ParallelMandelbrot:

Изследване на скалируемостта за теста на Манделброт в мултипроцеоср с паралелизъм до 16 със статично балансиране

Автор: Ерик Пламенов Здравков, ФН: 62511, Софтуерно инженерство, курс 3

СЪДЪРЖАНИЕ

1.Анализ

1.1. Функционален анализ

[1] Ефективно генериране на теста на Манделброт използвайки Message Passing Interface

В този пример е използвана Master-Slave архитектурата. Има една мастър нишка, която задейства останалите нишки. Всяка от нишките върши своята работа в зависимост от поредния си номер и след това връща информацията на мастър нишката. Матрицата се разделя по най-простия възможен начин. Броят на редовете на матрицата се разделя на броя нишки. Всеки процес изчислява своите начална и крайна позиция и генерира теста на Манделброт за нея. Нека с **Sr** означим началната позиция на нишка r, с **iYmax -** броя на редовете на матрицата и с **N** – паралелизма.

Sr = iYmax / N × r

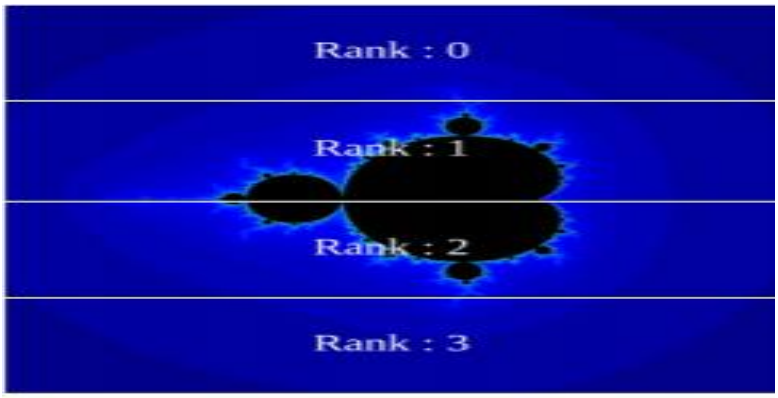
Нека с **Er** означим началната позиция на нишка **r.**

Er = ((iYmax / N) × (r × 1)) – 1

Чрез това разбиване на матрицата всички нишки ще изчисляват равен брой редове.

редове за всяка нишка = iYmax/N

Ако броят на редовете не може да се раздели на броя на нишките, то последната нишка ще изчисли остатъка. Това няма да повлияе върху продуктивността.



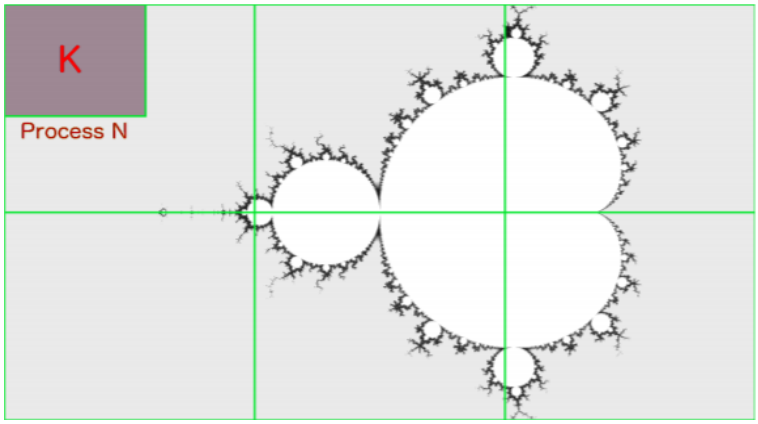
Фигура 1. Пример за N=4

Използваният подход не предизвиква добро ускорение, тъй като няк.оя нишка може да получи по-трудни задачи от другите и да забави процеса.

[2] Паралелно генериране на Манделброт

1) **Динамично балансиране**

В примера главния процес има само управленческа задача да възлага на процесите техните зависещи от K задачи. Поради динамичното балансиране главният процес анализира идващите съобщения от процесите и възлага следваща задача на вече свободен процес. След завършването на всички процеси, главният процес събира цялата информация.



Фигура 1. Динамично балансиране на работата в 2x3 grid.

Извършените опити показват, че за голям паралелизъм се постига почти линейно ускорение

[3] Паралелизиране на Манделброт

1) **Динамично балансиране**

В този пример също имаме главен процес, който разпределя работата на останалите процеси. В случая главният процес изпраща към всеки процес по един ред, след което изчаква за готови процеси и им изпраща още работа. Забелязва се, че при паралелизъм 32 ускорението е незначително. Това се обяснява от увеличеното време за комуникация между процесите.

1.2. Сравнителна таблица

Пример	Балансиране	Декомпозиране	Паралелизъм	Итерации	g	Ускорение	Размер
[1]	Статично	По редове	16	2000	1	3.97305	8000x8000
[2]	Динамично	Централ.	16	10000	-	14	12800×12800
[3]	Динамично	Централ.	16	256	-	11.559	4000*4000
PM	Статично	Редове и колони	16	256	16	8.566	3840x2160

1.3. Технологичен анализ

[1] Efficient Generation of Mandelbrot Set using Message Passing Interface

Intel® Xeon® Gold 6150 Processor

# of Cores – 18

# of Threads – 36

L1$ 1.125 MiB	**L1I$** 576 KiB 18x32 KiB 8-way set associative **L1D$** 576 KiB 18x32 KiB 8-way set associative
L2$ 18 MiB	18x1 MiB 16-way set associative
L3$ 24.75 MiB	18x1.375 MiB 11-way set associative

[2] Parallel generation of a Mandelbrot set

Клъстер от 13 четириядрени 64 битови Intel Процесори с 2 GB RAM и x84 64 architecture. Използвана е библиотеката MPI.

[3] Parallelizing Mandelbrot

University of Nevada, Reno's cluster. Използвана е библиотеката MPI.

2. Проектиране

2.1. Функционално проектиране

Използван е модела SPMD(single program,multiple data) и архитектурата Master-Slave. Единствената разлика е, че в нашата програма нямаме Master код. Master нишката пуска всички останали нишки с подходящи параметри, след което нишките започват да извършват работата си. След края на работата на последната нишка Master нишката конструира изображението въз основа на получените резултати от другите нишки. В случая самата Master нишка не извършва изчисления. Това е възможно да се оптимизира.

Използва се статично балансиране с декомпозиция на данните по два начина. Единият е по колони, другият е по редове. Целта е да се сравнят двата подхода. Разделяме матрицата на N на брой задания. Числото N зависи от грануларноста g и винаги N=p*g. Така всяка нишка ще обработи g задания. От първите p задания всяка нишка ще получи едно задание, от вторите p задания всяка нишка ще получи едно задание и така до края.

Код за извикване на нишките със съответните параметри:

```
for (int i=0; i<thread_count; i++) {
   MandelbrotRunnable r = new MandelbrotRunnable(m,g,h,w,sX,eX,sY,Y,
i,thread_count);
   tr[i] = new Thread(r);
   tr[i].start();
}
```

Всяка нишка взема като параметри матрицата на Манделброт, грануларността, височината на изображнието, дължината на изображението, началото на интервала на Манделброт за абцисата, края на интервала на Манделброт за абцисата, началото на интервала на Манделброт за ординатата, края на интервала на Манделброт за ординатата, номера на нишката и броя на нишките.

Код за изчакване на останалите нишки от главната нишка:

```
  for(int i = 0; i < thread_count; i++) {
   try {
      tr[i].join();
   } catch (Exception e) {
      System.out.println(e.getMessage());
   }
}
```

2.2. Технологично проектиране

2.2.1 Тестова среда

Тестовата среда е следната:

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

CPU(s): 32

On-line CPU(s) list: 0-31

Thread(s) per core: 2

Core(s) per socket: 8

Socket(s): 2

NUMA node(s): 2

Vendor ID: GenuineIntel

CPU family: 6

Model: 45

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

Stepping: 7

CPU MHz: 2109.497

CPU max MHz: 3000.0000

CPU min MHz: 1200.0000

BogoMIPS: 4389.52

Virtualization: VT-x

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

L3 cache: 20480K

NUMA node0 CPU(s): 0-7,16-23

NUMA node1 CPU(s): 8-15,24-31

2.2.2 Наръчник за птребителите

Извикването на програмата става по следния начин:

./runMe.sh <Threads> <G> <Width> <Height> <StartX> <EndX> <StartY> <EndY>

2.2.3 Използвани технологии

Проектът е разработен на езика Java, като е използвана библиотеката, която съдържа класа Complex, който се използва за комплексните числа. За създаване на нишките и за извършването на паралелната работа се използва класа Thread.

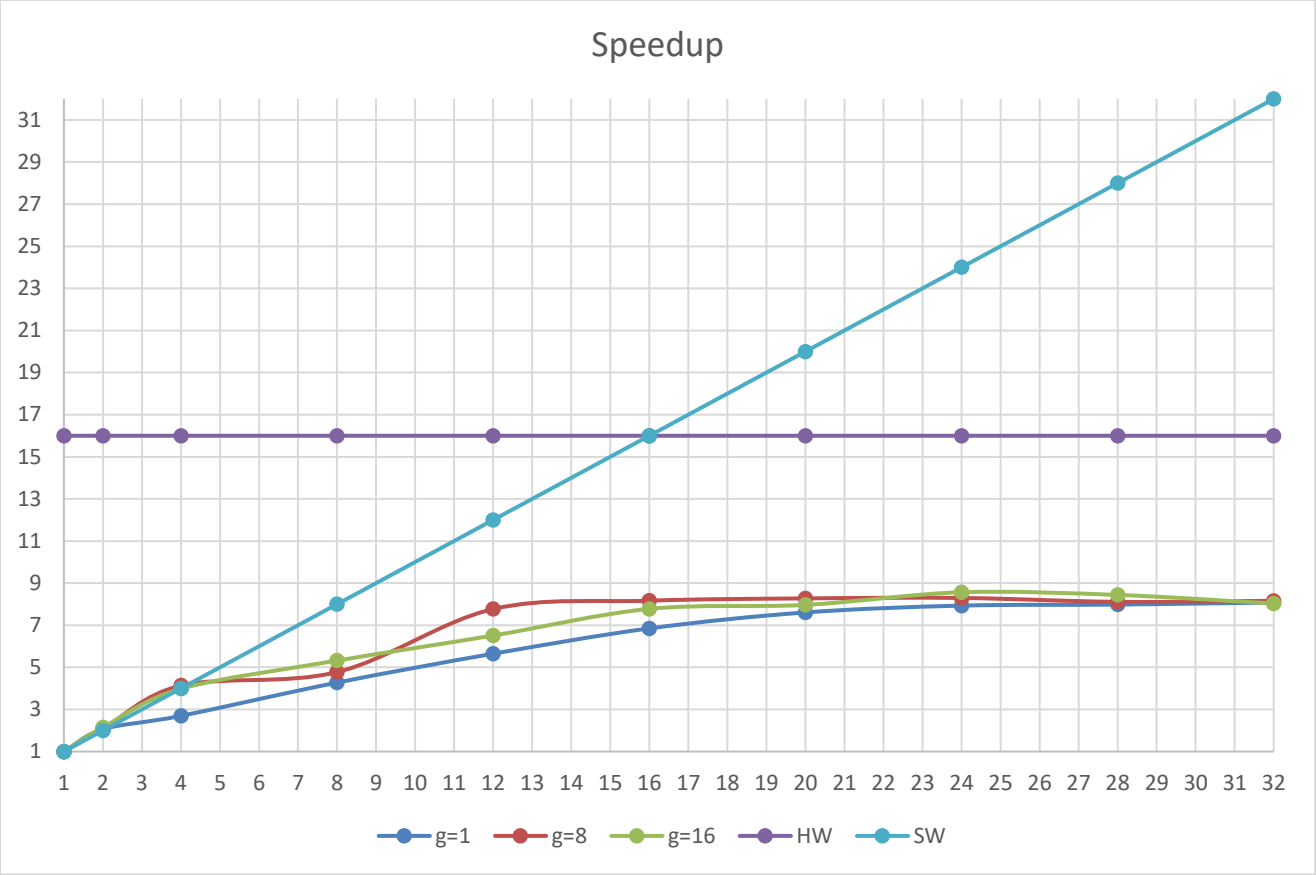
3.Тестване

Във всеки тестов план размера на екрана е 4K (3840x2160) и е изследвана областта на Манделброт -1.85 <= Re <= 0.5 и -1 <= Im <= 1 с дълбочина 256 цвята.

3.1. Статично балансиране по редове

#	p	g	T_p (1)	T_p (2)	T_p (3)	T_p=min($T_p^{(i)}$)	$S_p = T_1/T_p$	$E_p = S_p/p$
1	1	1	***19763***	***19352***	***19428***	**19352**	1	1
2	2	1	*9649*	*9400*	*10122*	**9400**	2.0587234	1.029361702
3	4	1	*7536*	*7183*	*7239*	**7183**	2.6941389	0.673534735
4	8	1	*6487*	*4589*	*4526*	**4526**	4.2757402	0.534467521
5	12	1	*3428*	*3451*	*3428*	**3428**	5.6452742	0.470439518
6	16	1	*2892*	*2826*	*3147*	**2826**	6.8478415	0.427990092
7	20	1	*2589*	*2544*	*2555*	**2544**	7.6069182	0.380345912
8	24	1	*2505*	*2518*	*2440*	**2440**	7.9311475	0.330464481
9	28	1	*2425*	*2430*	*2433*	**2425**	7.9802062	0.285007364
10	32	1	*2395*	*2414)*	*2400*	**2395**	8.080167	0.252505219
11	1	8	***20322***	***20424***	***20715***	**20322**	1	1
12	2	8	*9810*	*10060*	*12893*	**9810**	2.0715596	1.035779817
13	4	8	*4915*	*5027*	*6433*	**4915**	4.1346897	1.033672431
14	8	8	*4258*	*4745*	*4386*	**4258**	4.7726632	0.596582903
15	12	8	*2798*	*2616*	*2951*	**2616**	7.7683486	0.647362385
16	16	8	*2489*	*2560*	*2578*	**2489**	8.1647248	0.510295299
17	20	8	*2473*	*2456*	*2496*	**2456a**	8.27443	0.413721498
18	24	8	*2450*	*2495*	*2469*	**2450**	8.2946939	0.345612245
19	28	8	*2505*	*2520*	*2506*	**2506**	8.1093376	0.2896192
20	32	8	*2491*	*2530*	*2580*	**2491**	8.1581694	0.254942794
21	1	16	***25276***	***19898***	***20120***	**19898**	1	1
22	2	16	*9272*	*9288*	*9648*	**9272**	2.1460311	1.073015531
23	4	16	*4998*	*5336*	*5044*	**4998**	3.9811925	0.995298119
24	8	16	*3741*	*4509*	*4296*	**3741**	5.3188987	0.664862336
25	12	16	*3056*	*3310*	*3110*	**3056**	6.5111257	0.542593805
26	16	16	*2573*	*2582*	*2559*	**2559**	7.7756936	0.485980852
27	20	16	*2498*	*2509*	*2545*	**2498**	7.9655725	0.398278623
28	24	16	*2347*	*2360*	*2323*	**2323**	8.5656479	0.356901995
29	28	16	*2358*	*2396*	*2376)*	**2358**	8.4385072	0.301375257
30	32	16	*2477*	*2499*	*2502*	**2477**	8.0331046	0.251034518

..

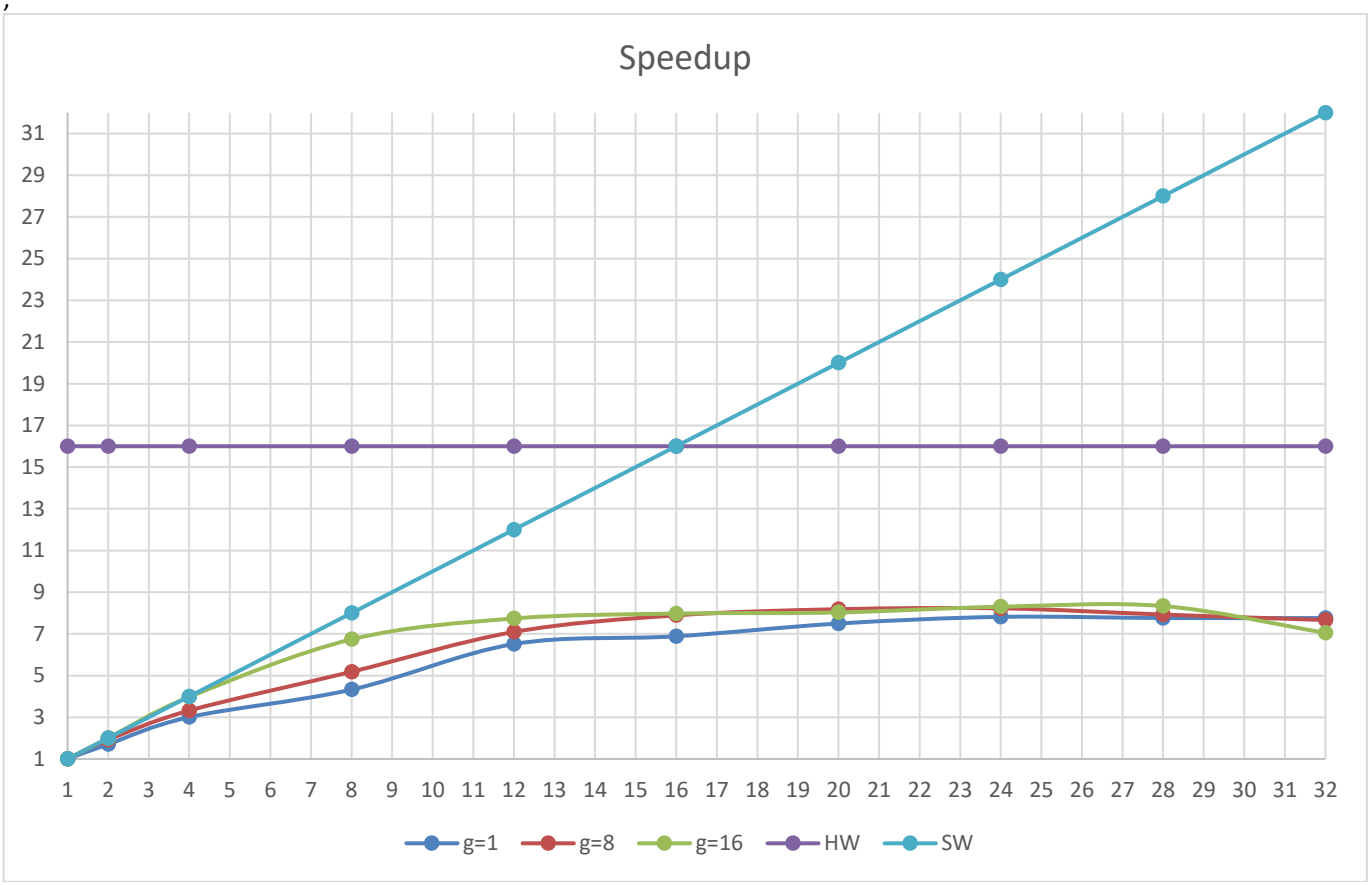


3.2. Статично балансиране по колони

#	p	g	T_p (1)	T_p (2)	T_p (3)	T_p=min($T_p^{(i)}$)	$S_p = T_1/T_p$	$E_p = S_p/p$
1	1	1	***19070***	***19252***	***19121***	**19070**	1	1
2	2	1	*11184*	*11200*	*11522*	**11184**	1.7051144	0.852557225
3	4	1	*6404*	*6406*	*6340*	**6340**	3.0078864	0.751971609
4	8	1	*4514*	*4569*	*4406*	**4406**	4.3281888	0.541023604
5	12	1	*3776*	*3085*	*2929*	**2929**	6.5107545	0.542562877
6	16	1	*2793*	*2798*	*2770*	**2770**	6.8844765	0.430279783
7	20	1	*2680*	*2544*	*2555*	**2544**	7.4960692	0.374803459
8	24	1	*2505*	*2518*	*2440*	**2440**	7.8155738	0.325648907
9	28	1	*2528*	*2457*	*2559*	**2457**	7.7614978	0.277196349
10	32	1	*2516*	*2556*	*2458*	**2458**	7.7583401	0.242448129
11	1	8	***20060***	***19014***	***21494***	**19014**	1	1
12	2	8	*10887*	*10484*	*9921*	**9921**	1.9165407	0.958270336
13	4	8	*5724*	*5955*	*5972*	**5724**	3.3218029	0.830450734
14	8	8	*3840*	*3669*	*4001*	**3669**	5.1823385	0.647792314
15	12	8	*2848*	*2718*	*2681*	**2681**	7.0921298	0.591010817

16	16	8	*2410*	*2482*	*2533*	**2410**	7.8896266	0.49310166
17	20	8	*2373*	*2356*	*2324*	**2324**	8.1815835	0.409079174
18	24	8	*2312*	*2372*	*2405*	**2312**	8.2240484	0.342668685
19	28	8	*2399*	*2446*	*2510*	**2399**	7.9258024	0.283064372
20	32	8	*2510*	*2481*	*2499*	**2481**	7.6638452	0.239495163
21	1	16	***20147***	***20107***	***20120***	**20107**	1	1
22	2	16	*10041*	*10125*	*10101*	**10041**	2.0024898	1.001244896
23	4	16	*5177*	*5189*	*5061*	**5061**	3.9729303	0.993232563
24	8	16	*3012*	*2984*	*3012*	**2984**	6.7382708	0.842283847
25	12	16	*2701*	*2667*	*2598*	**2598**	7.7394149	0.644951245
26	16	16	*2522*	*2528*	*2533*	**2522**	7.9726408	0.498290048
27	20	16	*2505*	*2532*	*2595*	**2505**	8.0267465	0.401337325
28	24	16	*2483*	*2422*	*2466*	**2422**	8.3018167	0.345909028
29	28	16	*2567*	*2501*	*2413*	**2413**	8.3327808	0.297599313
30	32	16	*2930*	*2851*	*3012*	**2851**	7.0526131	0.22039416

,



3.3. Сравнение между двата подхода

Двете графики изглеждат идентични. Това показва, че няма голяма разлика дали ще разделим по редове или по колони. Забелязва се, че при разделянето по колони получаваме малко по-добро ускорение, но съществена разлика няма. И в двата случая се вижда, че след надхвърлянето на p=24 следва лек спад в ускорението. Това се случва, тъй като тестовата машина е с 16 ядра и пускането на повече нишки предизвиква свръхтовар. При разделянето по редове максималното ускорение Sp=8.57 е при p=24 и g=16. При разделянето по колони максималното ускорение е Sp=8.30 е при p=28 и g=16. Забелязва се, че най-добро ускорение и в двата случая се постига при g=16, което е средна грануларност. Най-лошо ускорение се постига при g=1, което е най-едрата гранлуарност. Виждаме, че за голямо ниво на паралелизъм не можем да постигнем линейно ускорение.

4.Източници

[1] Efficient Generation of Mandelbrot Set using Message Passing Interface

Bhanuka Manesha Samarasekara Vitharana Gamage, Vishnu Monn Baskaran

School of Information Technology, Monash University Malaysia

(https://arxiv.org/pdf/2007.00745.pdf?fbclid=IwAR3oqiBHzvQPsiyxzxPWOeDt_OBAsmXww0X5GtiSZgllQLQuwcY8B8JT4X4)

[2] Parallel generation of a Mandelbrot set

Antonio Lagana and Leonardo Pacifici

Department of Chemistry, Biology and Biotechnology, University of Perugia

(https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjNkOHr6cjxAhXxwAIHHSXtAP8QFnoECA8QAw&url=http%3A%2F%2Fservices.chm.unipg.it%2Fojs%2Findex.php%2Fvirtlcomm%2Farticle%2Fview%2F112%2F108&usg=AOvVaw0WfT8Wu5GwmLwH3XOTBLB5f)

[3] Parallelizing Mandelbrot

Raymond M. Pistoresi

Department of Computer Science and Engineering, University of Nevada, Rino

(https://github.com/rpistoresi/parallelizing-mandelbrot/blob/master/Report.pdf)