پروژه دوم برنامه نویسی موازی

کسرا نوربخش ۸۱۰۱۰۰۲۳۰

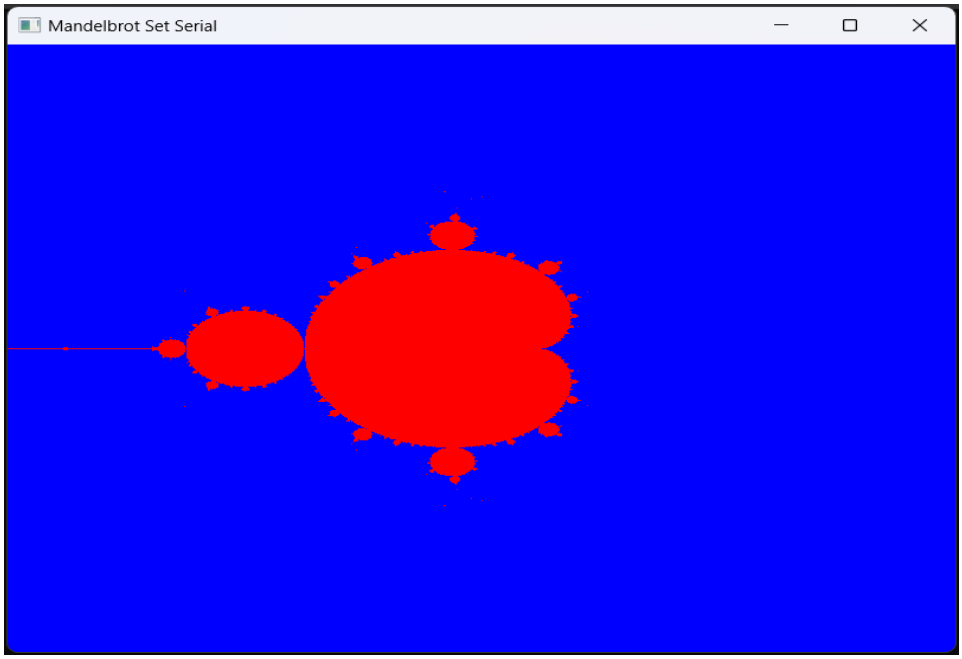
شهنام فیضیان 810100197

مقدمه

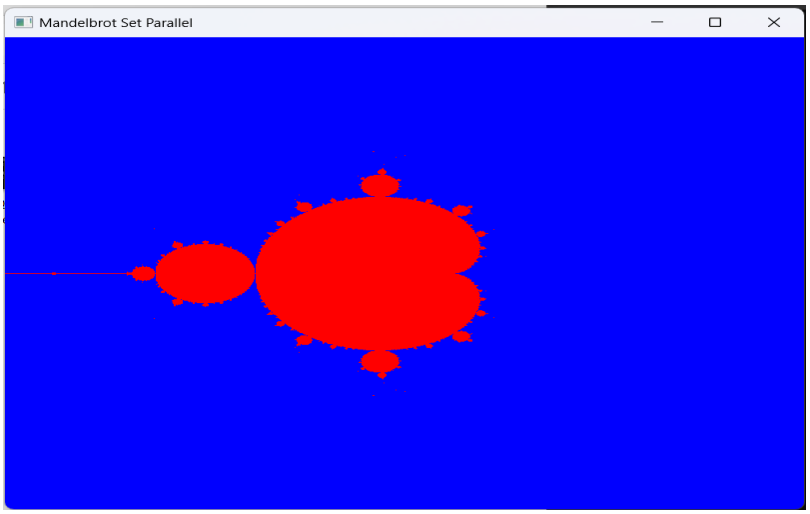
در این پروژه در دو ســـوال اول به دنبال پیدا کردن مرز های واگرایی و هم گرایی داده ها، با انجام دادن عملیات مجذور آن ها و اضافه کردن 1 یا 2 عدد ثابت رفتیم. در سوال آخر هم با استفاده از تکنیک Monte Carlo به تخمین عدد π پرداختیم. ابتدا پیاده سازی در حالت سریال انجام دادیم و سپس پیاده سازی موازی با استفاده از OpenMP. همچنین در بخش مربوط به هر سوال میزان speedup(که با استفاده از کتابخانه ipp صورت گرفت) و همچنین خروجی، گزارش شد. برای نشان دادن اشکال ایجاد شده در دو سوال اول از کتابخانه SFML استفاده گردید.

Mandelbrot Set

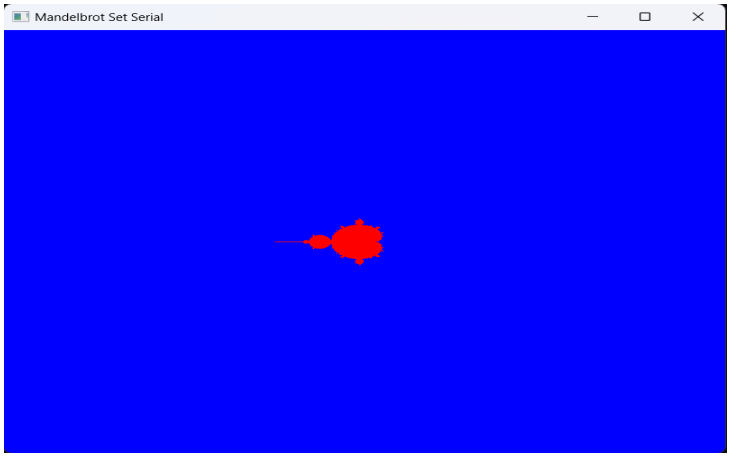
ابتدا به مشاهده خروجی ها و میزان تسریع می پردازیم:



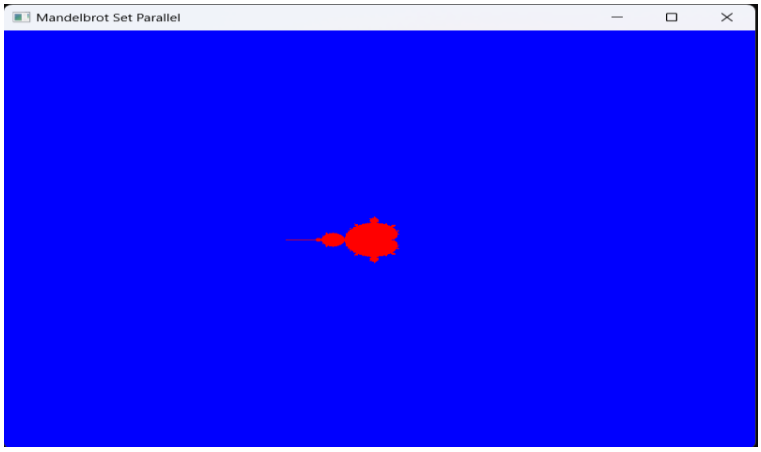
عکس 1: خروجی Mandelbrot سریال، برای c = 4j



عکس 2: خروجی Mandelbrot موازی، برای c = 4j



عکس 3: خروجی Mandelbrot سریال، برای c = 16j



عکس 4: خروجی Mandelbrot موازی، برای c = 16j

```
Serial Calculation Run time = 25146902568 cycles
Parallel Calculation Run time = 7839497293 cycles
Calculation Speedup = 3.20772

Serial Rendering Run time = 362039998 cycles
Parallel Rendering Run time = 1502483351 cycles
Rendering Speedup = 0.240961

Total Speedup = 2.73057
```

عکس 5: میزان تسریع Mandelbrot، برای نمایشگر با 8 thread

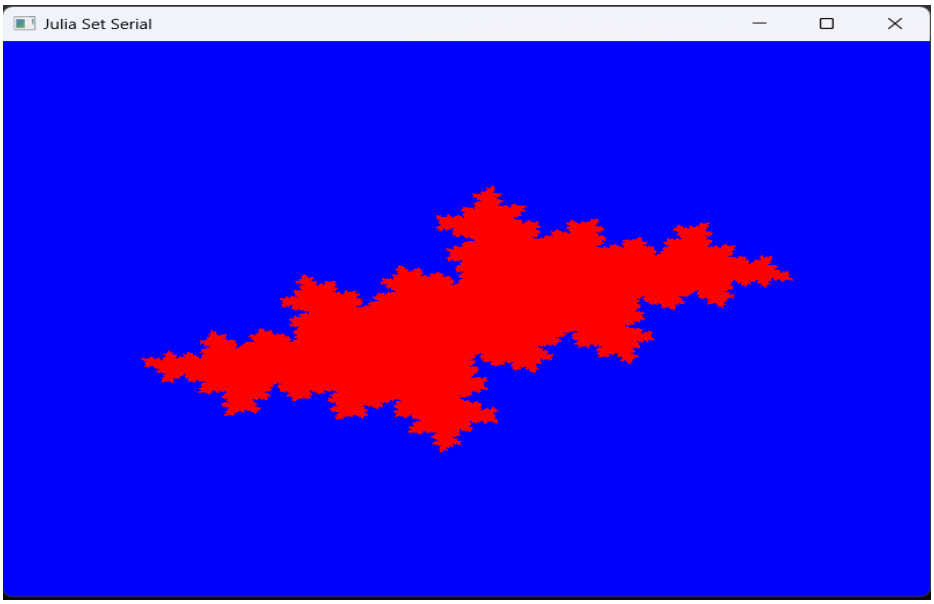
```
Serial Calculation Run time = 22382608810 cycles
Parallel Calculation Run time = 6571405736 cycles
Calculation Speedup = 3.40606

Serial Rendering Run time = 690120922 cycles
Parallel Rendering Run time = 748145567 cycles
Rendering Speedup = 0.922442

Total Speedup = 3.15221
```

عکس 6: میزان تسریع Mandelbrot، برای نمایشگر با 2 thread

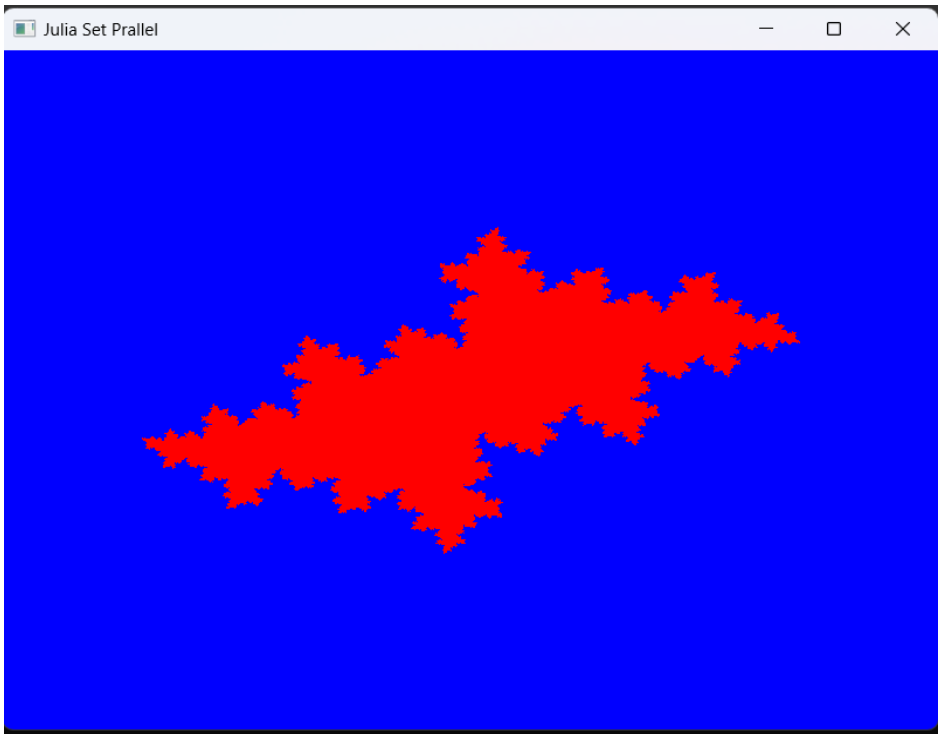
در این سوال، 2 تابع محاسبه گر(سریال و موازی) و 2 تابع نمایشگر(سریال و موازی) نوشتیم، که تابع محاسبه گر بر روی یک board دو بعدی مشخص می کنند که کدام اعداد بعد از به توان رسیدن به تعداد maxIterations واگرا شده اند و کدام همگرا، سپس، تابع نمایشگر این نقاط را با رنگ آبی و قرمز، مطابق با صورت پروژه نمایش می دهد. ابتدا ما برای تابع نمایشگر به صورت پیش فرض، یعنی Thread 8 ، داشتیم که چون پردازش بسیار کمی داشت، سربار مدیریت، ایجاد و هماهنگی این thread 8 باعث کمتر شدن تسریع موازی از سریال می شد که به همین منظور num_threads(2) قرار دادیم که باز هم موجب افزایش سرعت نشد ولی وضعیت بهتری از حالت قبل داشت. چون مشخص نیست که هر عدد بعد از چند بار به توان رسیدن، احتمالاً واگرا خواهد شد، از schedule(dynamic) استفاده کردیم.

Julia Set

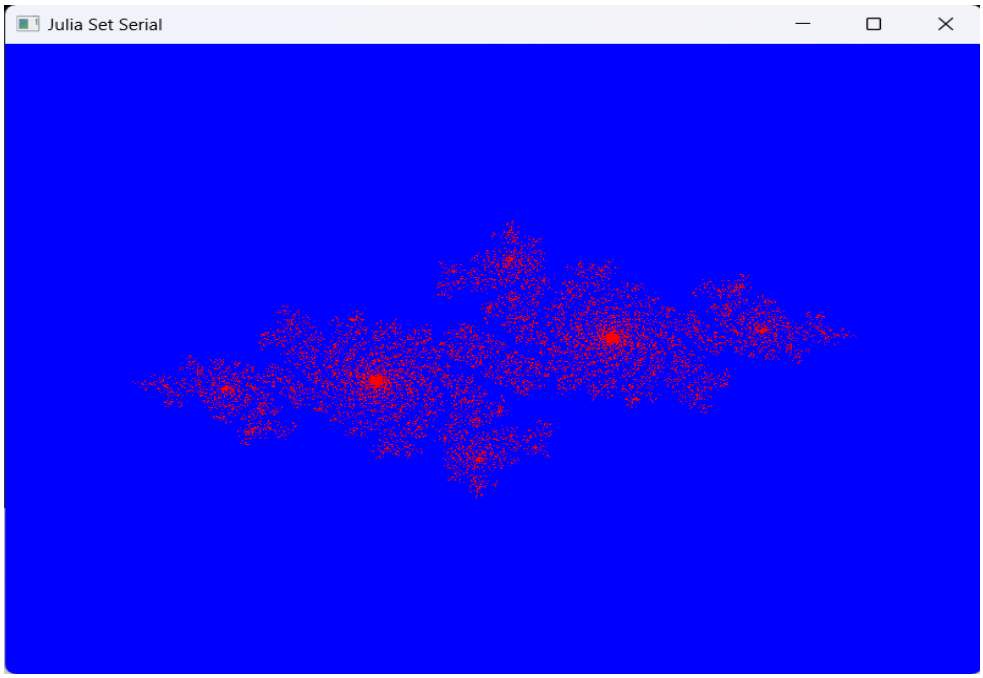
ابتدا به مشاهده خروجی ها و میزان تسریع می پردازیم:



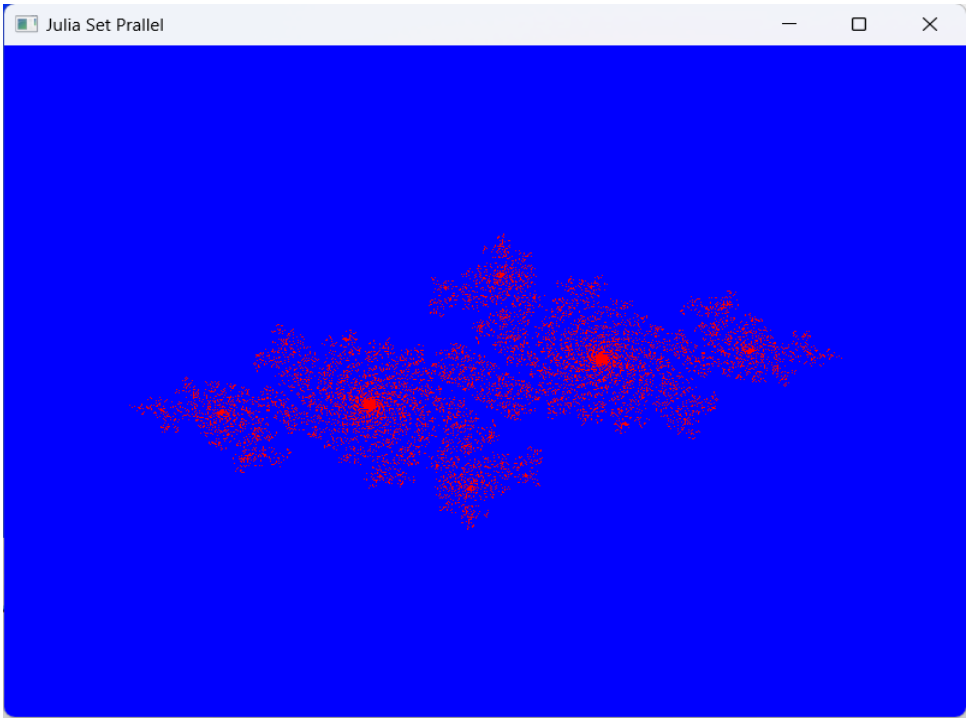
عکس 7: خروجی Julia سریال، برای c=(-0.5, 0.5j)



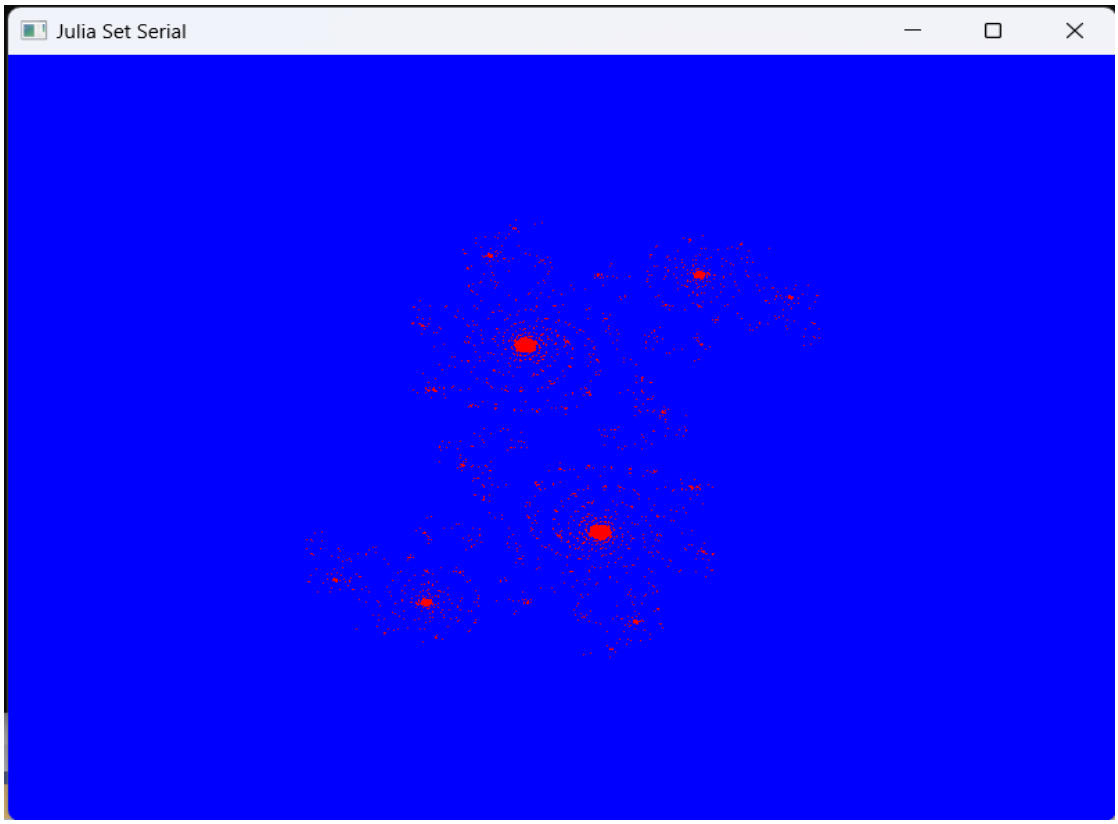
عکس 8: خروجی Julia موازی، برای c=(-0.5, 0.5j)



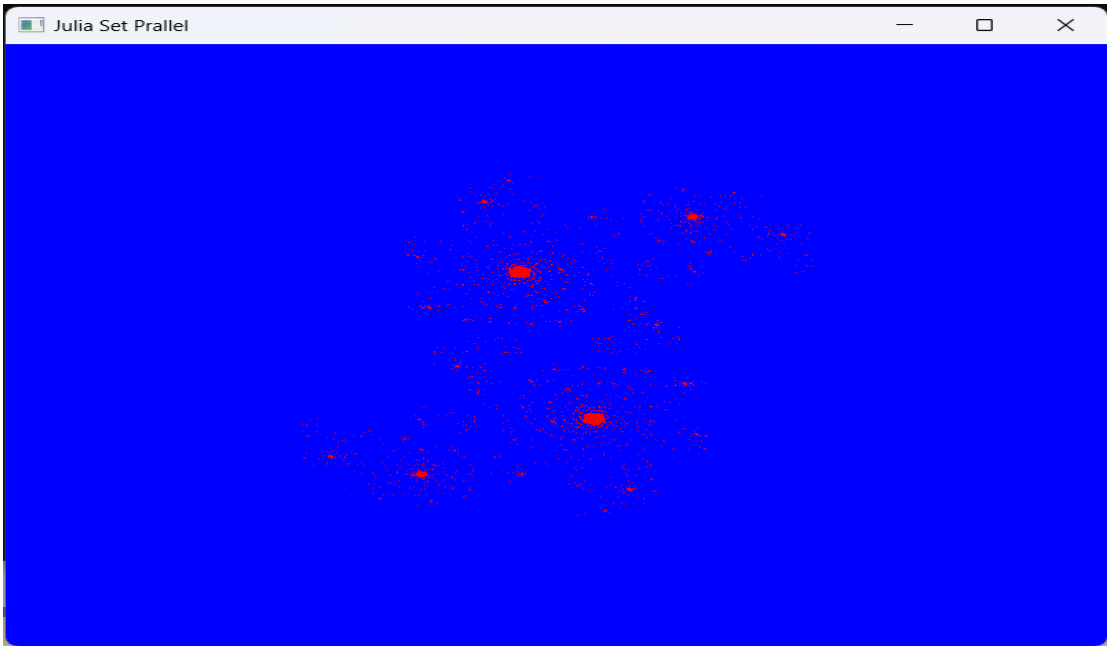
عکس 9: خروجی Julia سریال، برای c=(-0.7, 0.27015j)



عکس 10: خروجی Julia موازی، برای c=(-0.7, 0.27015j)



عکس 11: خروجی Julia سریال، برای c=(0.355, 0.355j)



عکس 12: خروجی Julia موازی، برای c=(0.355, 0.355j)

```
Serial Calculation Run time = 26788063677 cycles
Parallel Calculation Run time = 7594628796 cycles
Calculation Speedup = 3.52724

Serial Rendering Run time = 649903586 cycles
Parallel Rendering Run time = 767271846 cycles
Rendering Speedup = 0.847032

Total Speedup = 3.28131
```

عکس 13: میزان تسریع Julia

در این سوال هم، 2 تابع محاسبه گر(سریال و موازی) و 2 تابع نمایشگر(سریال و موازی) نوشتیم، مابقی کد هم مشابه بخش اول می باشد. برای خروجی در c های متفاوت، نیاز بود تا مقدار maxIterations را عوض کنیم تا شبیه به شکل های صورت پروژه باشد.

Monte Carlo

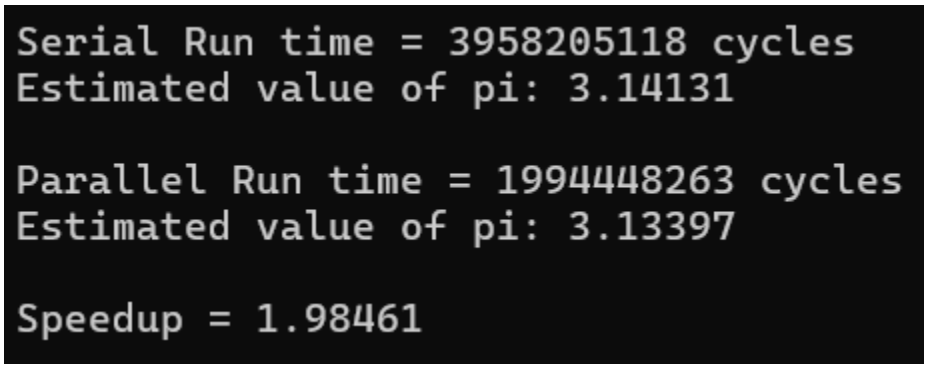
ابتدا به مشاهده خروجی ها و میزان تسریع می پردازیم:

```
Serial Run time = 31106101 cycles
Estimated value of pi: 3.13544

Parallel Run time = 29416094 cycles
Estimated value of pi: 3.12608

Speedup = 1.05745
```

عکس 14: خروجی و میزان تسریع وقتی که 50000 عدد تصادفی داریم

```
Serial Run time = 422859447 cycles
Estimated value of pi: 3.14302

Parallel Run time = 211461786 cycles
Estimated value of pi: 3.13554

Speedup = 1.9997
```

عکس 15: خروجی و میزان تسریع وقتی که 100000 عدد تصادفی داریم



عکس 16: خروجی و میزان تسریع وقتی که 1000000 عدد تصادفی داریم

برای این سوال حلقه را با استفاده از OpenMP موازی کردیم و برای اینکه تعداد نقطه هایی که داخل دایره قرار می گیرد درست محاسبه شود، از directive، reduction(+:points_inside_circle) استفاده شد. وقتی که تعداد اعداد تصادفی کمی(50000) داریم، هم تقریب مناسبی برای عدد π نداریم و هم چون محاسبات کم است، سربار مدیریت، ایجاد و هماهنگی Thread ها باعث کم بودن میزان تسریع می شود.