

Курсов проект

по

Разпределени софтуерни архитектури

летен семестър, учебна година 2019/2020

**Тема 17**

**Изобразяване на фрактал по формулата**

Ръководители : проф. Васил Цунижев, ас. Христо Христов

Изготвил: Иван Ивов Чучулски

Факултетен номер : 62167

Специалност : Софтуерно инженерство

Дата : Проверил :……………………………..

1. Анализ
   1. същност на проблема
   2. разгледани решения
   3. реализирано решение
      1. език, библиотеки, софт модел (master-slaves)
2. Проектиране
   1. клас диаграма
   2. модел на паралелизма,
   3. интерфейс
3. Тестване
   1. среда, на която са проведени тестовете
   2. резултати
4. Анализ на резултатите
5. Източници

--------------------------------------------------------

1. **Анализ**
   1. **същност на проблема**

Фракталите са математически обекти, които представляват множество от точки върху комплексната равнина. Те са детайлно изследвани за пръв път в средата на 20-ти век от математика Беноа Манделброт. Фракталите намират приложение в моделирането на структурата на природни обекти и взаимодействията между тях, фрактална графика за компютърни изображения, анализ и компресия на сигнали, метод за създаване на картини и архитектурен дизайн и други.

Целта на проекта е да се създаде програма, която да визуализира фрактал, който е зададен чрез рекурентна формула **.**

Реализирани са две програми, които използват асинхронна паралелна обработка на escape-time алгоритъм за пресмятане и визуализиране на множеството на Манделброт. Първата програма използва статично балансиране на изпълнението,

да се допълни ….

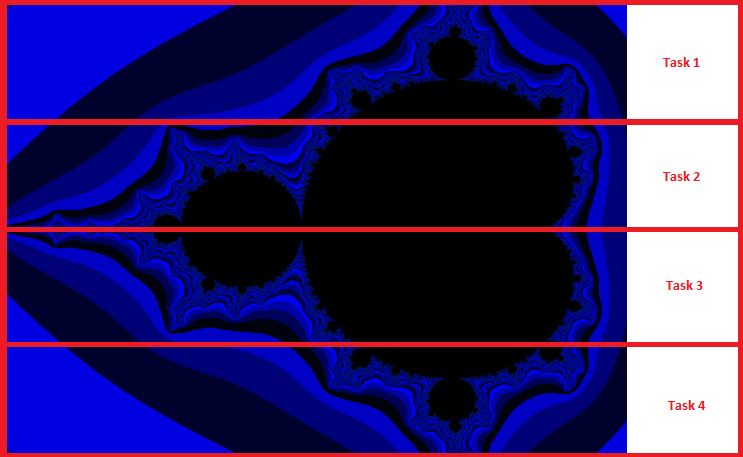
* 1. **преглед на функционалността на приложения, които пресмятат множеството на Манделброт (функционален анализ)**

Източник [1] представя математическа дефиниция на множеството на Манделброт и escape-time алгоритъма за пресмятането и оцветяването на множеството. За да определим дали точка от комплексната равнина принадлежи на множеството на Манделброт, трябва да пресметнем границата на редицата, чиито членове се получават при прилагане на зададената рекурентна формула започвайки от точката . Ако границата на редицата от комплексни числа не клони към безкрайност при пресмятане на безбройно много членове на тази редица, то точката принадлежи на множеството. Разбира се, безкрайността не може да бъде точно моделирана в компютърните изчисления, затова идеята на escape-time алгоритъмът е да използваме големината на модула на комплексното число за определяне дали точката принадлежи на множеството. Задаваме число, което обозначава максималния брой итерации, които правим на всяка точка и пресмятаме броя на итерациите, за които модулът на числото остава по-малък от дадена стойност. Отношението на максималния брой итерации и броя итерации за дадена точка можем да използваме за задаване на цвят на точката при генериране на изображението. В [1] е представена реализация на последователна програма, която използва escape-time алгоритъма на езика C#.

Източник [2] представя два начина за имплементиране на паралелна реализация на escape-time алгоритъма на езика C и програмата да позволява местене в полученото изображение. Първо е направена декомпозиция по данни SPMD и статично балансиране на изпълнението. Създаването и управлението на паралелните процеси става по модела Master-Slaves. Master частта разделя изображението на блокове, които се състоят от реда и колони. След това създава изпълняващите процеси и ги пуска. Първата нишка обработва първия такъв блок, втората работи върху втория и така нататък, докато всички не приключат.

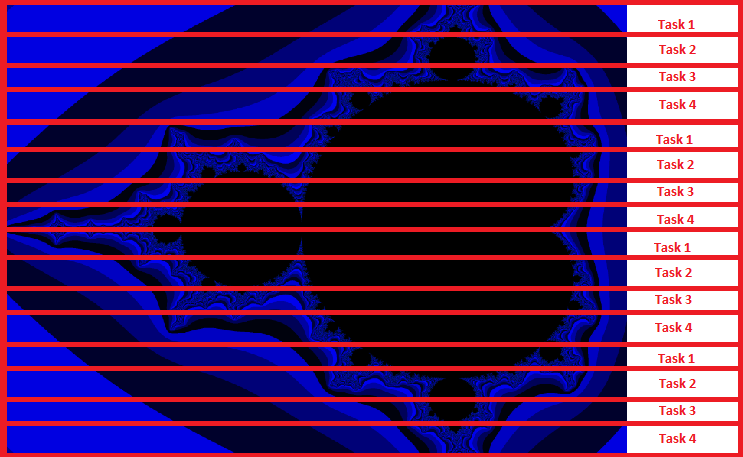
След това е представено решение, като грануларността е средна и е използвано динамично балансиране с централен процес, т.е. топология звезда. Изображението е разделено на задания, като заданията представляват хоризонталните редове на изображението. Master частта създава масив, която да представлява заданията, след това създава обработващите процеси и ги пуска да работят. Те трябва да достъпват масива със заданията, за да вземат задание, след това го да го обработят и отново да потърсят задание, докато няма повече задания. Master частта изчаква всички Slaves да приключат и след това програма приключва. Обозначено е, че достъпването на масива със заданията трябва да не наруши структурата от данни и следователно обработващите процеси трябва да осигурят синхронизация помежду си, когато взимат задание.

Източник [3] представлява имплементация на първата идея от източник [2] на езика Java,т.е. Master-Slaves програма, при която е направена SPMD декомпозиция по данните на блокове с големина реда и колони, и е използвано статично балансиране на изпълнението.



Фигура 1. Декомпозиция на данните на блокове и статично балансиране на изпълнението, източник [3]

Също така е представена втора имплементация на Master-Slaves програма, при която е направена SPMD декомпозиция по данните и използвана по-фина грануларност, като са задавани различен брой редове, които да се съдържат в даден блок. Балансирането на изпълнението отново е статично.



Фигура 2. Декомпозиция на данните по редове и статично балансиране на изпълнението, източник [3]

Показани са резултати от тестване….

// туй мойто вече

При този метод има няколко проблема като на първо място поради едрата грануларност е възможно някои нишки да получат блок, който се състои главно от точки от множеството, като това означава по-тежки сметки и така тяхното изпълнение ще се забави. Тъй като балансирането е статично, други нишки, които са получили по-лек блок и са приключили работата си по-бързо няма как да помогнат на тези, които се бавят.

За решението на тези проблеми първо е използването на средна грануларност, като заданията няма да бъдат блокове от по няколко реда, а всеки ред от изображението се прави на задание.

Също така може да се направи различно статично балансиране, при което нишките няма да обработват последователни редове, а вместо това . Т.е. дадена нишка обработва първо реда, който съответства на нейния индекс, после „прескача“ „“ редове напред, пак обработва един ред и пак „прескача“ докато не се изчерпат редовете.

* 1. **нефункционален анализ**

1. **Проектиране**
2. **Тестване**
3. **Анализ на получените резултати**
4. **Източници**

[1]Bastian Fredriksson, “An introduction to the Mandelbrot set”, публикуван: 01.2015г, <https://www.kth.se/social/files/5504b42ff276543e4aa5f5a1/An_introduction_to_the_Mandelbrot_Set.pdf>

[2]Douglas Thain, „University of Notre Dame, Operating System Principles“, 17.02.2020г, <https://www3.nd.edu/~dthain/courses/cse30341/spring2020/project3/>

[3]Thomas Uhrig, “Calculating Mandelbrot Set with Java Tasks”, публикуван: 05.11.2012г, <https://tuhrig.de/calculating-mandelbrot-set-with-java-tasks/>

<https://bitbucket.org/wordless/mandelbrot/src/master/Mandelbrot/>

[4]Brian Goetz, “Java Concurrency In Practice” 2006, издател : “Addison-Wesley”

[5]Eugen Paraschiv, “LinkedBlockingQueue vs ConcurrentLinkedQueue”, последна модификация:03.06.2020г,

<https://www.baeldung.com/java-queue-linkedblocking-concurrentlinked>

1. **Списък с фигури**

Фигура 1. Декомпозиция на данните на блокове и статично балансиране на изпълнението., източник [3]