UNIVERSIDADE FEDERAL DE SÃO CARLOS
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Arquiteturas de Alto Desempenho
Prática 4
Fractais com CPU e GPU no Matlab

Guilherme Gomes Arcencio - 769731 - Engenharia de Computação
Guilherme Silva Castro - 769763 - Ciência da Computação

1 Introdução

Neste trabalho, será explorada a geração e visualização de fractais utilizando a plataforma MATLAB, realizando uma análise comparativa de desempenho entre o processamento em CPU e a aceleração via GPU, explorando paralelismo. Fractais são objetos matemáticos de complexidade infinita e autossimilaridade e oferecem um excelente estudo de caso para explorar arquiteturas de processamento paralelo. O uso da computação em GPU permite uma aceleração drástica no tempo de renderização, aproveitando sua capacidade intrínseca de paralelismo massivo para calcular simultaneamente o estado de múltiplos pontos na imagem do fractal.

2 Desenvolvimento teórico

Conjunto de Mandelbrot é um conjunto de pontos no plano complexo (um plano onde os números têm uma parte real e uma parte imaginária).O princípio fundamental é testar cada ponto c em uma grade do plano complexo para ver se ele pertence ao conjunto. Como não podemos iterar infinitamente para provar que a sequência não diverge, usamos uma aproximação, definimos um número máximo de iterações(*maxiter*) e definimos um "raio de escape".

3 Desenvolvimento prático

Está matematicamente provado que, se o módulo (distância da origem) de qualquer Z_n na sequência exceder 2, a sequência inevitavelmente tenderá ao infinito. Portanto, para cada ponto c da nossa grade:

- Iniciamos a sequência com Z=0.
- Iteramos a equação Z=Z^2 +c até (*maxiter*) vezes.
- Em cada iteração, verificamos se Z>2.

Se isso acontecer, o ponto c "escapou"e não pertence ao conjunto. Paramos de iterar para esse ponto e registramos quantas iterações foram necessárias para escapar. Usaremos esse número para colorir o ponto.

Se o loop terminar (atingir *maxiter*) e Z nunca exceder 2, assumimos que o ponto c está dentro do conjunto de Mandelbrot.

```
1 function [mandelbrot_img, execution_time] =
     generate_mandelbrot()
2     % Inicia o cron metro para medir o tempo de execu   o
3     tic;
```

```
4
5       %% 1. Defini  o dos Par metros
6       % Par metros emp ricos sugeridos
7       grid_size = 1000;     % Resolu  o da imagem (1000x1000
            pixels)
8       max_iter = 500;       % N mero m ximo de itera  es por
            ponto
9
10      % Limites do plano complexo (regi o de visualiza  o)
11      % O conjunto de Mandelbrot est  contido principalmente
            entre -2 e 1 no eixo real
12      % e -1.5 e 1.5 no eixo imagin rio.
13      x_lim = [-2.0, 1.0];
14      y_lim = [-1.5, 1.5];
15
16      %% 2. Mapeamento do Plano Complexo para o Plano de Imagem
17      % Cria vetores para os eixos real (x) e imagin rio (y).
18      % linspace cria um vetor de 'grid_size' pontos igualmente
            espa ados.
19      x = linspace(x_lim(1), x_lim(2), grid_size);
20      y = linspace(y_lim(1), y_lim(2), grid_size);
21
22      % Inicializa a matriz que ir  conter a imagem final.
23      % Cada elemento guardar  o n mero de itera  es para o
            ponto correspondente escapar.
24      % Inicializamos com 'max_iter' para que os pontos que
            pertencem ao conjunto
25      % (que nunca escapam) fiquem com a cor m xima.
26      mandelbrot_matrix = max_iter * ones(grid_size, grid_size)
            ;
27
28      %% 3. Itera  o sobre cada Ponto (Pixel) da Grade
29      % Este    o n cleo do algoritmo. Vamos testar cada ponto
            'c'.
30      for row = 1:grid_size
31          for col = 1:grid_size
32              % Pega o valor do eixo imagin rio (y) e real (x)
                    para este pixel.
33              % Note que 'row' corresponde ao eixo Y e 'col' ao
                    eixo X.
34              % Em MATLAB, o eixo Y    invertido em matrizes,
```

```
                ent o usamos y(grid_size - row + 1)
35              % para a orienta  o correta da imagem, ou
                    simplesmente transpomos no final.
36              % Vamos fazer da forma mais simples:
37              c_imag = y(row);
38              c_real = x(col);
39              c = c_real + 1i * c_imag; % Forma o n mero
                    complexo 'c'
40
41              % Inicializa a sequ ncia de Mandelbrot para este
                     'c'
42              z = 0; % Z0 = 0    a defini  o padr o do
                    conjunto de Mandelbrot
43
44              % Loop de itera  o para a sequ ncia Z(n+1) = Z
                    (n)^2 + c
45              for n = 1:max_iter
46                  % Aplica a f rmula
47                  z = z^2 + c;
48
49                  % Verifica a condi  o de escape
50                  if abs(z) > 2
51                      % O ponto escapou! Armazena o n mero de
                            itera  es 'n'
52                      mandelbrot_matrix(row, col) = n;
53                      % Interrompe o loop interno, pois n o
                            precisamos mais testar este ponto
54                      break;
55                  end
56              end
57          end
58          % Opcional: Mostrar progresso na janela de comando
59          % if mod(row, 50) == 0
60          %     fprintf('Processando linha %d de %d...\n', row,
                grid_size);
61          % end
62      end
63
64      %% 4. Medi  o do Tempo e Gera  o da Imagem
65      execution_time = toc; % Para o cron metro e armazena o
            tempo
```

```
66      fprintf('Tempo de execu  o (CPU Serial): %.4f segundos
            .\n', execution_time);
67
68      % Cria o objeto da imagem
69      mandelbrot_img = mandelbrot_matrix;
70
71      % Exibe a imagem resultante
72      figure; % Cria uma nova janela de figura
73      imagesc(x, y, mandelbrot_img); % Usa imagesc para mapear
            valores para cores
74      colormap(jet); % Aplica um mapa de cores (experimente '
            hot', 'parula', 'turbo')
75      colorbar; % Mostra a barra de cores
76      axis equal; % Garante que a propor  o da imagem n o
            seja distorcida
77      axis tight;
78      title(sprintf('Conjunto de Mandelbrot (%d itera  es)',
            max_iter));
79      xlabel('Parte Real');
80      ylabel('Parte Imagin ria');
81  end
```

Listing 1: Gerando Maldelbrot via CPU

Com isso temos a visualização da imagem abaixo.

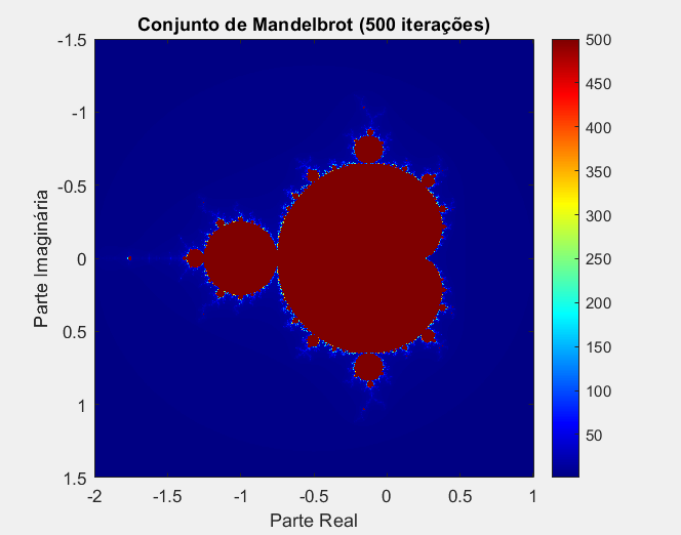

Figura 1: Mandelbrot CPU.

Para a versão com GPU vamos utilizar a classe *gpuarray* O processo será:

- Criar as matrizes de dados na CPU como antes.

- Transferi-las para a memória da GPU usando a função gpuArray.
- Executar o loop de iterações, onde todas as operações ($.^2$, +, abs) são realizadas em paralelo na GPU.
- Usar uma "máscara"lógica para identificar e atualizar apenas os pontos que escapam a cada iteração.
- Ao final, trazer a matriz resultante de volta para a memória principal da CPU usando a função gather() para poder visualizá-la.

```
1
2  function [mandelbrot_img_gpu, execution_time_gpu] =
       generate_mandelbrot_gpu()
3      % Verifica se uma GPU compat vel est  dispon vel
4      if ~canUseGPU
5          error('GPU n o encontrada ou n o suportada.
               Verifique sua instala  o do Parallel Computing
               Toolbox.');
6      end
7
8      fprintf('Iniciando c lculo na GPU...\n');
9
10     % Inicia o cron metro para medir o tempo de execu  o
11     tic;
12
13     %% 1. Defini  o dos Par metros (os mesmos da vers o
           CPU para uma compara  o justa)
14     grid_size = 1000;
15     max_iter = 500;
16     x_lim = [-2.0, 1.0];
17     y_lim = [-1.5, 1.5];
18
19     %% 2. Mapeamento do Plano Complexo (ainda na CPU)
20     x = linspace(x_lim(1), x_lim(2), grid_size);
21     y = linspace(y_lim(1), y_lim(2), grid_size);
22     [X, Y] = meshgrid(x, y);
23
24     % A matriz de pontos 'c'    criada na CPU.
25     % Note que Y    transposto para corresponder
           orienta  o da imagem
26     C = X + 1i * Y';
27
```

```
%% 3. Transfer ncia de Dados para a GPU
% Usamos gpuArray para mover as matrizes para a mem ria da GPU.
% A partir daqui, todas as opera  es nessas matrizes ocorrer o na GPU.
C_gpu = gpuArray(C);
Z_gpu = gpuArray(zeros(grid_size, grid_size));
mandelbrot_gpu = gpuArray(max_iter * ones(grid_size, grid_size));

%% 4. Loop de Itera  o Paralelo na GPU
% O loop agora itera de 'n=1' a 'max_iter', e dentro do loop,
% a opera  o  aplicada a TODA a matriz de uma s  vez.
for n = 1:max_iter
    % Z = Z^2 + C, executado para todos os 1 milh o de pontos em paralelo na GPU
    Z_gpu = Z_gpu.^2 + C_gpu;

    % Cria uma m scara l gica para encontrar os pontos que escaparam NESTA itera  o
    % Condi  es:
    % 1. O m dulo de Z  > 2
    % 2. O ponto ainda n o foi marcado como "escapado" (seu valor ainda  max_iter)
    mask = abs(Z_gpu) > 2 & mandelbrot_gpu == max_iter;

    % Usa a m scara para atualizar APENAS os pontos que acabaram de escapar,
    % marcando-os com a itera  o atual 'n'.
    mandelbrot_gpu(mask) = n;
end

%% 5. Recupera  o dos Dados da GPU
% Para visualizar a imagem, precisamos trazer os resultados de volta
% para a mem ria principal da CPU com a fun  o gather().
mandelbrot_img_gpu = gather(mandelbrot_gpu);

```

```
58      execution_time_gpu = toc; % Para o cron metro
59      fprintf('Tempo de execu  o (GPU Paralelo): %.4f
            segundos.\n', execution_time_gpu);
60
61      % A parte de visualiza  o     a mesma da vers o CPU
62      figure;
63      imagesc(x, y, mandelbrot_img_gpu);
64      colormap(jet);
65      colorbar;
66      axis equal;
67      axis tight;
68      title(sprintf('Conjunto de Mandelbrot (GPU, %d
            itera  es)', max_iter));
69      xlabel('Parte Real');
70      ylabel('Parte Imagin ria');
71  end
```

Listing 2: Gerando Maldelbrot via GPU

Com isso temos a visualização da imagem abaixo.

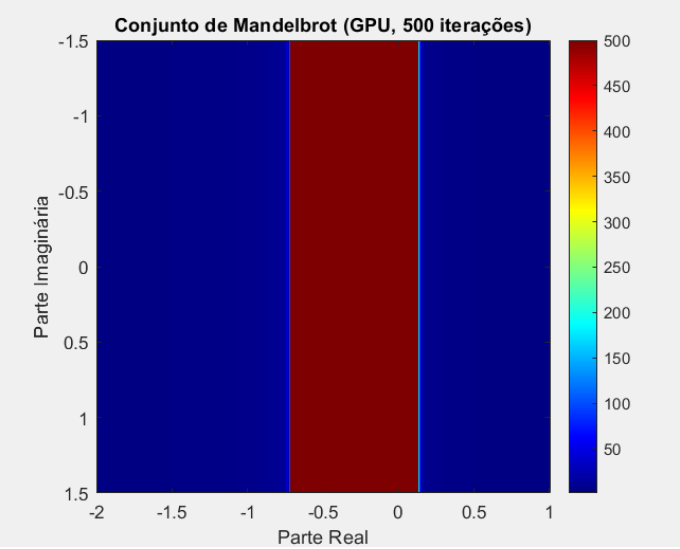

Figura 2: Mandelbrot GPU.

4 Apresentação dos resultados

Após as execuções das versões podemos observar a seguinte comparação.

```
--- Iniciando Versão Serial (CPU) ---
Tempo de execução (CPU Serial): 1.4186 segundos.
Tempo final CPU: 1.4186 segundos
----------------------------------------
--- Iniciando Versão Paralela (GPU) ---
Iniciando cálculo na GPU...
Tempo de execução (GPU Paralelo): 0.6106 segundos.
Tempo final GPU: 0.6106 segundos
========================================
      RESULTADO DA COMPARAÇÃO
========================================
Tempo de execução da CPU: 1.4186 s
Tempo de execução da GPU: 0.6106 s

=> A versão em GPU foi 2.32 vezes mais rápida que a versão em CPU.
fx >>
```

Figura 3: Comparativo.

5 Conclusões

A implementação do algoritmo de geração do Conjunto de Mandelbrot foi realizada e comparada em duas arquiteturas distintas: CPU (processamento serial) e GPU (processamento paralelo).A análise comparativa desses tempos demonstra claramente a superioridade da abordagem paralela em GPU para esta tarefa computacionalmente intensiva. A versão implementada na GPU foi aproximadamente 2.32 vezes mais rápida do que a versão executada na CPU.

Este speedup significativo (fator de 2.32) é um indicativo robusto da eficácia do processamento paralelo em GPUs para problemas que, como a geração do Conjunto de Mandelbrot, envolvem um grande número de cálculos independentes. A natureza intrínseca do algoritmo de Mandelbrot, onde a convergência de cada ponto no plano complexo pode ser determinada independentemente dos outros, alinha-se perfeitamente com a arquitetura massivamente paralela das GPUs, que são otimizadas para executar milhares de operações simultaneamente.

Em suma, os resultados confirmam que a utilização de GPUs proporciona uma aceleração substancial para a geração do Conjunto de Mandelbrot, validando a abordagem de computação paralela como uma estratégia eficiente para otimizar o desempenho em cenários que demandam alto poder de processamento.