Nvidia Turing GPU

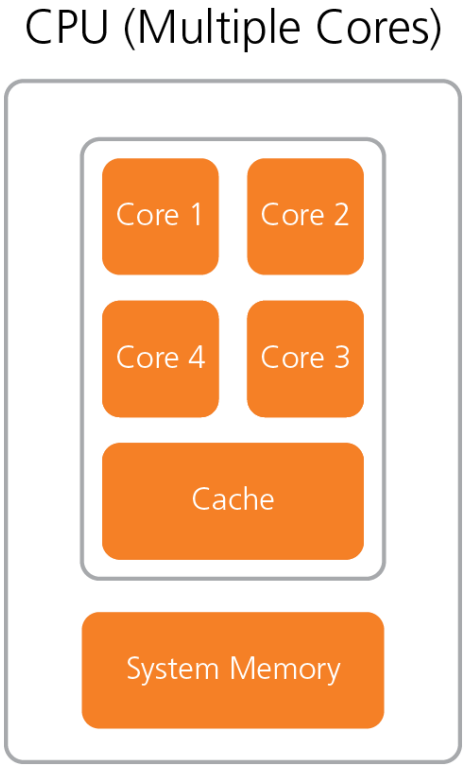
Haocheng Hua (hh7)

Hassan Dbouk (hdbouk2)

Mario Lopez Gonzalez (mariol4)

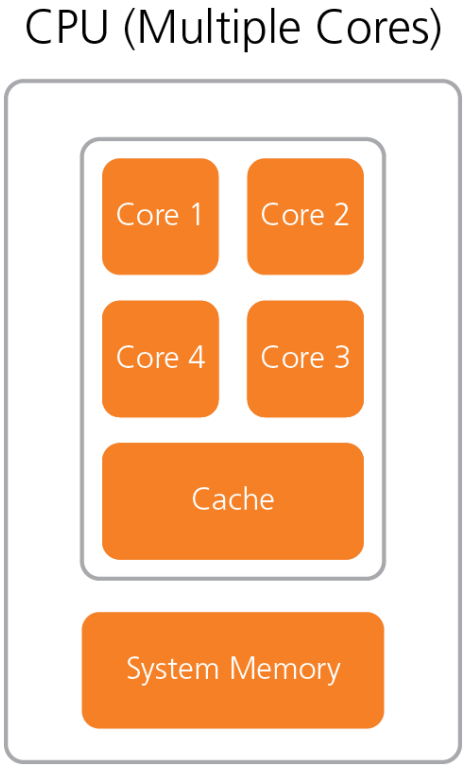
CPUs vs GPUs

- CPUs are *general* purpose processors
- They excel at performing *different* types of tasks that are *sequential* in nature
  - OS kernels, web browsing, text editing, …
- Often contain sophisticated control logic (branch prediction, scheduling, …) and massive caches (for lower latency)



CPUs vs GPUs

- CPUs are *general* purpose processors
- They excel at performing *different* types of tasks that are *sequential* in nature
  - OS kernels, web browsing, text editing, …
- Often contain sophisticated control logic (branch prediction, scheduling, …) and massive caches (for lower latency)



Jack of all trades, master of none!

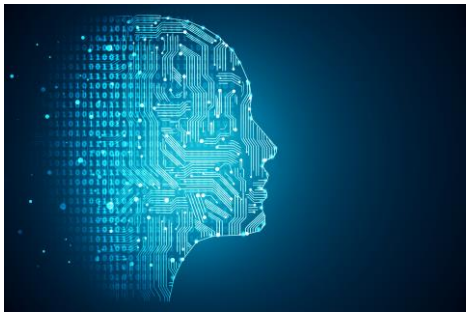
Emerging Applications

- Dedicated processors are required to support emerging applications such as:
  - High performance computing
  - Artificial intelligence – neural networks
  - Graphics rendering – gaming , CGI, multimedia editing

Rendering Graphics

High Performance Computing

Artificial Intelligence



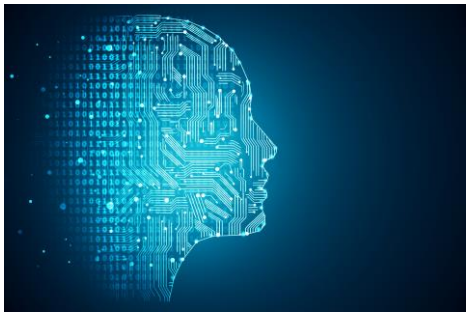
Emerging Applications

- Dedicated processors are required to support emerging applications such as:
  - High performance computing
  - Artificial intelligence – neural networks
  - Graphics rendering – gaming , CGI, multimedia editing

Rendering Graphics

High Performance Computing

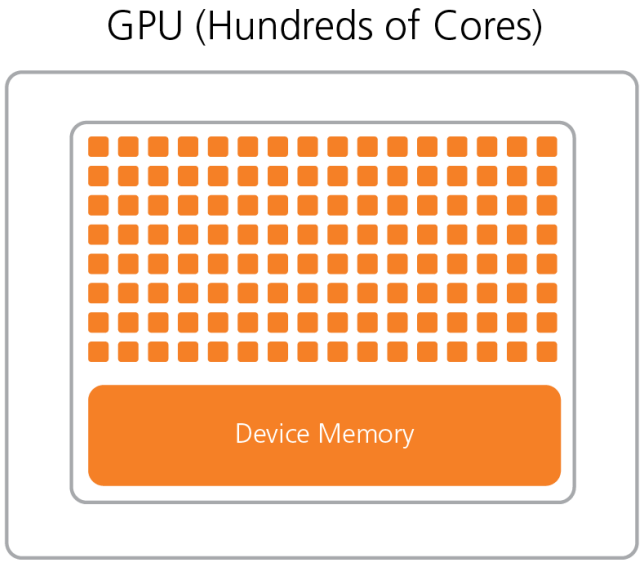
Artificial Intelligence



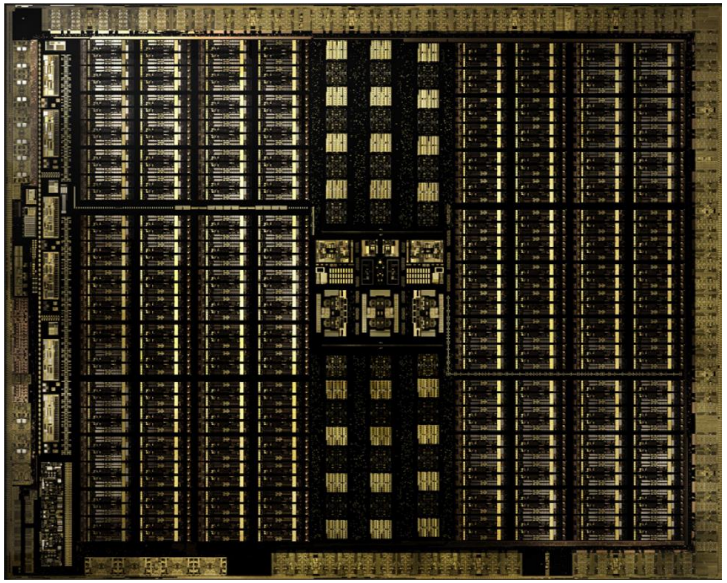
High data parallelism and repetitive operations!

GPUs – Overview

- Moderate clock frequency
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



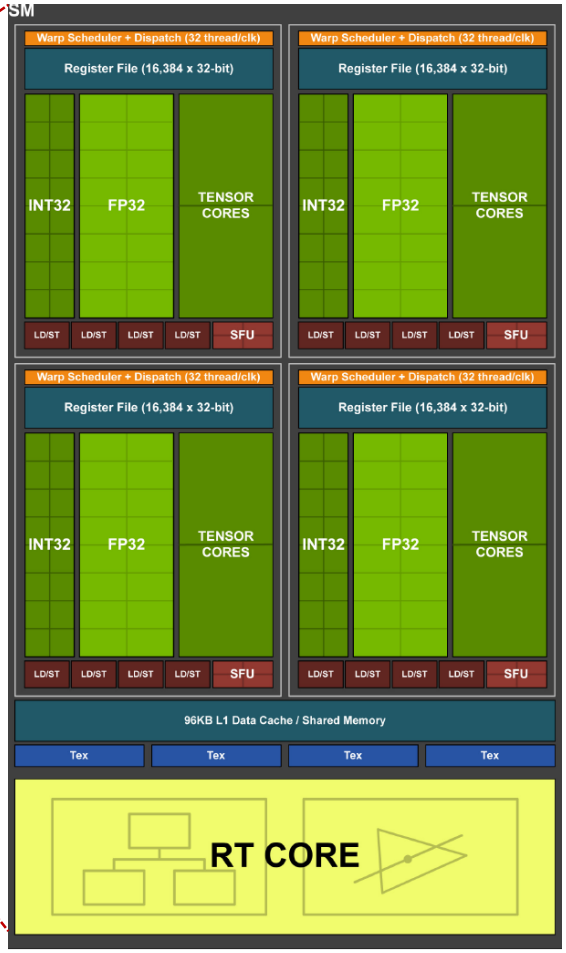
Nvida GPU: Turing (TU102) Overall Architecture



- Base Ref CLK Freq/ Boost CLK Freq – 1455MHz/1770MHz
- **18.6 billion transistors on TSMC's 12 nm FFN Process with 754mm2 die area**
- 72 Streaming Multiprocessor (SM)
- 12 X (1 Memory Controller – 512KB L2 Cache)
- 384-bit 7 GHz GDDR6

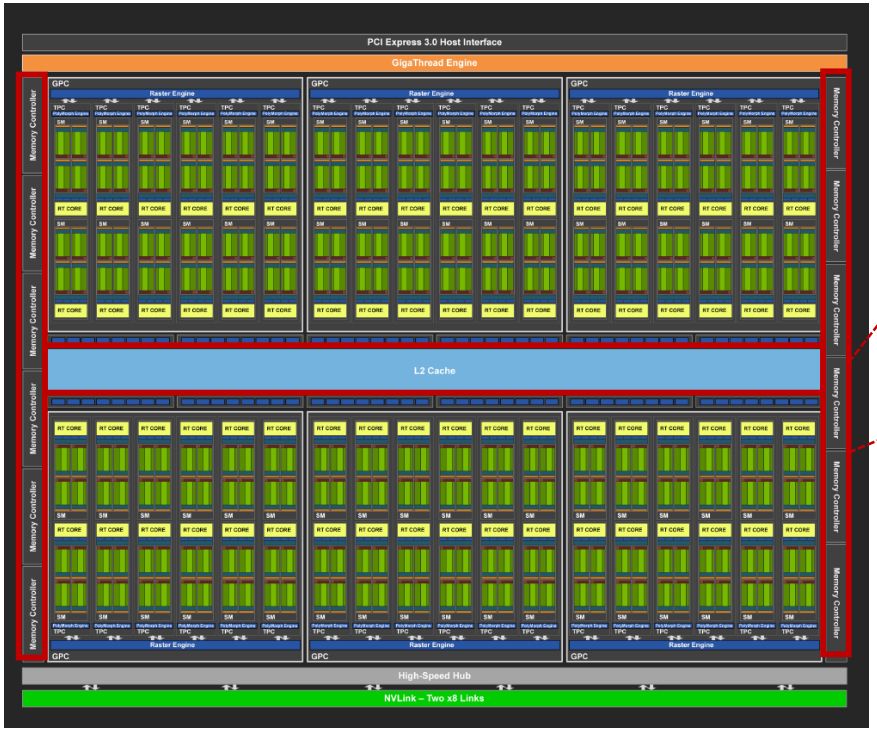
Ref [1]

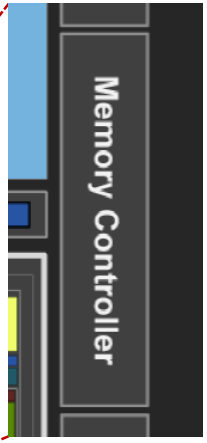
TU102 Overall Architecture

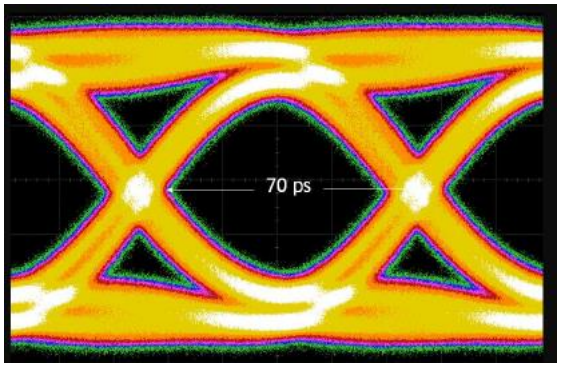


- Base Ref CLK Freq/ Boost CLK Freq – 1455MHz/1770MHz
- 18.6 billion transistors on TSMC's 12 nm FFN Process with 754mm2 die area
- **72 Streaming Multiprocessor (SM)**
- 12 X (1 Memory Controller – 512KB L2 Cache)
- 384-bit 7 GHz GDDR6

TU102 Overall Architecture





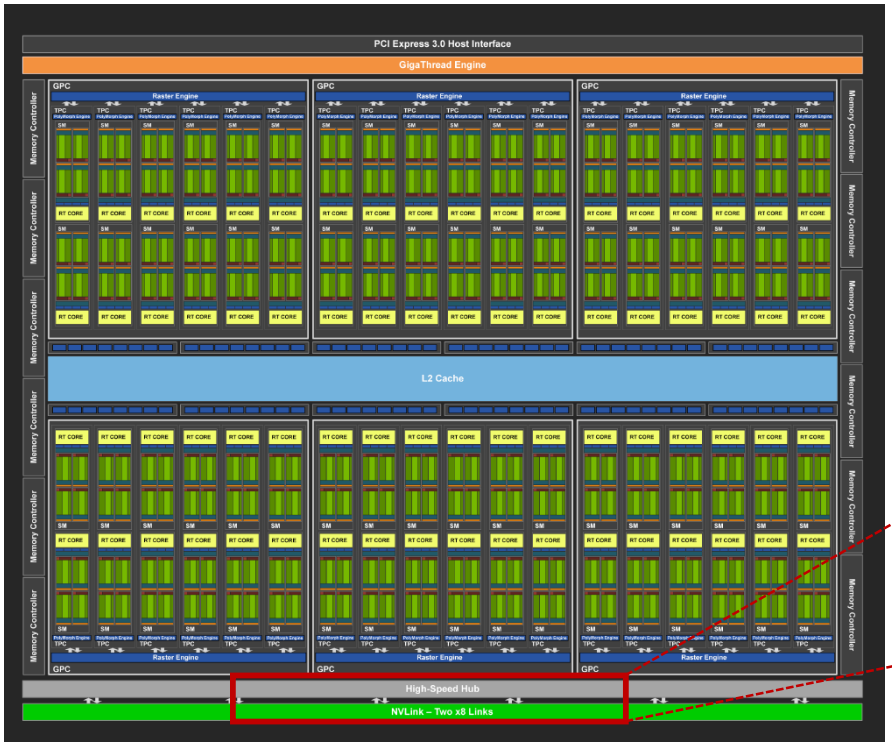


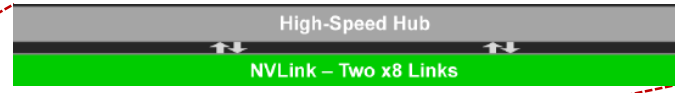
- Base Ref CLK Freq/ Boost CLK Freq – 1455MHz/1770MHz
- 18.6 billion transistors on TSMC's 12 nm FFN Process with 754mm2 die area
- 72 Streaming Multiprocessor (SM)
- **12 X (1 Memory Controller – 512KB L2 Cache)**
- **384-bit 7 GHz GDDR6**

TU102 Overall Architecture

- **PCI Express 3.0 Host Interface**
- **Giga Thread Engine**
- 2 NVLINK Channels
- Raster Engine / PolyMorph Engine / ROP Unit

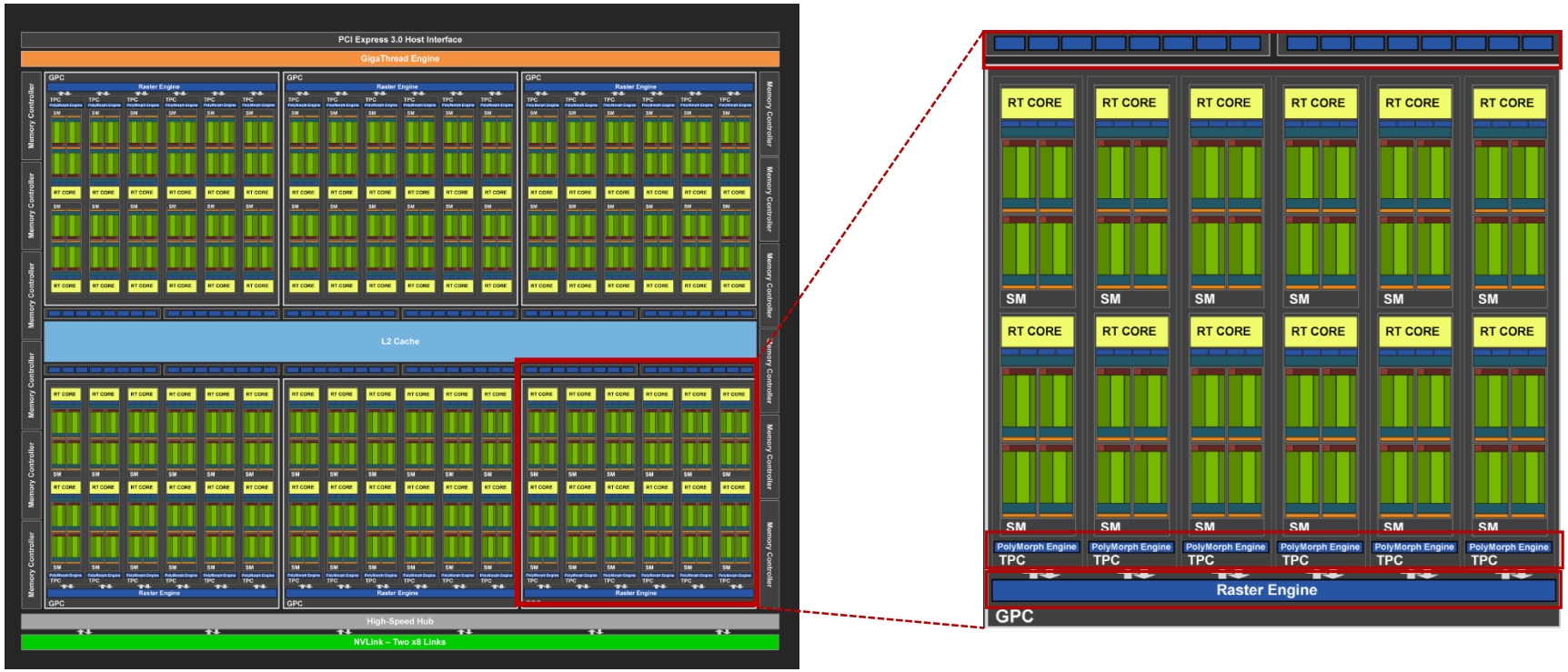
TU102 Overall Architecture





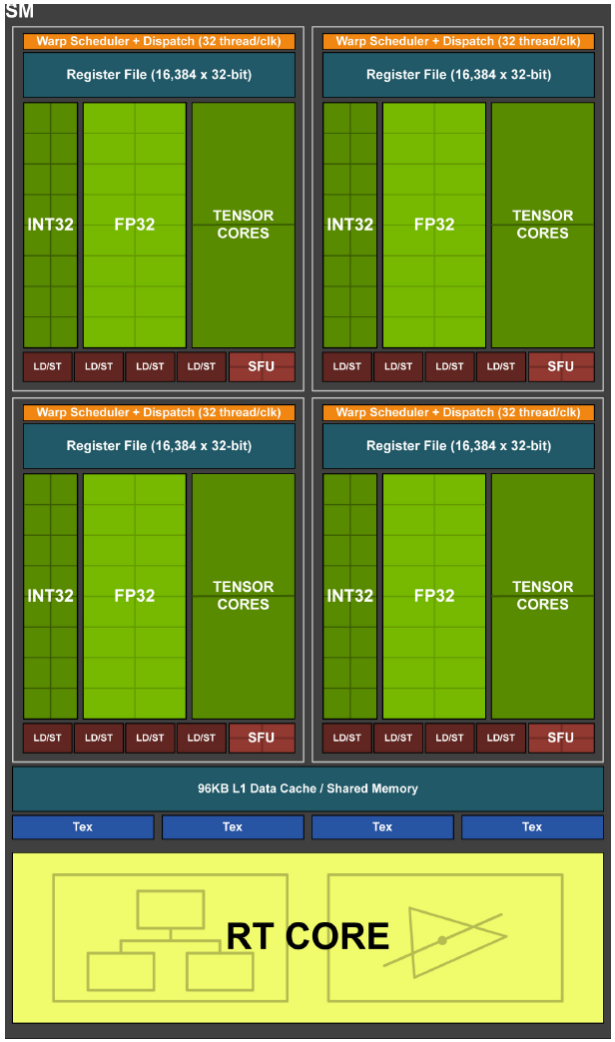
- PCI Express 3.0 Host Interface
- Giga Thread Engine
- **2 NVLINK Channels (50GB/sec bidirectional bandwidth per link)**
- Raster Engine / PolyMorph Engine / ROP Unit

TU102 Overall Architecture



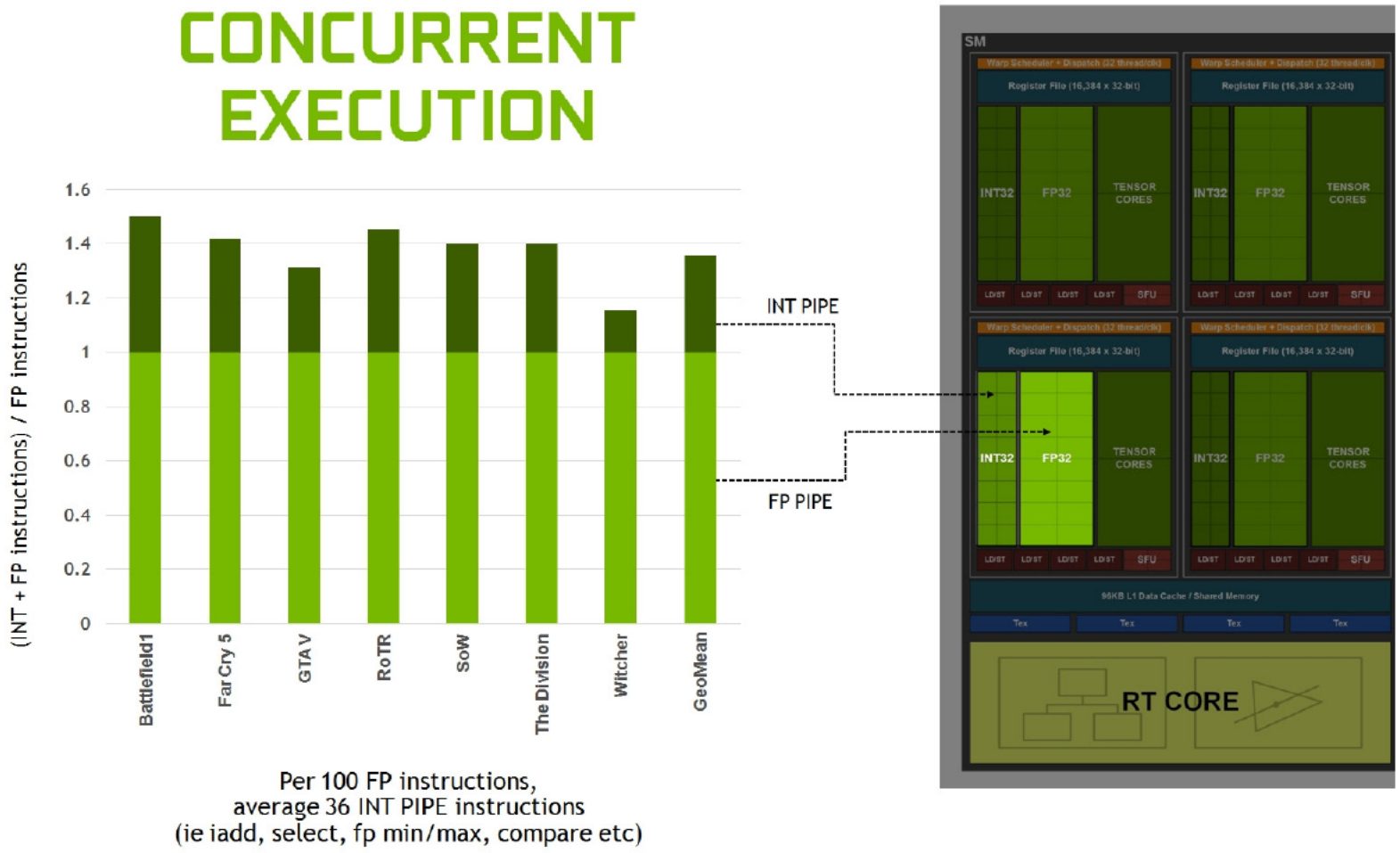
- PCI Express 3.0 Host Interface
- Giga Thread Engine
- 2 NVLINK Channels
- **Raster Engine / PolyMorph Engine / ROP Unit**

SM Architecture – Key Features



- 4 Processing Blocks(PB)
- 4 X 64 KB Register File
- 96KB Configurable Unified L1 Cache/Shared Memory(64KB+32KB or 32KB+64KB)
- 4 X 4 LD/ST Units
- 4 X 1 Special Function Unit(SFU), sin, cos, etc.
- RT CORE + 4 Texture Unit(TMU)
- 4 X Warp Scheduler + Dispatch(32 Thread/Clk)
- 4 X 16 Int32/16 Fp32 Datapath
- 4 X 2 Tensor Cores

New Feature - Concurrent FP and Int Execution



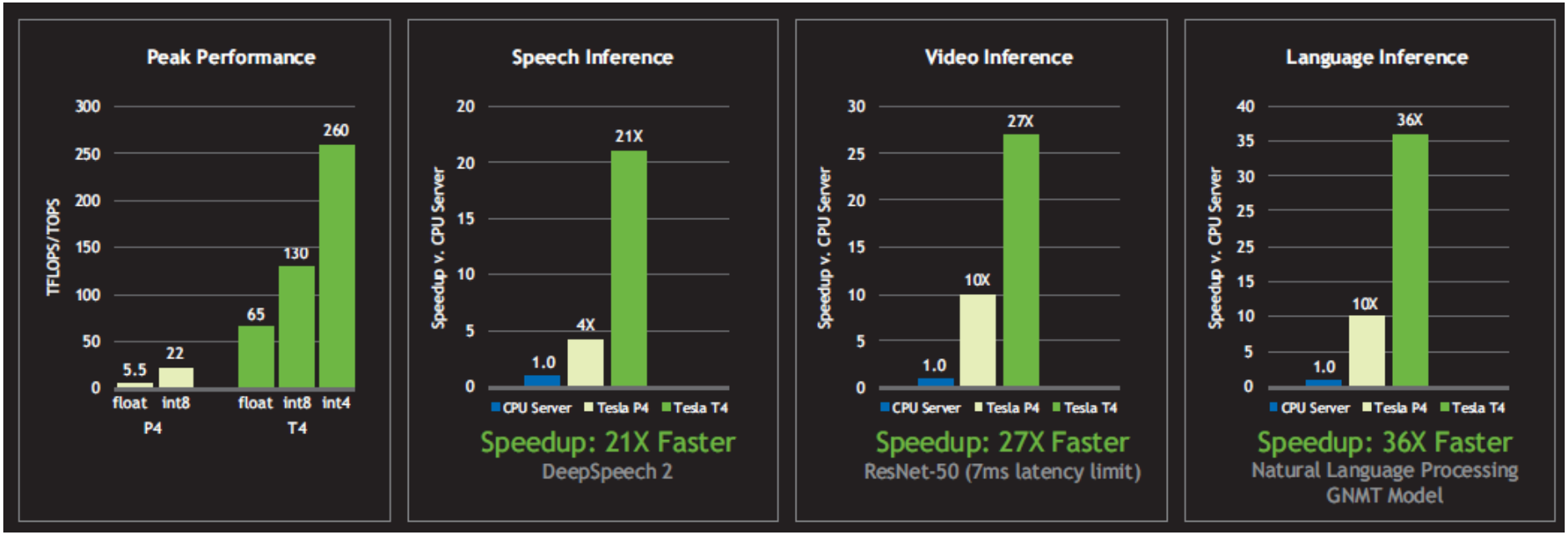
14 TFLOPS + 14 TIOPS

New Feature – Tensor Cores with Int8 and Int4 precision model

Precision Model	Throughput(TOPS)
FP 16	144
Int 8	288
Int 4	455

Ref [5]

Speedup compared with Pascal GPU and CPU Server



Ref [5]

Nvidia GPU: Programming & ISA

- Nvidia GPUs can be programmed using CUDA (**C**ompute **U**nified **D**evice **A**rchitecture), a C++ like language developed by Nvidia.
- GPU computation relies on multi-threaded programming, which explains why CUDA programming is thread-centric.
- A compiler translates CUDA code into the Parallel Thread Execution (PTX) virtual ISA, which guarantees compatibility across generations of GPUs.
- PTX instructions describe the operations on a *single* CUDA Thread and usually map one-to-one with hardware instructions

Ref [3]

DAXPY: CUDA vs C

```
// Invoke DAXPY
daxpy(n, 2.0, x, y);
// DAXPY in C
void daxpy(int n, double a, double *x, double *y)
{
          for (int i = 0; i < n; ++i)
               y[i] = a*x[i] + y[i];
}
```

Using C for CPU

```
// Invoke DAXPY with 256 threads per Thread Block
__host__
int nblocks = (n+ 255) / 256;
   daxpy<<<nblocks, 256>>>(n, 2.0, x, y);
// DAXPY in CUDA
__global__
void daxpy(int n, double a, double *x, double *y)
{
   int i = blockIdx.x*blockDim.x + threadIdx.x;
   if (i < n) y[i] = a*x[i] + y[i];
}
```

Using CUDA for Nvidia GPU

Ref [3]

DAXPY: Using PTX

```
shl.u32 R8, blockIdx, 8    ; Thread Block ID * Block size
                           ;(256 or 2^8)
add.u32 R8, R8, threadIdx  ; R8 = i = my CUDA Thread ID
shl.u32 R8, R8, 3          ; byte offset
ld.global.f64 RD0, [X+R8]; RD0 = X[i]
ld.global.f64 RD2, [Y+R8]; RD2 = Y[i]
mul.f64 RD0, RD0, RD4      ; Product in RD0 = RD0 * RD4
                           ; (scalar a)
add.f64 RD0, RD0, RD2      ; Sum in RD0 = RD0 + RD2 (Y[i])
st.global.f64 [Y+R8], RD0; Y[i] = sum (X[i]*a + Y[i])
```

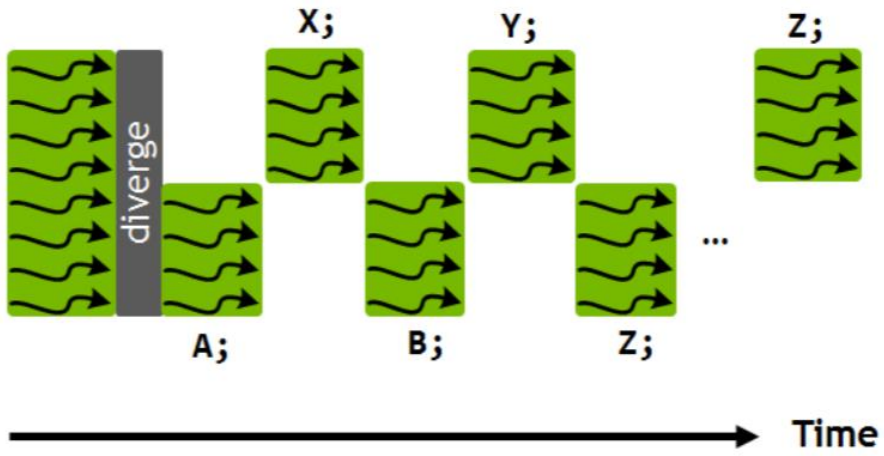
- Each thread executes the above code
- The conditional branch code is omitted for simplicity

Ref [3]

Multi Threading

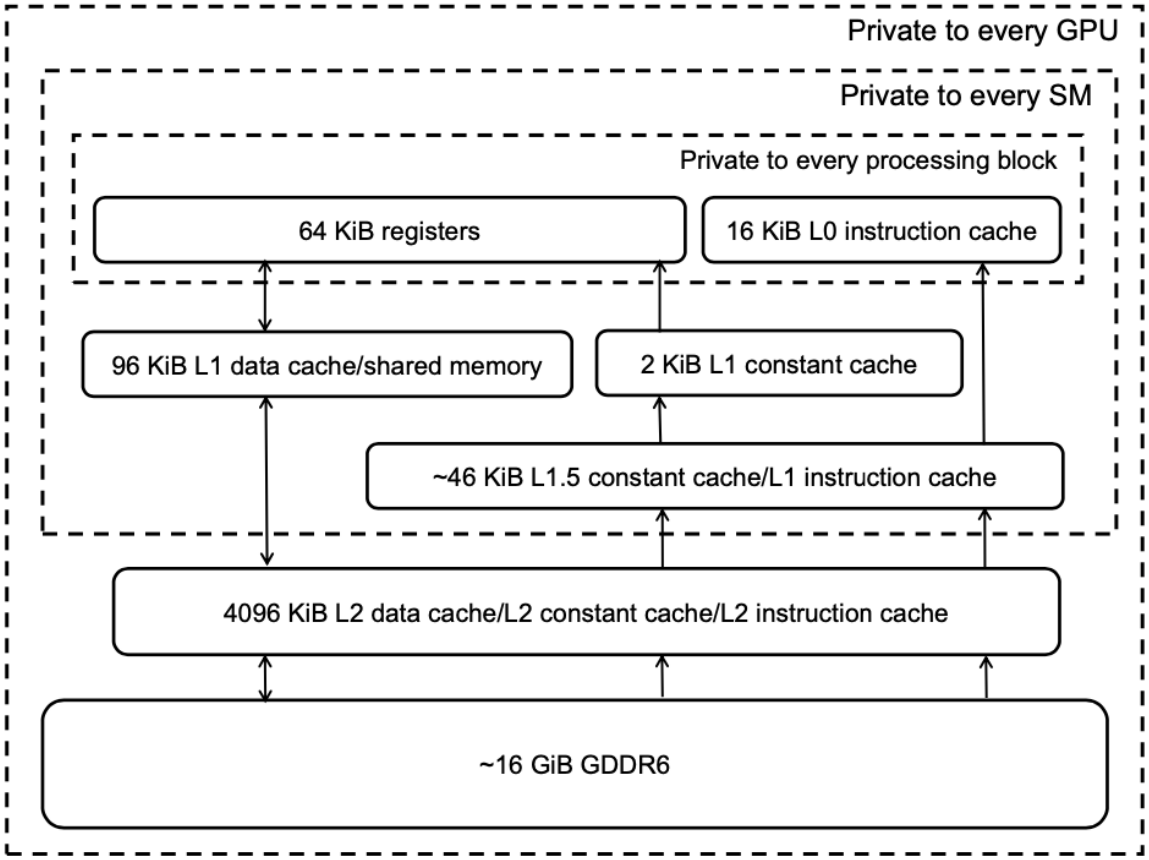
- GPUs can be classified as single-instruction multiple-thread (SIMT)
- Independent thread scheduling allows the GPU to yield execution of any thread

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



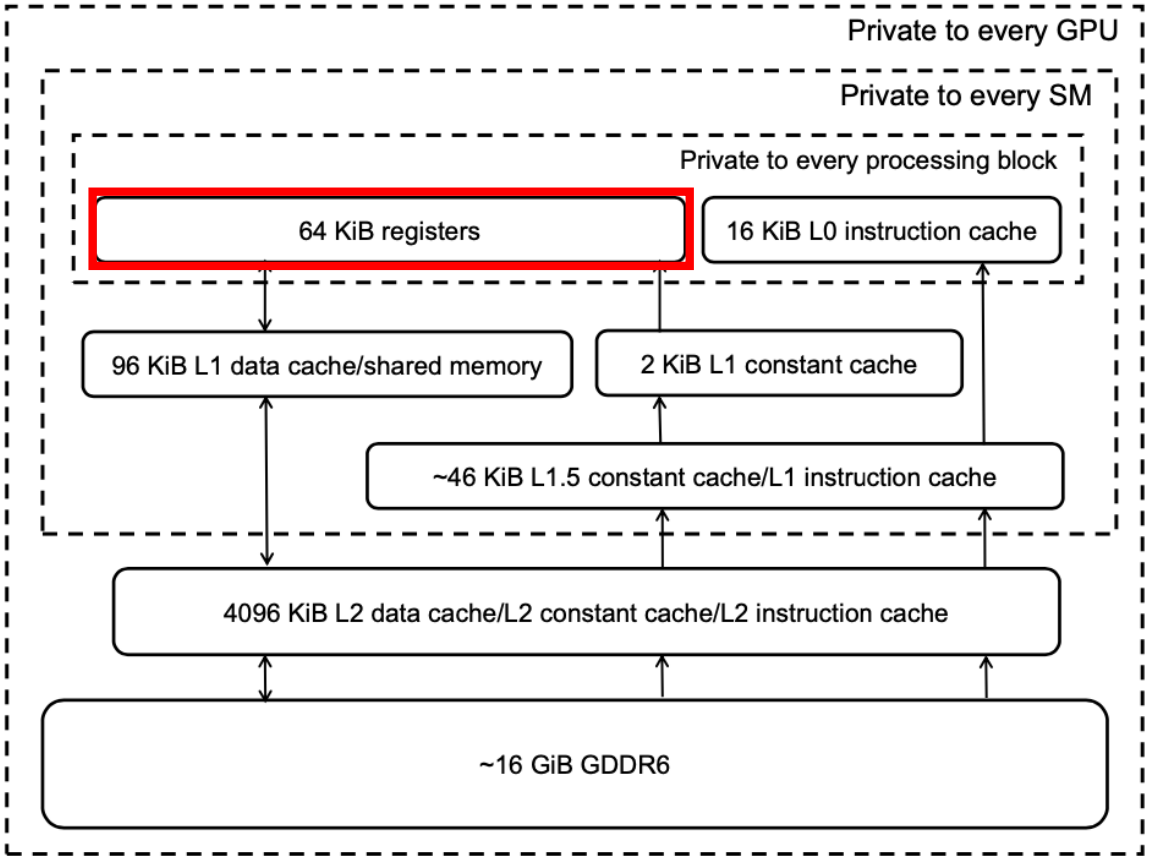
Ref [2]

GPU Memory hierarchy



Ref [4]

Registers


Private to every GPU
Private to every SM
Private to every processing block
64 KiB registers
16 KiB L0 instruction cache
96 KiB L1 data cache/shared memory
2 KiB L1 constant cache
~46 KiB L1.5 constant cache/L1 instruction cache
4096 KiB L2 data cache/L2 constant cache/L2 instruction cache
~16 GiB GDDR6

Registers

- Two types of registers
  - **Uniform Registers** (separate, integer-only, scalar datapath in parallel with main datapath)
  - **Regular Registers**
- Register files divided in **two banks with dual 32-bit ports** each.
- Each port only supports **one 32-bit read per cycle**.

- Problem?

Ref [1]

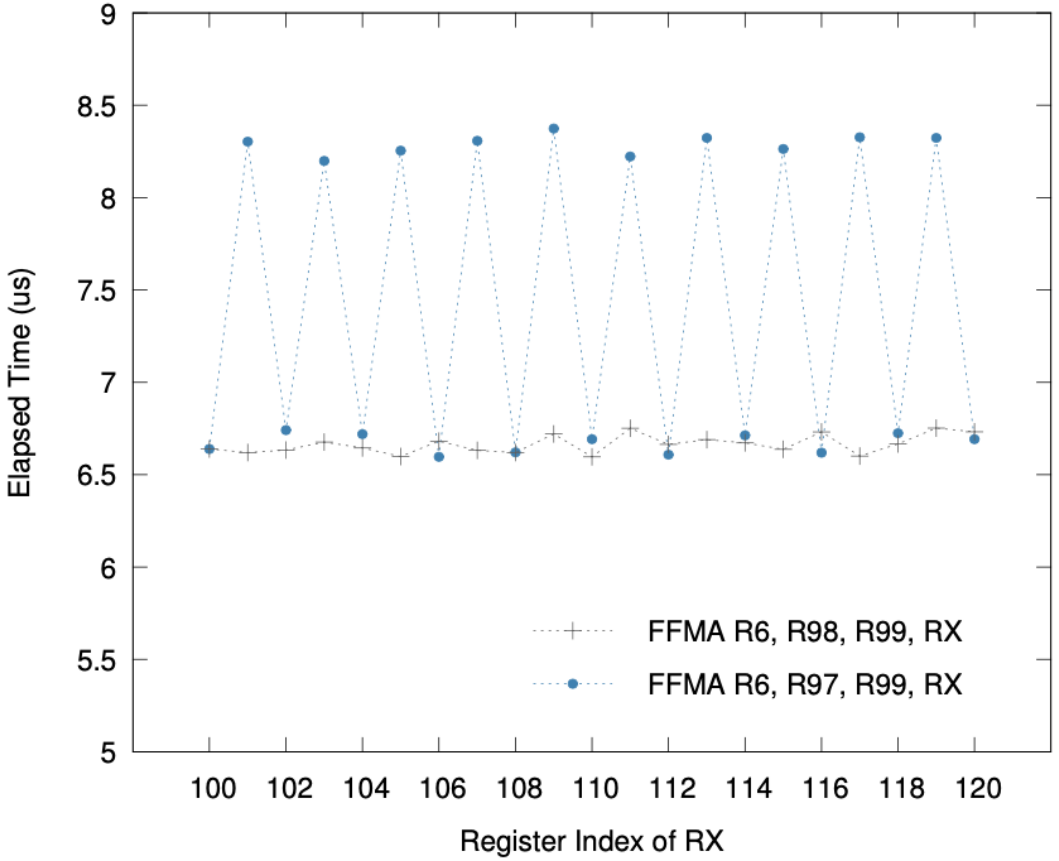
Registers

- Two types of registers
  - **Uniform Registers** (separate, integer-only, scalar datapath in parallel with main datapath)
  - **Regular Registers**
- Register files divided in **two banks with dual 32-bit ports** each.
- Each port only supports **one 32-bit read per cycle**.

- Problem?
  - Instructions requiring 3 operands (e.g. FFMA) will suffer a stall if the three operands map to the same bank due to a *register bank conflict*

Ref [1]

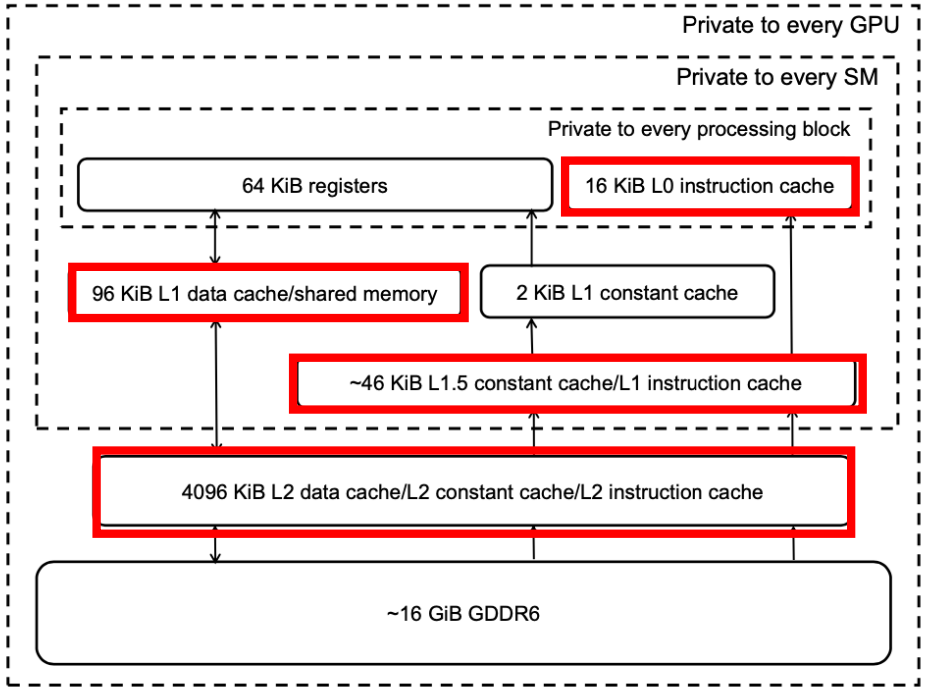
Register Bank Conflict



Ref [4]

Cache

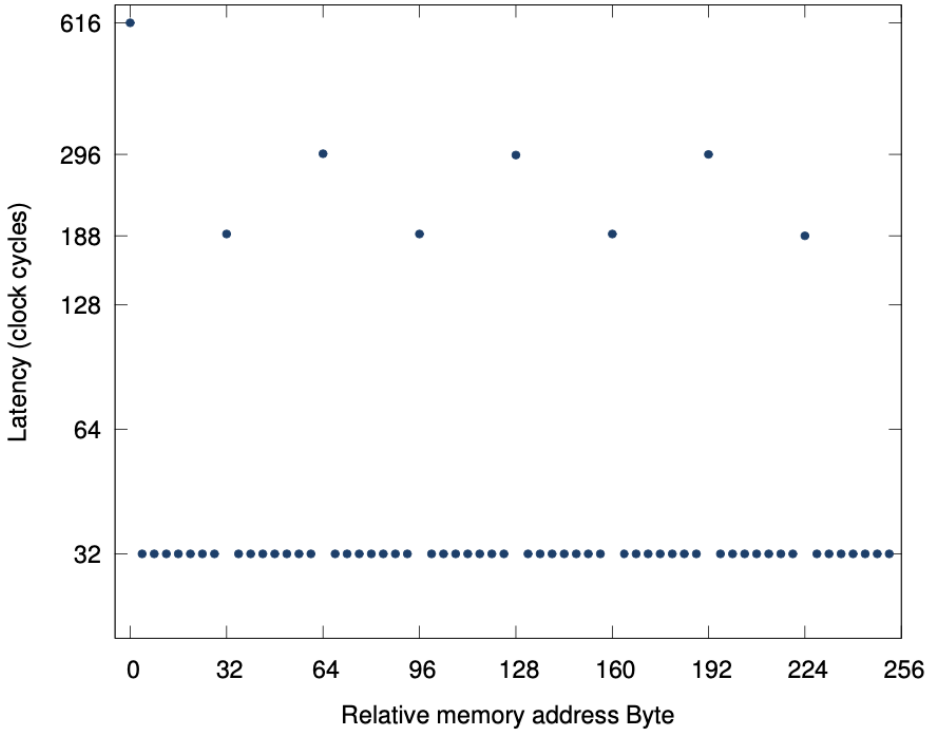
- **Non-LRU** replacement policy
  - It aims at preserving large arrays from eviction caused by sparse memory access
- L1 data cache can be either 32 or 64 KiB (**configurable by the user**)
- L1 data cache line size: 16 B
- L1 data cache indexed by **virtual addresses**
- L2 cache **16-way set associative**
- L2 data cache line size: 32 B
- L2 data cache indexed by **physical addresses**



Ref [4]

Cache Performance

- L1 cache hit: 32 cycles
- L2 cache hit: 188 cycles
- L2 miss and TLB hit: 296 cycles
- L2 miss and TLB miss: 616 cycles

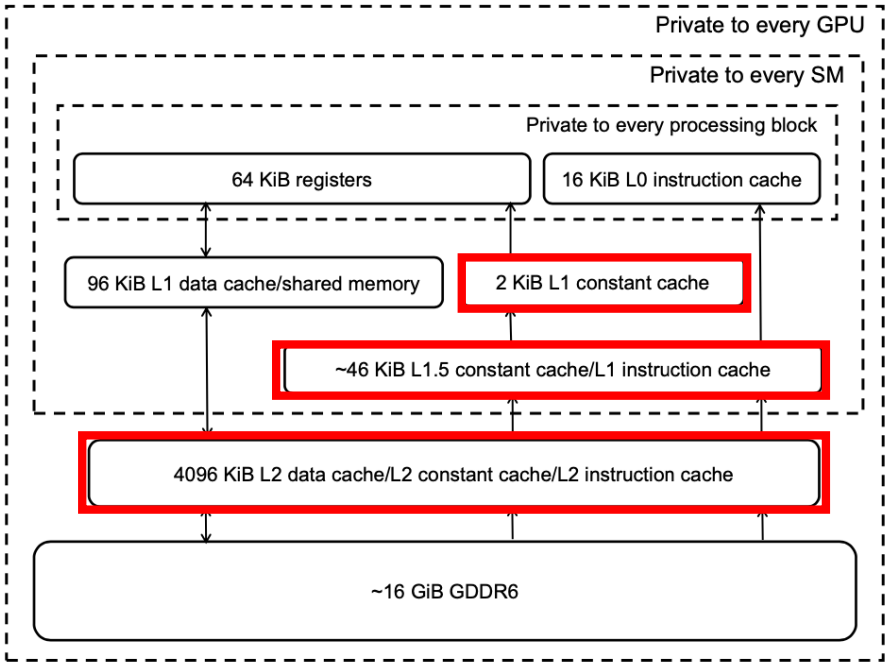


Ref [4]

Constant Memory

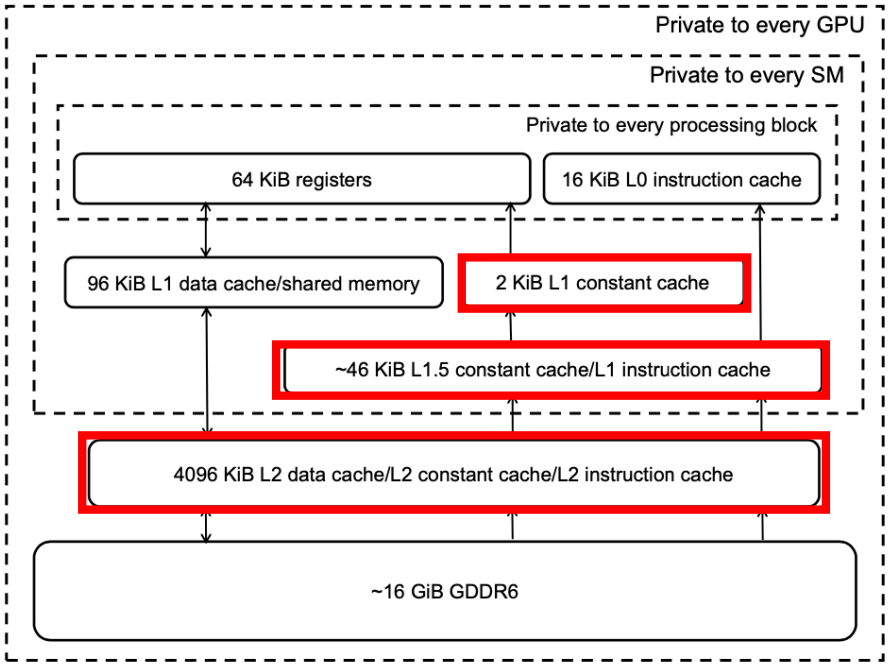
- What is the constant memory?

- **3 levels** of constant cache
- L1 uses a **non-LRU** replacement policy
- It supports **broadcasting**, when all threads within a warp access:
  - The same address: data is sent **simultaneously**
  - Diverging addresses: the accesses are **serialized**

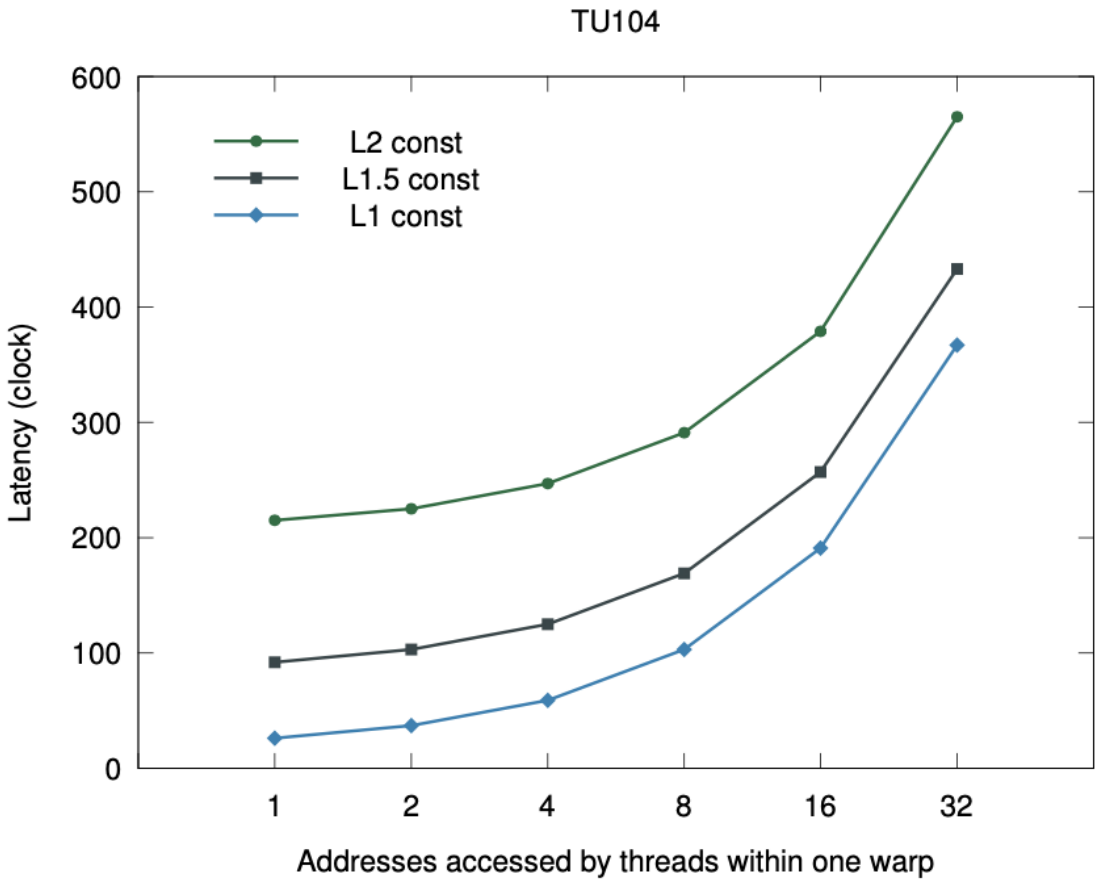


Constant Memory

- What is the constant memory?
  - __constant__ keyword
  - Kernel invocation parameters
  - Immediate constants
- **3 levels** of constant cache
- L1 uses a **non-LRU** replacement policy
- It supports **broadcasting**, when all threads within a warp access:
  - The same address: data is sent **simultaneously**
  - Diverging addresses: the accesses are **serialized**



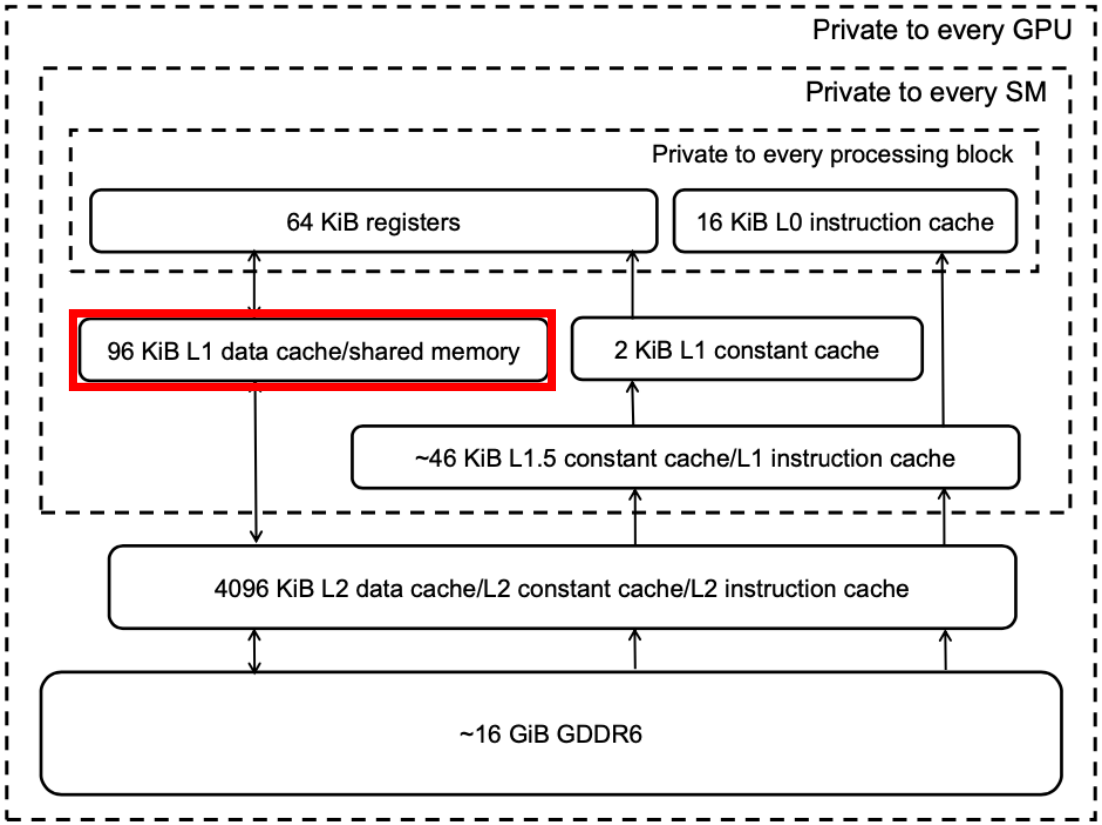
Performance of Constant Memory



Ref [4]

Shared Memory

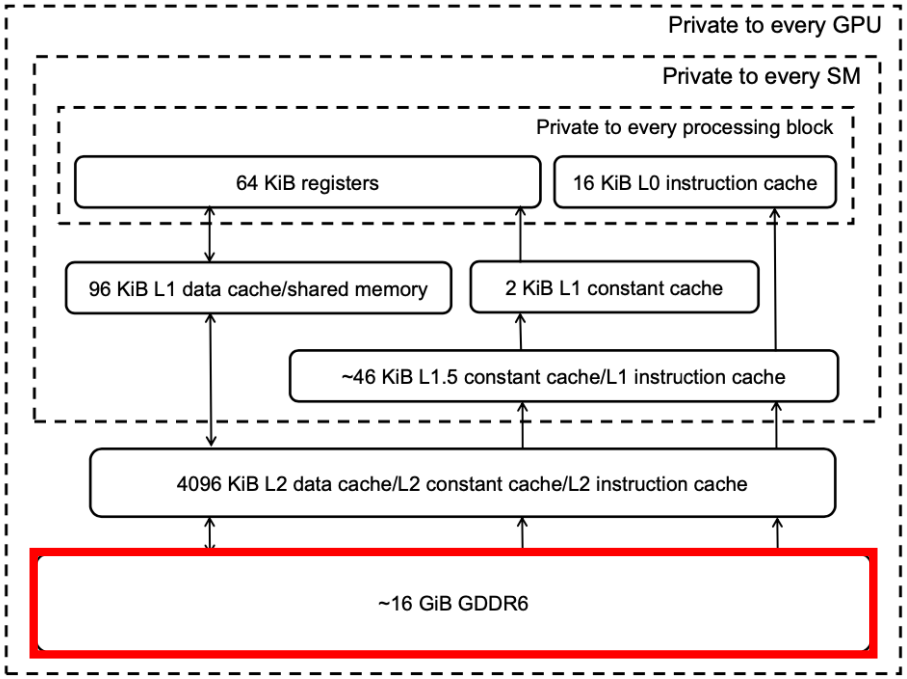
- **Shared among threads within a thread block**
- Either 64KiB or 32KiB (**configurable by the user**)

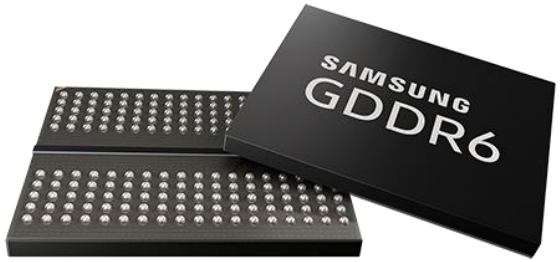


Global Memory

- **GDDR6 memory**
- **Lower bandwidth** but **higher clock rate** compared to previous GPUs using an HBM memory

- Size: 15,079 MiB
- Max Clock Rate: 5,001 MHz
- Theoretical Bandwidth: 320 GiB/s





References

- [1] Nvida Turing GPU Architecture: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
- [2] Nvida Tesla V100 (Volta) GPU Architecture: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
- [3] Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. 6th edition.
- [4] Jia, Zhe, et al. "Dissecting the NVidia Turing T4 GPU via Microbenchmarking." *arXiv preprint arXiv:1903.07486* (2019).
- [5] Burgess, John. "RTX ON–The NVIDIA TURING GPU." *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE, 2019.