

# EP1: Cálculo do Conjunto de Mandelbrot em Paralelo com Pthreads e OpenMP

Pedro Bruel, Anderson Andrei, e Alfredo Goldman

MAC 5742-0219 *Introdução à Programação Concorrente, Paralela e Distribuída*

## 1. Introdução

Você já ouviu falar do Conjunto de Mandelbrot<sup>1</sup>? Seu descobridor foi Benoit Mandelbrot, que trabalhava na IBM durante a década de 1960 e foi um dos primeiros a usar computação gráfica para mostrar como complexidade pode surgir a partir de regras simples. Benoit fez isso gerando e visualizando imagens de geometria fractal.

Um desses fractais foi nomeado *Conjunto de Mandelbrot* pelo matemático Adrien Douady. O Conjunto de Mandelbrot pode ser informalmente definido como o conjunto dos números complexos  $c$  para os quais a função  $f_c(z) = z^2 + c$  não diverge quando é iterada começando em  $z = 0$ . Isto é, a sequência  $f_c(0), f_c(f_c(0)), f_c(f_c(f_c(0))), \dots$  é sempre limitada. A Figura 1 mostra uma região do Conjunto de Mandelbrot conhecida como *Seahorse Valley*.

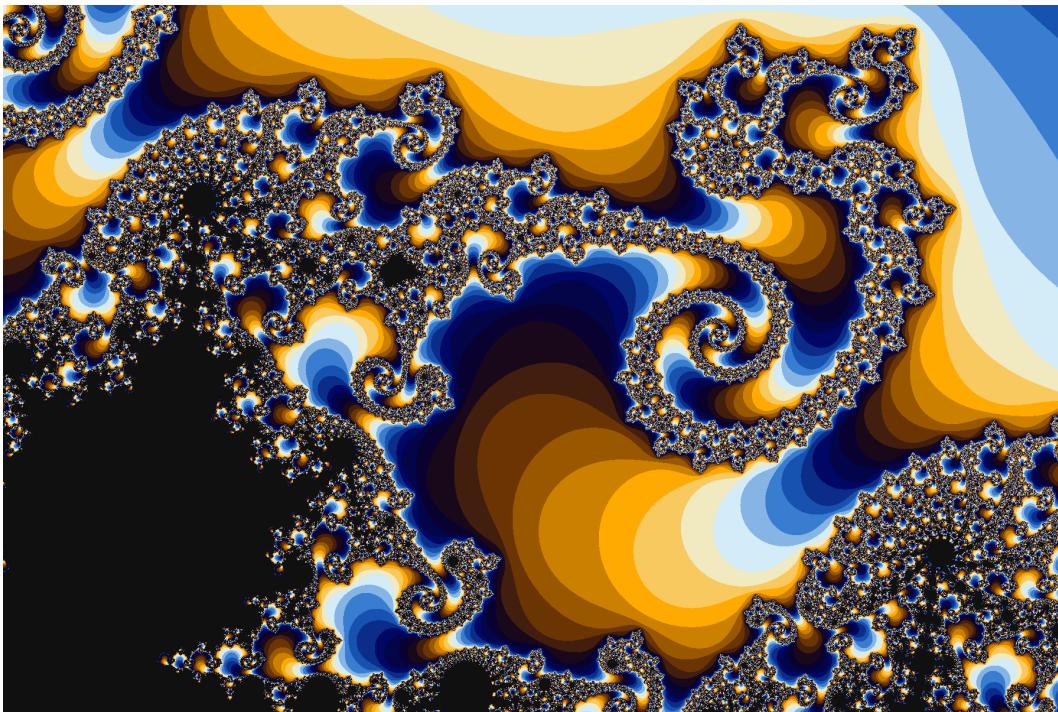


Figura 1: Seahorse Valley

As cores na Figura 1 indicam o número da iteração onde a função  $f_c(z)$  convergiu. A Figura 1 foi gerada usando o código fonte fornecido junto com este documento. O código na linguagem C e os arquivos em L<sup>A</sup>T<sub>E</sub>X necessários para gerar este documento estão disponíveis no *GitHub*<sup>2</sup>. O resto deste documento descreve as tarefas que você e seu grupo deverão realizar no EP1.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Mandelbrot\\_set](https://en.wikipedia.org/wiki/Mandelbrot_set) [Acessado em 24/04/2020]

<sup>2</sup><https://github.com/phrb/MAC5742-0219-EP1> [Acessado em 27/04/2020]

## 2. Tarefas

Você e seu grupo deverão parallelizar o código para o cálculo do Conjunto de Mandelbrot usando a biblioteca Pthreads e as diretivas de compilador fornecidas pelo OpenMP. Depois, vocês deverão medir o tempo de execução das versões sequencial, parallelizada com Pthreads e parallelizada com OpenMP. Você deve medir o tempo de execução para diferentes tamanhos de entrada e números de *threads* nas versões parallelizadas, em quatro regiões do Conjunto de Mandelbrot.

A implementação necessária para parallelizar o código fornecido usando Pthreads e OpenMP é relativamente simples. A parte mais trabalhosa do EP1 é a elaboração de um relatório, no formato *Jupyter Notebook* usando a linguagem Julia, que apresente gráficos com os resultados obtidos com as parallelizações, e os discuta de forma metódica, como vocês fizeram nos miniEPs.

### 2.1. Parallelização com Pthreads e OpenMP

O código fornecido está triplicado, e cada arquivo `.c` tem um propósito diferente:

- `mandelbrot_seq.c`: não precisa ser parallelizado
- `mandelbrot_omp.c`: deve ser parallelizado com OpenMP
- `mandelbrot_pth.c`: deve ser parallelizado com Pthreads

Vocês podem modificar à vontade os arquivos `mandelbrot_omp.c` e `mandelbrot_pth.c`, desde que eles continuem produzindo a mesma saída produzida por `mandelbrot_seq.c`. Vocês também podem modificar o arquivo sequencial, dado que a saída continue correta.

### 2.2. Experimentos

Para cada uma das três versões do programa, vocês deverão realizar medições do tempo de execução para diferentes tamanhos de entrada. Nas versões parallelizadas vocês deverão também medir, para cada tamanho de entrada, o tempo de execução para diferentes números de *threads*.

Vocês devem fazer um número de medições e analisar a variação dos valores obtidos. Sugerimos 10 medições para cada experimento, e também que vocês usem a média e o intervalo de confiança das 10 medições nos seus gráficos. Caso observem variabilidade muito grande nas medições, resultando num intervalo de confiança muito grande, vocês podem realizar mais medições, sempre apresentando a média e o intervalo de confiança. **Não é recomendado** fazer menos de 10 medições.

Vocês devem analisar também o impacto das porções não parallelizáveis do código sequencial: as operações de I/O e alocação de memória. Uma vez que você verifique que as versões parallelizadas produzem o resultado correto, elas **não precisam** realizar I/O e alocação de memória nos testes de desempenho, pois esses custos são fixos e assim aceleramos os experimentos.

A Tabela 1 lista os experimentos que devem ser feitos: os valores para o número de *threads* e de execuções, e os tamanhos de entrada. Cada experimento deverá ser repetido nas quatro regiões especificadas na Figura 6. As coordenadas para cada região podem ser obtidas executando no diretório `src`:

```
$ make mandelbrot_seq
gcc -o mandelbrot_seq -std=c11 mandelbrot_seq.c
$ ./mandelbrot_seq
usage: ./mandelbrot_seq c_x_min c_x_max c_y_min c_y_max image_size
examples with image_size = 11500:
  Full Picture:      ./mandelbrot_seq -2.5 1.5 -2.0 2.0 11500
  Seahorse Valley:   ./mandelbrot_seq -0.8 -0.7 0.05 0.15 11500
  Elephant Valley:   ./mandelbrot_seq 0.175 0.375 -0.1 0.1 11500
  Triple Spiral Valley: ./mandelbrot_seq -0.188 -0.012 0.554 0.754 11500
```

O número de iterações para o critério de convergência foi escolhido<sup>3</sup> de forma a produzir uma imagem interessante em diferentes níveis de magnificação, mas ter um tempo de execução razoável para tamanhos grandes de entrada.

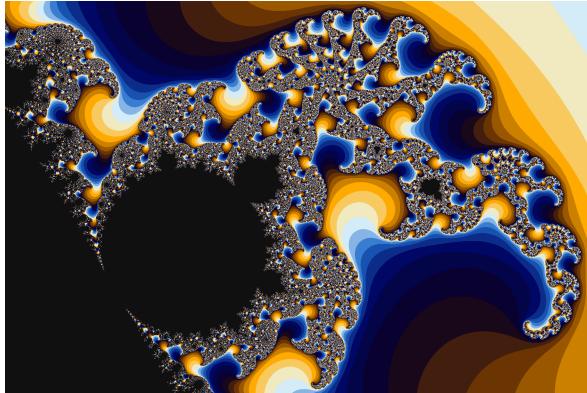


Figura 2: *Elephant Valley*

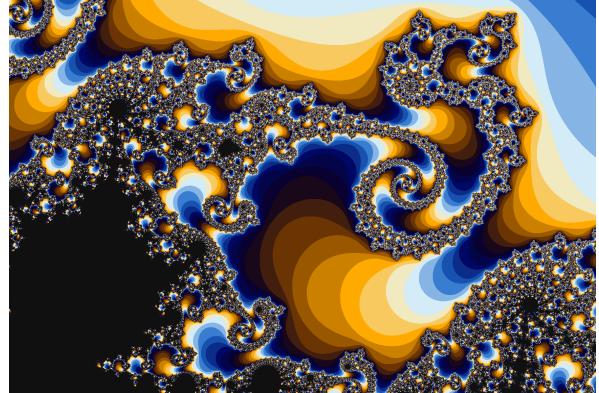


Figura 3: *Seahorse Valley*

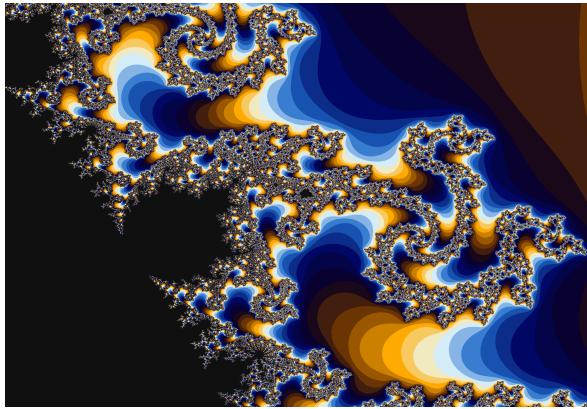


Figura 4: *Triple Spiral Valley*

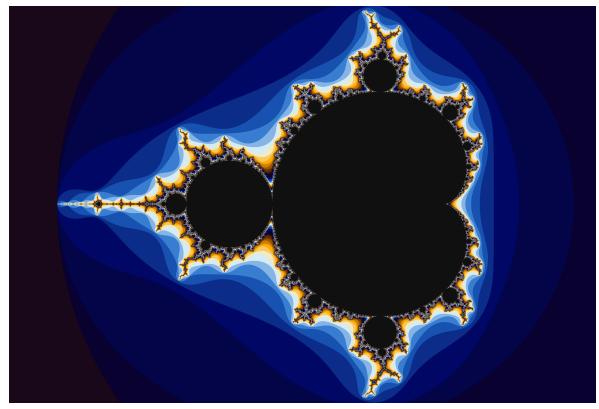


Figura 5: *Full Picture*

Figura 6: Regiões do Conjunto de Mandelbrot

	Pthreads	OpenMP	Sequencial
Regiões	<i>Triple Spiral, Elephant, Seahorse &amp; Full</i>		
I/O e Aloc. Mem.	Sem	Com e sem	
Nº de <i>Threads</i>	$2^0 \dots 2^5$	-	
Tamanho da Entrada		$2^4 \dots 2^{13}$	
Nº de Execuções		10	

Tabela 1: Experimentos

### 2.3. Apresentação dos Resultados

Depois de realizar os experimentos vocês deverão elaborar gráficos que evidenciem o comportamento das três versões do programa com relação à variação dos parâmetros descritos na

<sup>3</sup><https://goo.gl/WpL9hs> [Acessado em 27/04/2020]

Tabela 1. Os gráficos deverão ser claros e legíveis, com eixos nomeados. Deverão apresentar a média e o intervalo de confiança das 10 execuções para cada cenário experimental.

Neste EP, você e seu grupo deverão utilizar Notebooks Jupyter e a linguagem Julia para gerar os gráficos e produzir um documento com as análises. Os experimentos podem ser automatizados com a ferramenta *screen*, ou feitos a partir do Notebook, como nos miniEPs. A automação dos experimentos e da visualização dos dados gerados é fundamental para a pesquisa em Ciência da Computação, pois permite gerar e analisar grandes conjuntos de dados sem muito esforço manual.

#### 2.4. Discussão dos Resultados

Vocês deverão analisar os resultados obtidos e tentar responder a algumas perguntas:

- Como e por que as três versões do programa se comportam com a variação:
  - Do tamanho da entrada?
  - Das regiões do Conjunto de Mandelbrot?
  - Do número de *threads*?
- Qual o impacto das operações de I/O e alocação de memória no tempo de execução?

Vocês conseguem pensar em mais perguntas interessantes?

### 3. Entrega

Vocês deverão entregar no site do curso **um único arquivo por grupo**, no formato **.tar**, contendo:

- Um arquivo **.ipynb** com as análises e gráficos
- As três versões do código fonte usado
- Um arquivo **.csv** com as medições feitas

A entrega deve ser feita até o **dia 21/05/20**.

### 4. Tecnologias

Esta seção descreve brevemente algumas tecnologias usadas no EP1.

#### 4.1. Shell scripting & GNU screen

Para usar a máquina virtual vocês vão precisar usar um emulador do *shell* do Linux, como o **bash**. O arquivo **run\_measurements.sh** contém o necessário para gerar as medições de tempo de execução das três versões do programa, mas vocês vão precisar modificá-lo para medir o impacto das operações de I/O e alocações de memória.

Para deixar o *script* rodando na sua máquina, faça o seguinte:

```
$ screen  
$ ./run_measurements.sh  
<Ctrl+A><D>
```

O comando **screen** lança uma seção da qual você pode se desconectar sem parar a execução de um comando. A sequência **<Ctrl+A>** seguida de **<D>** desconectará você da sessão. Para voltar, basta executar:

```
$ screen -r
```

#### 4.2. Linux *perf*

Um dos comandos executados por `run_measurements.sh`, expandindo as variáveis, é:

```
$ perf stat -r 10 ./mandelbrot_seq -2.5 1.5 -2.0 2.0 512
```

O comando acima executa dez repetições da versão sequencial do programa, calculando a imagem inteira do Conjunto de Mandelbrot com um tamanho de imagem de 512 *pixels*. Modificar os parâmetros desse comando será suficiente para realizar os experimentos do EP1.

### 5. Critério de Avaliação

A nota do EP1 vai de **0.0** a **10.0**, e a avaliação será feita da maneira descrita a seguir.

#### 5.1. Apresentação e Análise de Medidas para o Programa Sequencial

Vale **2.0**, divididos da seguinte maneira:

- Relatório: **2.0**
  - Apresentação e Análise dos Experimentos: **1.8**
  - Clareza do texto e figuras: **0.2**

#### 5.2. Apresentação e Análise de Medidas para o Programa em Pthreads

Vale **4.0**, divididos da seguinte maneira:

- Implementação: **2.0**
  - Código compila sem erros e *warnings*: **0.9**
  - Código executa sem erros e produz o resultado correto: **0.9**
  - Boas práticas de programação e clareza do código: **0.2**
- Relatório: **2.0**
  - Apresentação e Análise dos Experimentos: **1.8**
  - Clareza do texto e figuras: **0.2**

#### 5.3. Apresentação e Análise de Medidas para o Programa em OpenMP

Vale **4.0**, divididos da seguinte maneira:

- Implementação: **2.0**
  - Código compila sem erros e *warnings*: **0.9**
  - Código executa sem erros e produz o resultado correto: **0.9**
  - Boas práticas de programação e clareza do código: **0.2**
- Relatório: **2.0**
  - Apresentação e Análise dos Experimentos: **1.8**
  - Clareza do texto e figuras: **0.2**

Se tiver dúvidas, envie uma mensagem no fórum do curso, ou envie e-mails para `pdro.bruel@gmail.com`, `anderson.andrei.silva@usp.br`, ou `gold@ime.usp.br`. Bom EP!