

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

Pedro Bruel e Alfredo Goldman

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

1. Introdução

Você já ouviu falar do Conjunto de Mandelbrot¹? 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*.

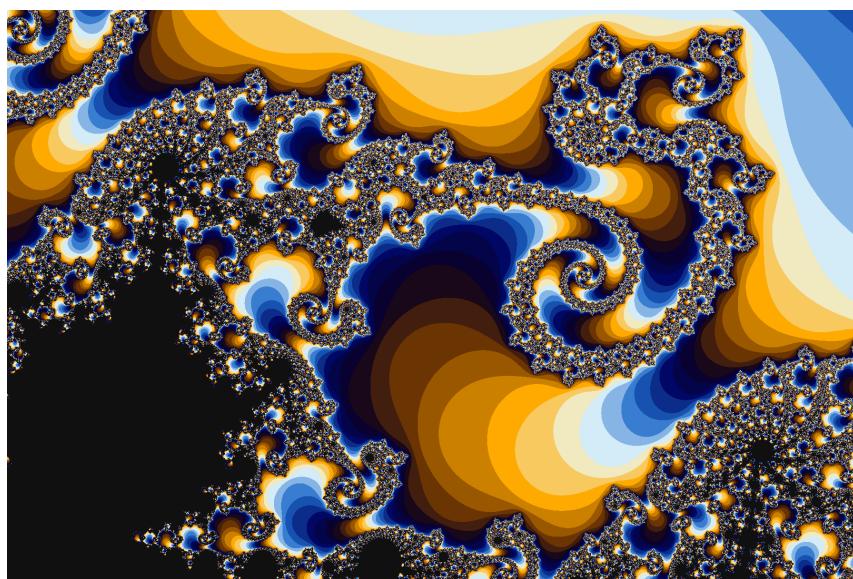


Figura 1: Seahorse Valley

¹https://en.wikipedia.org/wiki/Mandelbrot_set [Acessado em 30/03/2017]

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^AT_EX necessários para gerar este documento estão disponíveis no *GitHub*². O resto deste documento descreve as tarefas que você e seu grupo deverão realizar no EP1.

2. Tarefas

Você e seu grupo deverão paralelizar 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, paralelizada com Pthreads e paralelizada com OpenMP. Você们 deverão 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 paralelizar o código fornecido usando Pthreads e OpenMP é relativamente simples. A parte mais trabalhosa do EP1 é a elaboração de um relatório, seguindo o padrão deste documento, que apresente e discuta de forma científica os resultados que vocês obtiverem com as paralelizações.

2.1. Paralelização com Pthreads e OpenMP

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

- **mandelbrot_seq.c**: não precisa ser paralelizado
- **mandelbrot_omp.c**: deve ser paralelizado com OpenMP
- **mandelbrot_pth.c**: deve ser paralelizado 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**.

²<https://github.com/phrb/MAC5742-0219-EP1> [Acessado em 30/03/2017]

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 paralelizadas vocês deverão também medir, para cada tamanho de entrada, o tempo de execução para diferentes números de *threads*. Preferencialmente os experimentos deverão ser realizados na Google Compute Engine (GCE), usando os US\$300 “de brinde” que o serviço fornece por um ano. O tipo de instância que vocês devem usar é o **n1-standard-8**, que tem 8 *cores* e 30 GB de memória.

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 desvio padrão das 10 medições nos seus gráficos. Vocês podem fazer mais medições, desde que apresentem a média e o desvio padrão. **Não é recomendado** fazer menos de 10 medições.

Vocês devem analisar também o impacto das porções não paralelizáveis do código sequencial: as operações de I/O e alocação de memória. As versões paralelizadas **não precisam** realizar I/O e alocação de memória.

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 regiões³ 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 escolhido⁴ de forma a produzir uma imagem interessante em diferentes níveis de magnificação, mas ter um tempo de execução razoável para tamanhos grandes de entrada.

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 Tabela 1. Os gráficos deverão ser claros

³<http://www.nahee.com/Derbyshire/manguide.html> [Acessado em 30/03/2017]

⁴<https://goo.gl/WpL9hs> [Acessado em 30/03/2017]

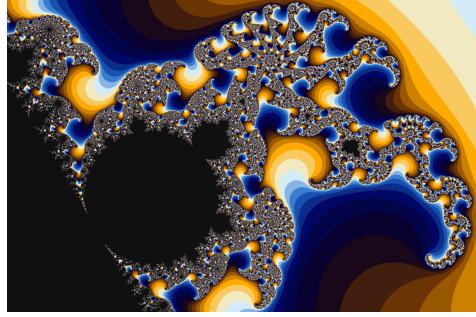


Figura 2: *Elephant Valley*

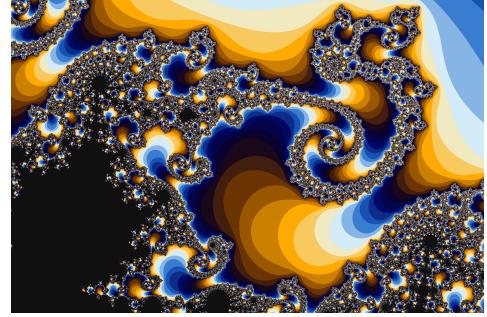


Figura 3: *Seahorse Valley*

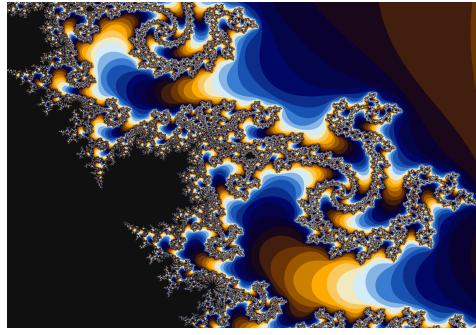


Figura 4: *Triple Spiral Valley*

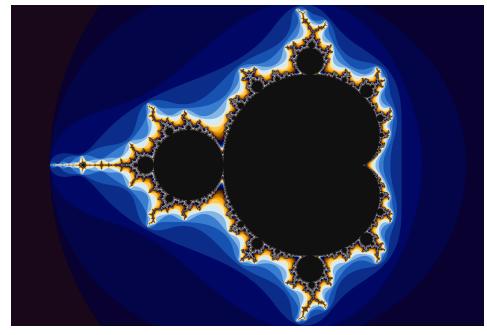


Figura 5: *Full Picture*

Figura 6: Regiões do Conjunto de Mandelbrot

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

Tabela 1: Experimentos

e legíveis, com eixos nomeados. Deverão apresentar a média e o desvio padrão das 10 execuções para cada cenário experimental.

Recomendamos que vocês usem ferramentas como a biblioteca `matplotlib` da linguagem `Python`. Se fizerem isso vocês conseguirão automatizar a realização dos experimentos e a geração dos gráficos. A automatização dos experimentos e da visualização dos dados gerados é fundamental para 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:

- Por que você acha que fizemos a recomendação de se realizar mais de uma medição?
- Por que você acha que existe variabilidade entre execuções do mesmo programa?
- 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?

2.5. Entrega no PACA

Vocês deverão entregar no PACA **apenas um relatório e código fonte por grupo**. A entrega deve ser um único arquivo nos formatos `.tar`, `.zip`, ou qualquer formato que o `tar` consiga descompactar. A entrega deve ser feita **até dia xx/xx**.

3. Tecnologias

Esta seção descreve brevemente algumas tecnologias usadas no EP1. O monitor estará disponível na **Sala 00 às Quintas**, das **10h às 11h30**, para tirar dúvidas do EP1.

3.1. Google Compute Engine (GCE)

O grupo deve possuir uma conta no Google e se cadastrar na Google Compute Engine. Será necessário fornecer um cartão de crédito, mas não se preocupem, pois a GCE não cobrará nada automaticamente, nem mesmo ao fim do ano gratuito.

Com a conta vocês terão acesso a US\$300 que poderão usar por um ano. Isso é **muito mais do que suficiente** para este EP e pra vocês brincarem à vontade durante um ano.

Vocês devem criar uma instância do tipo **n1-standard-8**, que tem 8 *cores* e 30 GB de memória, usando a imagem que o monitor vai compartilhar com o e-mail que vocês forneceram no registro do Grupo.

3.2. 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 instância da GCE, 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
```

3.3. 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.

3.4. L^AT_EX

Instalem o L^AT_EX na máquina que vão usar para escrever o relatório e usem o arquivo **enunciado_ep1.tex** e o **Makefile** no repositório do EP1 como modelo.

4. Critério de Avaliação

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

4.1. Apresentação e Análise de Medidas para o Programa Sequencial

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

- Relatório: **2.0**

- Apresentação e Análise para o Tempo de Execução: **0.9**
- Apresentação e Análise para o Tempo de Execução sem I/O e sem memória: **0.9**
- Clareza do texto e figuras: **0.2**

4.2. 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 para o Tempo de Execução: **0.9**
- Apresentação e Análise para o Tempo de Execução sem I/O e sem memória: **0.9**
- Clareza do texto e figuras: **0.2**

4.3. 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 para o Tempo de Execução: **0.9**
- Apresentação e Análise para o Tempo de Execução sem I/O e sem memória: **0.9**
- Clareza do texto e figuras: **0.2**

O monitor estará disponível na **Sala 00 às Quintas**, das **10h às 11h30**, para tirar dúvidas do EP1. Bom EP!

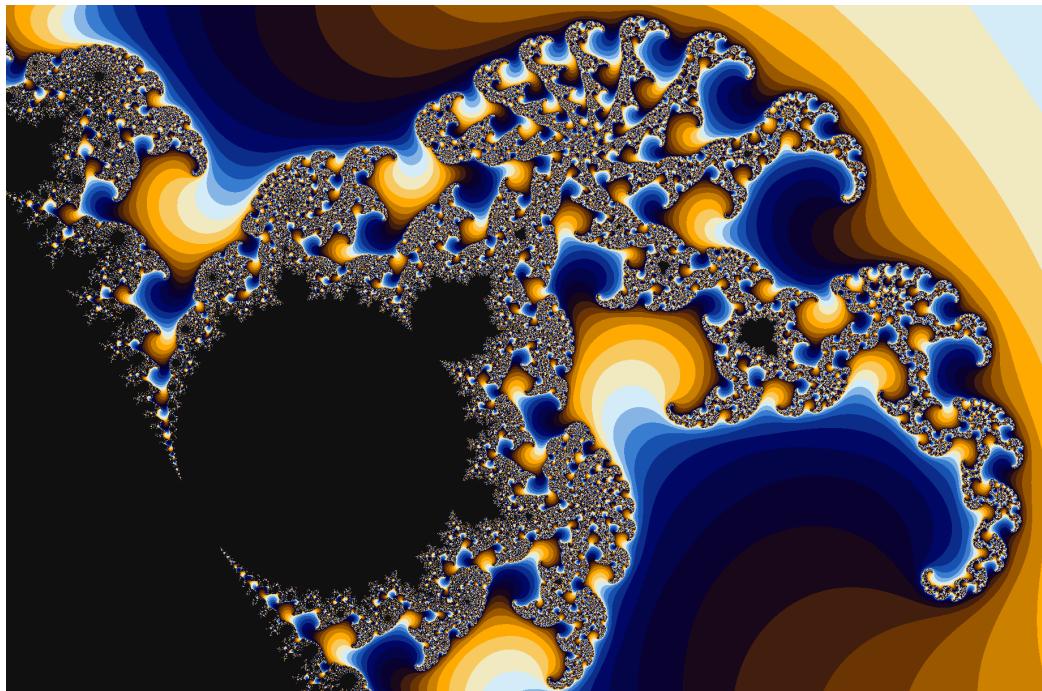


Figura 7: Elephant Valley