Open in app ↗ Sign up Sign in

Medium Search

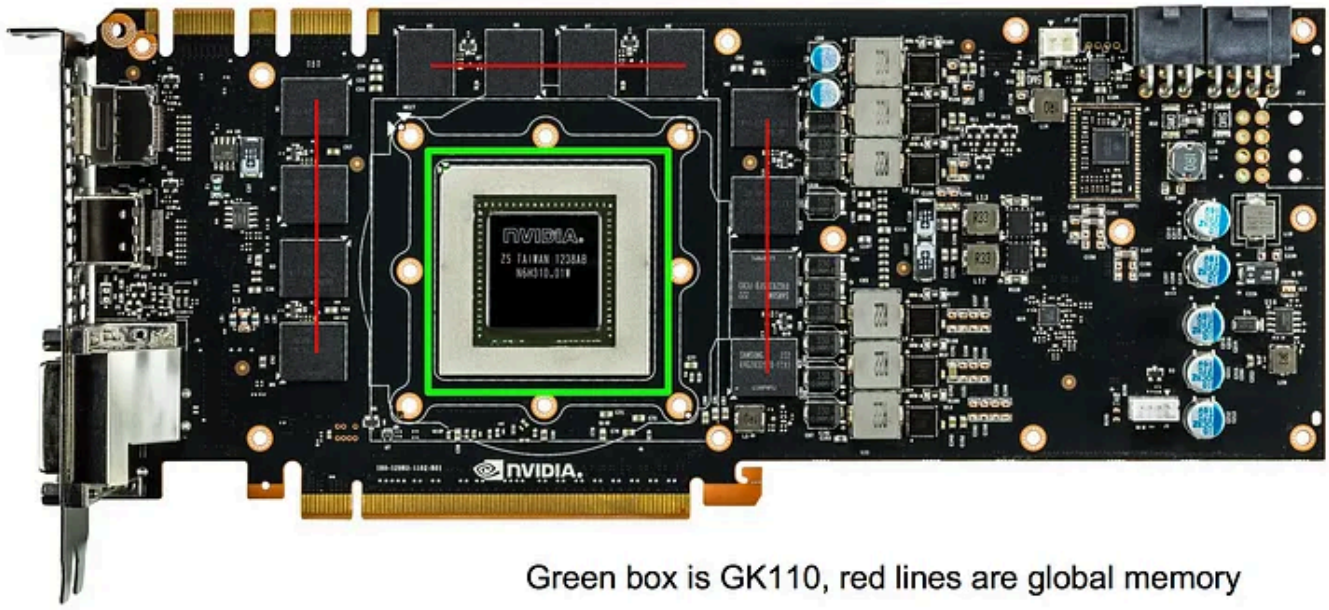
# Basic Concepts in GPU Computing

5 min read · Oct 10, 2017

Hao Gao Follow

Listen Share

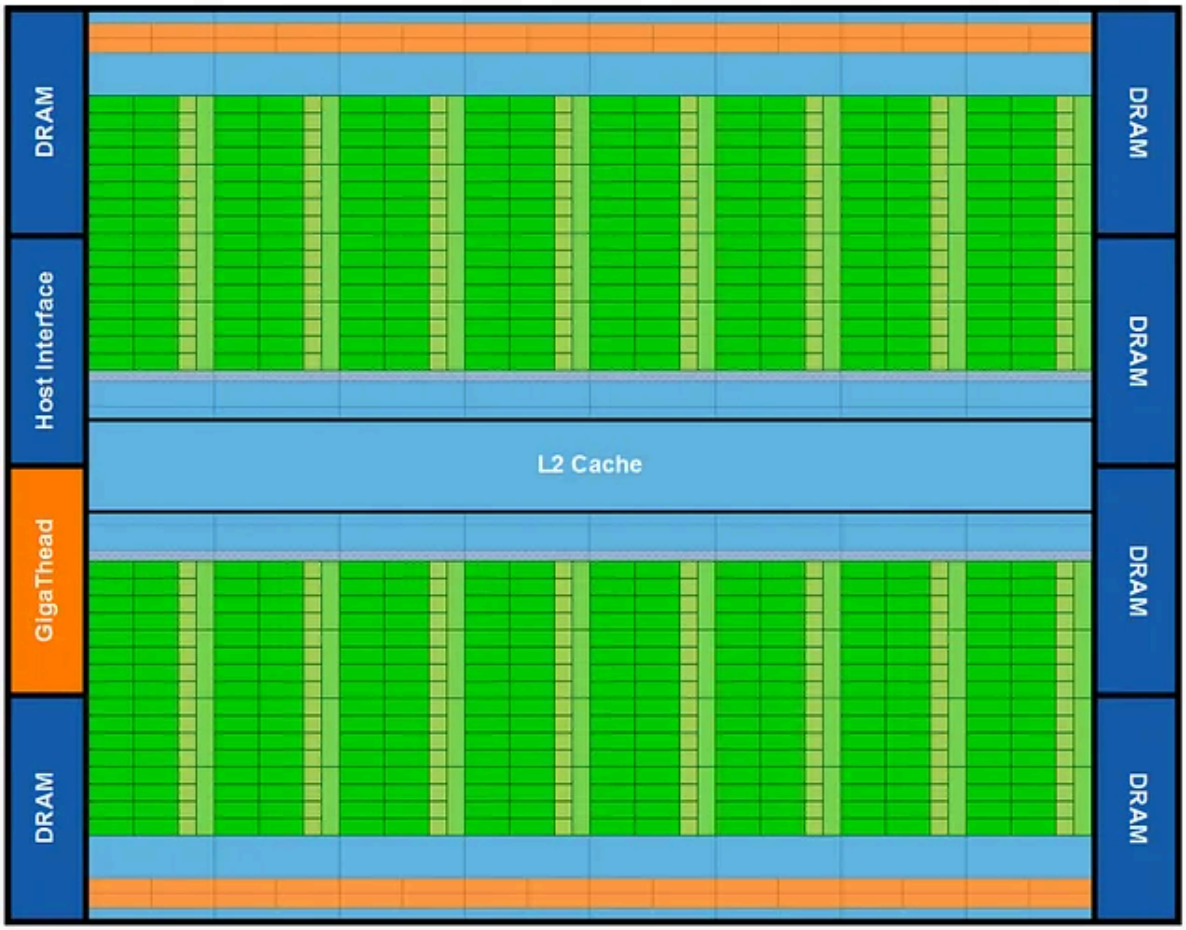
This post mainly goes through the white paper of the Fermi architecture to showcase the concepts in GPU computing. A GPU Card has several memory dies and a GPU unit, as shown in the following image. Memory dies compose of global memory in cuda computing.




GPU Card [2]

## GPU Architecture

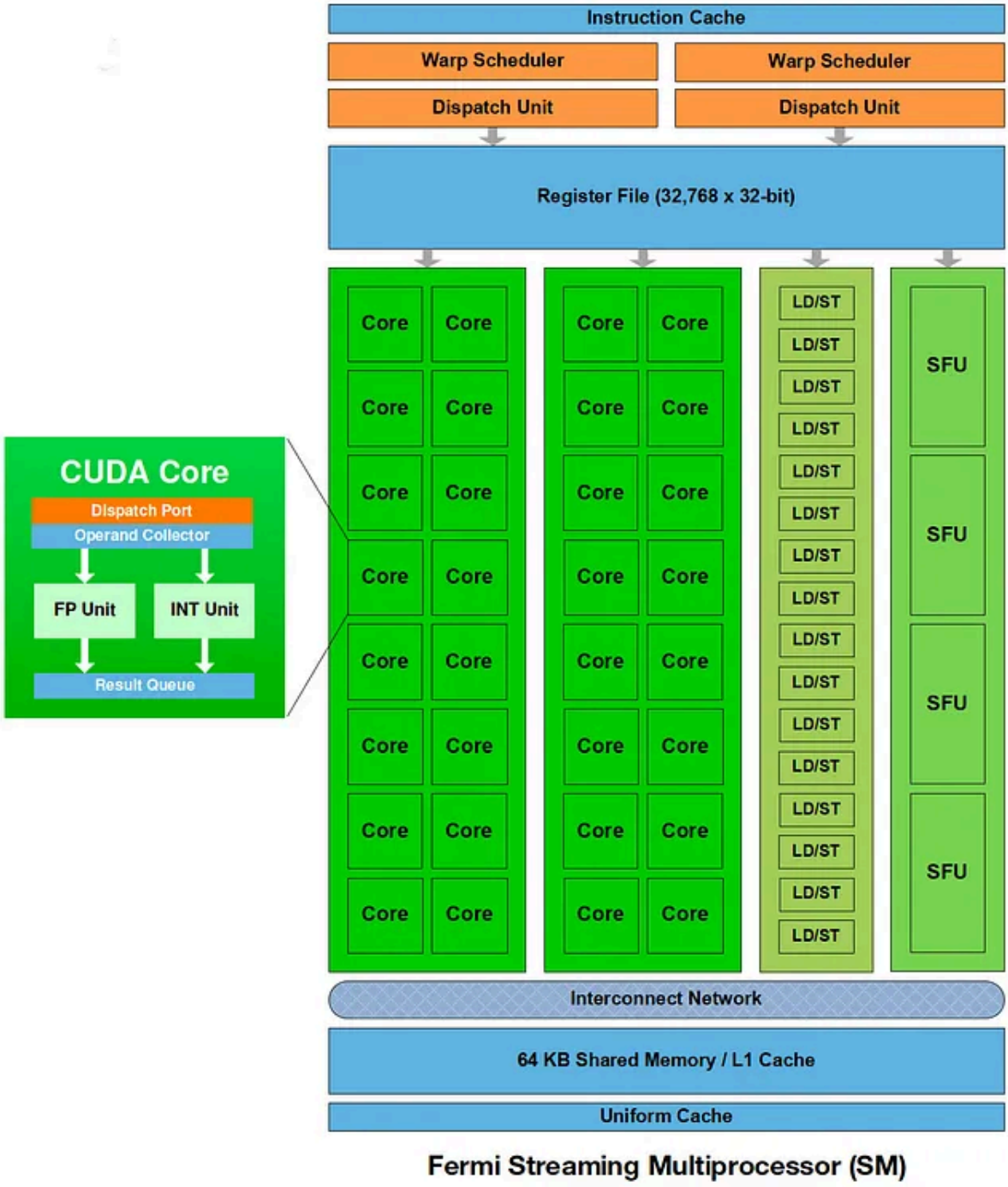
The following graph shows the Fermi architecture. This GPU has 16 streaming multiprocessor (SM), which contains 32 cuda cores each. Every cuda is an execute unit for integer and float numbers.



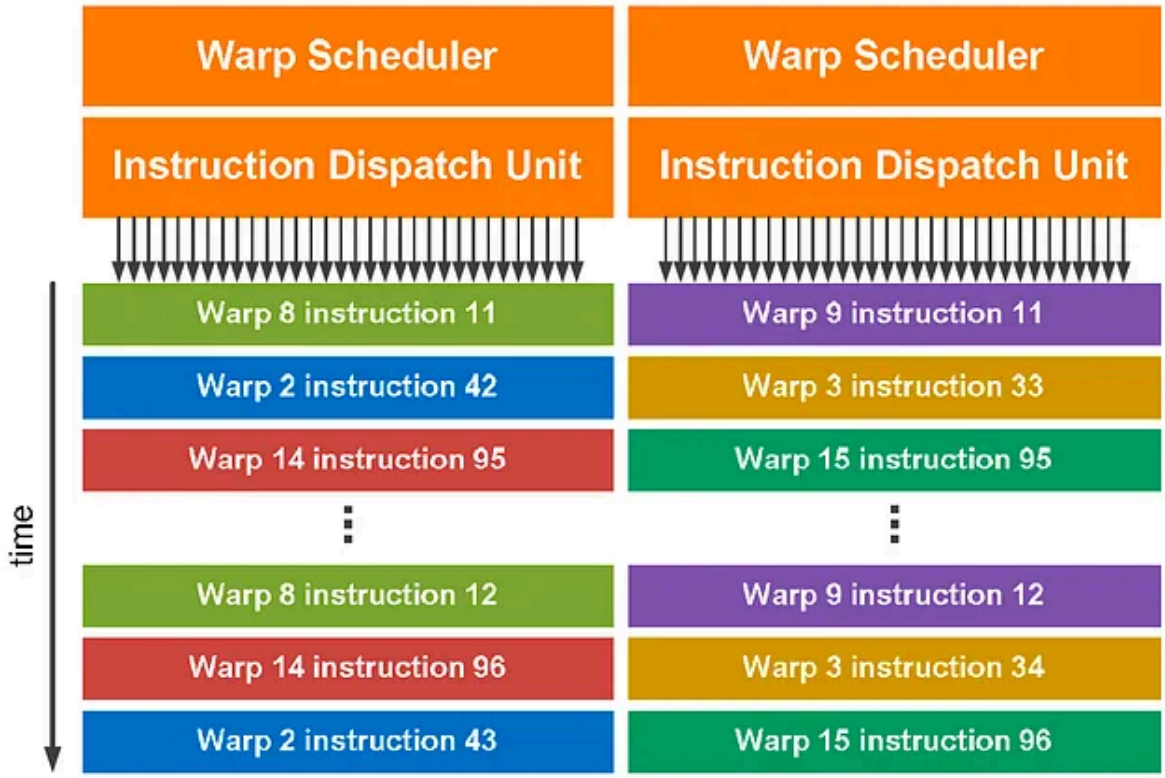

**Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).**

Fermi Architecture[1]

As shown in the following chart, every SM has 32 cuda cores, 2 Warp Scheduler and dispatch unit, a bunch of registers, 64 KB configurable shared memory and L1 cache. Cuda cores is the execute unit which has one float and one integer compute processor. The SM schedules threads in group of 32 threads called warps. The Warp Schedulers means two warps can be issued at the same time.


Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
LD/ST
SFU
CUDA Core
Dispatch Port
Operand Collector
FP Unit
INT Unit
Result Queue
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache
Fermi Streaming Multiprocessor (SM)

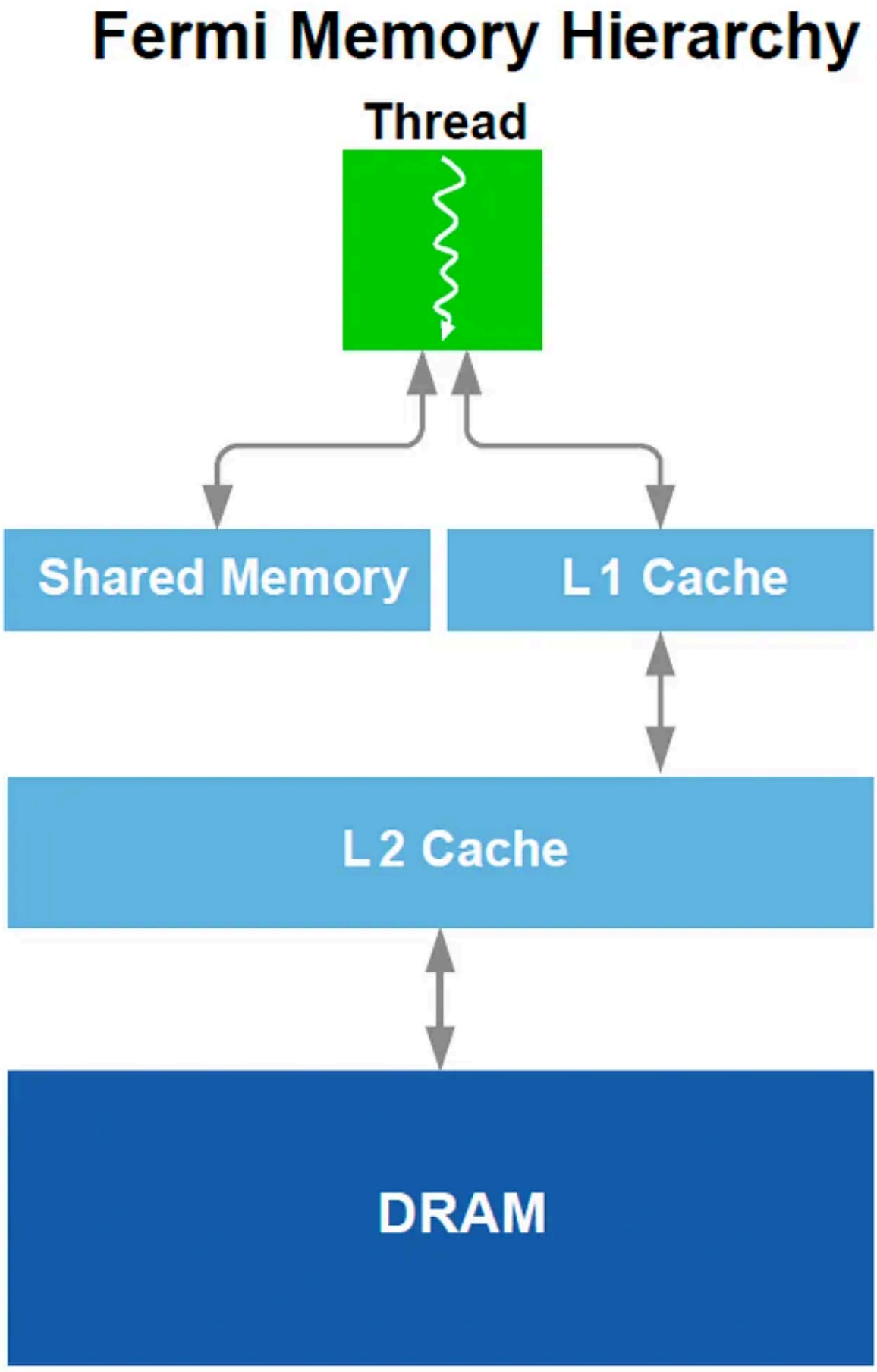
SM[1]




Threads are executing in warps [1]

### Memory Hierarchy

The fastest memory is registers just as in CPU. L1 cache and shared memory is second, which is also pretty limited in size. The SM above can have 48 KB shared memory and 16 KB L1 cache, or 16 shared memory and 48 L1 cache. L1 cache caches local and global memory, or only local memory varying among different GPU model. L2 cache caches local and global memory. Global memory acts like the ram in CPU computation, which is much slower than L1 and L2 cache.




Memory Hierarchy[1]

These are the specs of GRID K520, which is the GPU of AWS g2 instance. The architecture of this GPU is Kepler. Sorry, I don't have a Fermi CPU around. Here is the code to abtain the following information. If you compare it with the office specs of GRID K520, you can see it is a half card.

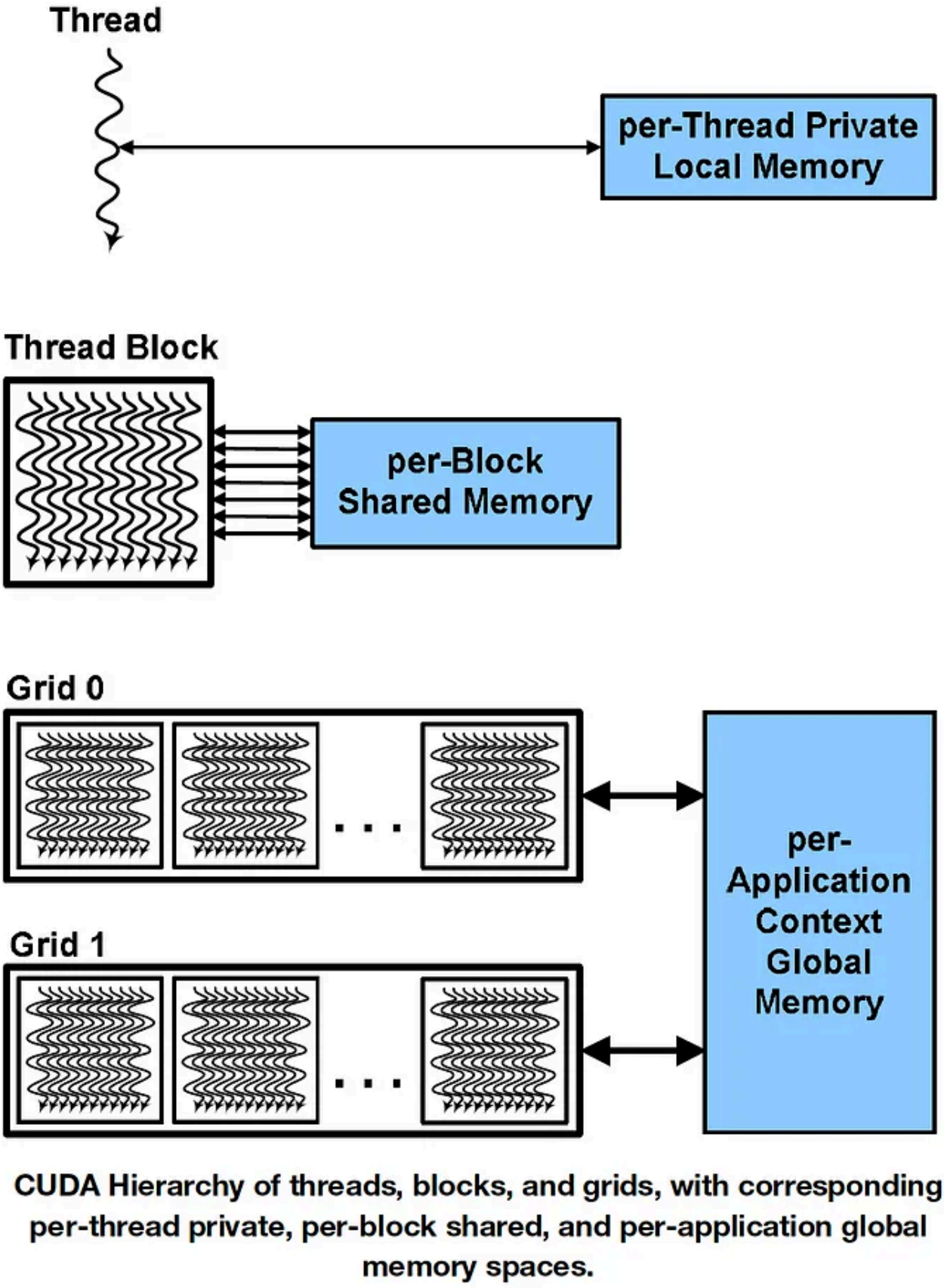
```
Device number: 0
  Device name: GRID K520
  Compute capability: 3.0

Clock Rate: 797000 kHz
  Total SMs: 8
  Shared Memory Per SM: 49152 bytes
  Registers Per SM: 65536 32-bit
  Max threads per SM: 2048
  L2 Cache Size: 524288 bytes
  Total Global Memory: 4232577024 bytes
  Memory Clock Rate: 2500000 kHz

Max threads per block: 1024
  Max threads in X-dimension of block: 1024
  Max threads in Y-dimension of block: 1024
  Max threads in Z-dimension of block: 64

Max blocks in X-dimension of grid: 2147483647
  Max blocks in Y-dimension of grid: 65535
  Max blocks in Z-dimension of grid: 65535

Shared Memory Per Block: 49152 bytes
  Registers Per Block: 65536 32-bit
  Warp size: 32
```

## Logical Hierarchies

### Computation Hierarchy

LOGICALLY, threads are organised in blocks, which are organised in grids. As a block executes in one SM, the number of blocks per grid is limited by SM. For Fermi and Kepler, one block can have 1024 threads maximum. Threads in a block are split into warps to execute in the same SM. The number of grids is limited by the global memory size.

### Memory Hierarchy




**CUDA Hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces.**

Memory Hierarchy [1]

Shared memory is allocated per block. The shared memory per block is limited by the shared memory per SM. If one block uses 24kb shared memory in the above K420 GPU, two blocks may stay in the same SM.

The shared memory is split into banks. Here is the best explanation I've seen[3].

> For nvidia (and amd for that matter) gpus the local memory is divided into memorybanks. Each bank can only address one dataset at a time, so if a halfwarp tries to load/store data from/to the same bank the access has to be serialized (this is a bank conflict). For gt200 gpus there are 16 banks (32banks for fermi), 16 or 32 banks for AMD gpus (57xx or higher: 32, everything below: 16)), which are interleaved with a granuity of 32bit (so byte 0-3 are in bank 1, 4-7 in bank 2, ..., 64-69 in bank 1 and so on). For a better visualization it basically looks like this:
>
> ```
> Bank    |      1      |      2      |      3      |...
> Address |  0  1  2  3 |  4  5  6  7 |  8  9 10 11 |...
> Address | 64 65 66 67 | 68 69 70 71 | 72 73 74 75 |...
> ...
> ```
>
> So if each thread in a halfwarp accesses successive 32bit values there are no bank conflicts. An exception from this rule (every thread must access its own bank) are broadcasts: If all threads access the same address, the value is only read once and broadcasted to all threads (for GT200 it has to be all threads in the halfwarp accessing the same address, iirc fermi and AMD gpus can do this for any number of threads accessing the same value).

It is important to understand this in order to write program avoiding banks confict.

The latest Volta architecture added tensor units and other features to boost neural network computation [4].

**References**


1. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

2.http://courses.cms.caltech.edu/cs179/2015_lectures/cs179_2015_lec05.pdf

3. https://stackoverflow.com/questions/3841877/what-is-a-bank-conflict-doing-cuda-opencl-programming

4. http://www.nvidia.com/object/volta-architecture-whitepaper.html


Cuda Gpu