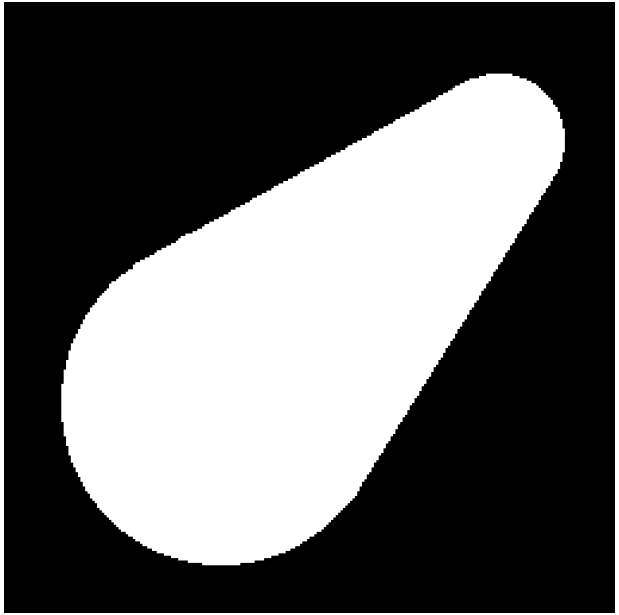
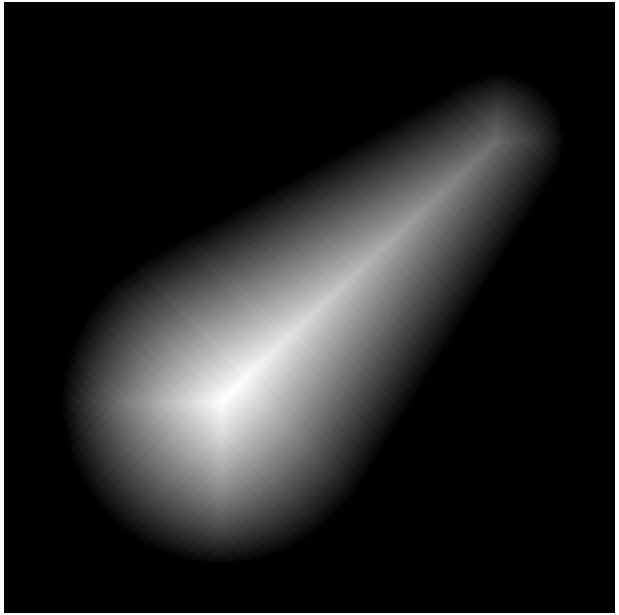
Computação Paralela

Trabalho-OpenMp: paradigma de memória partilhada

Márcio Rocha PG41086
Tiago Pereira A61032

Transformada da distância





Algoritmo

```
for (int row = 0; row < rows; row++) {
    for (int col = 0; col < columns; col++) {
        pInt = pixelData[col + row * columns];
        //se for branco
        if(pInt == MAX_VALUE){
            hadwhite = 1;
            int minNei = MAX_VALUE;
            //percorre os vizinhos
            for (int r = row-1; r <= row+1; r++) {
                for (int c = col-1; c <= col+1; c++) {
                    //se nao for o pixel onde estou ou existir no array
                    if((r != row && c != col) || (r <= -1 || r >= rows + 1 || c <= -1 || c >= columns + 1)){
                        int neiPixel = pixelData[c + r * columns];
                        if(neiPixel < minNei)
                            minNei = neiPixel;
                    }
                }
            }
            if(minNei < MAX_VALUE){
                mpixelData[col + row * columns] = minNei + 1;
                if(max_gray < minNei + 1)
                    max_gray = minNei + 1;
            }
            else
            {
                mpixelData[col + row * columns] = pInt;
            }
        }
        else
        {
            mpixelData[col + row * columns] = pInt;
        }
    }
}
cupthree = pixelData;
pixelData = mpixelData;
mpixelData = cupthree;
```

Algoritmo

```
//percorre os vizinhos
for (int r = row-1; r <= row+1; r++) {
    for (int c = col-1; c <= col+1; c++) {
        //se nao for o pixel onde estou ou existir no array
        if((r != row && c != col) || (r <= -1 || r >= rows + 1 || c <= -1 || c >= columns + 1)){
            int neiPixel = pixelData[c + r * columns];
            if(neiPixel < minNei)
                minNei = neiPixel;
        }
    }
}
```

Algoritmo

```
if(minNei < MAX_VALUE){
    mpixelData[col + row * columns] = minNei + 1;
    if(max_gray < minNei + 1)
        max_gray = minNei + 1;
}
else
{
    mpixelData[col + row * columns] = pInt;
}
```

Paralelização

#pragma_omp_parallel for num_threads(MAX_THREADS)

Máquina usada:

- Ivy Bridge r652 dual processor: 20 cores físicos, 40 lógicos

Testes e métricas

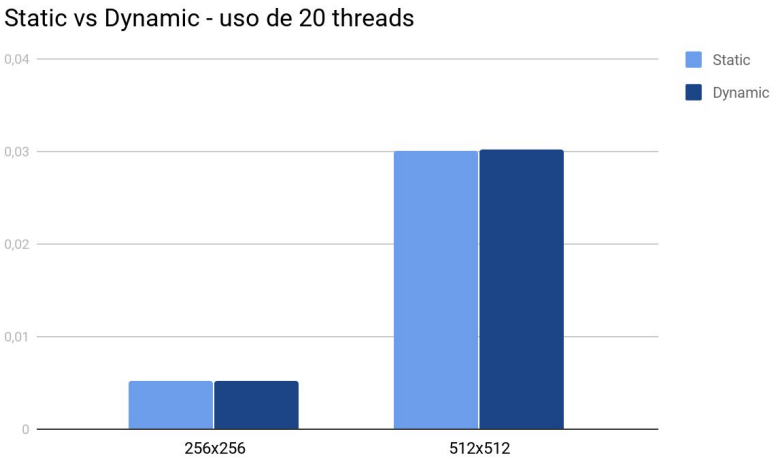
Testes:

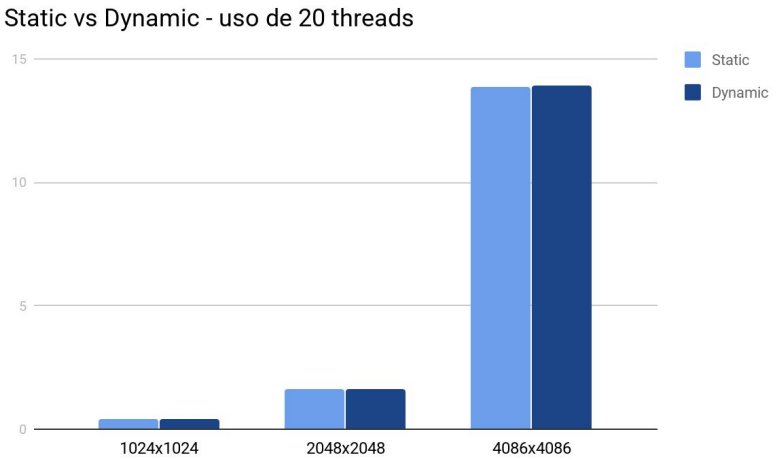
- static
- dynamic
- guided

Métricas:

Imagem de vários tamanhos

Testes-Static vs Dynamic





Testes-(Dynamic,k)

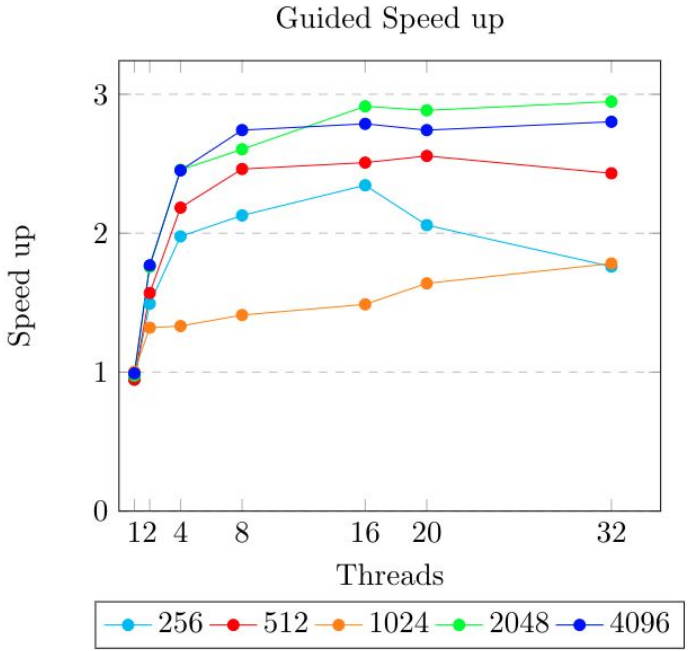
	k = 1	k = 2	k = 4	k = 8	k = 16	k = 32
256	0.00452	0.00454	0.00455	0.00458	0.00456	0.00454
512	0.02782	0.02753	0.02725	0.02715	0.02736	0.02797
1024	0.39457	0.40699	0.38428	0.40631	0.38689	0.39840
2048	1.57927	1.61117	1.58781	1.59674	1.58099	1.59339
4096	13.18661	13.379499	13.42573	13.49609	13.45234	13.27744

Resultados-Guided

Tempo de execução:

	sequencial	2 threads	4 threads	8 threads	16 threads	20 threads	32 threads
256	0.0090	0.0057	0.0043	0.0040	0.0042	0.0041	0.0048
512	0.0692	0.0417	0.0300	0.0266	0.0261	0.0256	0.0269
1024	0.5814	0.4425	0.4386	0.4141	0.3926	0.3563	0.3278
2048	4.6496	2.5786	1.8458	1.7414	1.5564	1.5721	1.5387
4096	36.3160	20.3411	14.6761	13.1271	12.9125	13.1232	12.8456

Resultados-Escalabilidade



Conclusão

Em suma, podemos concluir que o *bottleneck* deste algoritmo é *memory bound.*