fraktala o rozmiarze 400x400 pikseli i maksymalnej liczby iteracji 100:
  - 1 wątek: 0.054532 s
  - 2 wątki: 0.033964 s
  - 4 wątków: 0.043172 s
  - 8 wątków: 0.032614 s
  - 16 wątków: 0.033809 s
- Dla fraktala o rozmiarze 800x800 pikseli i maksymalnej liczby iteracji 200:
  - 1 wątek: 0.139069 s
  - 2 wątki: 0.083773 s
  - 4 wątków: 0.094051 s
  - 8 wątków: 0.072829 s
  - 16 wątków: 0.063677 s
- Dla fraktala o rozmiarze 1600x1600 pikseli i maksymalnej liczby iteracji 400:
  - 1 wątek: 0.836983 s
  - 2 wątki: 0.42953 s
  - 4 wątków: 0.431483 s
  - 8 wątków: 0.377958 s
  - 16 wątków: 0.336537 s

Wnioski:

- Im mniejszy fraktal, tym oczywiście krótszy był czas wykonywania obliczeń.
- Wraz ze zwiększeniem liczby wątków, zmniejszał się czas wykonywania obliczeń.
- Największe różnice występowały między czasem wykonania dla jednego i dwóch wątków.
- Im większy fraktal, tym różnice były większe – dla najmniejszego fraktala czasy wykonania dla 2-16 wątków były porównywalne, natomiast dla największego – różnice wynosiły nawet 0.05 s.
- Dla 4 wątków czas wykonania programu był nieznacznie dłuższy niż dla 2 wątków.
- Ostatecznie najlepsze wyniki otrzymano dla 8 lub 16 wątków.