Programação Paralela

T8: Geração de Imagem em Paralelo com CUDA

Rafael Vales Bettker

Paralelização

Parte 1

Um bloco com várias threads, onde cada uma delas fica responsável por computar um frame da imagem.

Parte 2

Vários blocos, cada um responsável por uma linha de pixels da imagem e cada thread desse bloco por um frame.

Parte 1: main()

```
cudaMallocManaged(&pic, frames * width * width *
sizeof(unsigned char));

cudaDeviceProp devProp;
cudaGetDeviceProperties(&devProp, 0);
int maxThreadsPerBlock = devProp.maxThreadsPerBlock;

int numThreads = (frames > maxThreadsPerBlock) ?
maxThreadsPerBlock : frames;
computeImage<<<1, numThreads>>>(pic, frames, width);

cudaDeviceSynchronize();
cudaFree(pic);
```

Parte 1

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

    int idx = threadIdx.x;
    int stride = blockDim.x;
    for (int frame = idx; frame < frames; frame += stride) {
        for (int row = 0; row < width; row++) {
            for (int col = 0; col < width; col++) {
                // ...
            }
        }
    }
}
```

Experimentos (1)

Variações de entrada

- Width: 1024;
- Frames: 100, 200, 400, 800 e 1600.

Resultados (1)

width frames	wave.cpp	wavecuda1.cu	speedup
1024 100	42.4468s	0.6151s	69.0
1024 200	84.7403s	0.6732s	125.9
1024 400	169.3149s	1.3003s	130.2
1024 800	338.5237s	2.3285s	145.4
1024 1600	676.7345s	4.2005s	161.1

Parte 2: main()

```
cudaMallocManaged(&pic, frames * width * width *
sizeof(unsigned char));

cudaGetDeviceProperties(&devProp, 0);
int maxThreadsPerBlock = devProp.maxThreadsPerBlock;
int maxBlocks = devProp.maxGridSize[0];

int numBlocks = (width > maxBlocks) ? maxBlocks : width;
int numThreads = (frames > maxThreadsPerBlock) ?
maxThreadsPerBlock : frames;
computeImage<<<numBlocks, numThreads>>>(pic, frames, width);

cudaDeviceSynchronize();
cudaFree(pic);
```

Parte 2

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

    int idxFrame = threadIdx.x, strideFrame = blockDim.x;
    int idxRow = blockIdx.x, strideRow = gridDim.x;
    for (int f = idxFrame; f < frames; f += strideFrame) {
        for (int r = idxRow; r < width; r += strideRow) {
            for (int col = 0; col < width; col++) {
                // ...
            }
        }
    }
}
```

Experimentos (2)

Variações de entrada

- Width: 512, 1024, 2048;
- Frames: 100, 200, 400, 800.

Resultados (2)

width frames	wave.cpp	wavecuda2.cu	speedup
512 100	10.7236s	0.0081s	1323.9
512 200	21.3183s	0.0181s	1177.8
512 400	42.4642s	0.0377s	1126.4
512 800	84.7404s	0.0778s	1089.2
1024 100	42.4468s	0.0440s	964.7
1024 200	84.7403s	0.0849s	998.1
1024 400	169.3149s	0.1954s	866.5
1024 800	338.5237s	0.5235s	646.7
2048 100	169.3148s	0.1709s	990.7
2048 200	338.4813s	0.3496s	968.2
2048 400	676.6940s	0.7951s	851.1

Gráfico (frames)

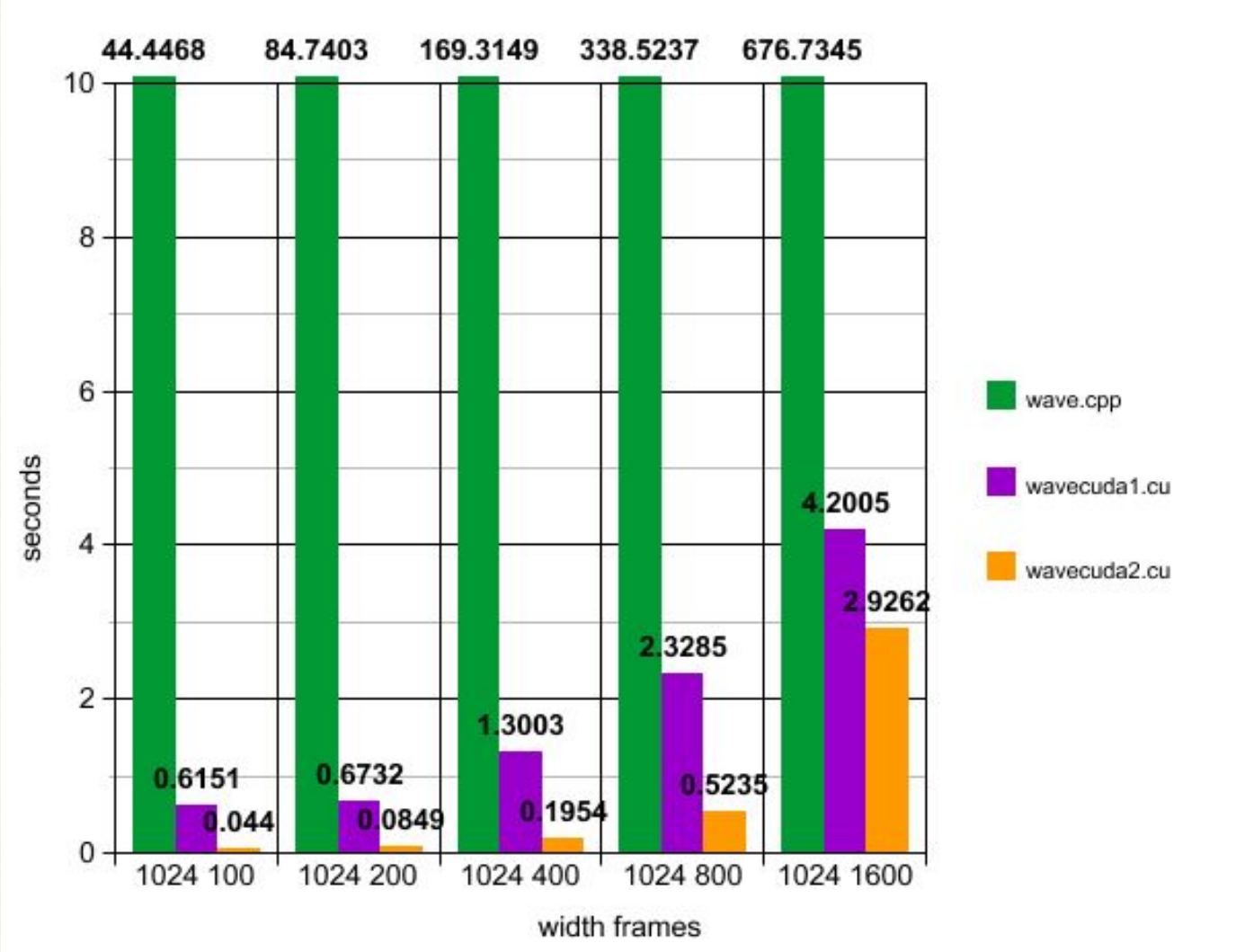
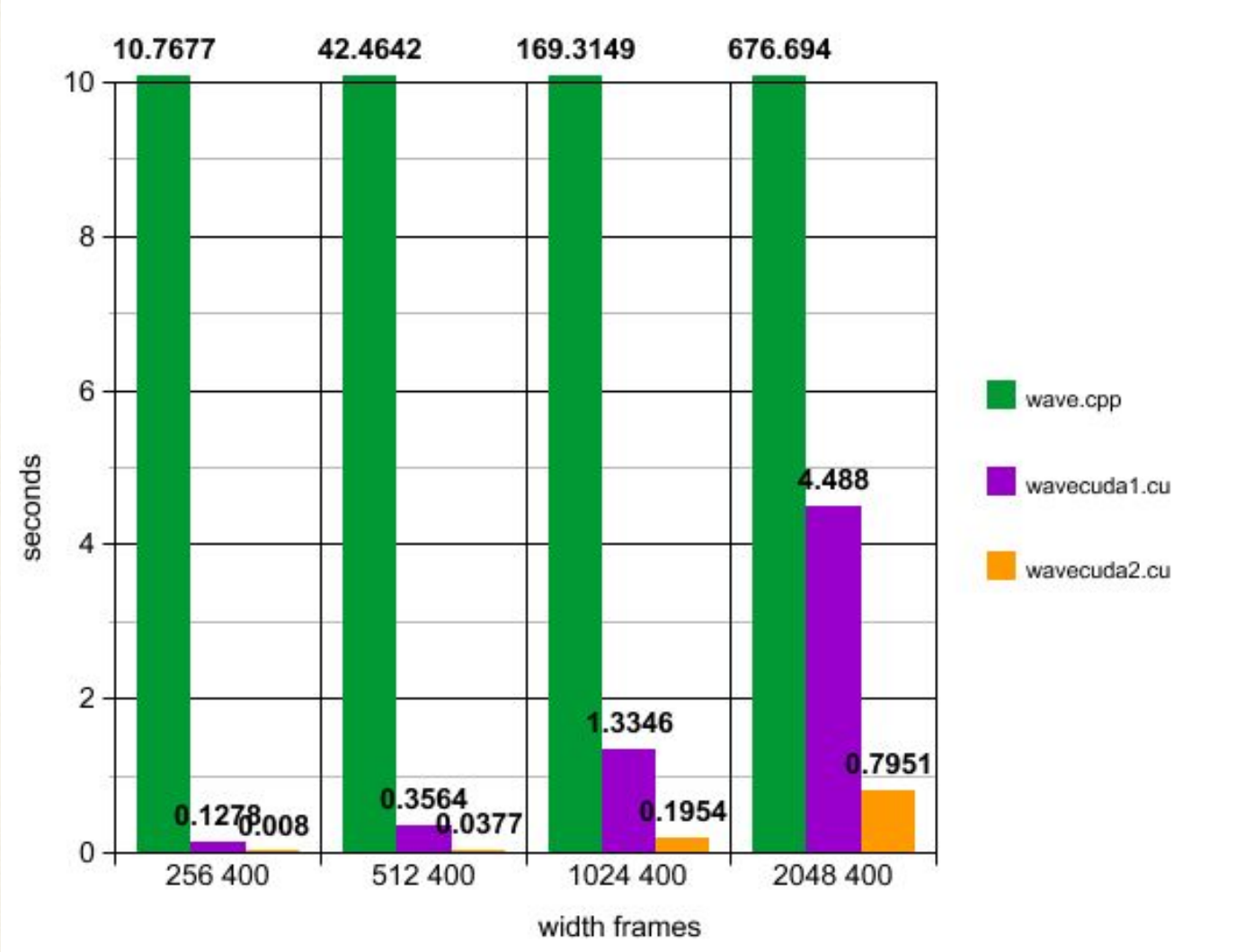


Gráfico (width)



NVPROF

```
==679== NVPROF is profiling process 679, command: ./wavecuda1 1024 400
computing 400 frames of 1024 by 1024 picture
compute time: 1.3334 s
==679== Profiling application: ./wavecuda1 1024 400
==679== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  1.33335s         1  1.33335s  1.33335s  1.33335s  computeFrame(unsigned ch
      API calls:   83.63%  1.33337s         1  1.33337s  1.33337s  1.33337s  cudaDeviceSynchronize
                   15.59%  248.61ms         1  248.61ms  248.61ms  248.61ms  cudaMallocManaged
                    0.73%  11.708ms         1  11.708ms  11.708ms  11.708ms  cudaFree
                    0.01%  232.06us         1  232.06us  232.06us  232.06us  cudaGetDeviceProperties
                    0.01%  231.26us         1  231.26us  231.26us  231.26us  cuDeviceTotalMem
                    0.01%  158.75us        96  1.6530us     138ns  64.673us  cuDeviceGetAttribute
                    0.00%  44.323us         1  44.323us  44.323us  44.323us  cudaLaunchKernel
                    0.00%  32.636us         1  32.636us  32.636us  32.636us  cuDeviceGetName
                    0.00%  2.9060us         1  2.9060us  2.9060us  2.9060us  cuDeviceGetPCIBusId
                    0.00%  2.1940us         3     731ns     265ns  1.2820us  cuDeviceGetCount
                    0.00%  1.0820us         2     541ns     220ns     862ns  cuDeviceGet
                    0.00%     884ns         1     884ns     884ns     884ns  cudaGetDeviceCount
                    0.00%     255ns         1     255ns     255ns     255ns  cuDeviceGetUuid

==679== Unified Memory profiling result:
Device "Tesla T4 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
     202         -         -         -           -  48.75069ms  Gpu page fault groups
```

Ambiente de execuções

Dispositivo da plataforma Google Colaboratory

- Name: Tesla T4
- Total global memory: 15812263936
- Total shared memory per block: 49152
- Total registers per block: 65536
- Maximum threads per block: 1024
- Maximum dimension 0 of block: 1024
- Maximum dimension 0 of grid: 2147483647
- Clock rate: 1590000
- Total constant memory: 65536
- Number of multiprocessors: 40
- ...

Referências

- NVIDIA.

  Programming Guide :: CUDA Toolkit Documentation.

  https://docs.nvidia.com/cuda/cuda-c-programming-guide/