برنامه نویسی موازی

پروژه 2

اعضای گروه:

صفورا علوی پناه - 810100254

عاطفه میرزاخانی - 810100220

پیاده سازی mendelbrot set

کد ارائه‌شده برای تولید و ذخیره تصاویر مجموعه ماندلبروت به صورت سریال و موازی طراحی شده است. این برنامه از الگوریتم ماندلبروت برای محاسبه تعداد تکرارهای لازم برای هر نقطه در صفحه مختلط استفاده می‌کند. از تکنیک OpenMP برای تسریع محاسبات موازی استفاده شده است و قابلیت زوم در مجموعه با مشخص کردن نقاط مرکزی و فاکتور زوم فراهم شده است.

روش کار:

مراحل اجرای کد به صورت زیر است:

1. برای هر نقطه در صفحه مختلط، مقدار تکرارهای لازم تا خروج از دایره واحد (|z|>2) محاسبه می‌شود.
2. تعداد تکرارهای هر نقطه به رنگ RGB تبدیل می‌شود تا تصویر نهایی تولید شود. (نقاط همگرا به رنگ سفید و واگرا با درصد های مختلف از آبی به سیاه تعیین شده‌اند.)
3. تصویر تولیدشده به صورت سریال و موازی ذخیره می‌شود.
4. قابلیت زوم فراهم شده که با کاهش محدوده مختصات صفحه مختلط، جزئیات بیشتری از مجموعه نمایش داده می‌شود.

روش اجرای برنامه به دو صورت زیر است:

- **نسخه سریال**: تمام نقاط تصویر به ترتیب پردازش می‌شوند.
- **نسخه موازی**: از فریم ورک OpenMP برای تقسیم پردازش نقاط بین چندین نخ (thread) استفاده می‌شود.

- برای زوم کردن با استفاده یک حلقه با zoom_iteration = 3 سه تصویر برای زوم در هر یک از نسخه‌های موازی و سریال تولید می‌شود.

```
140     // Zoom parameters
141     float zoom_factor =0.8; // f < 1 = zoom in | f > 1 = zoom out
142     float center_x = -0.75;  // Center of zoom (real part)
143     float center_y = 0.0;    // Center of zoom (imaginary part)
144     int zoom_iterations = 3;
145
146     for (int i = 0; i < zoom_iterations; ++i)
147     {
148         cout << "Zoom iteration " << i + 1 << "...\n";
149
150         // Adjust bounds for zoom
151         float x_range = (x_max - x_min) * zoom_factor;
152         float y_range = (y_max - y_min) * zoom_factor;
153         x_min = center_x - x_range / 2;
154         x_max = center_x + x_range / 2;
155         y_min = center_y - y_range / 2;
156         y_max = center_y + y_range / 2;
157
```

نسخه سریال:

در نسخه سریال:
1. تابع `generate_mandelbrot_serial` با پیمایش خط‌به‌خط تصویر، مقدار تکرارهای لازم برای هر نقطه را محاسبه می‌کند.
2. عملیات سریال در حلقه‌ای ساده و بدون تقسیم‌بندی به چندین وظیفه انجام می‌شود.
3. نتایج محاسبات در یک آرایه ذخیره شده و سپس به فایل تصویری PPM تبدیل می‌شوند.

مزایا:

- ساده و قابل‌فهم.
- نیازی به هماهنگی بین رشته‌ها ندارد.

معایب:

- زمان اجرا با افزایش ابعاد تصویر یا تعداد تکرارها به‌شدت افزایش می‌یابد.
- استفاده بهینه‌ای از منابع پردازشی (مانند چند هسته‌ای بودن پردازنده) ندارد.

نسخه موازی:

در نسخه موازی:
1. از دستورالعمل‌های OpenMP برای موازی‌سازی حلقه محاسبات استفاده می‌شود.
2. هر رشته پردازشی بخشی از تصویر را پردازش کرده و مقادیر تکرارها را به آرایه خروجی می‌فرستد.
3. این عملیات باعث تسریع محاسبات در سیستم‌های چند هسته‌ای می‌شود.

مزایا:

- کاهش قابل‌توجه زمان اجرا.
- استفاده بهینه از پردازنده‌های چند هسته‌ای.

معایب:

- نیاز به هماهنگی بین رشته‌ها.
- وابستگی به تنظیم مناسب OpenMP و سیستم‌های پشتیبانی‌شده.

نتایج:

۱. زمان اجرای نسخه سریال و موازی

- زمان اجرای سریال برای تصاویر بزرگ معمولاً بیشتر است، در حالی که اجرای موازی به دلیل تقسیم‌بندی محاسبات، زمان کمتری نیاز دارد.
- در هر مرحله زوم، زمان اجرای هر دو نسخه اندازه‌گیری شده و مقایسه می‌شود.

۲. تصاویر تولیدی

- برای هر مرحله زوم، دو تصویر ذخیره می‌شود:
  - یک تصویر تولیدشده با نسخه سریال.
  - یک تصویر تولیدشده با نسخه موازی.
- تصاویر با وضوح بالا در فایل‌های PPM ذخیره می‌شوند و تفاوتی در کیفیت تصاویر سریال و موازی مشاهده نمی‌شود.

۳. مقایسه سرعت

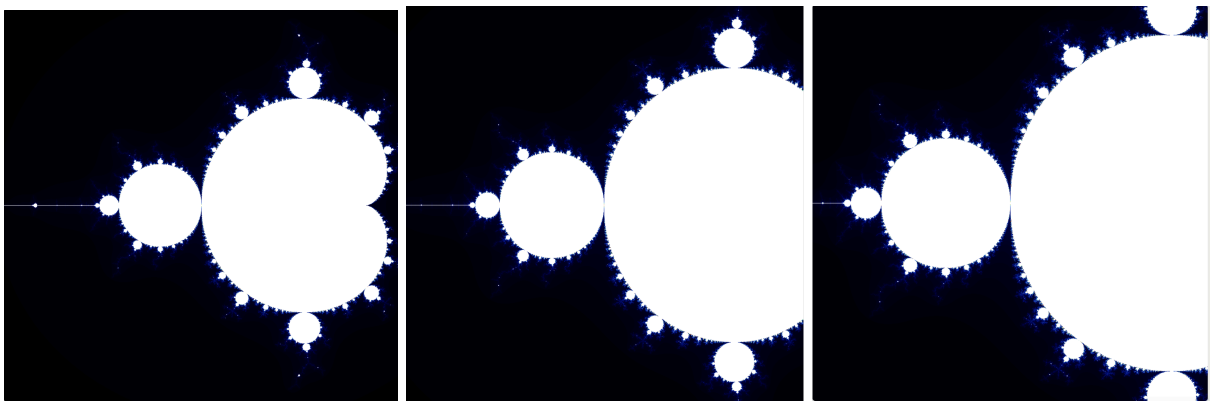
- Speedup با فرمول زیر محاسبه می‌شود: $speedUp = \frac{serial\ execution\ time}{parallel\ execution\ time}$
- افزایش تعداد هسته‌های پردازشی و مناسب بودن تقسیم‌بندی، سرعت‌دهی بیشتری ایجاد می‌کند.

خروجی ها:

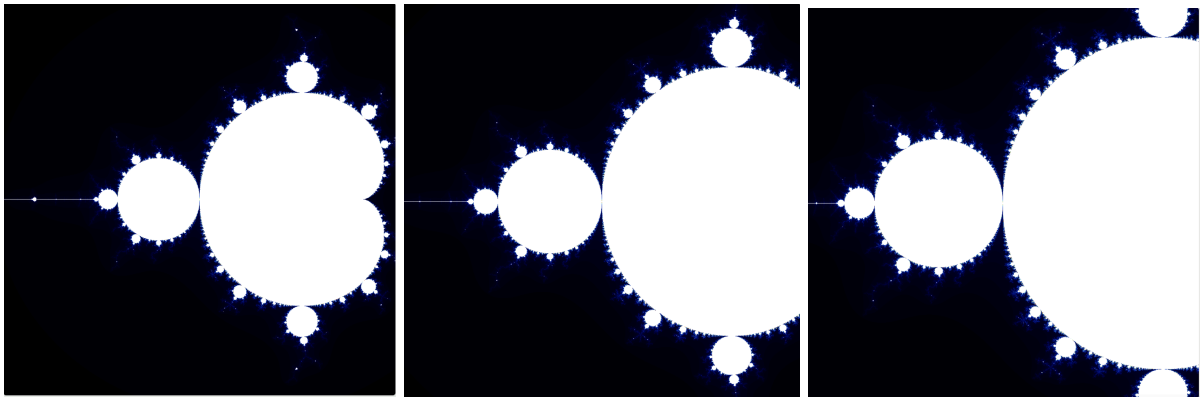
خروجی command line :

```
PS D:\uni\Terms\7\7-PP\CA\CA2\q1> make
g++ main1.cpp -fopenmp -o main1
PS D:\uni\Terms\7\7-PP\CA\CA2\q1> ./main1
Zoom iteration 1...
Serial execution time: 18.511 seconds
Saved serial image: mandelbrot_serial_zoom_1.ppm
Parallel execution time: 3.53 seconds
Saved parallel image: mandelbrot_parallel_zoom_1.ppm
Speedup (Serial / Parallel): 5.24391
Zoom iteration 2...
Serial execution time: 22.672 seconds
Saved serial image: mandelbrot_serial_zoom_2.ppm
Parallel execution time: 4.656 seconds
Saved parallel image: mandelbrot_parallel_zoom_2.ppm
Speedup (Serial / Parallel): 4.86942
Zoom iteration 3...
Serial execution time: 26.961 seconds
Saved serial image: mandelbrot_serial_zoom_3.ppm
Parallel execution time: 5.846 seconds
Saved parallel image: mandelbrot_parallel_zoom_3.ppm
Speedup (Serial / Parallel): 4.61187
PS D:\uni\Terms\7\7-PP\CA\CA2\q1>
```

خروجی نسخه موازی برای zoom factor = 0.8 که باعث zoom in سه مرحله‌ای می‌شود:



خروجی نسخه سریال برای zoom factor = 0.8 که باعث zoom in سه مرحله‌ای می‌شود:



پیاده سازی julia set

هدف این سوال، تولید تصویر یک نسخه از مجموعه جولیا با استفاده از فرمول تکرار Z(k+1) = Z(k)^2 + c است که c عدد مختلطی ثابت است. این سوال به دو صورت پیاده‌سازی شده است: نسخه سریال و نسخه موازی.

روش کار:

- تکرار فرمول julia:

برای هر نقطه مختصات (x, y) در فضای مختصات داده شده، مقدار اولیه Z محاسبه می‌شود. تا تعداد تکرار مشخص یا خروج از شعاع مشخص، مقدار Z با استفاده از فرمول به‌روزرسانی می‌شود. تعداد تکرار تا رسیدن به شرط خروج به‌عنوان مقدار رنگی هر نقطه استفاده می‌شود.

- تخصیص رنگ:

با توجه به تعداد تکرار، رنگ هر نقطه بر اساس یک مقیاس گرادیان تعیین شده و به فرمت RGB ذخیره می‌شود.

- ذخیره تصویر:

تصویر نهایی با فرمت PPM نوشته می‌شود که شامل اطلاعات رنگی هر پیکسل است.

نسخه سریال:

- تمام محاسبات بر روی یک هسته پردازنده انجام می‌شود.
- یک حلقه دوبخشی برای پیمایش تمام پیکسل‌های تصویر اجرا شده و هر پیکسل به صورت ترتیبی پردازش می‌شود.

نسخه موازی:

- با استفاده از OpenMP، پردازش بین چندین هسته توزیع می‌شود.
- از دستور #pragma omp parallel for و روش تقسیم‌بندی dynamic برای تعادل بهتر بار پردازشی استفاده شده است.
- هر رشته (Thread) به صورت مستقل محاسبات نقطه‌ها را انجام داده و نتایج به آرایه RGB نهایی افزوده می‌شود.

نتایج:

این برنامه را دو بار اجرا کردیم نتایج آنها به صورت زیر است.

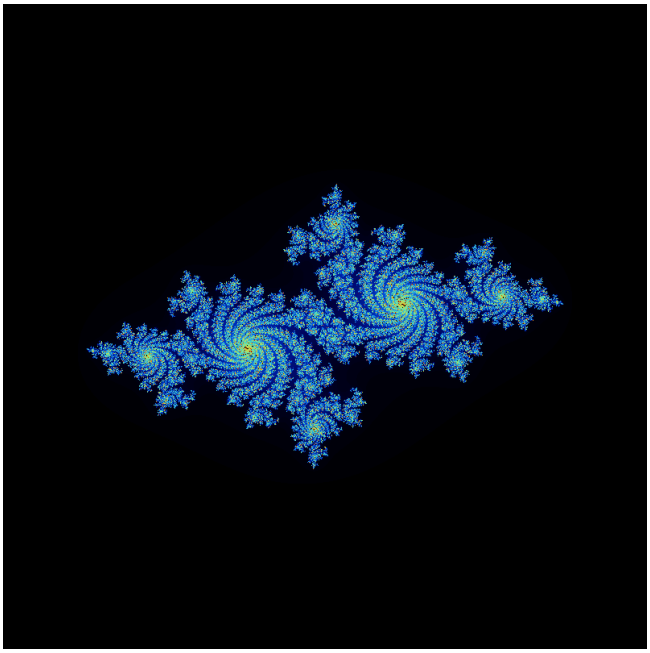
اجرای برنامه برای c=-0.7+0.27015j :

```
PS D:\uni\Terms\7\7-PP\CA\CA2\q2> make
g++ main.cpp -fopenmp -o main
PS D:\uni\Terms\7\7-PP\CA\CA2\q2> ./main
Plot a version of the Julia set for Z(k+1) = Z(k)^2 - 0.7 + 0.27015i
Serial execution time: 0.213 seconds
Parallel execution time (OpenMP): 0.0380001 seconds
Speedup (Serial / Parallel): 5.60525
PS D:\uni\Terms\7\7-PP\CA\CA2\q2>
```

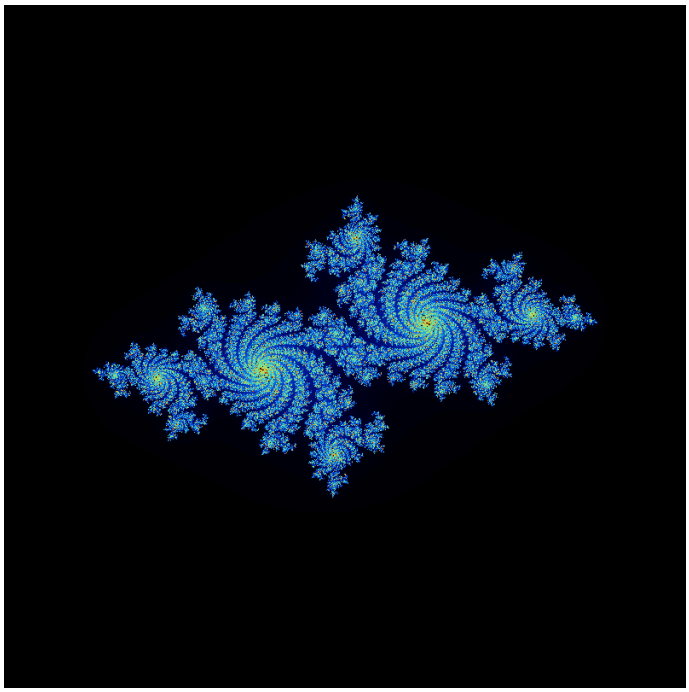
همانطور که میشود speedup ای برابر 5.6 دریافت کردیم.

خروجی های این برنامه:

خروجی سریال:



خروجی موازی:



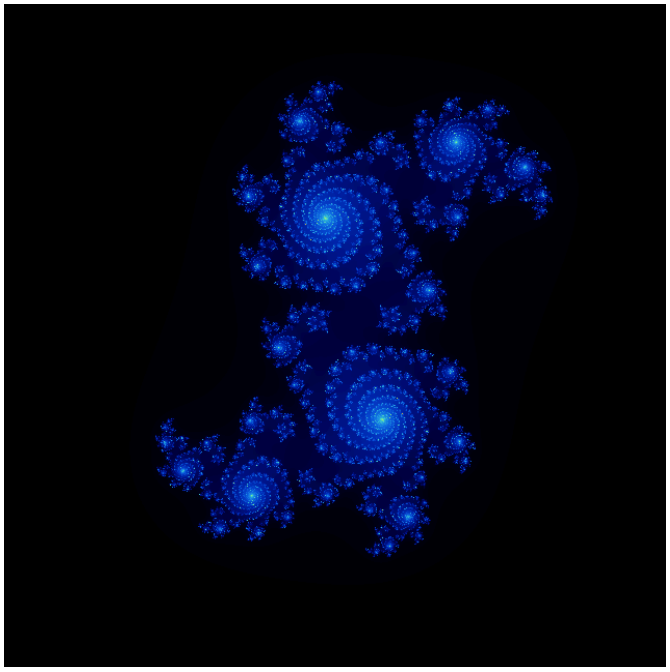
اجرای برنامه برای c=0.355+0.355j :

```
PS D:\uni\Terms\7\7-PP\CA\CA2\q2> ./main
Plot a version of the Julia set for Z(k+1) = Z(k)^2 + 0.355 + 0.355i
Serial execution time: 0.0780001 seconds
Parallel execution time (OpenMP): 0.0149999 seconds
Parallel execution time (OpenMP): 0.0149999 seconds
Speedup (Serial / Parallel): 5.20005
PS D:\uni\Terms\7\7-PP\CA\CA2\q2>
```

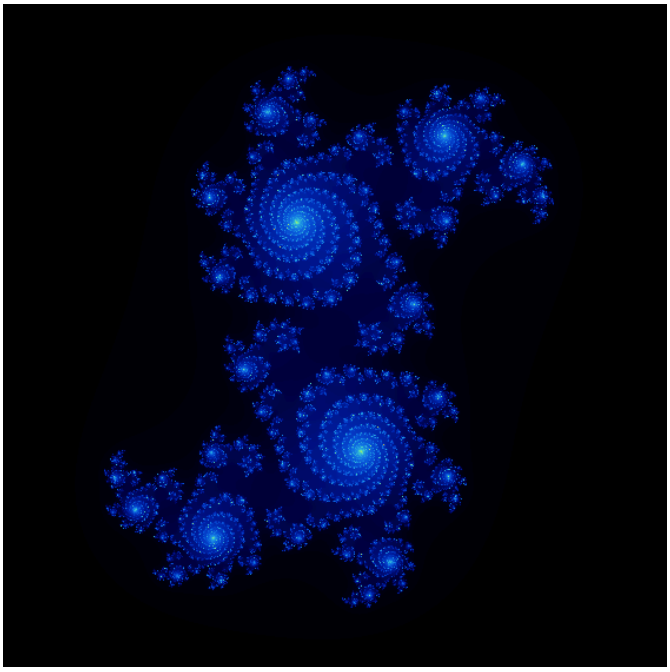
همانطور که میشود speedup ای برابر 5.2 دریافت کردیم.

خروجی های این برنامه:

خروجی سریال:



خروجی موازی:



پیاده سازی monte carlo

هدف این سوال محاسبه مقدار تقریبی عدد پی (π) با استفاده از روش مونت‌کارلو و مقایسه اجرای سریال و موازی آن است. این روش به‌طور تصادفی نقاطی را در یک فضای مشخص تولید می‌کند و از نسبت نقاط داخل دایره به کل نقاط برای تخمین عدد π بهره می‌گیرد. علاوه بر محاسبه مقدار π، این گزارش به مقایسه عملکرد نسخه‌های سریال و موازی الگوریتم با استفاده از OpenMP پرداخته است.

روش مونت‌کارلو:

- نقاط تصادفی در بازه [0,1] برای مختصات x و y تولید می‌شوند.
- بررسی می‌شود که آیا این نقاط داخل یک دایره با شعاع واحد قرار دارند.
- نسبت تعداد نقاط داخل دایره به کل نقاط ضرب در 4 مقدار تقریبی عدد π را می‌دهد.
- دقت الگوریتم مستقیماً به تعداد نقاط تولید شده وابسته است؛ با افزایش تعداد نقاط، خطای تخمین کاهش می‌یابد.

نسخه سریال:

- در نسخه سریال، تمام محاسبات روی یک هسته پردازشی انجام می‌شود.
- از کلاس‌های تصادفی مدرن C++ (مانند std::mt19937 و std::uniform_real_distribution) برای تولید نقاط تصادفی استفاده شده است که در مقایسه با std::rand از ایمنی بیشتری در محیط‌های چندنخی برخوردارند.

نسخه موازی:

- در نسخه موازی، با استفاده از کتابخانه OpenMP و دستور #pragma omp parallel for محاسبات بین چندین thread توزیع شده اند.
- همچنین از دستور #pragma omp atomic استفاده شده است. این دستور تضمین می‌کند که عملیات افزایش شمارش نقاط داخل دایره، به صورت ایمن و بدون بروز شرایط رقابتی بین thread ها انجام شود.

نتایج:

نتایج اجرای برای 10000000 نقطه تولید شده:

```
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> make
g++ main.cpp -fopenmp -o main
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14155
Serial execution time: 1.368 seconds

Parallel Pi estimation: 3.14155
Parallel execution time: 0.3 seconds

Speedup: 4.56
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14181
Serial execution time: 1.442 seconds

Parallel Pi estimation: 3.14182
Parallel execution time: 0.394 seconds

Speedup: 3.6599
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14287
Serial execution time: 1.398 seconds

Parallel Pi estimation: 3.14117
Parallel execution time: 0.292 seconds

Speedup: 4.78767
PS D:\uni\Terms\7\7-PP\CA\CA2\q3>
```

مقدار speedup ها در اجراهای مختلف: 4.56 و 3.65 و 4.78

نتایج اجرای برای 100000 نقطه تولید شده:

```
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14492
Serial execution time: 0.0190001 seconds

Parallel Pi estimation: 3.14936
Parallel execution time: 0.00199986 seconds

Speedup: 9.50072
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14376
Serial execution time: 0.023 seconds

Parallel Pi estimation: 3.14752
Parallel execution time: 0.00600004 seconds

Speedup: 3.83331
PS D:\uni\Terms\7\7-PP\CA\CA2\q3> ./main
Serial Pi estimation: 3.14768
Serial execution time: 0.0159998 seconds

Parallel Pi estimation: 3.14636
Parallel execution time: 0.00900006 seconds

Speedup: 1.77774
PS D:\uni\Terms\7\7-PP\CA\CA2\q3>
```

مقدار speedup ها در اجراهای مختلف: 9.5 و 3.83 و 1.77

که مشاهده می کنیم زمانی که تعداد نقاط تولید شده بیشتر است تخمین بهتری از عدد pi داریم.