Софийски университет “св. Климент Охридски”

Факултет по математика и информатика

Курсов проект

По Системи за паралелна обработка

Тема - изобразаване на фрактал: Mandelbrot set

Изготвил: Александър Витков, ФН 81618

Проверил: ……...

Задача

Имплементация на паралелен алгоритъм, генериращ изображение на даден регион от множеството Манделброт, дефинирано от формулата **F(Z) = C * cos(Z)**

Реализация

Имплементацията е на Rust, като са използвани следните third party библиотеки:
`clap` - за четене на command line аргументите
`num` - операции с комплекни числаа
`png` - png encoder
`crossbeam` - thread pool

Командни аргументи и default стойности:

`--size 640x480`
Задава размер на изходното изображение. В този документ ще наричаме тези променливи `output_width` и `output_height`

`--rect -2.0:2.0:-1.0:1.0`
Задава регион от комплексната равнина, който да бъде представен в изходното изображение

`--tasks 1`
Брой нишки, работещи върху изображението

`--granularity 1`
Грануларност на разбиването на данните

`--quiet`
Тих режим

`--output output.png`
Име на изходния файл

`--iter 50`
Брой итерации на алгоритъма за всеки пиксел

`--nooutput`
Няма да бъде генерирано изходно изображение

Ход на действие на програмата

В главната нишка, след успешно четене на command-line аргументите, програмата заделя масив с размер `buffer_size := output_width * output_height * 3`, който worker нишките ще запълнят с цветовите данни за изображението.

Този масив бива разбит на непресичащи се подмасиви, чиито размер зависи от броя нишки и грануларността, зададени от потребителя:
`chunk_size := buffer_size / (tasks * granularity)`

Създава се опашка `chunks_queue`, която държи указатели към парчетата от масива. Тя бива заключен под mutex, за да не стават лоши неща.

Създаван се `tasks` на брой нишки, всяка която действа по следния алгоритъм:

- В цикъл:
  - Изчакай mutex-а на опашката `chunks_queue` и го заключи
    - Ако опашката е празна, готови сме
    - Ако не, вземи си елемент `chunk` от нея.
  - Върни mutex-а на `chunks_queue` в дивата природа
  - Запълни `chunk` с нужните данни за изображението

Главната нишка изчаква worker нишките да приключат, след което извиква png encoder-а да си върши магиите, с което програмата завършва.

Модел на паралелизъм

Проблемът за генериране на Манделротското множество е тривиален за паралелизиране, понеже пресмятането на стойността за определен пиксел не зависи от другите пиксели - няма значение в какъв ред ги генерираме. Алгоритъм е от тип task decomposition е естествен за задачата.

Моята имплементация разбива огромния масив на подмасиви, но не дефинира предварително коя нишка върху кои подмасиви ще работи - всяка нишка си взима подмасив от опашката, като приключи с предишния.

Фактът, че имаме mutex не bottleneck-ва програмата, защото работещите нишки го държат заключен, само докато pop-нат указателя към следващия си подмасив, и пускат mutex-а преди да започнат да вършат работа върху изображението. Времето, в което държат mutex-а заключен е пренебрежимо в сравнение с времето за работа.

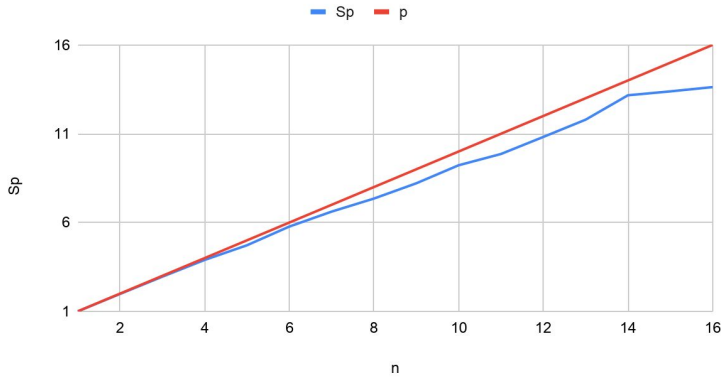
Тестване на програмата

Програмата е качена в https://github.com/alexvitkov/mandelbrot заедно с Dockerfile. Docker image-а инсталира Rust в контейнера, компилира програмата и я слага в $PATH. Entrypoint-ът е shell, така че контейнера трябва да бъде стартиран с -it флаг. Вътре може да се ползва програмата “mandelbrot”. Изваждането на output изображението оставям като упражнение за читателя.

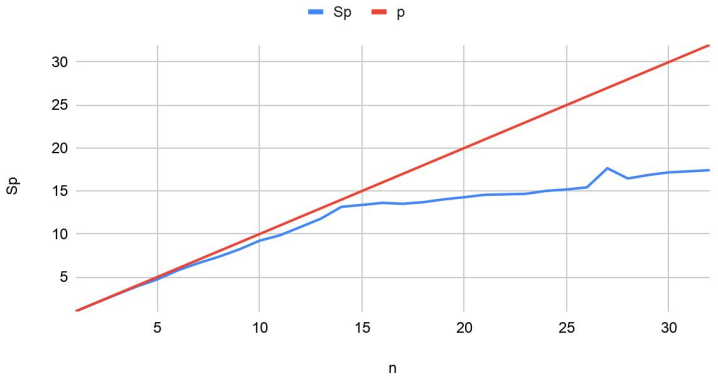
Резултати, описани отдолу, са от сървъра t5600.rmi.yaht.net със следните флагове:
`--size 15000x15000`
`--nooutput` - png encoder-ът е еднонишков и бавен за големи изображения
`--tasks 1 до 32`
`--granularity 16` - провеждането на тестове с друга грануларност отново оставям като упражнение за читателя

Понеже сървърът има 16 физически и 32 логически ядра, съм разделил диаграмите на две - левите са с данни до 16 нишки, десните са до 32 нишки. Причината да ги разделя е, че hyperthreading-ът ме кара да изглеждам зле.

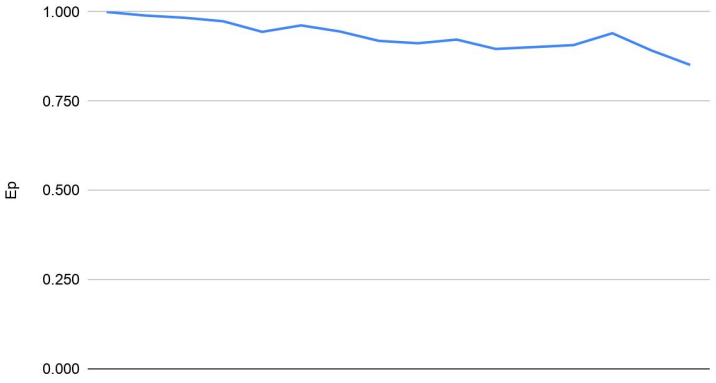
Ускорение, само с физическите ядра



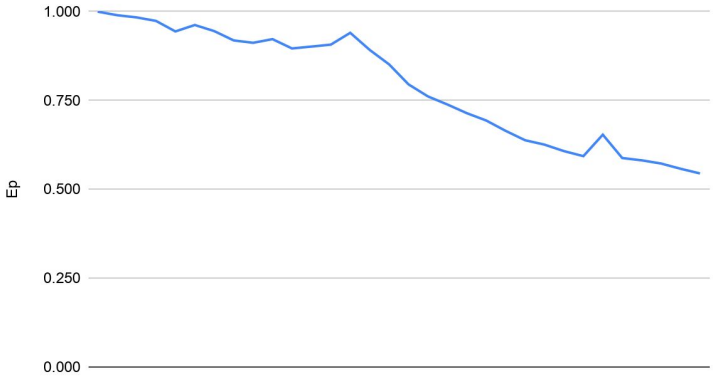
Ускорение, с логическите ядра



Ефикасност, само с физическите ядра



Ефикасност, с логическите ядра



Fastest е най-бързата нишка за определено p, Tn е най-бавната, която съвпада с общото време за работа.

p	Fastest	Tn	Sp	Ep
1	40.143s	40.143s	1.000	1.000
2	20.268s	20.275s	1.980	0.990
3	13.421s	13.601s	2.951	0.984
4	10.217s	10.304s	3.896	0.974
5	8.231s	8.502s	4.722	0.944
6	6.915s	6.952s	5.774	0.962
7	6.011s	6.067s	6.617	0.945
8	5.360s	5.460s	7.352	0.919
9	4.793s	4.889s	8.211	0.912
10	4.310s	4.351s	9.226	0.923
11	3.913s	4.071s	9.861	0.896
12	3.672s	3.710s	10.820	0.902
13	3.381s	3.404s	11.793	0.907
14	3.200s	3.049s	13.166	0.940
15	2.951s	2.999s	13.385	0.892
16	2.970s	2.946s	13.626	0.852
17	2.925s	2.969s	13.521	0.795
18	2.941s	2.929s	13.705	0.761
19	2.904s	2.860s	14.036	0.739
20	2.832s	2.810s	14.286	0.714
21	2.789s	2.756s	14.566	0.694
22	2.735s	2.746s	14.619	0.664
23	2.708s	2.735s	14.678	0.638
24	2.649s	2.673s	15.018	0.626
25	2.612s	2.643s	15.188	0.608
26	2.579s	2.602s	15.428	0.593
27	2.452s	2.274s	17.653	0.654
28	2.422s	2.438s	16.466	0.588
29	2.360s	2.380s	16.867	0.582
30	2.327s	2.337s	17.177	0.573
31	2.302s	2.321s	17.296	0.558
32	2.256s	2.303s	17.431	0.545

Структура на кода

`src/main.rs` парсира command line аргументите и ги пакетира в структура, която подава на `mandelbrot::compute()`

`src/v2.rs` е просто имплементация на 2D вектор структура, и няколко operator overload-а, за да се ползва по-удобно

`src/mandelbrot.rs` е файла, в който се извършва реалната работа. `mandelbrot::compute()` е де факто main функцията - създава големия масив, в който ще пишем, разбива го на подмасиви, създава thread pool-а, изчаква работата и връща завършения масив.

`mandelbrot::worker()` е функцията на worker нишките, която имплементира алгоритъма, описан в секцията “Ход на програмата”.

`mandelbrot::mandelbrot()` връща правилната стойност за определен пиксел.