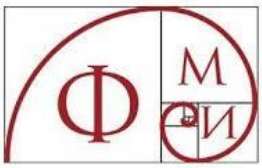


Софийски университет "Св. Климент Охридски"

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

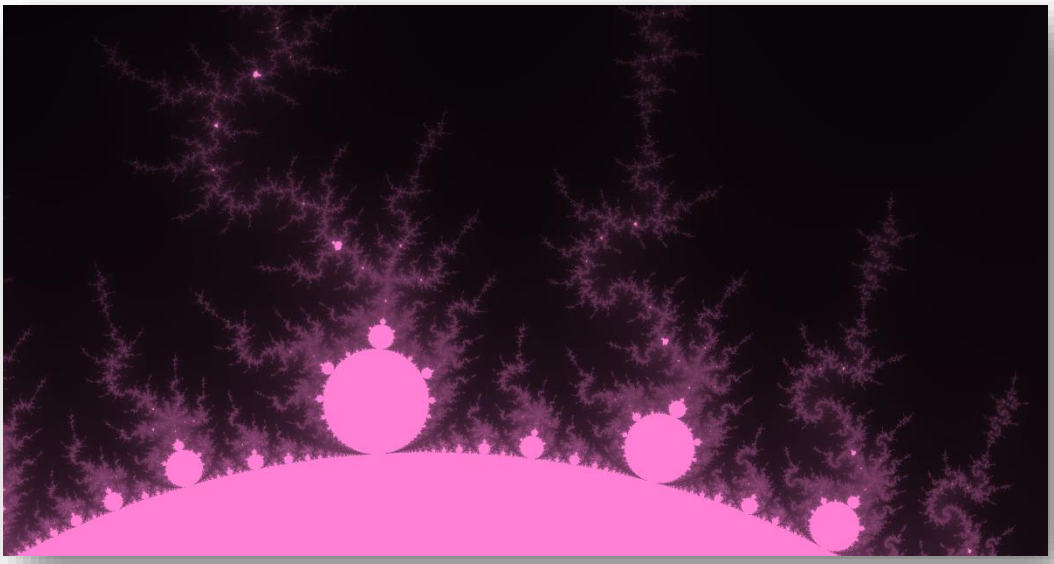
Този проект има за цел да изследва и реализира методи за паралелно генериране на множеството на Манделброт. Множеството на Манделброт е едно от най-известните фрактални множества и играе важна роля в математиката и компютърната графика. Паралелизацията на изчисленията е ключова за ефективното и бързо 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:

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

При динамичното балансиране main нишката е Master и създава останалите нишки, като им задава задача сами да си определят ивица, по която да работят. Main нишката изчаква останалите нишки да приключат работа преди визуализацията:

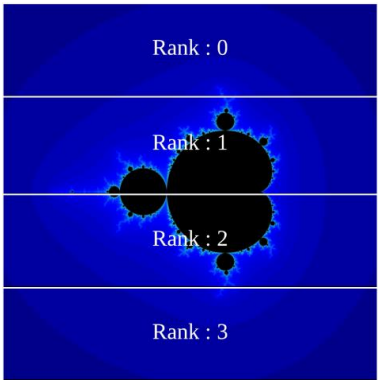
За работа с комплексни числа е използвана стандартната библиотека **<complex>**.

Анализ на различни 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 (източник за тях е [2]): 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

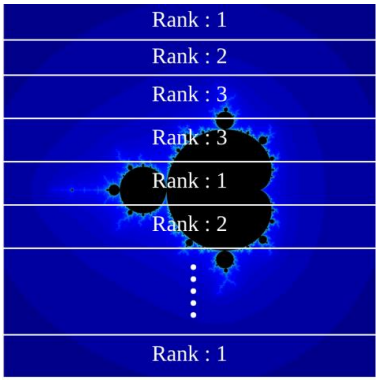
При паралелизация с две нишки (p = 2) резултатите са близки до теоретичния лимит, защото изображението се разделя на две симетрични части. Това позволява равномерно разпределение на работата между нишките. При увеличаване на броя на нишките, ускорението намалява, защото блоковете вече не са симетрични 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

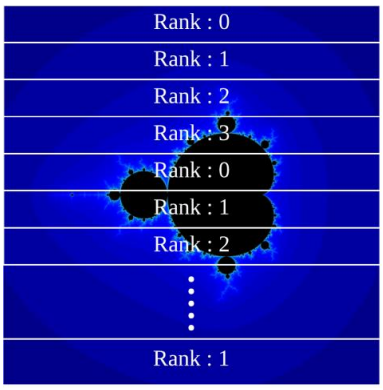
Master нишката създава p на брой нишки Slave, всяка от които поема първата свободна задача (ивица от 5 реда от матрицата на пикселите). Първата свободна задача се следи от std::atomic<int>, сочещ го. Реализиран е принципът First come, first served. При паралелизация с две нишки (p = 2) не се постига значително ускорение, защото главната нишка просто предава задачата на единствената второстепенна нишка. С увеличаване на броя на нишките, подялбата на работата става по-справедлива. 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*

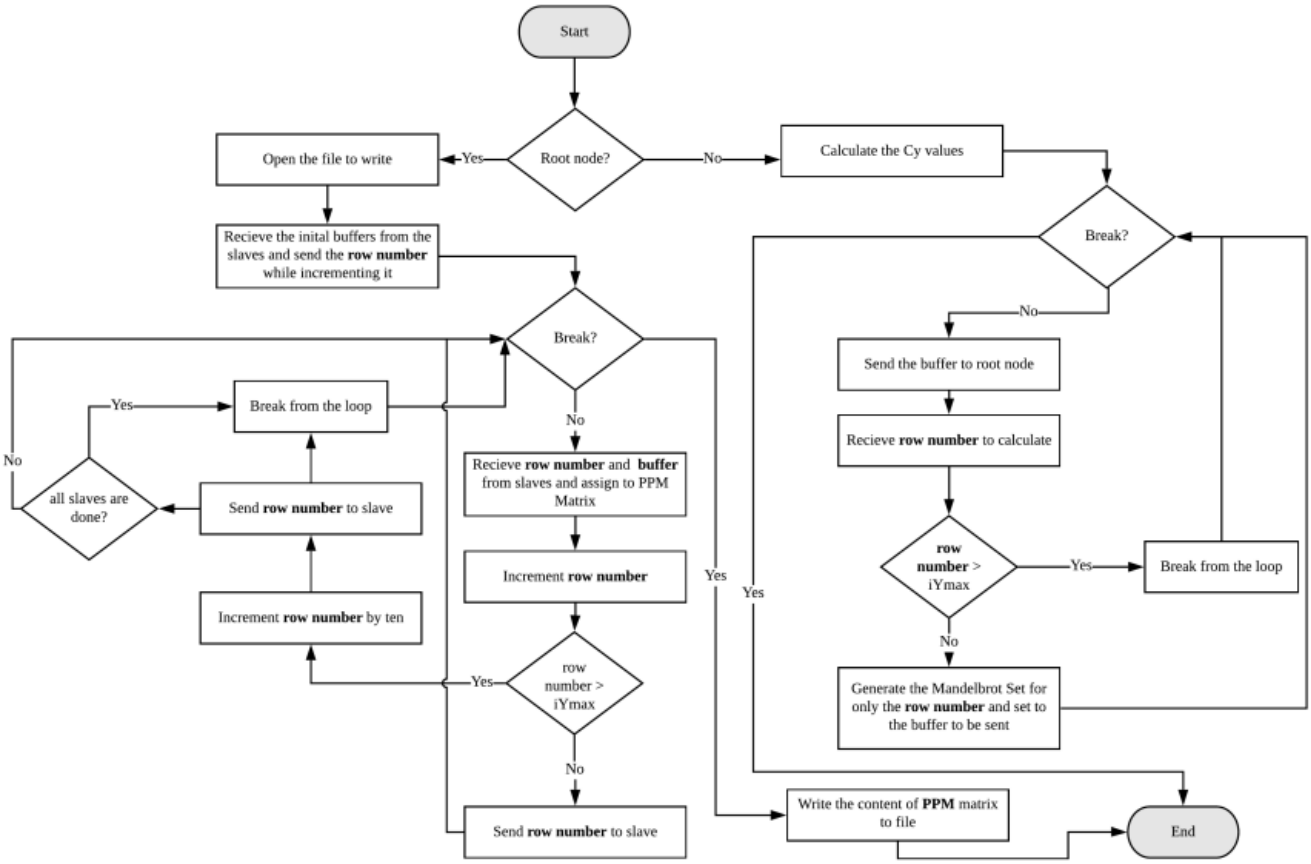
Статично циклично балансиране

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 диаграма за подхода към динамичното балансиране



Стартиране

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

./mandelbrot.exe

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

./mandelbrot_dynamic.exe

И двете програми разпознава 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 (важи само за програмата, реализираща статично циклично балансиране)
-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.

Тестване и настройка.

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

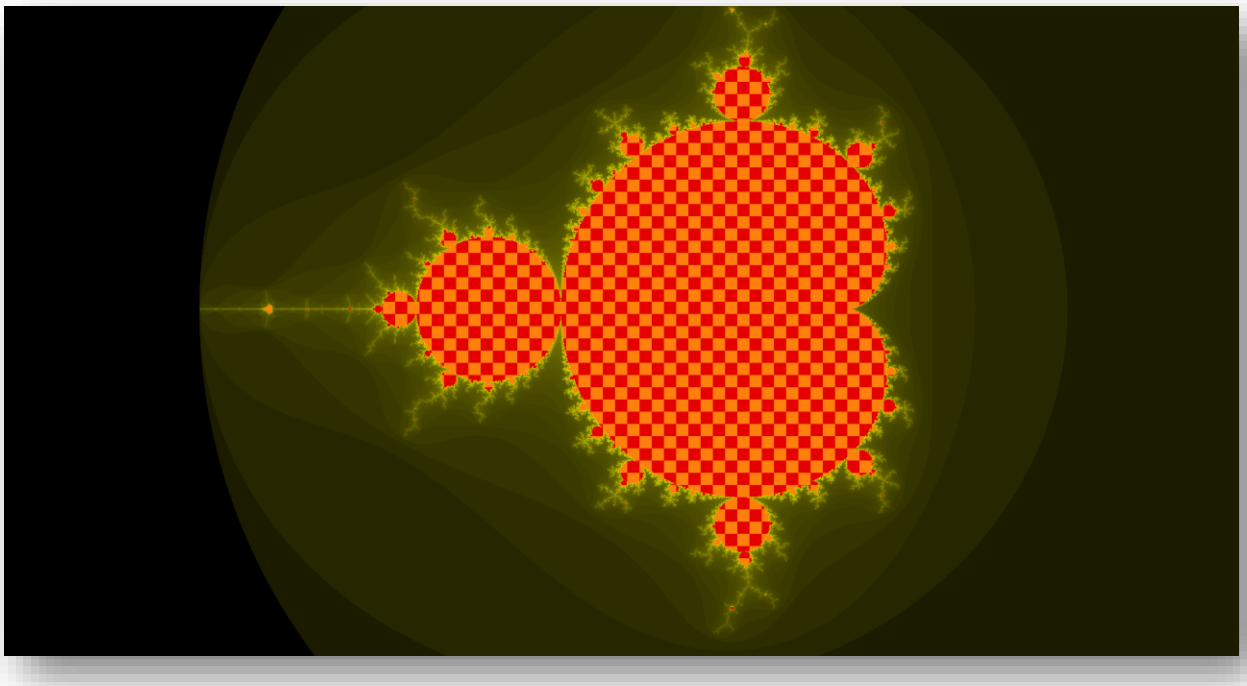
- p – паралелизъм (броят нишки)
- g – грануларност
- $Tp^{(i)}$ – време за изпълнение при паралелизъм p, получено на i-тия тест
- S_p – ускорение при паралелизмр
- E_p – ефективност при паралелизъм

Тестове, проведени на сървъра на адрес **t5600.rmi.yaht.net**

Спецификации:

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
CPU family: 6
Model name: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
CPU MHz: 3000.000
CPU max MHz: 3000.0000
CPU min MHz: 1200.0000
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 20480K

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***

Резултати при реализация със статично циклично балансиране

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	9.6063	9.6027	9.6162	9.6093	9.6027	1.9763	0.9881
4	1	5.0913	5.0894	5.0966	5.0930	5.0894	3.7288	0.9322
8	1	3.1057	3.1046	3.1089	3.1067	3.1046	6.1129	0.7641
16	1	2.0808	2.0801	2.0830	2.0815	2.0801	9.1237	0.5702
32	1	2.0872	2.0601	2.0772	2.0687	2.0601	9.2119	0.2879
1	2	18.9851	18.9715	18.9781	18.9731	18.9715	1	1
2	2	9.6065	9.5996	9.6029	9.6004	9.5996	1.9763	0.9881
4	2	5.0914	5.0878	5.0895	5.0882	5.0878	3.7288	0.9322
8	2	3.1058	3.1035	3.1046	3.1038	3.1035	6.1129	0.7641
16	2	2.0809	2.0794	2.0801	2.0795	2.0794	9.1237	0.5702
32	2	2.0788	2.0644	2.0859	2.0908	2.0644	9.1898	0.2872
1	4	19.0110	19.0159	18.9821	19.0149	18.9821	1	1
2	4	9.6196	9.6220	9.6049	9.6215	9.6049	1.9763	0.9881
4	4	5.0984	5.0997	5.0906	5.0994	5.0906	3.7288	0.9322
8	4	3.1100	3.1108	3.1053	3.1106	3.1053	6.1129	0.7641
16	4	2.0837	2.0842	2.0805	2.0841	2.0805	9.1237	0.5702
32	4	2.1032	2.0926	2.0950	2.0705	2.0705	9.1679	0.2865
1	8	19.0206	19.0103	18.9946	19.0432	18.9946	1	1
2	8	9.6244	9.6192	9.6113	9.6359	9.6113	1.9763	0.9881
4	8	5.1009	5.0982	5.0940	5.1070	5.0940	3.7288	0.9322
8	8	3.1116	3.1099	3.1073	3.1153	3.1073	6.1129	0.7641
16	8	2.0848	2.0836	2.0819	2.0872	2.0819	9.1237	0.5702
32	8	2.0960	2.0724	2.0871	2.0828	2.0724	9.1655	0.2864

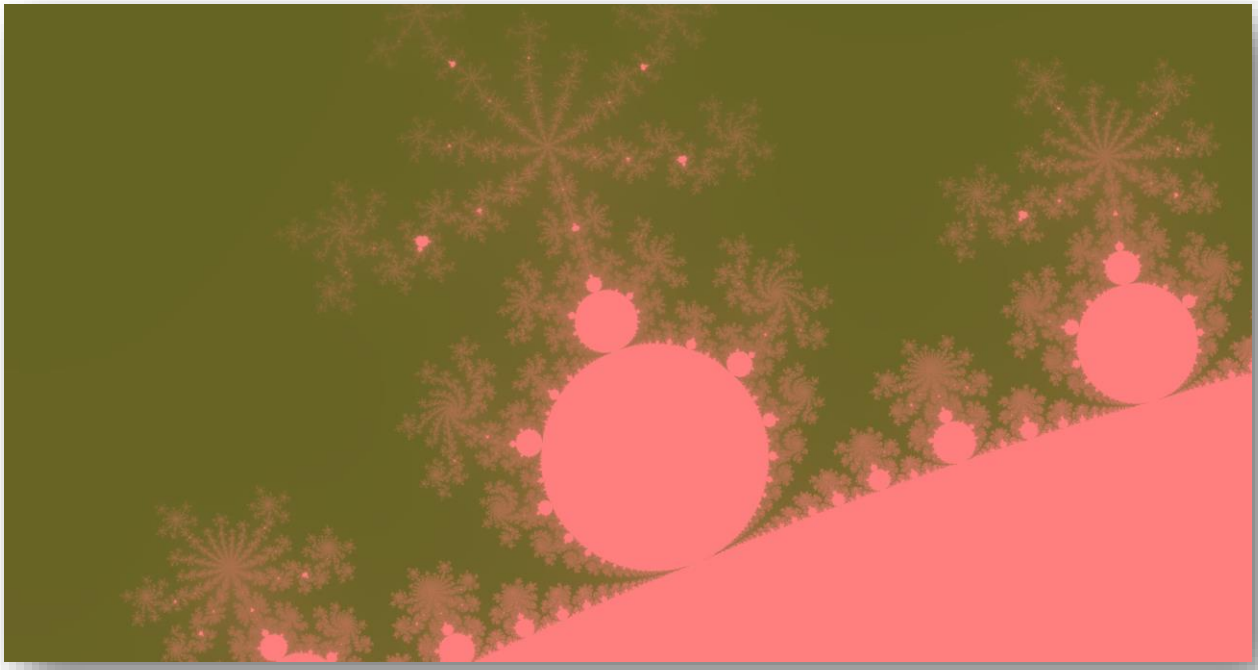
Фигура 4: *Графика на ускорението при статично циклично балансиране, 10000 итерации, по редове, област:* ***-1.4:0.6:-0.7:0.7****, размер на генерирано изображение:* ***2K***

Изложеното на ***Фигура 4*** е показателно за подобрението при ускорението с увеличаването на броя нишки. Физическите ограничения налагат спад в ефективността.

Тестове, проведени на личен компютър

Спецификации:

Model: AMD Ryzen 5 5600X
Cores: 6
Threads: 12
Base Clock Speed: 3.7 GHz
Max Boost Clock: Up to 4.6 GHz
L1 Cache:
L1 Instruction Cache: 32 KB per core
L1 Data Cache: 32 KB per core
L2 Cache: 3 MB shared across all cores
L3 Cache: 32 MB shared across all cores



Фигура 4: *Генерирано тестово изображение, 10000 итерации, по редове, област:* ***--0.375:--0.245:0.64:0.685.****, размер на генерирано изображение:* ***2K***

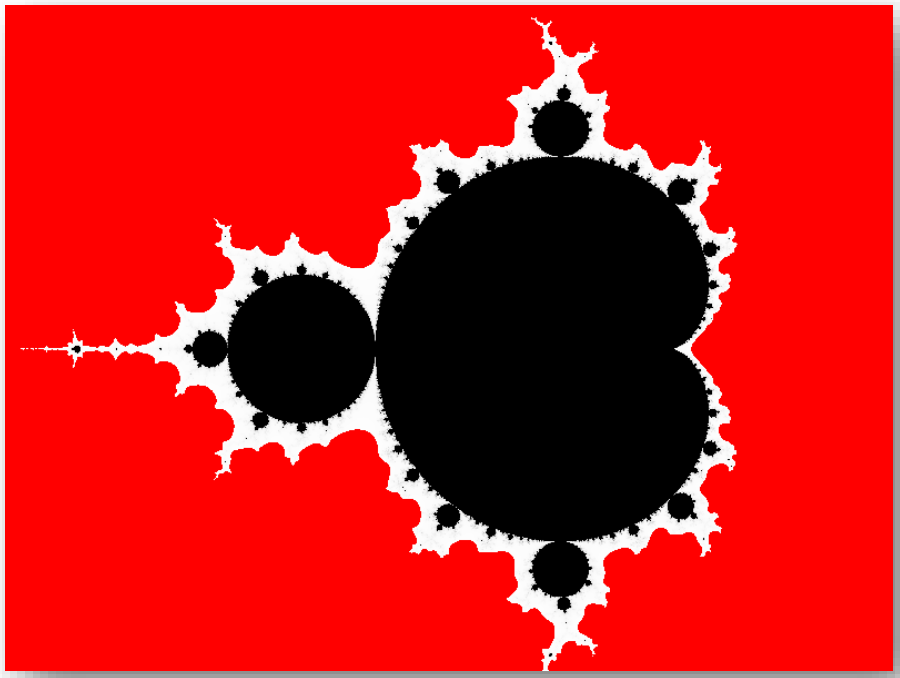
p	g	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	$T_p^{(4)}$	$minT_p^{(i)}$	$S_p = \frac{T_1}{T_p}$	$E_p = \frac{S_p}{p}$
1	1	23.608	23.0532	23.2526	23.5461	23.0532	1	1
2	1	13.6494	13.548	13.55802	13.46292	13.46292	1.712348	0.856174
4	1	8.99862	8.91558	8.94888	9.15228	8.91558	2.585721	0.64643
8	1	5.71911	6.43908	6.05748	5.897166	5.71911	4.030907	0.503863
16	1	5.141766	5.137668	4.790838	5.26395	4.790838	4.811935	0.300746

32	1	5.128248	5.05944	4.868394	4.578852	4.578852	5.034712	0.157335
1	2	23.60365	23.6046	23.63106	23.62279	23.60365	1	1
2	2	14.63426	14.63485	14.65125	14.64613	14.63426	1.612903	0.806452
4	2	11.12204	11.12249	11.13495	11.13106	11.12204	2.122241	0.53056
8	2	11.38317	11.31036	10.4737	11.41453	10.4737	2.253611	0.281701
16	2	11.72352	11.68036	11.09734	11.50625	11.09734	2.126964	0.132935
32	2	12.93347	11.43602	12.13043	13.08972	11.43602	2.063974	0.064499
1	4	23.7032	23.62708	23.69956	23.57442	23.57442	1	1
2	4	14.69598	14.64879	14.69373	14.61614	14.61614	1.612903	0.806452
4	4	11.16895	11.13308	11.16723	11.10827	11.10827	2.122241	0.53056
8	4	12.19574	11.79409	11.87809	11.51508	11.51508	2.047266	0.255908
16	4	11.13937	11.2751	11.0648	11.12109	11.0648	2.130579	0.133161
32	4	13.49165	13.21796	12.70266	13.38667	12.70266	1.855864	0.057996
1	8	23.57995	23.47261	23.50459	23.70319	23.47261	1	1
2	8	14.61957	14.55302	14.57285	14.69598	14.55302	1.612903	0.806452
4	8	11.11087	11.06029	11.07536	11.16894	11.06029	2.122241	0.53056
8	8	12.07409	11.06873	10.51191	10.12734	10.12734	2.317747	0.289718
16	8	11.84208	11.75461	11.13163	11.62159	11.13163	2.10864	0.13179
32	8	12.49612	12.25412	10.57708	11.5081	10.57708	2.219196	0.06935
1	16	23.58817	23.72545	23.5741	23.71549	23.5741	1	1
2	16	14.62467	14.70978	14.61594	14.7036	14.61594	1.612903	0.806452
4	16	11.11475	11.17943	11.10812	11.17474	11.10812	2.122241	0.53056
8	16	11.52808	11.00991	10.12796	10.48193	10.12796	2.327625	0.290953
16	16	11.86558	11.31674	11.60039	11.7257	11.31674	2.083117	0.130195
32	16	12.41111	13.08462	10.21857	10.7115	10.21857	2.306986	0.072093

Фигура 5: *Графика на ускорението при статично циклично балансиране, 10000 итерации, по редове, област:* ***--0.375:--0.245:0.64:0.685.****, размер на генерирано изображение:* ***2K***

Изложеното на ***Фигура 5*** повтаря тезата, изложена по-горе относно подобрението при ускорението с увеличаването на броя нишки. Тук физическите ограничения са по-големи, което е в тандем със спада в ефективността.

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:* ***1000x1000***

Тестове, проведени на сървъра на адрес **t5600.rmi.yaht.net**

p	g	T_p	$min\ T_p$	$S_p = \frac{T_1}{T_p}$	$E_p = \frac{S_p}{p}$
1	200	11.6675	11.3861	1	1
1	200	11.3861			
1	200	11.5051			
1	200	11.9937			
2	112	5.8062	5.7235	1.98936	0.99468
2	106	5.7235			
2	101	5.74618			
2	104	5.75341			
4	65	3.25867	3.17777	3.583047	0.895762
4	56	3.27023			
4	63	3.20524			
4	64	3.17777			
8	46	1.8435	1.8435	6.176349	0.772044
8	49	1.8936			
8	33	1.9006			

8	33	1.8835			
16	37	1.3289	1.3079	8.705635	0.544102
16	22	1.3079			
16	36	1.3219			
16	23	1.3469			
32	30	1.4132	1.3982	8.143399	0.254481
32	16	1.4412			
32	14	1.3982			
32	28	1.4132			

Фигура 6: *Графика на ускорението при статично циклично балансиране, 10000 итерации, по редове, област-2****:-1.5:-1.4:1.6****, размер на генерирано изображение:* ***1000x1000***

Фигура 6 показва, че ускорението, постигнато с динамично балансиране, е по-лошо от това със статично. Това се наблюдава поради необходимостта от комуникация между Master процеса и Slave процеса.

Източници:

[1] 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

[2] Bhanuka M., Baskaran V., 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.