

Geração de Fractais de Mandelbrot em OpenMP

**Identificando
oportunidades de
paralelização**

```
// compute frames
double delta = Delta;
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;
            double x = cx;
            double y = cy;
            int depth = 256;
            double x2, y2;
            do {
                x2 = x * x;
                y2 = y * y;
                y = 2 * x * y + cy;
                x = x2 - y2 + cx;
                depth--;
            } while ((depth > 0) && ((x2 + y2) < 5.0));
            pic[frame * width * width + row * width + col] = (unsigned char)depth;
        }
    }
    delta *= 0.98;
}
```

```
// compute frames
double delta = Delta;
for (int frame = 0; frame < frames; frame++) {
    const double xMin = xMid - delta;
    const double yMin = yMid - delta;
    const double dw = 2.0 * delta;
    for (int row = 0; row < height; row++) {
        const double cy = yMin + row * dw;
        for (int col = 0; col < width; col++) {
            const double cx = xMin + col * dw;
            double x = cx;
            double y = cy;
            int depth = 255;
            double x2, y2;
            do {
                x2 = x * x;
                y2 = y * y;
                y = 2 * x * y + y2;
                x = x2 - y2 + cx;
                depth--;
            } while ((depth > 0) && (x2 > 1.0 || y2 > 1.0));
            pic[frame * width * width + row * width + col] = (unsigned char)depth;
        }
    }
    delta *= 0.98;
}
```



```
// compute frames
double delta = Delta;
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;
            double x = cx;
            double y = cy;
            int depth = 256;
            double x2, y2;
            do {
                x2 = x * x;
                y2 = y * y;
                y = 2 * x * y + cy;
                x = x2 - y2 + cx;
                depth--;
            } while ((depth > 0) && ((x2 + y2) < 5.0));
            pic[frame * width * width + row * width + col] = (unsigned char)depth;
        }
    }
    delta *= 0.98;
}
```




Caso 1

Utilizando schedule auto

**Compilador escolhe
como mapear iterações
para threads**

```
// compute frames
double delta = Delta;
for (int frame = 0; frame < frames; frame++) {
    const double xMin = xMid - delta;
    const double yMin = yMid - delta;
    const double dw = 2.0 * delta / width;
#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++) {
            const double cx = xMin + col * dw;
            double x = cx;
            double y = cy;
            int depth = 256;
            double x2, y2;
            do {
                x2 = x * x;
                y2 = y * y;
                y = 2 * x * y + cy;
                x = x2 - y2 + cx;
                depth--;
            } while ((depth > 0) && ((x2 + y2) < 5.0));
            pic[frame * width * width + row * width + col] = (unsigned char)depth;
        }
    }
}
```

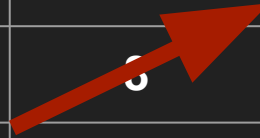


Casos base

Tamanho	Frames	Tempo
1024	32	43.90s
1024	64	84.14s
2048	32	175.45s
1024	64	48.37s
1024	64	32.16s
1024	64	31.82s
2048	32	101.94s
2048	32	66.2s
2048	32	67.61s

Tamanho	Frames	Tempo	Threads	Speedup
1024	32	25s	2	1.75
1024	32	15.65s	4	2.80
1024	32	15.98s	8	2.74
1024	64	48.37s	2	1.74
1024	64	32.16s	4	2.61
1024	64	31.82s	8	2.64
2048	32	101.94s	2	1.72
2048	32	66.2s	4	2.65
2048	32	67.61s	8	2.59

Tamanho	Frames	Tempo	Threads	Speedup
1024	32	25s	2	1.75
1024	32	15.65s	4	2.80
Speedups consideráveis!	32	5.98s	8	2.74
		8.37s	2	1.74
				2.61
				2.64
2048	32	101.94s	2	1.72
2048	32	66.2s	4	2.65
2048	32	67.61s	8	2.59




Caso 2

Utilizando schedule dynamic

- **O worksize é dividido em partes iguais**
- **Cada thread escolhe um chunk**
- **Após terminar o trabalho, escolhe o próximo**
- **Até que não existam mais chunks**

```
// compute frames
double delta = Delta;
for (int frame = 0; frame < frames; frame++) {
    const double xMin = xMid - delta;
    const double yMin = yMid - delta;
    const double dw = 2.0 * delta / width;
#pragma omp parallel for schedule(dynamic) num_threads(num_threads)
    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;
            double x = cx;
            double y = cy;
            int depth = 256;
            double x2, y2;
            do {
                x2 = x * x;
                y2 = y * y;
                y = 2 * x * y + cy;
                x = x2 - y2 + cx;
                depth--;
            } while ((depth > 0) && ((x2 + y2) < 5.0));
            pic[frame * width * width + row * width + col] = (unsigned char)depth;
        }
    }
    delta *= 0.98;
}
```



Tamanho	Frames	Tempo	Threads	Speedup
1024	32	25s	2	1.83
1024	32	15.65s	4	2.96
			8	2.90
				1.76
				2.83
1024	64	31.82s	8	2.76
2048	32	101.94s	2	1.74
2048	32	66.2s	4	2.74
2048	32	67.61s	8	2.68

**Speedups maiores !
(em relação ao caso 1)**