Софийски университет „Св. Климент Охридски“,
Факултет по математика и информатика

Курсов проект

по Системи за паралелна обработка

Тема: „Изобразяване на фрактал – Julia set“

Изготвил:
Ивайло Росенов Михайлов, 81158, Компютърни науки, курс 3, поток 2, група5

Научен ръководител:
ас. Христо Христов

Дата: 06.06.2017г. Проверка: …………………….

1. Цел на проекта

Целта на проекта е реализация на паралелен алгоритъм генериращ множеството на Жулия (Julia set) от комплексни числа посредством формулата F(z)=z^2+c, където z и c са комплексни числа и c е предварително зададена константа. Изискванията към програмата са следните:

- Програмата трябва да използва паралелни процеси (нишки) за да разпредели работата по търсенето на точките от множеството на Жулия на повече от един процесор;
- Програмата трябва да осигурява и генерирането на изображение (например.png),показващо така намереното множество;
- Програмата да позволява (разбира от) команден параметър, който задава големината на генерираното изображение, като широчина и височина в брой пиксели. При не въведен от потребителя команден параметър, за големина на изображението, програмата задава параметри по подразбиране;
- Команден параметър за възможност за приближение и изместване на образа, или разглеждане на конкретна част от множеството;
- Команден параметър, който да задава максималният брой нишки, на които да се разпределя работата по генерирането на множеството. При не въведен от потребителя максимален брой нишки да се въвежда една нишка по подразбиране;
- Команден ред указващ името на генерираното изображение (-o picture_name.png). При не въвеждане на име на файл от потребителя програмата да задава име по подразбиране;
- Програмата да извежда подходящи съобщения на различните етапи от работата си. Както и времето отделено за изчисляването на всички точки от множеството и времето за изпълнението на цялата програма;

2. Описание на алгоритъма

Програмата реализираща алгоритъма за генериране на множеството на Жулия дава възможност на потребителя да генерира изображение с желани размери и цвят изобразяващо множеството на Жулия във вид на фрактал. Алгоритъмът е на Cuda – платформа за паралелно паралелно програмиране на nvidia – под архитектура Maxwell.

Алгоритъмът, който използва програмата е често срещан при генериране на фрактали с помощта на Cuda платформата. Първоначално на програмата се задава брой нишки в един блок (cuda thread block) и след това изчислява броя необходими блокове, за да се покрие цялото изображение,

така че всеки пиксел да се изчислява от една нишка. Алгоритъмът изчислява един пиксел/точка от изображението/множеството в зависимост от номера на блока, в който се намира, и собствения си номер. На базата на позицията си изчислява комплексно число p, зависещо също и от зададеното приближение и отместване на изображението. Числото p бива итерирано с формулата F(p)=p^2+c определен брой пъти, докато или абсолютната му стойност надвиши 4(излезе от множеството) или докато не минат определен брой итерации. Броя на итерациите се записва в масив, на базата, на който след това се изчисляват цветовете на изображението. Извършва се един вид mapping на множеството чрез цели числа, за да може по-лесно да се изчислят цветовете на изображението.

3. Реализация

Реализацията на алгоритъма и пралелизирането на изчисляването е на езика Cuda/C++. В алгоритъмът се използват само първите две измерения на блоковете и мрежата от блокове, т.к. изображението е в 2D.

Ще разгледаме главните функции за реализирането на алгоритъма:

Първоначално след като бъде зададена предварително бройката на нишките в блок, изчисляваме броя блокове в мрежата от блокове необходими за покриването на цялото изображение, така, че всяка нишка в блок да изчислява един пиксел/точка от множеството. Това става чрез функцията CalcGrid, на която като първи параметър се задава ширина или височина на изображението, а като втори съответно брой нишки по ширина или височина на блока от нишки, и така накрая като резултат се връща едното от измеренията на мрежата от блокове с нишки – брой блокове по ширина или по височина.

```
180    //изчисляваме бройката на блоковете
181    int CalcGrid(int x, int y) { return x / y + (x % y ? 1 : 0); }
182
```

След като вече знаем размерностите на мрежата от блокове и на самите блокове, извикваме функцията CalcJulia да се изпълни паралелно за всяка нишка в съответния брой блокове и нишки в тях, като подаваме параметри масива от данни, който трябва да бъде попълнен след изпълнението на всички нишки, комплексната константа c, колко да е приближението (1,2,3,4.5 и т.н.), размерностите на изображението и отместването в пиксели по x и y.

```
363     C = make_cuDoubleComplex(R,I);
364
365     cout<<"Calculating grid dimension x..."<<endl;
366     gridx=CalcGrid(dimx,blockx);
367     cout<<"Calculating grid dimension y..."<<endl;
368     gridy=CalcGrid(dimy,blocky);
369
370     dim3 bs(blockx,blocky);
371     dim3 gs(gridx,gridy);
372
373     cout<<"Calculating Julia set..."<<endl;
374
375     //start timer2
376     gettimeofday(&c1,NULL);
377
378     CalcJulia<<<gs,bs>>>(ddata,C,zoom,dimx,dimy,movex,movey);
379     check_return(cudaMemcpy(hdata, ddata, dataSize, cudaMemcpyDeviceToHost));
380
```

CalcJulia с помощта на ToComplex изчислява комплексното число p на базата на: позиция на нишката в мрежата от блокове с нишки; приближение; отместване по височина и ширина и размерите на изображението.

```
71  __device__ cuDoubleComplex ToComplex(int x, int y,double z,int dx,int dy,double mx,double my)
72  {
73      double jx = (x - dx / 2) / (0.5 * z * dx) + my;
74      double jy = (y - dy / 2) / (0.5 * z * dy) + mx;
75
76      return make_cuDoubleComplex(jx,jy);
77  }
```

```
79  __global__ void CalcJulia(int* data,cuDoubleComplex c,double z,int dx,int dy,double mx,double my)
80  {   //стандарно изчисляване на номер на елемент в масива
81      int i =  blockIdx.x * blockDim.x + threadIdx.x;
82      int j =  blockIdx.y * blockDim.y + threadIdx.y;
83
84      if(i<dx && j<dy)
85      {
86          cuDoubleComplex p = ToComplex(i,j,z,dx,dy,mx,my);
87          data[i*dy+j] = FindLimit(p,c);
88      }
89  }
```

Изчисляването дали точката излиза от множеството и клони към безкрайност при итериране с формулата F(p)=p^2+c изчисляваме чрез функциите FindLimit и JuliaFunc.

```
53  //Julia функцията
54  __device__ cuDoubleComplex JuliaFunc(cuDoubleComplex p,cuDoubleComplex c)
55  {
56      return cuCadd(cuCmul(p,p),c); //F(x)=x^2+c
57  }
```

```
59  //проверка дали точката излита от множеството
60  __device__ int FindLimit(cuDoubleComplex p,cuDoubleComplex c)
61  {
62      int it =1;
63      while(it <= MAXITR && cuCabs(p) <= 4)
64      {
65          p=JuliaFunc(p,c);
66          it++;
67      }
68      return it;
69  }
```

Накрая попълнения масив ddata се копира обратно в хост частта на програмата и с помощта на функциите dwell_color и save_image се изчисляват цветовете на пикселите и се генерира png изображението.

4. Стартиране на програмата

Програмата може да се стартира без никакви аргументи от командния ред, както и с команда –h, която принтира следното помощно меню, където можем да видим какви аргументи може да подадем на програмата и в какви граници.

```
OPTIONS:
        -sx <num pixels>: image width in pixels
        -sy <num pixels>: image height in pixels
        -bx <num blocks>: block dimension x (min 1, max 1024)
        -by <num blocks>: block dimension y (min 1, max 1024)
                Maximum of 1024 threads per block allowed
        -mx <num pixels>: offset in pixels horizontally (statrs at up left corner 0)
        -my <num pixels>: offset in pixels vertically (statrs at up left corner 0)
        -z <scale>: zoom
        -rp <double>: real part of the c constant
        -ip <double>: imaginary part of the c constant
        -col <r/g/b>: color of the image r/g/b for read/green/blue
        -o <fileName.png>: file name in png extension (maximum 32 symbols)
EXAMPLE CONSTANTS:
        c = 1                   # dentrite fractal
        c = -0.123 + 0.745i # douady's rabbit fractal
        c = -0.750 + 0i     # san marco fractal
        c = -0.391 - 0.587i # siegel disk fractal
        c = -0.7 - 0.3i     # NEAT cauliflower thingy
        c = -0.75 - 0.2i    # galaxies
        c = -0.75 + 0.15i   # groovy
        c = -0.7 + 0.35i    # frost
        c = -0.835-0.2321i  # dragon
```

5. Резултати

Ще покажем резултати от изпълнението на програмата със зададени стойности:

- Изображение – 4096x4096;
- Отместване в пиксели по хоризонтал - 0;
- Отместване в пиксели по вертикала - 0;
- Приближение – 0.6, за да може фрактала да се побере в границите на изображението;
- Константа c= - 0.835 - 0.2321i;
- Цвят – зелен;
- Изходен файл – julia.png

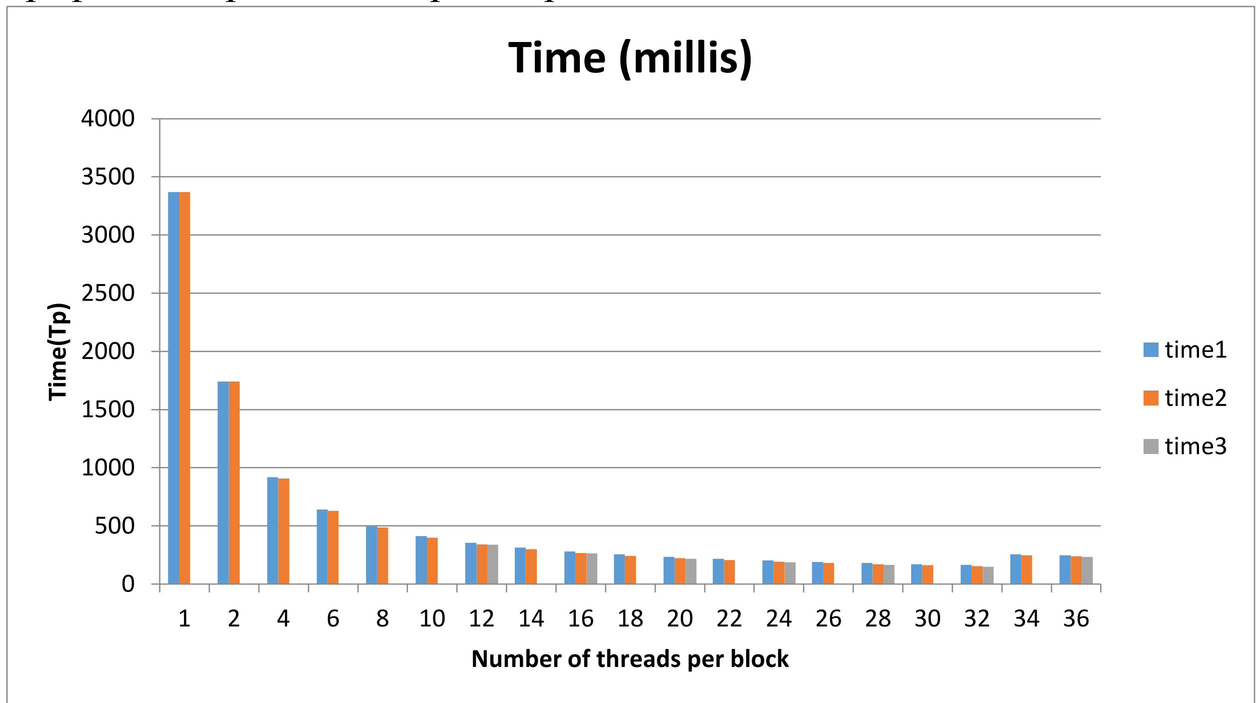
В таблицата долу са резултатите от изпълненията на програмата с различен брой ядра. Времената time1, time2 и time3 са измерени съответно при размерност на блоковете x на y:

	time1		time2		time3	
threads	x	y	x	y	x	y
1	1	1	1	1		
2	2	1	1	2		
4	4	1	2	2		
6	6	1	3	2		
8	8	1	4	2		
10	10	1	5	2		
12	12	1	6	2	3	4
14	14	1	7	2		
16	16	1	8	2	4	4
18	18	1	9	2		
20	20	1	10	2	5	4
22	22	1	11	2		
24	24	1	12	2	6	4
26	26	1	13	2		
28	28	1	14	2	7	4
30	30	1	15	2		
32	32	1	16	2	8	4
34	34	1	17	2		
36	36	1	18	2	9	4

Времената са измервани в милисекунди, като най-добро време за текущите параметри е измерено при размерност на блока 8x8 – 146,649 милисекунди.

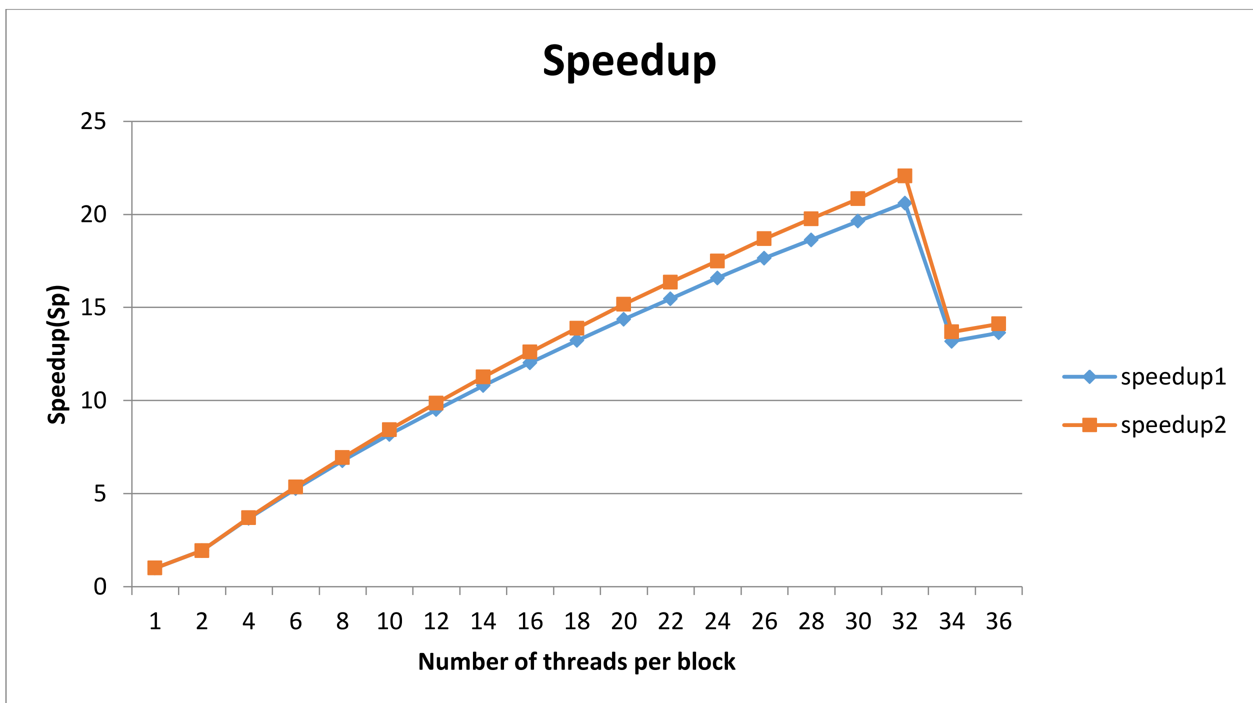
threads	time1	speedup 1	efficiency 1	time2	speedup 2	efficiency 2	time3
1	3370,44	1	1	3370,44	1	1	
2	1740,55	1,936422	0,968211	1740,55	1,936422	0,968211	
4	918,566	3,669241	0,91731	908,546	3,709708	0,927427	
6	640,234	5,264388	0,877398	628,758	5,360473	0,893412	
8	498,247	6,764597	0,845575	486,005	6,93499	0,866874	
10	412,195	8,17681	0,817681	399,559	8,4354	0,84354	
12	354,581	9,505416	0,792118	341,646	9,865299	0,822108	338,213
14	312,044	10,80117	0,771512	299,332	11,25987	0,804277	
16	280,159	12,03045	0,751903	267,413	12,60387	0,787742	262,712
18	254,906	13,22229	0,734571	242,716	13,88635	0,771464	
20	234,587	14,36755	0,718377	222,138	15,17273	0,758637	216,902
22	217,946	15,46456	0,702935	206,035	16,35858	0,743572	
24	203,219	16,58526	0,691053	192,658	17,49442	0,728934	186,123
26	190,94	17,65183	0,678916	180,307	18,69279	0,718953	
28	180,9	18,63151	0,665411	170,563	19,76067	0,705738	163,848
30	171,661	19,63428	0,654476	161,732	20,83966	0,694655	
32	163,57	20,60549	0,643922	152,742	22,06623	0,68957	147,649
34	255,76	13,17814	0,387592	246,268	13,68607	0,402531	
36	247,19	13,63502	0,37875	238,706	14,11963	0,392212	233,118

- Графика на времената спрямо броя нишки:



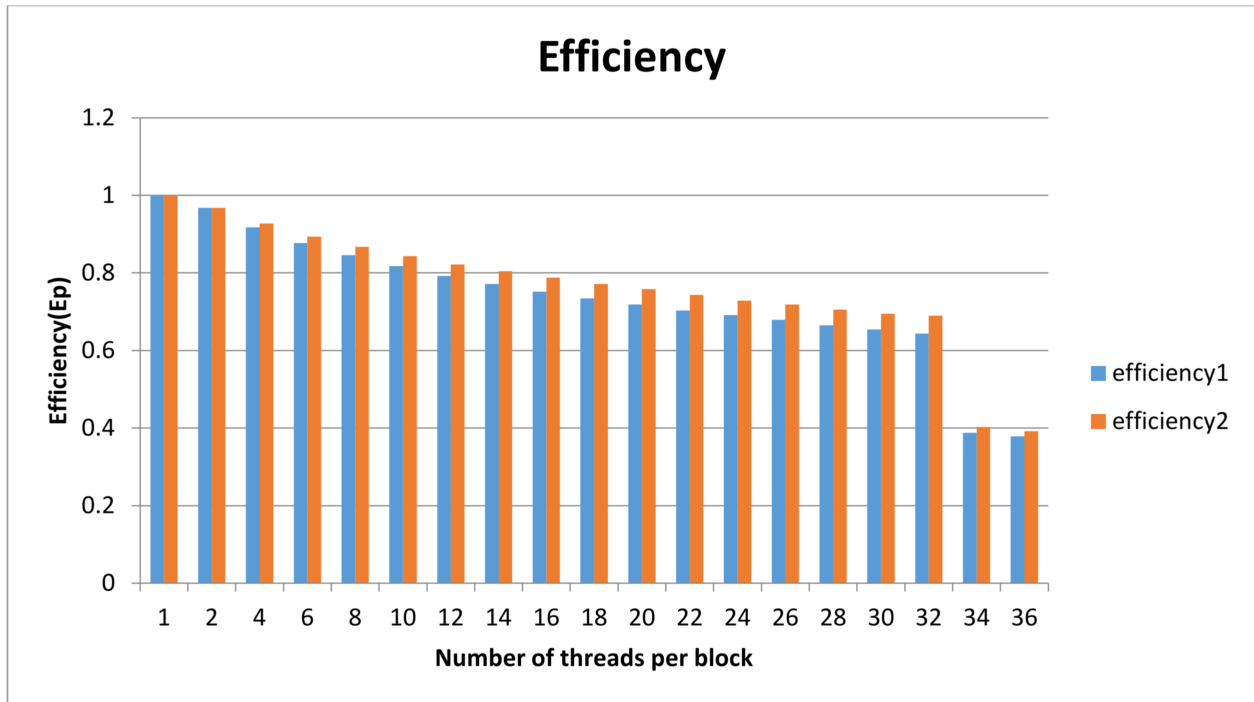
T1 е времето за изпълнение на програмата от 1 нишка на блок. Tp – времето за изпълнение на програмата от p нишки на блок.

- Графика на ускорението спрямо броя нишки:



Ускорението при използване на повече от една нишка се изчислява по формулата: Sp = T1 / Tp.

- Графика на ефективността спрямо броя нишки:



Ефективността се изчислява по формулата: Ep = Sp / p.

Тестовете са проведени на конфигурация:
Два процесора - intel Xeon E5-2660 с по 10 физически ядра и 20 нишки;
Оперативна памет - DDR3 64GB;
Графична карта – Nvidia Quadro M4000, Maxwell архитектура, 256-bit, 192GB/s скорост на пренос на данни, 1662 графични процесора, 8GB DDR5 памет;

Използвани източници при изготвяне на проекта:

https://gist.github.com/knotman90/9b4e47dc04e7dbfefe4a

https://github.com/chaitanyav/CUDA

http://www.davidespataro.it/cuda-julia-set-fractals/

http://www.lnmiit.ac.in/Mathematics/GPULecture7.pdf

https://github.com/Teknoman117/cuda/blob/master/julia_example/julia.cu

http://chaitanyav.github.io/2014/07/14/julia-set-cuda/

https://en.wikipedia.org/wiki/Julia_set