CUDA Programming Cheat Sheet by m_amendola

NVIDIA CUDA C Programming

Cuda Kernels

A CUDA Kernel function is defined using the __global__ keyword.

A Kernel is executed N times in parallel by N different threads on the device

Each thread has a unique ID stored in the built-in *threadIdx* variable, a struct with components x,y,z.

Each thread block has a unique ID stored in the built-in *blockIdx* variable, a struct with components x,y,z.

Kernel Configuration

Kernel Execution Config-uration

```
kernelFunction<<<num_blocks, num_threads>>>(params)
```

num_blocks	The number of thread blocks along each dimension of the grid.
num_th-reads	The number of threads along each dimension of the thread block

Memory Workflow

First we allocate and "build" the input on the **host**.
Then we allocate dynamic memory on the **device**, obtaining pointers to the allocated memory areas.
Finally, we **initialize** the memory on the device and we **copy** the memory from the host to the device.

At the end of the computation, we may want to copy the memory from the device to the host.

Copy operation is *blocking*.

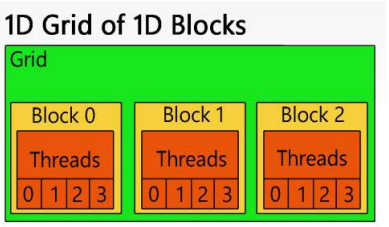
CUDA Thread Organization

Thread are grouped in blocks and can be organized in 1 to 3 dimensions.

Blocks are grouped into grids which can be organized in 1 to 3 dimensions.

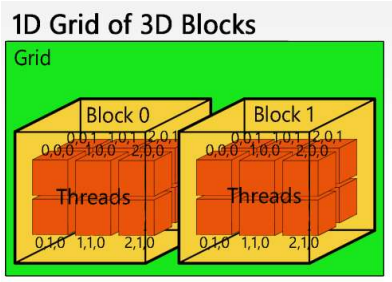
Blocks are executed independently.

1D Grid of 1D Blocks




```
int index = blockIdx.x * blockDim.x + threadIdx.x;
```

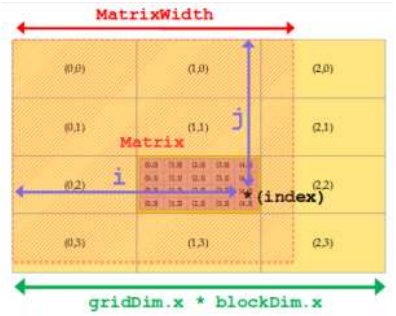
1D Grid of 3D Blocks




```
int index = blockIdx.x  blockDim.x  blockDim.y  blockDim.z + threadIdx.z  blockDim.y  blockDim.x + threadIdx.y  blockDim.x + threadIdx.x;
```

Memory Allocation API Functions

2D Grid of 2D Blocks applied on a Matrix



The index of each thread is identified by two coordinates i and j.
We can find i applying the rule of 1D Grid of 1D Blocks over the x axis:

```
int i = blockIdx.x * blockDim.x + threadIdx.x;
```

And we can find j applying the rule of 1D Grid of 1D Blocks over the y axis:

```
int j = blockIdx.y * blockDim.y + threadIdx.y;
```

Thus, knowing that a row in the grid is large *GridDim.x times BlockDim.x*, we can calculate the index:

```
int index = j gridDim.x  blockDim.x +i;
```

CUDA Events

Declaring a Cuda Event	`cudaEvent_t event;`
Allocating the event	`cudaEventCreate(&event);`
Recording the Event.	`cudaEventRecord(event);`
Synchronizing the event	`cudaEventSynchronize(event);`
Find elapsed time between two events	`cudaEventElapsedTime(&elapsed, a, b);`
Free event variables	`cudaEventDestroy(event);`

Dynamic memory allocation	`cudaMalloc ((void **) &udev, N*sizeof(double));`
Memory Initialization on device	`cudaMemset(void *devPtr, int val, size_t count;`
Copying data from host to device	`cudaMemCpy(void dst, void src, size_t size, cudaMemcpyHostToDev`

CUDA Streams

GPU operations on CUDA use execution queues called streams.

Operations pushed in a stream are executed according to a FIFO policy.

There is a default Stream, called *stream 0*.

Operations pushed in a non-default stream will be executed after all operations on default stream are emptied.

Operations assigned to default stream introduce implicit synchronization barriers among other streams.

CUDA Streams API

Create a stream	`cudaStreamCreate(stream1) ;`
Deallocate a stream	`cudaStreamDestroy(stream)`
Block host until all operations on a stream are completed.	`cudaStreamSynchronize(stream);`

We can use stream to obtain the concurrent execution of the same kernel or different kernels.

Synchronization operations

Explicit Synchronization	Implicit Synchronization
cudaDeviceSynchronize() blocks host code until all operations on device are completed	Operations assigned to default stream
cudaStreamWaitEvent(stream, event) blocks all operations assigned to a stream until event is reached.	Memory Allocations on device
	Settings operations on device
	Page-locked memory allocations

Copying data from device to host	`cudaMemCpy(void *dst*, *void* src, size_t size, cudaMemcpyDeviceToH`

After 4.0, CUDA supports **Unified Virtual Addressing** meaning that the systems itself knows where the buffer is allocated. The *direction* parameter must be set to **cudaMemcpyDefault**.

CUDA API

https://docs.nvidia.com/cuda/cuda-runtime-api/index.html

Global Memory

Declaring a static variable	`__device__ type variable_name;`
Declaring a dynamic variable	`cudaMalloc((void **) &ptr, size);`
Deallo-cating a dynamic variable	`cudaFree(ptr)`
Allocating an aligned 2D buffer where elements are padded so that each row is aligned	`cudaMallocPitch(&ptr, &pitch, width*sizeof(float), height)`

cudaMallocPitch returns an integer pitch that can be used to access row element with stride access. For example:

```
float *row = devPtr + r * pitch;
```

Shared Memory

Static variable declaration inside the kernel.	`__shared__ type shmem[SIZE];`
Dynamic variable allocation outside the kernel	`extern __shared__ type *shmem;`

Constant memory

Declaring a static variable	`__constant__ type variable_name;`
Copy memory from host to device.	`cudaMemcpyToSymbol(variable_name, &host_src, sizeof(type), cudaI`

We cannot declare a dynamic variable on the costant memory

Texture Memory

Managing texture memory

Allocate global memory on device	`cudaMalloc(&M, memsize)`
Create a texture reference.	`texture<datatype, dim> MtextureRef;`
Create a channel descriptor	`cudaChannelFormatDesc Mdesc = cudaCreateChannelDesc<datatype>()`
Bind the texture reference to memory.	`cudaBindTexture(0, MtextureRef, M, Mdesc)`
Unbind at the end.	`cudaUnbindTexture(MTextureRef);`
In order to access the texture memory, we can use the texture reference *Mtexture-reRef.**	`text1Dfetch(MtextureRef, address);`
Accessing 2D cuda array.	`text2Dfetch(MtextureRef, address);`
Accessing 3D cuda array.	`text3Dfetch(MtextureRef, address);`

Asynchronous Data Transfers

Allocates page-locked memory on the host.	`cudaMallocHost(buffer, size)`
Frees page-locked memory.	`cudaFreeHost(buffer)`
Registers an existing host memory range for use by CUDA.	`cudaHostRegister()`
Unregisters a memory range that was registered with cudaHostRegister.	`cudaHostUnregister()`
Copies data between host and device.	`cudaMemcpyAsync(dest_buffer, src_buffer, dest_size, src_size, c`

These operations must be queued into a non-default stream.

Page-locked Memory

Pageable memory is memory which is allowed to be paged in or paged out whereas **page-locked memory** is memory not allowed to be paged in or paged out.

Page out is moving data from RAM to HDD, while *page in* means moving data from HDD to RAM. These operations occurs when the main memory does not have enough free space.

Source: https://leimao.github.io/blog/Page-Locked-Host-Memory-Data-Transfer/

Error Handling

All CUDA API functions returns an error code of type *cudaError*.

The constant *cudaSuccess* means no error.

cudaGetLastError return the status of the internal error variable. Calling this function resets the internal error to cudaSuccess.

Macro for Error Handling

```
#define CUDA_CHECK(X) {\
cudaError_t _m_cudaStat = X;\
if(cudaSuccess != _m_cudaStat) {\
fprintf(stderr,"\nCUDA_ERROR: %s in file %s line %d\n",\
cudaGetErrorString(_m_cudaStat), __FILE__, __LINE__);\
exit(1);\
} }
...
CUDA_CHECK( cudaMemcpy(d_buf, h_buf, buffSize,
cudaMemcpyHostToDevice) );
```

nvidia

Download the CUDA Programming Cheat Sheet

PDF (recommended)

PDF (5 pages)

Alternative Downloads

PDF (black and white)

LaTeX

Comments

No comments yet. Add yours below!

Created By

m_amendola

Add a Comment

Your Comment

Your Name

Your Email Address

Your Comment

Post Your Comment

Metadata

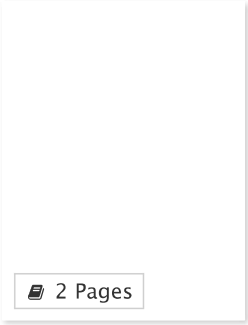
Languages: English

Published: 22nd July, 2023

Related Cheat Sheets

nvidia tlt Cheat Sheet

Latest Cheat Sheet



Football advanced metrics Cheat Sheet

This cheatsheet contains almost all the advanced metrics both commonly known and created by specific analytics companies cited in the sheet.

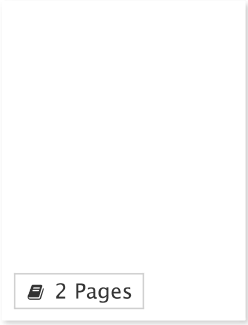
☆ ☆ ☆ ☆ ☆ (0)

90Quantile

8 Nov 24

football, soccer, metrics

Random Cheat Sheet



PTSD Therapeutics Cheat Sheet

☆ ☆ ☆ ☆ ☆ (0)

kfisher17

23 Feb 20, updated 24 Feb 20

pharmacy, ptsd, trauma, ocd

About Cheatography

Cheatography is a collection of 6583 cheat sheets and quick references in 25 languages for everything from French to travel!

Behind the Scenes

If you have any problems, or just want to say hi, you can find us right here:

DaveChild

SpaceDuck

Cheatography

Recent Cheat Sheet Activity

90Quantile updated Football advanced metrics.
17 hours 37 mins ago

TME520 updated Planet X3 for MSDOS keyboard mapping.
1 day 14 hours ago

DaveLee published PKV & GKV – Unterschiede & Wechsel.
2 days 4 hours ago

Latest Cheat Sheets RSS Feed