Софийски университет „Св. Климент Охридски“

Факултет по математика и информатика

Курсов проект

по Разпределени софтуерни архитектури

Тема: „Изобразяване на фрактал – Множество на Манделброт“

Изготвил: Ангел Николаев Беширов, фак. № 62012, СИ, курс 3, група 5

Ръководител: проф. Васил Георгиев, ас. Христо Христов

Дата: 09.06.2019 Проверка: ……..

1. Цел на проекта и описание на задачата

Целта на проекта е да се напише програма, която визуализира и генерира изображение на множеството на Манделброт по формулата:
(1) $F(Z) = C * \cos(Z)$

Започваме от $Z_0 = 0$ и $C = a + ib$ и итерираме използвайки горната формула. Получаваме редицата $\{Z_0, Z_1, Z_2, Z_3 \dots Z_{n,} Z_{n+1} \dots\}$.

Чрез итерирането разделяме точките в комплексната равнина на две категории:

- точки, принадлежащи на множеството на Манделброт (стабилни), ако след N итерации Z_N НЕ клони към безкрайност, т.е. $\lim_{n \to N} Z_n \neq \infty$
- точки, непринадлежащи на множеството на Манделброт (нестабилни), ако за някакъв брой итерации по-малък от N открием Z_i, което клони към безкрайност, т.е. $\lim_{n < N} Z_n = \infty$

Програмата трябва да използва паралелни процеси за да разпредели работата по търсенето на точките от множеството на Манделброт на повече от един процесор.

Програмата позволява (разбира от) следните командни параметри:

- Команден параметър, който задава големината на генерираното изображение, като широчината и височина в брой пиксели. Той има вида: **„-s 640x480“** (или **„-size“**). При невъведен от потребителя команден параметър, за големина на изображението програмата подразбира – широчината (width) **640px** и височината (height) **480px**.
- Команден параметър, който задава частта от комплексната равнина, в която ще търсим визуализиция на множеството на Манделброт: **„-r -2.0:2.0:-1.0:1.0“** (или **„-rect“**). Стойността на параметъра се интерпретира по следния начин: $\mathrm{a} \in [-2.0, 2.0],\ \mathrm{b} \in [-1.0, 1.0]$. При невъведен от потребителя, програмата приема, че е зададена стойност по подразбиране: **„-2.0:2.0:-2.0:2.0“**
- Команден параметър, който задава максималния брой нишки (паралелни процеси) на които разделяме работата по генерирането на изображението: **„-t 3“** (или **„–tasks”**). При невъведен от потребителя параметър за брой нишки – програмата подразбира **1** нишка.
- Команден параметър, който задава грануларността на разделянето на изображението на сегменти. При невъведен от потребителя параметър за грануларност програмата подразбира **1**.
- Команден параметър указващ името на генерираното изображнеие: **„-o zad18.png“** (или **„-output“**). Съответно програмата записва генерираното

изображение в този файл. Ако този параметър е изпуснат (не е зададен от потребителя), се избира име по подразбиране: **„zad18.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 на Манделброт (пресмятане на множеството на Манделброт).
- Команден параметър, който осигурява възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето, през което програмата е работила (без „подходящите“ съобщения от предишната точка). Параметърът за тази цел е **„-q“** (или **„-quiet“**). Тихият режим не отменя записването на изображението в изходния файл.

2. Реализация и описание на алгоритъма

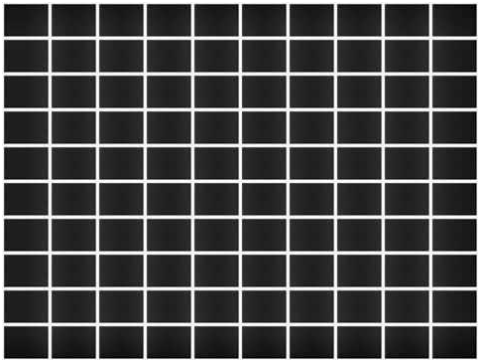
Алгоритъмът е имплементиран на Java 8 и са използвани 2 външни библиотеки - **Apache Commons CLI** и **Apache Commons Math3**. За да стартираме програмата извикваме класът Runner.java и му подаваме параметрите, за които не искаме да използваме стойностите по подразбиране. Пример за извикване:

java Runner –t 4 –g 2 –s 2000x2000

стартира програмата с 4 нишки, грануларност 2 и размер на изображението 2000 широчина и 2000 височина. Архитектурата на програмата е по Master/Slave модела, като класа Runner е master а задачите, които обработват отделните сегменти от изображението са slaves.

След като обработи аргументите подадени от потребителя, Runner класа разделя изображението на отделни сегменти, на база аргументите за брой задачи и грануларност (Фиг. 1). Всеки сегмент е с височина и широчина съответно: height/(granularity*numOfThreads) и width/(granularity*numOfThreads). След разделянето, използвам ExecutorService, който е подинтерфейс на Executor интерфейса, отговорен за стартирането на нишки в приложението. Имплементацията, която съм използвал е ThreadPoolExecutor придобита чрез статичния метод Executors.newFixedThreadPool(numOfThreads). След това създавам numOfThreads MandelbrotWorkers и ги изпращам на ExecutorService-а за изпълнение. Всеки MandelbrotWorker съдържа всички сегменти, на които е разделено изображението и инденс от 0 до numOfThreads съответстващ на индекса на нишката. Всяка нишка решава кои сегменти от изображението да обработи в зависимост от индекса си. Нишка X с индекс x обработва сегмент Y с индекс y ако

y % numOfThreads == x (остатъкът от деленето на индекса на сегмента за обработка от изображението на броя нишки е равен на индекса на нишката).



Фиг. 1 Разделяне на изображението на сегменти

Обработката на сегмента представлява изобразяване (mapping) на всеки пиксел от съответната част от изображението към точка C от комплексната равнина. След което проверява дали съответната точка от комплексната равнина принадлежи на множеството на Манделброт използвайки Escape time алгоритъма.

В Escape time алгоритъма, координатите на точката C се използват за начална стойност на итерацията по формула (1). На всяка итерация се проверява дали новата стойност Z_n не клони към безкрайност и дали броят итерации не надминава максималния брой, зададен предварително в програмата. Ако след определения максимален брой итерации стойността все още не клони към безкрайност, то точката принадлежи на множеството на Манделброт и се оцветява в черно. Ако стойността клони към безкрайност и броят итерации не е надминал максималния брой, то тогава точката се оцветява в различен цвят в зависимост от броя итерации, които са били направени.

След това ExecutorService-а се изключва, като се извиква shutdown() метода, което предотвратява изпращането на повече задачи за изпълнение в услугата и изчаква изпълнението на всички нишки работници да приключи, като извиква метода awaitTermination(long timeout, TimeUnit unit). Накрая генерираното изображение се записва във файл с името, което е подадено от командния ред или името по подразбиране (zad18.png).

3. Тестване

При извикване на java Runner –g 4 –t 16 –s 2000x2000

Програмата работи с коефициент на грануларност 4, 16 нишки и размер на изображението 2000 широчина и 2000 височина, което означава че разделяме изображението на сегменти с височина 2000/64 и широчина 2000/64 закръглени нагоре.

Тествах програмата на тестовия сървър t5600.rmi.yaht.net със следните параметри:

- Размер на изображението – 2000x2000 (1:1)
- Грануларност – 1, 2, 4 и 8
- Брой нишки 1, 2, 4, 6, 8, 10 ... 32

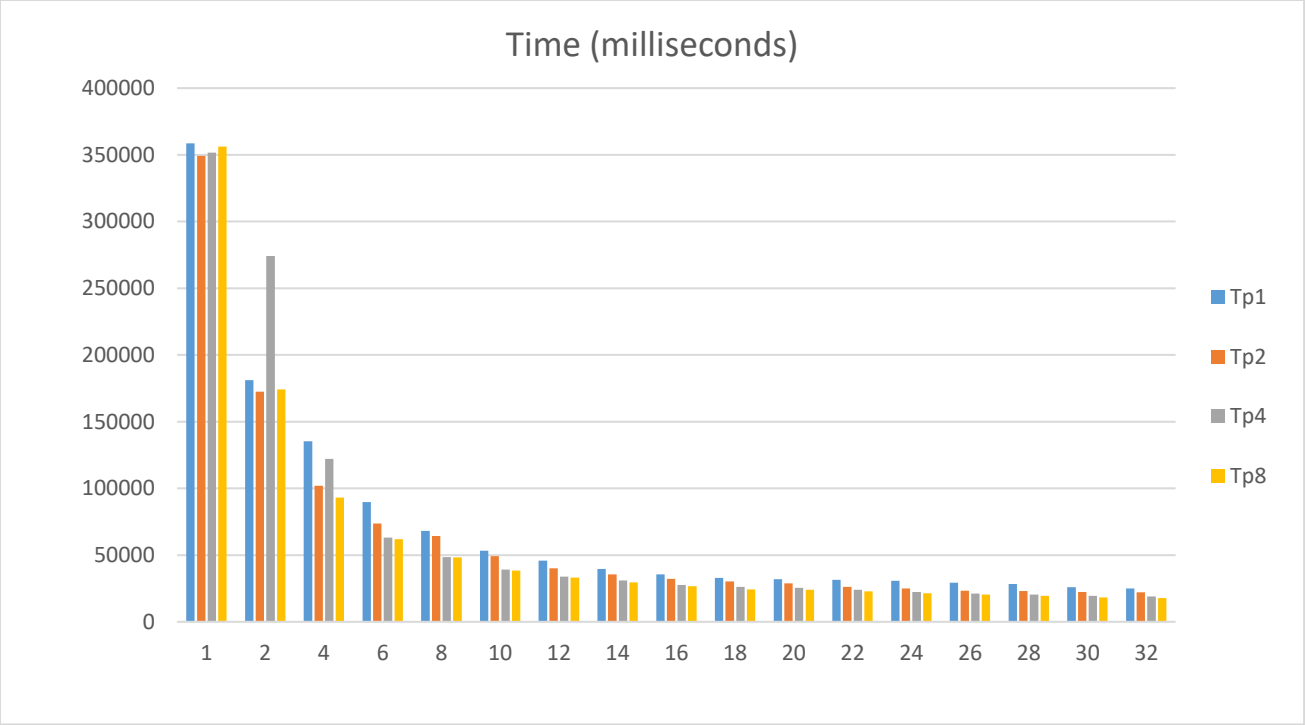
Останалите параметри са взети по подразбиране.

Всеки тест проведох по 3 пъти, като в таблицата записах най-бързото от 3-те времена.

Време за изпълнение

Tp1, Tp2, Tp4, Tp8 са времето за изпълнение на програмата при грануларност съответно 1, 2, 4 и 8.

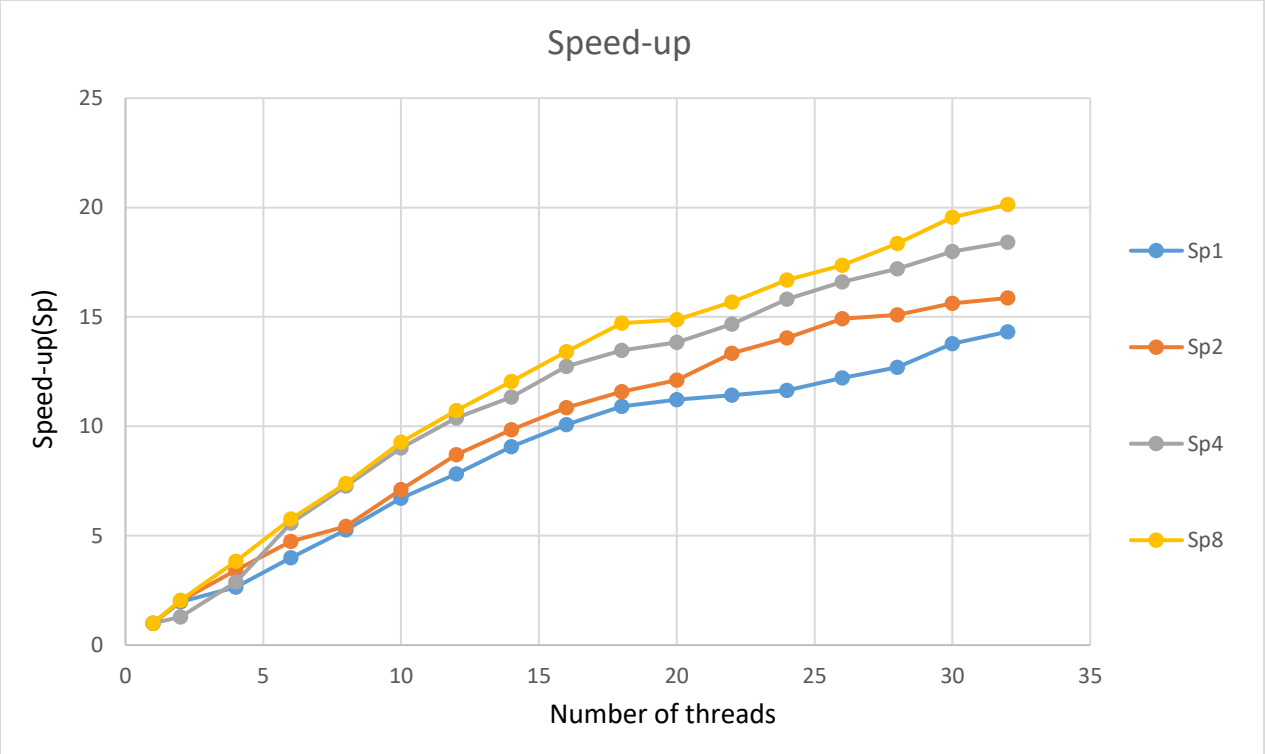
Threads	Tp1	Tp2	Tp4	Tp8
1	358521	349215	351542	356214
2	181113	172444	274120	174215
4	135214	102013	122041	93146
6	89756	73745	63074	61800
8	68173	64401	48391	48213
10	53375	49147	39038	38386
12	45833	40142	33859	33211
14	39507	35504	31014	29563
16	35553	32171	27605	26573
18	32848	30142	26104	24215
20	31948	28841	25417	23946
22	31405	26174	23975	22717
24	30804	24875	22241	21340
26	29341	23411	21174	20524
28	28232	23125	20434	19410
30	26022	22354	19548	18213
32	25031	22014	19084	17692



Ускорение

Sp1, Sp2, Sp4, Sp8 са ускорението на програмата при грануларност съответно 1, 2, 4 и 8.

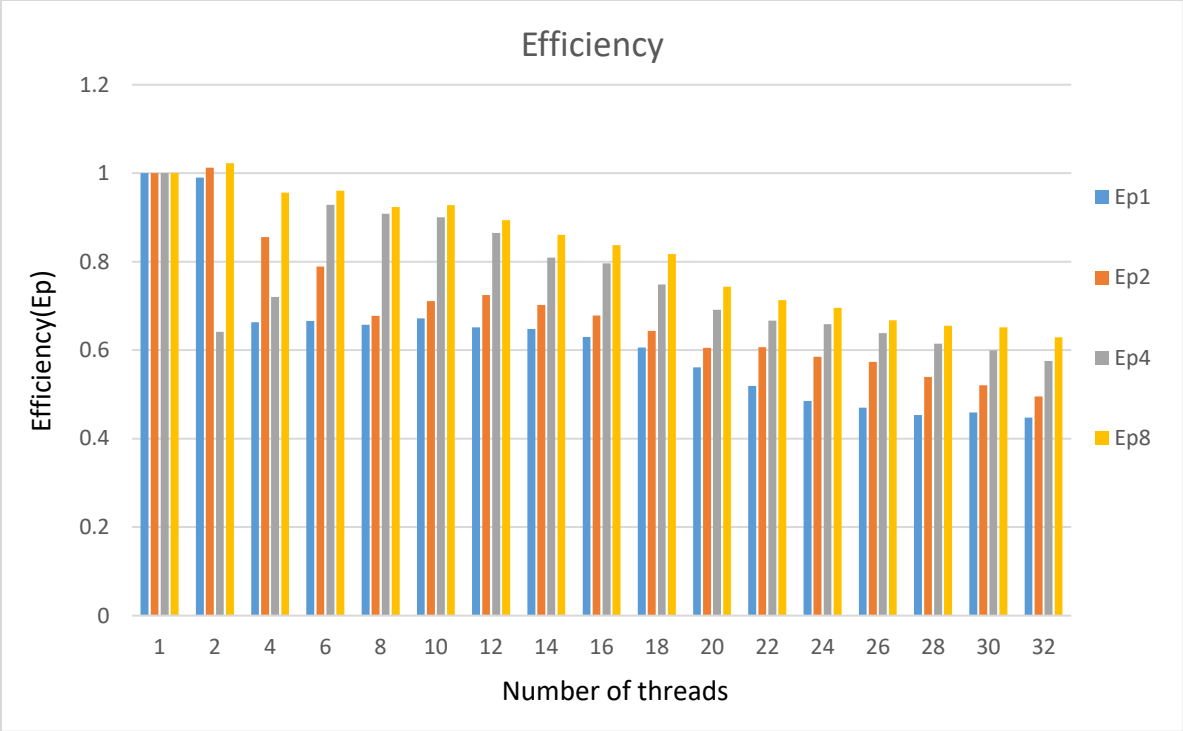
Threads	Sp1	Sp2	Sp4	Sp8
1	1	1	1	1
2	1.97954	2.02509	1.28244	2.04468
4	2.65151	3.42324	2.88052	3.82425
6	3.9944	4.73544	5.57349	5.76398
8	5.25899	5.42251	7.26462	7.38834
10	6.71702	7.10552	9.00512	9.27979
12	7.82233	8.69949	10.3825	10.7258
14	9.07487	9.83593	11.3349	12.0493
16	10.0841	10.855	12.7347	13.4051
18	10.9145	11.5857	13.467	14.7105
20	11.222	12.1083	13.831	14.8757
22	11.416	13.3421	14.6629	15.6805
24	11.6388	14.0388	15.806	16.6923
26	12.2191	14.9167	16.6025	17.356
28	12.6991	15.1012	17.2038	18.3521
30	13.7776	15.622	17.9835	19.5582
32	14.3231	15.8633	18.4208	20.1342



Ефективност

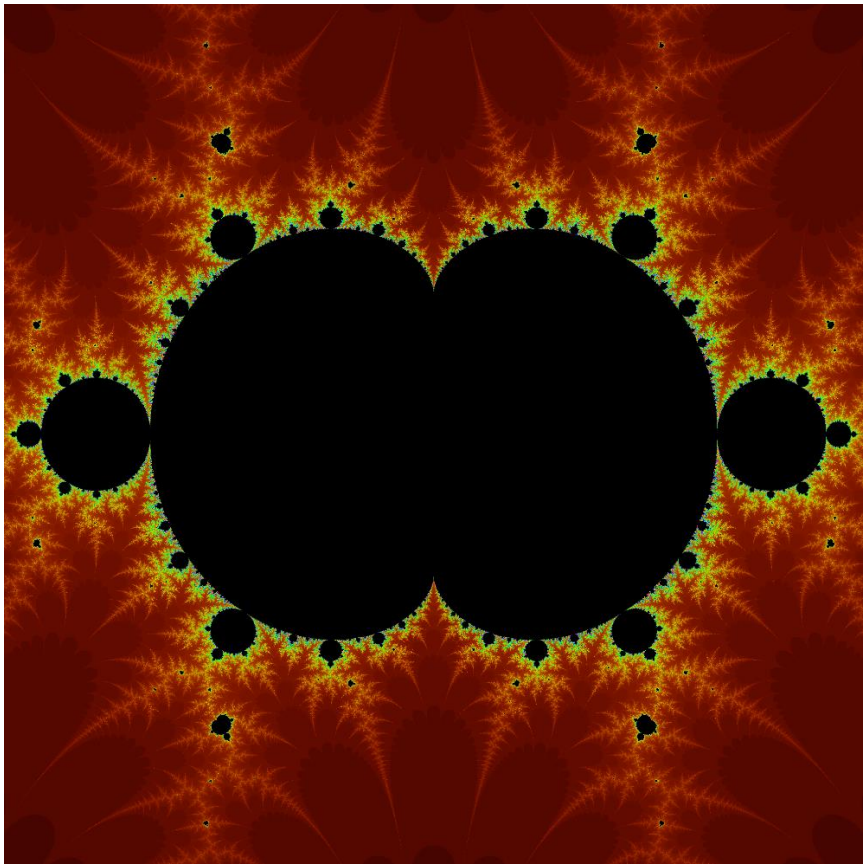
Ep1, Ep2, Ep4, Ep8 са ефективността на програмата при грануларност съответно 1, 2, 4 и 8.

Threads	Ep1	Ep2	Ep4	Ep8
1	1	1	1	1
2	0.98977	1.01255	0.64122	1.02234021
4	0.66288	0.85581	0.72013	0.9560636
6	0.66573	0.78924	0.92891	0.96066343
8	0.65737	0.67781	0.90808	0.92354241
10	0.6717	0.71055	0.90051	0.92797895
12	0.65186	0.72496	0.86521	0.8938153
14	0.64821	0.70257	0.80964	0.8606656
16	0.63026	0.67844	0.79592	0.8378194
18	0.60636	0.64365	0.74817	0.81724826
20	0.5611	0.60541	0.69155	0.74378602
22	0.51891	0.60646	0.66649	0.71275016
24	0.48495	0.58495	0.65858	0.69551312
26	0.46997	0.57372	0.63856	0.66753744
28	0.45354	0.53933	0.61442	0.65543166
30	0.45925	0.52073	0.59945	0.65194092
32	0.4476	0.49573	0.57565	0.62919328



Резултатите от тестовете могат да бъдат обяснени с това че при по-висока грануларност (например 8) сегментите са по-малки и се разпределят по-равномерно на отделните нишки и не се случва сценарий при който една нишка получава няколко по-трудни сегмента (сегменти, които съдържат повече точки от множеството на Манделброт), което забавя цялостното изпълнение на програмата.

$\boldsymbol{T1}$ – времето за изпълнение на серийната програма.
$\boldsymbol{Tp}$ – времето за изпълнение на паралелната програмата, използваща p нишки.
$\boldsymbol{Sp} = \boldsymbol{T1}/\boldsymbol{Tp}$ – ускорението на програмата при използването на p нишки.
$\boldsymbol{Ep} = \boldsymbol{Sp}/\boldsymbol{p}$ – ефективността на програмата при използването на p нишки.



Фиг. 2 Резултат от изпълнението на програмата

4. Използвани ресурси

[1] Wikipedia contributors, "Mandelbrot set," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Mandelbrot_set&oldid=900489579 (последно посетен на 2019-06-06).

[2] Wikipedia contributors, "Master/slave (technology)," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Master/slave_(technology)&oldid=893227569 (последно посетен на 2019-06-06).

[3] Weisstein, Eric W. "Mandelbrot Set." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/MandelbrotSet.html (последно посетен на 2019-06-06)

[4] J. D. Jones, “Coloring by numbers, Escape Time Algorithm”, https://mcanv.com/Art/escape_time_algorithm.html, (последно посетен на 2019-06-05)