Курсов Проект по Разпределени софтуерни архитектури

Изобразяване на Фрактал

Изготвил: Виктор Христов, Софтуерно инженерство, курс 3, №62151

Дата: 16.06.2020

Ръководител на курса: проф. Васил Цунижев

Асистент: Христо Христов

Съдържание

1. Увод

1.1. Цел и предназначение на проектираното приложение

Целта на този проект е да се създаде програма, която генерира изображение на множество на Манделброт с функция $\mathbf{Z_{n+1}=C*cos(Z_n)}$ за комплексно число C чрез реализиране на паралелен алгоритъм.

Множеството на Манделброт е вид фрактал. Това са фигури, които рекурсивно съдържат себеподобни фигури по своите граници. Всеки фрактал се изобразява чрез множество от комплексни числа. Комплексните числа се пресмятат чрез използване на комплексна равнина. Това е координатна система, при която реалната част на числото се представя чрез абсцисната ос, а имагинерната - чрез ординатната. След като се знае стойността на комплексното число, то тогава може да се определи дали то принадлежи към конкретния фрактал. Прилага се следната система:

1) $Z_0=0$
2) $Z_{n+1}=C*\cos(Z_n)$

Ако след избран от програмиста брой итерации на функцията не може да се определи дали комплексното число клони към безкрайност, то тогава се приема, че принадлежи към това множество на Манделброт.

За да се генерира пълното изображение, ще е нужно да се изчисли всяка точка от зададената комплексна равнина, независимо дали е част от множеството на Манделброт или не. Единствено след като е приключило изчислението ще може да се запази генерираното изображение във визуален формат.

1.2. Преглед на решения на задачата на Манделброт

В тази подточка ще се разгледат три решения на задачата за изобразяване на множество на Манделброт. Всяко от тях обяснява какво представлява задачата, как чрез комплексни числа тя се изобразява на координатна система и как се пресмята дали едно число е част от множеството. Поради това при този анализ ще се обърне внимание само на по-значителните части от статиите, главно какви алгоритми прилагат решенията и как са тествани.

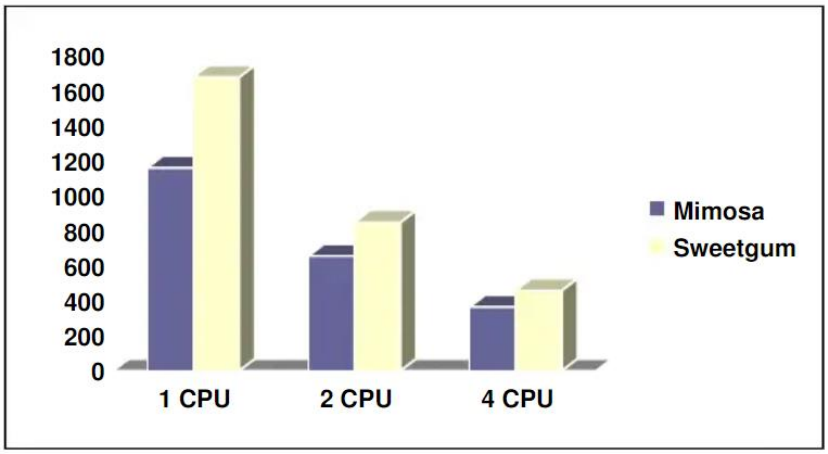
1) Parallel Implementation and Analysis of Mandelbrot Set Construction [1]

- Авторът обяснява „escape time“ алгоритъма, чрез който се извършва оцветяването на всеки пиксел от изображението. Алгоритъмът се състои от многократно изчисляване на функцията $Z_{n+1}=Z_n^2 + C$. Константата C се определя спрямо координатите на пиксела, който в момента се пресмята, а броят максимални итерации е предварително зададен от програмиста. При всяка итерация се извършва проверка дали е удовлетворено условието за излизане.
  - Ако е изпълнено това условие, то се приема, че пикселът е извън множеството на Манделброт и се оцветява спрямо броя на итерации, които се били нужни, за да се изпълни условието.
  - Ако всички итерации приключат без да се изпълни условието, то се приема, че пикселът принадлежи на множеството и се оцветява в един конкретен цвят, като общоприетия вариант е черен цвят.

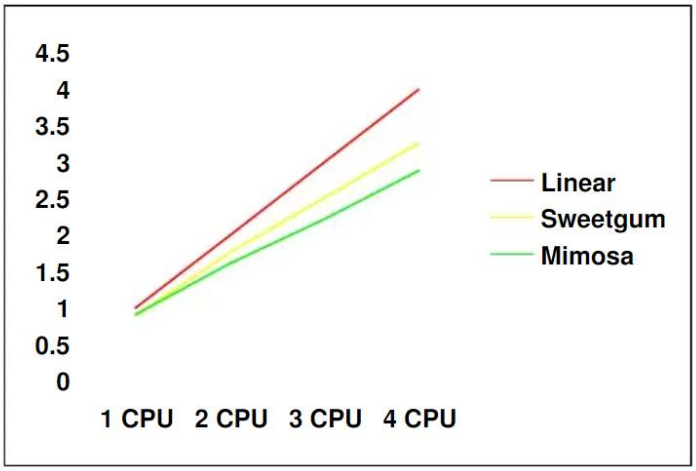
- Авторът обръща внимание на проблема, че при разделянето на изображението на големи задания се получава лошо балансиране на работата на процесите, и предлага своето решение.

- Идеята на неговия паралелен алгоритъм е всеки процес да изчислява случайно избрани пиксели от изображението. Инициализират се два вектора, които представляват интервала от допустими стойности на реалната и имагинерната ос от комплексната равнина. На случаен принцип се генерира число, спрямо което стойностите на векторите се разбъркват. След това векторите се използват, за да се генерира изображението.

- Комуникацията между процесите е минимизирана. Преди да се извършат изчисленията, процесите получават информация за векторите, след което използват тази информация, за да пресметнат заданията, които са им разпределени. Накрая главния процес събира всичките задания и сглобява пълното изображение.



Фигура 1: Графика на тестови резултати представяща време за изпълнение спрямо брой процеси



Фигура 2: Графика на тестови резултати представяща ускорение спрямо брой процеси

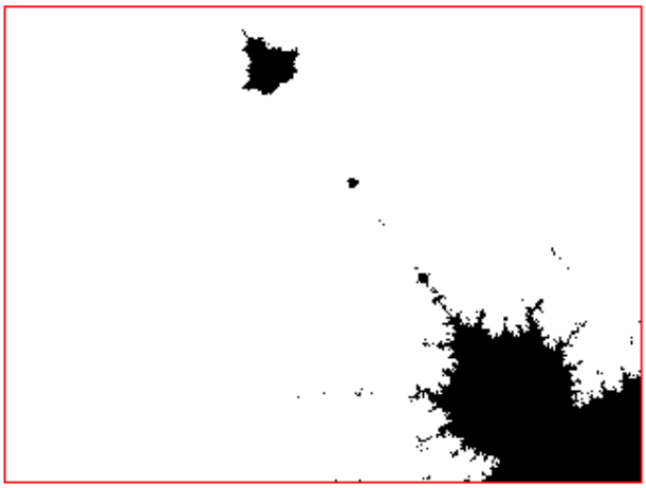
- Авторът е показал следните графики като резултат от проведените тестове. Полученото ускорение е близо до линейното, но не може да се направи точен извод, защото е тествано единствено до 4 процеса. Още от изпълнение с 3 процеса започва да намалява ефективността, което подсказва че при тестване с повече от 4 процеса резултатите няма да са задоволителни.

- Под въпрос е колко е правилно да се използва случайно-генерирано число при разбъркване на стойностите на векторите. Случайността призовава единствено допълнителен хаос и несигурност в програмата. В повечето случаи ще се получи разбъркване на координатите по такъв начин, че всеки процес да обработва равномерно-балансирани задания. Но в някои случаи разбърквания ще доведат до концентриране на точките от множеството в малък брой задания и няма да реши проблема на едрата грануларност. Това щеше да си проличи, ако авторът беше показал подробно тестови резултати.

- Авторът описва своя алгоритъм на високо ниво и без показване на примерен код, поради което възникват въпроси относно реализацията на паралелния алгоритъм. Неговият подход решава проблема за балансирането на работата в повечето случаи, но не е обяснено как точно заданията се разпределят между процесите след като те получат информацията от векторите.

2) Parallel Fractal Image Generation - A Study of Generating Sequential Data with Parallel Algorithms [2]

- Авторът на тази статия подчертава важността от броя на итерациите при изчисляване дали точка принадлежи на множеството на Манделброт. Някои точки се нуждаят от огромен брой итерации, за да се изпълни условието на „escape time“ алгоритъма, споменат в първия източник.



Фигура 3: Сектор от множеството на Манделброт след изчисление с 150 итерации

Фигура 4: Сектор от множеството на Манделброт след изчисление с 1500000 итерации

- Колкото повече итерации се извършват, толкова по-правилно ще бъде крайното изображение. Но от друга гледна точка, колкото повече итерации се извършват, толкова по-бавно ще се генерира изображението и приключи изпълнението на програмата. Поради това е важно да има компромис между точността и бързодействието на алгоритъма.

- Относно алгоритъма за паралелна обработка, авторът също разглежда проблема за балансиране на данните и показва, че при едра грануларност малък брой процеси ще вършат по-голямата част от работата. Като решение на проблема предлага вместо отделни задания всеки процес да изчислява определени редове от изображението. Балансирането на данните се извършва статично. Първия процес изчислява 1ви, 5ти, 9ти... ред от изображението, втория процес изчислява 2ри, 6ти, 10ти... ред и аналогично за останалите процеси.

- Авторът отбелязва, че това не е перфектно решение. Причината е, че някои редове изискват повече изчисления от други, но този подход е достатъчно ефективен за повечето видове фрактали.

- Самото изчисление не изисква комуникация между процесите, защото всеки точка от изображението е независима една от друга. Комуникация е нужна единствено при сглобяване на крайния резултат. В тази имплементация той се принтира на конзолата като ASCII изображение. Това довежда до нужда от синхронизация между процесите, защото те изчисляват алтернириращи редове, но трябва всеки процес да си изведе редовете в конзолата в правилна подредба.

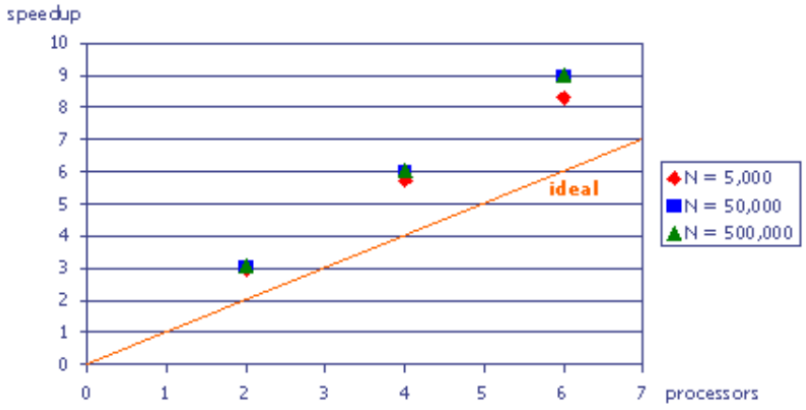
- Авторът решава този проблем чрез имплементиране на „Polling“ синхронизация. Чрез този алгоритъм извеждането на завършеното изображение се разделя на задания с еднакъв размер. Главния процес изпраща съобщение към останалите процеси, чрез което иска да получи техните първи изчислени редове. След това главния процес изчислява първия свой ред и ако при завършване на изчислението е получил първите редове на всички процеси, то тогава цялото задание се принтира и се изтрива от паметта. В противен случай главният процес продължава да изчислява следващите свои редове докато не получи отговор от всички други процеси. След извеждането на първото задание, алгоритъмът се повтаря за следващите задания докато не се изведе цялото изображение.

- Авторът използва този алгоритъм, защото си е поставил ограничение, че никой процес не трябва да съдържа информация за цялото завършено изображение. Ако нямаше това ограничение, то можеше главния процес да съдържа буфер, в който останалите процеси да записват своите редове и да се премахне напълно нуждата от синхронизация.

- За проверка на алгоритъма се изпълняват тестове върху изображение с размер 160x120. Получени са следните резултати:

average $T_{P,N}$	P = 1 (serial)	P = 2	P = 4	P = 6
N = 5,000	1.43199333 s	0.48863167 s	0.25034200 s	0.17215833 s
N = 50,000	14.18023330 s	4.67098667 s	2.36133667 s	1.58265667 s
N = 500,000	141.93433300 s	46.43553330 s	23.48320000 s	15.71240000 s

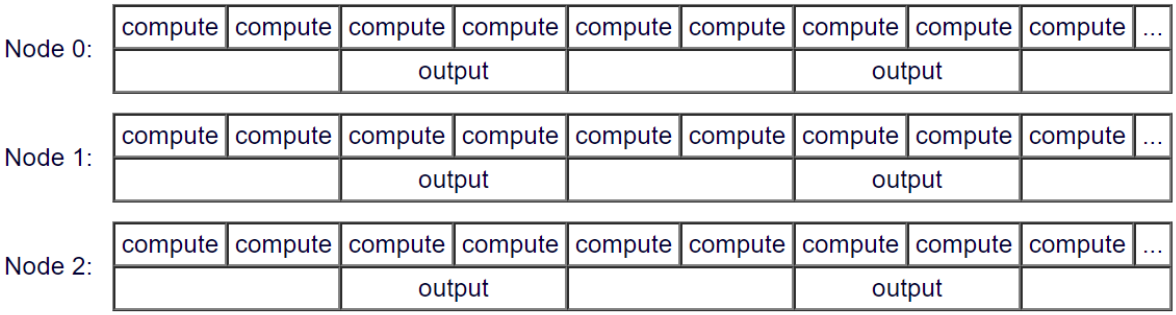
Фигура 5: Таблица с тестови резултати, където N – брой итерации, P – брой процеси и $T_{P,N}$ – времето за изпълнение на теста



Фигура 6: Графика на тестови резултати представящ ускорение спрямо брой процеси при тестване на различен брой итерации

- Както се вижда, авторът получава суперлинейна аномалия в своите резултати. Това е аномалия, която възниква когато ускорението на някой тест се изчисли да е по-голямо от броя на използваните процеси в теста. Тази аномалия предупреждава, че при последователното изпълнение на програмата е възникнало значително закъснение, което липсва при паралелно изпълнение.

Фигура 7: Визуализиране на работата при последователно изпълнение на програмата



Фигура 8: Визуализиране на работата при паралелно изпълнение на програмата

- Авторът открива и описва къде лежи проблема, който поражда тази аномалия. При изпълнението на програмата с единствен процес, изчисляването и принтирането на редовете става напълно последователно. Когато се изпълнява програмата с повече от един процес, то изчисленията стават паралелно. Уникалното е, че изпращането на съответните редове към главния процес става във фонов режим докато процесите изчисляват следващите редове. Така се достига допълнителен слой паралелизъм, който спестява времето нужно за изпращане на редовете. Докато в последователното изпълнение един процес отговаря за всяко принтиране, което допълнително забавя програмата и води до по-силно проявено ускорение при сравнение с резултатите при паралелно изпълнение.

3) Parallel Mandelbrot in Julia, C++, and OpenCL [3]

- Julia като програмен език не съдържа експлицитни нишки, но позволява размяна на съобщения между различни процеси работници, чрез което се синхронизира тяхното изпълнение. Използва се споделен масив, в който всичките процеси работници записват своите изчислени задания. Главната програма изчаква всеки процес да приключи изпълнението си, но останалите детайли за паралелния алгоритъм не стават ясни.

- Авторът реализира паралелен алгоритъм за решаване на задачата на Манделброт първоначално чрез езика Julia, след което транслира кода към C++ и OpenCL с цел да се сравнят резултатите в различни езици.

- При паралелизирането на алгоритъма в C++, авторът използва Threading Building Blocks, за да раздели работата на програмата на различни задания спрямо зададена грануларност. TBB е модел за паралелно програмиране, който е базиран на динамична обработка. [4] TBB сам създава и планира процесите, които ще работят върху заданията на програмата. Когато един процес работник приключи своето задание, той получава следващото в опашката и продължава своята работа. Така всеки процес не губи време в чакане на останалите, а вместо това постоянно получава нови задания да обработва.

- Чрез TBB авторът реализира динамично разпределяне на работата, защото кой процес кое задание ще обработи не се определя преди да се стартира програмата.

- Тестовете на програмата са следните:

	CPU	GPU
MBA	2-core Intel Core i7 @ 2.0GHz	
MBP	4-core Intel Core i7 @ 2.6GHz	Nvidia GT750M
MP	6-core Intel Xeon @ 2.5GHz	2 x AMD D700

Фигура 9: Архитектура на тестовите машини, където MBA – MacBook Air 2012, MBP – MacBook Pro 2013, MP – Mac Pro 2013

	MBA	MBP CPU	MBP GPU	MP CPU	MP GPU
Julia	10.9	6.15		6.51	
Parallel Julia	5.55 [3]	2.53 [4]		2.45 [5]	
Serial C++	10.05	8.45		7.52	
TBB C++ [6]	2.65	1.21		0.74	
OpenCL	1.40	0.45	1.05	0.28	0.07

Фигура 10: Време в секунди за изпълнение на проведените тестове на всеки алгоритъм

Julia: Parallel Speedup

	Speedup	Cores	Speedup/Core
MBA	1.96	2	0.98
MBP	2.43	4	0.61
MP	2.66	6	0.44

C++: Parallel Speedup

	Speedup	Cores	Speedup/Core
MBA	3.79	2	1.90
MBP	6.98	4	1.75
MP	10.16	6	1.69

Фигура 11: Ускорение при изпълнение на проведените тестове на паралелния алгоритъм на Julia и C++

- На пръв поглед отново се получава суперлинейна аномалия при изпълнението на паралелния C++ алгоритъм, но благодарение на Хипернишковата технология на Intel всяко ядро има възможност да изпълнява до два процеса едновременно. Тази технология е приложима, защото алгоритъмът е реализиран чрез TBB. Така реално хардуерната граница на ускорението ще е два пъти по броят използвани ядра и аномалията се избягва.

1.3. Извод от разгледаните решения на задачата на Манделброт

1) Използвани технологии в решенията

- Първият източник разработва задачата на C++, за да може да приложи библиотека EasyBMP. За паралелизиране на задачата е използвана серията от библиотеки Message Passing Interface, както и Portable Batch System за да се достави програмата към двете тестови машини – Mimosa и Sweetgum. Mimosa е система, базирана на разпределена памет, докато Sweetgum е базирана на споделена памет.

- Вторият източник описва реализацията на алгоритъма под формата на псевдокод и не става ясно на какъв език е написан. Споменато е, че в тази реализация също се използва Message Passing Interface, което подсказва, че програмния език е C, C++ или Fortran. Тестовете са извършени върху суперкомпютърът на университета в Монтана.

- Третия източник реализира алгоритъма на езиците Julia, C++ и фреймуърка OpenCL. C++ имплементацията допълнително използва Threading Building Blocks на Intel. Тестовете са извършени на следните три машини: MacBook Air 2012, MacBook Pro 2013 и Mac Pro 2013.

2) Избор на език

- Най-подходящия език за реализацията на този проект ще бъде обектно-ориентиран език, който позволява програмиране от високо ниво. Така разработената програма ще може лесно да се раздели на компоненти, както и да бъде по-разбираема, защото кодът на високо ниво се абстрахира от детайли, които биха евентуално довели до объркване. Също така езикът трябва да поддържа работа с паралелни процеси.

- C++ е добър избор за език, който покрива горните изисквания. Той е използван в решаване на задачата при разгледаните образци, което подсигурява, че ще е възможно да се създаде удовлетворяваща програма чрез него.

- Поради своя личен опит, като алтернатива на C++ избрах да разработя програмата на Java. Той също покрива горните изисквания и дори е език на по-високо ниво от C++ поради допълнителния слой абстракция, който възниква от виртуалната машина на Java. Това би улеснило процеса на създаване на решението.

3) Избор на софтуерна архитектура

- Задачата, която този проект цели да реши е генериране на изображение на фрактал чрез паралелна обработка. В този случай изображението може да се представи като една матрица. Тази матрица лесно ще се обработи, ако се приложи модела Single Program Multiple Data. Чрез него цялостното изображение ще се декомпозира на множество от задания, всяко от които съдържа еднакъв брой редове от изображението. Тези задания ще бъдат паралелно изчислени от различни процеси, които ще се стартират от една програма.

- Следователно най-удобно ще бъде да се ползва йерархичната архитектура Master – Slaves. Чрез нея логиката на програмата ще се раздели на един главен компонент, който ще съдържа всички важни данни нужни за изпълнението и един компонент Slave, който ще отговаря за пресмятане на изображението спрямо указанията на компонента Master.

4) Метод на балансиране и други имплементационни детайли

- Работата на всеки процес ще бъде балансирана динамично, както е подходено в третия разгледан източник. Така всеки процес ще обработва първото свободно задание и няма да има нужда от статично определяне кой процес за кое задание отговаря. В този случай ще е нужна синхронизация между работата на процесите и реализацията на паралелния алгоритъм ще се затрудни, но за сметка на това ще се получи решение, което е по-елегантно и по-разбираемо от високо ниво.

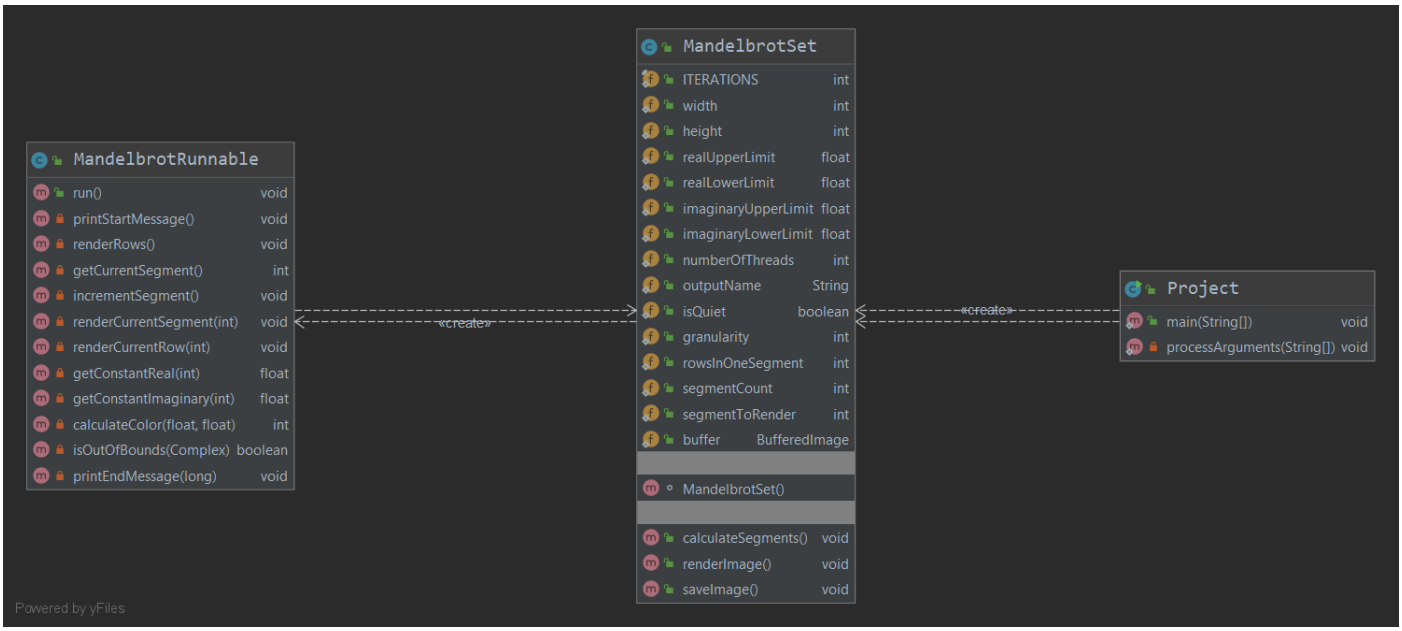
- Цветът на всяка точка от матрицата ще се определя чрез имплементиране на escape time алгоритъма, който бе обяснен в анализа на първия източник.

- За генериране на изображението ще се използва един главен буфер, който ще се съдържа в Master компонента. Той ще запазва цветовете на всички точки, изчислени от паралелните процеси. Така ще се избегне проблемът със синхронизация на крайния изход, който възниква във втория източник.

2. Проектиране

2.1. Диаграми и дизайн на програмата

1) Клас диаграма

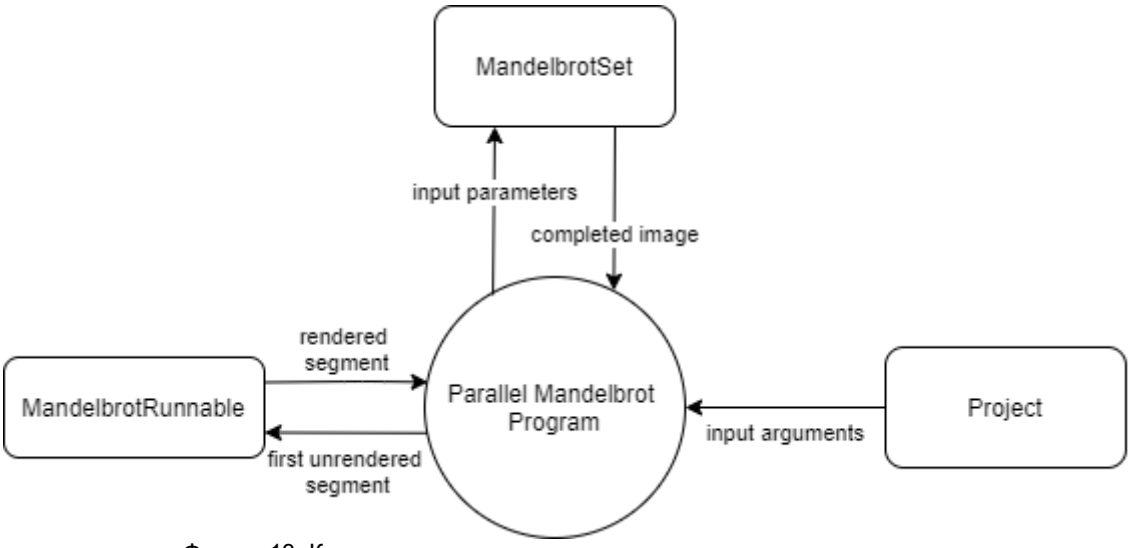


Фигура 12: Клас диаграма на програмата

Програмата ще се раздели на три различни компонента, които са изобразени чрез класовете в диаграмата. Накратко описано:

- Компонент Project показва програмата на най-високо ниво на абстракция. Той отговаря за обработване на входните аргументи при стартирането на програмата и извикване на методите на MandelbrotSet, за да се изпълни задачата.
- Компонент MandelbrotSet отговаря за запазването на всички нужни параметри на програмата. Той стартира процесите, които паралелно ще изчислят изображението, след което се приспива докато не приключат всички други процеси. Накрая той записва изображението във файл.
- Компонент MandelbrotRunnable представя процесите, които паралелно ще изчисляват изображението. Неговото единствено задължение е да пресметне заданията, които получава и да ги записва в общия буфер, който се намира в MandelbrotSet.

2) Контекстна диаграма на поток на данните

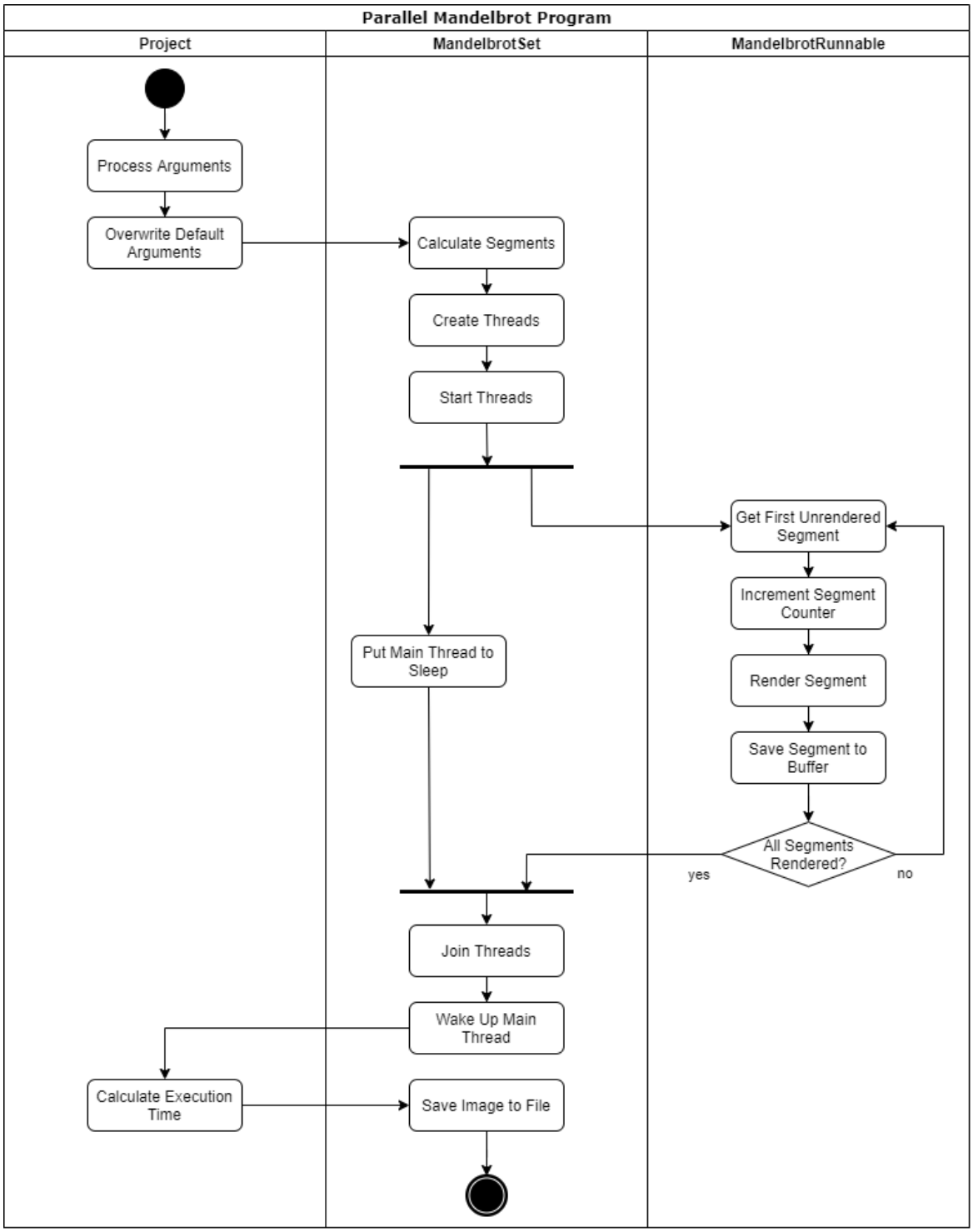


Фигура 13: Контекстна диаграма на поток на данните на програмата

Тази диаграма показва как данните ще се движат и променят по време на изпълнението на програмата. Спрямо компонентите:

- Компонент Project обработва входните аргументи при стартирането на програмата, след което ги записва в MandelbrotSet.
- Компонент MandelbrotSet запазва и модифицира своите параметри спрямо работата на другите компоненти. Накрая на изпълнението използва своя буфер, за да запази завършеното изображение като файл.
- Компонент MandelbrotRunnable взима номера на първото необработено задание, а след завършване на изчислението записва заданието в общия буфер на MandelbrotSet.

3) Диаграма на дейностите



Фигура 14: Диаграма на дейностите на програмата

Тази диаграма описва как протича изпълнението на програмата. Нейното обяснение съвпада със следващата подточка 2.2.

2.2. Подробно описание на изпълнението на програмата

1) Стартиране на програмата

Програмата може да бъде стартирана с или без входни аргументи. Възможните аргументи на програмата са следните:

1) Размер на изображението
2) Сектор от изображението, който да се визуализира
3) Брой паралелни процеси на програмата
4) Име на генерирания файл
5) Дали програмата да е в тих режим
6) Коефициент на грануларност

Ако липсват някои входни аргументи или се въведат в невалиден формат, то програмата стартира с предварително зададените от програмиста параметри. Входен аргумент, който е в правилен формат се записва на мястото на съответния параметър по подразбиране.

2) Декомпозиция на данните

След като параметрите на програмата са вече финализирани, програмата трябва да декомпозира цялостното изображение на задания. Това зависи от два параметъра: броя паралелни процеси ‚p‘ и коефициента на грануларност ‚g‘. Коефициентът g показва колко задания ще се заделят за всеки процес.

Програмата извършва декомпозиция по редове, което означава че всяко задание ще се състои от един или повече редове. Така едно задание ще се състои минимално от 1 ред, а максимално – от толкова редове, колкото е височината на изображението. Колко реда съдържа всяко задание се определя като височината на изображението се раздели на броя задания.

- Например за p = 5 и g = 8 получаваме, че изображението ще се раздели на p * g = 40 задания. Ако изображението има височина 2000 пиксела, то броя редове във всяко задание ще бъде 2000 / 40 = 50.

Важно е да се обърне внимание на случая, в който височината не се дели точно на броя задания. В тази неудобна ситуация остатъкът от делението ще е броя на редовете, които не се включват в никое задание и остават необработени. Разгледани са три варианта за решение на този проблем, но всеки от тях има страничен ефект, който може да доведе до разминаване между това, което програмиста очаква и това, което програмата генерира.

- Първото решение е да се увеличи броя на редовете, които заданията обработват. Това води до цялото изображение да се изчислява с по-малък брой задания и така получаване на по-едра грануларност, отколкото е било предназначено.
- Второто решение е да се увеличи броя на заданията, за да се покрият висящите редове. Това води до цялото изображение да се изчислява с по-голям брой задания и така получаване на по-финна грануларност, отколкото е било предназначено.
- Третото решение е да се увеличи броя на редовете на само част от заданията. Това води до изчисляване на цялото изображение със първоначалния брой задания, но се губи свойството всяко задание да има еднакъв размер данни.

И трите решения могат да доведат до други бъдещи проблеми, главно свързани с неочаквани резултати по време на тестването. Но от по-голямо значение е последните редове да не бъдат оставени висящи, поради което трябва да се имплементира някое от решенията. Тази програма имплементира първото решение поради лесната му реализация. В случай на неточно делене, броят на редовете, обработени при от задание се увеличава с единица. Допълнителният ред при всяко задание е достатъчен, за да се покрият всички висящи редове.

3) Стартиране на паралелната обработка

След като завърши декомпозицията на данните, главния процес създава толкова на брой процеса, колкото е стойността на параметъра ‚p‘. Тези процеси се стартират, след което главния процес заспива докато не приключат всички други процеси. В противен случай главния процес би приключил своето изпълнение преди останалите процеси и като резултат изображението, което се запазва във файл ще бъде частично генерирано.

4) Паралелна обработка на задачата и балансиране на работата

Както бе споменато в извода от анализа, разпределянето на заданията ще става динамично като всеки процес заема първото свободно задание. Програмата ще съдържа една споделена променлива, която представлява номера на първото свободно задание. Всеки процес ще изпълнява цикъл, в който се извършват следните действия:

I. **Прочитане на споделената променлива и запазване на номера на заданието, което се заема, в локална променлива**. Използването на локална променлива е задължително, защото номерът на обработваното задание е нужен при определяне на кои редове трябва да се пресметнат от процеса, а използването на споделената променлива за тази цел би довело до проблеми. Тя може да се промени по всяко време като най-опасното от тях е когато процес изчислява редовете на своето задание.

II. **Проверка дали номерът на заданието е валиден.** Ако той е по-голям от общия брой задания, то процеса прекратява цикъла и завършва изпълнението си.

III. **Писане в споделената променлива чрез инкрементиране на стойността ѝ.** Така останалите процеси се уведомяват, че предното по номер задание е вече зает.

IV. **Изчисляване на редовете от заданието.** Ако редовете, определени от заданието, надвишат височината на изображението, то се смятат редовете до последния допустим.

V. **Записване на изчислените редове в общ буфер.** Поради факта, че буферът е двумерна матрица и всяко задание представлява различен набор от редове, не е възможно да се получи конфликт при записване на редовете в буфера и няма нужда от синхронизация на тази стъпка.

VI. **Приключване на итерацията и започване на следващата.** Така цикълът се връща към стъпка I.

При реализацията на динамично балансиране на работата е нужна синхронизация между процесите при четенето и писането в споделената променлива. Приложен е модела Parallel Random Access Machine, по-конкретно схемата Exclusive Read, Exclusive Write. Използваната споделена променлива не може да се чете или променя от повече от един процес.

Ако тази променлива е заета от един процес и други се опитат да я достъпват, то останалите процеси се приспиват докато първия процес не я освободи. Нормално това би било предпоставка за създаване на bottleneck и забавяне на изпълнението на програмата, но в този алгоритъм цялата синхронизация се извършва върху два реда код, съответно четенето и писането. Това осигурява бързото заемане и освобождаване на споделената променлива и се избягва процесите да стоят значително време приспани.

5) Приключване на програмата

След приключването на всички процеси, главния процес се събужда. Той изчислява времето нужно за изпълнение на паралелната обработка като от сегашното време в UNIX формат се извади времето, в което е стартирана програмата. След това главния процес използва вече запълнения буфер, за да запази готовото изображение като файл и приключи изпълнението на програмата.

3. Тестване

3.1. Архитектура на тестовата машина

Тестовия план за валидиране, че програмата работи коректно, е извършен на сървърът на ФМИ. Той има следните характеристики:

- Име на модел: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
- Архитектура: x86_64
- Брой процесори: 32
- Брой сокети: 2
- Брой ядра на сокет: 8
- Брой нишки на ядро: 2
- Памет: 62,7 GB

3.2. Тестови параметри

По време на провеждане на тестовия план ще се изменят параметрите, за да се определени до каква степен влияят на крайния резултат. Подробното описание на използваните параметри:

1) Размер на изображението

   Размерът на изображението се разделя на дължина и височина. То се означава във формат [дължина]x[височина]. В тестовите таблици се отбелязва чрез колона с име „Size“.

2) Сектор от изображението, който да се визуализира

   Секторът на изображението указва горна и долна граница на реалната и имагинерната ос от комплексната равнина, която ще визуализира генерирания фрактал. Всяка точка от равнината може да се представи като комплексно число **C = a + i * b**, където ‚a‘ е стойността на реалната част на C, а ‚b‘ – стойността на имагинерната част на C. Поради това секторът ще бъде определен чрез два интервала, един за ‚a‘ и един за ‚b‘, от вида [x, y], където ‚x‘ е долната граница на съответната ос, а ‚y‘ – горната граница на съответната ос. В тестовите таблици се отбелязва чрез колони с име ‚a‘ и ‚b‘.

3) Брой паралелни процеси на програмата

   Този параметър показва колко паралелни процеса работят по време на изпълнението на програмата. Тестовия план ще изследва работа на програмата с 1 до 32 процеса. В тестовите таблици се отбелязва чрез колона с име ‚p‘.

4) Коефициент на грануларност

   Коефициента на грануларност определя колко задания се заделят за всеки процес. Както бе споменато в подточка 2.2, този коефициент се използва за пресмятане на колко задания да се декомпозира изображението. В тестовите таблици се отбелязва чрез колона с име ‚g‘.

3.3. Забележки относно провеждането на тестовете

- Единственият параметър, който влияе на крайния резултат, но никога не се променя е броят на извършените итерации за определяне дали една точка принадлежи на множеството на Манделброт. Всички тестове са извършени с 500 такива итерации.

- Времето за изпълнението на тестовете се измерва в милисекунди. В тестовите таблици то се отбелязва чрез колоните с име „Test X“, където X е номерът на теста.

- Извършени са 5 на брой теста за всяка комбинация от параметри. Те се сортират в нарастващ ред и от тях се взима най-бързото изпълнение.

- Този най-бърз резултат в таблиците се оцветява в зелено и се използва за пресмятане на ускорението и ефективността.

- Ускорението от паралелната обработка се пресмята по формулата $S_p = T_1 / T_p$. В тестовите таблици то се отбелязва чрез колона с име „Sp (Test 1)“.

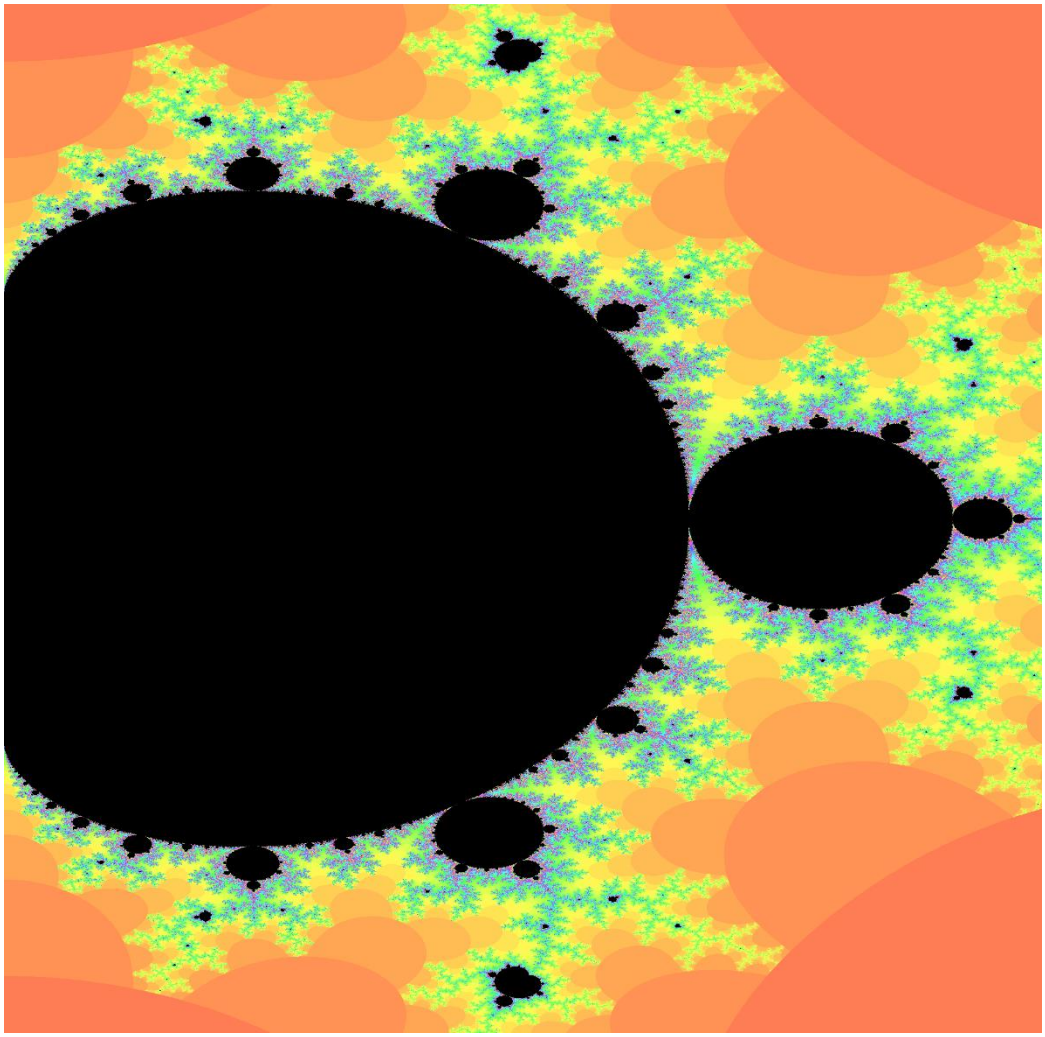
- Ефективността от паралелната обработка се пресмята по формулата $E_p = S_p / p$. В тестовите таблици то се отбелязва чрез колона с име „Ep“.

3.4. Тестване с променлив коефициент на грануларност

Първата група от тестове е извършена като се фиксират всички параметри с изключение на брой паралелни процеси и коефициента на грануларност. Избрани са следните параметрите за константи:

- Размер на изображението – 2000x2000
- Сектор на изображението – ‚a‘ е в интервал [0, 2] и ‚b‘ е в интервал [-1.5, 1.5]

Като резултат от изпълнението на програмата с тези параметри се получава следното изображение:



Фигура 15: Резултат от изпълнение на програмата за тестване на коефициент на грануларност

Тази група от тестове се разделя на три подгрупи, всяка от които има различен коефициент на грануларност. Техните стойности на променливия параметър са:

- g = 1
- g = 4
- g = 16

Получени са следните таблици в резултат от тестовете:

1) Тестване с g = 1

p	g	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	1	2000x2000	[0,2]	[-1.5,1.5]	159632	159842	167571	169078	169739	1,00	100,00%
2	1	2000x2000	[0,2]	[-1.5,1.5]	82952	83774	85742	89886	160851	1,92	96,22%
3	1	2000x2000	[0,2]	[-1.5,1.5]	96840	98042	101829	102552	102746	1,65	54,95%
4	1	2000x2000	[0,2]	[-1.5,1.5]	67037	67347	69153	69281	69361	2,38	59,53%
5	1	2000x2000	[0,2]	[-1.5,1.5]	66485	66788	67614	70197	70783	2,40	48,02%
6	1	2000x2000	[0,2]	[-1.5,1.5]	49540	49858	50325	50844	52188	3,22	53,70%
7	1	2000x2000	[0,2]	[-1.5,1.5]	34752	49880	51009	52503	53249	4,59	65,62%
8	1	2000x2000	[0,2]	[-1.5,1.5]	40153	40270	40328	40699	41383	3,98	49,69%
9	1	2000x2000	[0,2]	[-1.5,1.5]	40112	40144	40775	41062	41841	3,98	44,22%
10	1	2000x2000	[0,2]	[-1.5,1.5]	33860	34238	34530	34544	36516	4,71	47,14%
11	1	2000x2000	[0,2]	[-1.5,1.5]	33491	33716	33952	34318	34406	4,77	43,33%
12	1	2000x2000	[0,2]	[-1.5,1.5]	29819	29932	29957	30081	33020	5,35	44,61%
13	1	2000x2000	[0,2]	[-1.5,1.5]	29098	29432	29710	30981	32476	5,49	42,20%
14	1	2000x2000	[0,2]	[-1.5,1.5]	26181	26341	26622	28774	30147	6,10	43,55%
15	1	2000x2000	[0,2]	[-1.5,1.5]	25512	25650	25723	25724	27168	6,26	41,71%
16	1	2000x2000	[0,2]	[-1.5,1.5]	23363	23365	23440	23697	25892	6,83	42,70%
17	1	2000x2000	[0,2]	[-1.5,1.5]	22367	22575	22593	22991	23054	7,14	41,98%
18	1	2000x2000	[0,2]	[-1.5,1.5]	20829	21245	21343	21448	22521	7,66	42,58%
19	1	2000x2000	[0,2]	[-1.5,1.5]	20409	20606	20831	20869	20969	7,82	41,17%
20	1	2000x2000	[0,2]	[-1.5,1.5]	19074	19155	19230	19570	19594	8,37	41,85%
21	1	2000x2000	[0,2]	[-1.5,1.5]	18749	18868	18902	18966	19148	8,51	40,54%
22	1	2000x2000	[0,2]	[-1.5,1.5]	17578	17787	17810	17846	18264	9,08	41,28%
23	1	2000x2000	[0,2]	[-1.5,1.5]	17404	17508	17550	17611	17633	9,17	39,88%
24	1	2000x2000	[0,2]	[-1.5,1.5]	16380	16749	16777	16780	17196	9,75	40,61%
25	1	2000x2000	[0,2]	[-1.5,1.5]	16200	16241	16323	16631	16815	9,85	39,42%
26	1	2000x2000	[0,2]	[-1.5,1.5]	15268	15480	15575	15758	15986	10,46	40,21%
27	1	2000x2000	[0,2]	[-1.5,1.5]	15121	15248	15346	15626	15637	10,56	39,10%
28	1	2000x2000	[0,2]	[-1.5,1.5]	14785	14823	14858	15043	15094	10,80	38,56%
29	1	2000x2000	[0,2]	[-1.5,1.5]	14760	14805	14887	15081	15085	10,82	37,29%
30	1	2000x2000	[0,2]	[-1.5,1.5]	13995	14009	14133	14188	14588	11,41	38,02%
31	1	2000x2000	[0,2]	[-1.5,1.5]	13821	14057	14069	14392	14482	11,55	37,26%
32	1	2000x2000	[0,2]	[-1.5,1.5]	13624	13702	13739	13829	13856	11,72	36,62%

Фигура 16: Тазблица с резултати от проведените тестове за g = 1

2) Тестване с g = 4

p	g	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	4	2000x2000	[0,2]	[-1.5,1.5]	158800	159289	160675	162572	168925	1,00	100,00%
2	4	2000x2000	[0,2]	[-1.5,1.5]	80844	81398	81543	84126	85195	1,96	98,21%
3	4	2000x2000	[0,2]	[-1.5,1.5]	54556	58924	59135	61070	64235	2,91	97,03%
4	4	2000x2000	[0,2]	[-1.5,1.5]	45353	45735	45834	46425	48273	3,50	87,54%
5	4	2000x2000	[0,2]	[-1.5,1.5]	35602	35646	35865	37432	37475	4,46	89,21%
6	4	2000x2000	[0,2]	[-1.5,1.5]	29860	29974	30003	32040	32638	5,32	88,64%
7	4	2000x2000	[0,2]	[-1.5,1.5]	26074	26276	27593	27836	29275	6,09	87,01%
8	4	2000x2000	[0,2]	[-1.5,1.5]	23596	23684	23831	26493	26844	6,73	84,12%
9	4	2000x2000	[0,2]	[-1.5,1.5]	21005	21011	21198	21284	22646	7,56	84,00%
10	4	2000x2000	[0,2]	[-1.5,1.5]	18897	18939	19023	19081	19280	8,40	84,03%
11	4	2000x2000	[0,2]	[-1.5,1.5]	17195	17395	17468	17495	17565	9,24	83,96%
12	4	2000x2000	[0,2]	[-1.5,1.5]	16102	16116	16274	16291	16549	9,86	82,18%
13	4	2000x2000	[0,2]	[-1.5,1.5]	15124	15227	15271	15344	16264	10,50	80,77%
14	4	2000x2000	[0,2]	[-1.5,1.5]	14128	14158	14165	14219	14444	11,24	80,29%
15	4	2000x2000	[0,2]	[-1.5,1.5]	13190	13205	13261	13298	13338	12,04	80,26%
16	4	2000x2000	[0,2]	[-1.5,1.5]	12650	12698	12704	12803	12913	12,55	78,46%
17	4	2000x2000	[0,2]	[-1.5,1.5]	12134	12345	12563	12584	12679	13,09	76,98%
18	4	2000x2000	[0,2]	[-1.5,1.5]	11819	11919	12026	12078	12145	13,44	74,64%
19	4	2000x2000	[0,2]	[-1.5,1.5]	11595	11728	11760	11793	11827	13,70	72,08%
20	4	2000x2000	[0,2]	[-1.5,1.5]	11102	11382	11433	11481	11567	14,30	71,52%
21	4	2000x2000	[0,2]	[-1.5,1.5]	11159	11172	11275	11315	11335	14,23	67,77%
22	4	2000x2000	[0,2]	[-1.5,1.5]	10620	10709	10861	10868	10954	14,95	67,97%
23	4	2000x2000	[0,2]	[-1.5,1.5]	10426	10649	10757	10873	10887	15,23	66,22%
24	4	2000x2000	[0,2]	[-1.5,1.5]	10189	10318	10368	10532	10582	15,59	64,94%
25	4	2000x2000	[0,2]	[-1.5,1.5]	9891	9914	9966	9969	10002	16,05	64,22%
26	4	2000x2000	[0,2]	[-1.5,1.5]	9830	9831	9865	9874	9921	16,15	62,13%
27	4	2000x2000	[0,2]	[-1.5,1.5]	9629	9682	9950	10006	10010	16,49	61,08%
28	4	2000x2000	[0,2]	[-1.5,1.5]	9342	9455	9553	9667	9690	17,00	60,71%
29	4	2000x2000	[0,2]	[-1.5,1.5]	9282	9328	9371	9380	9432	17,11	58,99%
30	4	2000x2000	[0,2]	[-1.5,1.5]	9092	9118	9137	9138	9247	17,47	58,22%
31	4	2000x2000	[0,2]	[-1.5,1.5]	8932	9071	9110	9195	9405	17,78	57,35%
32	4	2000x2000	[0,2]	[-1.5,1.5]	8828	8991	8997	9076	9194	17,99	56,21%

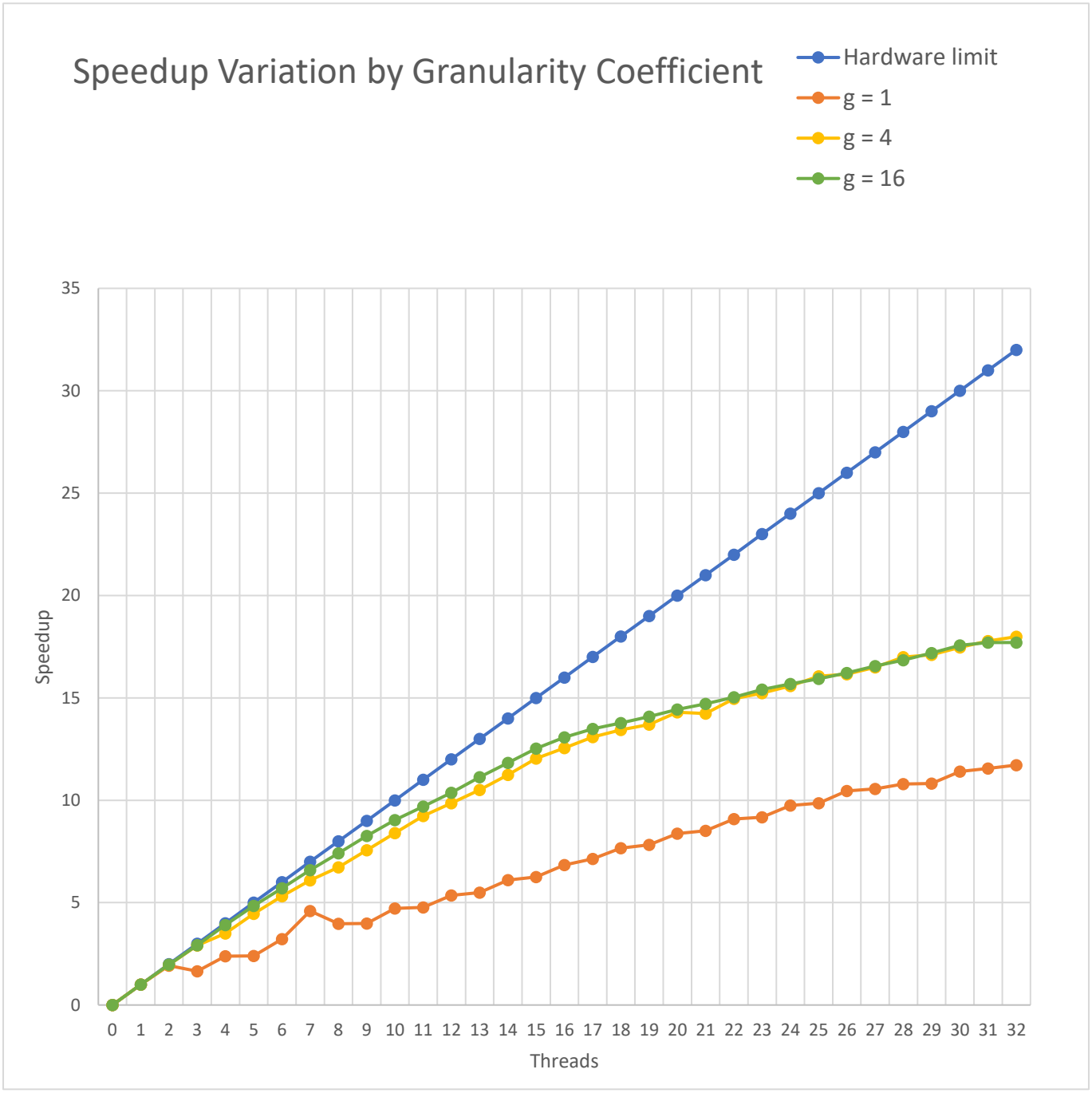
Фигура 17: Тазблица с резултати от проведените тестове за g = 4

3) Тестване с g = 16

p	g	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	16	2000x2000	[0,2]	[-1.5,1.5]	158602	158984	159409	160679	160973	1,00	100,00%
2	16	2000x2000	[0,2]	[-1.5,1.5]	80234	80741	80967	80973	82171	1,98	98,84%
3	16	2000x2000	[0,2]	[-1.5,1.5]	54349	54506	54562	54720	55140	2,92	97,27%
4	16	2000x2000	[0,2]	[-1.5,1.5]	40612	40725	41212	41237	41445	3,91	97,63%
5	16	2000x2000	[0,2]	[-1.5,1.5]	32786	32790	32898	33043	33291	4,84	96,75%
6	16	2000x2000	[0,2]	[-1.5,1.5]	27812	27848	27938	27961	27983	5,70	95,04%
7	16	2000x2000	[0,2]	[-1.5,1.5]	24092	24162	24238	24439	24477	6,58	94,05%
8	16	2000x2000	[0,2]	[-1.5,1.5]	21397	21404	21411	21642	21785	7,41	92,65%
9	16	2000x2000	[0,2]	[-1.5,1.5]	19191	19301	19383	19428	19585	8,26	91,83%
10	16	2000x2000	[0,2]	[-1.5,1.5]	17562	17583	17619	17620	17634	9,03	90,31%
11	16	2000x2000	[0,2]	[-1.5,1.5]	16365	16371	16413	16455	16486	9,69	88,10%
12	16	2000x2000	[0,2]	[-1.5,1.5]	15287	15290	15311	15319	15323	10,37	86,46%
13	16	2000x2000	[0,2]	[-1.5,1.5]	14247	14272	14375	14438	14481	11,13	85,63%
14	16	2000x2000	[0,2]	[-1.5,1.5]	13410	13435	13480	13497	13518	11,83	84,48%
15	16	2000x2000	[0,2]	[-1.5,1.5]	12656	12659	12709	12746	12765	12,53	83,55%
16	16	2000x2000	[0,2]	[-1.5,1.5]	12127	12221	12330	12484	12635	13,08	81,74%
17	16	2000x2000	[0,2]	[-1.5,1.5]	11763	11774	11840	11862	11877	13,48	79,31%
18	16	2000x2000	[0,2]	[-1.5,1.5]	11516	11623	11644	11670	11742	13,77	76,51%
19	16	2000x2000	[0,2]	[-1.5,1.5]	11261	11372	11399	11424	11444	14,08	74,13%
20	16	2000x2000	[0,2]	[-1.5,1.5]	10988	11071	11120	11123	11136	14,43	72,17%
21	16	2000x2000	[0,2]	[-1.5,1.5]	10780	10834	10882	10896	10972	14,71	70,06%
22	16	2000x2000	[0,2]	[-1.5,1.5]	10551	10570	10579	10596	10626	15,03	68,33%
23	16	2000x2000	[0,2]	[-1.5,1.5]	10290	10332	10423	10429	10466	15,41	67,01%
24	16	2000x2000	[0,2]	[-1.5,1.5]	10114	10120	10152	10155	10177	15,68	65,34%
25	16	2000x2000	[0,2]	[-1.5,1.5]	9956	9960	9962	9973	10069	15,93	63,72%
26	16	2000x2000	[0,2]	[-1.5,1.5]	9778	9798	9803	9807	9809	16,22	62,39%
27	16	2000x2000	[0,2]	[-1.5,1.5]	9583	9606	9624	9637	9640	16,55	61,30%
28	16	2000x2000	[0,2]	[-1.5,1.5]	9416	9424	9431	9454	9474	16,84	60,16%
29	16	2000x2000	[0,2]	[-1.5,1.5]	9227	9243	9254	9281	9348	17,19	59,27%
30	16	2000x2000	[0,2]	[-1.5,1.5]	9028	9030	9038	9164	9178	17,57	58,56%
31	16	2000x2000	[0,2]	[-1.5,1.5]	8962	8977	8985	9038	9228	17,70	57,09%
32	16	2000x2000	[0,2]	[-1.5,1.5]	8959	8983	8985	9038	9042	17,70	55,32%

Фигура 18: Тазблица с резултати от проведените тестове за g = 16

4) Сравнителна графика и коментари върху проведените тестове



Фигура 19: Сравнителна графика на трите проведени тестови подгрупи

- Първата подгрупа представя резултата от проблема, срещнат при най-едра възможна грануларност. Този проблем бе разгледан при фаза анализ и чрез тези тестове се уверяваме, че той реално води до лошо ускорение и не бива да се подценява.

- При тестване с по-голям коефициент на грануларност този проблем се решава, защото данните се декомпозират на повече задания, което позволява да се балансира работата на процесите по-ефективно.

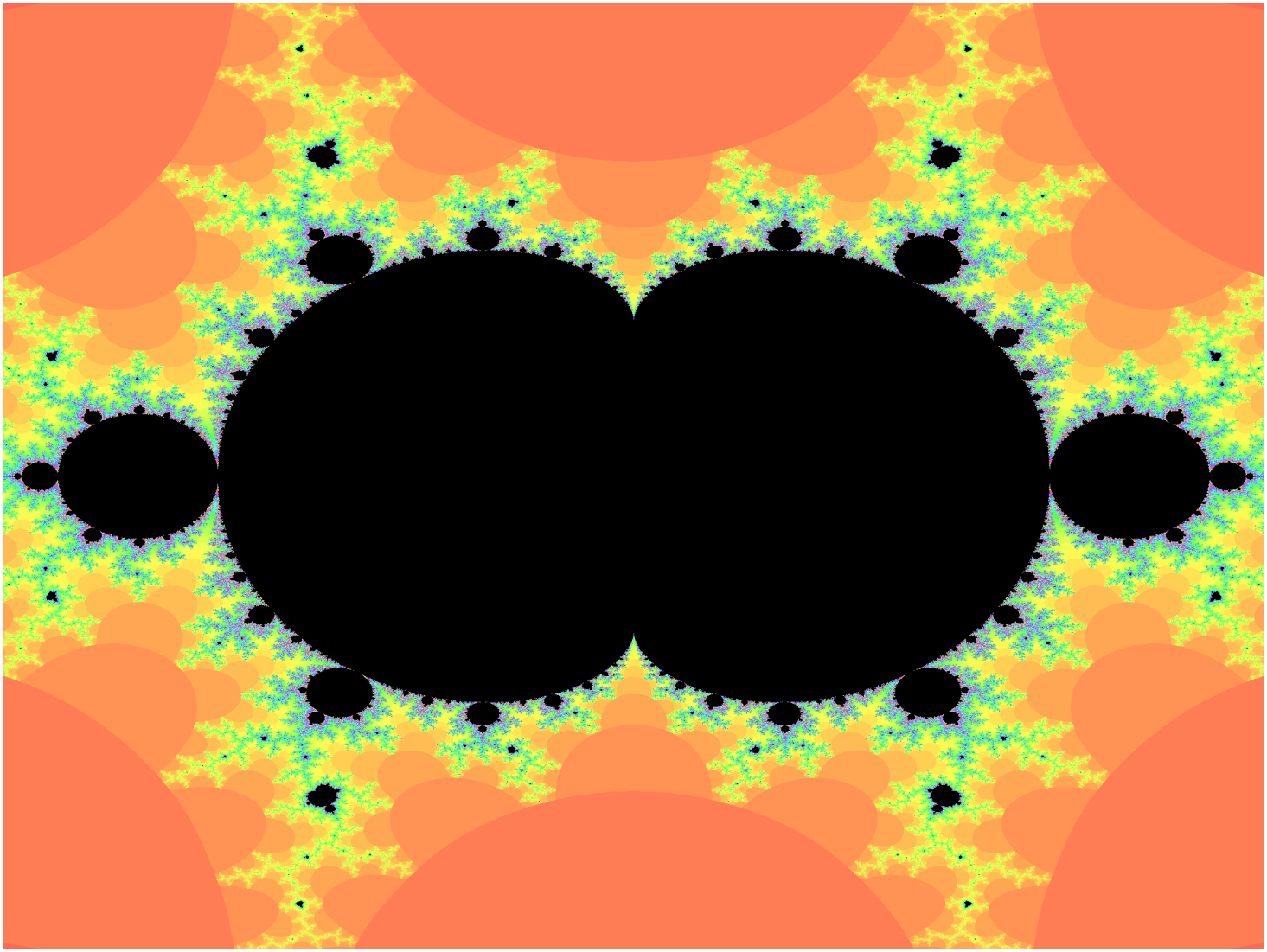
- Не се наблюдава значителна разлика между сериите тестове с g = 4 и g = 16 и даже при тестове с повече от 22 процеса резултатите почти съвпадат. Ако разгледаме и времената за изпълнение на тестовете се вижда, че времената не се променят значително при добавяне на повече процеси. Това показва, че избрания алгоритъм за декомпозиране на данните не е достатъчно гъвкав и ограничава ефективността от паралелната обработка при опит на достигане на по-финна грануларност.

3.5. Тестване с променлив размер на изображението

Втората група от тестове е извършена като се фиксират всички параметри с изключение на брой паралелни процеси и размера на изображението. Избрани са следните параметрите за константи:

- Коефициент на грануларност – достатъчно висока стойност, за да се получи най-финната възможна грануларност на този алгоритъм. Целта е работата да се балансира максимално ефективно, поради което всяко задание трябва да се състои от един ред.
- Сектор на изображението – ‚a' е в интервал [-2, 2] и ‚b' е в интервал [-2, 2]

Като резултат от изпълнението на програмата с тези параметри се получава следното изображение:



Фигура 20: Резултат от изпълнение на програмата за тестване на размер

Тази група от тестове се разделя на три подгрупи, всяка от които има различен размер на изображението. Техните стойности на променливия параметър са:

- Size = 640x480
- Size = 1280x960
- Size = 4096x4096

1) Тестване с размер 640x480

p	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	640x480	[-2,2]	[-2,2]	9144	9154	9179	9292	9346	1,00	100,00%
2	640x480	[-2,2]	[-2,2]	4842	4886	4895	4914	4925	1,89	94,42%
3	640x480	[-2,2]	[-2,2]	3403	3407	3427	3442	3444	2,69	89,57%
4	640x480	[-2,2]	[-2,2]	2652	2662	2666	2680	2690	3,45	86,20%
5	640x480	[-2,2]	[-2,2]	2211	2215	2219	2230	2232	4,14	82,71%
6	640x480	[-2,2]	[-2,2]	1869	1885	1920	1929	1940	4,89	81,54%
7	640x480	[-2,2]	[-2,2]	1673	1676	1678	1684	1698	5,47	78,08%
8	640x480	[-2,2]	[-2,2]	1487	1536	1543	1548	1582	6,15	76,87%
9	640x480	[-2,2]	[-2,2]	1361	1376	1379	1396	1405	6,72	74,65%
10	640x480	[-2,2]	[-2,2]	1260	1286	1292	1296	1309	7,26	72,57%
11	640x480	[-2,2]	[-2,2]	1200	1202	1205	1215	1227	7,62	69,27%
12	640x480	[-2,2]	[-2,2]	1122	1154	1179	1189	1191	8,15	67,91%
13	640x480	[-2,2]	[-2,2]	1088	1090	1090	1092	1137	8,40	64,65%
14	640x480	[-2,2]	[-2,2]	1022	1031	1049	1067	1075	8,95	63,91%
15	640x480	[-2,2]	[-2,2]	971	1016	1049	1055	1082	9,42	62,78%
16	640x480	[-2,2]	[-2,2]	964	988	995	996	1012	9,49	59,28%
17	640x480	[-2,2]	[-2,2]	952	956	985	992	993	9,61	56,50%
18	640x480	[-2,2]	[-2,2]	928	929	969	980	986	9,85	54,74%
19	640x480	[-2,2]	[-2,2]	905	908	911	920	923	10,10	53,18%
20	640x480	[-2,2]	[-2,2]	892	899	908	913	916	10,25	51,26%
21	640x480	[-2,2]	[-2,2]	863	869	884	885	891	10,60	50,46%
22	640x480	[-2,2]	[-2,2]	859	861	863	864	865	10,64	48,39%
23	640x480	[-2,2]	[-2,2]	837	842	844	845	855	10,92	47,50%
24	640x480	[-2,2]	[-2,2]	825	828	831	845	847	11,08	46,18%
25	640x480	[-2,2]	[-2,2]	835	838	840	841	841	10,95	43,80%
26	640x480	[-2,2]	[-2,2]	810	824	825	826	828	11,29	43,42%
27	640x480	[-2,2]	[-2,2]	795	801	811	811	815	11,50	42,60%
28	640x480	[-2,2]	[-2,2]	773	780	798	803	805	11,83	42,25%
29	640x480	[-2,2]	[-2,2]	758	786	789	794	799	12,06	41,60%
30	640x480	[-2,2]	[-2,2]	756	756	758	776	782	12,10	40,32%
31	640x480	[-2,2]	[-2,2]	748	767	776	779	779	12,22	39,43%
32	640x480	[-2,2]	[-2,2]	778	780	780	790	795	11,75	36,73%

Фигура 21: Тазблица с резултати от проведените тестове за размер 640x480

2) Тестване с размер 1280x960

p	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	1280x960	[-2,2]	[-2,2]	34864	35228	37465	38561	38854	1,00	100,00%
2	1280x960	[-2,2]	[-2,2]	18339	18340	18641	18843	18951	1,90	95,05%
3	1280x960	[-2,2]	[-2,2]	12819	12904	13083	13182	13209	2,72	90,66%
4	1280x960	[-2,2]	[-2,2]	10037	10041	10047	10083	10170	3,47	86,84%
5	1280x960	[-2,2]	[-2,2]	8241	8322	8357	8408	8532	4,23	84,61%
6	1280x960	[-2,2]	[-2,2]	6926	6940	7046	7063	7098	5,03	83,90%
7	1280x960	[-2,2]	[-2,2]	6021	6157	6176	6184	6201	5,79	82,72%
8	1280x960	[-2,2]	[-2,2]	5465	5515	5521	5522	5531	6,38	79,74%
9	1280x960	[-2,2]	[-2,2]	4867	4919	4973	4990	5011	7,16	79,59%
10	1280x960	[-2,2]	[-2,2]	4480	4504	4545	4559	4576	7,78	77,82%
11	1280x960	[-2,2]	[-2,2]	4104	4126	4246	4322	4367	8,50	77,23%
12	1280x960	[-2,2]	[-2,2]	3879	3885	3898	3926	3951	8,99	74,90%
13	1280x960	[-2,2]	[-2,2]	3612	3624	3650	3674	3725	9,65	74,25%
14	1280x960	[-2,2]	[-2,2]	3401	3484	3505	3578	3579	10,25	73,22%
15	1280x960	[-2,2]	[-2,2]	3295	3322	3336	3366	3413	10,58	70,54%
16	1280x960	[-2,2]	[-2,2]	3050	3079	3101	3180	3232	11,43	71,44%
17	1280x960	[-2,2]	[-2,2]	3033	3041	3064	3071	3137	11,49	67,62%
18	1280x960	[-2,2]	[-2,2]	2988	2991	3008	3012	3036	11,67	64,82%
19	1280x960	[-2,2]	[-2,2]	2921	2926	2968	2983	2996	11,94	62,82%
20	1280x960	[-2,2]	[-2,2]	2775	2847	2862	2863	2881	12,56	62,82%
21	1280x960	[-2,2]	[-2,2]	2763	2800	2802	2813	2814	12,62	60,09%
22	1280x960	[-2,2]	[-2,2]	2696	2704	2742	2750	2755	12,93	58,78%
23	1280x960	[-2,2]	[-2,2]	2681	2684	2689	2690	2693	13,00	56,54%
24	1280x960	[-2,2]	[-2,2]	2603	2605	2611	2614	2624	13,39	55,81%
25	1280x960	[-2,2]	[-2,2]	2543	2561	2585	2586	2589	13,71	54,84%
26	1280x960	[-2,2]	[-2,2]	2501	2514	2535	2536	2547	13,94	53,62%
27	1280x960	[-2,2]	[-2,2]	2445	2465	2482	2511	2517	14,26	52,81%
28	1280x960	[-2,2]	[-2,2]	2425	2439	2442	2446	2481	14,38	51,35%
29	1280x960	[-2,2]	[-2,2]	2394	2403	2405	2420	2435	14,56	50,22%
30	1280x960	[-2,2]	[-2,2]	2379	2389	2404	2407	2408	14,65	48,85%
31	1280x960	[-2,2]	[-2,2]	2374	2385	2391	2397	2406	14,69	47,37%
32	1280x960	[-2,2]	[-2,2]	2358	2365	2369	2374	2382	14,79	46,20%

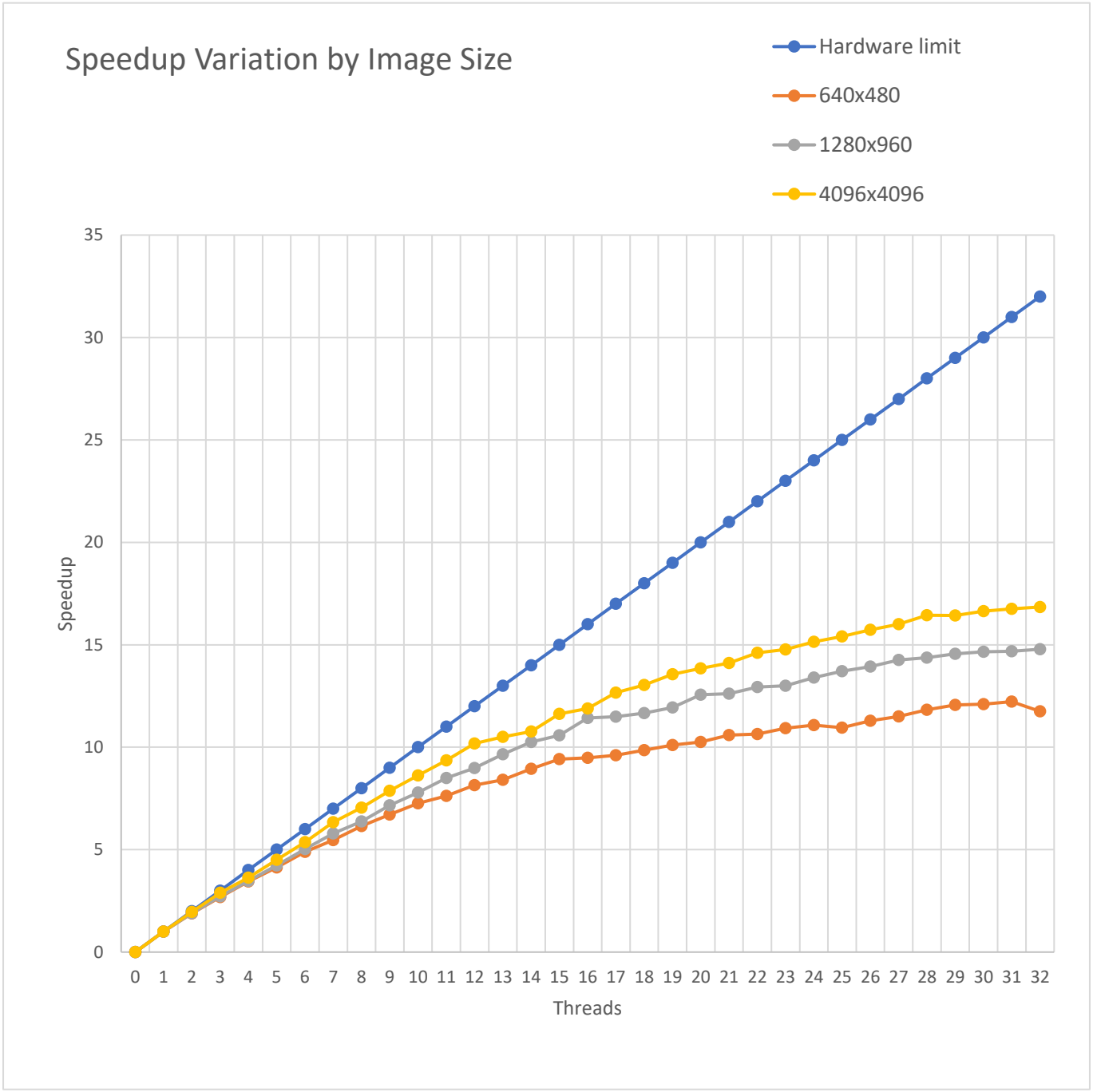
Фигура 22: Тазблица с резултати от проведените тестове за размер 1280x960

3) Тестване с размер 4096x4096

p	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	4096x4096	[-2,2]	[-2,2]	481243	483803	515559	517070	525533	1,00	100,00%
2	4096x4096	[-2,2]	[-2,2]	245796	255253	266790	268001	273638	1,96	97,89%
3	4096x4096	[-2,2]	[-2,2]	166139	170373	176554	177608	178617	2,90	96,55%
4	4096x4096	[-2,2]	[-2,2]	132503	133393	133407	134671	135508	3,63	90,80%
5	4096x4096	[-2,2]	[-2,2]	106869	114253	109203	112237	108704	4,50	90,06%
6	4096x4096	[-2,2]	[-2,2]	89671	90300	90501	90960	91547	5,37	89,45%
7	4096x4096	[-2,2]	[-2,2]	75918	77051	77580	79104	80223	6,34	90,56%
8	4096x4096	[-2,2]	[-2,2]	68313	69437	70117	70466	70710	7,04	88,06%
9	4096x4096	[-2,2]	[-2,2]	61170	61367	61564	61936	62794	7,87	87,41%
10	4096x4096	[-2,2]	[-2,2]	55842	55844	55945	56303	56491	8,62	86,18%
11	4096x4096	[-2,2]	[-2,2]	51446	51660	51742	51996	52409	9,35	85,04%
12	4096x4096	[-2,2]	[-2,2]	47268	48240	48410	48609	48897	10,18	84,84%
13	4096x4096	[-2,2]	[-2,2]	45821	46712	46871	47306	47367	10,50	80,79%
14	4096x4096	[-2,2]	[-2,2]	44692	45088	45092	45100	45587	10,77	76,91%
15	4096x4096	[-2,2]	[-2,2]	41393	42374	43502	43543	43736	11,63	77,51%
16	4096x4096	[-2,2]	[-2,2]	40473	40816	40968	41461	41785	11,89	74,32%
17	4096x4096	[-2,2]	[-2,2]	37991	38588	39199	39387	39536	12,67	74,51%
18	4096x4096	[-2,2]	[-2,2]	36913	37311	37353	37472	37491	13,04	72,43%
19	4096x4096	[-2,2]	[-2,2]	35504	35687	36049	36283	36314	13,55	71,34%
20	4096x4096	[-2,2]	[-2,2]	34762	35028	35034	35065	35371	13,84	69,22%
21	4096x4096	[-2,2]	[-2,2]	34106	34127	34194	34366	34437	14,11	67,19%
22	4096x4096	[-2,2]	[-2,2]	32947	33190	33247	33269	33558	14,61	66,39%
23	4096x4096	[-2,2]	[-2,2]	32579	32586	32604	32807	32845	14,77	64,22%
24	4096x4096	[-2,2]	[-2,2]	31775	31896	32169	32234	32316	15,15	63,11%
25	4096x4096	[-2,2]	[-2,2]	31227	31375	31407	31441	31484	15,41	61,64%
26	4096x4096	[-2,2]	[-2,2]	30597	30863	30934	31124	31124	15,73	60,49%
27	4096x4096	[-2,2]	[-2,2]	30062	30222	30258	30329	30581	16,01	59,29%
28	4096x4096	[-2,2]	[-2,2]	29263	29771	29915	29956	30072	16,45	58,73%
29	4096x4096	[-2,2]	[-2,2]	29292	29448	29476	29602	29757	16,43	56,65%
30	4096x4096	[-2,2]	[-2,2]	28920	29118	29316	29317	29350	16,64	55,47%
31	4096x4096	[-2,2]	[-2,2]	28721	28818	28946	29022	29108	16,76	54,05%
32	4096x4096	[-2,2]	[-2,2]	28567	28656	28708	28853	29186	16,85	52,64%

Фигура 23: Тазблица с резултати от проведените тестове за размер 4096x4096

4) Сравнителна графика и коментари върху проведените тестове



Фигура 24: Сравнителна графика на трите проведени тестови подгрупи

- В първата подгрупа не се получава добро ускорение, защото програмата приключва твърде бързо и не е възможно точно да се определи колко ефективно ще е паралелната обработка.

- Наблюдава се подобрение на ускорението в следващите две подгрупи. Колкото по-голямо е изображението, толкова повече време е нужно за неговото изчисление. А колкото повече време е нужно за изпълнението на програмата, толкова по-ясно ще се прояви ефекта на ускорението.

- Третата подгрупа представя най-голямото изображение от трите подгрупи, поради което изисква най-много време за изчисление и получава най-добро ускорение като резултат.

3.6. Тестване с променлив сектор на изображението

Последната група от тестове е извършена като се фиксират всички параметри с изключение на брой паралелни процеси и сектора на изображението. Избрани са следните параметрите за константи:

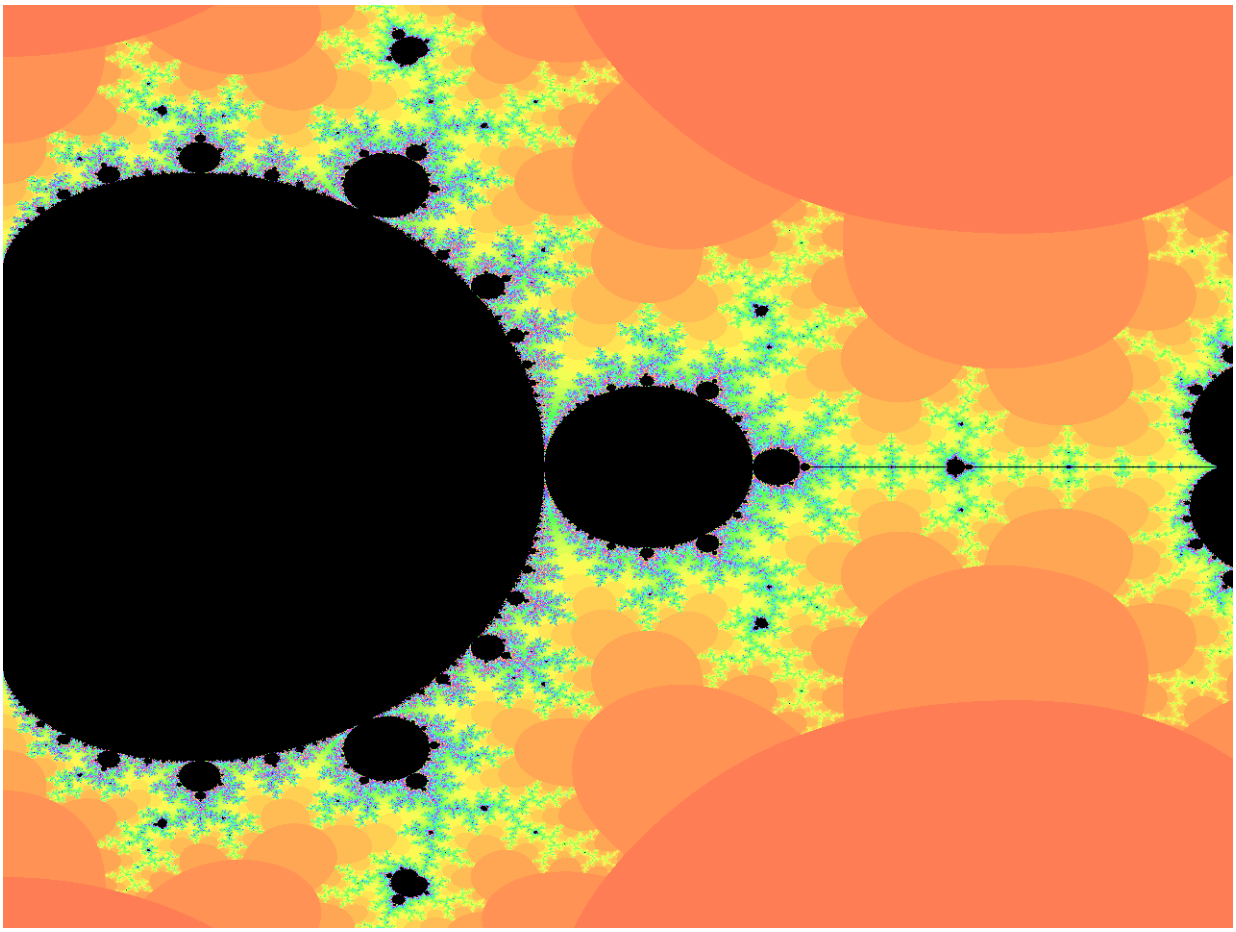
- Коефициент на грануларност – достатъчно висока стойност, за да се получи най-финната възможна грануларност на този алгоритъм. Целта е работата да се балансира максимално ефективно, поради което всяко задание трябва да се състои от един ред.
- Размер на изображението - 1280x960

Тази група от тестове се разделя на три подгрупи, всяка от които генерира изображението в различен сектор. Следователно всяка подгрупа ще получи различно крайно изображение.

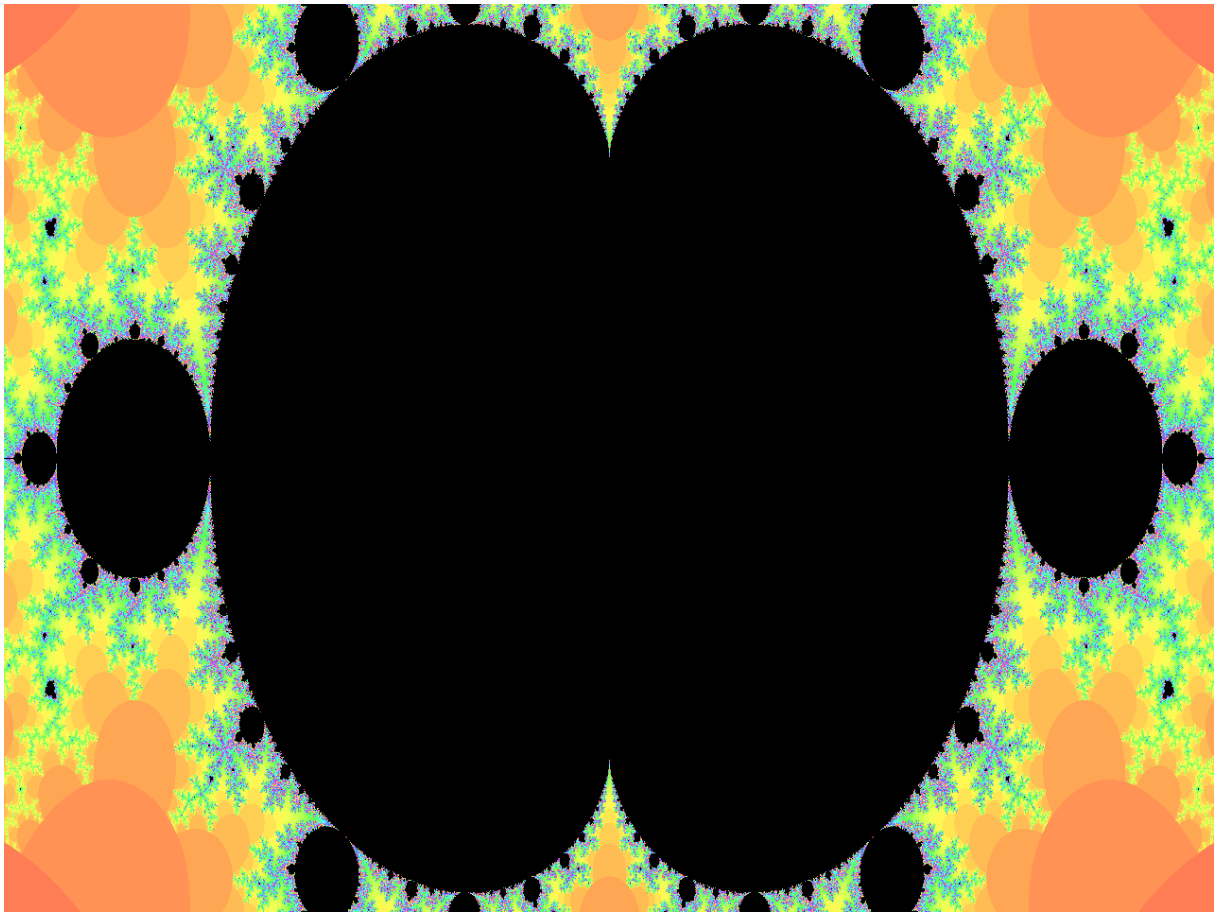
Стойностите на променливия параметър са:

- Сектор с граници ‚a‘ да е в интервал [0,3] и ‚b‘ да е в интервал [-1.5,1.5]
- Сектор с граници ‚a‘ да е в интервал [-2,2] и ‚b‘ да е в интервал [-1,1]
- Сектор с граници ‚a‘ да е в интервал [-1,1] и ‚b‘ да е в интервал [-1,1]

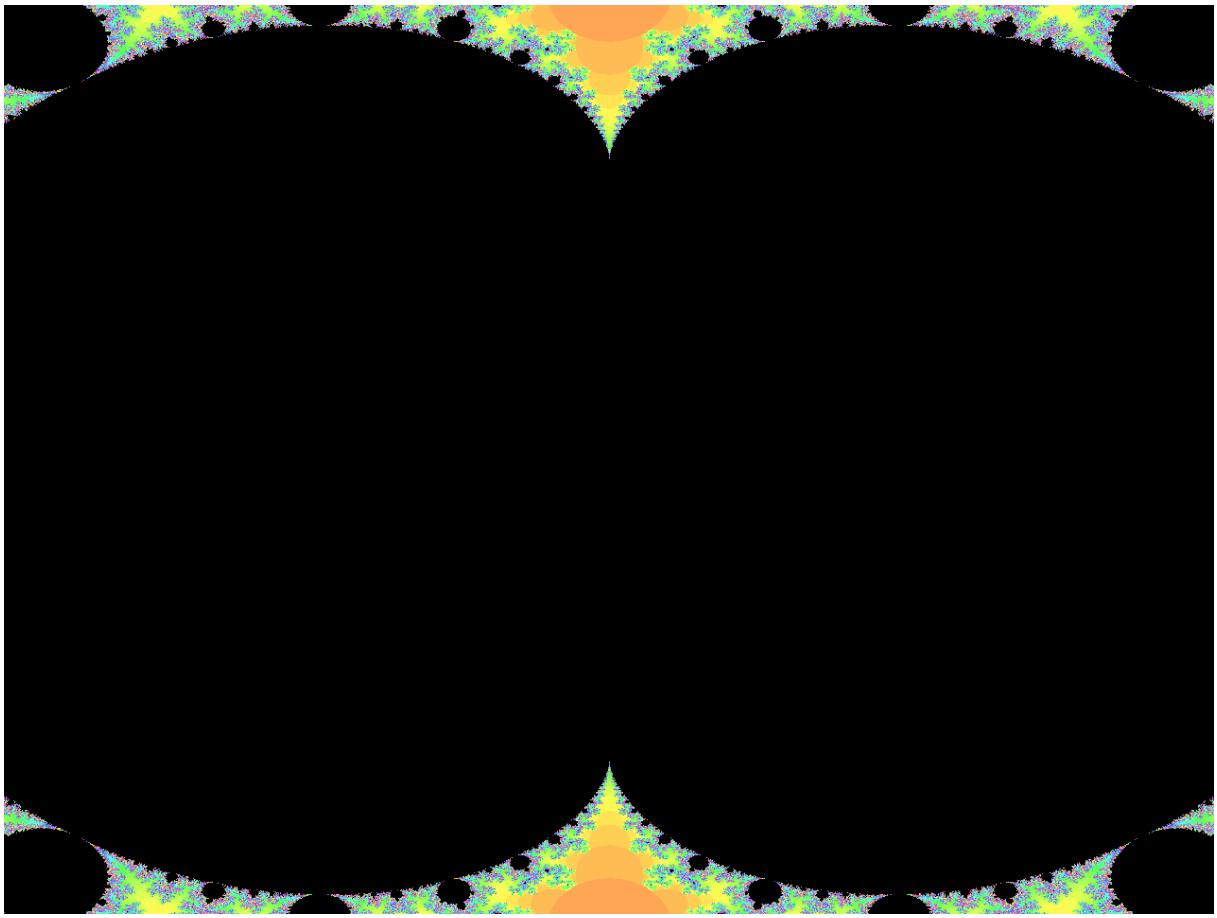
Генерираните изображения имат следния вид:



Фигура 25: Резултат от изпълнение на програмата с граници ‚a‘ да е в интервал [0,3] и ‚b‘ да е в интервал [-1.5,1.5]



Фигура 26: Резултат от изпълнение на програмата с граници ‚a‘ да е в интервал [-2,2] и ‚b‘ да е в интервал [-1,1]



Фигура 27: Резултат от изпълнение на програмата с граници ‚a‘ да е в интервал [-1,1] и ‚b‘ да е в интервал [-1,1]

1) Тестване на сектор с граници ‚a‘ да е в интервал [0,3] и ‚b‘ да е в интервал [-1.5,1.5]

0	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	1280x960	[0,3]	[-1.5,1.5]	30744	31118	31241	31264	31265	1,00	100,00%
2	1280x960	[0,3]	[-1.5,1.5]	16169	16344	16361	16530	16532	1,90	95,07%
3	1280x960	[0,3]	[-1.5,1.5]	11144	11217	11248	11285	11302	2,76	91,96%
4	1280x960	[0,3]	[-1.5,1.5]	8542	8625	8686	8727	8738	3,60	89,98%
5	1280x960	[0,3]	[-1.5,1.5]	7036	7095	7100	7121	7127	4,37	87,39%
6	1280x960	[0,3]	[-1.5,1.5]	5988	6052	6055	6066	6082	5,13	85,57%
7	1280x960	[0,3]	[-1.5,1.5]	5257	5261	5264	5267	5295	5,85	83,55%
8	1280x960	[0,3]	[-1.5,1.5]	4699	4718	4749	4793	4793	6,54	81,78%
9	1280x960	[0,3]	[-1.5,1.5]	4232	4247	4249	4251	4267	7,26	80,72%
10	1280x960	[0,3]	[-1.5,1.5]	3858	3875	3879	3890	3909	7,97	79,69%
11	1280x960	[0,3]	[-1.5,1.5]	3580	3586	3618	3627	3630	8,59	78,07%
12	1280x960	[0,3]	[-1.5,1.5]	3394	3406	3436	3438	3483	9,06	75,49%
13	1280x960	[0,3]	[-1.5,1.5]	3166	3201	3225	3244	3249	9,71	74,70%
14	1280x960	[0,3]	[-1.5,1.5]	2954	2973	3020	3021	3155	10,41	74,34%
15	1280x960	[0,3]	[-1.5,1.5]	2868	2891	2971	3025	3074	10,72	71,46%
16	1280x960	[0,3]	[-1.5,1.5]	2832	2857	2857	2861	2863	10,86	67,85%
17	1280x960	[0,3]	[-1.5,1.5]	2653	2720	2751	2762	2778	11,59	68,17%
18	1280x960	[0,3]	[-1.5,1.5]	2632	2674	2730	2798	2859	11,68	64,89%
19	1280x960	[0,3]	[-1.5,1.5]	2564	2566	2569	2611	2622	11,99	63,11%
20	1280x960	[0,3]	[-1.5,1.5]	2480	2489	2499	2499	2518	12,40	61,98%
21	1280x960	[0,3]	[-1.5,1.5]	2462	2470	2497	2502	2504	12,49	59,46%
22	1280x960	[0,3]	[-1.5,1.5]	2373	2401	2415	2421	2422	12,96	58,89%
23	1280x960	[0,3]	[-1.5,1.5]	2313	2314	2341	2343	2350	13,29	57,79%
24	1280x960	[0,3]	[-1.5,1.5]	2298	2313	2313	2331	2336	13,38	55,74%
25	1280x960	[0,3]	[-1.5,1.5]	2232	2273	2278	2290	2298	13,77	55,10%
26	1280x960	[0,3]	[-1.5,1.5]	2231	2231	2233	2243	2256	13,78	53,00%
27	1280x960	[0,3]	[-1.5,1.5]	2150	2196	2208	2210	2214	14,30	52,96%
28	1280x960	[0,3]	[-1.5,1.5]	2110	2126	2154	2181	2182	14,57	52,04%
29	1280x960	[0,3]	[-1.5,1.5]	2081	2081	2094	2123	2147	14,77	50,94%
30	1280x960	[0,3]	[-1.5,1.5]	2084	2144	2147	2152	2157	14,75	49,17%
31	1280x960	[0,3]	[-1.5,1.5]	2048	2058	2078	2078	2123	15,01	48,42%
32	1280x960	[0,3]	[-1.5,1.5]	2100	2128	2128	2148	2150	14,64	45,75%

Фигура 28: Тазблица с резултати от проведените тестове за сектор

2) Тестване на сектор с граници ‚a‘ да е в интервал [-2,2] и ‚b‘ да е в интервал [-1,1]

p	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	1280x960	[-2,2]	[-1,1]	68000	68027	68421	68451	68654	1,00	100,00%
2	1280x960	[-2,2]	[-1,1]	35300	35588	35631	35659	35664	1,93	96,32%
3	1280x960	[-2,2]	[-1,1]	23840	24079	24149	24163	24269	2,85	95,08%
4	1280x960	[-2,2]	[-1,1]	18087	18340	18392	18450	18473	3,76	93,99%
5	1280x960	[-2,2]	[-1,1]	14779	14888	15074	15139	15191	4,60	92,02%
6	1280x960	[-2,2]	[-1,1]	12530	12569	12632	12653	12670	5,43	90,45%
7	1280x960	[-2,2]	[-1,1]	10772	10784	10798	10838	10852	6,31	90,18%
8	1280x960	[-2,2]	[-1,1]	9443	9548	9564	9604	9720	7,20	90,01%
9	1280x960	[-2,2]	[-1,1]	8525	8535	8560	8567	8577	7,98	88,63%
10	1280x960	[-2,2]	[-1,1]	7737	7747	7758	7759	7794	8,79	87,89%
11	1280x960	[-2,2]	[-1,1]	7094	7132	7134	7137	7137	9,59	87,14%
12	1280x960	[-2,2]	[-1,1]	6633	6652	6707	6717	6737	10,25	85,43%
13	1280x960	[-2,2]	[-1,1]	6161	6176	6181	6225	6241	11,04	84,90%
14	1280x960	[-2,2]	[-1,1]	5761	5763	5785	5825	5846	11,80	84,31%
15	1280x960	[-2,2]	[-1,1]	5722	5755	5772	5784	5800	11,88	79,23%
16	1280x960	[-2,2]	[-1,1]	5278	5323	5403	5438	5521	12,88	80,52%
17	1280x960	[-2,2]	[-1,1]	5045	5074	5099	5211	5222	13,48	79,29%
18	1280x960	[-2,2]	[-1,1]	5009	5086	5136	5146	5169	13,58	75,42%
19	1280x960	[-2,2]	[-1,1]	4802	4957	4972	5012	5022	14,16	74,53%
20	1280x960	[-2,2]	[-1,1]	4781	4863	4865	4885	4893	14,22	71,11%
21	1280x960	[-2,2]	[-1,1]	4598	4728	4772	4778	4782	14,79	70,42%
22	1280x960	[-2,2]	[-1,1]	4456	4521	4664	4666	4683	15,26	69,37%
23	1280x960	[-2,2]	[-1,1]	4402	4428	4576	4584	4584	15,45	67,16%
24	1280x960	[-2,2]	[-1,1]	4325	4471	4504	4505	4507	15,72	65,51%
25	1280x960	[-2,2]	[-1,1]	4379	4391	4399	4431	4431	15,53	62,11%
26	1280x960	[-2,2]	[-1,1]	4127	4295	4318	4349	4352	16,48	63,37%
27	1280x960	[-2,2]	[-1,1]	4044	4071	4089	4255	4256	16,82	62,28%
28	1280x960	[-2,2]	[-1,1]	4000	4119	4185	4195	4229	17,00	60,71%
29	1280x960	[-2,2]	[-1,1]	3906	3955	4124	4128	4144	17,41	60,03%
30	1280x960	[-2,2]	[-1,1]	3840	3864	3874	3920	4064	17,71	59,03%
31	1280x960	[-2,2]	[-1,1]	4005	4029	4030	4031	4064	16,98	54,77%
32	1280x960	[-2,2]	[-1,1]	3806	3964	4006	4017	4019	17,87	55,83%

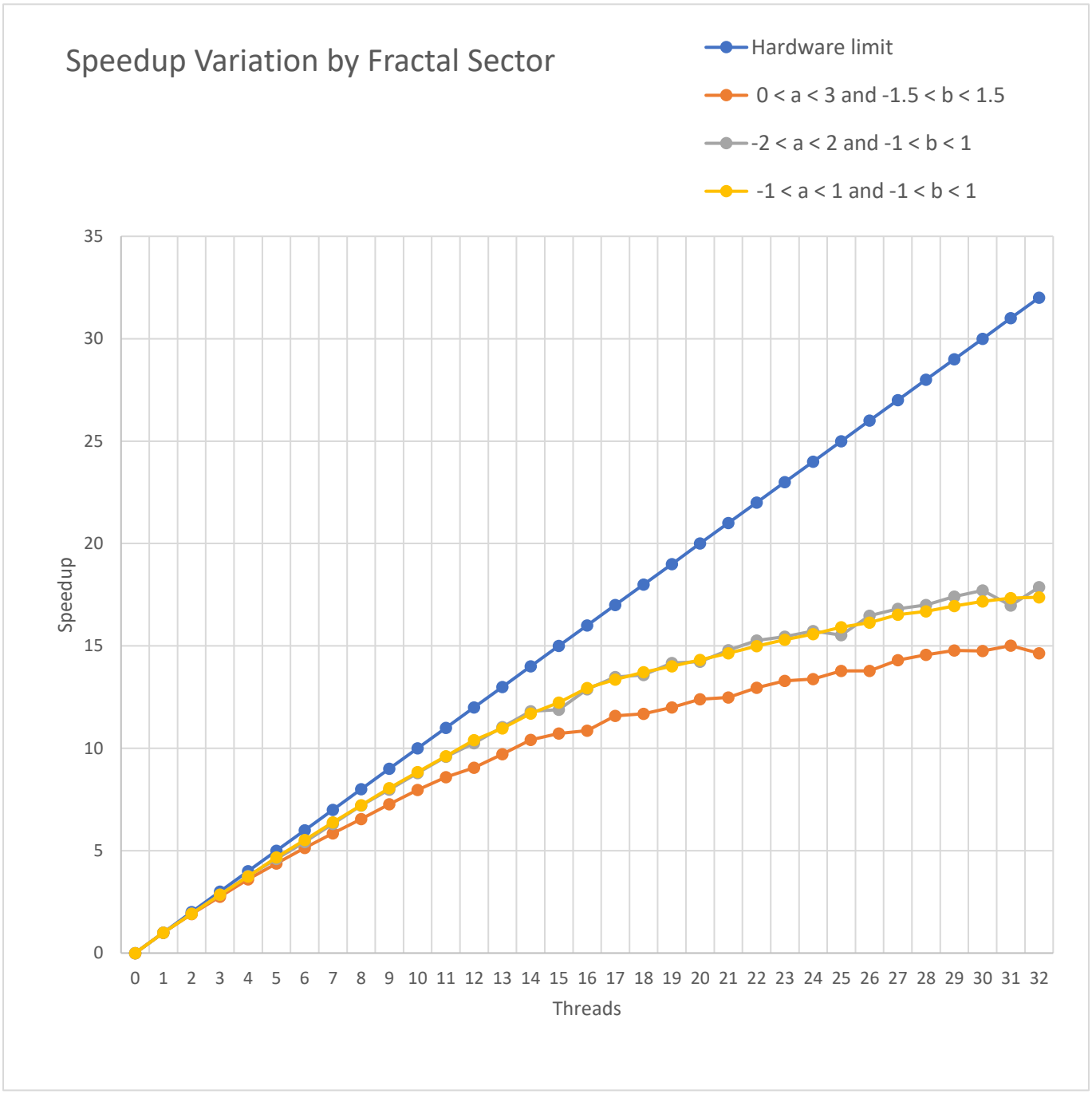
Фигура 29: Тазблица с резултати от проведените тестове за сектор

3) Тестване на сектор с граници ‚a‘ да е в интервал [-1,1] и ‚b‘ да е в интервал [-1,1]

p	Size	a	b	Test 1	Test 2	Test 3	Test 4	Test 5	Sp (Test 1)	Ep
1	1280x960	[-1,1]	[-1,1]	100403	100756	100795	101621	101904	1,00	100,00%
2	1280x960	[-1,1]	[-1,1]	52232	52251	52347	52400	52621	1,92	96,11%
3	1280x960	[-1,1]	[-1,1]	35120	35544	35605	35617	35702	2,86	95,30%
4	1280x960	[-1,1]	[-1,1]	26875	27100	27219	27261	27267	3,74	93,40%
5	1280x960	[-1,1]	[-1,1]	21498	21930	21947	21953	21996	4,67	93,41%
6	1280x960	[-1,1]	[-1,1]	18169	18262	18265	18285	18430	5,53	92,10%
7	1280x960	[-1,1]	[-1,1]	15739	15812	15849	15860	15881	6,38	91,13%
8	1280x960	[-1,1]	[-1,1]	13911	13953	14091	14149	14231	7,22	90,22%
9	1280x960	[-1,1]	[-1,1]	12471	12494	12520	12553	12608	8,05	89,45%
10	1280x960	[-1,1]	[-1,1]	11362	11426	11428	11444	11462	8,84	88,37%
11	1280x960	[-1,1]	[-1,1]	10443	10478	10485	10486	10521	9,61	87,40%
12	1280x960	[-1,1]	[-1,1]	9653	9714	9740	9742	9822	10,40	86,68%
13	1280x960	[-1,1]	[-1,1]	9150	9160	9181	9194	9196	10,97	84,41%
14	1280x960	[-1,1]	[-1,1]	8585	8591	8632	8652	8772	11,70	83,54%
15	1280x960	[-1,1]	[-1,1]	8208	8210	8356	8434	8467	12,23	81,55%
16	1280x960	[-1,1]	[-1,1]	7756	7835	7923	7931	7970	12,95	80,91%
17	1280x960	[-1,1]	[-1,1]	7514	7536	7645	7712	7765	13,36	78,60%
18	1280x960	[-1,1]	[-1,1]	7321	7369	7428	7428	7430	13,71	76,19%
19	1280x960	[-1,1]	[-1,1]	7168	7175	7175	7196	7215	14,01	73,72%
20	1280x960	[-1,1]	[-1,1]	7012	7014	7014	7044	7066	14,32	71,59%
21	1280x960	[-1,1]	[-1,1]	6860	6867	6875	6879	6885	14,64	69,70%
22	1280x960	[-1,1]	[-1,1]	6699	6721	6726	6738	6740	14,99	68,13%
23	1280x960	[-1,1]	[-1,1]	6560	6575	6575	6594	6603	15,31	66,54%
24	1280x960	[-1,1]	[-1,1]	6445	6451	6483	6501	6501	15,58	64,91%
25	1280x960	[-1,1]	[-1,1]	6309	6328	6352	6361	6372	15,91	63,66%
26	1280x960	[-1,1]	[-1,1]	6224	6229	6231	6238	6249	16,13	62,04%
27	1280x960	[-1,1]	[-1,1]	6075	6113	6123	6124	6129	16,53	61,21%
28	1280x960	[-1,1]	[-1,1]	6015	6019	6029	6032	6041	16,69	59,61%
29	1280x960	[-1,1]	[-1,1]	5923	5940	5941	5950	5951	16,95	58,45%
30	1280x960	[-1,1]	[-1,1]	5847	5850	5864	5878	5889	17,17	57,24%
31	1280x960	[-1,1]	[-1,1]	5791	5801	5821	5833	5859	17,34	55,93%
32	1280x960	[-1,1]	[-1,1]	5778	5809	5810	5812	5816	17,38	54,30%

Фигура 30: Таблица с резултати от проведените тестове за сектор

4) Сравнителна графика и коментари върху проведените тестове



Фигура 31: Сравнителна графика на трите проведени тестови подгрупи

- В първата подгрупа не се получава задоволително ускорение, докато в другите две подгрупи се наблюдава подобрение.

- Както при проведени тестове за размер, забелязва се, че точността на ускорението е пряко свързана с времето нужно за изпълнение на програмата. В този случай колкото повече точки, които принадлежат на множеството на Манделброт съдържа определения сектор, толкова по-точно ще се получи ускорението.

- Третата подгрупа съдържа най-голям брой точки от множеството, поради което изисква най-много време за изчисление и получава най-добро ускорение като резултат.

4. Списък с източници

- [1] Isaac K. Gäng, David Dobson, Jean Gourd and Dia Ali,

  Parallel Implementation and Analysis of Mandelbrot Set Construction,
  https://www.academia.edu/1399383/Parallel_Implementation_and_Analysis_of_Mandelbrot_Set_Construction

- [2] Matthias Book,

  Parallel Fractal Image Generation - A Study of Generating Sequential Data with Parallel Algorithms, presented May 3, 2001,

  http://matthiasbook.de/papers/parallelfractals/files/parallelfractals-paper.pdf

- [3] Distrust Simplicity,

  Parallel Mandelbrot in Julia, C++, and OpenCL,
  January 9, 2015,

  http://distrustsimplicity.net/articles/mandelbrot-speed-comparison/

- [4] Intel,

  Tutorial: Develop an Application with Intel® Threading Building Blocks - Overview,
  November 9, 2018

  https://software.intel.com/content/www/us/en/develop/documentation/tbb-tutorial/top/overview.html