MAC 0219/5742 – Introdução à Computação Concorrente, Paralela e Distribuída

Prof. Dr. Alfredo Goldman

EP2 - Mandelbrot `versão 0.9`

Enunciado original: Pedro Bruel

Monitores: Giuliano Belinassi e Matheus Tavares

1 Introdução

Você já ouviu falar do Conjunto de Mandelbrot[1] [2]? 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.

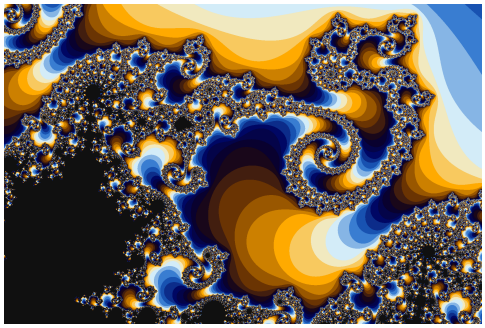

Figura 1: Seahorse Valley. Fonte: Pedro Bruel.

As cores na Figura 1 indicam o número da iteração onde a sequência $z^2 + c$ divergiu.

2 Definição Matemática e Heurística de Cálculo

Aqui vamos denotar $\mathbb{C}$ como sendo o corpo dos números complexos, $i^2 = -1$ a raiz imaginária, e se $z = a + bi$ é um número complexo, onde a, b são números reais, então $|z| = \sqrt{a^2 + b^2}$ é seu valor absoluto.

Seja $c \in \mathbb{C}$ um número complexo, n um inteiro não negativo, e considere a sequência nos complexos:

$$z_n = \begin{cases} 0, & \text{se } n = 0 \\ z_{n-1}^2 + c, & \text{se } n > 0 \end{cases} \tag{1}$$

Dizemos que c pertence ao Conjunto de Mandelbrot se, para todo inteiro $j \geq 0$, vale que $|z_j| \leq 2$.

[1] `https://en.wikipedia.org/wiki/Mandelbrot_set`

[2] `https://www.youtube.com/watch?v=NGMRB4O922I`

Infelizmente, não podemos verificar computacionalmente[3] todos os inteiros $j \geq 0$ para decidir se c pertence ao Conjunto de Mandelbrot, mas podemos usar a seguinte heurística: Defina um número máximo de iterações M. Se $|z_j| \leq 2$ para todo $j < M$, então pinte o ponto correspondente de **preto**, indicando que ele pertence ao Conjunto. Caso o valor ultrapasse 2 em uma certa iteração j, então a cor a ser pintada deverá ser proporcional a j.

Sendo assim, para gerar a imagem do Conjunto de Mandelbrot é necessários escolher os seguintes atributos:

1. Dois pontos $c_0, c_1 \in \mathbb{C}$ determinando os dois extremos do retângulo da imagem.
2. As dimensões $H \times W$ da imagem, onde W é sua largura e H é sua altura, ambas em pixels. Com isto, você pode definir um tamanho de passo Δx e Δy.
3. Um número inteiro $M > 0$ determinando o número máximo de iterações da heurística descrita acima.

A Figura 2 ilustra os parâmetros descritos acima.

3 Tarefas

Neste EP, vocês deverão concluir as seguintes tarefas:

1. **Implementação Sequencial** – Implementar a heurística do cálculo do Conjunto de Mandelbrot com a finalidade de gerar uma imagem. Vocês estão livres para escolher quais cores usar em sua implementação e como atribuir as intensidades em função de j. Os cálculos devem ser feitos utilizando ponto flutuante IEEE-754 de 32-bits (o tipo `float`, na maioria das arquiteturas).
2. **Paralelização com OpenMP** – A partir do código sequencial implementado, vocês deverão utilizar as diretivas de compilador OpenMP para paralelizar o código em CPU. Essa tarefa deve ser extremamente simples.
3. **Paralelização em GPU** – Novamente, a partir do código sequencial implementado, vocês deverão utilizar a extensão CUDA para implementar o cálculo do Conjunto de Mandelbrot na GPU. Os cálculos também deverão ser feitos em ponto flutuante de 32-bits.

   Nota: Se você não tem acesso à um computador com GPU da NVIDIA, você pode utilizar a máquina `dota` da rede linux [4]. Mas note que esta máquina possui duas GPUs, então é necessário utilizar a função `cudaSetDevice()` antes de chamar as operações de CUDA. Os alunos da pós também podem usar a máquina `brucutuIV` da rede IME. **Dica:** Sempre que rodarem, em máquinas com sistemas de arquivos em rede, programas que façam leitura/escrita em disco, salvem seus arquivos em

[3] `https://cs.stackexchange.com/questions/42685/in-what-sense-is-the-mandelbrot-set-computable`

[4] Mais informações em: `https://linux.ime.usp.br/wiki/doku.php?id=faq:cuda`

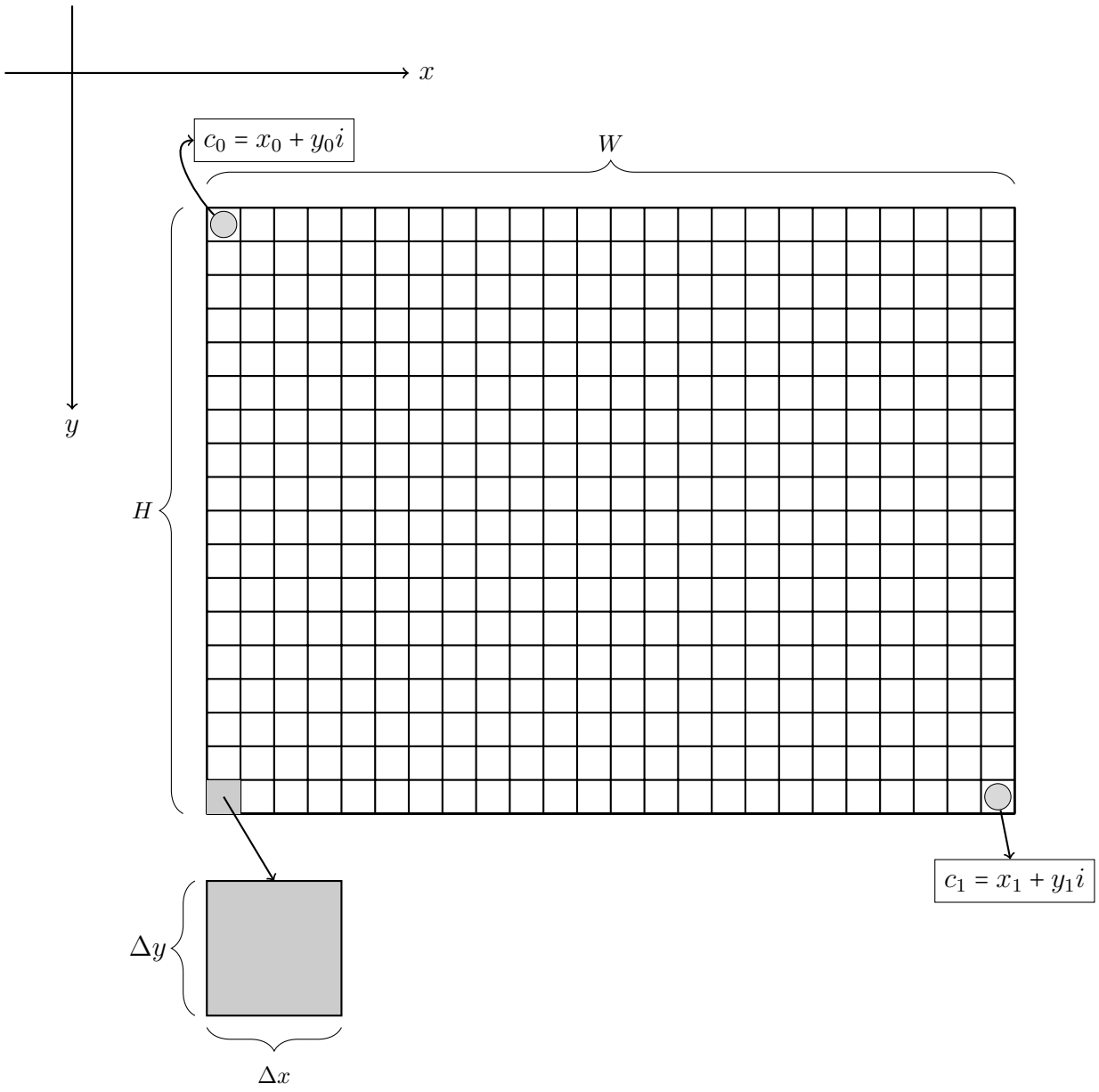

Figura 2: Esquema dos atributos da imagem a ser gerada.

diretórios locais da máquina como `/tmp` ou `/var/tmp` e só ao fim da execução movam-nos para sua `home`. Isto é importante pois escrever e ler da `home` provavelmente implicará um acesso em rede, que pode atrasar seu programa.

Caso você tenha dificuldades de usar a `dota` ou `brucutuIV`, envie uma mensagem para os monitores por email. Existe uma outra máquina que possívelmente lhes podemos conceder acesso para realizar os experimentos.

4. **Coleta dos Resultados** – Por fim, vocês deverão medir o tempo de execução das versões sequencial, paralelizada com OpenMP e na GPU. Vocês deverão medir o tempo de execução para diferentes tamanhos de entrada e números de *threads* nas versões em OpenMP, e diferentes números de *threads* por *bloco* (e por consequência, o número de blocos para cobrir todo o problema) na versão GPU. Certifiquem-se de testar também com um número de *threads* por *bloco* múltiplo de 32, e um número não múltiplo de 32[5], visando observar se há diferença de tempo. Lembrando novamente que é necessário alguma base estatística nesse experimento.

   Com os tempos calculados, vocês deverão medir o *Speedup* da versão paralelizada com OpenMP, e da versão em GPU, usando como base a versão sequencial em CPU. Não esqueçam de especificar *com detalhes* qual CPU e qual GPU vocês usaram nos testes, quantas execuções foram feitas, intervalo de confiança, tamanho da imagem, regiões do Conjunto de Mandelbrot usados, e outras coisas que vocês acharam interessantes.

5. **Tarefa Bônus:** – Como um bônus, vocês também podem implementar os cálculos utilizando ponto flutuante IEEE-754 de 64-bits, em ambas as versões CPU e em GPU, e comparar as velocidades. A razão desse experimento é que normalmente as placas de vídeo da NVIDIA inserem 1 unidade de precisão dupla para cada 32 unidades de precisão simples, o que implica em uma performance de 1/32 avos quando comparado com a performance de pico.

   Nota: Se você implementar o bônus, não se esqueça de mostar as comparações de tempo no relatório.

4 Especificação do Programa

Seu programa deve ser capaz de atender os seguintes requisitos:

1. Gerar uma imagem em formato `PNG` de acordo com o nome do arquivo de saída especificado no `ARGV`. Recomendamos o uso da `libpng` para isso.

2. Aceitar um parâmetro no `ARGV` especificando se o cálculo do Conjunto de Mandelbrot será efetuado na CPU ou na GPU. No caso CPU, deverá ser possível especificar o número de *threads* a ser usado. No caso GPU, deverá ser possível especificar o número de *threads* por *bloco* a ser usado.

[5]Para entender porque usar um valor de *bloco* múltiplo de 32 ou não e porque isso pode afetar a performance, estude sobre *warps* em programação para GPU.

3. Conter um `Makefile` que gera o binário `mbrot`. Este binário deve aceitar como argumento a seguinte linha de comando:

   ```
   mbrot <C0_REAL> <C0_IMAG> <C1_REAL> <C1_IMAG> <W> <H>
                 <CPU/GPU> <THREADS> <SAIDA>
   ```

   onde:

   - `C0_REAL` é a parte real de c_0.
   - `C0_IMAG` é a parte imaginária de c_0.
   - `C1_REAL` é a parte real de c_1.
   - `C1_IMAG` é a parte imaginária de c_1.
   - `W` é a largura em pixels da imagem a ser gerada.
   - `H` é a altura em pixels da imagem a ser gerada.
   - `CPU/GPU` é `cpu` se especificado para executar na CPU, e `gpu` se especificado pra executar na GPU.
   - `THREADS` é o número de *threads* a ser executado na CPU, e o número de *threads* por *bloco* na GPU.
   - `SAIDA` é o caminho para o arquivo de saída.

   Novamente, você não precisa sanitizar os parâmetros de entrada.

5 Dicas

1. Em C, vocês podem implementar a precisão simples e dupla simultaneamente especificando uma macro `REAL` como sendo o tipo do ponto flutuante, e recompilar o programa. Veja a flag de compilação `-D` do GCC. Em C++ é ainda mais fácil, pois basta usar `templates`.

2. Para usar números complexos em C, basta incluir o `complex.h`. Em C++ e em CUDA C++, vocês podem usar o `thrust::complex<>`. Todos esses tipos de complexos são compatíveis *bit* a *bit* uns com os outros, ou seja, você pode efetuar um *cast* sem problemas.

3. Você pode usar os utilitários `nvidia-smi` e `cudatop` para examinar as características de sua GPU, bem como ver informações de processos em execução.

4. Este EP demanda um entendimento do funcionamento de OpenMP e CUDA. **Não deixe este EP para a última hora**.

6 Entrega

Deverá ser entregue um pacote no sistema PACA com uma pasta com o nome e o sobrenome do estudante que o submeteu no seguinte formato: `nome_sobrenome.zip`. Se o EP for feito em trio, o formato deve ser:

`nome1_sobrenome1.nome2_sobrenome2.nome3_sobrenome3.zip`

Somente um estudante do time submeterá a tarefa. Essa pasta deve ser comprimida em formato `ZIP` e deve conter dois itens:

- Os códigos fonte do programa, em conjunto com um `Makefile` que o compila, gerando o executável especificado.
- Um relátorio em `.txt` ou `.pdf` contendo o nome dos integrantes, as informações pedidas na seção `Coleta dos Resultados` e uma breve explicação sobre a sua solução e desafios encontrados. Imagens também podem ser inseridas no arquivo. Relatórios em `.doc`, `.docx` ou `odt` **não** serão aceitos.

Em caso de dúvidas, use o fórum de discussão do Paca. A data de entrega deste Exercício Programa é até às **16:00h do dia 23 de Maio**. Não se esqueça que o professor irá sortear um aluno para explicar sua implementação do EP neste mesmo dia. Não é necessário preparar slides, apenas falar brevemente sobre o código e as decisões de projeto tomadas.