Проект

по Системи за Паралелна Обработка

Паралелен Алгоритъм за

 Генериране на Фрактали

Изготвил:

Божидар Михайлов, Фн:80797, КН, 4 група, 3 курс

Ръководител:

ас. Христо Христов

Проверил: ........................

(ас. Христо Христов)

Дата:11.06.2012г.

**Съдържание**:

1. Условие на проекта..................................................3
2. Общо описание на представената програма.........4
3. Тестови измервания и анализ на метричните показатели………......................................................5
4. **Условие на проекта**

Нека разгледаме формулата:

$$(1)F(Z)=Z^{2}\* e^{Z^{2}}+C$$

Вашата задача е да напишете програма за визуализиране на множеството на Манделброт,

определено от формула (1).

 Програмата трябва да използва паралелни процеси (нишки) за да разпредели работата по търсенето на точките от множеството на Манделброт на повече от един

процесор. Програмата трябва да осигурява и генерирането на изображение (например .png), показващо така намереното множество.

 Изискванията към програмата са следните:

* Програмата да 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: $а\in [-2.0,2.0],b\in [-1.0, 1.0]$ . 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 невъведен от потребителя команден параметър за брой нишки – програмата подразбира 1 нишка;

* Команден параметър указващ името на генерираното изображение: „o

zad20.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: „zad20.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);

* 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;

1. **Общо описание на представената програма**

За реализацията на програмата е използван/а езика/платформата Java SE 1.8 и съответните средства от нея за многонишково програмиране.

Функционалността е реализирана в един клас ***MandelbrotGenerator*.** Инстанции на класа могат да се създават както с конструкторът му, така и с Factory метода *cliMGeneratorFactory,* който значително улеснява създаването на инстанция с правилните опции, parse-нати от параметрите предадени на програмата в конзолата.

Възможните параметри са както в заданието, с изключение на ‘-q’ той има следните свойства: ‘-q’ – задава колко ‘тихo‘ e логването на програмата. Ако се подаде просто ‘-q’ се подразбира ниво ‘2’ – „най-тихо“, ако не се изобщо не се подаде параметър ‘-q’ се подразбира ниво 1 – “сравнително тихо“. Ниво 0 представлява логване на практически всяко събитие в програмата. За parse-ване на командните параметри е ползвана библиотеката Apache Commons CLI.

След създаване на инстанция на ***MandelbrotGenerator*** се извиква метода generate(), за да се генерира изображението. Тук се извършва разпределянето на работата по нишки и записването на готовият резултат, след като се изчакат всички нишки да свършат работа. Разпределянето се извършва на прост принцип: всяка нишка генерира интервал от линии от изображението. Големината на интервалите е равна за всички нишки, въпреки че може да се случи част от нишките да попаднат на региони с „по-бързо излизащи“ точки.

Същинската работа се върши от метода processLineRange, който се изпълнява от всяка нишка. Тук се итерира по всеки пиксел от зададения интервал от линии, по координатите на пиксела се пресмята съответстващата му точка в зададената част от комплексната равнина и се извиква метода *iteratePoint*, който просто прилага зададеното рекурентно правило за пресмятане на следващата точка докато настъпи едно от следните събития:

* + Достигане на максималният брой итерации – такова ограничение се налага,

тъй като за някои точки са необходими твърде много итерации за проверка за принадлежност към множеството

* + „излизане“ на точката от кръга с радиус 2 около началото на координатната система

След итерирането на всяка точка, тя се оцветява в черно ако не е излязла от множеството, а ако е излязла от множеството, от HSB цветното пространство се избира цвят с фиксирани Saturation и Brightness, и Hue – пропорционално на стъпките за които точката е излязла от множеството. Всяка нишка лог-ва сама за колко време е приключила работата си.

 След приключване на работата на всички нишки изображението се записва (от една нишка) чрез вградените в Java платформата средства за боравене с изображения.

1. **Тестови измервания и анализ на метричните показатели**

Разработеното приложение е тествано на 24 ядрен мултипроцесор с цел да се оценят ускорението S (забързване, speed-up) и ефективността Е (efficiency) на описания алгоритъм, където ако T(p) е времето необходимо за завършване на работата на алгоритъм с p на брой нишки то:

S(p) = T(1)/T(p) E(p) = S(p)/p

За по-голяма точност за всяко p са направени по 5 измервания, от които се взима средното време.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| p | Зам. №1 (s) | Зам. №2 (s) | Зам. №3 (s) | Зам. №4 (s) | Зам. №5 (s) | Средно (s) | S(p) | E(p) |
|  |  |  |  |  |  |  |  |  |
| 1 | 15.8275 | 16.6133 | 15.9507 | 16.2515 | 14.9052 | 15.9097 | 1.0000 | 1.0000 |
| 2 | 10.3369 | 10.1640 | 11.3609 | 10.4308 | 9.2877 | 10.3161 | 1.5422 | 0.7711 |
| 3 | 7.0242 | 8.0754 | 7.2141 | 7.2377 | 7.1835 | 7.3470 | 2.1655 | 0.7218 |
| 4 | 6.9652 | 6.6496 | 6.8834 | 6.9973 | 6.9606 | 6.8912 | 2.3087 | 0.5772 |
| 5 | 5.2533 | 5.2751 | 5.3834 | 5.2404 | 6.2922 | 5.4889 | 2.8985 | 0.5797 |
| 6 | 4.8909 | 4.6083 | 4.6768 | 4.6191 | 4.6307 | 4.6851 | 3.3958 | 0.5660 |
| 7 | 4.2583 | 4.5588 | 4.2624 | 4.2383 | 4.1099 | 4.2855 | 3.7124 | 0.5303 |
| 8 | 3.8670 | 3.7451 | 3.8274 | 3.7017 | 3.8180 | 3.7919 | 4.1957 | 0.5245 |
| 9 | 3.7206 | 3.6418 | 3.4648 | 3.9028 | 3.5786 | 3.6617 | 4.3448 | 0.4828 |
| 10 | 3.2388 | 3.5871 | 3.2725 | 3.4321 | 3.2660 | 3.3593 | 4.7360 | 0.4736 |
| 11 | 3.0311 | 3.2935 | 3.8057 | 3.3149 | 3.2657 | 3.3422 | 4.7603 | 0.4328 |
| 12 | 3.1763 | 3.0150 | 3.1676 | 3.0096 | 2.9984 | 3.0734 | 5.1766 | 0.4314 |
| 13 | 2.9675 | 3.1249 | 2.8055 | 3.0411 | 2.9991 | 2.9876 | 5.3252 | 0.4096 |
| 14 | 2.7454 | 3.0689 | 3.2343 | 2.8400 | 2.8009 | 2.9379 | 5.4153 | 0.3868 |
| 15 | 2.9624 | 3.2413 | 3.2513 | 2.7980 | 2.5570 | 2.9620 | 5.3713 | 0.3581 |
| 16 | 2.6689 | 2.6302 | 2.7820 | 2.5831 | 2.5802 | 2.6489 | 6.0062 | 0.3754 |
| 17 | 2.7745 | 3.2189 | 3.1579 | 3.1995 | 2.6028 | 2.9907 | 5.3197 | 0.3129 |
| 18 | 2.5469 | 2.6364 | 2.7108 | 2.6606 | 2.6866 | 2.6483 | 6.0076 | 0.3338 |
| 19 | 2.9497 | 2.5562 | 2.8355 | 2.5458 | 2.4198 | 2.6614 | 5.9779 | 0.3146 |
| 20 | 2.7123 | 2.6186 | 2.6091 | 2.3930 | 2.5540 | 2.5774 | 6.1728 | 0.3086 |
| 21 | 2.3779 | 2.4494 | 2.6044 | 2.6811 | 2.8005 | 2.5827 | 6.1602 | 0.2933 |
| 22 | 2.7688 | 2.3034 | 2.6686 | 2.8230 | 2.3847 | 2.5897 | 6.1435 | 0.2792 |
| 23 | 2.5235 | 2.8587 | 2.5138 | 2.6369 | 2.9300 | 2.6926 | 5.9087 | 0.2569 |
| 24 | 2.6418 | 2.6301 | 2.5981 | 2.6099 | 2.6686 | 2.6297 | 6.0500 | 0.2521 |