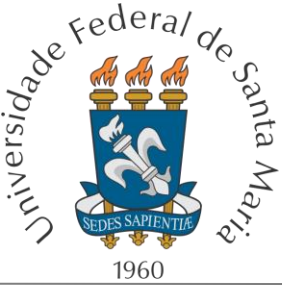
ELC129 - Programação Paralela



Trabalho 4: Geração de Fractais de Mandelbrot em OpenMP

MARLON LEONER DA SILVA RODRIGUES

Análise do Código

Analisando o código verifica-se a existência de três estruturas de repetição do tipo *for*, ou seja, três possibilidades de paralelização. São elas:

- Geração dos frames;
- Cálculo das linhas;
- Cálculo das colunas.

```
// compute frames
for (int frame = 0; frame < frames; frame++) {
  const double xMin = xMid - delta;
  const double yMin = yMid - delta;
  const double dw = 2.0 * delta / width;
```

```
for (int row = 0; row < width; row++) {
  const double cy = yMin + row * dw;
  for (int col = 0; col < width; col++) {
    const double cx = xMin + col * dw;
```

Alteração no Código

A única alteração de lógica que foi realizada em relação ao código original foi no cálculo da variável *delta*. No código original existe dependência na variável *delta*, com isso, a paralelização pode ser prejudicada. Para que a geração de frames pudesse ser paralelizada, foi alterada o forma de calcular o *delta*.

- Código original: *delta* é alterado ao final do *loop* mais externo.

```
for (int frame = 0; frame < frames; frame++) {
  const double xMin = xMid - delta;
  const double yMin = yMid - delta;
  const double dw = 2.0 * delta / width;
  for (int row = 0; row < width; row++) {···
  }
  delta *= 0.98;
}
```

- Código alterado: *delta* é calculado dependendo do frame que está sendo gerado.

```
for(int frame = 0 ; frame < frames ; frame++) {
   double delta = Delta * pow(0.98, frame);
   const double xMin = xMid - delta;
   const double yMin = yMid - delta;
   const double dw = 2.0 * delta / width;
   for (int row = 0; row < width; row++) {···
   }
}
```

Solução 1

Para a primeira solução foi utilizado o escalonamento dinâmico, com o trabalho sendo dividido em espaços de trabalho definido pela variável *chunk*. O valor da *chunk* é definido pela divisão do número de *frames* pela quantidade de *threads*, ou seja:

- chunk = frames / threads

```
#pragma omp parallel shared(pic, width) num_threads(threads)
{
    #pragma omp for schedule(dynamic, chunk)
    for(int frame = 0 ; frame < frames ; frame++) {
        double delta = Delta * pow(0.98, frame);
        const double xMin = xMid - delta;
        const double yMin = yMid - delta;
        const double dw = 2.0 * delta / width;
        for (int row = 0; row < width; row++) {···
        }
    }
}
```

Figura X: Código paralelizado na solução 1

Solução 2

Para a segunda solução foi utilizado o escalonamento estático, sem a definição do valor de *chunk*, sendo assim, as tarefas são divididas igualmente entre o número total de *threads*.

```
#pragma omp parallel shared(pic) num_threads(threads)
{
    #pragma omp for schedule(static)
    for(int frame = 0 ; frame < frames ; frame++) {
        const double delta = Delta * pow(0.98, frame);
        const double xMin = xMid - delta;
        const double yMin = yMid - delta;
        const double dw = 2.0 * delta / width;
        for (int row = 0; row < width; row++) {···
        }
    }
}
```

Figura X: Código paralelizado na solução 2

Metodologia

- Para os testes foram utilizados os seguintes valores:
  - Número de Threads: 2, 4 e 8;
  - Largura dos Frames: 256, 512 e 1024;
  - Quantidade de Frames: 32, 64 e 128;
  - Número de execuções: 10

Resultados Obtidos

No quadro abaixo, temos os valores obtidos com a execução do programa sem paralelização, ou seja, uma execução sequencial.

Frames	256	512	1024
32	4.95	19.78	79.13
64	9.47	37.92	151.48
128	20.81	83.18	332.71

Tabela 1: valores (em segundos) obtidos na execução do programa sequencial.

Resultados Obtidos - Solução 1

Na tabela abaixo, temos os tempos de execução da solução 1 com o valor de 1024 para a largura dos *frames*.

Frames	2Threads	4Threads	8Threads
32	41.69	23.89	23.27
64	79.19	47.16	44.99
128	181.35	110.69	102.4

Tabela 2: valores (em segundos) obtidos na execução da solução 1.

Resultados Obtidos - Solução 1

Na tabela abaixo, temos os valores de *speedup* e eficiência, calculados com base nos dados representados na tabela 2.

Frames	2Threads	4Threads	8Threads
32	1.90 / 95%	3.31 / 83%	3.40 / 43%
64	1.91 / 96%	3.21 / 80%	3.37 / 42%
128	1.83 / 92%	3.01 / 75%	3.25 / 41%

Tabela 3: valores de *speedup* e eficiência obtidos na execução da solução 1.

Resultados Obtidos - Solução 2

Na tabela abaixo, temos os tempos de execução da solução 2 com o valor de 1024 para a largura dos *frames*.

Frames	2Threads	4Threads	8Threads
32	41.7	24.24	23.26
64	79.17	46.04	44.71
128	181.26	111.02	102.21

Tabela 4: valores (em segundos) obtidos na execução da solução 2.

Resultados Obtidos - Solução 2

Na tabela abaixo, temos os valores de *speedup* e eficiência, calculados com base nos dados representados na tabela 2.

Frames	2Threads	4Threads	8Threads
32	1.90 / 95%	3.26 / 82%	3.40 / 43%
64	1.91 / 96%	3.29 / 82%	3.39 / 42%
128	1.84 / 92%	3.00 / 75%	3.26 / 41%

Tabela 5: valores de *speedup* e eficiência obtidos na execução da solução 2.

Programação Paralela

Trabalho 4: Geração de Fractais de Mandelbrot
em OpenMP

Marlon Leoner da Silva Rodrigues

Santa Maria

23 de Abril de 2019