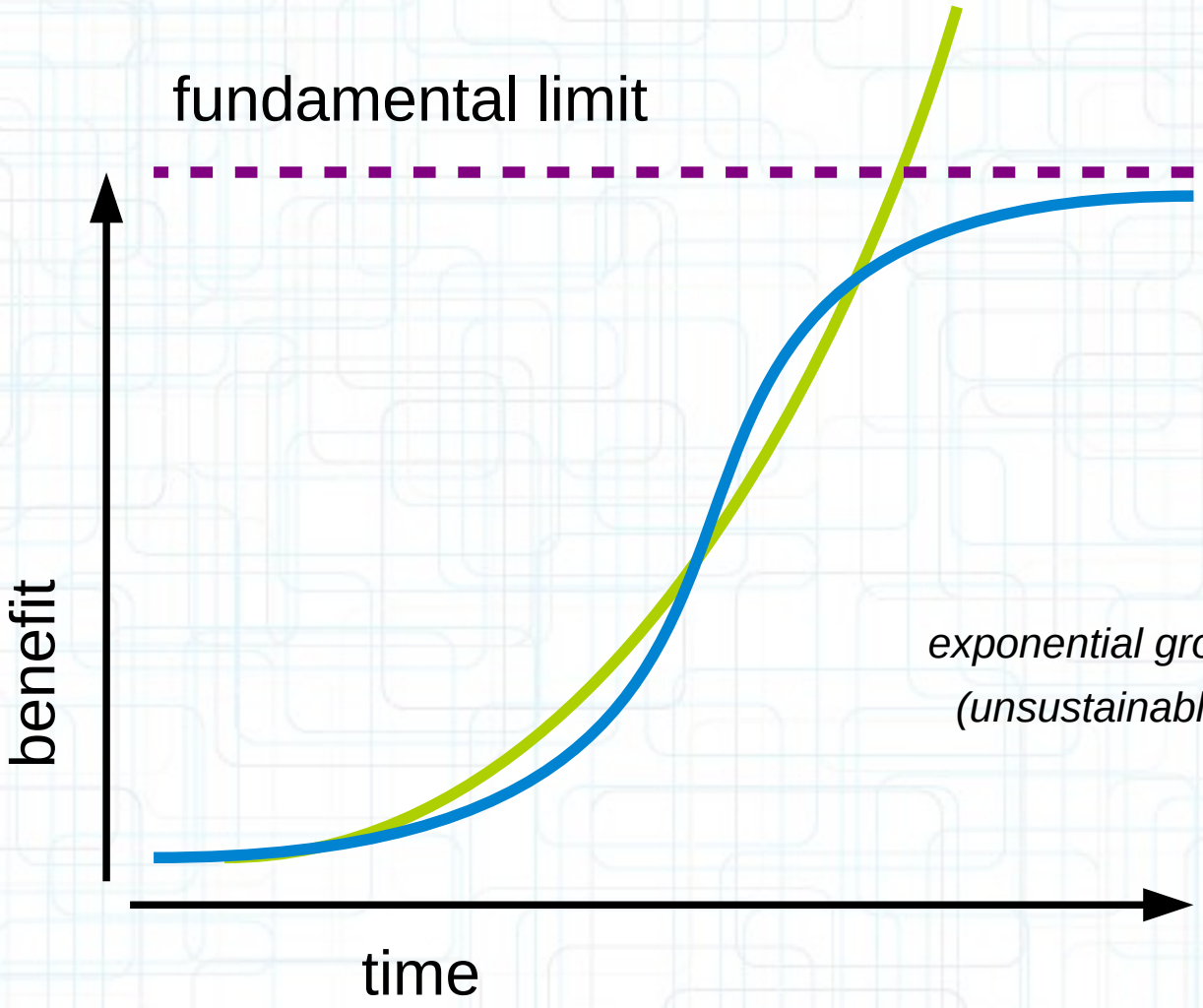
CS 594 – **Scientific Computing for Engineers**: Spring 2012

Accelerators

Jakub Kurzak
kurzak@eecs.utk.edu
http://icl.cs.utk.edu/~kurzak

The Technological S-curve



exponential growth = explosion
(unsustainable by definition)

What's wrong with CPUs?

- Power Wall – Power Dissipation
  - The chip will melt if running any faster (higher clock rate)
- Frequency Wall – Pipeline Depth
  - To crank up the clock shorter pipeline stages are required
  - To have shorter pipeline stages, more stages are required
  - When code branches, pipelines are flushed

    (there is not enough Instruction Level Parallelism in serial code)
- Memory Wall – DRAM Latency
  - DRAM can provide plenty of bandwidth, but very high latency
  - If data does not reside in cache, it can cost 1000 cycles to access it
  - Prefetching reached the point of diminishing returns

What is an accelerator?

British for *gas pedal*

What is an accelerator?

- A device that runs very fast?
  - Possibly very big?
  - Possibly very hot and power hungry?
- A device that is power efficient (high FLOPS / Watt)?
  - What if not very fast (e.g., the chip in your cell phone)?
- A device that is hard to program?
- A special purpose device (for specific workloads)?
- A device not capable of running:
  - Standard code (ANSI C serial code)?
  - Legacy code (legacy numerical libraries, LAPACK and alike)?
  - Standard operating system (Linux, Windows, Mac OS, …)?
- A device not capable of running them fast?

Cell Processor

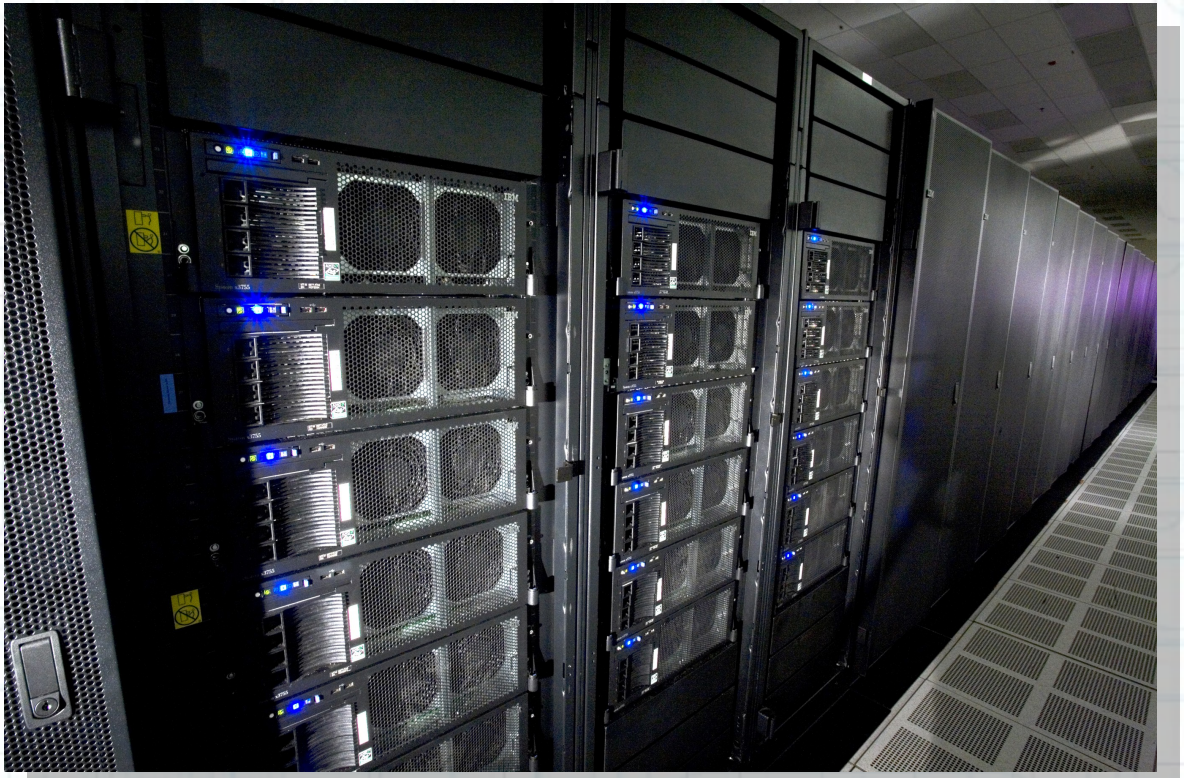
- Mercury PCI accelerator cards

- IBM server blades

- Sony Playstation 3

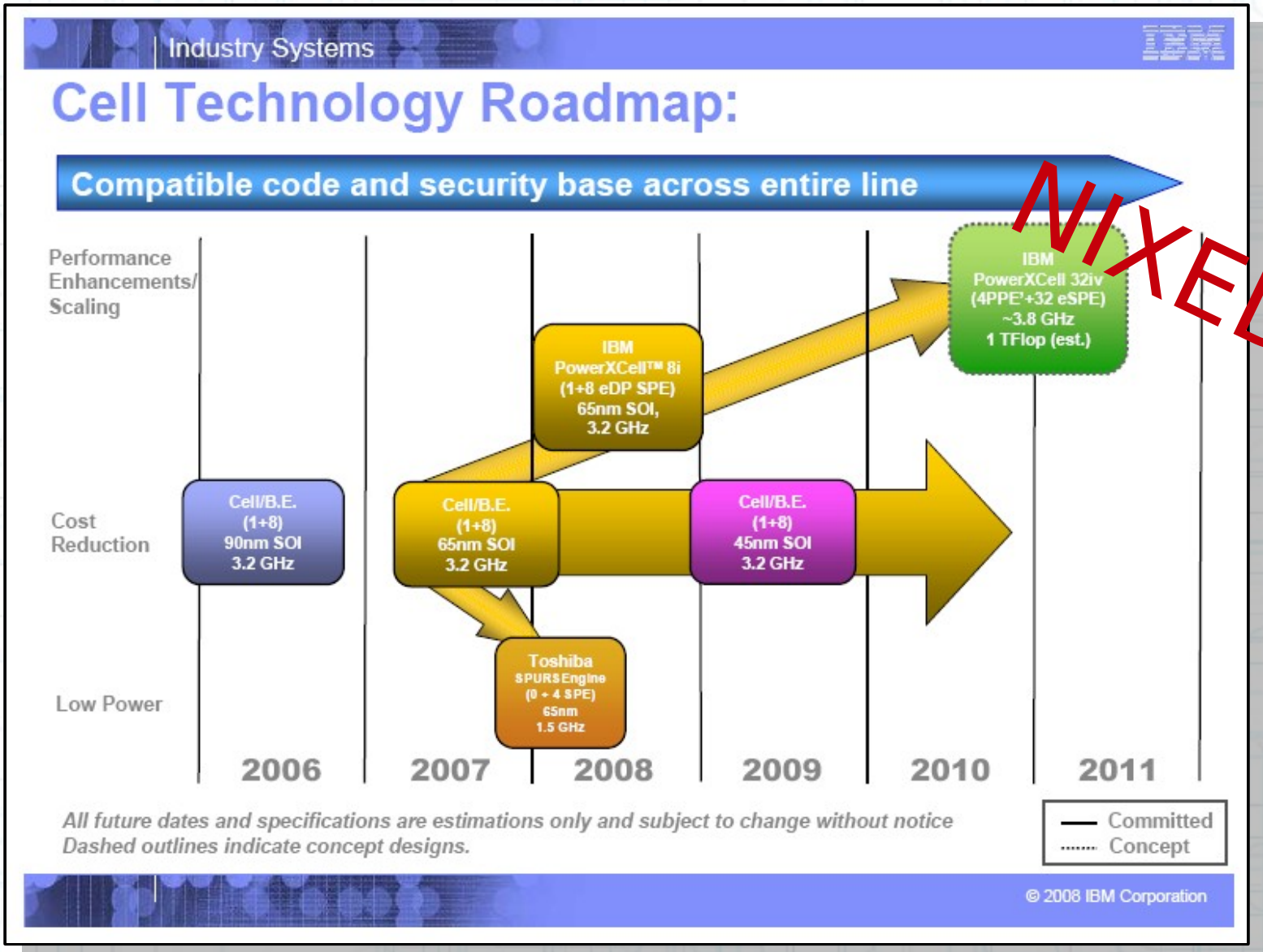
- Embedded systems

Roadrunner @ Los Alamos



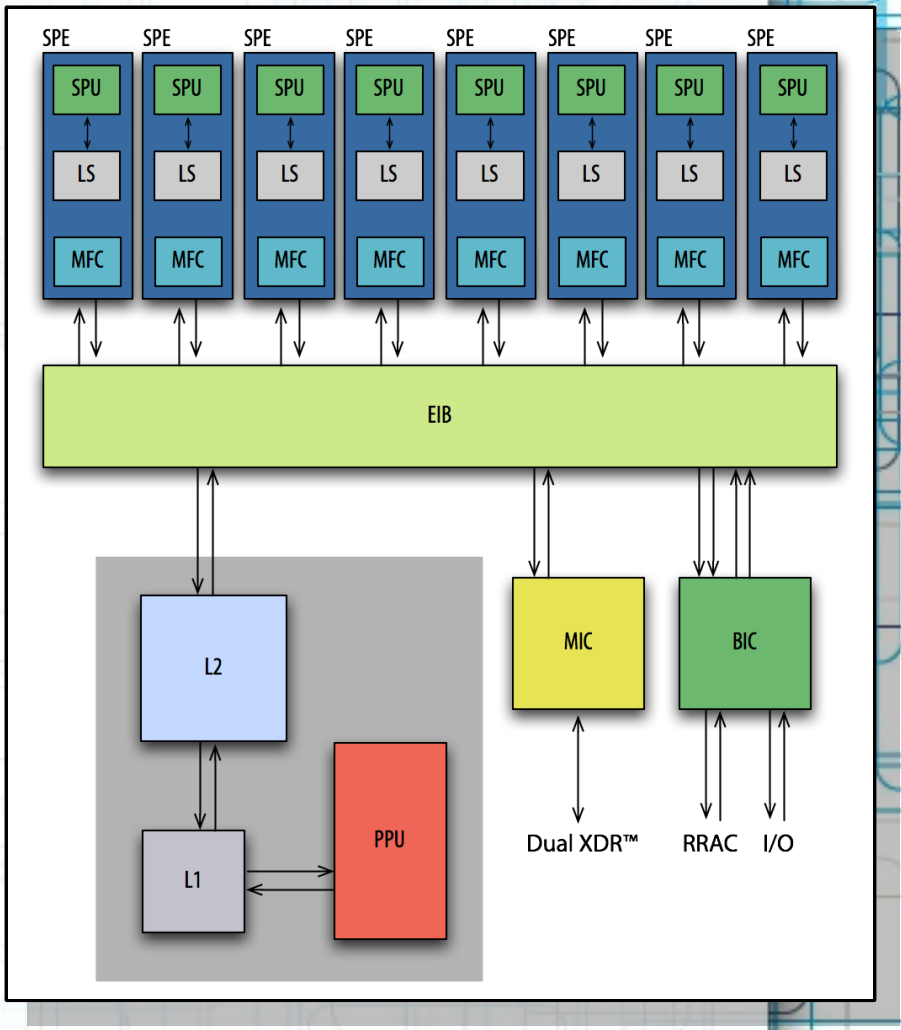
In June 2008 The Roadrunner supercomputer at Los Alamos National Laboratory crossed the performance of **1 PetaFLOPS** (10^{15} floating point operations per second) using **6,480 AMD Opteron** dual-core processors and **12,960 IBM PowerXCell 8i** processors.

Cell Demise


Industry Systems
IBM
Cell Technology Roadmap:
Compatible code and security base across entire line
NIXED!
Performance Enhancements/ Scaling
Cost Reduction
Low Power
Cell/B.E. (1+8) 90nm SOI 3.2 GHz
IBM PowerXCell™ 8i (1+8 eDP SPE) 65nm SOI, 3.2 GHz
Cell/B.E. (1+8) 65nm SOI 3.2 GHz
Toshiba SPURS Engine (0 + 4 SPE) 65nm 1.5 GHz
Cell/B.E. (1+8) 45nm SOI 3.2 GHz
IBM PowerXCell 32iv (4PPE'+32 eSPE) ~3.8 GHz 1 TFlop (est.)
2006
2007
2008
2009
2010
2011
All future dates and specifications are estimations only and subject to change without notice
Dashed outlines indicate concept designs.
Committed
Concept
© 2008 IBM Corporation

Cell Design

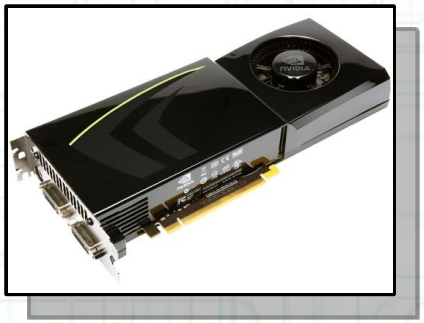
- **PPE** – Power Processing Element
- **SPE** – Synergistic Processing Element
  - **SPU** – Synergistic Processing Unit
  - **LS** – Local Store
  - **MFC** – Memory Flow Controller
- **EIB** – Element Interconnect Bus



Cell Challenges

- SIMD Vectorization – Exploiting Instruction Level Parallelism (ILP)
  - Fully SIMD architecture
  - No scalar registers
  - No scalar instructions
  - No SIMD = 1% performance
- Parallelization – Exploiting Thread Level Parallelism (TLP)
  - Octa-core architecture
  - No parallelization = 12.5% performance
- Communication – Exploiting Comm / Comput Overlapping
  - No cache coherency
  - “Software managed caches”

GPU

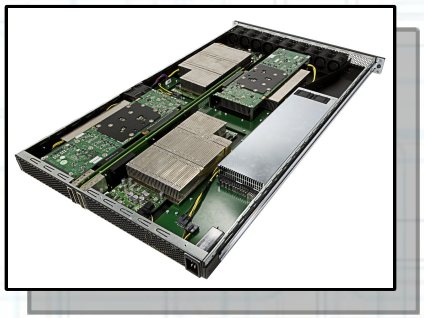


- PCI accelerators

- Multi-GPU Desk-side servers

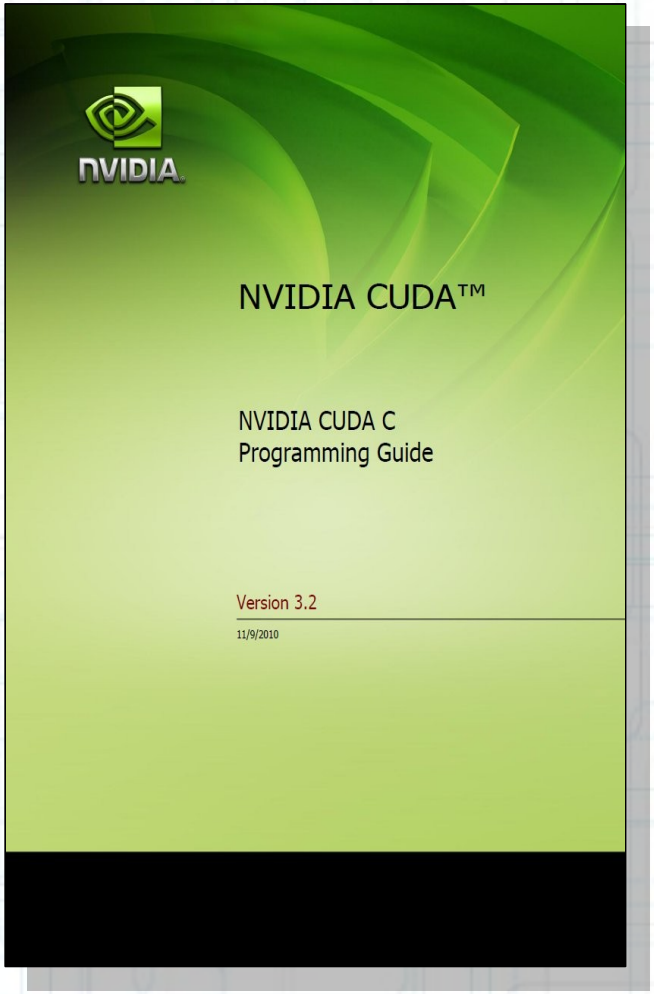
- Gaming consoles

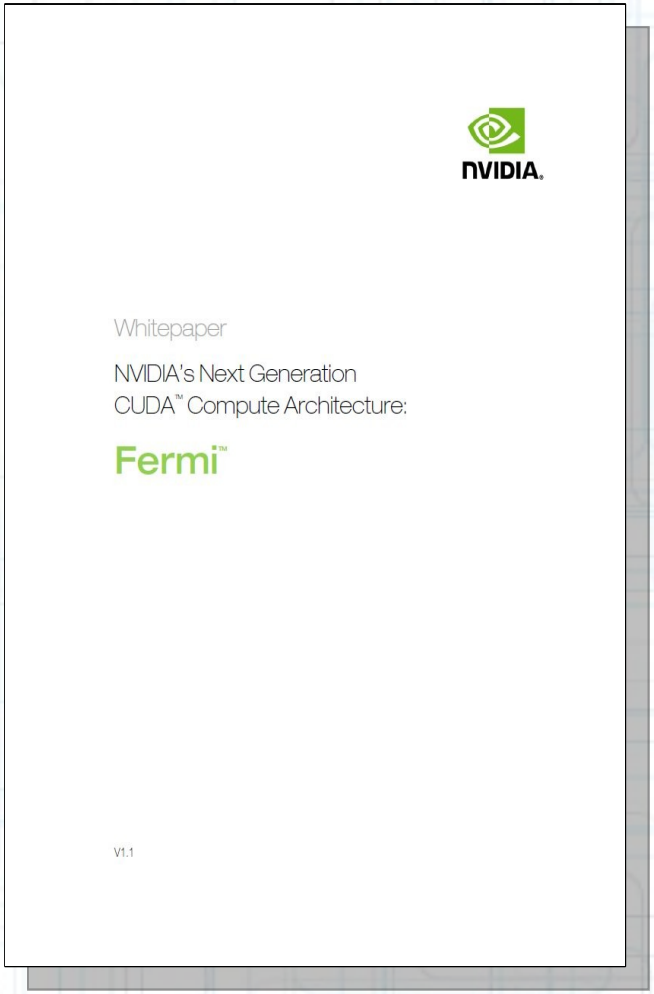
- Embedded system



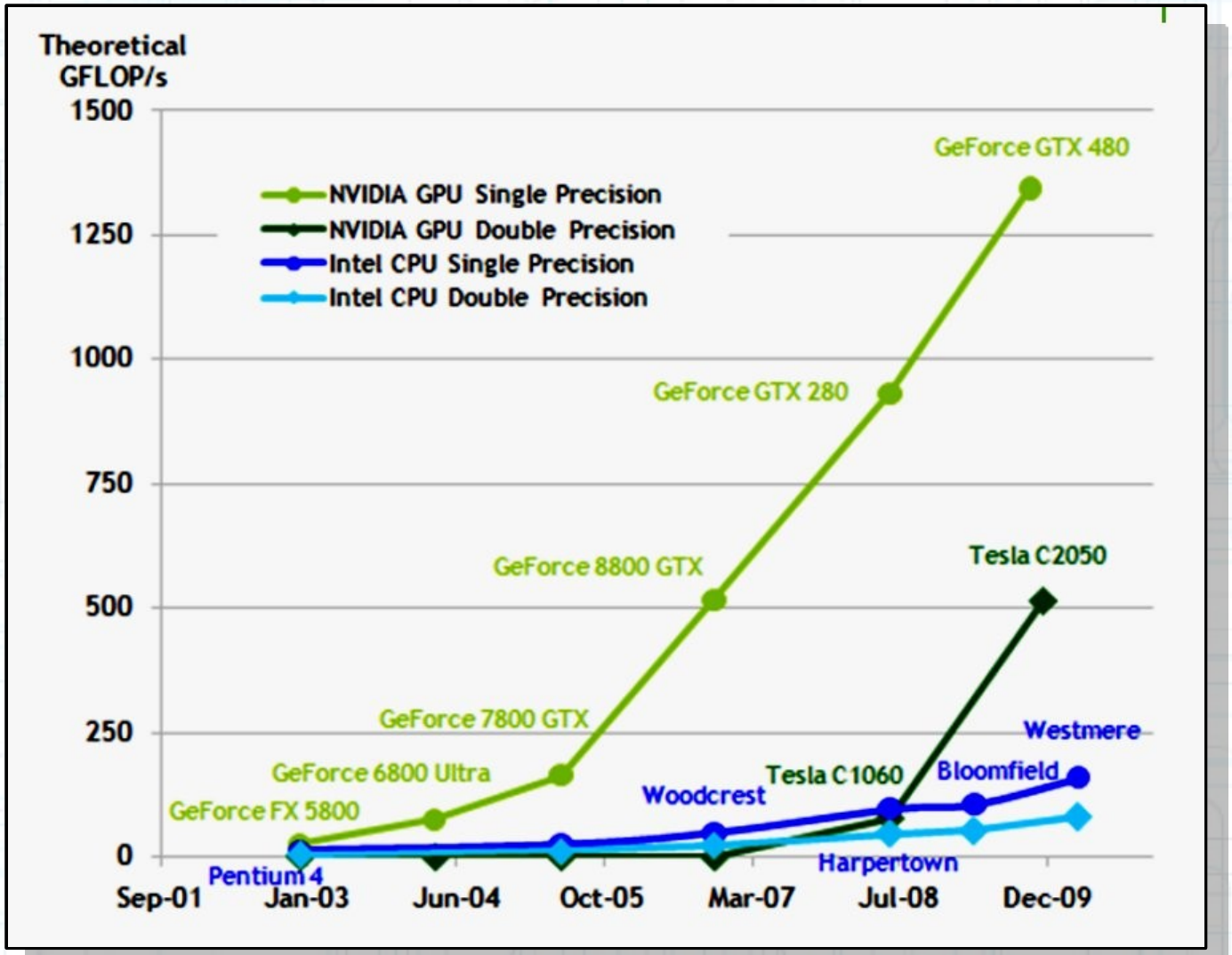
- Multi-GPU blade servers

Sources

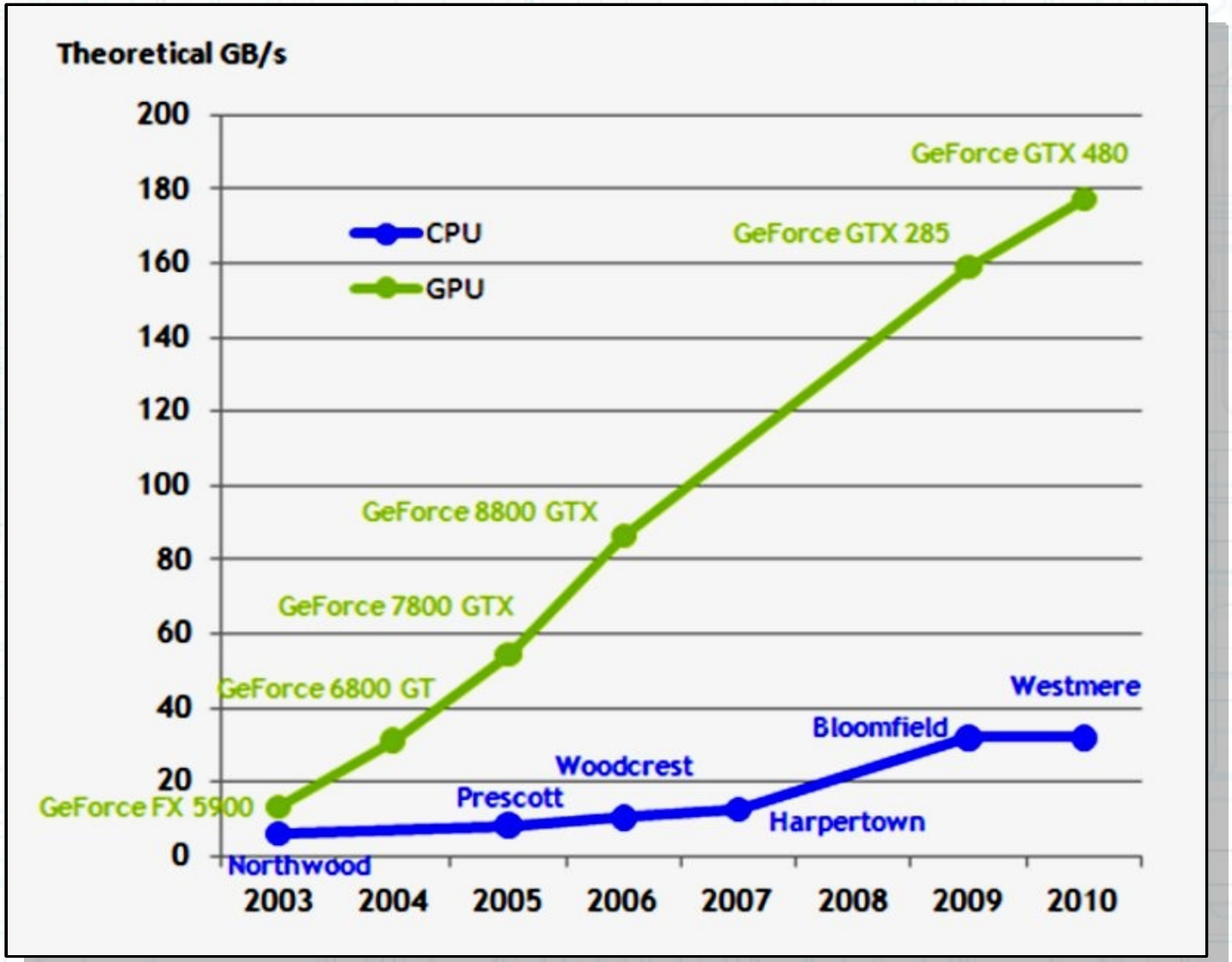

NVIDIA
NVIDIA CUDA™
NVIDIA CUDA C
Programming Guide
Version 3.2
11/9/2010


NVIDIA
Whitepaper
NVIDIA's Next Generation
CUDA™ Compute Architecture:
Fermi™
V1.1

GPU vs CPU – GFLOPS



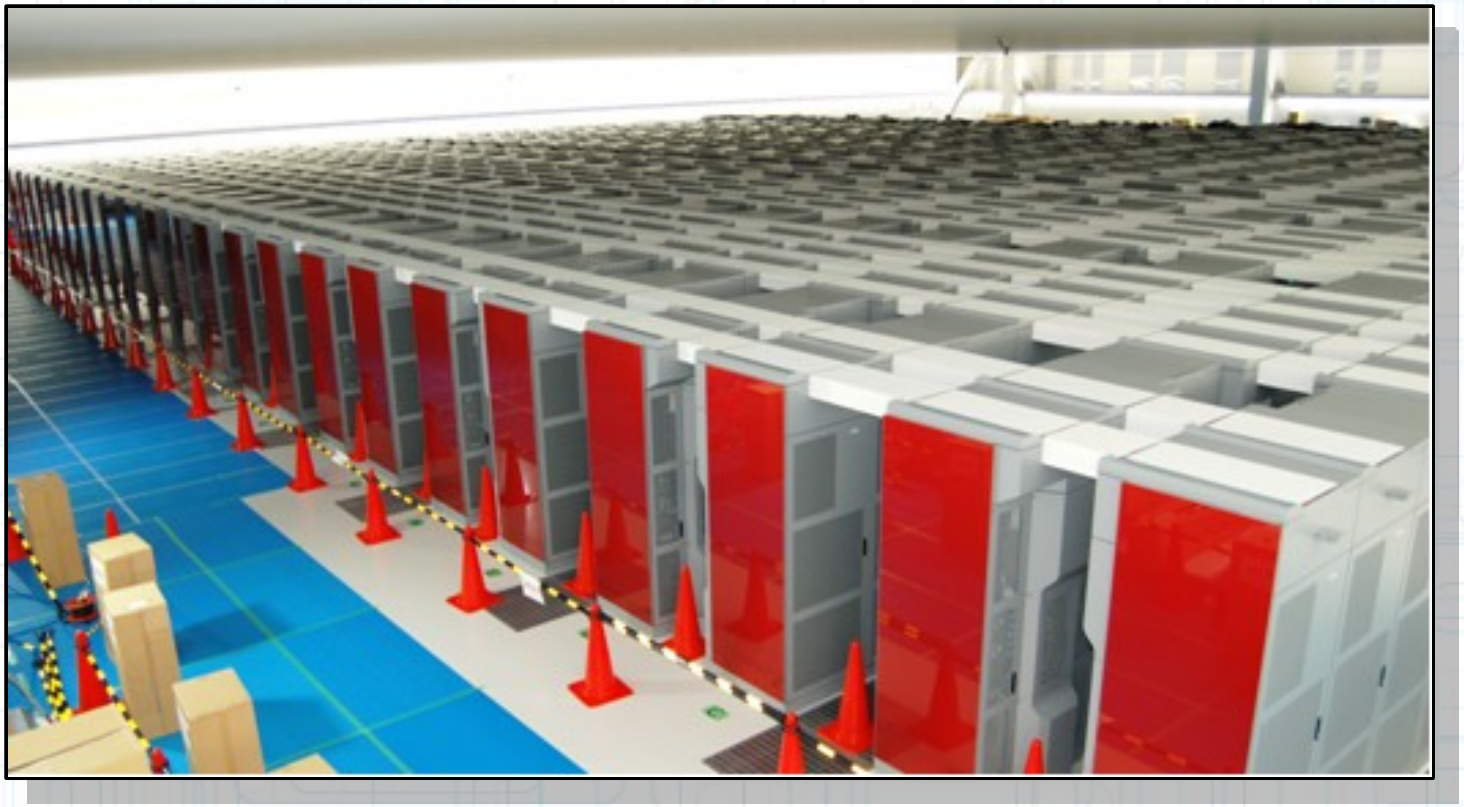
GPU vs CPU – GB/s



Tianhe-1A @ NSC in Tianjin

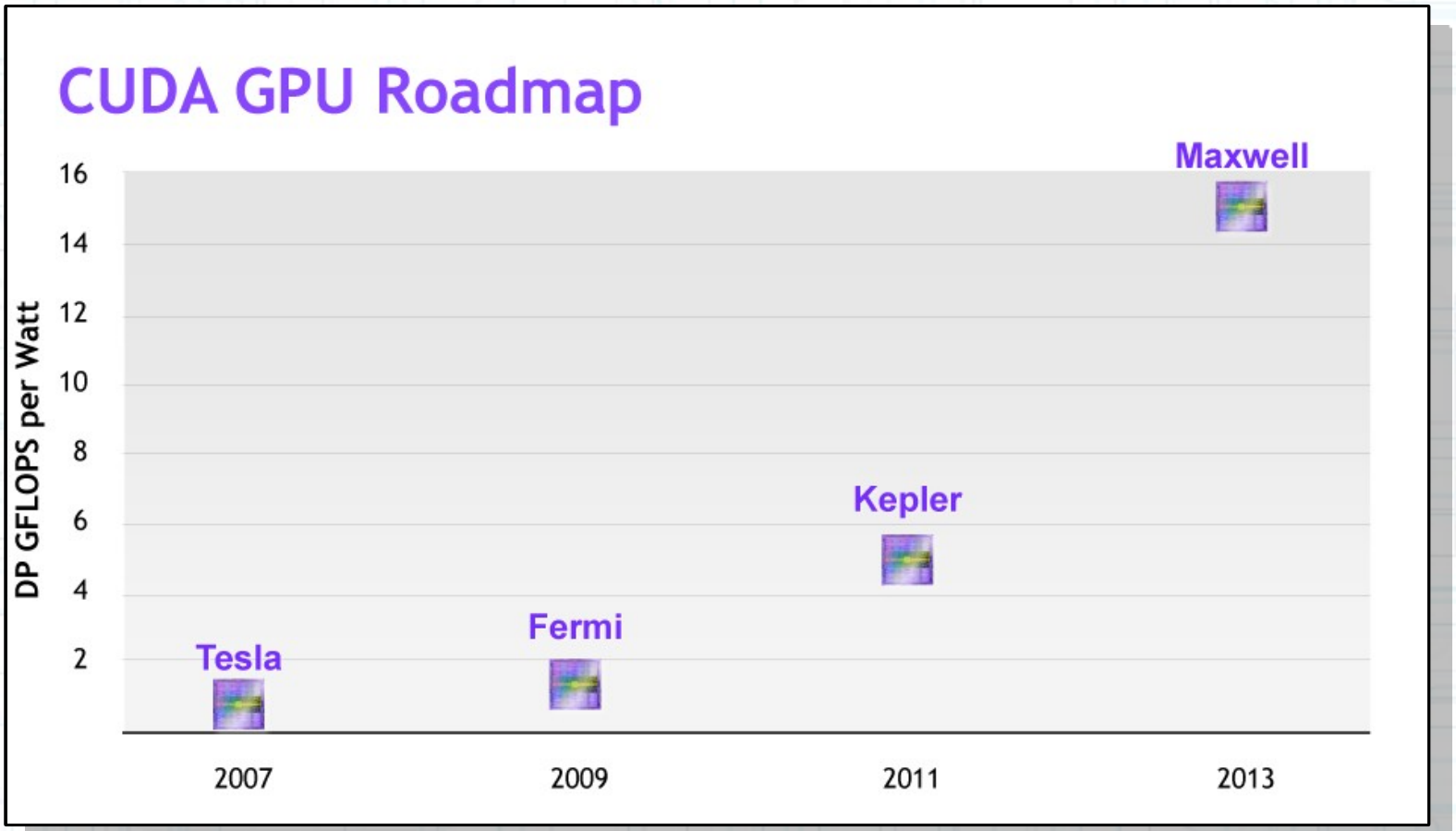
In November 2010 the Tianhe supercomputer at National Supercomputer Center in Tianjin crossed the performance of **2.5 PetaFLOPS** (10^{15} floating point operations per second) using **14,336 Intel Xeon** 6-core processors and **7,168 Nvidia Fermi** processors.

K computer @ RIKEN in Kobe

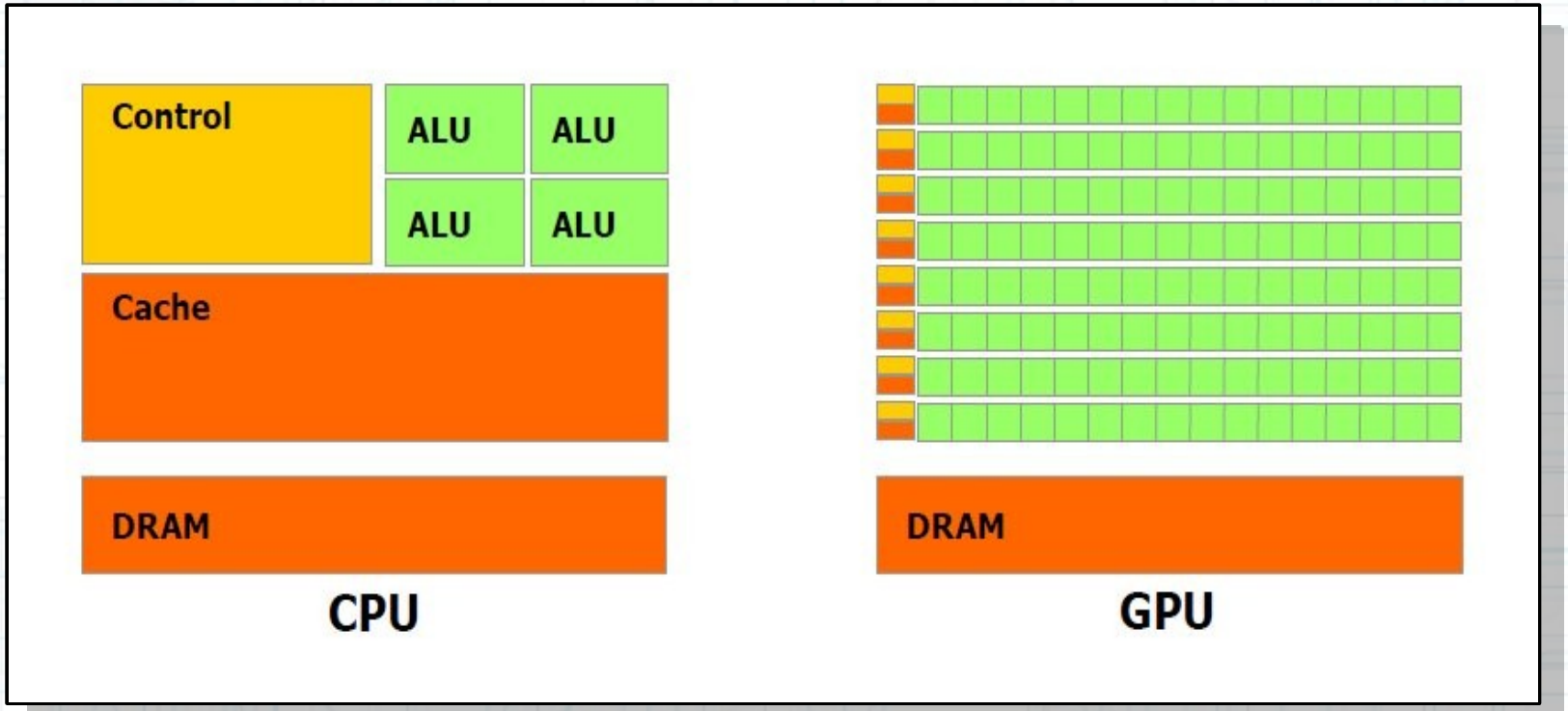


In November 2011 the K computer at RIKEN Advanced Institute for Computational Science in Kobe crossed the performance of **10 PetaFLOPS** (10^{15} floating point operations per second). using **88,128 SPARC64 VIIIfx** 8-core processors (homogeneous system – no accelerators).

Nvidia Roadmap

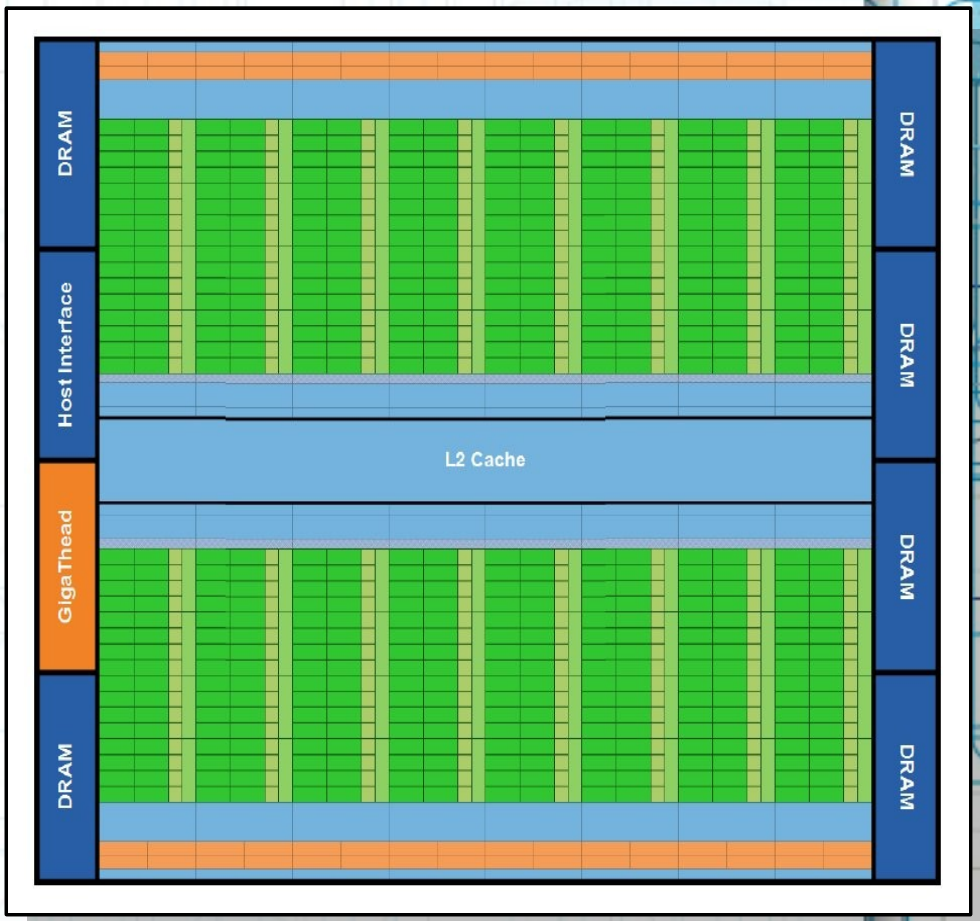


CPU vs GPU – Design



Nvidia Fermi

- 3.0 billion transistors
- 16 Streaming Multiprocessors (SMs) (currently 14)
- 32 “CUDA cores” per SM
- 512 “CUDA cores” (currently 448)
- up to 6 GB of DRAM
- clock around 1.1 GHz



execution units

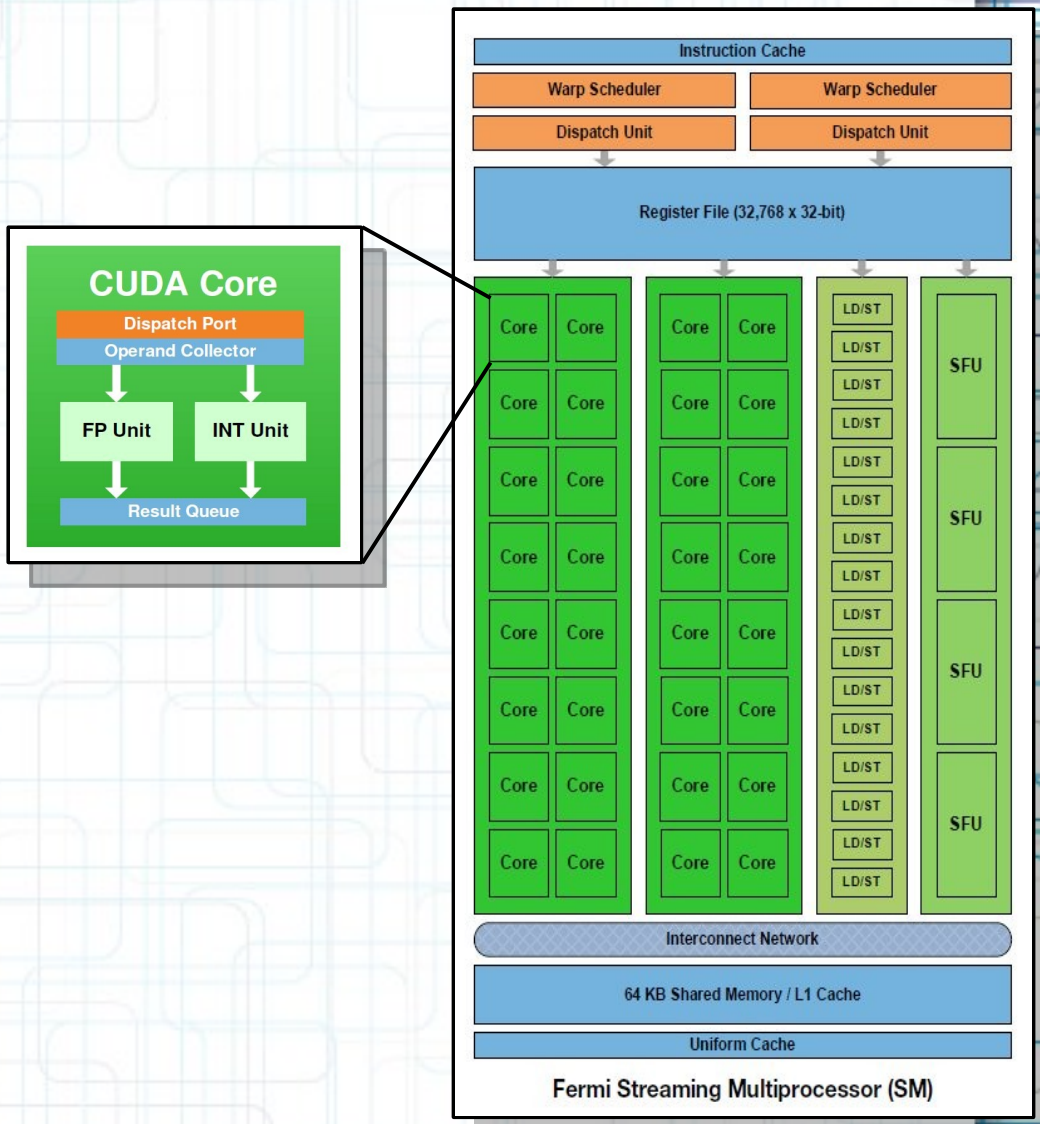
scheduler and dispatch

register file and L1 cache

Fermi's Multiprocessor

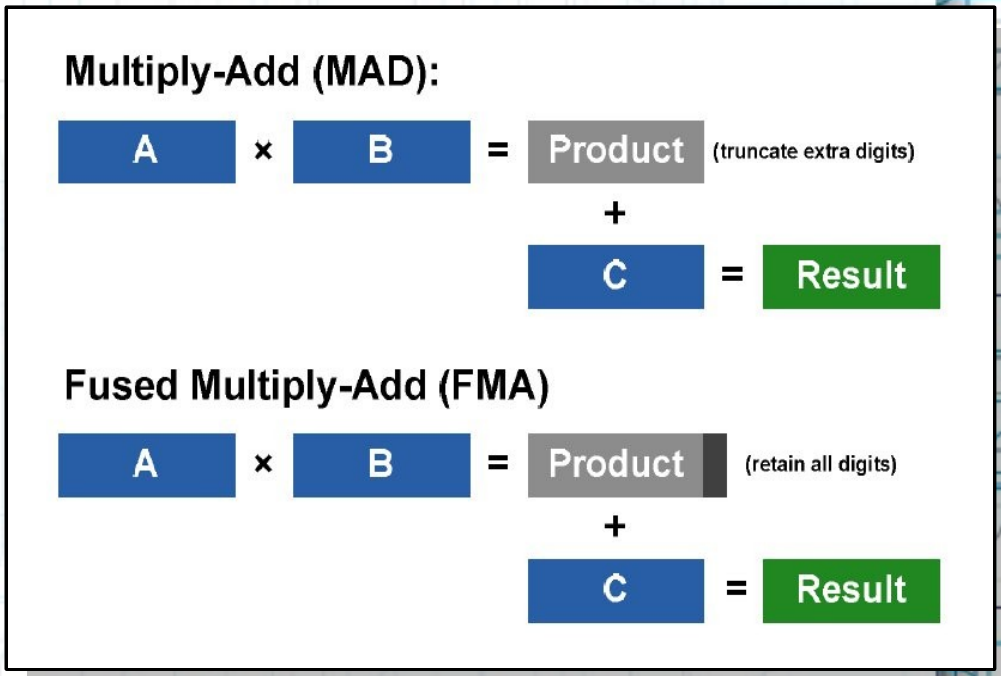
- Core – CUDA core
- LD / ST – load / store unit
- SFU – special functions unit
  (sine, cosine, reciprocal, root)

- 32 CUDA cores
- 1024 threads
- 32,768 registers (32-bit)
- 64 KB shared memory / L1 cache



Fermi's Floating Point

- Full IEEE floating point
- Fused Multiply-Add (FMA)
- Fast double precision
  (half the speed of single)

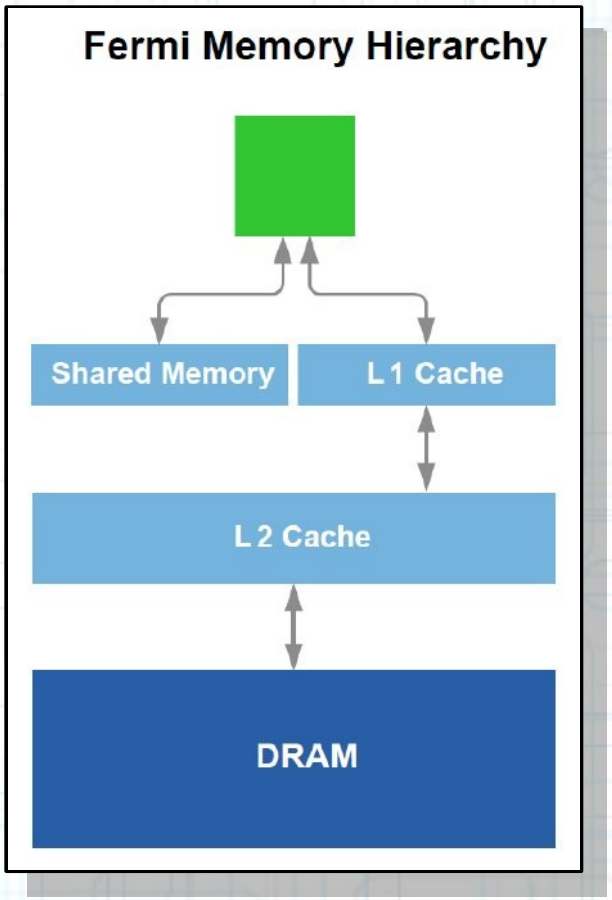


1.147 GHz x **32** cores x **14** SMs x **2** ops = **1028** GFLOPS in single precision

513 GFLOPS in double precision

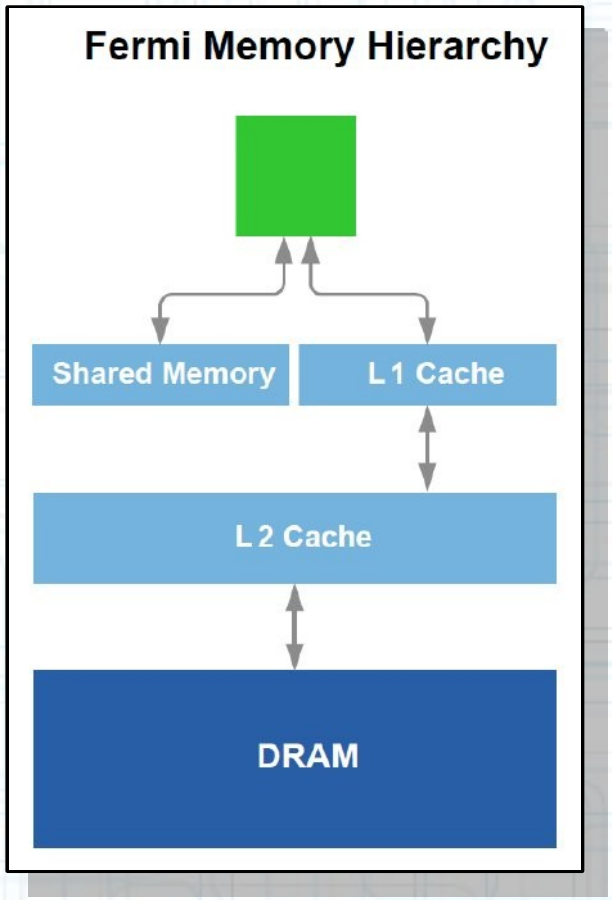
Fermi's Memory

- Shared Memory
  - each SM
  - 64 KB
- L2 Cache
  - all SMs
  - 768 KB
- DRAM
  - the GPU
  - up to 6 GB



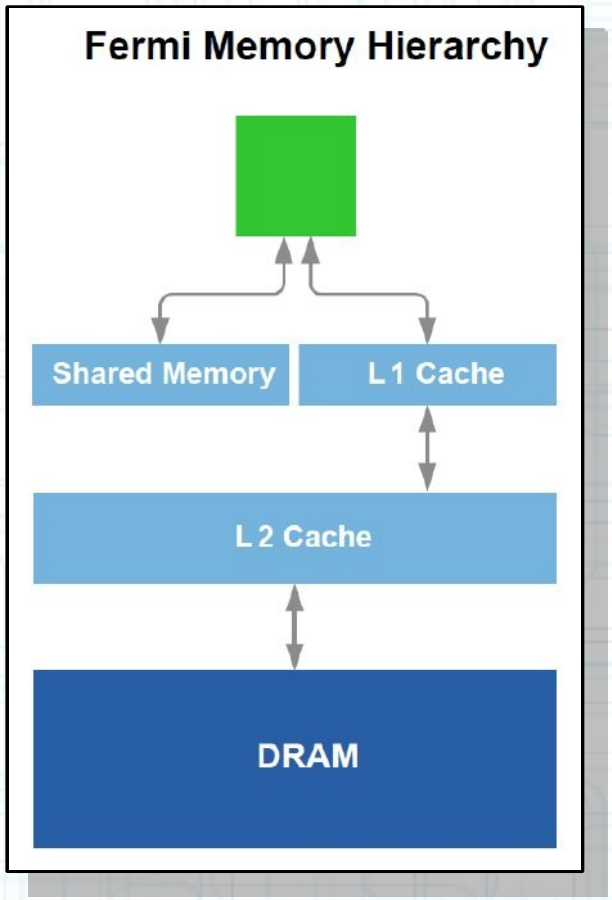
Shared Memory / L1 Cache

- Shared Memory
  - each SM
  - 64 KB
  - 48 KB shared memory + 16 KB L1 cache
  - 16 KB shared memory + 48 KB L1 cache
  - 32 memory banks (32-bit interleaved)



Global Memory & L2 Cache

- L2 Cache
  - all SMs
  - 768 KB
  - cache line is 128 bytes
    and maps to 128-byte aligned segment
    in device memory
- Global (Device) Memory
  - up to 6 GB
  - 6 partitions (256-byte interleaved)

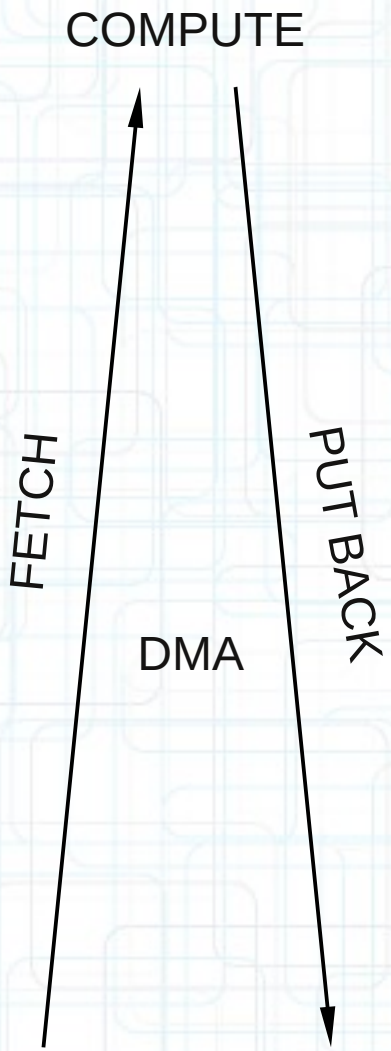


Host & Device

- GPU's device memory
  on the GPU board

- PCI Express x16 (8 GB/s)

- CPU's host memory
  on the motherboard

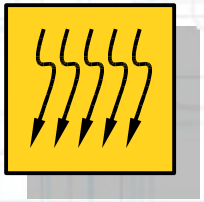


CUDA

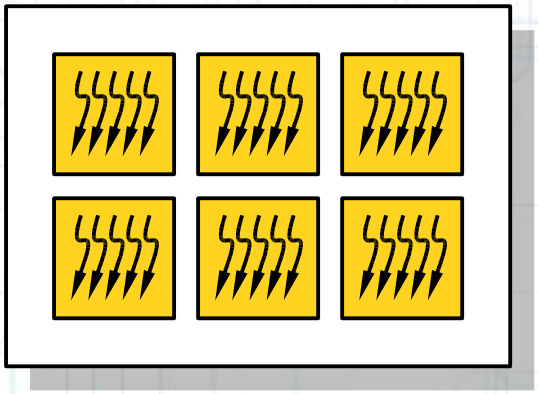
SOFTWARE

Thread

Thread Block



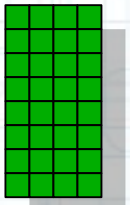
Grid



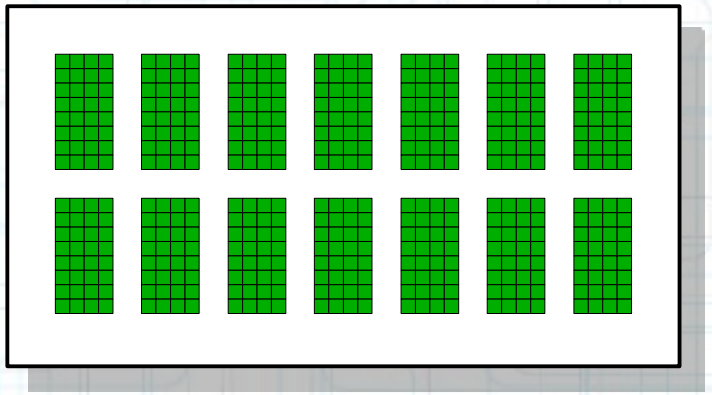
HARDWARE

CUDA core

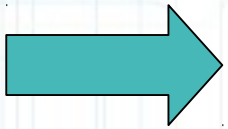
Multiprocessor



GPU



is executed by

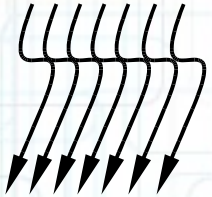


Hardware Multithreadig

SOFTWARE

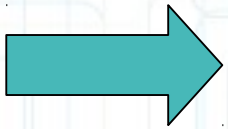
HARDWARE

Many Threads

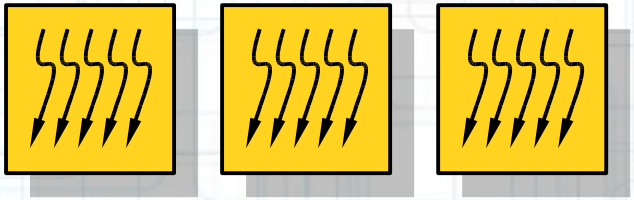


CUDA core

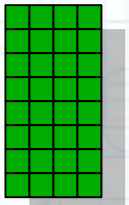
are mapped to



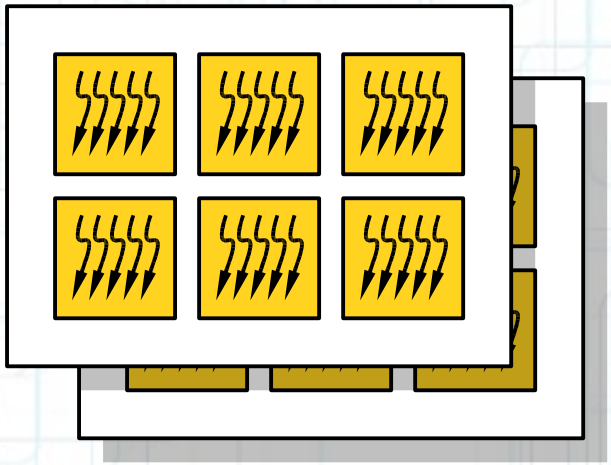
Many Thread Blocks



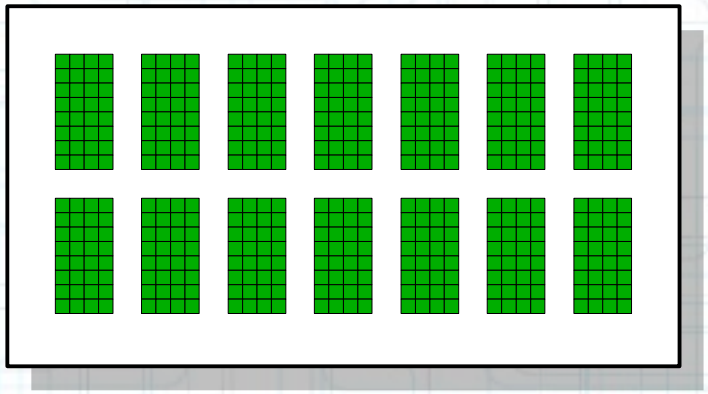
Multiprocessor



Many Grids

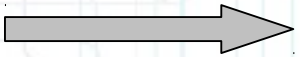


GPU



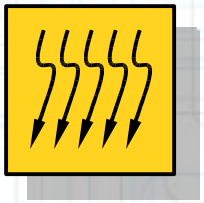
CUDA

Thread



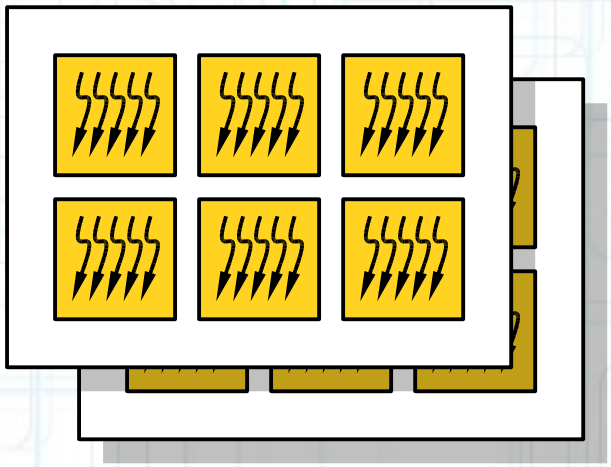
Per-thread registers
Per-thread local memory

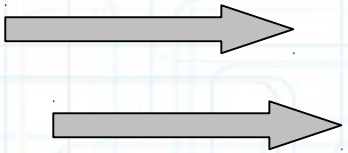
Thread Block



Per-block shared memory

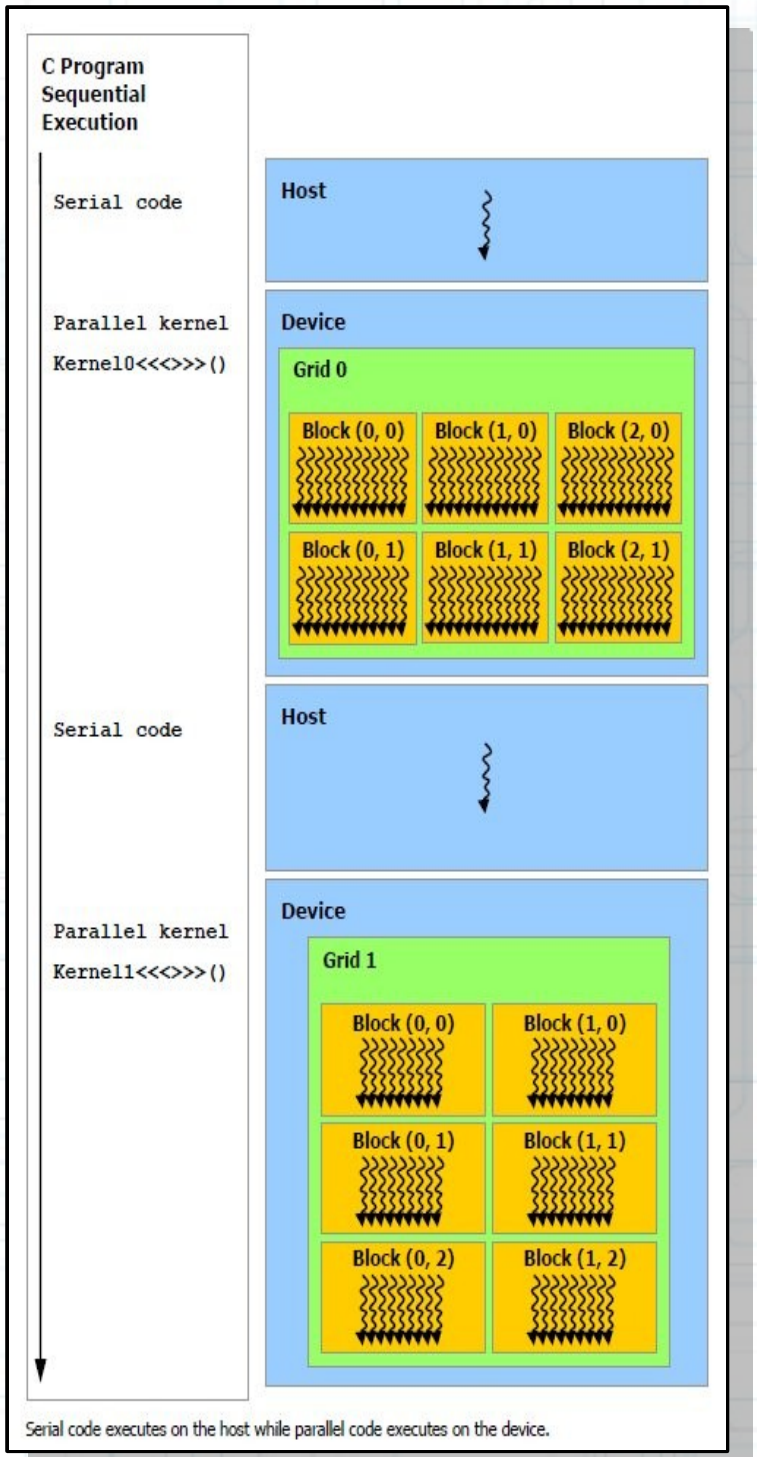
Grid





Global Memory

Heterogeneous Programming



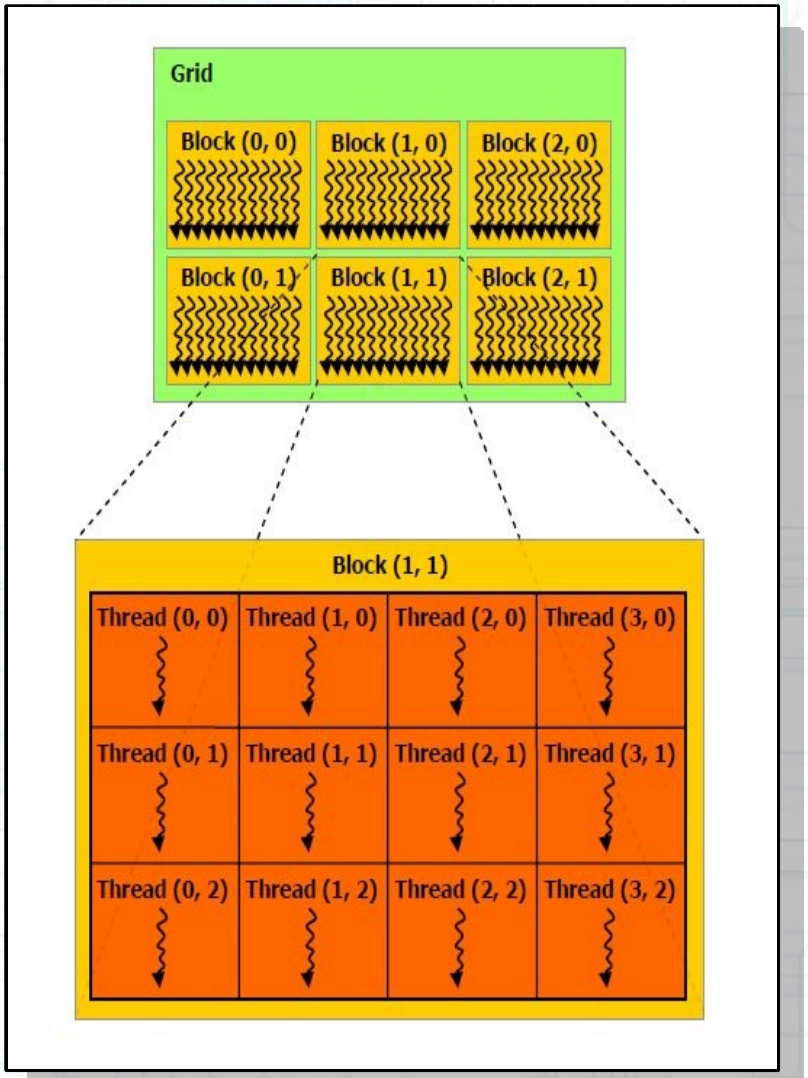
Host → Device
Data

Device → Host
Data

Host → Device
Data

Device → Host
Data

Thread Hierarchy



- Blocks in a grid
  - are organized in a 1D, 2D grid
- Threads in a block
  - are organized in a 1D, 2D, 3D grid

- 32 consecutive threads are a **warp**
- SM schedules one warp at a time

Code Example

```
int main (int argc, char **argv)
{
    // ...
    my_cuda_code(...);
    // ...
}

extern "C"
void my_cuda_code(...)
{
    cudaMalloc(...);
    cudaMemcpy(...);

    dim3 dimGrid(60, 80, 1);
    dim3 dimBlock(32, 4, 1);

    my_cuda_kernel<<<dimGrid, dimBlock>>>(...);

    cudaThreadSynchronize();

    cudaMemcpy(...);
}
```

```
__global__
void my_cuda_kernel(...)
{
    __shared__ float shared_array[16][16];

    int m, n, k;

    int my_block_x = blockIdx.x;
    int my_block_y = blockIdx.y;
    int my_block_z = blockIdx.z;

    int my_thread_x = threadIdx.x;
    int my_thread_y = threadIdx.y;
    int my_thread_z = threadIdx.z;

    // Do some work

    __syncthreads();

    // Do some more work
}
```

Synchronize all thread blocks:

Make sure that all thread blocks in the grid completed.

Synchronize all threads:

Make sure that all threads in a thread block completed.

The Data-Parallel Challenge

- The only way to synchronize threads in a block is a barrier.
  - Registers are thread-private.
  - Local memory is thread-private.
  - Communication can only be done through shared memory (write – barrier – read).
- The only way to synchronize thread blocks is a barrier.
  - Shared memory is private to a thread block.
  - Communication can only be done through device memory (write – barrier – read).

Partitioning Resources

	Compute Capability				
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**
Maximum x- or y-dimension of a grid of thread blocks	65535				
Maximum number of threads per block	512				1024
Maximum x- or y-dimension of a block	512				1024
Maximum z-dimension of a block	64				
Warp size	32				
Maximum number of resident blocks per multiprocessor	8				
Maximum number of resident warps per multiprocessor	24		32		48
Maximum number of resident threads per multiprocessor	768		1024		1536
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K
Maximum amount of shared memory	16 KB				48 KB

Performance 101

- Thread divergence
  - Threads in a warp have to branch in tandem.
  - If they do not, all different execution paths taken are serialized.

- Warp serialization (bank conflicts)
  - Threads in a warp have to access different banks of the shared memory.
  - If multiple threads access the same bank, accesses are serialized.
  - Unless they actually access the same address.

    In that case broadcast is performed (no penalty).

Performance 101

- Coalescing (of memory accesses)
  - All threads in a warp can access the same cache-line aligned memory in one memory transaction
  - If warp accesses span multiple cache lines, multiple transactions will result.
- Partition Camping
  - Accesses to the device memory have to go to different partitions.
  - If all transactions go to the same partition, a fraction of peak bandwidth will be delivered.

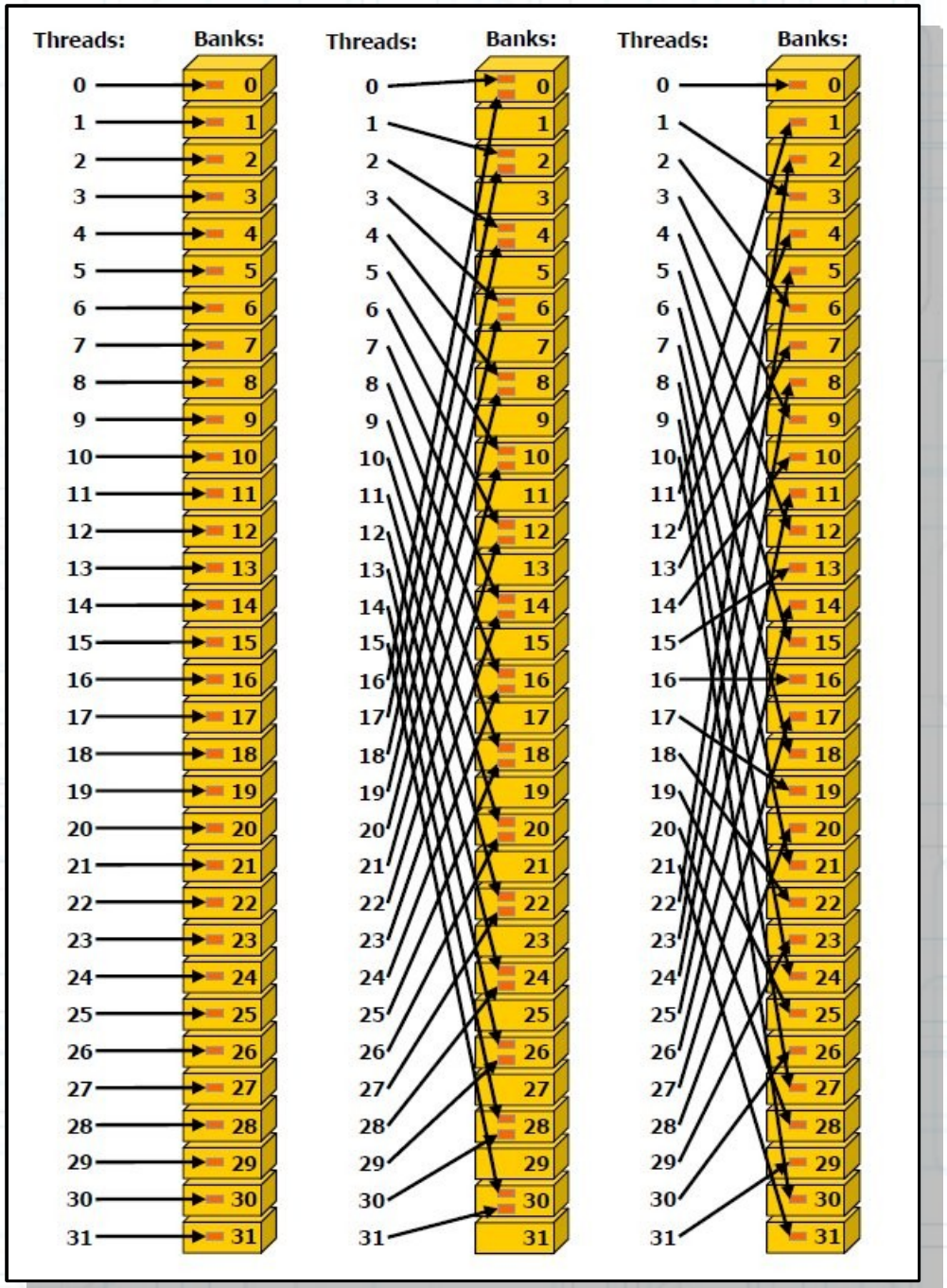
Thread Divergence

- Warp Scheduling
  - CUDA cores in a multiprocessor share instruction-issue logic.
  - SM can issue / complete one warp (32 threads) at a time, provided that all threads follow the same execution path.
  - Threads need to branch in tandem.
  - If threads take different branches, the execution is serialized. (Different execution paths are executed in sequence.)
  - It resembles the Japanese 31-legged race. (If one kid is out of step, everyone falls on the face.)

Warp Serialization

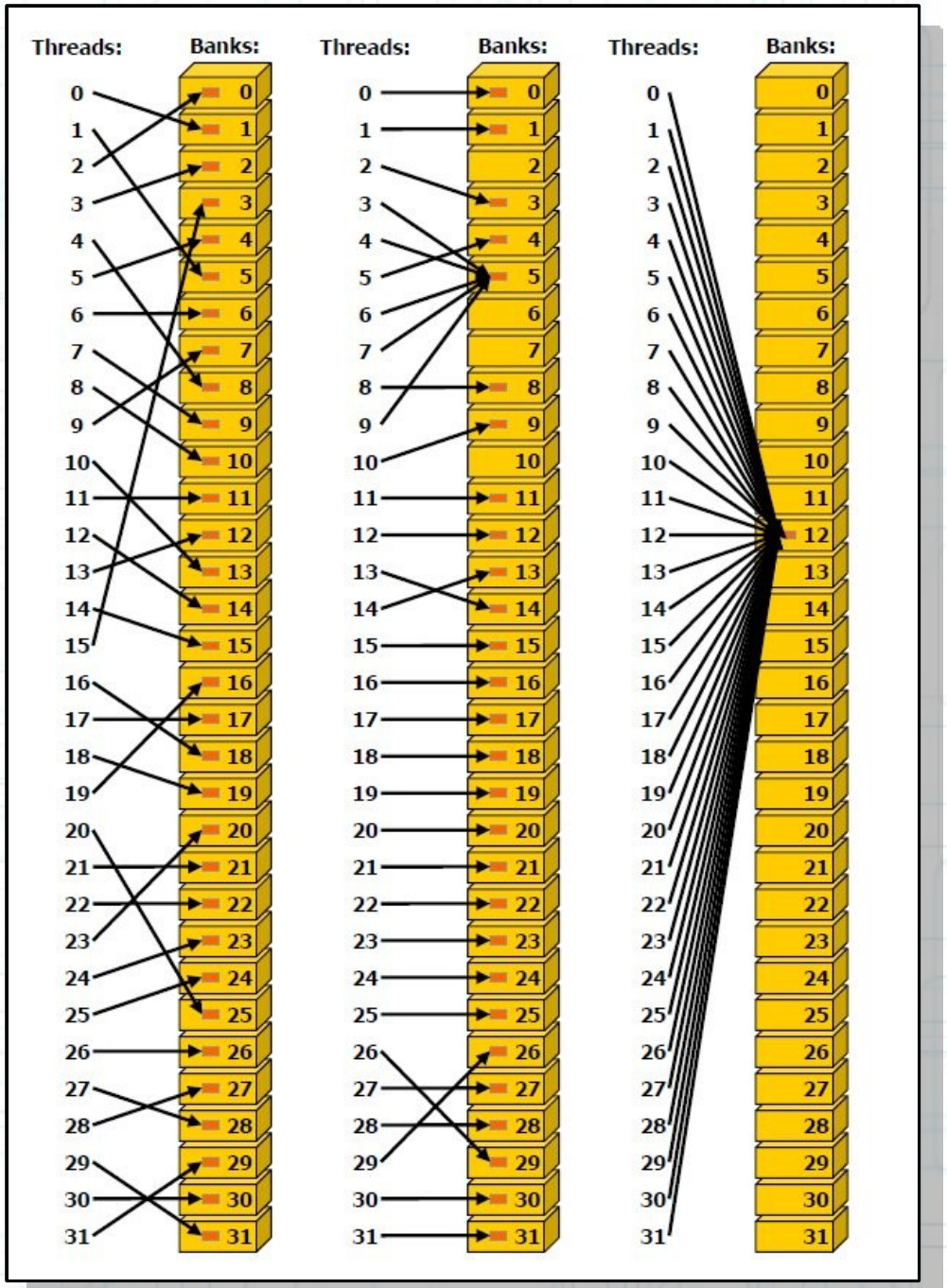
- Shared Memory Bank Conflicts
  - The SM can issued / complete one load / store to / from shared memory for one warp (32 threads) in each cycle, provided that all threads access different banks.
  - Shared memory locations 32 elements apart access the same bank.
  - If threads in a warp access the same bank, the memory operation is serialized.
  - The exception is the broadcast operation, when different threads in a warp access the exact same memory location (no serialization).

Bank Conflicts



- stride one – no conflicts
- stride two – 2-way conflict
- stride three – no conflicts

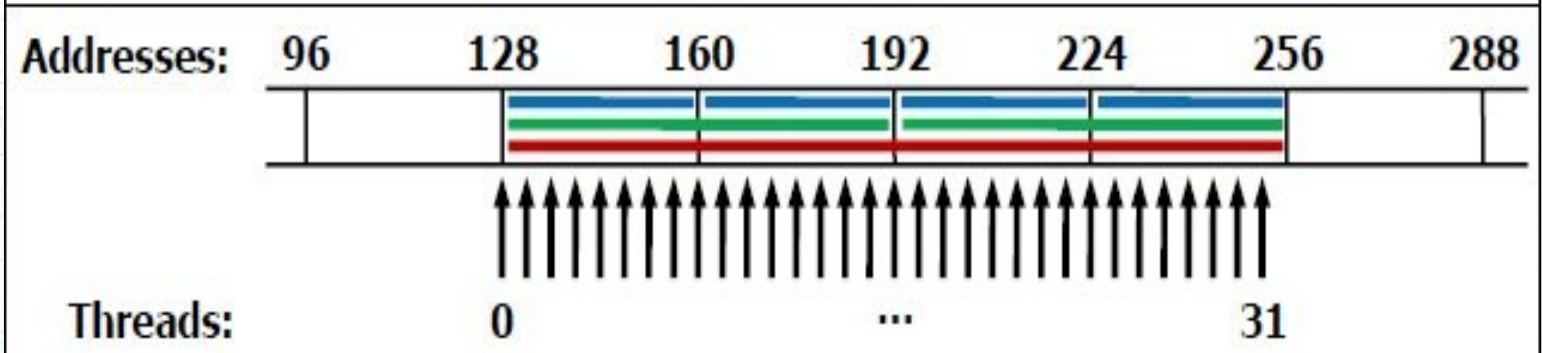
Bank Conflicts



- Conflict free examples
  - random permutations
  - multiple broadcasts
  - single broadcast

Coalescing

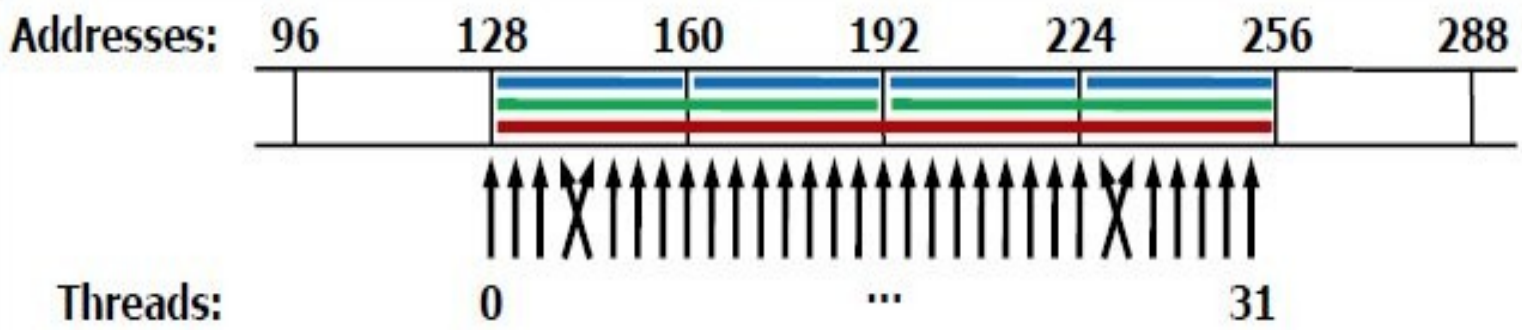
Aligned and sequential



Compute capability:	1.0 and 1.1	1.2 and 1.3	2.0
Memory transactions:	Uncached		Cached
	1 x 64B at 128 1 x 64B at 192	1 x 64B at 128 1 x 64B at 192	1 x 128B at 128

Coalescing

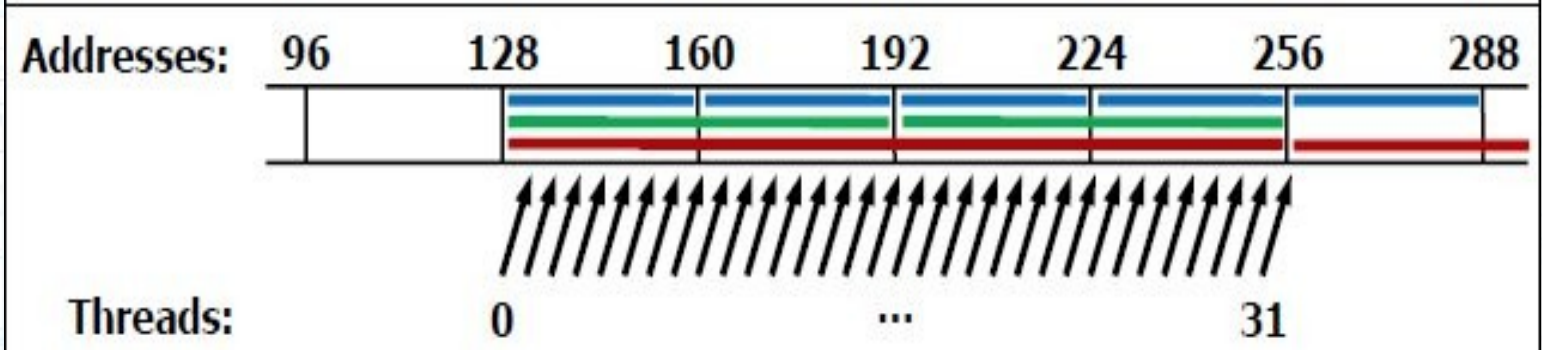
Aligned and non-sequential



Compute capability:	1.0 and 1.1	1.2 and 1.3	2.0
Memory transactions:	Uncached		Cached
	8 x 32B at 128 8 x 32B at 160 8 x 32B at 192 8 x 32B at 224	1 x 64B at 128 1 x 64B at 192	1 x 128B at 128

Coalescing

Misaligned and sequential



Compute capability:	1.0 and 1.1	1.2 and 1.3	2.0
Memory transactions:	Uncached		Cached
	7 x 32B at 128 8 x 32B at 160 8 x 32B at 192 8 x 32B at 224 1 x 32B at 256	1 x 128B at 128 1 x 64B at 192 1 x 32B at 256	1 x 128B at 128 1 x 128B at 256

GPU Optimization Summary

- Identify data parallelism in your application
  - Find independent work (separate data)
- Design the block grid (partition your data)
- Identify data parallelism within the thread block
- Design your thread grid
  - For instance, replace three nested loops with 3D thread grid
  - Use the shared memory to rearrange data if necessary (communicate data between the treads)
  - Make sure all threads in a warp follow the same execution path
  - Make sure all thread in a warp access different banks of shared memory
  - Make sure warp accesses to the main memory are cache-line aligned
  - Make sure accesses to the main memory balance load among all partitions
- Make sure there are many warps per thread block
- Make sure there are many thread blocks per multiprocessor
- Pay attention to partitioning registers among threads
- Pay attention to partitioning shared memory among thread blocks