Primeiro Trabalho - Relatório Final

Computação Concorrente (MAB117) - 2021/1 REMOTO

Cálculo e visualização do conjunto de Mandelbrot

Gabriel Magalhães Nunes de Souza - 118088665
Raphael Felipe de Cantuaria Mesquita - 118020104

Link para o repositório: https://github.com/gbrmgls/comp-conc/tree/main/trab1

1. Descrição do problema

O que é?

"O conjunto de Mandelbrot é um fractal definido como um conjunto de pontos *c* no plano complexo para o qual a sequência, definida recursivamente,

$$z_0 = 0$$

$$z_{n+1} = {z_n}^2 + c$$

não tende ao infinito."

Como calcular?

Se cada ponto *c* do plano complexo é na forma

$$c = x + iy$$

então a sequência se expande como

$$Z_0 = 0$$

$$\begin{aligned} Z_1 &= Z_0^2 + c \\ &= x + iy \end{aligned}$$

$$\begin{aligned} Z_2 &= Z_1^2 + c \\ &= (x + iy)^2 + x + iy \\ &= x^2 + 2ixy - y^2 + x + iy \\ &= x^2 - y^2 + x + (2xy + y)i \end{aligned}$$

$$Z_3 = Z_2^2 + c = \ldots$$

Se separarmos, na sequência, os termos reais e imaginários como as coordenadas x e y do plano complexo e substituirmos z_n pelo ponto $x_n + y_n i$ e *c* pelo ponto *a* + *bi*, temos

$$x_{n+1} = {x_n}^2 - {y_n}^2 + a$$

e

$$y_{n+1} = 2x_n y_n + b$$

Isso permite que implementemos um algoritmo mais simples e apresentemos graficamente o fractal.

No entanto, isso não é suficiente. Aplicar infinitas expansões ou iterações é inviável. Então escolhe-se um número limitado *n* de iterações para definir se um ponto está no conjunto. Dessa forma, falta apenas uma forma de descobrir se a partir de um ponto *c* dado, a expansão tenderá ao infinito. De fato existe uma forma para isso:

"Uma vez que o valor absoluto de z_n é maior que 2 (na forma cartesiana, quando $x_n^2 + y_n^2 > 2^2$) a sequência tenderá ao infinito e, portanto, *c* não pertence ao *conjunto de Mandelbrot*".

Assim, obtemos uma forma viável para o cálculo e apresentação do fractal.

Como apresentar/desenhar?

Existem várias formas para apresentar graficamente o conjunto de Mandelbrot. Uma dessas formas é pintar de preto todos os pontos que estão contidos no conjunto e pintar, na escala de cinza, baseado no número de iterações, os pontos que não pertencem ao conjunto. Ou seja, quanto maior o número de iterações necessárias para definir que a expansão de um ponto tende ao infinito, mais branco fica. Assim, se considerarmos que os pontos no plano complexo são *pixels* na tela, conseguimos apresentar o fractal de Mandelbrot.

Pontos a se considerar no processamento

Apresentar o fractal depende diretamente de qual a dimensão da região da tela precisaremos para desenhá-la. Depende também de quantas iterações utilizaremos para definir se um ponto está no conjunto de Mandelbrot ou não. Então, se quisermos apresentar o fractal numa região da tela de 500x500 *pixels*, onde cada ponto/pixel é calculado com *n=1000* iterações, teremos passado por $2{,}5 \times 10^8$ iterações - um número consideravelmente elevado.

Paralelismo de dados

O modo para se formar a figura é iterar cada pixel e realizar os cálculos. Mas como a sua apresentação é basicamente colorir os *pixels* na tela, então cada pixel pode ser processado individualmente. Assim é possível aplicar uma forma concorrente para resolver esse problema.

2. Projeto inicial da solução concorrente

Estratégias iniciais

Uma possível estratégia para calcular o conjunto de Mandelbrot é dividir a tela em X vetores com Y de comprimento (sendo X a largura da tela e Y a altura) e alocar cada vetor para uma thread. Para escolher qual thread calcula quais vetores,

podemos dividir igualmente uma sequência de vetores em sequência para cada thread, utilizando uma relação entre a divisão da largura da tela pelo número de threads e o índice de cada thread.

Podemos também dividir o trabalho entre as threads fazendo com que elas alternem entre os vetores; no caso de duas threads, por exemplo, alocamos os vetores de índice par na primeira thread e de índice ímpar na segunda thread.

Desbalanceamento de carga

Essas estratégias, porém, causam um desbalanceamento na carga que cada thread tem. No melhor caso, se nenhum pixel de um vetor necessitar de mais de uma iteração para verificar se pertence ao conjunto, todo o vetor fará no máximo Y iterações no total. Por outro lado, no pior caso, se todos os pixels precisarem de 1000 iterações, uma única thread terá que calcular 1000 x Y iterações no total.

Estratégia definitiva

A estratégia escolhida por nós foi a de calcular um pixel por thread. Ao iniciar, a thread verifica uma variável global que indica qual é o último pixel cujo todos os pixels anteriores já foram processados e então seleciona o pixel na posição seguinte. Cada thread avisa que está processando o pixel quando começar a verificação, e, então, o marcará como processado ao terminar. Logo em seguida ela verifica se o próximo pixel está sendo ou já foi processado, e, em caso positivo, continuará procurando até o próximo pixel que ainda não foi calculado e indicará o pixel anterior na variável global de controle. Após essa verificação, ela verifica a variável de controle e começa o processo novamente.

Essa estratégia não só balanceia a carga de processamento das threads igualmente, mas também evita que a matriz inteira seja varrida procurando pelo próximo pixel não processado toda vez que uma thread for iniciada.

Estruturas de dados principais

Cada pixel é uma estrutura de dados que permite que as informações necessárias para sua representação gráfica pela biblioteca especializada sejam salvas, bem como seu estado atual de processamento. Os pixels são representados em uma matriz de resolução arbitrária.

3. Projeto inicial dos casos de teste

Para testar a corretude do cálculo do algoritmo, utilizamos uma matriz similar calculada de modo sequencial. Essa matriz será comparada pixel por pixel com seu resultado concorrente em uma bateria de testes com matrizes de diversas resoluções. O ganho de performance do algoritmo será determinado pelo tempo utilizado para o cálculo concorrente e sequencial de toda a matriz.

Durante a execução do código principal, executamos o algoritmo sequencial e o algoritmo concorrente seguidos um do outro e calculamos o tempo necessário

para o término de cada um individualmente. O mesmo frame é calculado pelas duas implementações, nos dando assim dados de melhoria de velocidade a cada quadro mostrado resultante na animação de aproximação a um ponto do conjunto. No código entregue, cada execução calcula 100 quadros por algoritmo, totalizando 200 quadros. No final de cada iteração, a igualdade entre os resultados sequencial e concorrente é verificada e, caso os dois sejam equivalentes em todas as execuções, o programa retorna um sinal positivo. Caso contrário, retorna um sinal negativo.

Os testes foram executados em um Intel Core I7 de 12 núcleos rodando o sistema operacional Ubuntu na versão 21.04.

Número de threads	**Número máximo de iterações**	**Tempo total sequencial (em segundos)**	**Tempo total concorrente (em segundos)**	**Speed up**
2	10000	27.696645	16.630900	1.665373
2	100000	41.389629	23.454101	1.764708
2	1000000	179.157404	93.016747	1.926077
4	10000	27.532828	10.952933	2.513740
4	100000	41.185765	14.491163	2.842130
4	1000000	178.826789	50.293225	3.555683
8	10000	27.514555	9.435707	2.916003
8	100000	41.441378	10.983802	3.772954
8	1000000	178.639791	31.226739	5.720732

4. Discussão

O desempenho do código foi conforme o esperado, pois pela natureza do código, o controle do cálculo de cada pixel é independente e não necessita de algoritmos muito avançados para obter melhora na performance concorrente.

Em questão de melhorias, tendo em vista que o cálculo matemático do conjunto de Mandelbrot é estudado há tempos, existem outras formas mais avançadas para melhorar a performance do código sequencial, mas que fogem do objetivo do projeto. Para melhor visualização do usuário, uma limitação na taxa de quadros por segundo da atualização da janela pode ser implementada.

Trabalhar com a lógica concorrente, lidando com o gerenciamento de acesso de memória compartilhado pelas threads é um conceito novo para os componentes da dupla, o que gerou certas dificuldades na implementação do código, levando em consideração o costume com o pensamento sequencial. Fora isso, nenhuma outra grande dificuldade foi grande o suficiente para ser mencionada.

5. Referências bibliográficas

Getting Started with OpenGL using GLFW [C++]**:**
https://www.youtube.com/watch?v=kwxCP_LLZJ4

Coding Challenge #21: Mandelbrot Set with p5.js:
https://www.youtube.com/watch?v=6z7GQewK-Ks

This equation will change how you see the world (the logistic map):
https://www.youtube.com/watch?v=ovJcsL7vyrk&t

Conjunto de Mandelbrot:
https://pt.wikipedia.org/wiki/Conjunto_de_Mandelbrot