T8 - Programação com CUDA

Bruno Alves

WAVE.CPP

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

  }
```

WAVECUDA1

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

          int frame = threadIdx.x;
            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 = ...

                pic[frame * width * width + row * width + col] = color;
              }
            }

        }
```

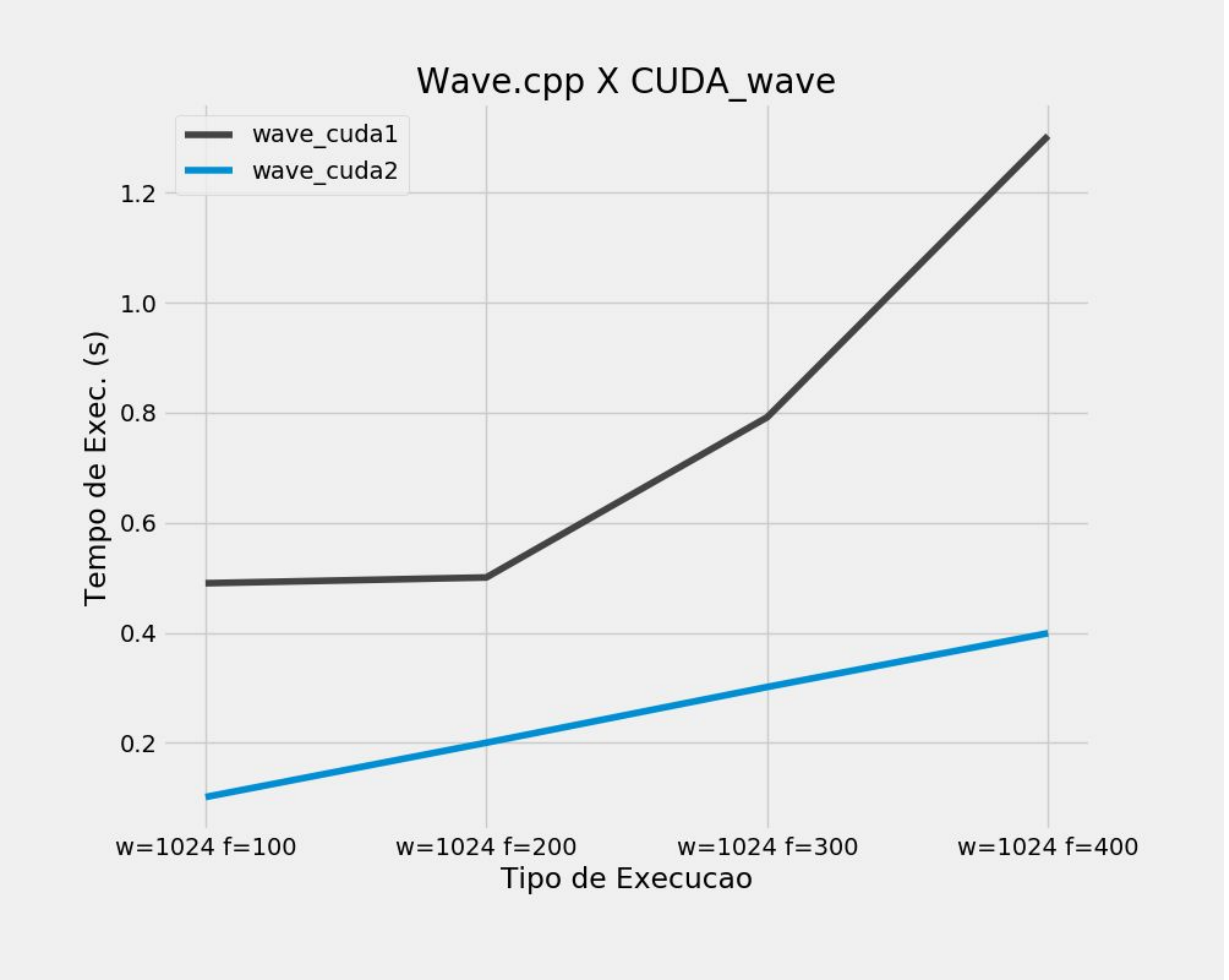
WAVECUDA1

```
// MAIN:
  unsigned char* pic;
  int N = frames * width * width;

  // Alocação da memória compartilhada
  cudaMallocManaged(&pic, N*sizeof(unsigned char));

  // Um bloco com FRAMES threads
  wave<<<1, frames>>>(width, pic);
```

PARTE I


Wave.cpp X CUDA_wave
wave_cuda1
wave_cuda2
1.2
1.0
0.8
0.6
0.4
0.2
Tempo de Exec. (s)
w=1024 f=100
w=1024 f=200
w=1024 f=300
w=1024 f=400
Tipo de Execucao

WAVECUDA2

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

  int row = threadIdx.x;
  int start_frame = (frames/2) * blockIdx.x;
  int stop_frame = (frames/2) *  (blockIdx.x + 1);

  for(int frame = start_frame; frame < stop_frame; frame++){
    for(int col = 0; col < width; col++){
    /// ...
    }
  }

}
```

WAVECUDA2

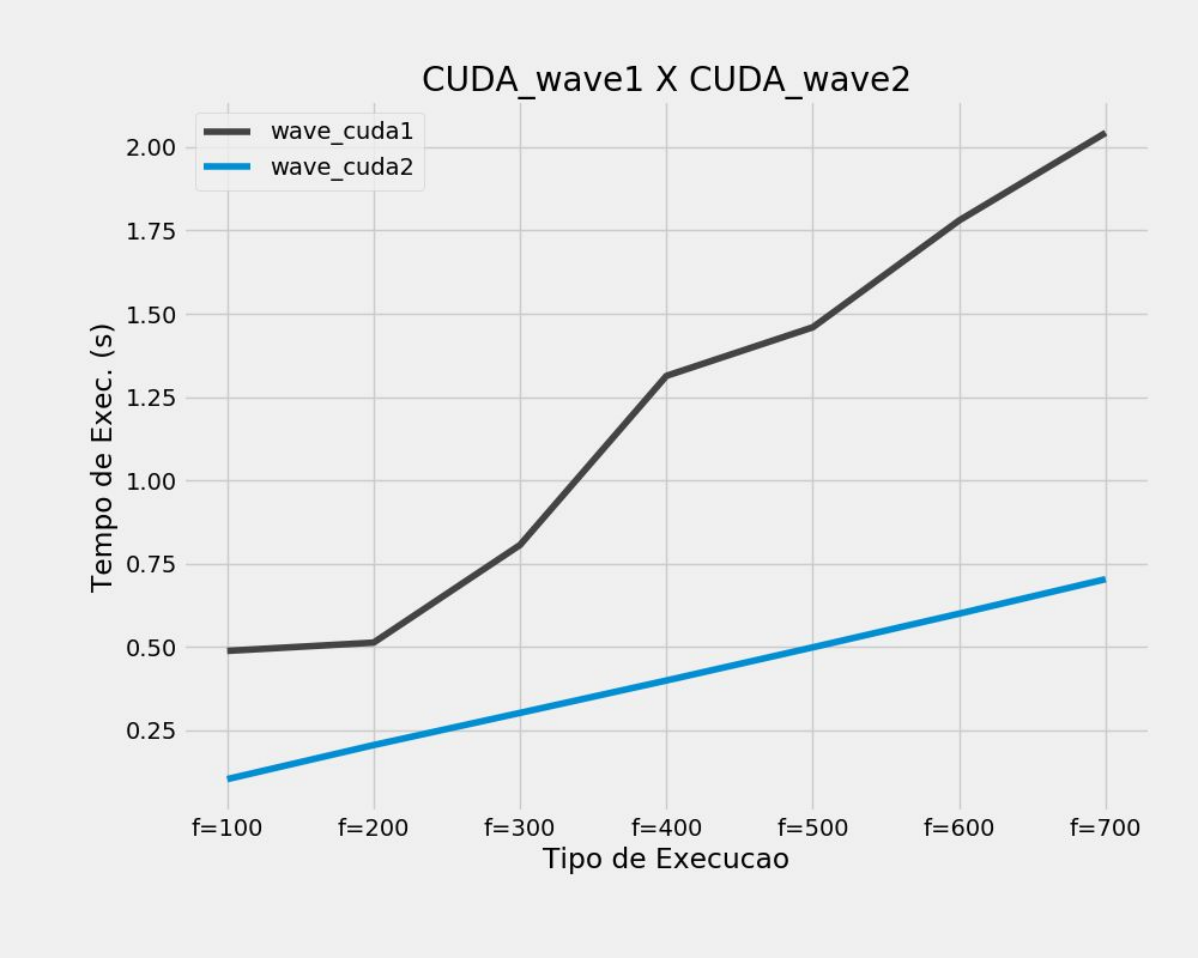
```
// allocate picture array
unsigned char* pic;
int N = frames * width * width;

// O tamanho do vetor é alocado.
cudaMallocManaged(&pic, N*sizeof(unsigned char));


// Uma thread é criada para cada linha de pixels e
// são criados dois blocos que calculam os valores
// para a metade dos frames informados
wave<<<2, width>>>(pic, width, frames);

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

PARTE II


CUDA_wave1 X CUDA_wave2
wave_cuda1
wave_cuda2
Tempo de Exec. (s)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
f=100
f=200
f=300
f=400
f=500
f=600
f=700
Tipo de Execucao