GPU radionica

Luko Gjenero
SRCE

GPU programiranje

CUDA C

CUDA RUNTIME API

CUDA

- Linkovi
  - http://docs.nvidia.com/cuda/index.html

CUDA C

- Dohvat dostupnih CUDA uređaja
  - Dohvat postojećih uređaja
  - Dohvat postavki pojedinog uređaja
  - Odabir uređaja

CUDA C

- `cudaError_t cudaGetDeviceCount (int * count)`
- `cudaError_t cudaSetDevice (int device)`
- `cudaError_t cudaGetDevice (int * device)`

CUDA C

- `cudaError_t cudaGetDeviceProperties (struct cudaDeviceProp * prop, int device)`
  - Ime
  - Količina memorije: globalna, dijeljena
  - Max. dretvi po bloku
  - Max. dimenzija grida
  - Major i minor verzija
  - ...
- `cudaError_t cudaChooseDevice (int * device, const struct cudaDeviceProp * prop)`

CUDA C

- Dohvatiti broj CUDA uređaja
- Za svaki uređaj ispisati ime i količinu globalne memorije

CUDA C

- Kernel
  - Definicija posla koji će obaviti GPU dretva u jednom prolazu
- Memorija
  - Definicija globalne memorije

CUDA C

```
__global__ void KernelFunction (
    float * param1,
    int param2)
{ ... }
```

CUDA C

```
int main()
{
    ...
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    KernelFunction<<<numBlocks, threadsPerBlock>>>(param1, param2);
    ...
}
```

CUDA C

```
__global__ void KernelFunction (
    float * param1,
    int param2)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
}
```

CUDA C

- `cudaError_t cudaMalloc (void ** devPtr, size_t size)`
- `cudaError_t cudaMallocHost (void ** ptr, size_t size)`
- `cudaError_t cudaMemcpy (void * dst, const void * src, size_t count, enum cudaMemcpyKind kind)`
- `cudaError_t cudaFree (void * devPtr)`
- `cudaError_t cudaFreeHost (void * ptr)`

CUDA C

- Napisat kernel koji zbraja dva vektora
- Kreirati dva N dimnezionalna vektora
- Zbrojiti na GPU

CUDA C

- Dijeljena memorija (shared)
- `__shared__`
- Brži pristup nego globalnoj
- Dijeli ju isti blok dretvi

CUDA C

```
__global__ void KernelFunction (
    float * param1,
    int param2)
{
    ...
    __shared__ float sharedData[100];
    ...
}
```

CUDA C

- Primjer algoritma
  - 2D matrica 4N x 4N
  - Dijeli se u blokove 4 x 4
  - Bij = Sum(Aij) / (i*4+j)

CUDA C

```
__global__ void KernelFunction (float * A, float * B, int numOfRows, int numOfCols)
{
    float sum = 0;
    for (int i = 0; i < blockDim.x; i++)
    {
        int row = blockIdx.x * blockDim.x + i;
        for (int j = 0; j < blockDim.x; j++)
        {
            int col = blockIdx.y * blockDim.y + j;
            sum += A[row * numOfCols + col];
        }
    }
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    int col = blockIdx.y * blockDim.y + threadIdx.y;
    B[row * numOfCols + col] = sum / (row * blockDim.x + col);
}
```

CUDA C

```
__global__ void KernelFunction (float * A, float * B, int numOfRows, int numOfCols)
{
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    int col = blockIdx.y * blockDim.y + threadIdx.y
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    As[threadIdx.x][threadIdx.y] = A[row * numOfCols + col];
    __syncthreads();
    float sum = 0;
    for (int i = 0; i < blockDim.x; i++)
    {
        for (int j = 0; j < blockDim.y; j++)
        {
            sum += As[i][j];
        }
    }
    B[row * numOfCols + col] = sum / (row * blockDim.x + col);
```

CUDA C

- Algoritma
  - 2D matrica 8N x 8N
  - Dijeli se u blokove 8 x 8
  - Bij = Avg(Aij) / (i*8+j)

CUDA C

- CUDA streams
  - Slijed naredbi

CUDA C

- `cudaError_t cudaStreamCreate (cudaStream_t *pStream)`
- `cudaError_t cudaStreamDestroy (cudaStream_t stream)`
- `cudaError_t cudaStreamQuery (cudaStream_t stream)`
- `cudaError_t cudaStreamSynchronize (cudaStream_t stream)`
- `cudaError_t cudaStreamWaitEvent (cudaStream_t stream, cudaEvent_t event, unsigned int flags)`

CUDA C

- ```
  cudaError_t cudaMemcpyAsync ( void* dst,
  const void* src, size_t count,
  cudaMemcpyKind kind, cudaStream_t stream
  = 0 )
  ```
  ```

CUDA C

- Prepraviti zadatak sa zbrajanjem dva vektora tako da koristi slijed naredbi
- Aplikacija neka čeka na završetak slijeda

CUDA C

- Agoritam
  - “Game of life” 2D matrica
  - Svaki sljedeći korak element matrice jednak
    - ½ lijevog susjeda
    - ¼ donjeg susjeda
    - 1/8 desno-dolje susjeda
    - 1/8 lijevo-gore susjeda
- Dobiti matricu nakon N iteracija (N >>)