ELC129 - Programação Paralela

T8: Geração de Imagem em Paralelo com CUDA

MARLON LEONER DA SILVA RODRIGUES

Parte 1

- Cada *frame* deve ser computado por uma *thread* diferente.

```
__global__ void fPixelGenerator(int width, unsigned char* pic) {

    int frame = threadIdx.x; // ID da thread

    for (int row = 0; row < width; row++) {
        for (int col = 0; col < width; col++) {
            float fx = col - 1024/2;
            float fy = row - 1024/2;
            float d = sqrtf( fx * fx + fy * fy );
            unsigned char color = (unsigned char) (160.0f + 127.0f *
                                                   cos(d/10.0f - frame/7.0f) /
                                                   (d/50.0f + 1.0f));

            pic[frame * width * width + row * width + col] = (unsigned char) color;
        }
    }
}
```

Parte 1

- Chamada da função, com número de *threads* igual ao número de *frames*.

```
// allocate picture array
unsigned char* pic = NULL;
cudaMallocManaged(&pic, frames * width * width * sizeof(unsigned char));

// start time
timeval start, end;
gettimeofday(&start, NULL);

// Pixel Generator
fPixelGenerator<<<1, frames>>>(width, pic);

// Wait for GPU to finish before accessing on host
cudaDeviceSynchronize();

// end time
gettimeofday(&end, NULL);
double runtime = end.tv_sec + end.tv_usec / 1000000.0 - start.tv_sec - st
printf("compute time: %.4f s\n", runtime);
```

Parte 2

- Implemente um programa que aproveite melhor o paralelismo da GPU.

```
int blockID = blockIdx.x;
int tBlocks = gridDim.x;

int fromBlocks = (blockID)     * (frames / tBlocks);
int toBlocks   = (blockID + 1) * (frames / tBlocks);

int threadID = threadIdx.x;
int tThreads = blockDim.x;

int fromThreads = (threadID)     * (width / tThreads);
int toThreads   = (threadID + 1) * (width / tThreads);
```

Parte 2

```
for (int frame = fromBlocks; frame < toBlocks; frame++) {
    for (int row = fromThreads; row < toThreads; row++) {
        for (int col = 0; col < width; col++) {
            float fx = col - 1024/2;
            float fy = row - 1024/2;
            float d = sqrtf( fx * fx + fy * fy );
            unsigned char color = (unsigned char) (160.0f + 127.0f *
                                                   cos(d/10.0f - frame/7.0f) /
                                                   (d/50.0f + 1.0f));

            pic[frame * width * width + row * width + col] = (unsigned char) color;
        }
    }
}
```

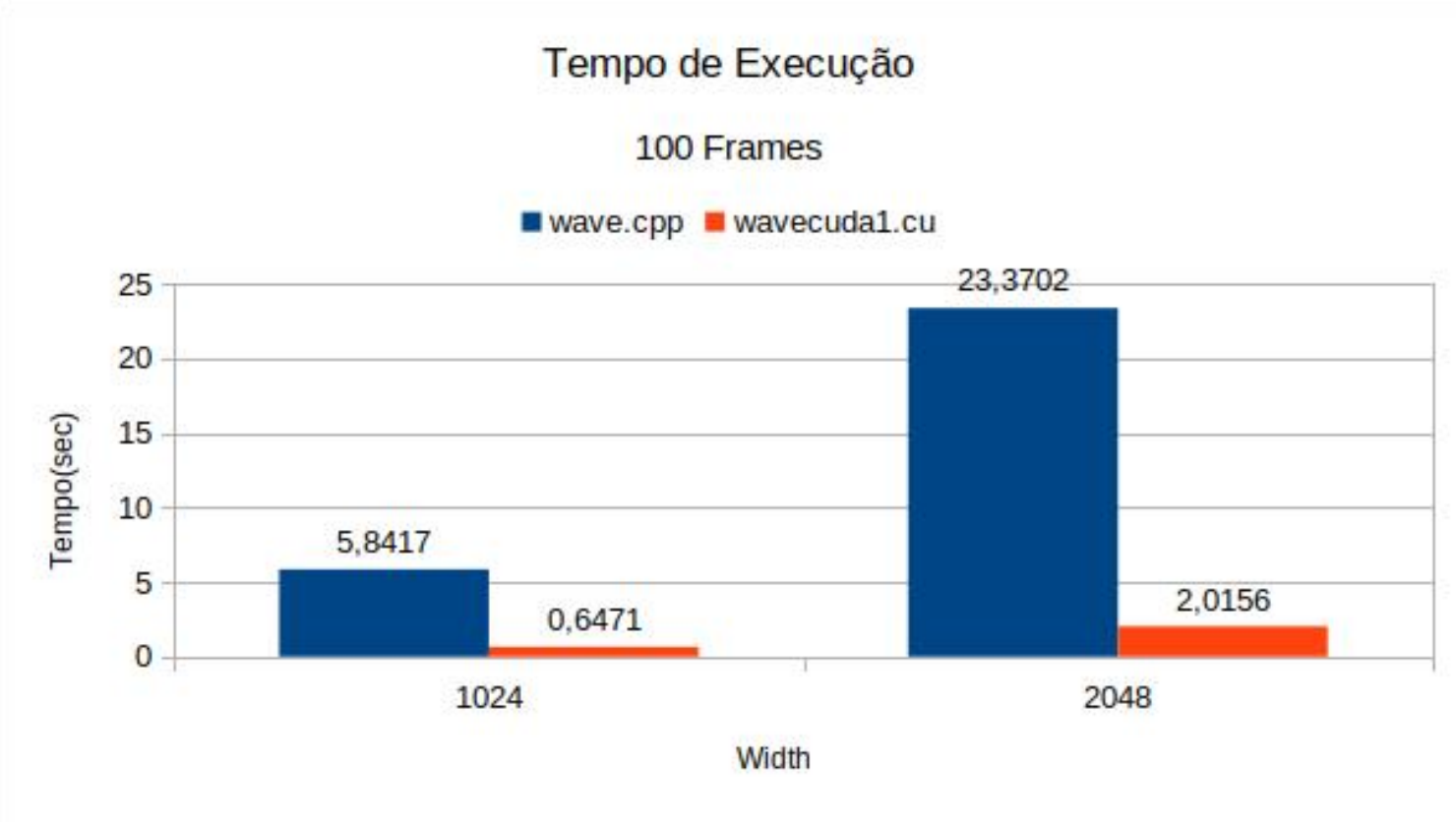
Metodologia

- Para os testes foram utilizados os seguintes valores:
  - Largura dos *frames*: 1024 e 2048;
  - Quantidade de *frames*: 100 e 200;
  - Número de execuções: 10.

- Os códigos foram execução na plataforma na nuvem Google Colaboratory.

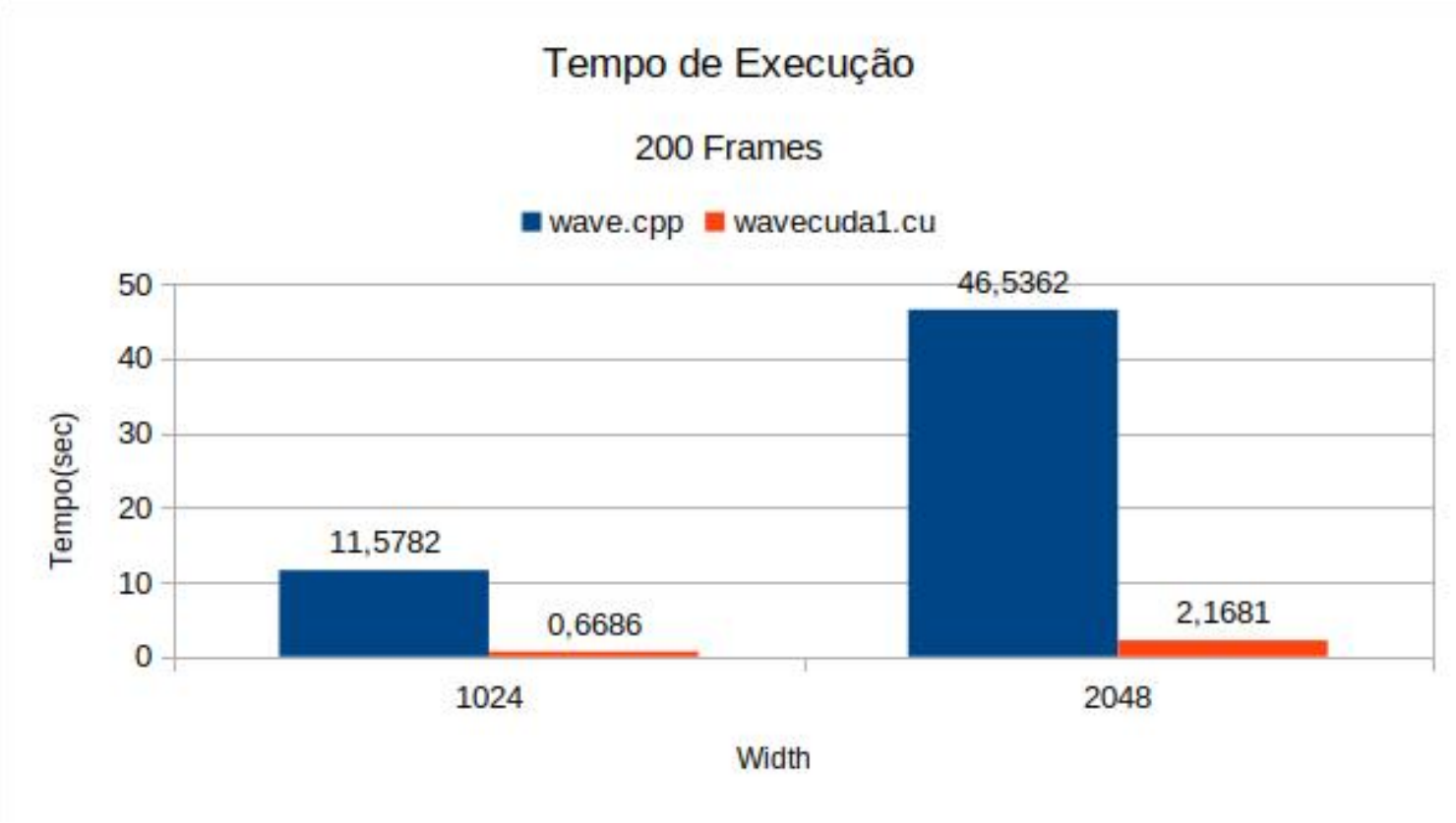
Resultados - Parte 1

- Wave.cpp x Wavecuda1.cu



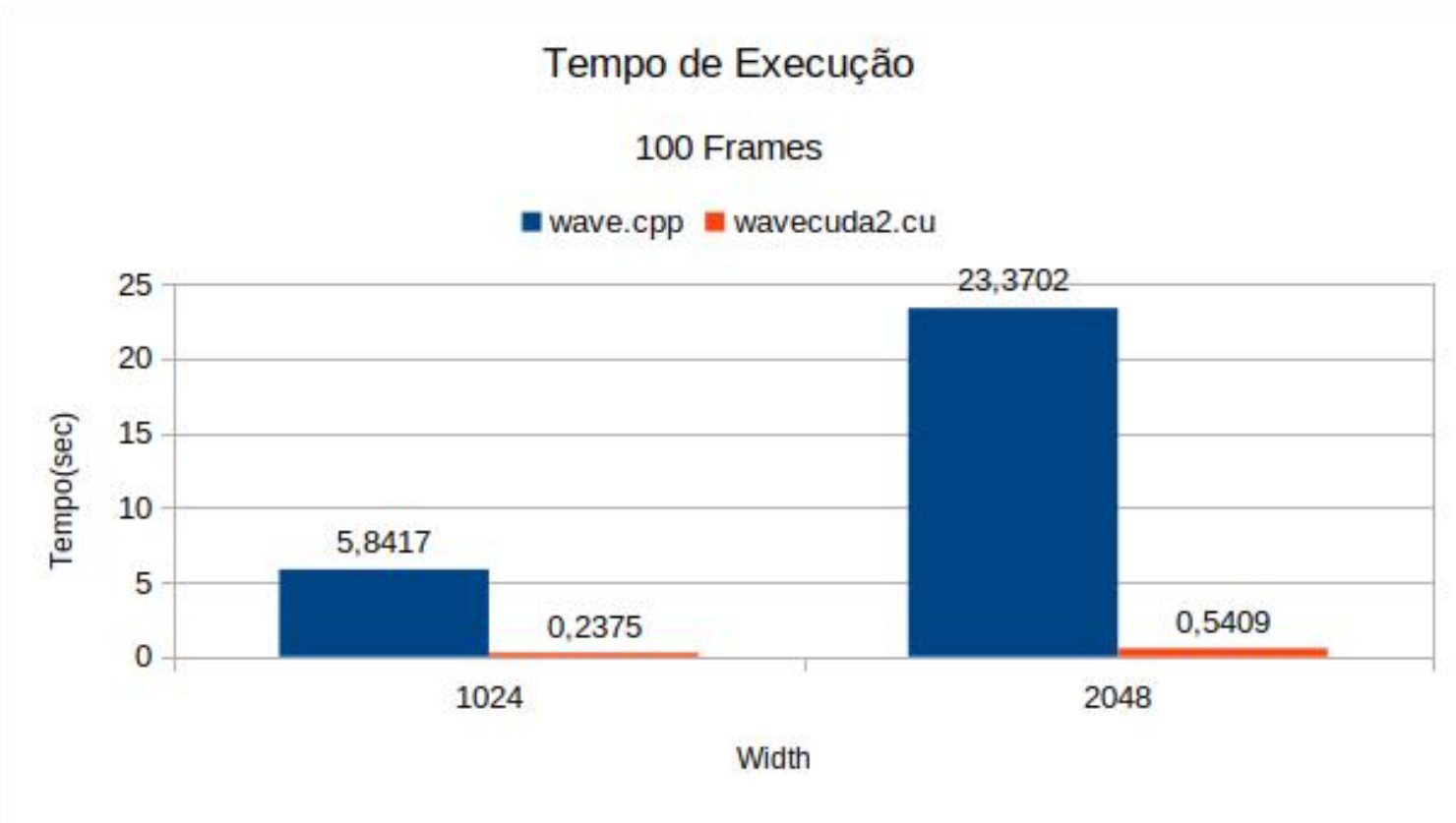
Resultados - Parte 1

- Wave.cpp x Wavecuda1.cu



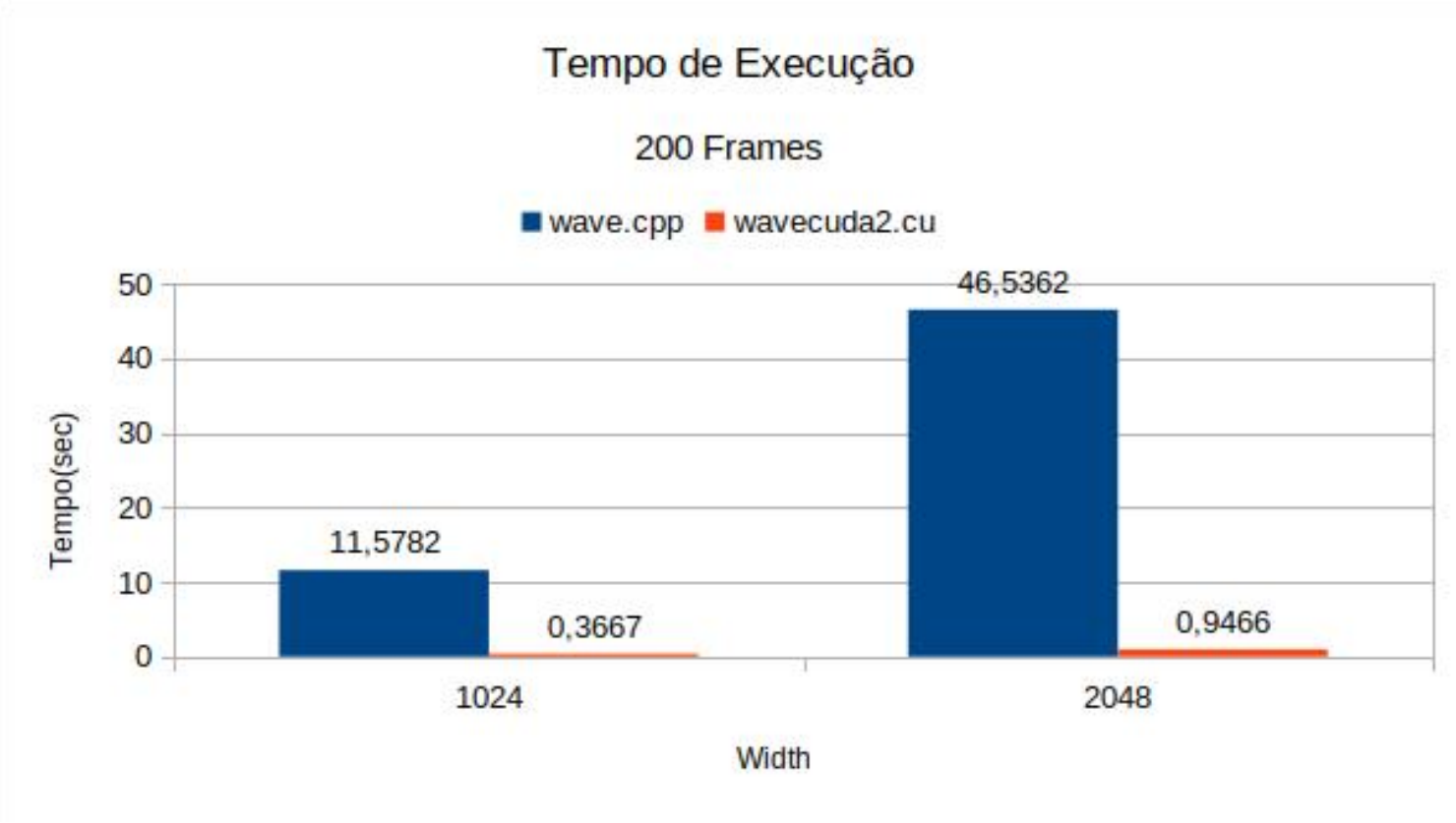
Resultados - Parte 2

- Wave.cpp x Wavecuda2.cu



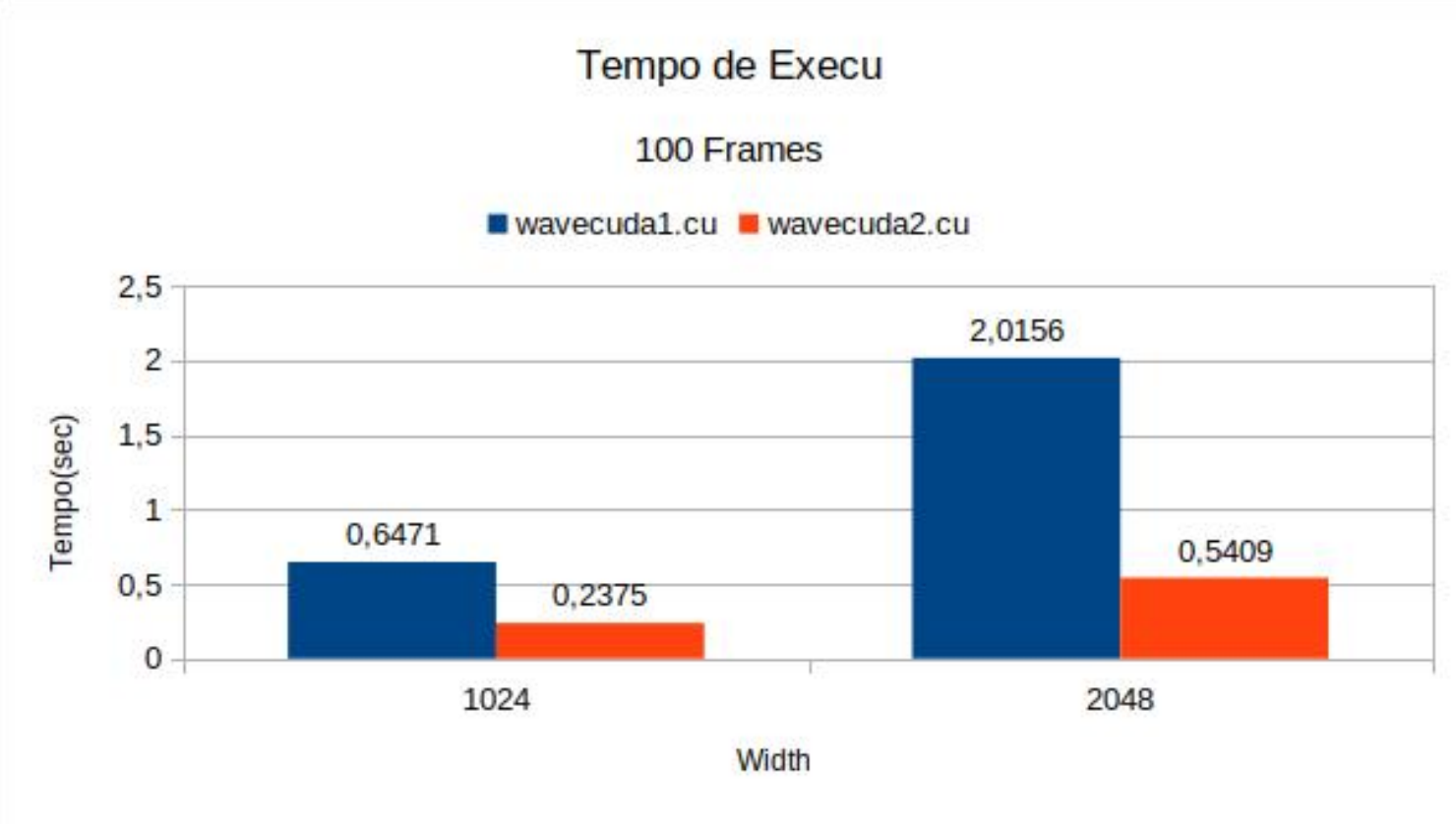
Resultados - Parte 2

- Wave.cpp x Wavecuda2.cu



Comparação

- Wavecuda1.cu x Wavecuda2.cu



Comparação

- Wavecuda1.cu x Wavecuda2.cu