Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Бакалавърска програма „Софтуерно инженерство”

Курс: Разпределени софтуерни архитектури

Летен семестър, 2018/2019 год.

Курсова работа

Задача 19: Изобразяване на фрактал:

Множество на Манделброт

Изготвил: Борис Глузов

Специалност: Софтуерно инженерство

Факултетен номер: 62076

Научни ръководители:

Христо Христов

проф. д-р. Васил Георгиев

Юни, 2019

Съдържание

1 Въведение: Множество на Манделброт

Фракталите са нов клон в математиката, който често бива определян като граничещ с изкуството. Една от ключовите фигури в изследването на фракталите е Беноа Манделброт, който им дава това име през втората половина на двадесети век.

Множеството на Манделброт се определя от точки в комплексната равнина. За да се определи това множество е необходима употребата на алгоритъм, базиран на следната рекурсивна формула: $Zn=Zn-1+C$, където C е точка от комплексната равнина.

Горепосочената формула разделя точките в комплексната равнина на две категории: точки, принадлежащи на множеството на Манделброт, и точки, непринадлежащи на множеството на Манделброт.

2 Условие на задачата

Задачата ни е да напишем програма за визуализиране на множеството на Манделброт, определено формулата $F(Z)=C*e^{-z}+Z^2$. Изискванията към програмата са следните:

- Програмата трябва да използва паралелно изпълнение на няколко нишки, за да разпредели работата по търсенето на точките от множеството на Манделброт, за да се възползва от допълнителните ядра на един процесор.
- Програмата трябва да осигурява генерирането на изображение (например **.png**), показващо така намереното множество.
- Програмата трябва да позволява да ѝ се задават следните параметри при изпълнението ѝ през команден интерфейс:
    - Команден параметър, който задава размерността на генерираното изображение като широчина **(width)** и височина **(height)** в брой пиксели:
      „-s 640x480“ (или **„-size“**).
      При невъведен параметър, програмата да приема стойност по подразбиране за параметъра: ***„640x480“***
    - Команден параметър, който 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 да се интерпретира както следва: **a∈[−2.0,2.0]**, **b∈[−1.0,1.0]**.
При невъведен параметър, програмата да приема стойност по подразбиране за параметъра: ***„-2.0:2.0:-2.0:2.0“***

- Команден параметър, който задава името на генерираното изображение.
  Той има вида:**„-o zad16.png“** (или **„-output“**). Програмата да записва генерираното изображение във файл със зададеното име.

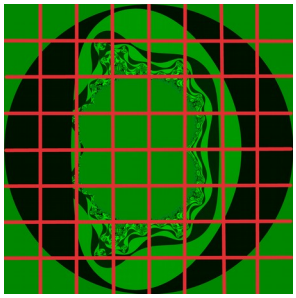
  При невъведен параметър, програмата да приема стойност по подразбиране за параметъра: ***„zad16.png“***

- Команден параметър, който задава максималния брой нишки (паралелни процеси), на които се разделя работата по генерирането на изображението: **„-t 3“** (или **„-tasks“**).
  При невъведен параметър, програмата да приема стойност по подразбиране за параметъра: ***„1“***

- Команден параметър, който задава на програмата „тих“ режим на работа. Когато програмата работи в „тих“ режим, се извежда еднинствено времето, през което програмата е работила. Тихият режим не отменя записването на изображението във изходния файл. Параметърът, който задава „тихия“ режим на работа на програмата трябва да бъде **„-q”** (или **„-quiet”**).
  При невъведен параметър, програмата да *приема, че не е зададен „тих“ режим.*

- Команден параметър, който задава броя на итерации, който да се използва при избирането на това дали една точка принадлежи на множеството. Параметърът, който задава броя итерации е “-i” или “-iters” последван от броя на итерациите. По подразбиране програмата ще изпозлва настройката “-iters 1000”.

- Команден параметър, който задава дали изображението да се разделя на плочици, т.е. във вида, и за всяка плочица да се



пресмята отделно. Като работата се разпределя между нишките по-оптимално. Параметърът, който задава този режим е “-tiled” и по подразбиране е изключен.

- Програмата трябва да 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. Примери за подходящи съобщения са:
  - **„Thread-<num> started.“**
  - **„Thread-<num> stopped.“**
  - **„Thread-<num> execution time was (millis): <num>“**
  - **„Threads used in current run: <num>“**
  - **„Total execution time for current run (millis): <num>“**

3. Описание на реализирания алгоритъм

За визуализирането на множеството на Манделброт, определено от формулата

$$F(Z)=C*e^{-z}+Z^2$$

се използва паралелизъм по данни. Множеството от входни данни се разделя на подмножества (блокове/парчета), върху които паралелно се изпълняват едни и същи операции.

Използвам **escape time** алгоритъма. Един от най-простите за реализиране на множеството на Манделброт. Какво представлява той, за всеки пиксел, минава през формулата $Z_{n+1}=Z_n^2+C$ докато $|Z_n|>2$ или максималния брой итерации е достигнат. Ще използвам резултата от итерациите, за да сложа определен цвят на всеки пиксел.

Въпросът, който остава да бъде решен, е как да присвоите цветове на точка от множеството на Манделброт. Първият ми опит беше да използвам map между броя на повторенията и броя на възможните стойности за RGB изображение. Започваме като разделяме броя на итерациите, съответстващи на конкретен пиксел, на максималния брой итерации, това ни дава число t на единичния интервал. RGB изображение може да има 256^3 цветове, така че ние умножаваме t с 256^3 и използваме резултата като индекс от масива с цветове.

Как се разделя работата по нишките, височината на зададената резолюция се разделя на подадения брой нишки. И всяка нишка започва да рисува и да итерира по широчина.

Начина на разделяне на работа между нишките е реалзиран чрез, разделянето на височината на изображението на части върху броя на нишките при non-tiled имплементацията. Ако параметъра tiled е избран изображението се разделя на numThreads * numThreads части по ширичина и височина, както показано на снимката по-горе. Всяка нишка си взима ресурс синхронизирано и започва работа по него.

1 Реализация

1.1 Класове

Първоначалната реализация на програмата, решаваща поставения проблем включва три основни класа:

- **ImageParams –** Този клас държи всички параметри свързани с генериране на изображението, както и вървещите нишки. Също така в него става и parse-ването на подадените параметри при пускането на програмата.
- **MandelbrotPiece –** този клас приема параметрите от **ImageParams** и спрямо тях изпълнява **escape time** алгоритъма. Какво вече споменах, за всеки пиксел, минава през формулата $Z_{n+1}=Z_n^2+C$ докато $|Z_n|>2$ или максималния брой итерации е достигнат и използва резултата от итерациите, за да сложи определен цвят на всеки пиксел.
- **LoggingMandelbrotPiece –** наследник на MandelbrotPiece, като единствената разлика е това че изкарва времето на стартиране до края на изпълнението. Понеже много нишки се очаква да output-ват времето си за изпълнение, този клас държи в себе си mutex който се използва за синхронизация на достъпа до std::cout
- **PieceFactory –** създава MandelbrotPiece
- **LoggingPieceFactory –** създава LoggingMandelbrotPiece
- **PieceManager –** този клас отговаря за пускането на нишките паралелно и тяхната синхронизация. Отговаря и за това колко време работи всяка нишка, като след като и последната нишка се изпълни, изкарва времето за което програмата е изрисувала фрактала.
- **TinyPngOut** – това е библиотеката използвана за записване на PNG изображения.

1.1 Резултат

Резултатът от общата работа на класовете в програмата е получаване на времето ѝ на изпълнение според броя на използваните нишки, както и генерирането на изображение, демонстриращо Множеството на Манделброт в зададената комплексна равнина.

1.2 Подобрение

При non-tiled версията, грануларността е твърде голяма, пък и части от самото изображение са наистина лесни за пресмятане което значи че някои от нишките ще приключат много бързо в сравнение с други. Поради тази причина е добавен Tiled параметър, който разделя по-умно изображението на части и така всяка MandelbrotPiece задача може да си вземе ново парче след като приключи с предишното. Възможно е синхронизацията при взимането на нов ресурс да забавя нишките, но пък така сме сигурни че няма на няколко нишки да им се паднат най-трудните части от изображението, ако разпределението беше само вертикално или само хоризонтално.

2 Резултати от проведените тестове и измервания

За тестване и измерване на резултатите е използвана предоставената двупроцесорна машина с 16 физически ядра, т.е. 32 логически ядра.

Направени са тестове за визуализиране на Множеството на Манделброт, определено от формулата. Получените данни за изображение с размери 2000x2000 (1:1) и граници за комплексната равнина:

- Реална част, принадлежаща от -2.0 до 2.0
- Имагинерна част, принадлежаща от -2.0 до 2.0

1.1 Резултати в табличен вид

Non-tiled (coarse grained)

Threads	Time	Optimal Time	Speedup	Optimal Speedup	Efficiency	Optimal Efficiency
1	279,000.00	279,000.00	1.00	1.00	1.00	1.00
2	130,100.00	139,500.00	2.14	2.00	1.07	1.00
4	128,954.00	69,750.00	2.16	4.00	0.54	1.00
8	70,000.00	34,875.00	3.99	8.00	0.50	1.00
12	57,500.00	23,250.00	4.85	12.00	0.40	1.00
16	36,362.00	17,437.50	7.67	16.00	0.48	1.00
20	32,512.00	13,950.00	8.58	20.00	0.43	1.00
24	28,098.00	11,625.00	9.93	24.00	0.41	1.00
28	23,298.00	9,964.29	11.98	28.00	0.43	1.00
32	18,291.00	8,718.75	15.25	32.00	0.48	1.00

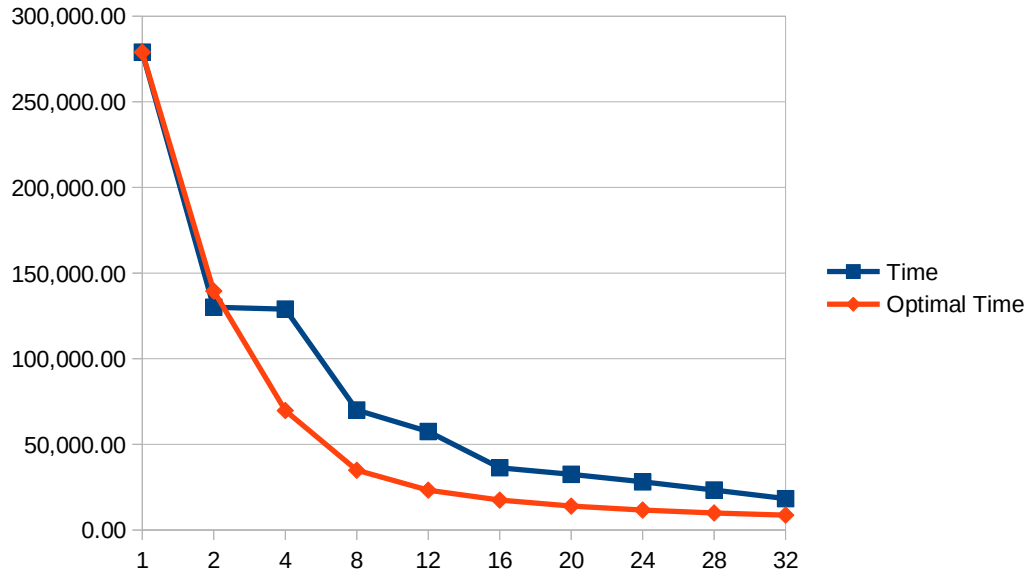
Tiled (fine grained) “-tiled”

Threads	Time	Optimal Time	Speedup	Optimal Speedup	Efficiency	Optimal Efficiency
1	279,000.00	279,000.00	1.00	1.00	1.00	1.00
2	105,108.00	139,500.00	2.65	2.00	1.33	1.00
4	64,361.00	69,750.00	4.33	4.00	1.08	1.00
8	38,238.00	34,875.00	7.30	8.00	0.91	1.00
12	22,401.00	23,250.00	12.45	12.00	1.04	1.00
16	16,657.00	17,437.50	16.75	16.00	1.05	1.00
20	15,440.00	13,950.00	18.07	20.00	0.90	1.00
24	13,996.00	11,625.00	19.93	24.00	0.83	1.00
28	12,920.00	9,964.29	21.59	28.00	0.77	1.00
32	11,291.00	8,718.75	24.71	32.00	0.77	1.00

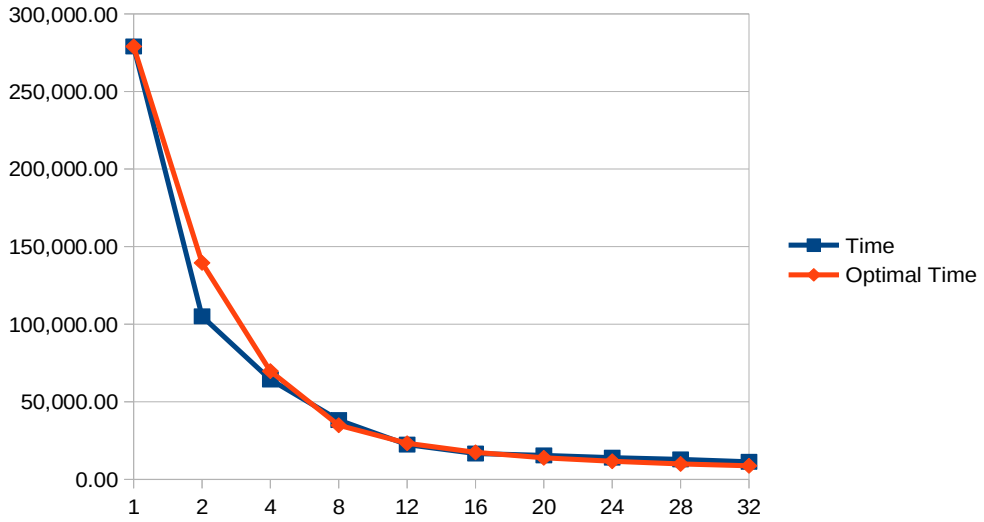
1.2 Резултати под формата на Chart

- Оптималното време за изпълнение на програмата срещу постигнатото време за изпълнение на програмата за различен брой нишки

Non-tiled

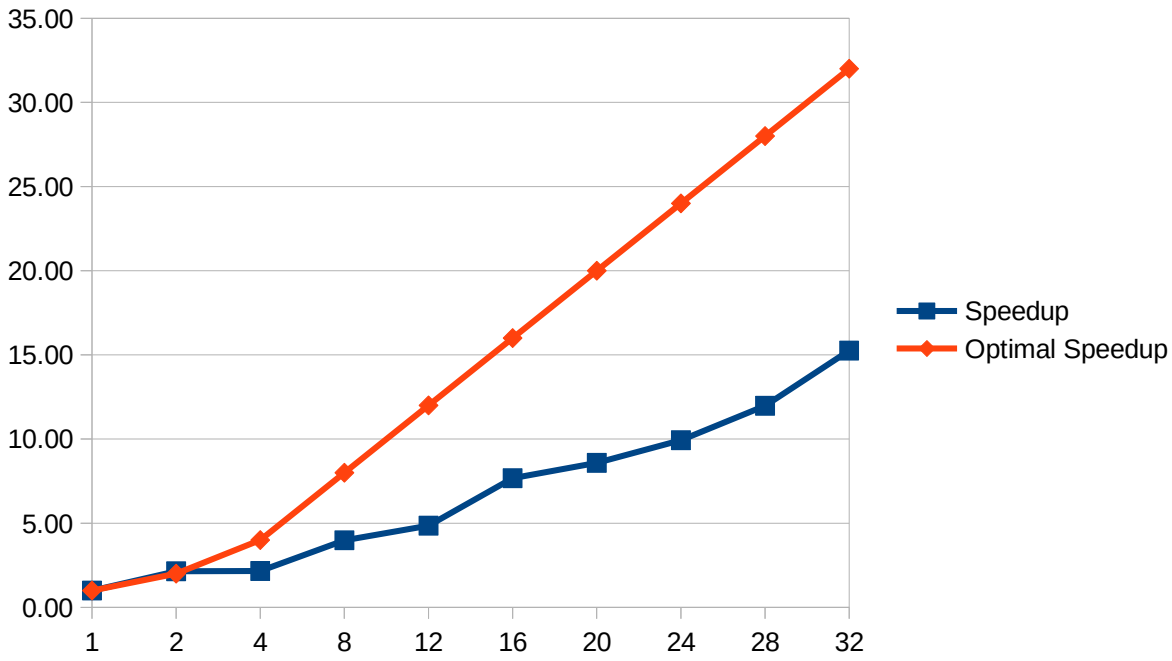


Tiled

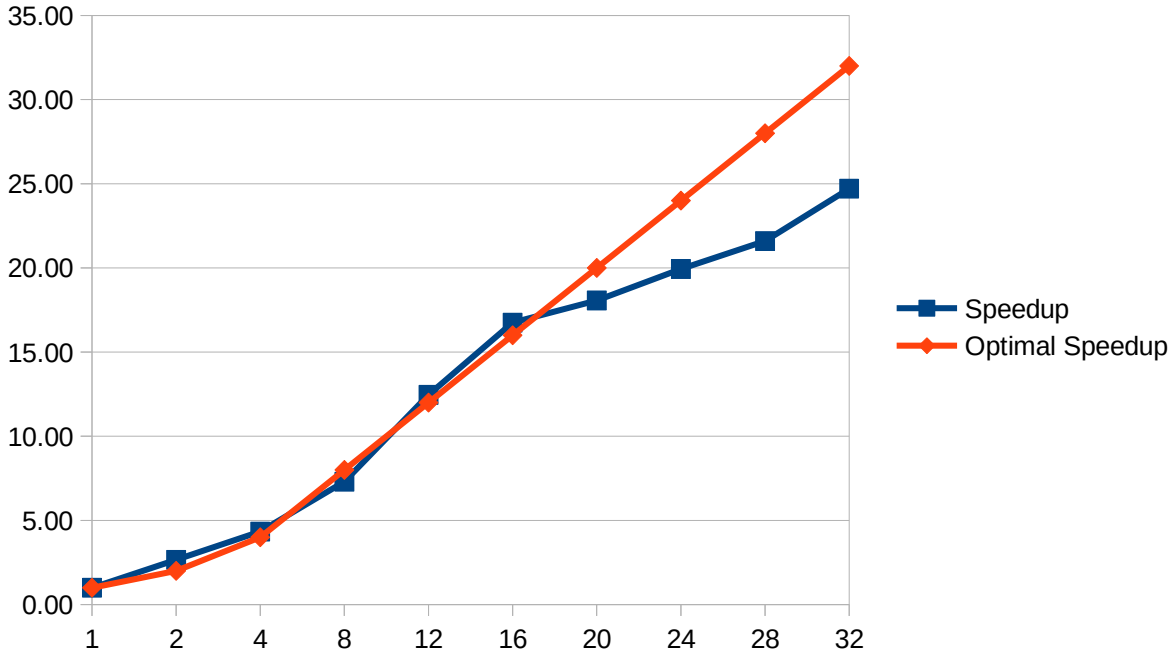


- Оптималното ускорение срещу постигнатото ускорение при изпълнението на програмата за различен брой нишки

Non-tiled

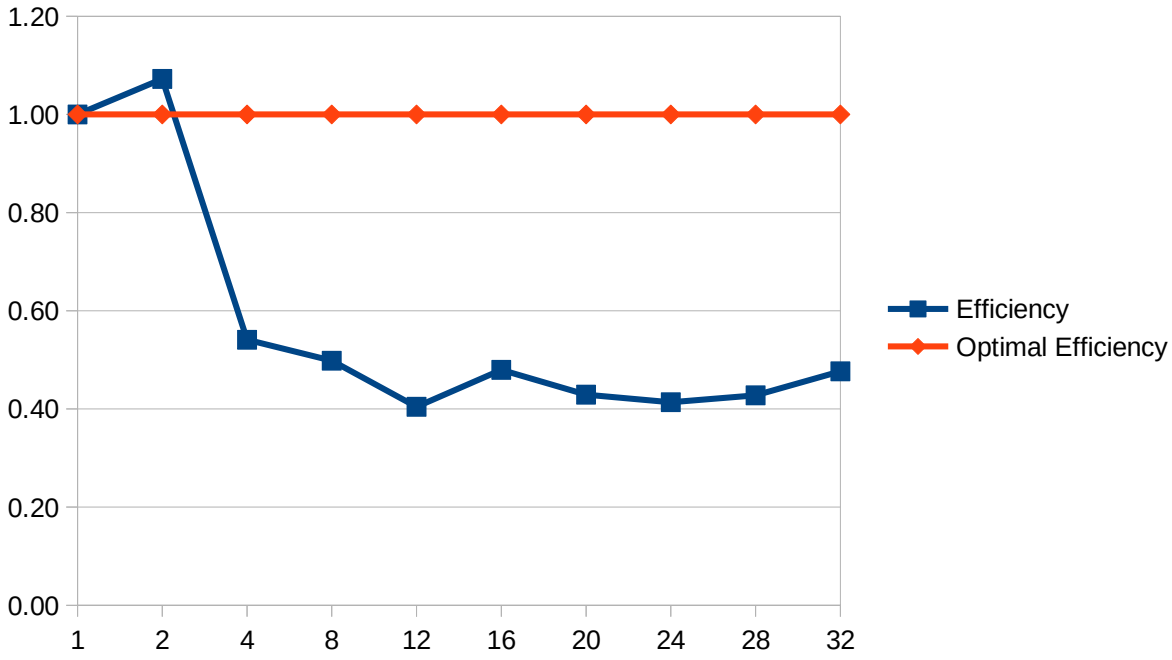


Tiled

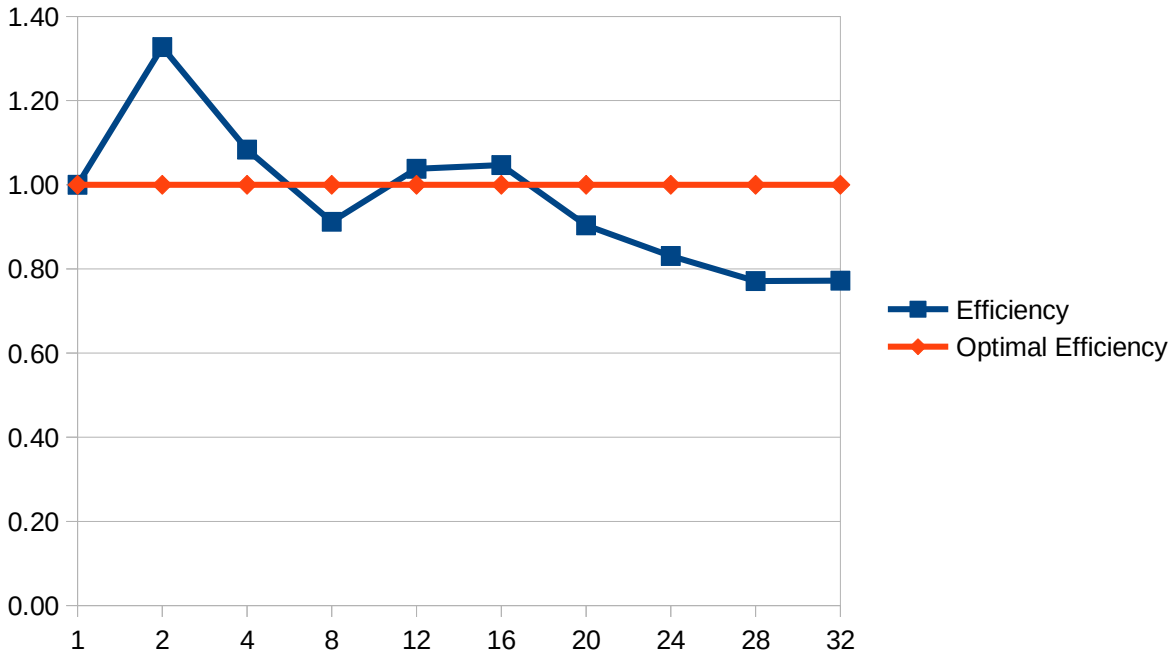


- Оптималната ефикасност при изпълнение на програмата срещу постигнатата ефикасност при изпълнението на програмата за различен брой нишки

Non-tiled



Tiled


1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
1
2
4
8
12
16
20
24
28
32
Efficiency
Optimal Efficiency