Съдържание

1. Цел на проекта

Целта на проекта е реализация на паралелен алгоритъм, генериращ множеството на Манделброт (Mandelbrot Set) от комплексни числа по формулата:

$$F(Z) = C * e^{-Z} + Z^2$$

Множеството се генерира от комплексните числа “Z” и “C”, където “C” е предварително зададена константа.

Изисквания към програмата:

- Работата по търсенето на точките от множеството на Манделброт трябва да стане чрез използването на паралелни процеси.
- Програмата трябва да генерира като изход файл, във формат “png” и визуализиращ намереното множество на Манделброт.
- Програмата трябва да извежда подходящи съобщения на различните етапи от работата:
  - Стартиране на даден процес;
  - Приключване на даден процес;
  - Време на изпълнение на даден процес в милисекунди;
  - Брой на използваните процеси;
  - Време на изпълнение на цялата програма в милисекунди.
- Програмата трябва да позволява командни параметри, които задават:
  - Големината на генерираното изображение;
  - Частта от комплексната равнина, в която се търси визуализацията на множеството;
  - Брой процеси, които да генерират множеството;
  - Име на изходния файл;
  - Тих режим на работата на програмата - извежда се само времето през което програмата е работила.

2. Описание на алгоритъма

Програмата използва Escape Time алгоритъма, чрез които се генерира множеството на Манделброт. Всеки пиксел, където x ∈ [0, imageWidth] и y ∈ [0, imageHeight], се взима и се изобразява в точка “C” от комплексната равнина, след което с тази точка програмата итерира по зададената формула (Z_0 = Complex(0.0,0.0)): $F(Z) = C * e^{-Z} + Z^2$, докато поредното Z_n не надхвърли 2 по абсолютна стойност, т.е. “C” излиза от множеството на Манделброт (escape condition) или не се достигне максималният брой предварително зададени итерации, в този случай считаме, че точката “C” принадлежи на множеството. На този принцип се анализират всички пиксели и се определя дали са част от множеството на Манделброт или са извън него. Спрямо броят итерации (т.е. спрямо каква е дистанцията на точката от множеството) на отделните пиксели им се асоциира съответен цвят.

3. Проектиране и реализация

Алгоритъмът е имплементиран на C++11. Реализиран е паралелизъм по данни, (Single Program-Multiple Data, SPMD), като нишките работят независимо една от друга. Архитектурата е по модела Master-Slaves, където ролята на Master играе основната програма.

При стартирането на програмата, спрямо избраният брой на процесите, които ще генерират множеството на Манделброт и избраната грануларност се изчисляват задачите и се разпределят поравно за всеки процес. Основната програма се грижи за стартирането на отделните процеси и изчакването на тяхното завършване. След успешно завършване на изчисленията, основната програма взима информацията за пикселите и генерира изображението.

Процесите използват два общи ресурса - thread_tasks и collected_data. В thread_tasks се съхраняват приготвените задачи за

отделните процеси, като всяка нишка, спрямо съответният й номер, има достъп до своите задачи, а в collected_data се записва генерираната информация за пикселите.

Функцията FindMandelbrot(int threadNum, int maxIterations) се пуска като задача на всеки процес и се грижи за изчисляването на зададените му предварително редове от изображението.

Функциите ConvertXToReal(int x, int imageWidth, double minR, double maxR) и ConvertYToImaginary(int y, int imageHeight, double minI, double maxI) се използват за изобразяване на пикселите в точка “C” от комплексната равнина.

Разпределение на задачите спрямо грануларността и броят на процесите:

Една задача, със зададена най-финна грануларност, представлява 1 ред за изчисление на изображението, като броят на редовете за изчисление в една задача, със зададена най-едрата грануларност, е imageHeight / numThreads.

Балансирането е статично, предварително е фиксиран броят задачи за всеки процес, като тоталният брой на задачите се изчислява по формулата:

tasksCount = granularity * numThreads

Като пример ще вземем генериране на изображение с размерност 1920x1080. Когато granularity = 1 (най-едрата грануларност) и numThreads = 32, всеки процес ще има да изчисли по ~34 реда на задача, като броят на задачите за всеки процес е 1 бр., тотално 32 задачи, или по формулата tasksCount = 1 * 32 = 32.

Нека имаме същото изображение, което искаме да генерираме отново с 32 нишки, но с грануларност 10. Тогава всеки процес ще има по ~4 реда за изчисление в една задача и ще има 10 задачи. Тотално това прави 320 задачи.

4. Стартиране на програмата

Пример за компилиране на програмата:

```
g++ main.cpp -o main -lpthread -std=c++11
```

Пример за стартиране на програмата[1]:

```
./main
```

Настройки по подразбиране:

- imageWidth x imageHeight - 1920x1080;
- maxIterations = 255;
- granularity = 1;
- numThreads = 1;
- minR = -2.5;
- maxR = 1.0;
- minI = -2.0;
- maxI = 2.0.

[1]По този начин програмата се стартира с настройките по подразбиране.

5. Тестове и резултати

Тестовете са направени със следните настройки:

- imageWidth x imageHeight - 1920x1080;
- maxIterations = 255;
- minR = -2.5;
- maxR = 1.0;
- minI = -2.0;
- maxI = 2.0.

Тестовете са направени върху тестовия сървър t5600.rmi.yaht.net:

- Два процесора Intel Xeon E5-2660 @ 2.20GHz с по 8 физически ядра и по 2 нишки на ядро;

❖ Общо 16 физически ядра и 32 нишки.

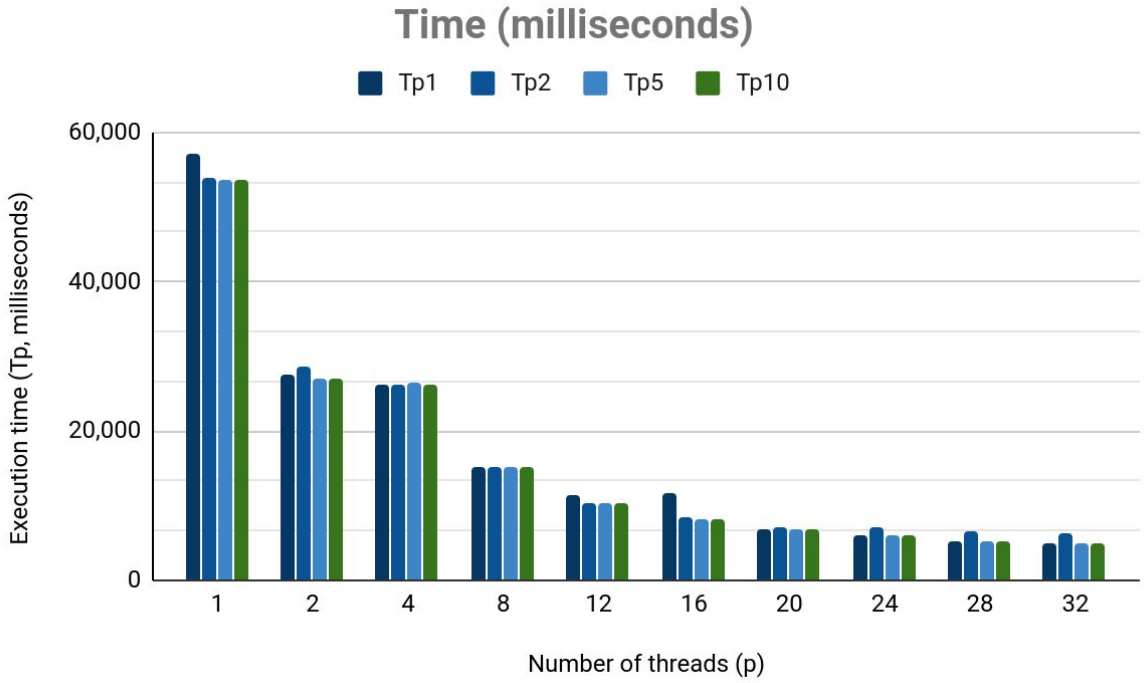
Параметрите numThreads и granularity варират. Таблиците съдържат информация за тестовете с 4 типа грануларност, където Tp1, Tp2, Tp5 и Tp10 са резултатите за минималното време за изпълнение (измерени в милисекунди) при коефициент на грануларност съответно 1, 2, 5 и 10. Аналогично е значението на Sp1, Sp2, Sp5 и Sp10, отразяващи ускорението спрямо различната грануларност и Ep1, Ep2, Ep5, Ep10, отразяващи ефективността спрямо различната грануларност.

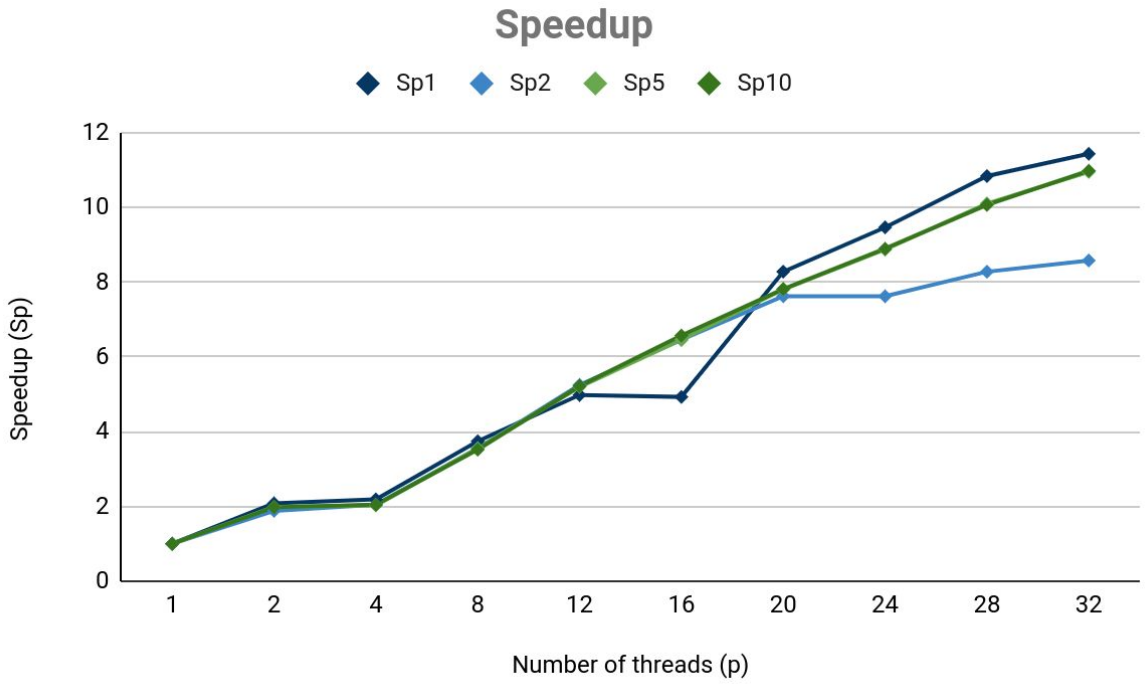
Използвани формули за изчисление на стойностите:

- ❖ Sp = T1 / Tp
- ❖ Ep = Sp / p
- ❖ p - брой на ядрата;
- ❖ g - коефициент на грануларност;
- ❖ Tp(n) - номер на тест за p ядра, например Tp(3) е резултат от тест 3 в милисекунди.

Таблица с изчисленото ускорение

p	Tp1	Tp2	Tp5	Tp10	Sp1	Sp2	Sp5	Sp10
1	57,223	53,793	53,568	53,541	1	1	1	1
2	27,493	28,601	26,943	27,082	2.081	1.881	1.988	1.977
4	26,161	26,250	26,429	26,177	2.187	2.049	2.027	2.045
8	15,290	15,133	15,079	15,229	3.743	3.555	3.552	3.516
12	11,497	10,255	10,313	10,262	4.977	5.246	5.194	5.217
16	11,613	8,331	8,299	8,150	4.927	6.457	6.455	6.569
20	6,914	7,059	6,866	6,852	8.276	7.620	7.802	7.814
24	6,048	7,061	6,040	6,022	9.461	7.618	8.869	8.891
28	5,282	6,501	5,304	5,321	10.834	8.275	10.100	10.062
32	5,007	6,273	4,894	4,878	11.429	8.575	10.946	10.976


Time (milliseconds)
Tp1
Tp2
Tp5
Tp10
60,000
40,000
20,000
0
Execution time (Tp, milliseconds)
1
2
4
8
12
16
20
24
28
32
Number of threads (p)


Speedup
Sp1
Sp2
Sp5
Sp10
12
10
8
6
4
2
0
Speedup (Sp)
1
2
4
8
12
16
20
24
28
32
Number of threads (p)

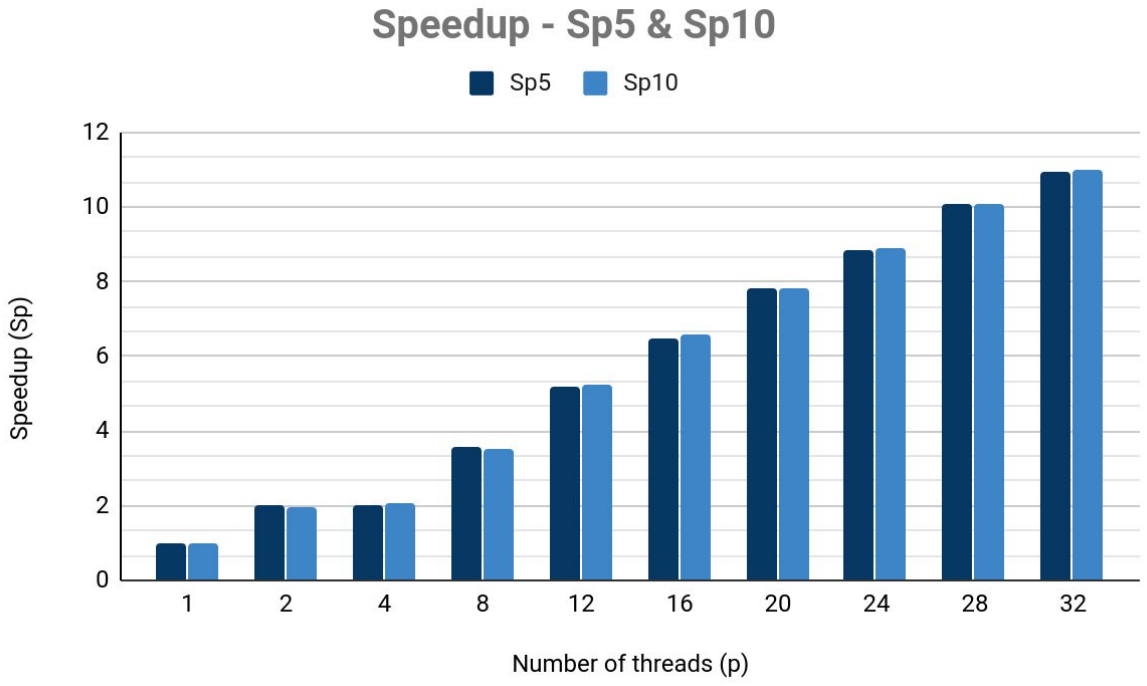


Таблица с изчислената ефективност

p	Tp1	Tp2	Tp5	Tp10	Ep1	Ep2	Ep5	Ep10
1	57,223	53,793	53,568	53,541	1	1	1	1
2	27,493	28,601	26,943	27,082	0.766	0.910	1.322	0.992
4	26,161	26,250	26,429	26,177	0.481	0.389	0.676	0.509
8	15,290	15,133	15,079	15,229	0.461	0.448	0.586	0.437
12	11,497	10,255	10,313	10,262	0.397	0.404	0.578	0.436
16	11,613	8,331	8,299	8,150	0.285	0.404	0.537	0.405
20	6,914	7,059	6,866	6,852	0.287	0.367	0.519	0.387
24	6,048	7,061	6,040	6,022	0.401	0.308	0.477	0.371
28	5,282	6,501	5,304	5,321	0.388	0.283	0.480	0.359
32	5,007	6,273	4,894	4,878	0.360	0.259	0.452	0.343

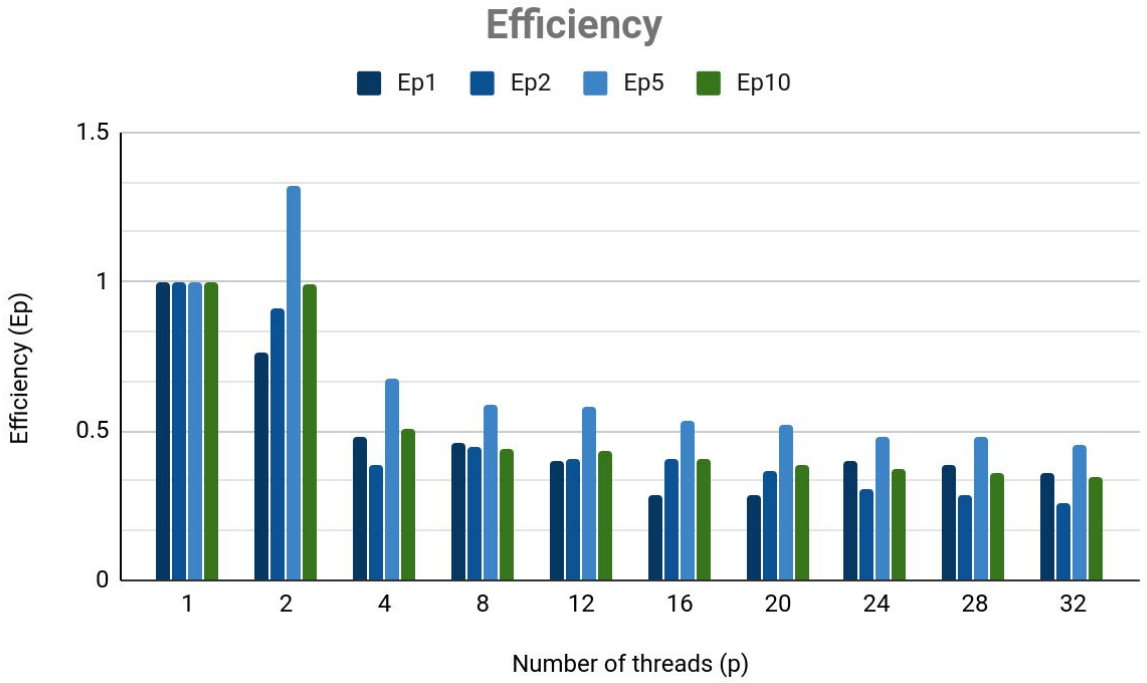


Таблица с пълните тестове

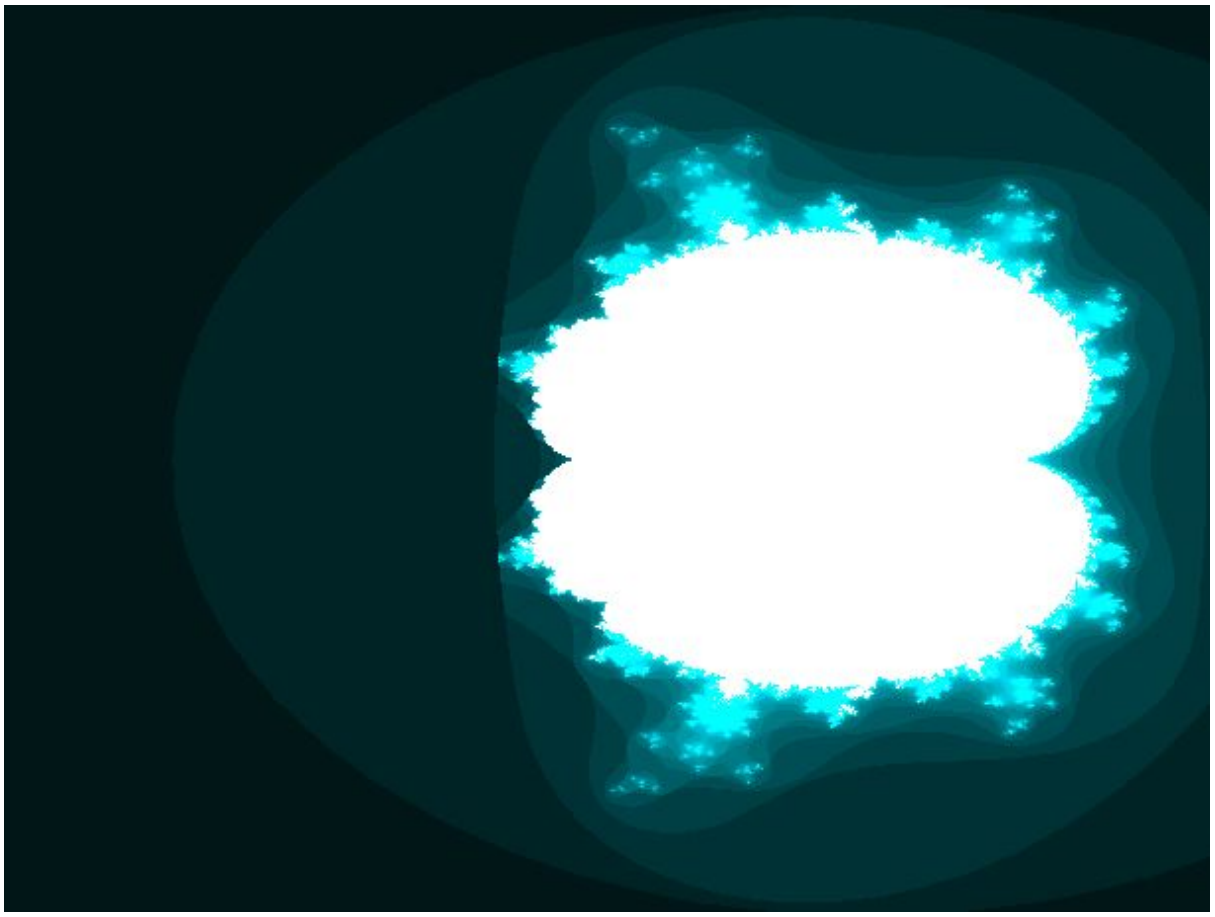
p	g	Tp(1)	Tp(2)	Tp(3)	Tp(4)	Tp(5)	min(Tp)	Sp	Ep
1	1	57,742	58,119	57,223	58,082	62,564	57,223	1	1
2	1	41,357	44,858	50,257	27,493	27,820	27,493	2.081	1.041
4	1	26,884	26,753	26,161	29,355	43,491	26,161	2.187	0.547
8	1	17,088	15,290	15,527	15,909	15,799	15,290	3.743	0.468
12	1	11,958	11,497	12,321	12,146	13,807	11,497	4.977	0.415
16	1	11,613	12,644	12,410	13,977	13,696	11,613	4.927	0.308
20	1	15,066	15,167	6,914	7,006	6,960	6,914	8.276	0.414
24	1	6,112	6,048	6,110	6,071	6,161	6,048	9.461	0.394
28	1	5,282	5,470	5,465	5,431	5,365	5,282	10.834	0.387
32	1	5,121	5,007	5,205	5,123	5,071	5,007	11.429	0.357
1	2	54,705	54,340	53,793	55,614	54,036	53,793	1	1
2	2	28,601	28,694	29,256	29,336	33,844	28,601	1.881	0.940
4	2	38,054	39,454	44,821	26,250	26,426	26,250	2.049	0.512
8	2	15,281	15,239	15,133	15,144	15,269	15,133	3.555	0.444
12	2	14,821	10,332	10,389	10,255	10,425	10,255	5.246	0.437
16	2	8,331	8,453	8,354	8,566	8,470	8,331	6.457	0.404

20	2	7,109	7,059	7,191	7,768	8,008	7,059	7.620	0.381
24	2	7,061	7,167	7,618	7,744	7,284	7,061	7.618	0.317
28	2	6,701	6,899	6,501	7,037	7,274	6,501	8.275	0.296
32	2	6,273	6,503	6,440	6,657	7,042	6,273	8.575	0.268
1	5	81,167	80,149	90,724	53,932	53,568	53,568	1	1
2	5	27,516	26,943	27,265	27,107	27,154	26,943	1.988	0.994
4	5	26,482	26,623	26,429	26,875	26,642	26,429	2.027	0.507
8	5	15,409	15,533	15,210	15,456	15,079	15,079	3.552	0.444
12	5	10,318	10,445	10,376	10,313	10,368	10,313	5.194	0.433
16	5	8,299	8,301	8,321	8,407	8,527	8,299	6.455	0.403
20	5	6,866	6,924	6,906	6,918	7,015	6,866	7.802	0.390
24	5	6,250	6,964	6,090	6,050	6,040	6,040	8.869	0.370
28	5	5,308	5,333	5,379	5,304	5,441	5,304	10.100	0.361
32	5	5,081	4,894	5,010	4,940	4,956	4,894	10.946	0.342
1	10	53,541	54,956	54,032	54,195	53,959	53,541	1	1
2	10	27,251	27,255	27,082	27,219	27,640	27,082	1.977	0.988
4	10	26,474	26,177	27,019	26,560	26,588	26,177	2.045	0.511
8	10	15,252	15,229	16,002	15,516	15,411	15,229	3.516	0.439
12	10	10,262	10,374	10,309	10,385	10,427	10,262	5.217	0.435
16	10	8,150	8,419	8,472	8,459	8,319	8,150	6.569	0.411
20	10	6,879	6,852	7,013	7,193	6,993	6,852	7.814	0.391
24	10	6,059	6,022	6,167	6,103	6,058	6,022	8.891	0.370
28	10	5,348	5,460	5,321	5,331	5,448	5,321	10.062	0.359
32	10	4,928	4,938	5,002	4,913	4,878	4,878	10.976	0.343

От получените резултати най-доброто ускорение е достигнато при 32 нишки с коефициент на грануларност 1 (11,429), докато най-добрата ефективност е достигната при 2 нишки с коефициент на грануларност 5 (1,322).

При анализ на получените резултати от сходни програми с нарастването на броя нишки, ефектът от грануларността трябва да се усили, т.е. при 32 нишки с най-финна грануларност, ускорението и ефективността трябва да са най-добри.

За подобен тип програми, желаното за достигане ускорение е около 17-18.



6. Източници

[1] Mcanv, Escape Time Algorithm
https://mcanv.com/Art/escape_time_algorithm.html#:~:text=The%20%22escape%20time%22%20part%20of,and%20assigns%20it%20a%20color.

[2] Wikipedia, Mandelbrot set
https://en.wikipedia.org/wiki/Mandelbrot_set