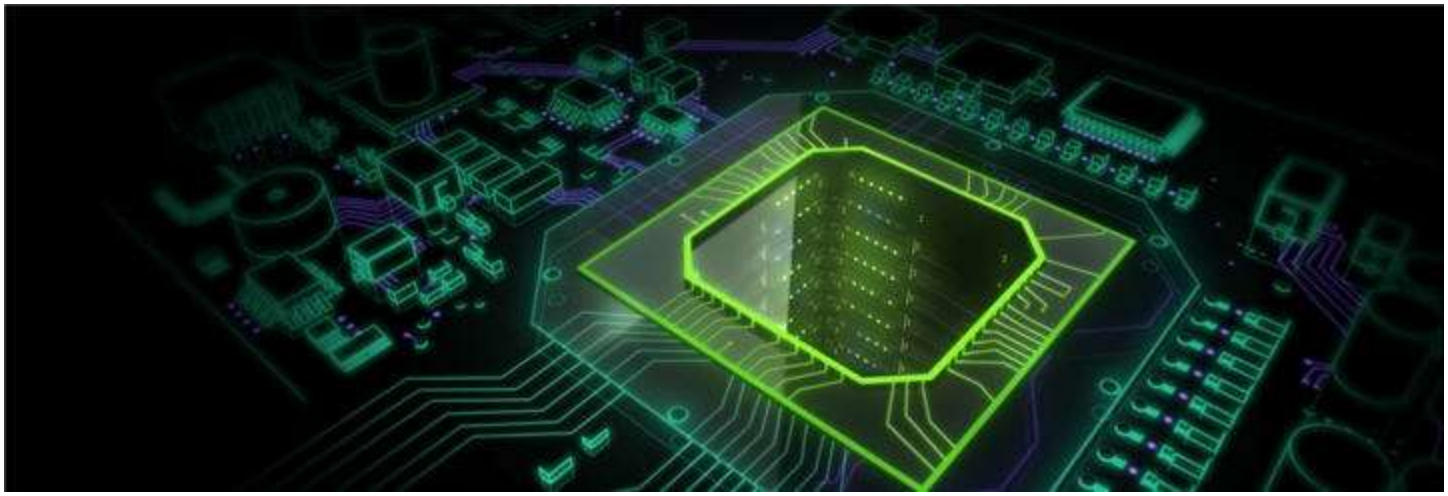
NYU
New York University

CSCI-GA.3033-012

Graphics Processing Units (GPUs): Architecture and Programming

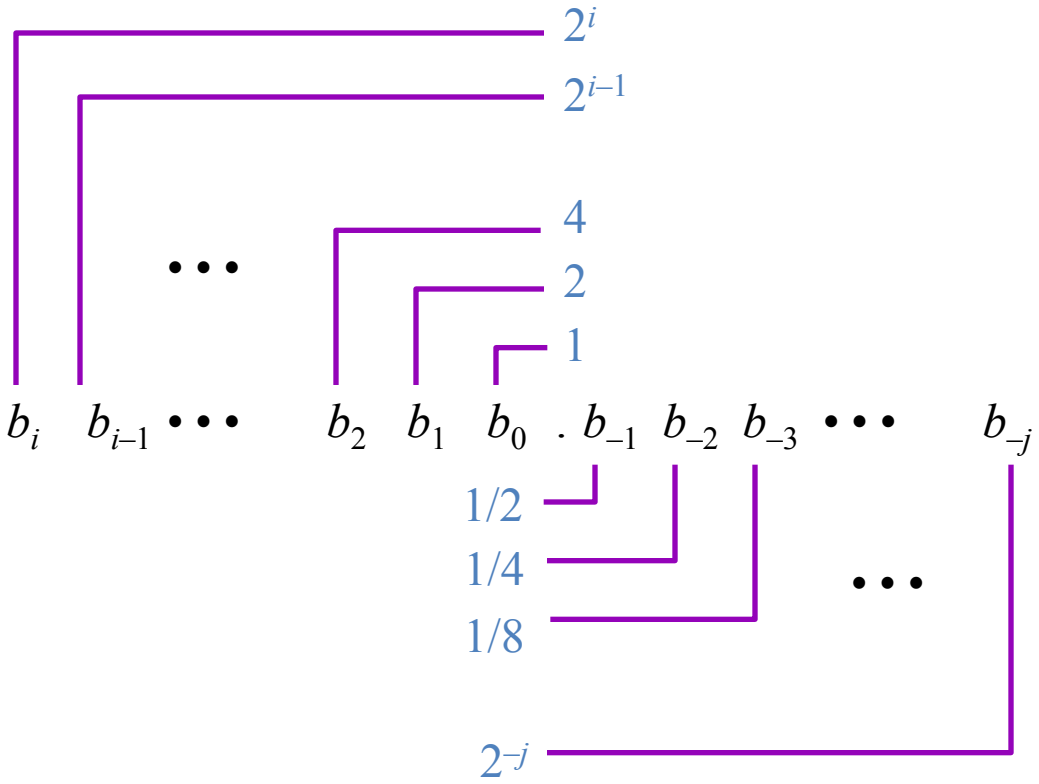
Lecture 8: Advanced Techniques

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Importance of Floating Points

- Many graphics operations are floating point operations
- GPU performance is measure in GFLOPS



Turing Award 1989 to William Kahan for design of the IEEE Floating Point Standards 754 (binary) and 854 (decimal)

What is Excel doing?

A1:	1.33333333333330000	=4/3

Floating Point

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576×10^9
- Representation:
  - sign, exponent, mantissa: $(-1)^{sign} \times mantissa \times 2^{exponent}$
  - more bits for mantissa gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit mantissa
  - double precision: 11 bit exponent, 52 bit mantissa

IEEE 754 floating-point standard

- Leading "1" bit of significand is implicit (called **hidden 1 technique,** except when exp = -127**)**
- Exponent is "biased" to make sorting easier
  - all 0s is smallest exponent
  - all 1s is largest exponent
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{sign} \times (1+\text{significand}) \times 2^{exponent - bias}$

- Example:

  - decimal: -.75 = - ($\frac{1}{2}$ + $\frac{1}{4}$)
  - binary: -.11 = -1.1 x 2^{-1}
  - floating point: exponent = 126 = 01111110

  - IEEE single precision: `10111111010000000000000000000000`

More about IEEE floating Point Standard

Single Precision:

$$(-1)^{sign} \times (1 + mantissa) \times 2^{exponent - 127}$$

The variables shown in red are the numbers stored in the machine

Important! Significant is always 0.XXXX

Floating Point Example

what is the decimal equivalent of

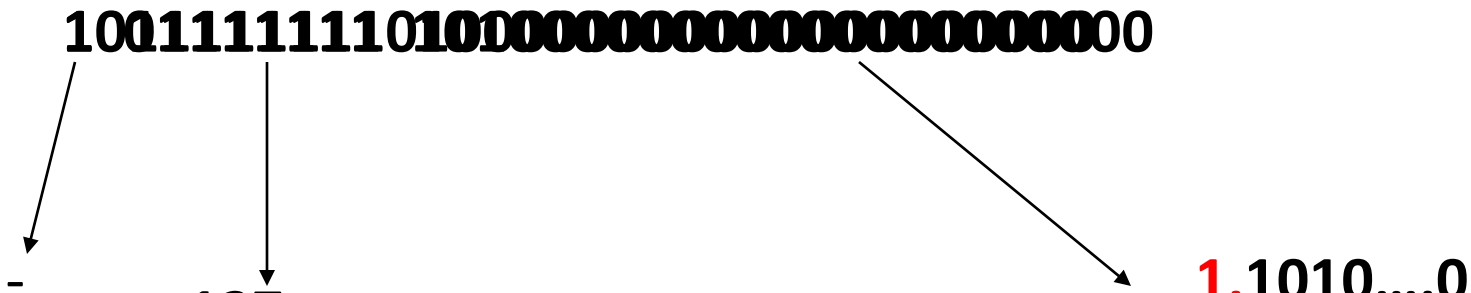
1 01110110 10110000…0

Special Patterns

- Representation of zero
  - No hidden one
  - Exponent and mantissa are 0s
- When all exponent bits are ones
  - If mantissa is zero -> infinity
  - If mantissa is nonzero -> Not a Number (NaN)

Floating Point: IEEE 754

What is the decimal equivalent of:



So:

- Real exponent = 127 -127 = 0
  - There is hidden 1

Final answer = -1.625

Algorithm Considerations

- Non representable numbers are rounded
- This rounding *error* leads to different results depending on the order of operations
  - Non-repeatability makes debugging harder
- A common technique to maximize floating point arithmetic accuracy is to presort data before a reduction computation.

So..

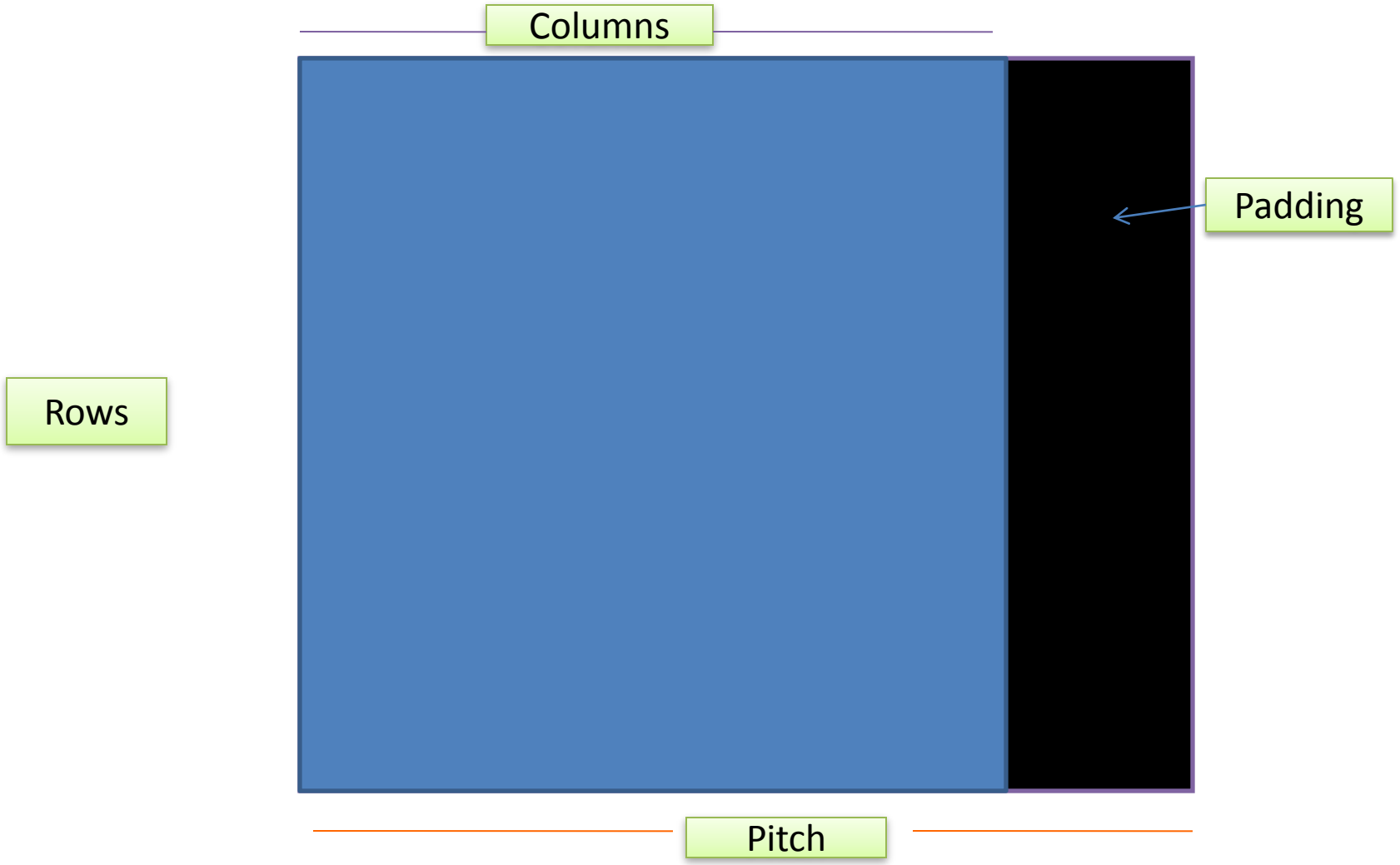
When doing floating-point operations in parallel you have to decide:

- How much accuracy is good enough?
- Do you need single-precision or double precision?
- Can you tolerate presorting overhead, if you care about rounding errors?

Memory Alignment

- Memory access on the GPU works much better if the data items are aligned at 64 byte boundaries.
- Hence, allocating 2D arrays so that every row starts at a 64-byte boundary address will improve performance.
- Difficult to do for a programmer!

Pitch



2D Arrays

- CUDA offers special versions of:
  - Memory allocation of 2D arrays so that every row is padded (if necessary). The function determines the best pitch and returns it to the program. The function name is cudaMallocPitch()
  - Memory copy operations that take into account the pitch that was chosen by the memory allocation operation. The function name is cudaMemcpy2D()

So..

Pitch is a good technique to speedup memory access

- There are two drawbacks that you have to live with:
  - Some wasted space
  - A bit more complicated elements access

Streams

- Sequence of operations that execute in order on device
- A stream can be sequence of kernel launches and host-device memory copies
- Can have several open to the same device at once
- Need GPUs with concurrent transfer/execution capability
- Potential performance improvement: can overlap transfer and computation

Streams

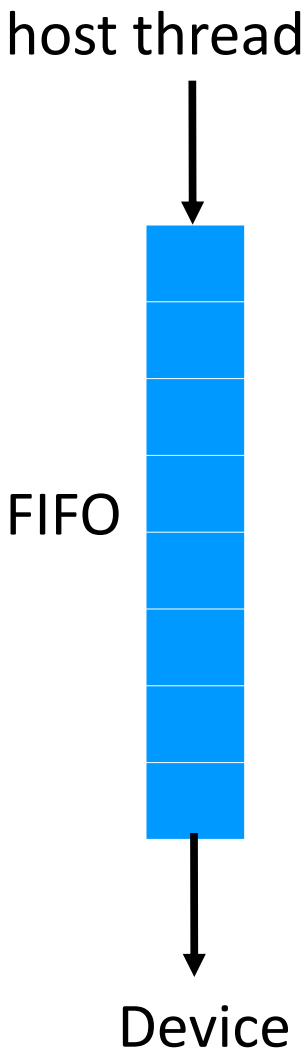
- By default all transfers and kernal launches are assigned to stream 0
  - This means they are executed in order

Streams

```
cudaStream_t stream[2];
for (int i = 0; i < 2; ++i)
    cudaStreamCreate(&stream[i]);
float* hostPtr;
cudaMallocHost(&hostPtr, 2 * size);
```

```
for (int i = 0; i < 2; ++i) {
    cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i * size,
                    size, cudaMemcpyHostToDevice, stream[i]);
    MyKernel<<<100, 512, 0, stream[i]>>>
          (outputDevPtr + i * size, inputDevPtr + i * size, size);
    cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i * size,
                    size, cudaMemcpyDeviceToHost, stream[i]);
}
```

```
for (int i = 0; i < 2; ++i)
    cudaStreamDestroy(stream[i]);
```



Streams

- The amount of overlap execution between two streams depends on:
  - Device supports overlap transfer and kernel execution
  - Devices supports concurrent kernel execution
  - Device supports concurrent data transfer
  - The order on which commands are issued to each stream

So..

- Streams are a good way to overlap execution and transfer, hardware permits.
- Don't confuse kernels, threads, and streams.

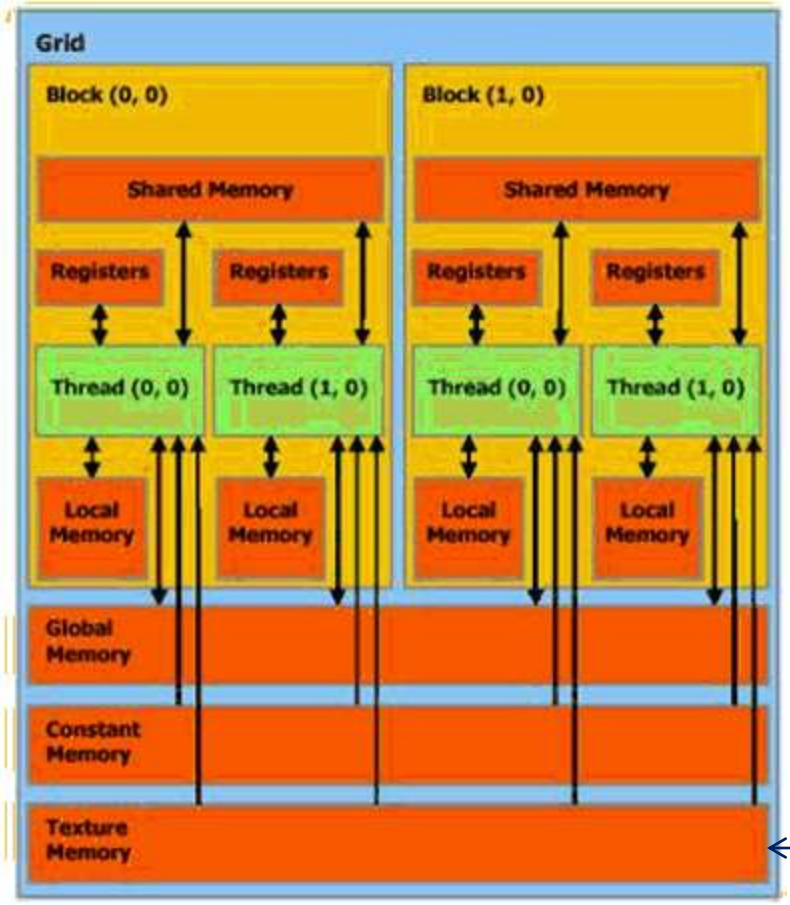
Pinned Pages

- Allocate page(s) from system RAM (cudaMallocHost() or cudaHostAlloc())
  - Accessible by device
  - Cannot be page out
  - Enables highest memory copy performance (cudaMemcpyAsyc())
- If too much pinned pages, overall system performance may greatly suffer.

So..

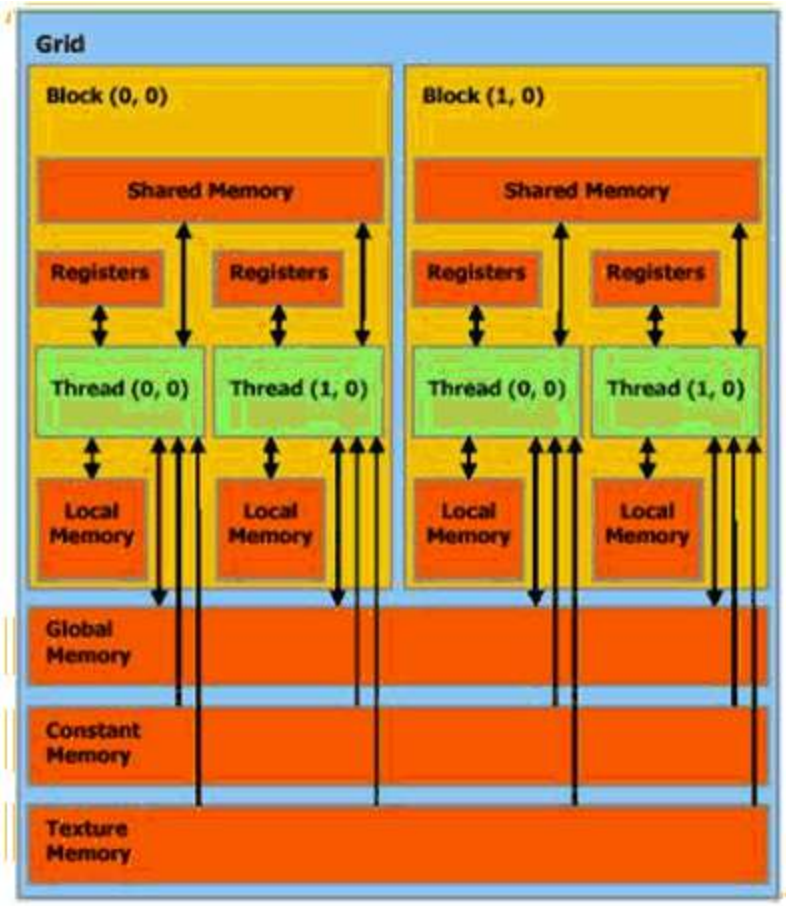
- If the CPU program requires a lot of memory, then pinned pages is not a good idea.

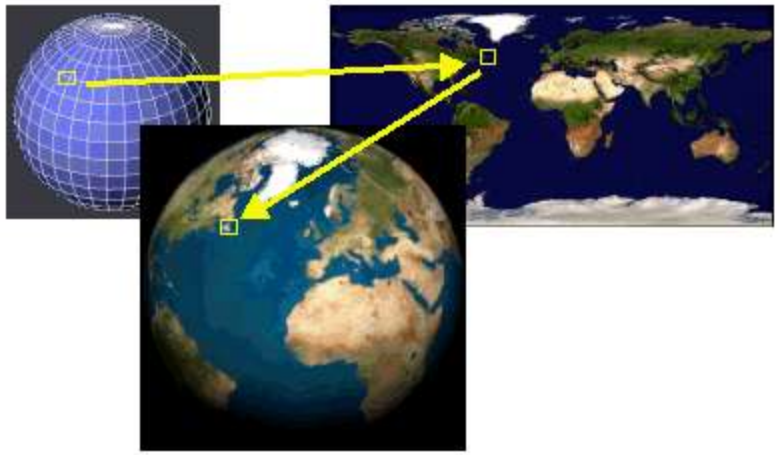
Texture Memory



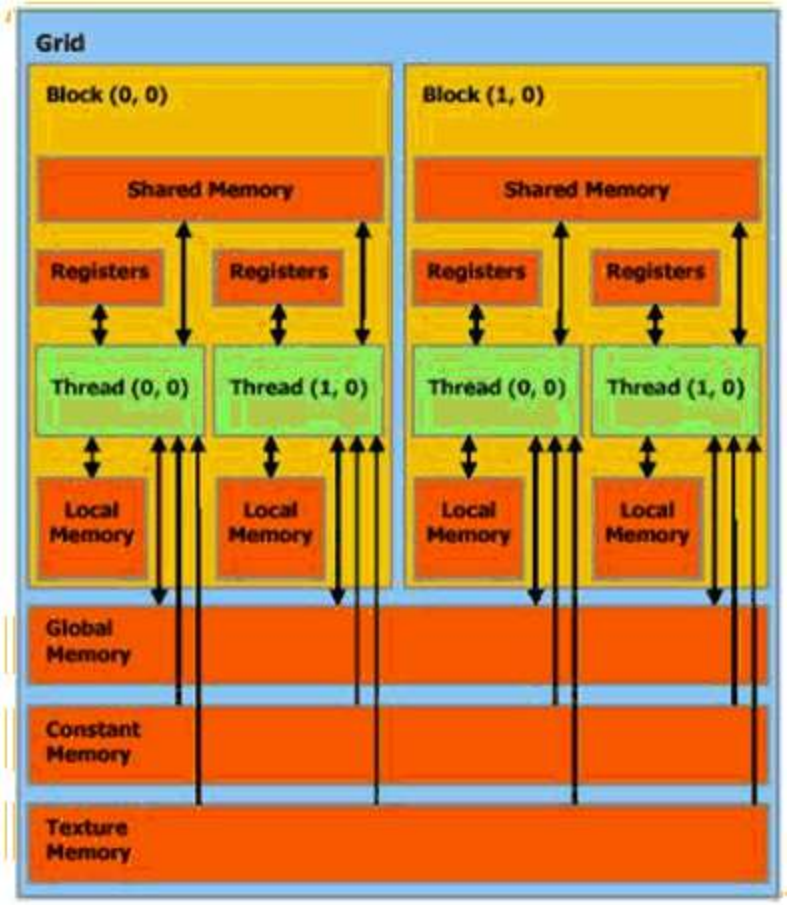
To accelerate frequently performed operations such as mapping a 2D "skin" onto a 3D polygonal model.

Texture Memory





Texture Memory



Capabilities:

- Ability to cache global memory
- Dedicated interpolation hardware
- Provides a way to interact with the display capabilities of the GPU.

The best performance is achieved when the threads of a warp read locations that are close together from a spatial locality perspective.

Texture Memory

- Read only and cached
- The texture cache is optimized for 2D spatial locality.
- Part of DRAM
- The process of reading a texture is called a *texture fetch*.
- Can be addressed as 1D, 2D, or 3D dimensional arrays.
- Elements of the array are called *texels*.

So..

- **R/O no structure → constant memory**

- **R/O array structured → texture memory (be careful when dealing with Fermi)**

Asynchronous Execution

- Some CUDA API calls and all kernel launches are asynchronous with respect to the host code.
- This means error-reporting is also asynchronous.
- Asynchronous transfer (cudaMemcpyAsync()) version *requires pinned host memory*
- On all CUDA-enabled devices, it is possible to overlap host computation with asynchronous data transfers and with device computations.

Asynchronous Execution

```
cudaMemcpyAsync(a_d, a_h, size, cudaMemcpyHostToDevice, 0);
kernel<<<grid, block>>>(a_d);
cpuFunction();
```

Conclusions

- There are many performance enhancement techniques in our arsenal:
  - Alignment
  - Streams
  - Pinned pages
  - Texture memory
  - Asynchronous execution
- If your program is making use of a lot of FP operations, be careful about rounding errors.