
Programação Paralela - ELC139

Mandelbrot Set

— Rhauani Fazul —

Sumário

- Mandelbrot Set;
- Análise do problema:
- Solução proposta;
- Análise de desempenho:
 - Speedup;
 - Eficiência;
- Referências.

Mandelbrot Set

- É obtido a partir do conjunto de valores de c para quais a sucessão de pontos gerados pela equação $Z_{n+1} = Z_n^2 + c$ não tende para o infinito do plano complexo;
- Gera uma imagem *fractal** infinitamente complexa em que, ao aproximá-la (em qualquer intensidade) será possível observar estruturas tão complexas quanto a imagem como um todo.

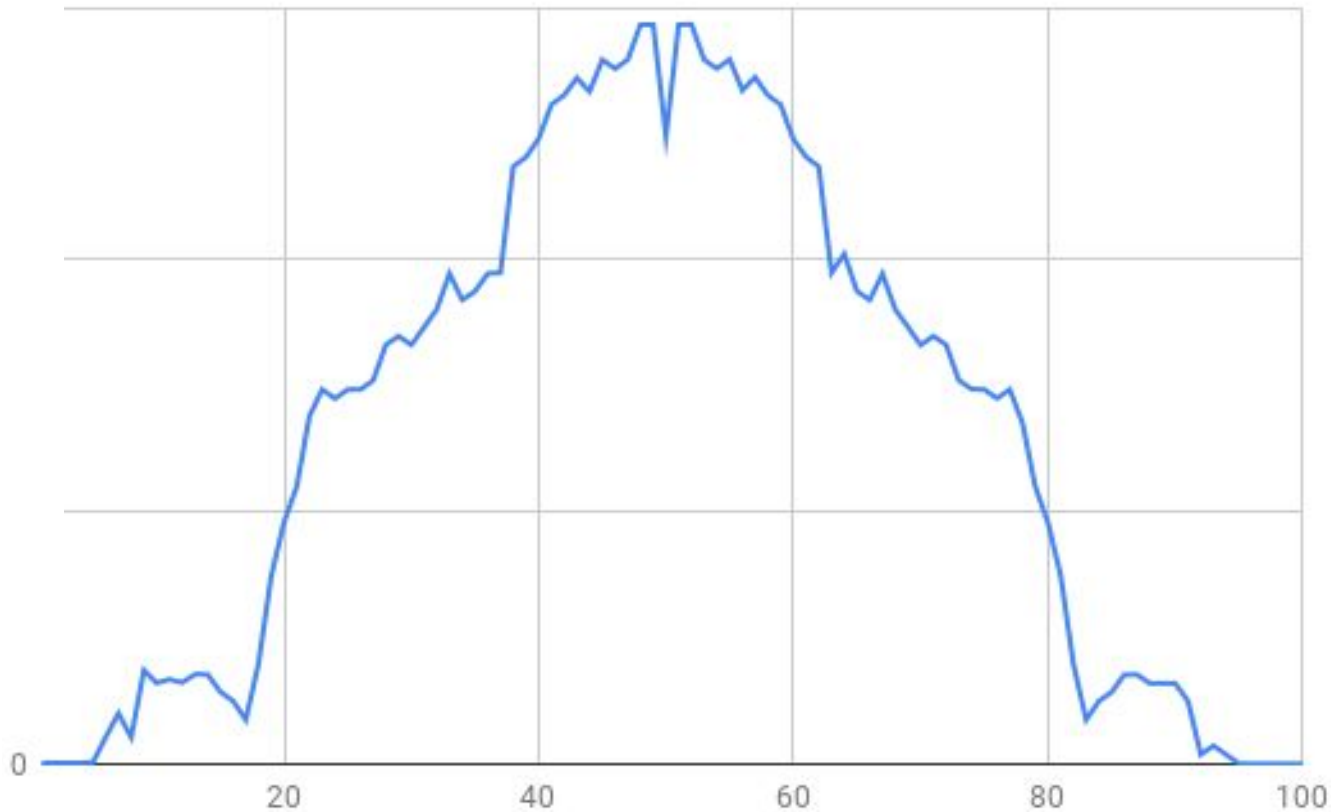
* Objeto que contém dentro de si próprio cópias menores dele mesmo e essas cópias, por sua vez, contém cópias ainda menores (e assim sucessivamente).

Análise do problema

- Estudo da lógica e estrutura do código base;
- Escolha de uma ferramenta de programação paralela:
 - **OpenMP**;
 - Como chegar ao melhor *speedup*?
- Escolha do ponto de paralelização:
 - **Laço mais externo** (linhas da matriz) tende a apresentar melhores resultados quando paralelizado.
- Divisão de trabalho:
 - As iterações do laço possuem uma carga de trabalho homogênea? **Não!**

Análise do problema

- Ao medir o tempo gasto em cada iteração do laço, percebeu-se uma tendência do tipo:



- Iterações centrais tendem a serem mais trabalhosas que as iterações iniciais e finais.

Análise do problema

- Com isso, é esperado que a divisão de trabalho convencional do OpenMP* não seja a melhor escolha para este problema;
- Uma distribuição “esparsa” das iterações entre as *threads* pode vir a ser mais satisfatória;
- Realizou-se o *profiling* do programa, comparando:
 - **#pragma omp parallel for schedule(static)**
 - **#pragma omp parallel for schedule(static, 1)**
 - equivalendo a:

```
#pragma omp parallel  
for (int i = tid; i < N; i += tid)
```

* estática e com *chunk size* = (total iterações / *nthreads*)

Análise do problema

- **#pragma omp parallel for schedule(static)**

Profile Part	Called
PID 10442, section 1, thread 1	5 856 184
PID 10442, section 1, thread 2	30 980 024
PID 10442, section 1, thread 3	31 898 928
PID 10442, section 1, thread 4	6 658 862

- **#pragma omp parallel for schedule(static, 1)**

Profile Part	Called
PID 10374, section 1, thread 1	18 716 652
PID 10374, section 1, thread 2	18 830 204
PID 10374, section 1, thread 3	19 016 552
PID 10374, section 1, thread 4	18 830 590

Mais “justo”!

Solução proposta

```
1  #pragma omp parallel for schedule(static, 1)
2  for (int r = 0; r < max_row; r++) {
3      for (int c = 0; c < max_column; c++) {
4          complex<float> z;
5          int n = 0;
6          while ( (abs(z) < 2) && (++n < max_n) )
7              z = pow(z, 2) + decltype(z) (
8                  (float) c * 2 / max_column - 1.5,
9                  (float) r * 2 / max_row - 1
10                 );
11          mat[r][c] = (n == max_n) ? '#' : '.';
12      }
13 }
```


Solução proposta

- Código disponível em: <https://goo.gl/itFjhf>

- Compilação:

- Makefile:

```
$ make
```

- Diretamente:

```
$ g++ -O3 -std=c++11 -fopenmp -o mandelbrot mandelbrot.cpp -Wall
```

- Execução:

```
$ ./mandelbrot <linhas> <colunas> <iterações>
```

Análise de desempenho

- Medição de tempo
 - Header: **<chrono>**
 - Clock: **high_resolution_clock**

```
typedef std::chrono::high_resolution_clock Clock;
```

```
auto t_start = Clock::now();
```

```
/* calcula mandelbrot set */
```

```
auto t_end = Clock::now();
```

```
duration_cast<nanoseconds>(t_end - t_start).count();
```

```
/* <microseconds>
```

```
<milliseconds>
```

```
.... */
```

Análise de desempenho

- Bateria de testes (30 execuções para cada configuração):

#	LINHAS	COLUNAS	ITERAÇÕES
A1	100	100	100
A2	100	100	1000
A3	100	100	10000
A4	100	100	100000

#	LINHAS	COLUNAS	ITERAÇÕES
B1	200	200	200
B2	200	200	2000
B3	200	200	20000
B4	200	200	200000

Análise de desempenho

- Médias dos tempos obtidos com a versão sequencial:

#	TEMPO (seg)
A1 { 100, 100, 100 }	0.0337
A2 { 100, 100, 1000 }	0.2748
A3 { 100, 100, 10000 }	2.6755
A4 { 100, 100, 100000 }	26.7032

#	TEMPO (seg)
B1 { 200, 200, 200 }	0.2359
B2 { 200, 200, 2000 }	2.169
B3 { 200, 200, 20000 }	21.439
B4 { 200, 200, 200000 }	213.3276

Análise de desempenho

- Médias dos tempos obtidos com a versão paralelizada com OpenMP (4 *threads*):

#	TEMPO (seg)
A1 { 100, 100, 100 }	0.0099
A2 { 100, 100, 1000 }	0.086
A3 { 100, 100, 10000 }	0.8146
A4 { 100, 100, 100000 }	9.3019

#	TEMPO (seg)
B1 { 200, 200, 200 }	0.0711
B2 { 200, 200, 2000 }	0.6608
B3 { 200, 200, 20000 }	7.109
B4 { 200, 200, 200000 }	82.0428

Análise de desempenho



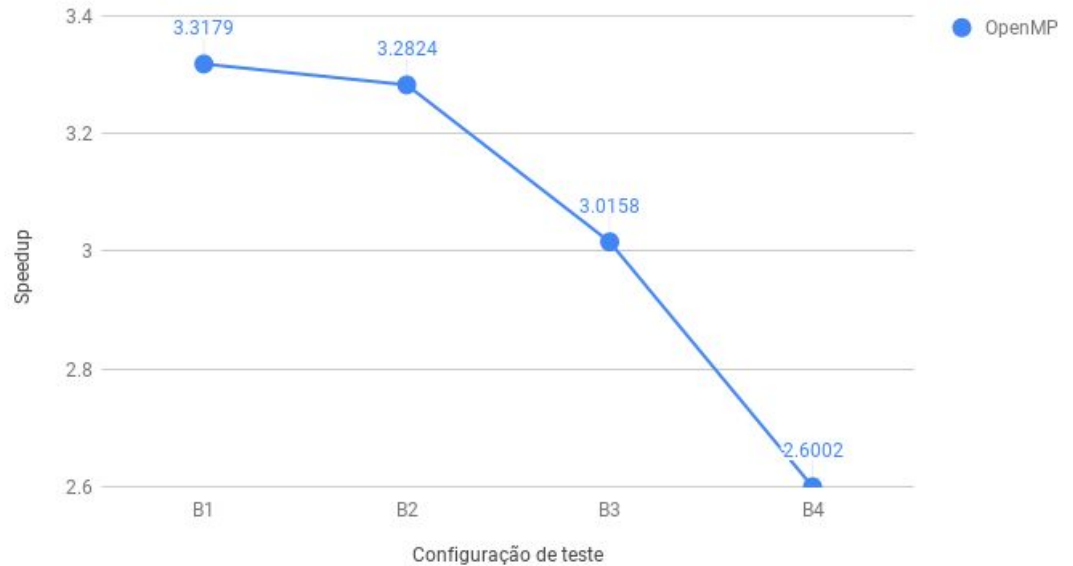
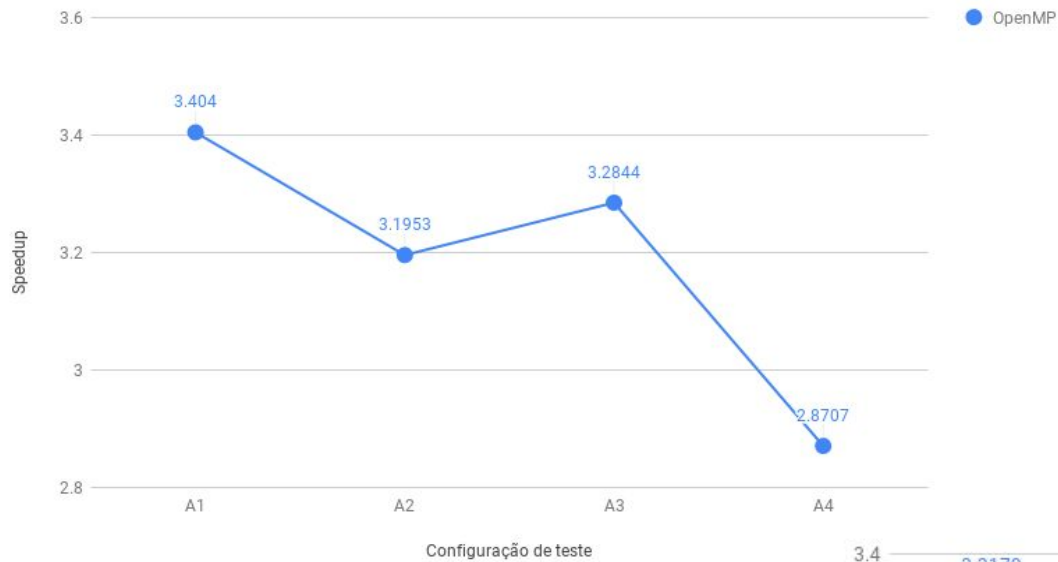
Speedup

$$S(p) = \frac{t_s}{t_p}$$

#	TEMPO (seg)
A1 { 100, 100, 100 }	3.404
A2 { 100, 100, 1000 }	3.1953
A3 { 100, 100, 10000 }	3.2844
A4 { 100, 100, 100000 }	2.8707

#	TEMPO (seg)
B1 { 200, 200, 200 }	3.3179
B2 { 200, 200, 2000 }	3.2824
B3 { 200, 200, 20000 }	3.0158
B4 { 200, 200, 200000 }	2.6002

Speedup



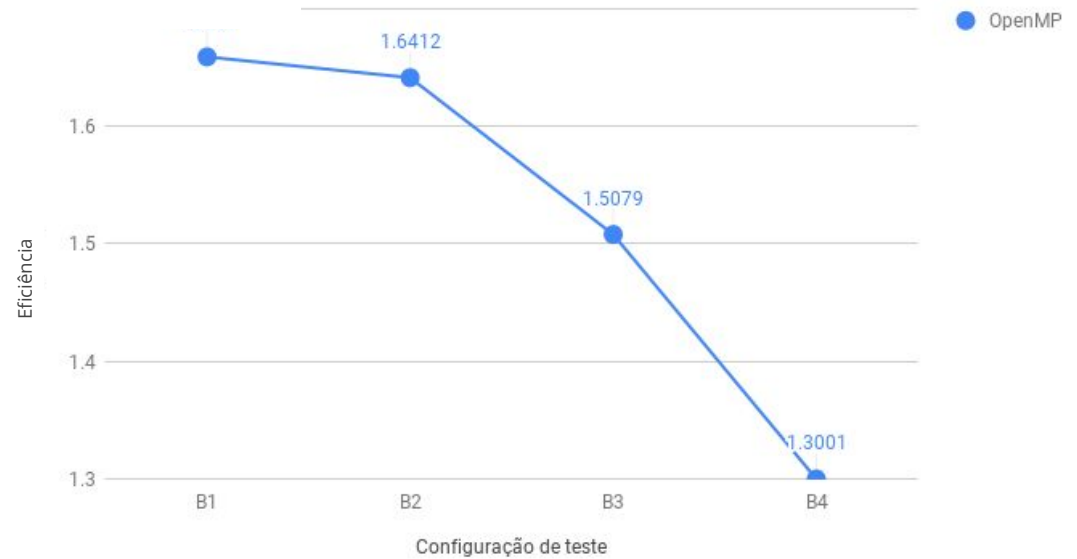
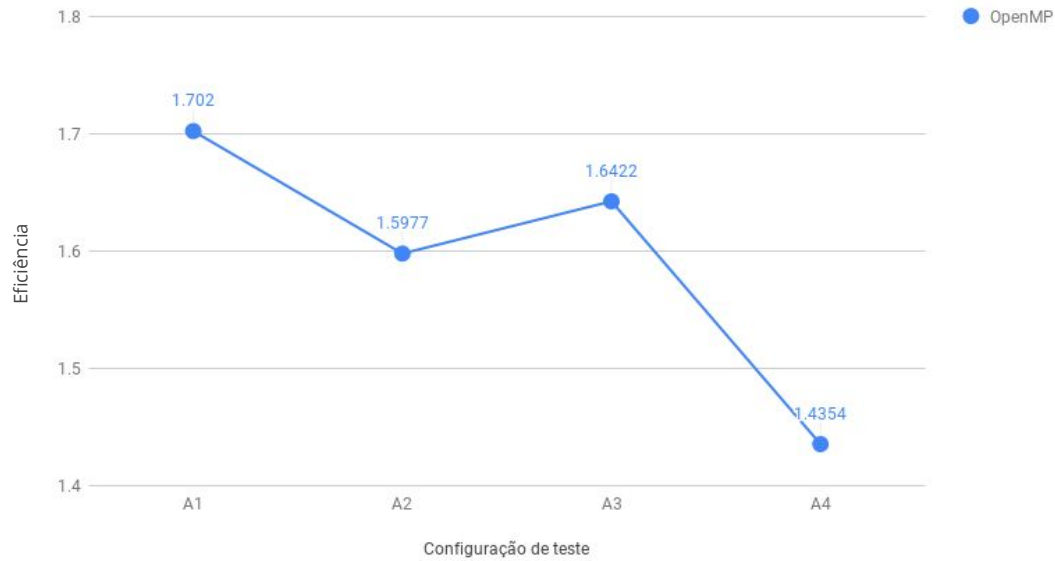
Eficiência

$$E = \frac{S(p)}{p}$$

#	TEMPO (seg)
A1 { 100, 100, 100 }	1.702
A2 { 100, 100, 1000 }	1.5977
A3 { 100, 100, 10000 }	1.6422
A4 { 100, 100, 100000 }	1.4354

#	TEMPO (seg)
B1 { 200, 200, 200 }	1.6589
B2 { 200, 200, 2000 }	1.6412
B3 { 200, 200, 20000 }	1.5079
B4 { 200, 200, 200000 }	1.3001

Eficiência



Referências

- Andrew Williams. **Computing the Mandelbrot set.** <https://goo.gl/apnk2A>
- Lawrence Livermore National Laboratory. **OpenMP.** <https://goo.gl/o2wTxR>
- Mark Bull. **OpenMP Tips, Tricks and Gotchas.** <https://goo.gl/L9Xhyp>
- Math - The University of Utah. **The Mandelbrot Set.** <https://goo.gl/XQeI4Y>
- OpenMP. **OpenMP C and C++ Application Program Interface.** <https://goo.gl/wPbQCn>
- OpenMP. **Summary of OpenMP 3.0 C/C++ Syntax.** <https://goo.gl/VdvSpi>
- SBAC-PAD/WSCAD. **Problems Set.** <https://goo.gl/Ef51T7>
- SBAC-PAD/WSCAD. **12th Marathon of Parallel Programming.** <https://goo.gl/eZX5gR>
- Wikipedia. **Mandelbrot set.** <https://goo.gl/u7CjRR>

Obrigado!

Perguntas?