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

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

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

int numThreads = (frames > maxThreadsPerBlock) ? maxThreadsPerBlock : frames;
computeFrame<<<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++) {
        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;
      }
    }
  }
}
```

Experimentos (parte 1)

Variações de entrada

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

Foram realizadas 5 execuções de cada, e então feita a média.

Resultados (parte 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

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

cudaDeviceProp devProp;
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 frame = idxFrame; frame < frames; frame += strideFrame) {
    for (int row = idxRow; row < width; row += strideRow) {
      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;
      }
    }
  }
}
```

Experimentos (parte 2)

Variações de entrada

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

Foram realizadas 5 execuções de cada, e então feita a média.

Resultados (parte 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

Referências

- NVIDIA.

  Programming Guide :: CUDA Toolkit Documentation.

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