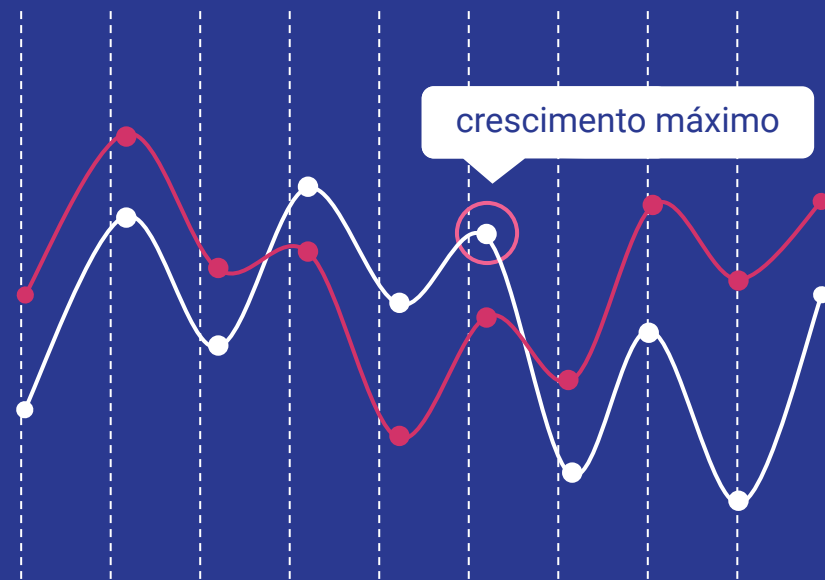


# T4: Geração de Fractais de Mandelbrot em OpenMP

Matheus Grisotti



# Oportunidades de Paralelismo

Identificação e Análise

# Existem 3 possíveis loops

## Loop de 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++) { ...  
    }  
    delta *= 0.98;  
}
```

# Existem 3 possíveis loops

## Loop de Linhas

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

# Existem 3 possíveis loops

## Loop de Colunas

```
for (int col = 0; col < width; col++) {  
    const double cx = xMin + col * dw;  
    double x = cx;  
    double y = cy;  
    int depth = 256;  
    double x2, y2;  
    do { ...  
    } while ((depth > 0) && ((x2 + y2) < 5.0));  
    pic[frame * width * width + row * width + col] = (unsigned char)depth;  
}
```

# Análise dos Loops

## Loop de Frames

### **Dependência**

A variável Delta é diferente em cada iteração do loop, logo ela deve ser incrementada de forma sequencial. Ou implementada de outra maneira.

## Loop de Linhas

### **Sem dependência**

É possível paralelizar sem problemas.

## Loop de Colunas

### **Sem dependência**

É possível paralelizar sem problemas.

# Solução

Loop de Linhas

Usar o Loop de Linhas é melhor que o de colunas pois o cada thread é associada menos vezes a um intervalo de iteração do FOR.

---

# Implementação



# Parallel FOR Schedule - Auto e Dynamic

Loop de Linhas (Programa 1 e 2 Respectivamente)

```
#pragma omp parallel for schedule(auto)
for (int row = 0; row < width; row++) {
    const double cy = yMin + row * dw;
    for (int col = 0; col < width; col++) { ...
    }
}
```

```
#pragma omp parallel for schedule(Dynamic)
for (int row = 0; row < width; row++) {
    const double cy = yMin + row * dw;
    for (int col = 0; col < width; col++) { ...
    }
}
```

# Resultados

# Resultados Programa 1

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	1024	64	53,16	1,00
2	1024	64	27,77	1,91
4	1024	64	14,71	3,61
8	1024	64	9,51	5,59

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	1024	32	27,60	1,00
2	1024	32	14,46	1,91
4	1024	32	7,65	3,61
8	1024	32	5,03	5,49

64 e 32 Frames de 1024 Pixels



# Resultados Programa 1

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	512	64	13,23	1,00
2	512	64	6,95	1,90
4	512	64	3,67	3,60
8	512	64	2,40	5,51

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	512	32	6,90	1,00
2	512	32	3,61	1,91
4	512	32	1,91	3,61
8	512	32	1,27	5,43

64 e 32 Frames de 512 Pixels



# Resultados Programa 2

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	1024	64	52,94	1,00
2	1024	64	27,15	1,95
4	1024	64	13,67	3,87
8	1024	64	8,50	6,23

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	1024	32	27,62	1,00
2	1024	32	14,16	1,95
4	1024	32	7,14	3,87
8	1024	32	4,43	6,23

64 e 32 Frames de 1024 Pixels



# Resultados Programa 2

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	512	64	13,22	1,00
2	512	64	6,78	1,95
4	512	64	3,42	3,87
8	512	64	2,13	6,21

Threads	Tamanho	Frames	Tempo(Segundos)	Speed Up
1	512	32	6,91	1,00
2	512	32	3,54	1,95
4	512	32	1,78	3,88
8	512	32	1,11	6,23

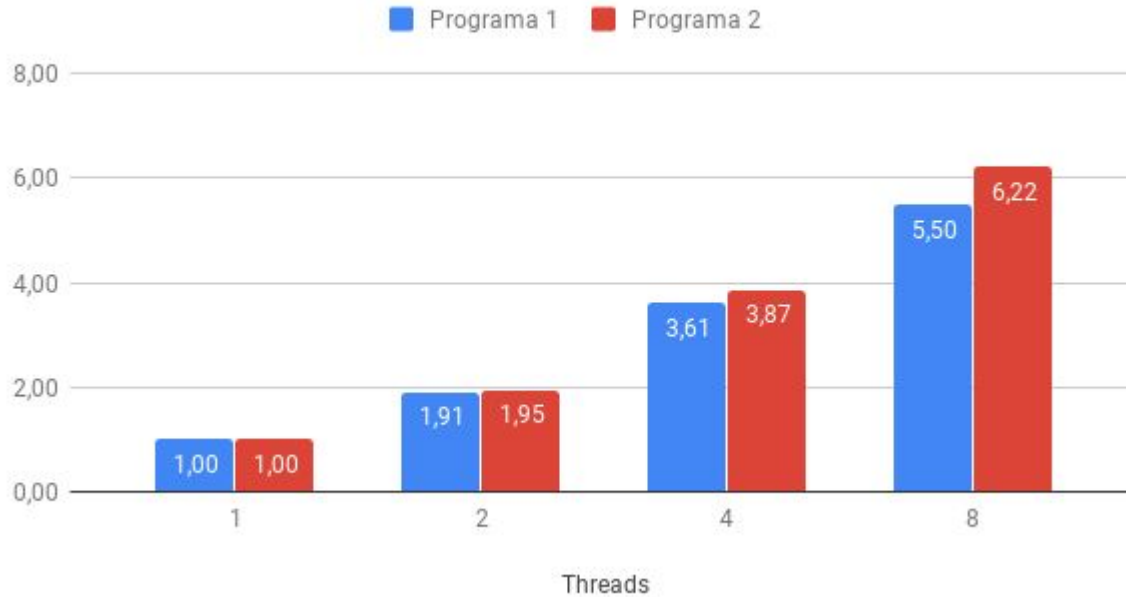
64 e 32 Frames de 512 Pixels



# Comparativo Programa 1 e 2

# Speed Up Programa 1 e 2

Speed Up Programa 1 e 2





# Speed Up Ratio

Conforme o número de Threads (Eixo X) aumenta, o desempenho do programa 2 em relação ao programa 1 aumenta linearmente.

