Trabalho 4 - elc139

Bruno da Silva Alves

bdalves@inf.ufsm.br

```
double delta = Delta;
for (int frame = 0; frame < frames; frame++)
{
    // ...
    for (int row = 0; row < width; row++) {
        for (int col = 0; col < width; col++)
{
            // ...
            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;
    }
```

Problema iterativo em que o valor corrente depende do valor imediatamente anterior não é possível paralelizar a computação.

```
double delta = Delta;
for (int frame = 0; frame < frames; frame++) {
    // ...
    for (int row = 0; row < width; row++) {
        for (int col = 0; col < width; col++) {
            // ...
            do {
                // ...
            } while ((depth > 0) && ((x2 + y2) <
5.0));
            pic[...] = (unsigned char)depth;
        }
    }
    delta *= 0.98;
    }
```

Dependência de dados

Solução:

```
for (int frame = 0; frame < frames; frame++) {
    delta = Delta * pow(0.98, frame);
}
```

Solução 1:

```
#pragma omp parallel for schedule (static, frames/num_threads)
for (int frame = 0; frame < frames; frame++) {

        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++) {
                const double cy = yMin + row * dw;
                for (int col = 0; col < width; col++) {
                        // ...
                        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[] = (unsigned char)depth;
                }
        }
}
```

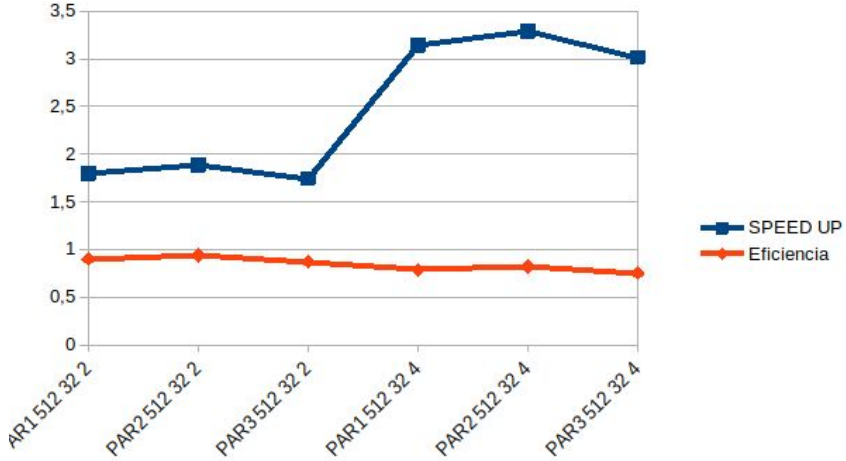
Solução 2:

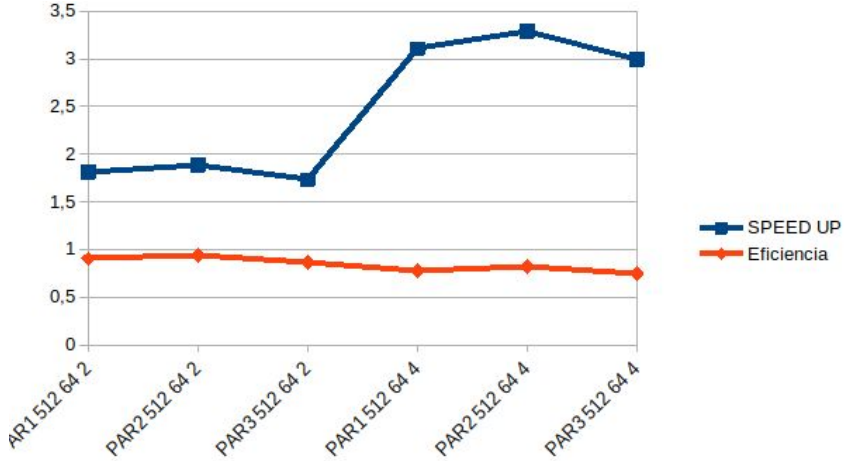
```
for (int frame = 0; frame < frames; frame++) {

        delta = Delta * pow(0.98, frame);
        const double xMin = xMid - delta;
        const double yMin = yMid - delta;
        const double dw = 2.0 * delta / width;

        #pragma omp parallel for schedule (dynamic, 1)
        for (int row = 0; row < width; row++) {
                const double cy = yMin + row * dw;
                for (int col = 0; col < width; col++) {
                     // ...
                        do {
                            //...
                        } while ((depth > 0) && ((x2 + y2) < 5.0));
                        pic[] = (unsigned char)depth;
                }
        }

}
```

Solução 3:

```
#pragma omp parallel for schedule (dynamic, width)
for (int i = 0; i < frames * width * width; i++) {

        int frame = i / width2;
        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++) {
        int row = (i / width) % width;

        const double cy = yMin + row * dw;

        // for (int col = 0; col < width; col++) {
        int col = i % width;

        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[] = (unsigned char)depth;


}
```

Resultados obtidos:





Resultados obtidos: