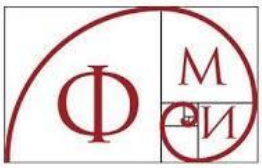


Софийски университет "Св. Климент Охридски"

Факултет по математика и информатика

КУРСОВ ПРОЕКТ

по

Системи за паралелна обработка

спец. Компютърни науки, 3 курс,

летен семестър,

2023/2024

MandelWorld

Паралелен тест на Манделброт при статично и динамично балансиране.

Грануларност и адаптивност към L1 D-Cache.

Калоян Цветков, 4MI0800017

Ръководители:

проф. д-р Васил Цунижев

ас. Христо Христов

Юни, 2024г.

Съдържание

Увод

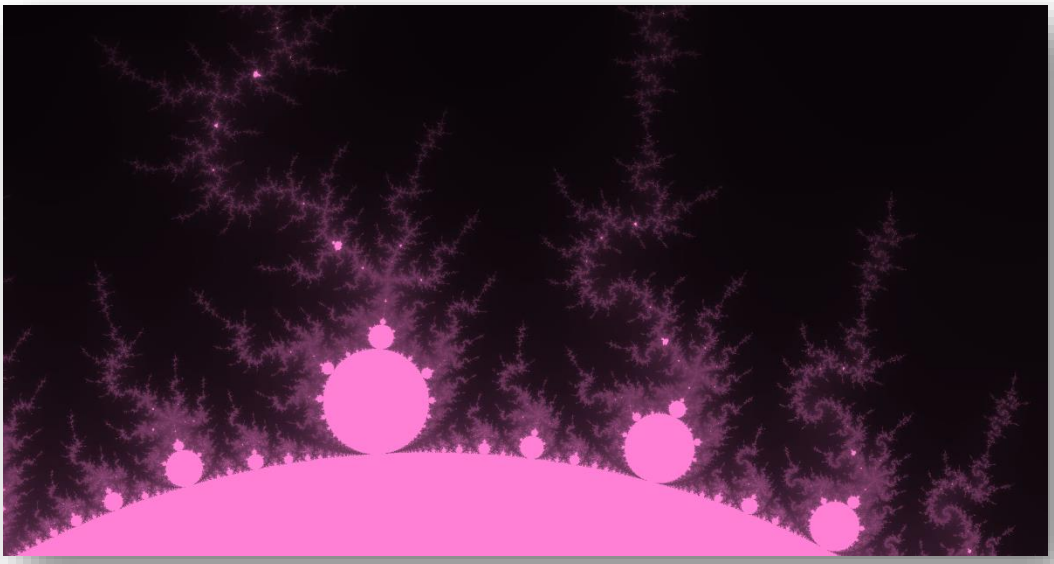
Този проект има за цел да изследва и реализира методи за паралелно генериране на множеството на Манделброт. Множеството на Манделброт е едно от най-известните фрактални множества и играе важна роля в математиката и компютърната графика. Паралелизацията на изчисленията е ключова за ефективното и бързо генериране на това множество, особено когато се работи с висока резолюция и големи мащаби.

Дефиниция на множеството на Манделброт

За дадено комплексно число $c \in C$, започваме с $z_0 = 0$ и образуваме редицата $z_{n+1} = z_n^2 + c$. Множеството на Манделброт се състои от всички точки $c \in C$ в комплексната равнина, за които редицата остава ограничена, тоест не отива към безкрайност. Тоест, множеството на Манделброт може да се дефинира както следва:

$$M = \{c \in \mathbb{C} \mid \exists A \in \mathbb{R}\ \forall n \in \mathbb{N}\ (|z_n| \leq A)\}$$

където $z_0 = 0$ и $z_{n+1} = z_n^2 + c$.



Фигура 1: *Множеството на Манделброт в граници* $x \in \{-1.5; -1.2\},\ y \in \{0.71, 0.73\}$

Защо има нужда от паралелизация

Изчисляването на множеството на Манделброт изисква голям брой итерации за всяка точка в комплексната равнина, особено при висока резолюция. Тези изчисления са независими едно от друго и могат да се изпълняват паралелно, което значително ускорява процеса. Паралелизацията позволява ефективното използване на многопроцесорни системи и графични процесори (GPU), като по този начин се намалява времето за генериране на изображението и се постига по-добра производителност. Това е особено важно за приложения, които изискват визуализация в реално време или изследване на фрактали с много висока детайлност. Този проект ще реализриа SIMD концепцията (Single Instruction, Multiple Data).

Влияние на L1 D-Cache

L1 D-Cache (L1 Data Cache) играе съществена роля в оптимизацията на производителността на алгоритмите за генериране на множеството на Манделброт. Кеш паметта от ниво 1 е най-бързата и най-близко разположената до процесора памет, което я прави критична за бързото извличане и записване на данни. Ефективното използване на ***L1 D-Cache*** може значително да намали времето за достъп до паметта и да увеличи скоростта на изпълнение на паралелните изчисления.

Структура на проекта

Една от основните цели на този проект е да предостави оптимални имплементации на динамично и статично балансиране, за да се сравнят и проверят теоретичните хипотези. Тези хипотези включват твърдението, че по-доброто балансиране на задачите при динамичното балансиране не компенсира системния свръхтовар, който произтича от естеството на динамичното балансиране.

За целта са разработени две имплементации на алгоритъма за генериране на множеството на Манделброт, различаващи се по методите за балансиране. Първата програма имплементира статично балансиране, а втората динамично централизирано балансиране с използване на ***Master-Slave*** модела.

Ще направим разделяне на матрицата на ленти, представляващи съвкупност от съседни редове. Всички имплементации използват споделена памет като начин на обмен. Програмите са имплементирани на езика C++17. За създаването на нишките във всяка от програмите необходимата логика е обособена във функция и е използван класа std::thread. По следния начин се създават и стартират нишките в програмата, реализираща статично балансиране:

```
for (int i = 0; i < NUM_THREADS; ++i)
{
    std::vector<int> startY, endY, startX, endX;
    for (int j = 0; j < GRANULARITY; ++j)
    {
        startY.push_back((j * NUM_THREADS + i) * chunkSizeY);
        endY.push_back((j * NUM_THREADS + i + 1) * chunkSizeY);
    }
    threads.emplace_back(renderTask,
                         std::ref(pixels),
                         i,
                         startY, endY);
}
```

Същинската работа, която нишката получава, е множество подзадачи за изчисление на ивици от матрицата pixels:

```
void renderTask(std::vector<unsigned char> &pixels,
                int number,
                const std::vector<int> &startY,
                const std::vector<int> &endY)
{

    auto startTime = std::chrono::steady_clock::now();
    for (int i = 0; i < startX.size(); ++i)
    {
        renderChunk(pixels, number, startY[i], endY[i]);
    }
    auto endTime = std::chrono::steady_clock::now();
    std::chrono::duration<double> duration = endTime - startTime;
    std::clog << "Thread " << number << " took "
              << duration.count() << " seconds.\n";
}
```

Main нишката изчаква останалите нишки да приключат работа преди визуализацията:

```
for (auto &thread : threads)
    thread.join();
```

При програмата, реализираща динамично балансиране, е използвана библиотеката MPI. Процесът Master изпраща задачи и получава отговори към Slave процесите със следния код:

Първоначално раздаване на задачи:

```
for (int rank = 1; rank < num_procs; ++rank)
{
    MPI_Send(&next_task, 1, MPI_INT, rank, 0, MPI_COMM_WORLD);
    next_task+=5;
}
```

Получаване на резултат и изпращане на нова задача:

```
for (int task = 0; task < HEIGHT/5; ++task)
{
    int result[2];
    MPI_Recv(result, 2, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    int rank = status.MPI_SOURCE;
    image[result[0]] = result[1];
    if (next_task < HEIGHT)
    {
        MPI_Send(&next_task, 1, MPI_INT, rank, 0, MPI_COMM_WORLD);
        next_task = min(next_task+5, HEIGHT);
    }
    else
    {
        int done_signal = -1;
        MPI_Send(&done_signal, 1, MPI_INT, rank, 0, MPI_COMM_WORLD);
    }
}
```

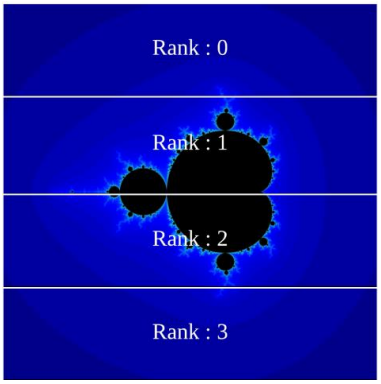
За работа с комплексни числа е използвана стандартната библиотека **<complex>**.

Анализ на различните имплементации:

Паралелното генериране на множеството на Манделброт може да бъде реализирано чрез различни методи за разпределение на задачите между нишките. По-долу са описани три основни подхода: делене на блокове, динамично балансиране и статично циклично балансиране. Всеки от тези подходи има своите предимства и недостатъци, които влияят на ефективността и скоростта на изчисленията при различни стойности на паралелизация.

Делене на блокове

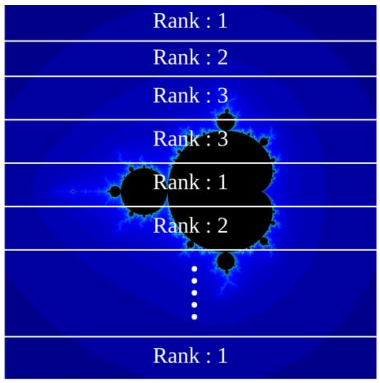
При паралелизация с две нишки (p = 2) резултатите са близки до теоретичния лимит, защото изображението се разделя на две симетрични части. Това позволява равномерно разпределение на работата между нишките. При увеличаване на броя на нишките, ускорението намалява, защото блоковете вече не са симетрични и задачите не са равномерно разпределени между нишките. В допълнение, някои блокове може да съдържат повече точки, изискващи повече итерации, което води до неравномерно натоварване и намаляване на ефективността. Освен това, този подход може да доведе до изчерпване на ресурси при голям брой нишки, което допълнително намалява производителността.



Фигура 2: *Схема на рапределяне на задачите при делене на блокове*

Динамично балансиране

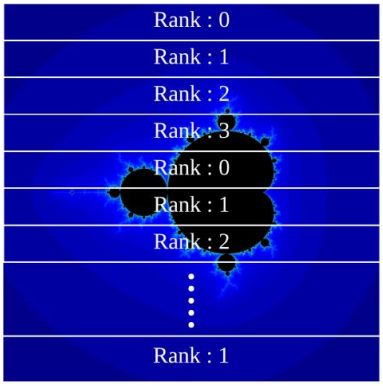
Mater нитката раздава задачи на принципа First come, first served. При паралелизация с две нишки (p = 2) не се постига значително ускорение, защото главната нишка просто предава задачата на единствената второстепенна нишка. С увеличаване на броя на нишките, подялбата на работата става по-справедлива и при 32 нишки ускорението е значително по-добро от това при статично циклично балансиране. Причината е, че комуникационният свръхтовар се компенсира от по-ефективното използване на процесорното време. Освен това, динамичното разпределение на задачите позволява на процесите, които приключат по-рано, да поемат нови задачи, което води до по-добро използване на наличните ресурси и по-голяма гъвкавост при управлението на натоварването.



Фигура 3: *Схема на рапределяне на задачите при динамично балансиране*

Статично циклично балансиране

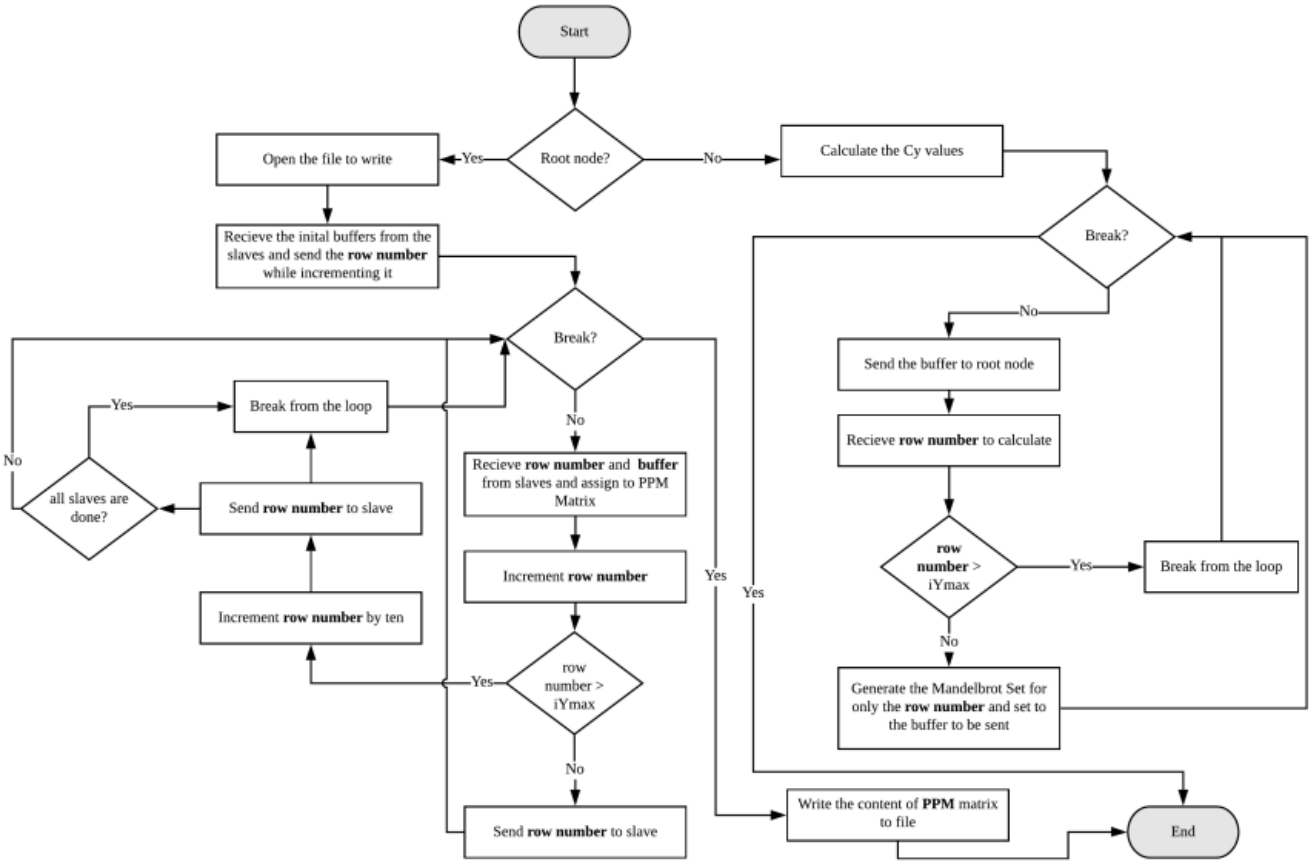
Този подход осигурява консистентно добро ускорение при всички стойности на паралелизация, защото се осигурява ефективно балансиране с минимизиран свръхтовар. Всеки процес получава равномерно разпределени задачи, което гарантира, че всички процеси ще бъдат натоварени равномерно през целия период на изчисленията. Този подход е особено ефективен при системи с хомогенни процесори, където времето за изпълнение на всяка задача е приблизително еднакво. Недостатъкът е, че може да бъде по-труден за имплементация в хетерогенни среди, където процесорите имат различна производителност и капацитет.



Фигура 4: *Схема на рапределяне на задачите при статично циклично балансиране*

UML диаграми за различните подходи

Динамично балансиране (от [4])



Стартиране

За да се стартира приложението, реализиращо циклично балансиране, в shell се изпълнява:

./mandelbrot.exe

Програмата разпонава следните аргументи, подадени на вход:

Опция	Описание
-c	Задава областта от комплексната равнина, в която множеството да бъде генерирано. Форматът е x?y?r?, съответно център с координати (x,y) и радиус на квадрат r; част се отрязва в зависимост от резолюцията; по подразбиране x=-0.4, y=0, r=1
-t	Задава паралелизъм, т.е. броя нишки, на които да се извършват изчисленията; по подразбиране е 4
-g	Задава грануларност, т.е. броя задачи, които всяка нишка да решава
-o	Задава изходен файл, в който да бъде записано изображението в PPM формат; по подразбиране е зададена директория "../results/default.ppm"
-s	Задава резолюция на генерираното изображение; по подразбиране е 512x270
-v	Задава цвят на точките, принадлежащи на множеството във формат r?g?b?, където стойностите са в интервала [0;255]; по-подразбиране е зададено r255g128b213

След визуализация, потребителят има възможност да се „движи“ по комплекцната равнина, използвайки клавишите W,A,S,D от клавиатурата си. По-едра стъпка на движение може да се постигне чрез използване на стрелките от клавиатурата.

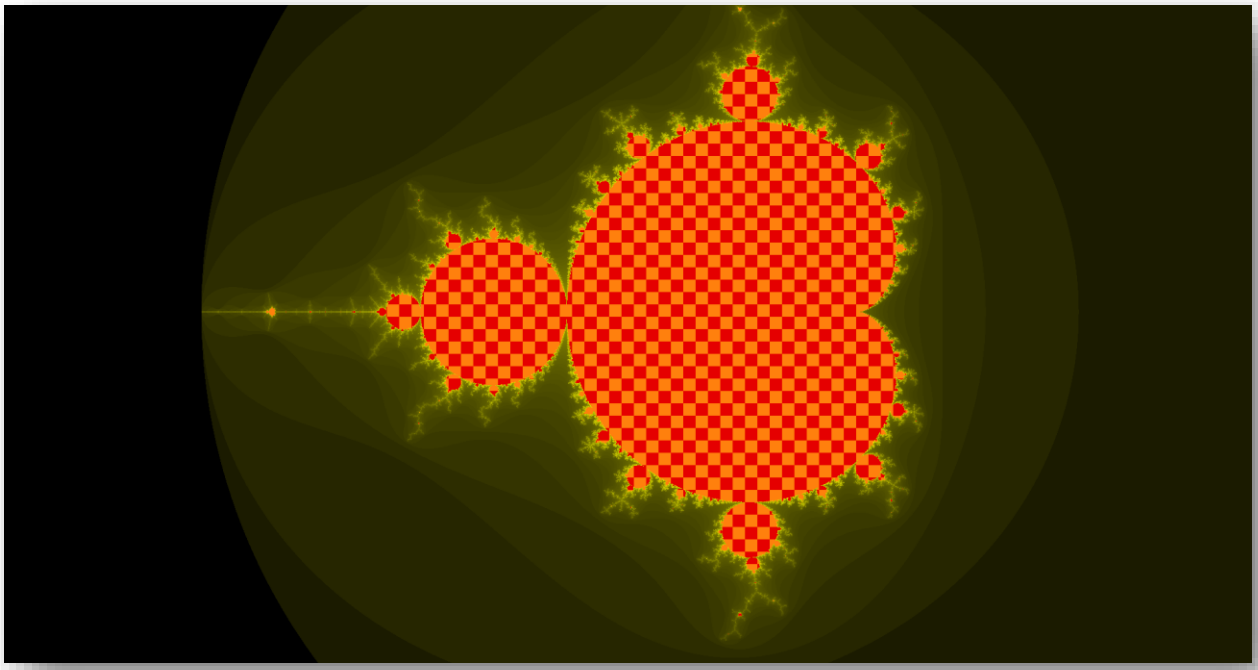
Програмата предоставя възможност и за приближаване и отдалечаване, съответно чрез използване на клавишите Z и X.

За да се стартира приложението, реализиращо динамично балансиране, в shell се изпълнява:

./Mandelbrot_mpi.exe

Тестване и настройка.

Всички тестови случаи са проведени при **10000** максимален брой итерации като параметър. Генерираното изображение е следното с резолюция **2048x1080** (**2K**):



Фигура 3: *Генерирано тестово изображение, 10000 итерации, по редове, област:* ***-1.4:0.6:-0.7:0.7****, размер на генерирано изображение:* ***2K***

Тестване и настройка на входните параметри

Тестовете са проведени на

Резултати при реализация със статично циклично балансиране

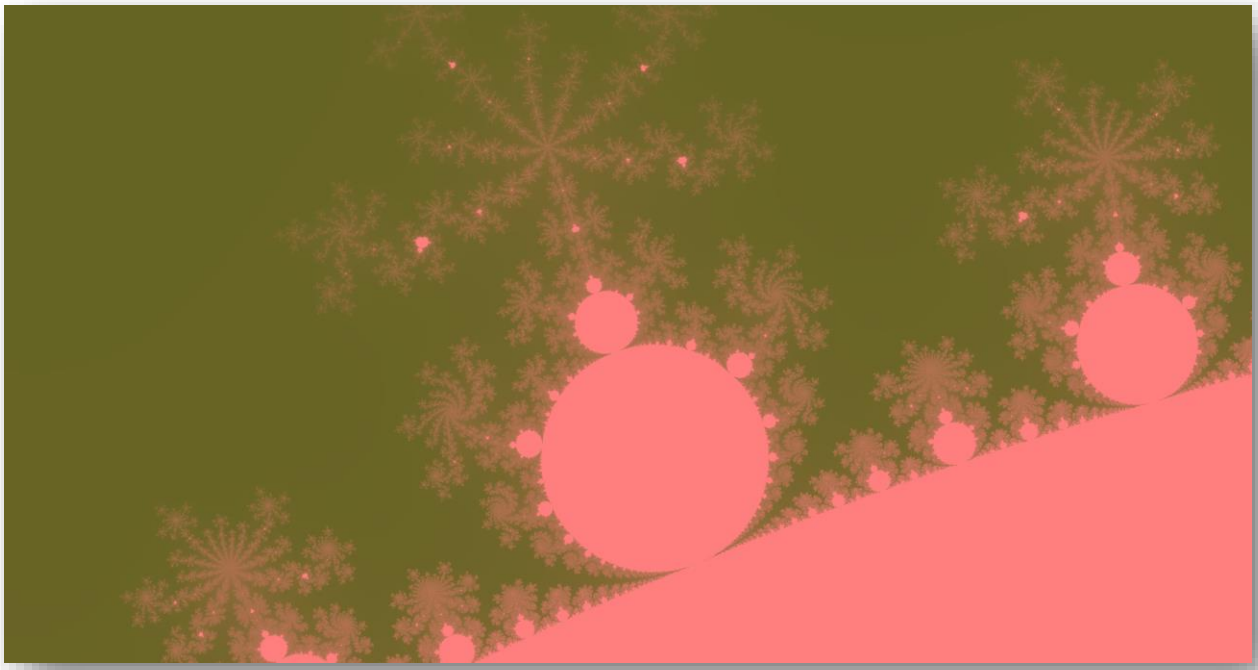
В представения тестов план под формата на таблица са използвани следните означения на колоните:

- p – паралелизъм (броят нишки)
- g – грануларност
- $Tp^{(i)}$ – време за изпълнение при паралелизъм p, получено на i-тия тест
- S_p – ускорение при паралелизъмр
- E_p – ефективност при паралелизъм

p	g	$T_p(1)$	$T_p(2)$	$T_p(3)$	$T_p(4)$	$min\, T_p(i)$	$S_p = \frac{T_1}{T_p}$	$E_p = \frac{S_p}{p}$
1	1	18.9848	18.9777	19.0044	18.9908	18.9777	1	1

2	1	9.49979	9.51945	9.5222	9.51234	9.49979	1.997697	0.998848
4	1	8.02845	8.01959	8.00047	8.113	8.00047	2.372073	0.593018
8	1	6.17173	6.21856	5.9983	6.07682	5.9983	3.163846	0.395481
16	1	5.51552	5.45996	5.42046	5.48528	5.42046	3.501124	0.21882
32	1	5.43994	5.41148	5.44172	5.50529	5.41148	3.506933	0.109592
1	2	18.9851	18.9715	18.9781	18.9731	18.9715	1	1
2	2	9.49816	9.50773	9.50175	9.48923	9.48923	1.999267	0.999633
4	2	5.36861	5.35933	5.52508	5.53053	5.35933	3.539901	0.884975
8	2	5.29307	5.30408	5.35845	5.2774	5.2774	3.594857	0.449357
16	2	5.29933	5.27374	5.34571	5.39629	5.27374	3.597352	0.224835
32	2	5.45263	5.35221	5.34527	5.41253	5.34527	3.549213	0.110913
1	4	19.011	19.0159	18.9821	19.0149	18.9821	1	1
2	4	9.48458	9.51378	9.51375	9.50706	9.48458	2.001364	1.000682
4	4	5.60042	5.55217	5.50423	5.66713	5.50423	3.448639	0.86216
8	4	5.32797	5.34923	5.37518	5.41939	5.32797	3.562727	0.445341
16	4	5.31309	5.4743	5.26671	5.31192	5.26671	3.604167	0.22526
32	4	5.29544	5.3499	5.33977	5.29254	5.29254	3.586577	0.112081
1	8	19.0206	19.0103	18.9946	19.0432	18.9946	1	1
2	8	9.51658	9.5193	9.51563	9.51365	9.51365	1.996563	0.998281
4	8	5.29137	5.31798	5.33339	5.32119	5.29137	3.589732	0.897433
8	8	5.30687	5.25086	5.42702	5.32512	5.25086	3.617426	0.452178
16	8	5.27795	5.29056	5.31064	5.28964	5.27795	3.598859	0.224929
32	8	5.4145	5.75173	5.50453	5.28688	5.28688	3.592781	0.112274

Фигура 4: *Графика на ускорението при статично циклично балансиране, 10000 итерации, по редове, област:* ***-1.4:0.6:-0.7:0.7****, размер на генерирано изображение:* ***2K***

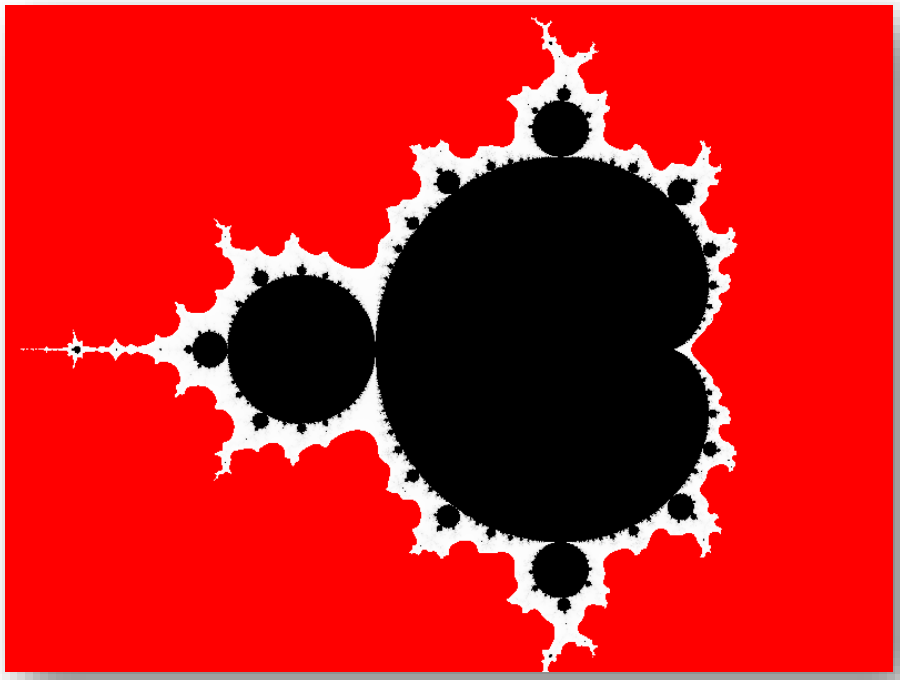


Фигура 3: *Генерирано тестово изображение, 10000 итерации, по редове, област:* ***--0.375:--0.245:0.64:0.685.****, размер на генерирано изображение:* ***2K***

p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	$T_p^{(4)}$	$minT_p^{(i)}$	$S_p = \frac{T_1}{T_p}$	$E_p = \frac{S_p}{p}$
1	1	23.608	23.0532	23.2526	23.5461	23.0532	1	1
2	1	22.749	22.58	22.5967	22.4382	22.4382	1.027409	0.513704
4	1	14.9977	14.8593	14.9148	15.2538	14.8593	1.551432	0.387858
8	1	9.53185	10.7318	10.0958	9.82861	9.53185	2.418544	0.302318
16	1	8.56961	8.56278	7.98473	8.77325	7.98473	2.887161	0.180448
32	1	8.54708	8.4324	8.11399	7.63142	7.63142	3.020827	0.094401
1	2	23.3843	23.3053	23.1614	23.3565	23.1614	1	1
2	2	15.6	15.5023	15.3149	15.3233	15.3149	1.512344	0.756172
4	2	9.81688	9.30263	10.0067	9.49314	9.30263	2.489769	0.622442
8	2	8.40554	8.57381	9.40845	9.11581	8.40554	2.755492	0.344437
16	2	8.5537	9.03196	8.97478	8.83072	8.5537	2.707764	0.169235
32	2	7.80806	7.68514	8.12891	7.95447	7.68514	3.01379	0.094181
1	4	23.2533	24.3563	23.317	23.2071	23.2071	1	1
2	4	14.2781	13.9781	14.2416	14.1597	13.9781	1.660247	0.830124
4	4	9.44493	9.43712	8.76388	9.54634	8.76388	2.648039	0.66201
8	4	8.77916	9.04246	9.0934	8.79496	8.77916	2.643431	0.330429
16	4	8.07237	8.37374	8.59877	8.54458	8.07237	2.874881	0.17968
32	4	8.40524	7.92108	7.71234	7.93077	7.71234	3.009087	0.094034

1	8	23.2414	23.3413	23.1844	23.2611	23.1844	1	1
2	8	13.5124	13.5361	13.4867	13.5867	13.4867	1.719057	0.859528
4	8	8.67023	8.39598	8.09321	9.61504	8.09321	2.864673	0.716168
8	8	9.08077	8.58912	8.23975	7.73062	7.73062	2.999035	0.374879
16	8	8.10317	8.16813	10.1077	9.47172	8.10317	2.861152	0.178822
32	8	7.86083	7.68595	7.64365	7.61636	7.61636	3.044026	0.095126
1	16	23.6633	23.1768	23.669	23.2632	23.1768	1	1
2	16	12.9061	12.2277	11.9253	12.2812	11.9253	1.719057	0.859528
4	16	6.97457	7.19602	7.0871	6.99146	6.97457	2.864673	0.716168
8	16	6.76396	6.94139	6.66059	6.87738	6.66059	2.999035	0.374879
16	16	6.84319	6.87748	6.75779	6.93549	6.75779	2.861152	0.178822
32	16	6.74971	6.82113	6.76531	6.81513	6.74971	3.044026	0.095126

Резултати при реализация със динамично балансиране



Фигура 3: *Генерирано тестово изображение, 10000 итерации, по редове, област-2****:-1.5:-1.4:1.6****, размер на генерирано изображение:* ***2K***

p	g	Tp	Sp	Ep
1	160	5.5476		
1	160	5.5187	1	1
1	160	5.6732		

1	160	5.6231		
2	84	4.3845	0.794481	0.39724
2	89	4.3913	0.795713	0.397856
2	86	4.3742	0.792614	0.396307
2	92	4.4022	0.797688	0.398844
4	46	4.0126	0.727092	0.181773
4	42	4.0708	0.737637	0.184409
4	51	3.9801	0.721202	0.180301
4	48	4.0342	0.731005	0.182751
8	23	3.2675	0.592078	0.07401
8	22	3.2323	0.5857	0.073212
8	28	3.2678	0.592132	0.074017
8	26	3.2901	0.596173	0.074522
16	11	3.367	0.610107	0.038132
16	13	3.3289	0.603204	0.0377
16	18	3.3187	0.601355	0.037585
16	12	3.3385	0.604943	0.037809

Източници:

[1]. S. Tornboulian. Indirect Addressing and load balancing for faster solution to Mandelbrot set on SIMD architectures, May, 1989

https://ntrs.nasa.gov/citations/19890018019

[2] Ali, Dia, Dobson, Davis, Gäng, Isaac, Gourd, Jean. Parallel Implementation and Analysys of Mandelbrot Set Construction. University of Southern Mississippi, 2008.

https://www.academia.edu/1399383/Parallel_Implementation_and_Analysis_of_Mandelbrot_Set_Construction

[3] Gomez, Erstesto S., Universidad de las Ciencias Inform'aticas, Cuba. 30.09.2020. MPI vs OpenMP: A case study on parallel generation of Mandelbrot set. Redalyc.

https://www.redalyc.org/journal/6738/673870835002/html/.

[4] Bhanuka, Manesha, Baskaran, Vishnu, Efficient Generation of Mandelbrot Set using Message Passing Interface, School of Information Technology, 2020.

https://arxiv.org/pdf/2007.00745

Заключение:

Паралелното генериране на множеството на Манделброт е важна задача, която позволява по-бързо и ефективно изследване на фрактални структури. Този проект предлага цялостно решение, включващо паралелизация, визуализация и интерактивна работа с множеството, като по този начин предоставя мощен инструмент за математици, компютърни графици и изследователи. Оптимизациите, включващи ефективно използване на L1 D-Cache, допълнително подобряват производителността и правят изчисленията по-ефективни.

Предоставените таблици и графики потвърждават предимството на статичното балансиране пред динамичното.