**Софийски университет „Св. Климент Охридски”Факултет по математика и информатика**

**Изследване на паралелизма при генериране на множеството на Манделброт**

Изготвил: Венелин Стаматов, Компютърни науки, ФН:81912

Ръководители: проф. д-р Васил Цунижев, ас. Христо Христов

# **Съдържание:**

1. Анализ

2. Проектиране

3. Тестване

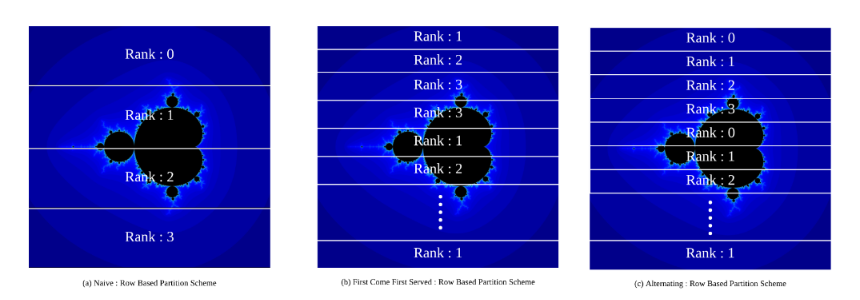
4. Източници

# 1. Анализ

В източник [1] автора изпробва 3 варианта:

1) Статично балансиране с циклично разпределение, при грануларност 1

2) Динамично балансиране чрез Master-slave модел, грануларноста е фина

3) Статично балансиране с циклично разпределение, при фина грануларност

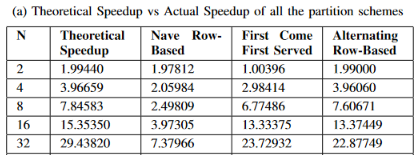
При тестване изплозва следната конфигурация:

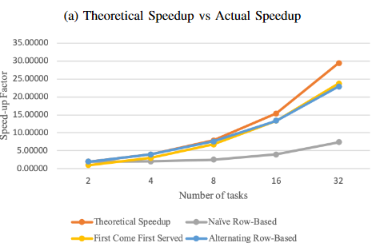


Физическите ядра са 18 на брой

Тества трите варианта с параметри размер на изображението 8000х8000 и 2000 максимум итерации при различен брой използвани процесори.

Резултатите му за втория и третия вариант са обещаващи





В този проект съм се опитал да реализирам третия вариант – статично циклично балансиране по редове, но при варираща гранлуларност(1, 5 и 10), което практически включва като случай и първия вариант.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Образец | Балансиране | Декомпозиция | Грануларност | Максимален паралелизъм | Макс.  итерации |
| [1].2 | статично | По редове | фина | 32 | 2000 |
| [1].3 | динамично |
| Този проект | статично | варираща | 255 |

# 2. Проектиране

Задачата изисква да се генерира за всеки пиксел от HxW матрица, на който съответствa комплексно число c=(r, im), число равно на броя на итерациите които са нужни за да „избяга“ от кръг с радиус 2 и център (0,0) при многократно прилагане на формулата f(x) = x2 + c, започвайки с x = 0.

За целта е реализиран алгоритъм на Java 8, използващ различен брой нишки които работят асинхронно, по модел SPMD, и минимална подзадача един ред от матрицата. Максималният брой на итерациите е 256.

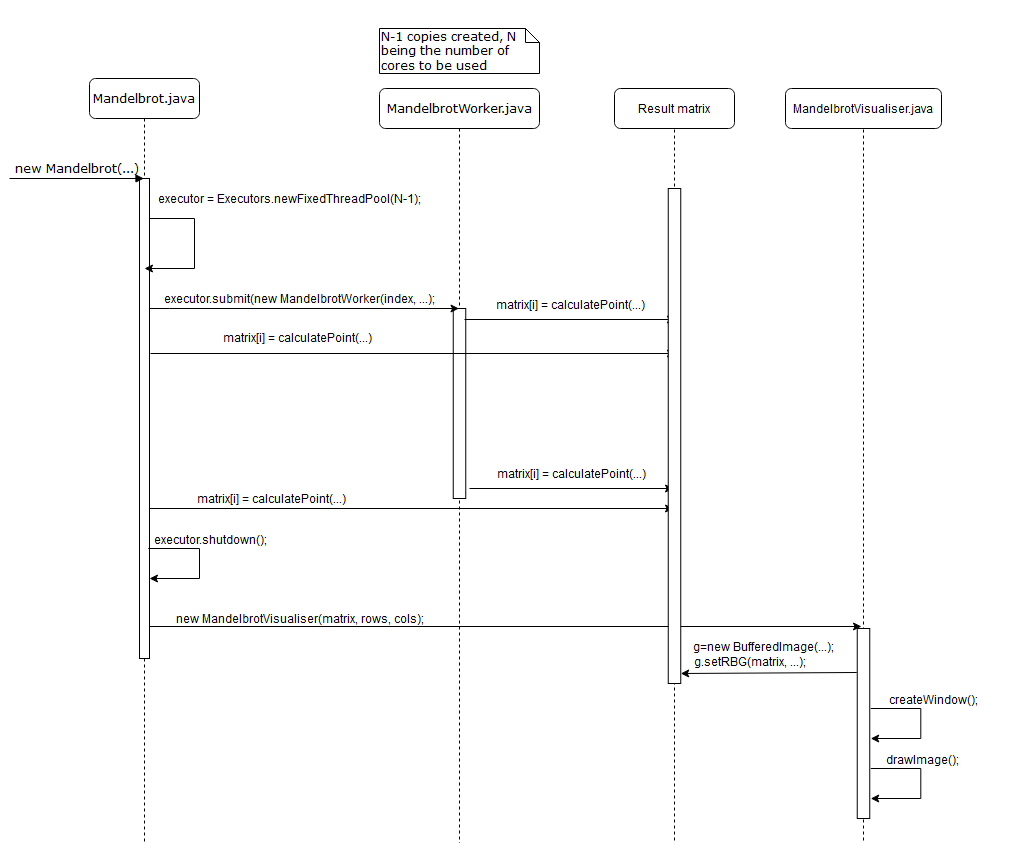
Имплементацията се състои от 3 класа:

Mandelbrot – съдържа алгоритъма за пресмятане на броя на итерациите, като използва оптимизираната версия на алгоритъма (виж [2]), както и масива в който се въвеждат стойностите за съответните пиксели, той е с размер height\*width и се разглежда като матрица където пиксел (x, y) се намира на индекс x\*width + y. Освен това в случай че е въведен параметър за използване на повече от 1 процесор, отговаря за пускането и изчакването на нишките работници, и в случай че е маркирана съответната опция, отваря на екрана прозорец в който е изобразено полученото множество. За манипулация на нишките е използван класа ExecutorService, а самите нишки са от класа MandelbrotWorker който имплементира интерфейса Runnable

MandelbrotVisualiser – отговаря за изобразяването на полученото множество, като използва класовете JFrame и JPanel от пакета javax.swing [3]

MandelbrotWorker – пресмята итерациите на пикселите на частта от матрицата за която отговаря, спрямо грануларността и индекс който се задава при създаване на нишката

Диаграма на последователностите:



# 3. Тестване

Тестването се извършва на сървър с характеристики:

Architecture: x86\_64

CPUs: 32

Model name: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz

CPU MHz: 2699.914

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

L3 cache: 20480K

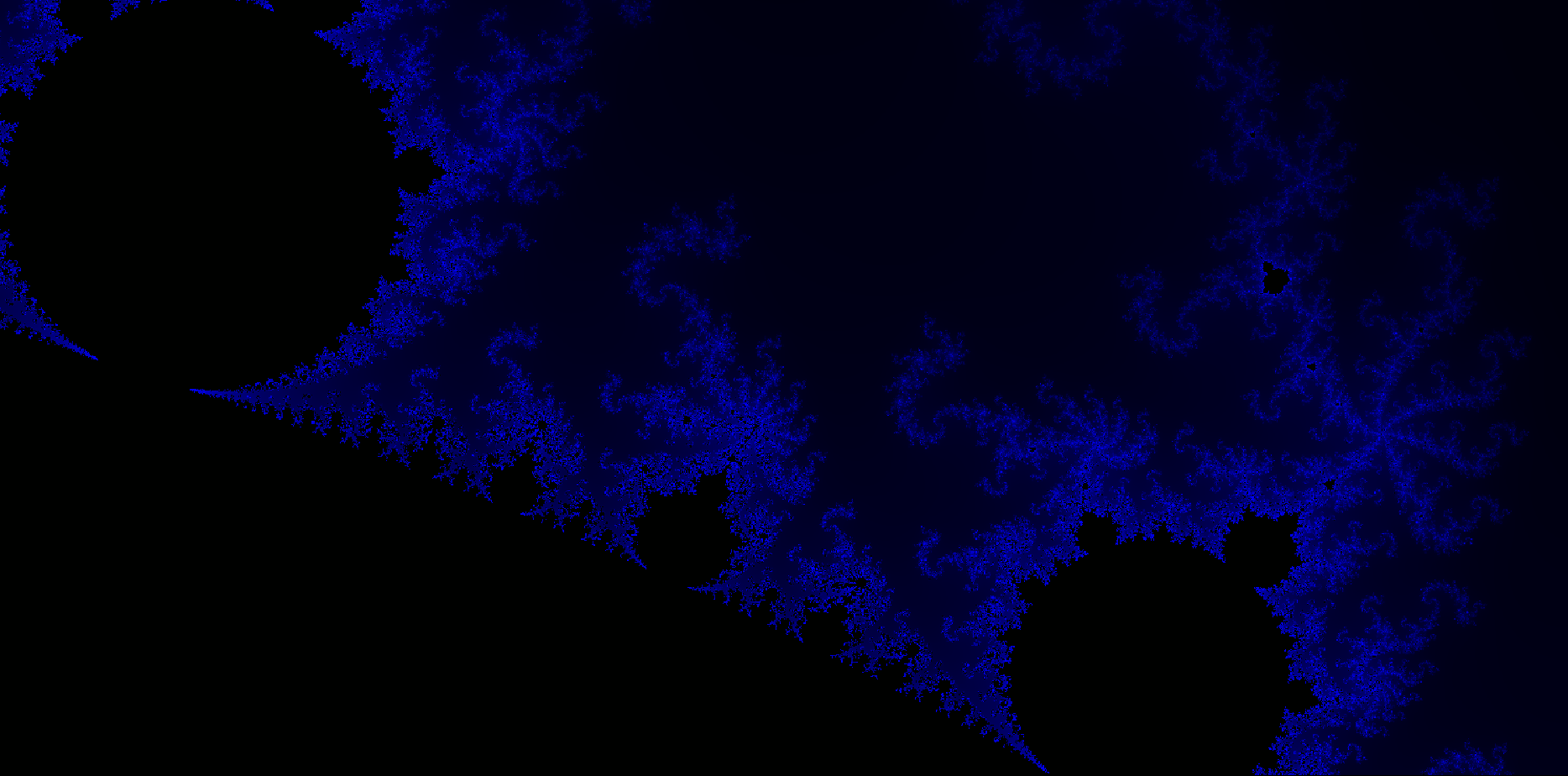
Програмата се изпълнява със следните параметри:

java Main "p" "g" "8000x8000" "-0.93x-0.257x-0.85"

където p и g са съответно броя на процесорите и грануларността, 3600х1800 е резолюцията- широчина х височина на растеризацията, (-0.93,-0.257) са координатите на горния ляв ъгъл, -0.85 е х-координатата на долния десен ъгъл, а у-координатата се пресмята спрямо предишните 3 и резолюцията по формулата

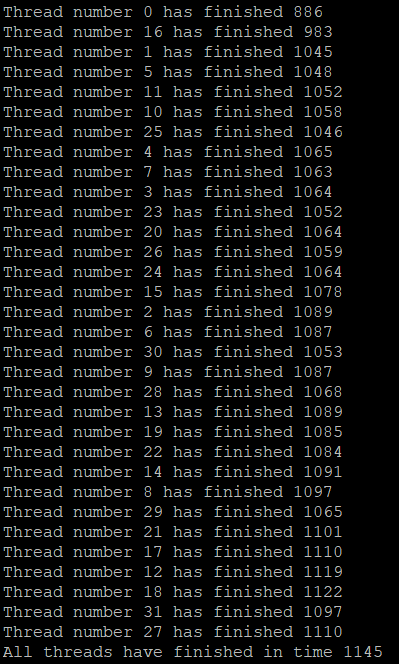
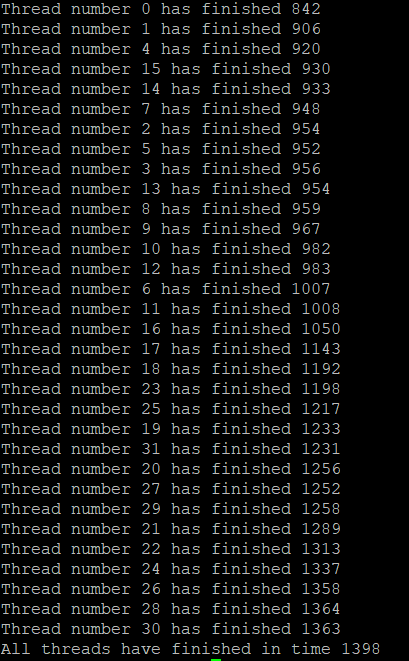
yEnd = yBegin + (xEnd - xBegin)\* (width/height)

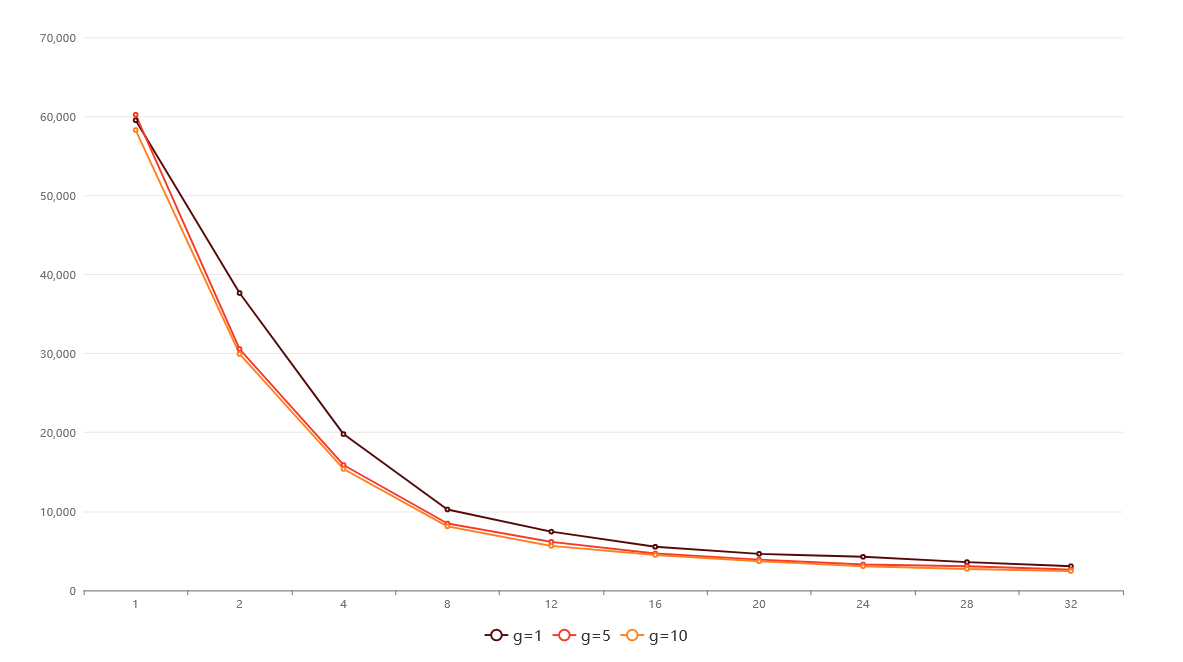
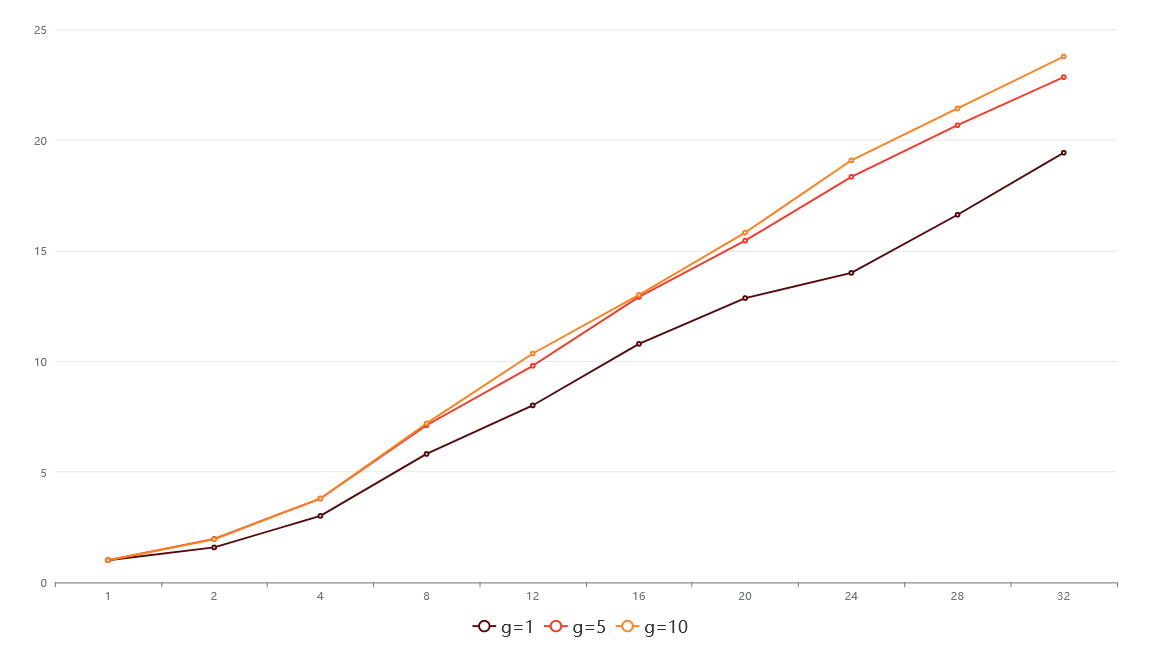
Областта с дадените координати е избрана с цел да демонстрира разликата в натоварването на нишките при различните грануларности, тъй като „черната“ част от областта изисква значително повече работа от оцветената част. В крайна сметка генерираната област при тези параметри изглежда така:



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***p*** | *g* | ***Тp* (1)** | ***Тp* (2)** | ***Тp* (3)** | ***Тp=*min(*Tp*(*i*))** | ***Sр* = *T*1*/Tp*** | ***Eр* = *Sp/p*** |
| 1 | 1 | ***59527*** | ***60837*** | ***61302*** | **59527** | 1 | 1 |
| 2 | 1 | *37645* | *37740* | *37465* | **37465** | 1.5888696 | 0.79443481 |
| 4 | 1 | *19792* | *19857* | *20167* | **19792** | 3.0076293 | 0.75190734 |
| 8 | 1 | *10248* | *10069* | *9904* | **9904** | 6.0103998 | 0.75129998 |
| 12 | 1 | *7437* | *7547* | *7569* | **7437** | 8.0041683 | 0.66701403 |
| 16 | 1 | *5519* | *5565* | *5639* | **5519** | 10.785831 | 0.67411442 |
| 20 | 1 | *4630* | *4657* | *4705* | **4630** | 12.856803 | 0.64284017 |
| 24 | 1 | *4253* | *4294* | *4236* | **4236** | 14.052644 | 0.58552683 |
| 28 | 1 | *3581* | *3632* | *3678* | **3581** | 16.62301 | 0.59367894 |
| 32 | 1 | *3064* | *3058* | *3079* | **3058** | 19.465991 | 0.60831221 |
| 1 | 5 | ***60218*** | *58818* | *57753* | **57753** | 1 | 1 |
| 2 | 5 | *30555* | *31140* | *31201* | **30555** | 1.8901325 | 0.94506627 |
| 4 | 5 | *15880* | *15969* | *16114* | **15880** | 3.6368388 | 0.9092097 |
| 8 | 5 | *8473* | *8444* | *8371* | **8371** | 6.8991757 | 0.86239697 |
| 12 | 5 | *6149* | *6097* | *6154* | **6097** | 9.4723635 | 0.78936362 |
| 16 | 5 | *4665* | *4707* | *4721* | **4665** | 12.380064 | 0.77375402 |
| 20 | 5 | *3897* | *3842* | *3791* | **3791** | 15.234239 | 0.76171195 |
| 24 | 5 | *3284* | *3258* | *3215* | **3215** | 17.963608 | 0.74848367 |
| 28 | 5 | *2913* | *2864* | *2881* | **2864** | 20.165154 | 0.72018406 |
| 32 | 5 | *2636* | *2679* | *2656* | **2636** | 21.909332 | 0.68466664 |
| 1 | 10 | ***58283*** | *57698* | *56864* | **56864** | 1 | 1 |
| 2 | 10 | *29934* | *29485* | *28885* | **28885** | 1.9686342 | 0.98431712 |
| 4 | 10 | *15373* | *15130* | *15400* | **15130** | 3.7583609 | 0.93959022 |
| 8 | 10 | *8107* | *8087* | *8054* | **8054** | 7.0603427 | 0.88254284 |
| 12 | 10 | *5631* | *5647* | *5752* | **5631** | 10.098384 | 0.841532 |
| 16 | 10 | *4483* | *4532* | *4613* | **4483** | 12.684363 | 0.7927727 |
| 20 | 10 | *3785* | *3715* | *3699* | **3699** | 15.372803 | 0.76864017 |
| 24 | 10 | *3054* | *3011* | *3016* | **3011** | 18.88542 | 0.78689251 |
| 28 | 10 | *2719* | *2754* | *2784* | **2719** | 20.913571 | 0.74691326 |
| 32 | 10 | *2451* | *2494* | *2474* | **2451** | 23.200326 | 0.7250102 |

Тук може да се види разликата в натоварването на нишките при грануларност 1 и 10 и паралелност 32:



Време и ускорение:

Ефективност: 

4. Източници:

[1] Bhanuka Manesha Samarasekara Vitharana Gamage and Vishnu Monn Baskaran, “Efficient Generation of Mandelbrot Set usingMessage Passing Interface”, https://arxiv.org/pdf/2007.00745.pdf

[2] Optimized escape time algorithm: https://github.com/aomader/mandelbrot/blob/master/src/mandelbrot/core/Algorithm.java

[3]Java Swing: <https://www.javatpoint.com/java-swing>

[4] Java Executor: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html