

**Софийски университет „Св. Климент Охридски”**

**Факултет по математика и информатика**

**Курсов проект**

**по РИТАрх**

Тема: „Изобразяване на фрактал – Mandelbrot set, v2“

**Изготвил:** Борис Георгиев Дечев, ФН: 71751, Информационни системи, курс II, група 1.

**Ръководител:** проф. Васил Георгиев, ас. Христо Христов

Дата: 01. 06. 2019 Проверил: ………….

**Съдържание:**

1. Увод, условие на проекта и функционалности ….. 3.
2. Проектиране и реализация ……………………….. 5.
3. Тестови измервания и анализ

на метричните показатели ……………………….. 7.

1. Увод, условие на проекта и функционалности.

Проектът реализира паралелен алгоритъм, визуализиращ множеството на Манделброт, дефинирано по формулата:

(1) F(Z) = C\*e-Z + Z2

в равнината на комплексните числа, тоест множеството от точки C в комплексната равнина, за които редицата {Z0, Z1, Z2, …, Zn, Zn+1, ….}, построена посредством формулата (1), е ограничена, като итерирането започва от Z0 = 0. Това условие може да се изрази и по следния начин:

C ∈ M , дефинирано с формула (1) limn -> ∞ sup|Zn| ≤ 2

С други думи, открием ли член на редицата, който надвишава 2 по абсолютна стойност, знаем, че тя клони към безкрайност, т.е е неограничена и съответното комплексно число C не принадлежи на множеството на Манделброт. Този факт е в основата на Escape time алгоритъма, който програмата използва, за да генерира множеството: всеки пиксел (x,y), x [0,width], y [0,height] , се изобразява в точка C от комплексната равнина с координати (c\_re, c\_im), след което с тази точка C програмата итерира покоординатно формула (1), докато или поредното Zn надхвърли 2 по абсолютна стойност, с което C излиза от множеството на Манделброт (escape condition), или се надхвърли определен максимален брой итерации, зададен предварително, без C да напусне търсеното множество, т.е в този случай C принадлежи на множеството на Манделброт. На този принцип всички пиксели от изображението с размери width и height биват отбелязани като принадлежащи или непринадлежащи на множеството на Манделброт, като им се асоциира и съответен цвят (черно за точките от множеството, друг цвят за тези извън него).

Програмата удовлетворява следните функционални изисквания:

- Позволява (разбира от) команден параметър, който задава големината на генерираното изображение, като широчина и височина в брой пиксели. Той има вида:„-s 640x480“(или „-size“). При невъведен 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 на Манделброт:„-r -2.0:2.0:-1.0:1.0“ (или „-rect“). Стойността на параметъра се интерпретира както следва: а ∈ [−2.0,2.0] ,b ∈ [−1.0,1.0]. При невъведен 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 невъведен от потребителя команден параметър за брой нишки – програмата подразбира 1 нишка;

- Команден параметър, указващ името на генерираното изображение:„-o zad16.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: „zad16.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).

Примери за подходящи съобщения:

„Thread- started.“,

„Thread- stopped.“,

„Thread- execution time was (millis): “,

„Threads used in current run: “,

„Total execution time for current run (millis): “ и т.н.

–Възможен е „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 е „-q“ (или „-quiet“); Тихият режим не отменя записването на изображението във изходният файл;

2. Проектиране и реализация.

Алгоритъмът е имплементиран на Java 11. Реализиран е паралелизъм по данни (Single Program-Multiple Data) или SPMD, като нишките работят асинхронно (независими са една от друга). Добавен е mandelbrot.jar файл, от който може да бъде стартирана програмата. Пример:

java -jar Mandelbrot.jar -t 32

Тази команда ще стартира програмата с 32 нишки и останалите параметри взети по подразбиране.

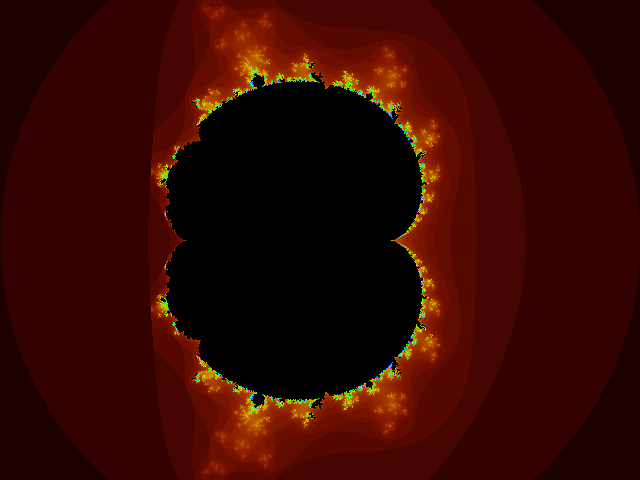
Класът Main.java се грижи всички опции да бъдат взети под внимание, създава инстанция на класа FractalMultithreading.java, подавайки му в конструктора цялата необходима информация като размер на изображението, брой на нишките, име на изображението и режим на работа.

FractalMultithreading.java от своя страна създава BufferedImage със подадените width и height. Определят се максимален брой итерации за определяне дали една точка е в множеството или не. След това се разделя изображението на отделни порции(по редове) и се пресмята евентуалният остатък от изображението(където при наличието на такъв, се добавя към последната нишка). Създават се зададеният брой нишки, като на всяка нишка се залага работата да пресмята различна порция от изображението.

Това пресмятане става в класа MandebrotGenerator.java, като в конструктора на класа се подава създаденото от FractalMultithreading.java изображение (за да се оцветят в съответния цвят нужните пиксели), масивът от цветове (от който вземаме и максималния брой итерации), началото и краят на порцията (за която ще пресмятаме), броя нишки (за принтиране на съобщението) и булева променлива за режима на работа. В началото на метода run() съобщаваме, номера на нишката, която е тръгнала (когато isQuiet == false) и след това по формулата (1) итерираме, докато не надхвърлим максималния брой итерации или не излезем извън кръга с радиус 2, в който лежи множеството на Манделброт. След това оцветяваме съответния пиксел, в съответния цвят. Накрая на операцията съобщаваме, че дадената нишка е приключила работата си и времето, за което я е свършила (когато isQuiet == false).

За улеснение, имаме клас Timer.java, чиито инстанции създаваме в началото на методите run() и в двата класа (FractalMultithreading.java и MandebrotGenerator.java), за да засечем времето, от началото до края на работата, както на всяка нишка (в MandebrotGenerator.java), така и за цялата програма (в FractalMultithreading.java).

Накрая, след като е пресметнал за всички порции от изображението, FractalMultithreading.java записва изображението във .png файл със зададеното име и принтира информация за използвания брой нишки (при isQuiet == false) и времето за изпълнение на цялата програма.



3. Тестови измервания и анализ на метричните показатели.

Тестваме с цел да оценим ускорението S (speed-up) и ефективността E (efficiency) на описания алгоритъм, където ако Tp е времето за изпълнение на програмата с p на брой нишки, a Sp и Ep – съответно ускорението и ефективността за p на брой нишки, то:

Sp = T1 / Tp

Ep = Sp / p

За по-голяма точност, правим по 5 замервания за всяко p, от които вземаме средното.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | Зам. №1 (millis) | Зам. №2 (millis) | Зам. №3 (millis) | Зам. №4 (millis) | Зам. №5 (millis) | Средно (millis) | S(p) | E(p) |
|  |  |  |  |  |  |  |  |  |
| 1 | 7706.0 | 7620.0 | 7986.0 | 7710.0 | 7667.0 | 7737.8 | 1.0000 | 1.0000 |
| 2 | 4188.0 | 4260.0 | 4097.0 | 4215.0 | 4250.0 | 4202.0 | 1.8415 | 0.9208 |
| 4 | 3654.0 | 3603.0 | 4753.0 | 3545.0 | 3536.0 | 3818.2 | 2.0266 | 0.5067 |
| 6 | 2589.0 | 2657.0 | 2646.0 | 2633.0 | 2644.0 | 2633.8 | 2.9379 | 0.4897 |
| 8 | 2173.0 | 3305.0 | 2092.0 | 2080.0 | 2069.0 | 2343.8 | 3.3014 | 0.4127 |
| 10 | 1865.0 | 1840.0 | 1853.0 | 1974.0 | 1789.0 | 1864.2 | 4.1507 | 0.4151 |
| 12 | 1684.0 | 1732.0 | 1590.0 | 1601.0 | 1652.0 | 1651.8 | 4.6845 | 0.3904 |
| 14 | 1600.0 | 1430.0 | 1671.0 | 1488.0 | 1655.0 | 1568.8 | 4.9323 | 0.3523 |
| 16 | 1611.0 | 1395.0 | 1629.0 | 1645.0 | 1440.0 | 1544.0 | 5.0115 | 0.3132 |
| 18 | 1565.0 | 1449.0 | 1561.0 | 1660.0 | 1438.0 | 1534.6 | 5.0422 | 0.2801 |
| 20 | 1394.0 | 1568.0 | 1344.0 | 1371.0 | 1609.0 | 1457.2 | 5.3101 | 0.2655 |
| 22 | 1335.0 | 1337.0 | 1234.0 | 1270.0 | 1231.0 | 1281.4 | 6.0386 | 0.2745 |
| 24 | 1319.0 | 1207.0 | 1274.0.0 | 1158.0 | 1284.0 | 1248.4 | 6.1982 | 0.2583 |
| 26 | 1278.0 | 1164.0 | 1186.0 | 1284.0 | 1268.0 | 1236.0 | 6.2604 | 0.2408 |
| 28 | 1222.0 | 1277.0 | 1265.0 | 1236.0 | 1278.0 | 1255.6 | 6.1626 | 0.2201 |
| 30 | 1259.0 | 1241.0 | 1290.0 | 1261.0 | 1279.0 | 1266.0 | 6.1120 | 0.2037 |
| 32 | 1272.0 | 1137.0 | 1206.0 | 1127.0 | 1270.0 | 1202.4 | 6.4353 | 0.2011 |

Наблюдаваме, че с нарастване броя на нишките, намаля времето за изпълнение на цялата програма, от където получаваме по-добро ускорение. Това от своя страна подобрява ефективността.

Източници:

[1] <https://en.wikipedia.org/wiki/Mandelbrot_set>

[2] <http://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/>

[3] <https://www.tutorialspoint.com/java/java_multithreading.htm>

[4] <http://commons.apache.org/proper/commons-cli/javadocs/api-release/index.html>