LESSON 3

Constant Memory and Random Numbers

TOPICS

- How to use constant memory
- How to create random numbers

- *Key Words: __constant__, cudaMemcpyToSymbol(), curand.h,*

Constant Memory

- The constant memory is the best way in which threads can read-only data on the GPU.

- In our applications using the constant memory instead of global memory can help us to reduce the memory bandwidth.

- To make use of the constant memory we have to add **__constant__** to the variables, and we do not have to use *cudaMalloc()* or *cudaFree()* functions to allocate or remove space in the memory.

- However, in some way we have to allocate our constant data on the GPU, so we are going to use the *cudaMemcpyToSymbol()* function. This function copies the data from the host to the constant memory on the device.

```
cudaError_t   cudaMemcpyToSymbol    ( const char *symbol,
const void * src,
size_t  count,
size_t  offset = 0,
enumcudaMemcpyKind  kind =cudaMemcpyHostToDevice
)
where:
symbol - Symbol destination on device
src- Source memory address
count- Size in bytes to copy
offset - Offset from start of symbol in bytes
kind - Type of transfer
```

- In the size of the constant memory is exceeded, it will generate the error:

  const space overflowed

- Each thread can:
  - Read / write per thread **registers**
    - ~1 cycle, extremely high throughput
  - Read / write per thread **shared memory**
    - ~5 cycle, high throughput
  - Read / write per thread **global memory**
    - ~500 cycle, modest throughput
  - Read / write per thread **constant memory**
    - ~5 cycle, high throughput

Example

- 01reduction_constant: This example coputes the dot product between two vectors using constant memory.

Practice

- 02mse_constant: This code must obtain the Mean Square Error (MSE) from different sections of a matrix using constant memory.

Random Numbers using CURAND library

- The CURAND library provides facilities that focus on the simple and efficient generation of high-quality pseudorandom and quasirandom numbers.

- A pseudorandom sequence of numbers satisfies most of the statistical properties of a truly random sequence but is generated by a deterministic algorithm.

- A quasirandom sequence of n -dimensional points is generated by a deterministic algorithm designed to fill an n -dimensional space evenly.

- CURAND consists of two pieces: a library on the host (CPU) side and a device (GPU) header file.

- The host-side library is treated like any other CPU library: users include the header file, /include/curand.h, to get function declarations and then link against the library.

- Random numbers can be generated on the device or on the host CPU. For device generation, calls to the library happen on the host, but the actual work of random number generation occurs on the device.

- The resulting random numbers are stored in global memory on the device. Users can then call their own kernels to use the random numbers, or they can copy the random numbers back to the host for further processing.

- For host CPU generation, all of the work is done on the host, and the random numbers are stored in host memory.

- The second piece of CURAND is the device header file, /include/curand_kernel.h.

- This file defines device functions for setting up random number generator states and generating sequences of random numbers.

- User code may include this header file, and user-written kernels may then call the device functions defined in the header file.

Example

- 03random_curand: This example generates random numers using the curand.h library.

Practice

- 04shared_random: This must generate random numbers to execute the reduction operation between two vectors.