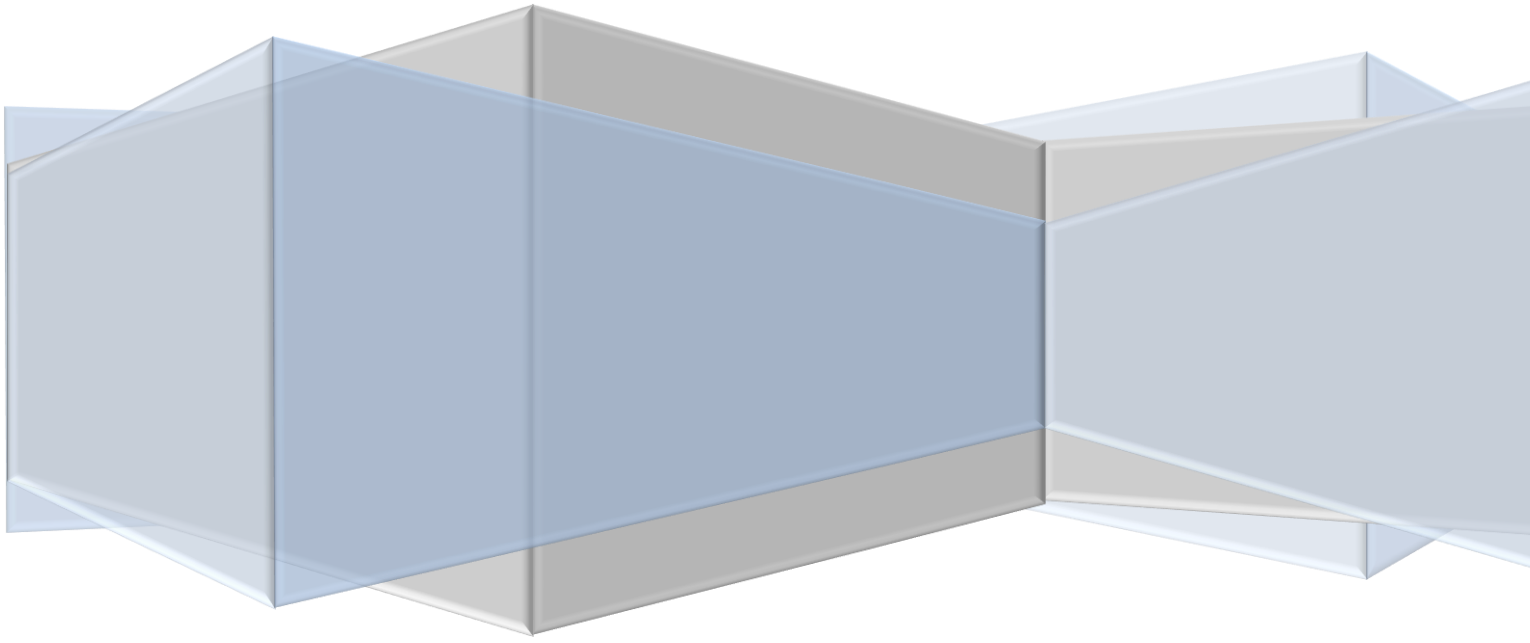
Изобразяване на фрактал по формулата

$F(z) = e^{\cos(c*z)}$

Проект по Системи за паралелна обработка

Добрин Цветанов Цветков, Компютърни науки, 3 курс, факултетен номер 81265



Проверил: .....

(ас. Христо Христов)

Contents

1. Увод

Проектираното приложение има за цел да изобрази фрактал, зададен от комплексни числа, който се получава по формулата F(z) = e^cos(c*z). Това означава, че за генерирането на множеството, зададено от тази формула, избираме точка в комплексната равнина C = a + ib и извършваме итеративен процес Z0 = (0,0), Z1 = F(Z0), Z2 = F(Z1) и т.н. Ако след определено време точката все още е близо до началото, то казваме, че тя принадлежи на множеството. В противен случай казваме, че точката клони към безкрайност и не принадлежи на разглежданото множество.

1.1. Използвани образци

Образец	Коментар
Примерен проект за генериране на фрактал от ас. Христо Христов[1]	Полезен за вникване в конкретния проблем – как се осъществява итеративният процес, как се проверява дали оставаме в множеството, как се генерира изображение с полученият фрактал.
Проект за генериране на множеството на Манделброт – Joni Salonen [2]	Предоставя хитра идея как на всеки пиксел да съпоставим комплексно число и как да фиксираме частта от комплексната равнина, в която ще търсим визуализация. Освен това показва друг начин за оцветяване на пикселите.

1.2. Описание на конкретната задача

Изискванията към задачата са:

- Програмата трябва да използва паралелни процеси (нишки) за да разпредели работата по търсенето на точките от множеството на повече от един процесор.
- Програмата трябва да осигурява и генерирането на изображение (например .png), показващо така намереното множество.
- Програмата да позволява команден параметър от вида „-s 640x480“ (или „-size“), който задава големината на генерираното изображение, като широчина и височина в брой пиксели. При не-въведен от потребителя команден параметър, за големина на изображението, програмата подразбира - широчина (width) 640px и височина (height) 480px.
- Команден параметър, който да задава частта от комплексната равнина, в която ще търсим визуализация на множеството: „-r -2.0:2.0:-1.0:1.0“ (или „-rect“). Стойността на параметъра се интерпретира както следва: a∈[−2.0,2.0],b∈[−1.0,1.0] . При не въведен от потребителя параметър програмата приема че е зададена стойност по подразбиране: „-2.0:2.0:-2.0:2.0“.
- Друг команден параметър, който задава максималния брой нишки (паралелни процеси) на които разделяме работата по генерирането на изображението: „-t 3“ (или „-tasks“); При не-въведен от потребителя команден параметър за брой нишки – програмата подразбира 1 нишка.

- Команден параметър указващ името на генерираното изображение: „-o zad17.png“ (или „- output“). Съответно програмата записва генерираното изображение в този файл. Ако този параметър е изпуснат се избира име по подразбиране: „zad17.png“.
- Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за завършване на всички изчисления по визуализирането на точките от множеството.
- Да се осигури възможност за „quiet“ режим на работа на програмата, при който се извежда само времето през което програмата е работила (без „подходящите“ съобщения от предходната точка). Параметърът за тази цел нека да е „-q“ (или „-quiet“); Тихият режим не отменя записването на изображението във изходният файл.

2. Реализация

Задачата е реализирана на Java, като за работа с комплексни числа се използва библиотеката **Apache Commons Math3[3]**.

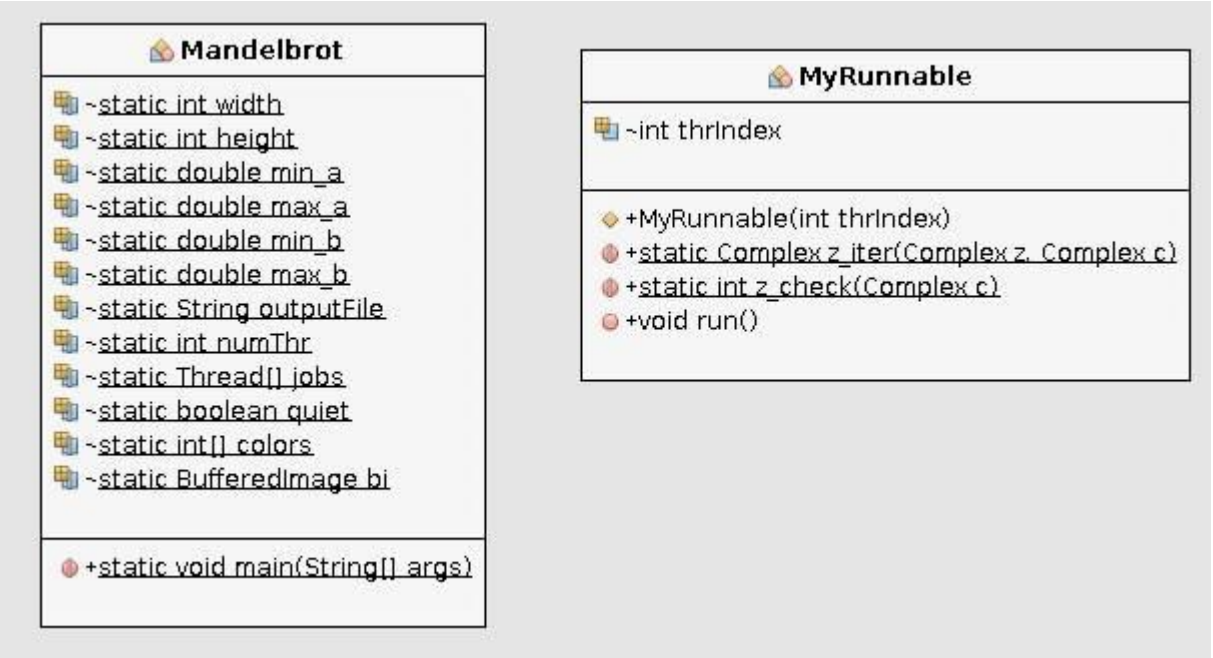
Програмата реализира техниката Single Program, Multiple Data – пускат се много нишки, които вършат една и съща работа върху различни части от изображението, тоест имаме декомпозиция по данни. Освен това имаме статично балансиране и разпределение. Моделът на работа е Master-slaves.

2.1. Алгоритъм

Работата протича по следния начин:

- Главната нишка засича начално време.
- След това преглежда всички командни параметри за да определи размер на изображението, частта от комплексната равнина, брой нишки, име на файл и режим на работа
- Главната нишка създава статичен обект от тип BufferedImage, където след това всички нишки да записват своите резултати, както и статичен масив от цели числа colors, който да определя цветове по това колко бързо се отдалечаваме от началната точка.
- Ако зададеният брой нишки от командният параметър е p, то нашата програма пуска p-1 slave нишки, като след това и тя започва същата работа като тях, тоест главната нишка също върши част от работата и с нея получаваме общо p на брой работещи нишки.
- За конкретна нишка, ако нейният индекс е k (0<=k<p), то тя изчислява всеки пиксел от ред k, k+p, k+2*p и т.н. до достигане на зададената височина на изображението.
- За проверката на конкретен пиксел гледаме дали за до 640 стъпки ще се отдалечим много от началната точка. Според броя на необходимите стъпки се задава цвят на този пиксел в статичният обект от тип BufferedImage
- Главната нишка изчаква приключването на всички останали, засича време на край на работата, изкарва на стандартният изход общото време за работа и записва полученото изображение във файл.

2.2. UML клас диаграма и описание



Класът Манделброт съдържа Master логиката. Променливите width и height са за размерите на изображението; min_a, max_a, min_b и max_b са за определяне на частта от комплексната равнина; outputFile е за име на файл, в който да бъде записан резултата; numThr е за броя нишки при текущото изпълнение; масивът jobs се използва за запазване на Slave нишките; quiet е за тих режим; colors е за генериране на масив от цветове, от който след това да се взима цвят според това за колко стъпки се отдалечаваме много от началната точка; bi е обектът, където всички нишки записват своите резултати. В main методът се проверява за подадени командни параметри, създава се обектът от тип BufferedImage, създава се масивът от цветове и масивът от нишки и се пускат Slave нишките. След което главната нишка също започва работа по фрактала чрез създване на нов обект от тип MyRunnable с индекс 0 и извикване на методът run на този обкет, тоест главната нишка върши работа на нулевият Slave от масива jobs. След приключване на методът run се извиква join() за всяка от Slave нишките, пресмята се време на приключване, изкарва се подходящо съобщение и резултатът се записва в .png файл.

Класът MyRunnable имплементира интерфейс Runnable и съдържа логиката на Slave нишките за генериране на фрактала. За създаването на обект от този клас е нужно да се подаде аргумент-индекс на нишката. Методът z_iter реализира една итерация по формулата от условието. Методът z_check прави до 640 итерации и пресмята след колко се отдалечаваме много от началната точка. Методът run е съществената част от логиката – обработват се само редовете, които съответстват на индекса на текущата нишка, на всеки пиксел се съпоставя комплексно число и се вика методът z_check. Според върнатият резултат се определя цвят и за съответният пиксел този цвят се записва в статичната променлива bi на класът Mandelbrot.

3. Тестване

3.1. Стартиране на програмата

Първо е необходимо да сложим двата .java файла и файлът commons-math3-3.5.jar в една директория. След това изпълняваме командата **javac –cp ".:commons-math3-3.5.jar" *.java**, за да компилираме нашата програма. За стартиране на програмата изпълняваме **java –cp ".:commons-math3-3.5.jar" Mandelbrot**, като накрая може да добавим допълнителни командни параметри, описани в условието на задачата (например **–t 4** за указване на работа с 4 нишки).

3.2. Резултати

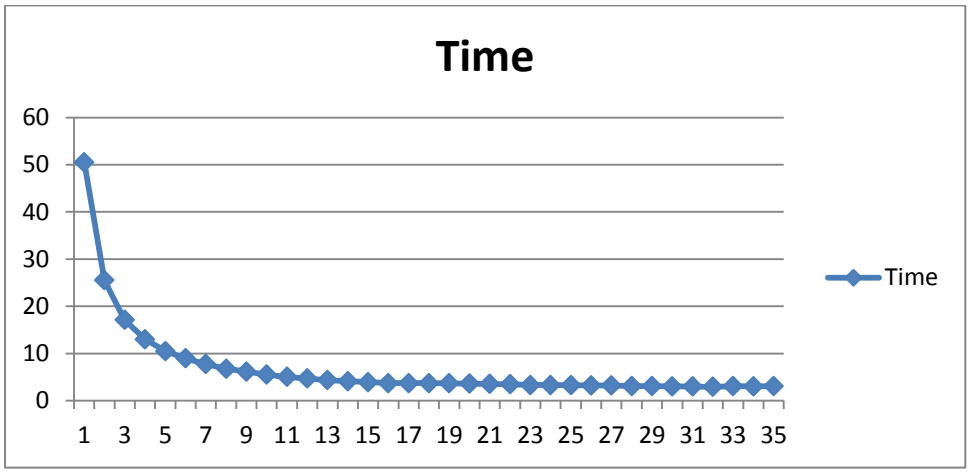
Програмата е тествана на сървър, предоставен от асистент Христо Христов. Ще покажем резултати от тестване на програмата със стойности по подразбиране за всички аргументи, с изключение на броят нишки.

- В табличен вид резултатите са:

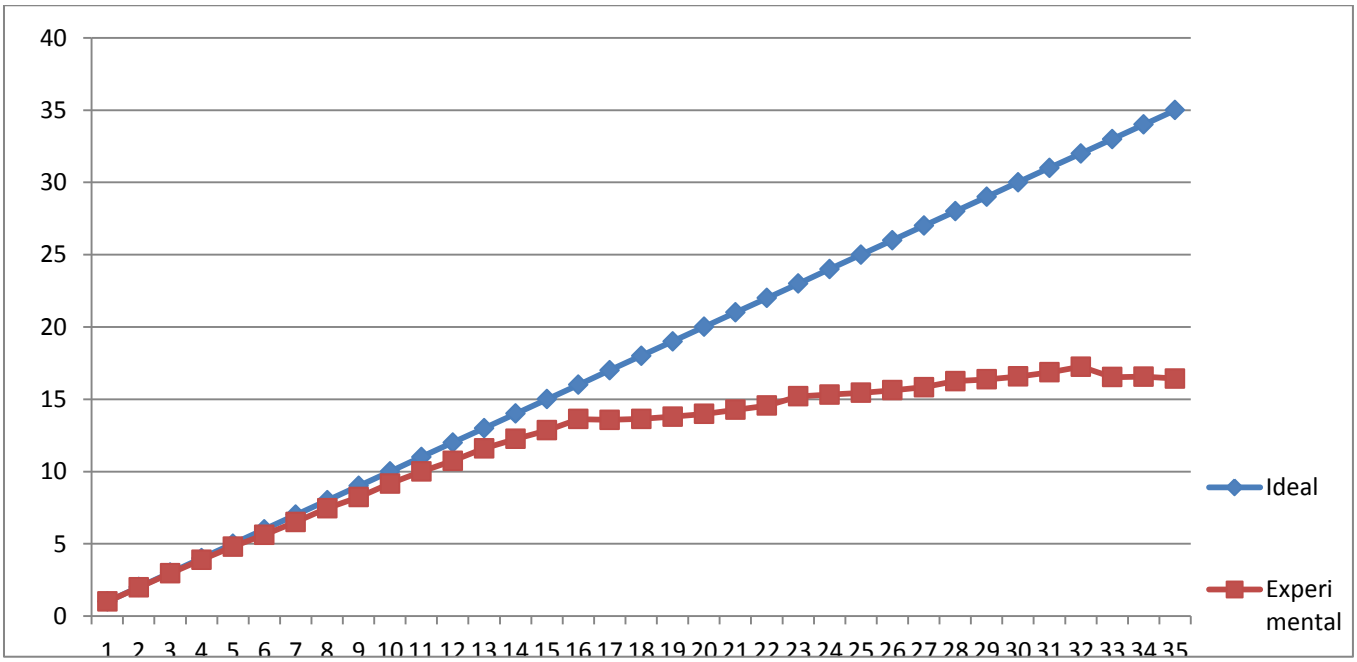
p	Tp	ms	Sp	Ep
1	50,517	50517	1	1
2	25,529	25529	1,978808	0,989404
3	17,126	17126	2,949726	0,983242
4	12,993	12993	3,888017	0,972004
5	10,51	10510	4,806565	0,961313
6	8,985	8985	5,622371	0,937062
7	7,76	7760	6,509923	0,929989
8	6,77	6770	7,461891	0,932736
9	6,145	6145	8,22083	0,913426
10	5,511	5511	9,166576	0,916658
11	5,052	5052	9,999406	0,909037
12	4,707	4707	10,73231	0,894359
13	4,354	4354	11,60243	0,892495
14	4,121	4121	12,25843	0,875602
15	3,93	3930	12,8542	0,856947
16	3,707	3707	13,62746	0,851716
17	3,723	3723	13,5689	0,79817
18	3,703	3703	13,64218	0,757899
19	3,663	3663	13,79115	0,72585
20	3,613	3613	13,98201	0,6991
21	3,54	3540	14,27034	0,67954
22	3,471	3471	14,55402	0,661546
23	3,323	3323	15,20223	0,660966
24	3,296	3296	15,32676	0,638615
25	3,271	3271	15,4439	0,617756
26	3,234	3234	15,62059	0,600792
27	3,19	3190	15,83605	0,58652
28	3,109	3109	16,24863	0,580308
29	3,086	3086	16,36973	0,564474
30	3,048	3048	16,57382	0,552461

31	2,996	2996	16,86148	0,543919
32	2,929	2929	17,24718	0,538974
33	3,057	3057	16,52502	0,500758
34	3,05	3050	16,56295	0,487146
35	3,075	3075	16,42829	0,46938

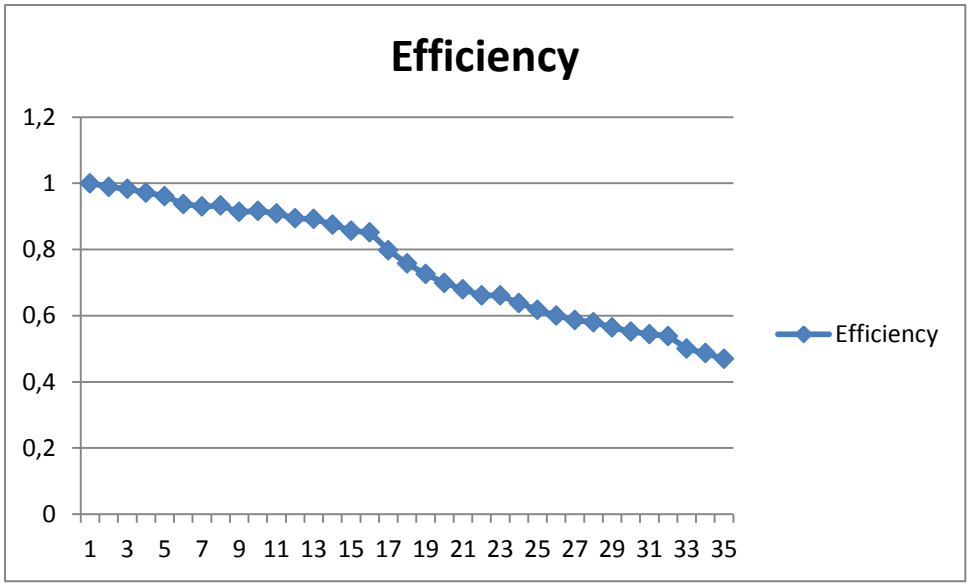
- Графика на времето за работа според броят нишки



- Графика на ускорението според броят нишки



- Графика на ефективността според броят нишки



4. Библиография

[1] – Примерен проект от ас. Христо Христов за генериране на фрактали - http://rmi.yaht.net/docs/example.projects/pfg.zip

[2] – Още един пример за генериране на множеството на Манделброт, изготвен от Joni Salonen - http://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/

[3] – Библиотека Apache Commons Math3, която използваме за работа с комплексни числа - http://commons.apache.org/proper/commons-math/userguide/complex.html