Chapter 7B

Multicores, Multiprocessors, and Clusters

History of GPUs

- A major justification for adding SIMD instruction
  - Many microprocessors were connected to graphic displays in PCs and workstations
  - So an increasing fraction of processing time was used for graphics.
  - Improve graphics processing by using the transistors available due to Moore's law
- Video graphics controller chips added functions to accelerate 2D and 3D graphics.
  - High-end graphics cards for TV advertisement and movies
  - Video graphics controllers had a target to shoot for as processing resources increased, just like microprocessors borrowed ideas from supercomputers

History of GPUs

- A major driving force was the computer game industry
  - 3D graphics cards for PCs and game consoles like Sony PlayStation
  - Positive feedback: Moore's Law $\Rightarrow$ lower cost, higher density, more functions
- Graphics Processing Units (GPU)
  - Evolved its own style of processing and terminology
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

Characteristics of GPUs

- GPUs are accelerators that supplement a CPU
  - No need to be able to perform all the tasks of a CPU
  - CPU-GPU combination: heterogeneous multiprocessing
- The programming interfaces to GPUs are
  - high-level application programming interfaces (APIs) such as OpenGL and Microsoft’s DirectX, coupled with
  - high-level graphics shading languages, such as NVIDIA’s C for Graphics (Cg), and Microsoft’s High lelel shader language (HLSL)

Characteristics of GPUs

- This environment leads to more rapid innovation in GPUs than in CPUs.
  - The language compilers target industry-standard intermediate languages instead of machine instructions.
  - GPU driver software generates optimized GPU-specific machine instructions.
  - While these APIs and languages evolve rapidly to embrace new CPU resources enabled by Moore’s law, GPGPU designers are free form backward binary instruction compatibility
- Graphics processing involves drawing vertices of 3D geometry primitives such as lines and triangles and shading or rendering pixel fragments of geometric primitives
  - Video games, for example, draw 20 to 30 times as many pixels as vertices.

Characteristics of GPUs

- Each pixel can be drawn independently, and each pixel fragment can be rendered independently.
  - To render millions of pixels per frame rapidly, the GPU evolved to execute many thread from vertex and pixel shader programs in parallel.
- The graphics data types are
  - Vertices: (x,y,z,w) coordinates, 32-bit floating-point number
  - Pixels: (red, green, blue, alpha) color components, 8-bit unsigned integer (single-precision floating-point number between 0.0 and 1.0 in recent GPUs)
- The working set is hundreds MB
  - Does not show temporal locality
  - A great del of data parallel in these tasks.

GPU Architecture

- GPU relay on having enough threads to hide the latency to memory
  - GPU do not rely on multilevel caches
  - Between the time of memory request and the time that data arrives, the GPU executes 100s K threads that are independent of that request
- The GPU main memory is thus oriented toward bandwidth rather than latency.
  - Even separate DRAM chips for GPU that are wider and have higher bandwidth that DRAMs for CPUs.
  - Traditionally had smaller main memories than conventional microprocessors.
    - GPUs: 1 GB or less, CPUs: 2 to 32 GB
  - For general-purpose computation, must include the time to transfer the data between CPU memory and GPU memory, since GPU is a coprocessor.

GPU Architecture

- GPUs were designed for a narrower set of applications.
- Given the four-element nature of the graphics data types, GPU s historically have SIMD instructions, like CPUs,
  - However, recent GPUs are focusing more on scalar instructions to improve programmability and efficiency.
- Unlike CPUs, there has been no support for double precision floating-point arithmetic.
  - In 2008, the first GPUs to support double precision in hardware were announced.
  - Nevertheless, single precision operations will still be 8 to 10 times faster than double precision, even on these new GPUs

GPU Architecture

- Recent GPUs are heading toward identical general-purpose processors to give more flexibility in programming.
    - Unifying the processors also delivers very effective load balancing.
    - Making them more like multicore designs found in mainstream computing.
    - In the past, GPU relies on heterogeneous special purpose processors to deliver performance needed for graphics applications.
- GPGPU (general-purpose GPU) : specify their applications to tap the high potential performance of GPUs.
    - Developed C-inspired programming language
    - NVIDIA's CUDA (compute unified device architecture)

CUDA

- A scalable parallel programming model and language based on C/C++.
  - It is a parallel programming platform for GPUs and multicore CPUs.
- The CUDA programming model has an SPMD software style.
- CUDA also provides a facility for programming multiple CPU cores as well
  - So CUDA is an environment for writing parallel programs for the entire heterogeneous computer system.
  - discrete GPU chip sits on a separate card that plugs into a standard PC over the PCI-Express interconnect

Graphics and Computing

- With the addition of CUDA and GPU computing, the GPU can be used as both a graphics processor and a computing processor at the same time
  - Combining these uses for visual computing applications.
- Underlying processor architecture of the GPU is exposed in two ways
  - As implementing the programmable graphics API
  - As a massively parallel processor array programmable in C/C++ with CUDA

GPU and Visual Computing

- GPU computing: using a GPU for computing via a parallel programming language and API
  - Without using the traditional graphics API and graphics pipeline model. <-> GPGPU
- Visual Computing: A mix of graphics processing and computing that lets you visually interact with computed objects via graphics, images, and video.

- GeFroce 8800 GTX
  - 16 MP: multiplrocessor
  - 8 SP/MP: streaming processor

Example: GeForce 8800 GTX

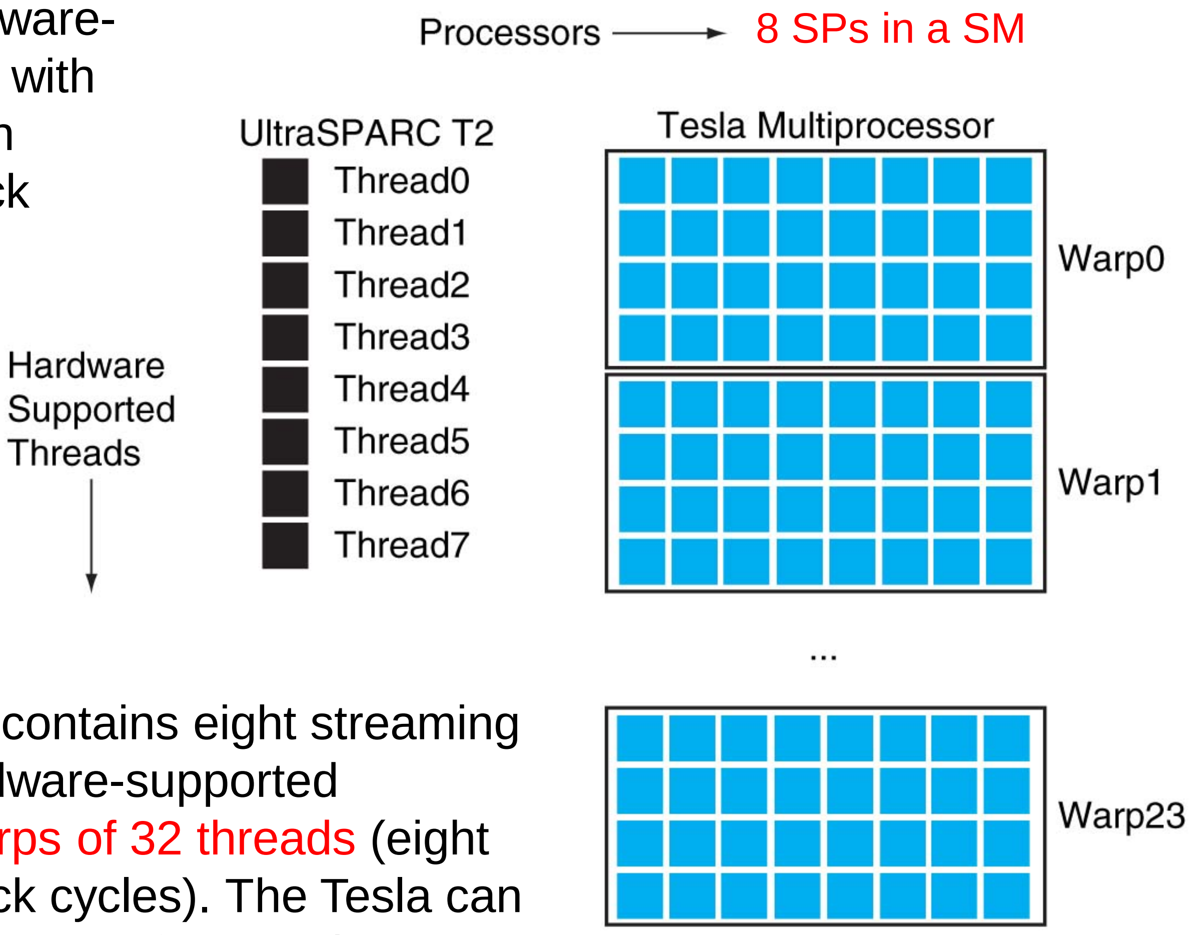
- Speak single precision multiply-add performance
  - 16 MPs x 8 SPs/ MP x 2 FLOPS/ instr/SP
    x 1.35 GHz = 345.6 GFLOPs/second
- Each multiprocessor has a software-managed local store with 16 KB plus 8192 32-bit registers
  - Each SP has 1024 32-bit registers
- The memory system of 8800 GTX consists of 6 partitions of 900 MHz Graphics DDR3 DRAM, each 8 bytes wide and with 128 MB of capacity.
  - Total memory size = 128 x 6 = 768 MB
  - Peak DDR3 memory bandwidth =
    6 x 8 bytes/transfer x 2 transfer/clock x 900 MHz
    = 86.4 GB/second

Example: 8800 GTX

- To hide memory latency, each streaming processor has HW-supported threads.
  - Each group of 32 threads is called warp
  - A warp is the unit of scheduling
  - The active threads in a warp execute in parallel in SIMD fashion
- Compare a Tesla multiprocessor to a SUN UltraSPARC T2 core.
  - Both are hardware multithreaded by scheduling threads over time, shown on the vertical axis.
  - Each Tesla multiprocessor consists of 8 streaming processors, which execute eight parallel threads per clock showing horizontally.

NVIDIA Tesla & Sun T2

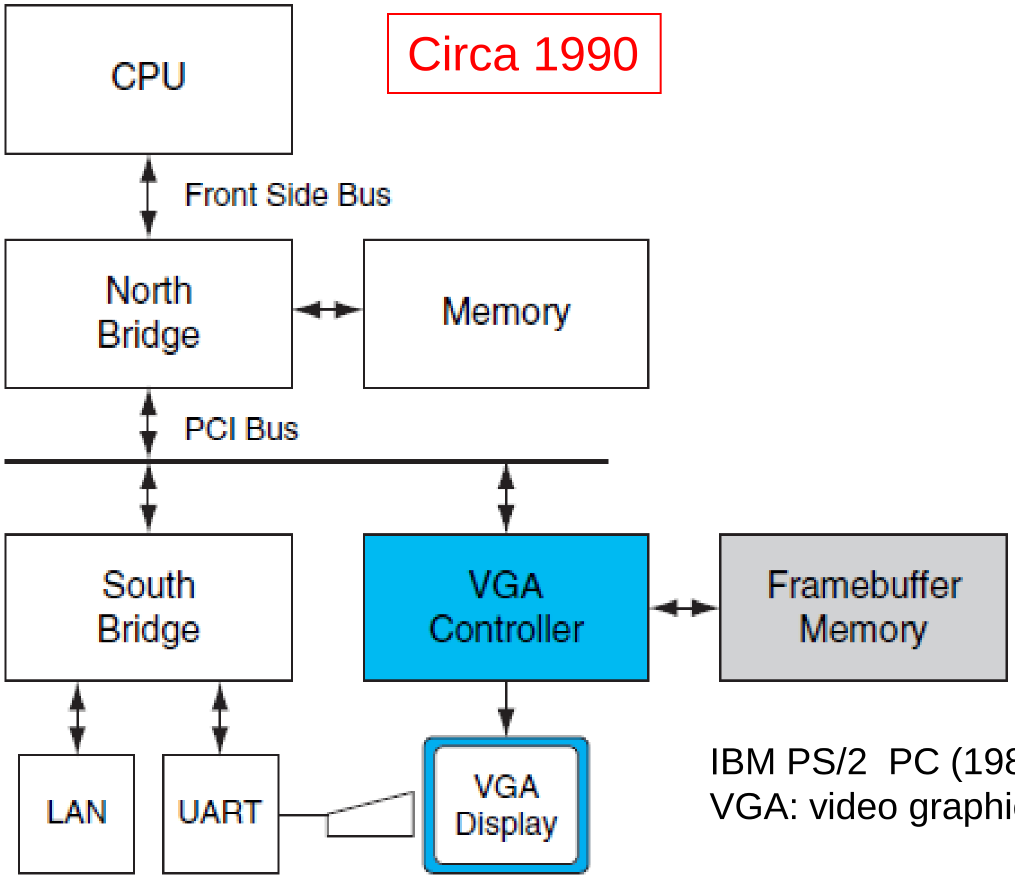
The T2 core is a single processor and uses hardware-supported multithreading with eight threads. The T2 can switch threads every clock cycle.



The Tesla multiprocessor contains eight streaming processors and uses hardware-supported multithreading with 24 warps of 32 threads (eight processors times four clock cycles). The Tesla can switch threads only every two or four clock cycles.

Graphics in the System

- A discrete GPU chip sits on a separate card that plugs into a standard PC over the PCI-Express interconnect



Display standards

QXGA (2028 x 1536),
Full HD (1920 x 1080p)
SXGA (1280 x 1024),
HD (1280 x 720p),
XGA (1024 X 768)
SVGA (800 x 600),
VGA (640 x 480),
QVGA (320 x 240)

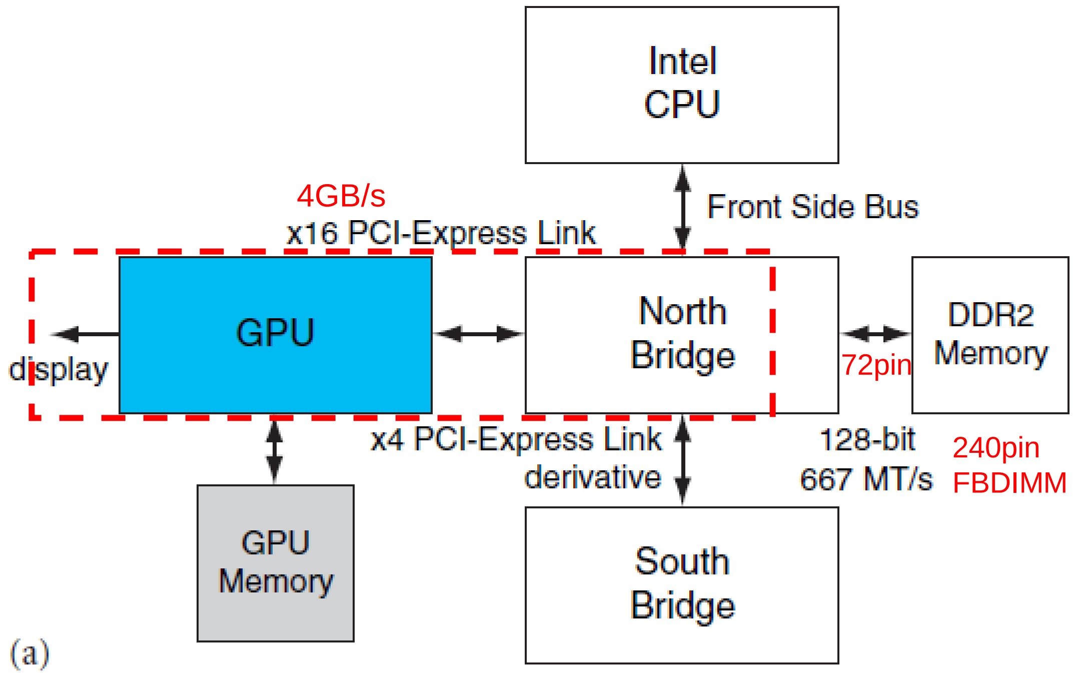
IBM PS/2 PC (1987)
VGA: video graphic array

FIGURE A.2.1 **Historical PC.** VGA controller drives graphics display from framebuffer memory.

Graphics in the System

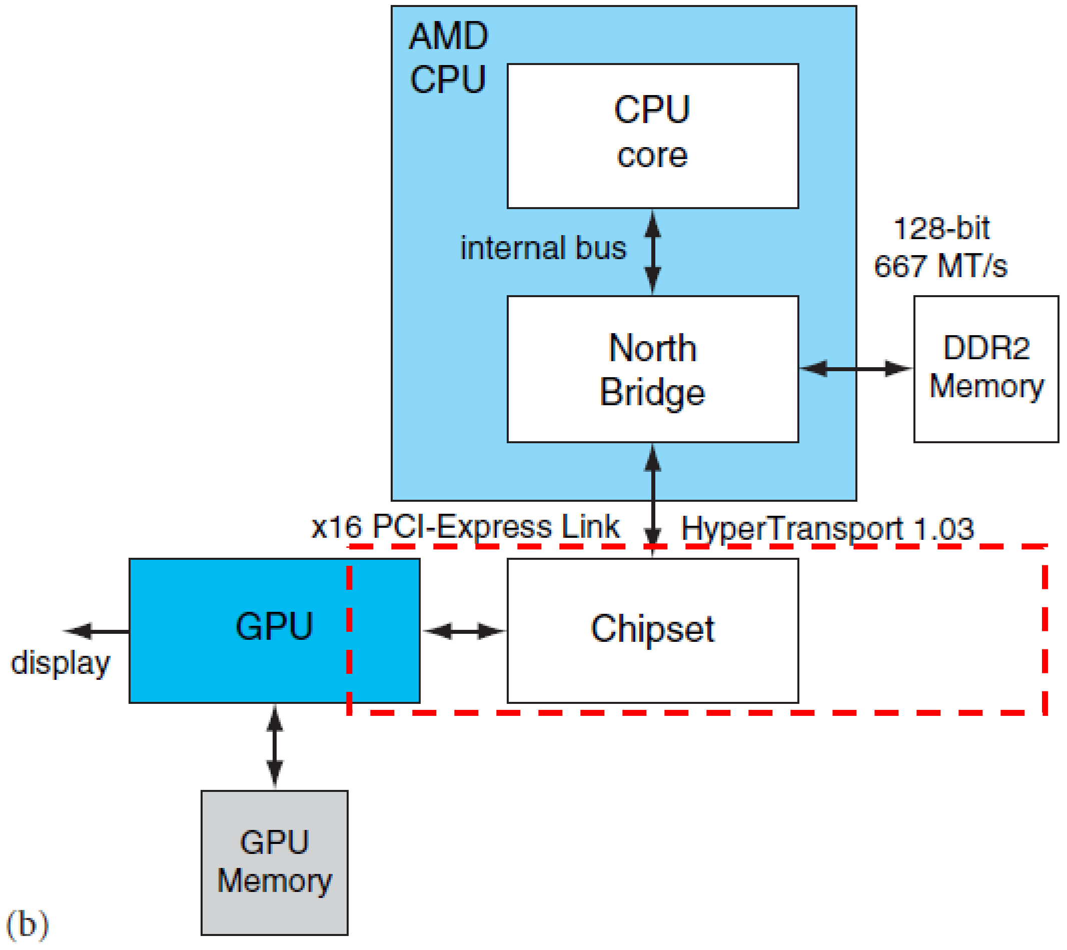
PCI_Express (PCIe)

A standard system I/O interconnect that uses point-to-point links.
links have a configurable number of lanes and bandwidth.
A lane is composed of a transmit and receive pair of differential lines.
Each lane is composed of 4 wires or signal paths, meaning conceptually, each lane is a full-duplex byte system.
The per-lane throughput of PCIe 2.0 rises from 250 MB/s to 500 MB/s.



A contemporary PC with an Intel CPU

Graphics in the System



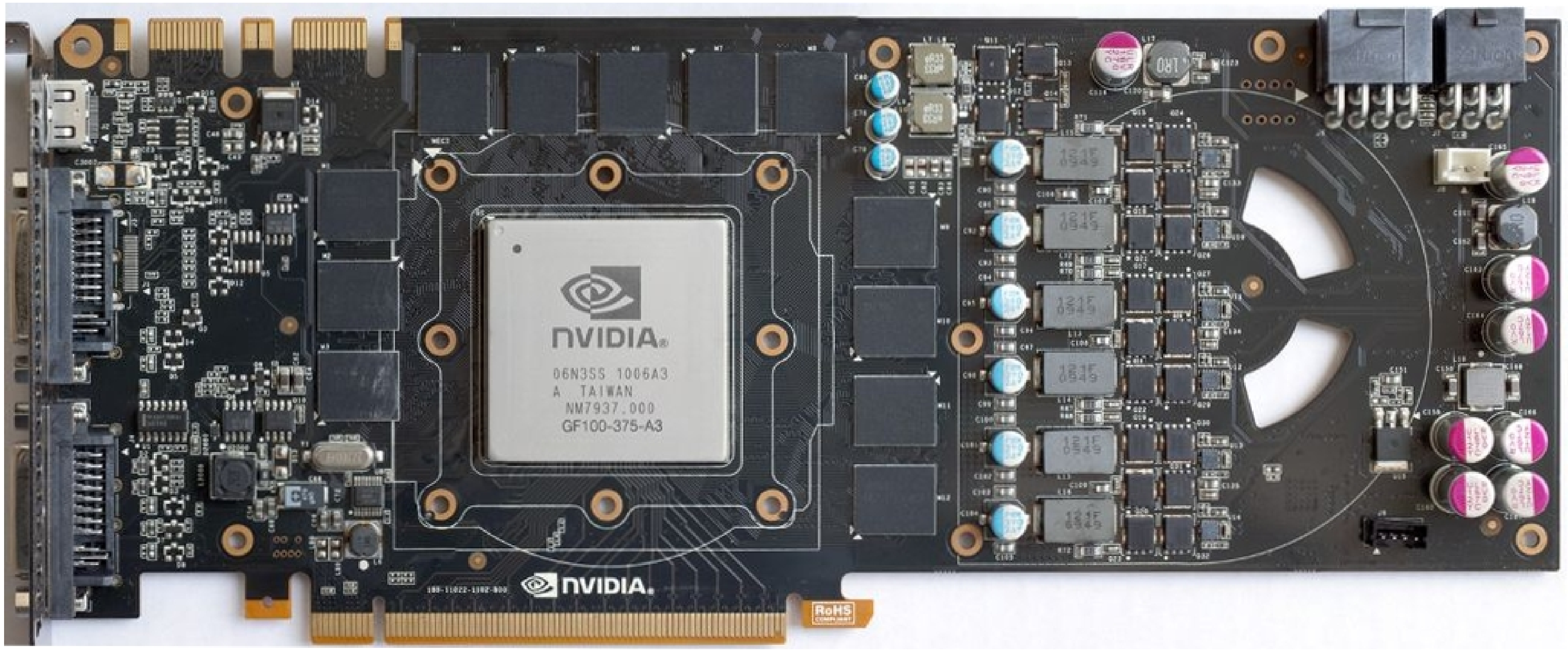
HyperTransport provides a high-speed, hig-performance, point-to-point dual simplex link for interconnectring IC components on a PCB.

A contemporary PC with an AMD CPU

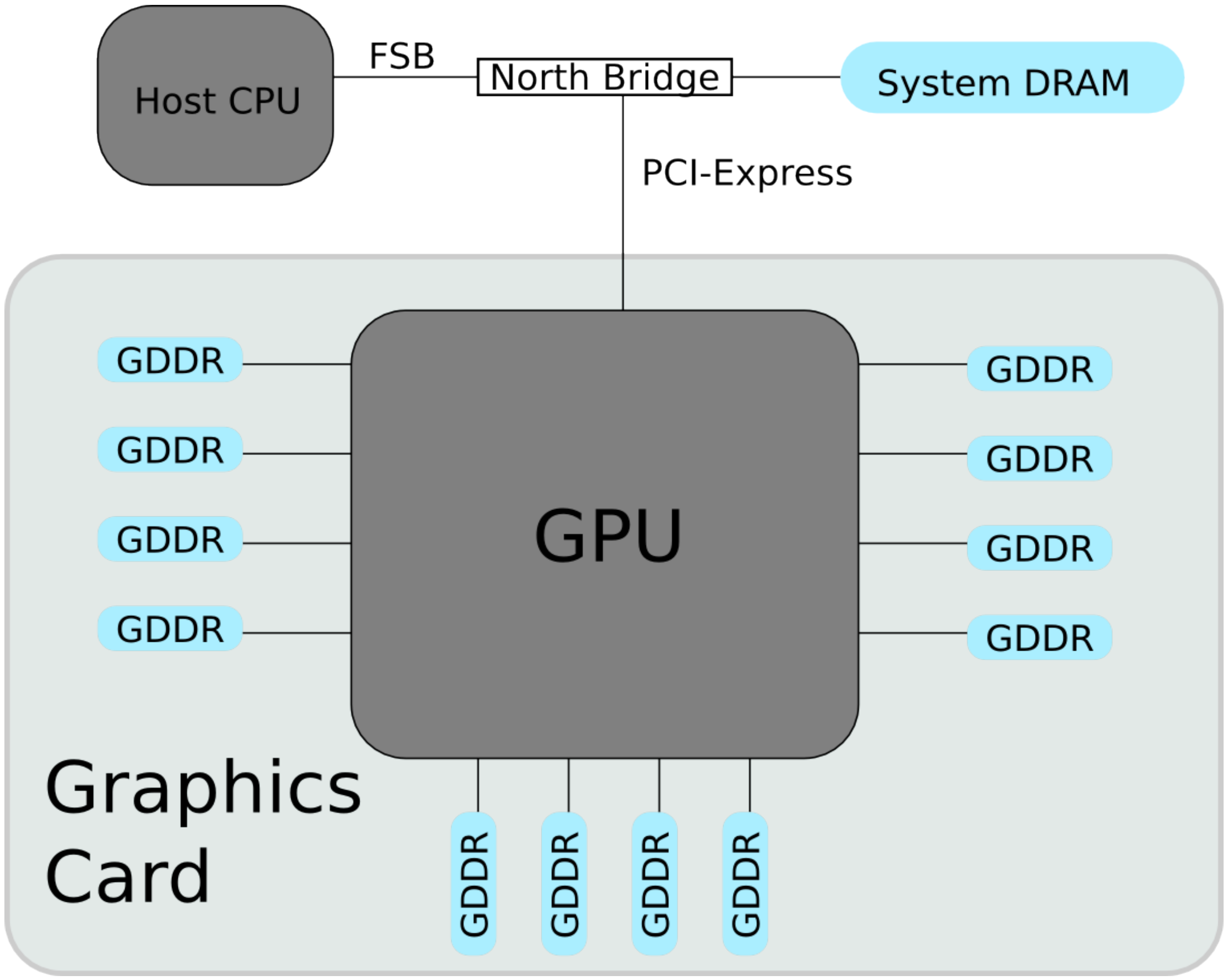
System Variations

- A low-cost system variation, a UMA system, uses only CPU system memory, omitting GPU memory from the system
- A high performance system variation uses multiple attached GPUs, typically 2 to 4 working in parallel with their displays daisy-chained.
  - An example is the NVIDIA SLI (scalable link interconnect) multi-GPU system, designed for high performance gaming and workstations.
- The next system category integrates the GPU with the north bridge (Intel) or chipsets (AMD) with and without dedicated graphic memory.

GPU Architecture



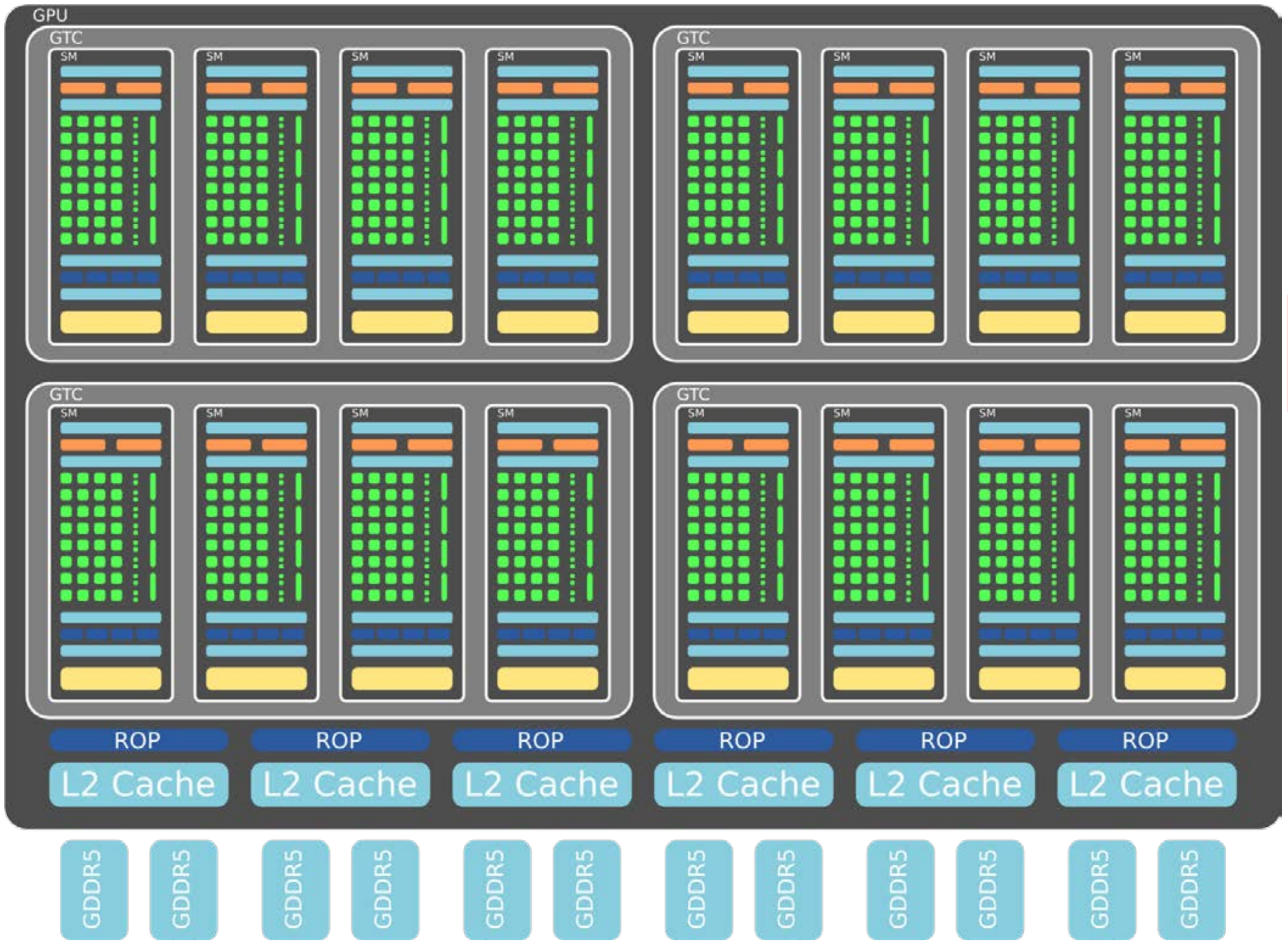
System Architecture



DDR: 2.5V
GDDR2: 2.5V
DDR2: 1.8V
GDDR3: 2.0V (Samsung)
1.8V (others)
DDR3: 1.5V
DDR4: 1.2V

GPU Architecture

NVIDIA Fermi, 512 Processing Elements (PEs)

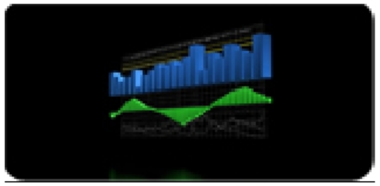


GTC: cluster
SM: streaming multiprocessor
ROP: raster operation

General Purposed Computing

Bio-Informatics and Life Sciences

Computational Electromagnetics and Electrodynamics

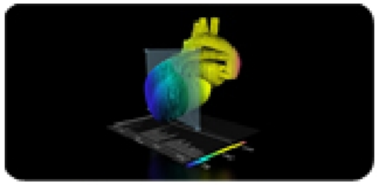

Computational Finance

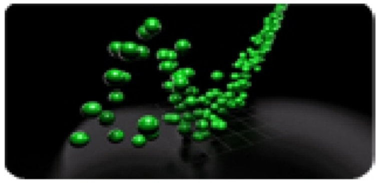
Computational Fluid Dynamics

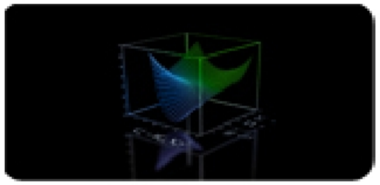
Data Mining, Analytics, and Databases

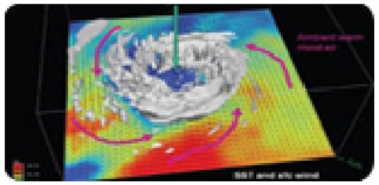
Imaging and Computer Vision

MATLAB Acceleration


Medical Imaging


Molecular Dynamics

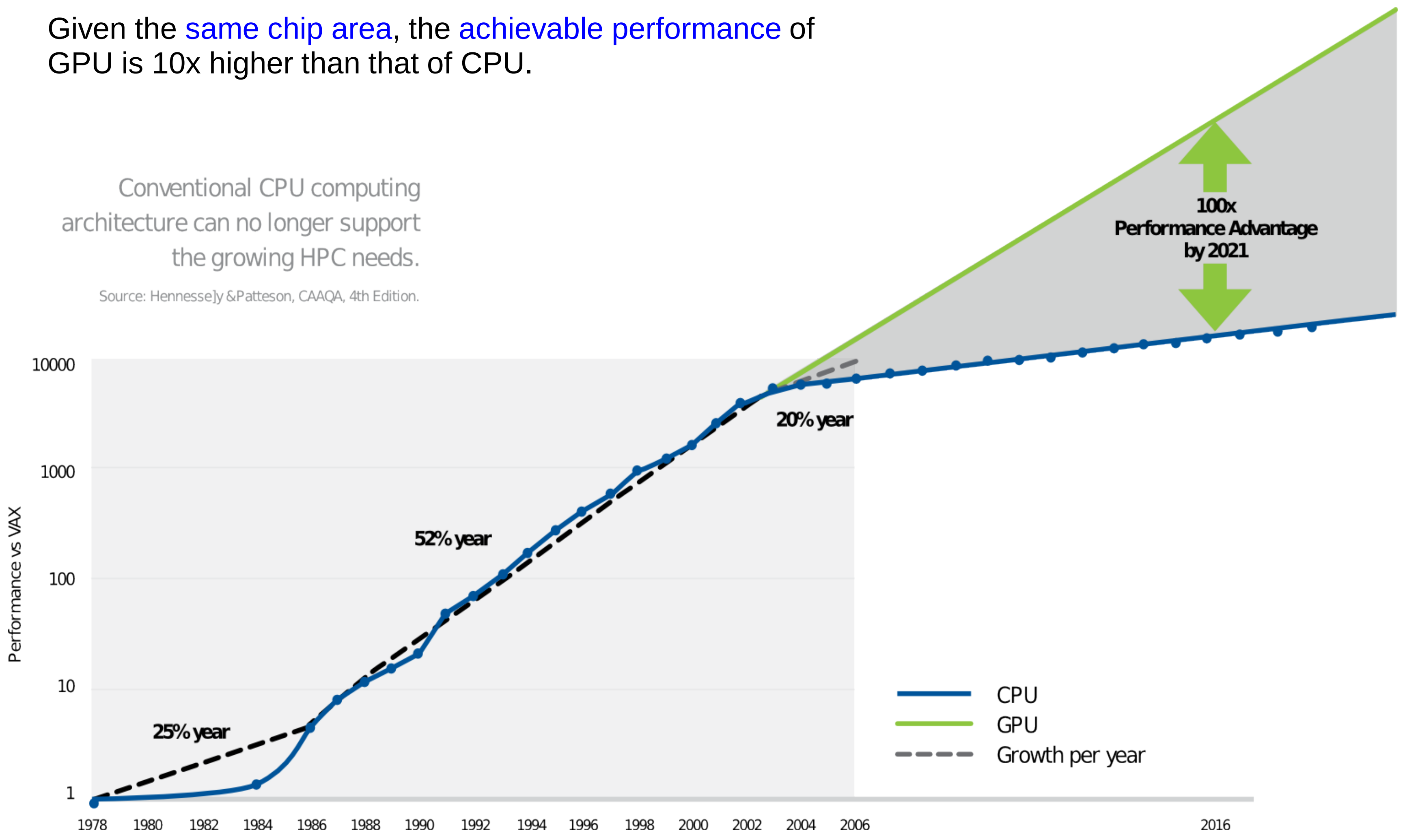

Numerical Packages


Weather, Atmospheric, Ocean Modeling, and Space Sciences

ref: http://www.nvidia.com/object/tesla_computing_solutions.html

The Gap Between CPU and GPU

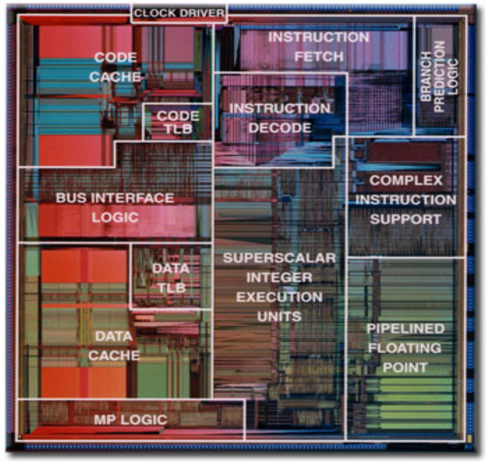
Given the same chip area, the achievable performance of GPU is 10x higher than that of CPU.



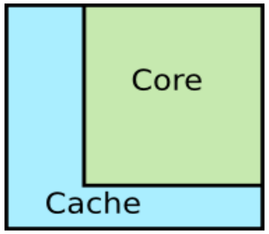
ref: Tesla GPU Computing Brochure

Evolution of Intel Pentium

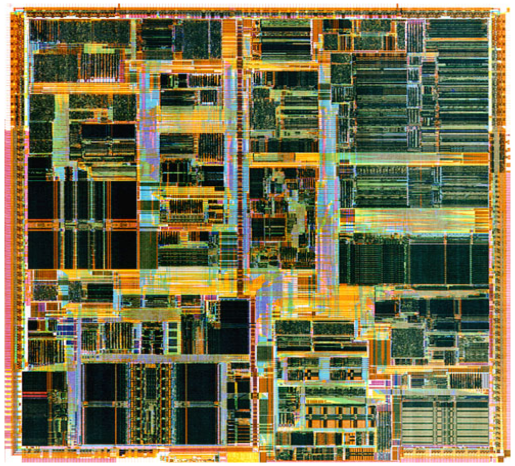
Pentium I

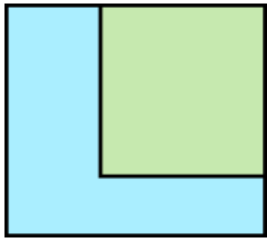


Chip area breakdown

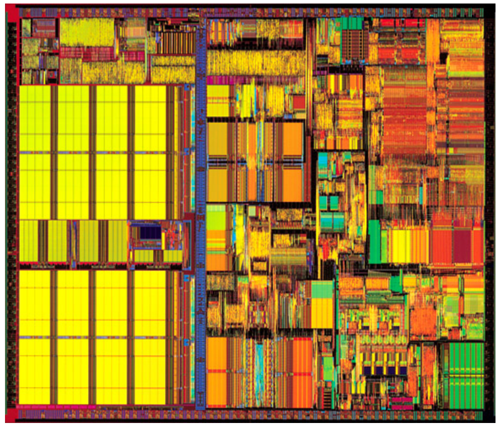


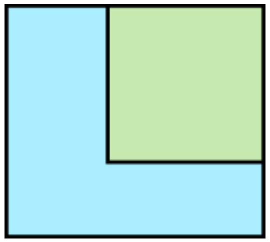
Pentium II



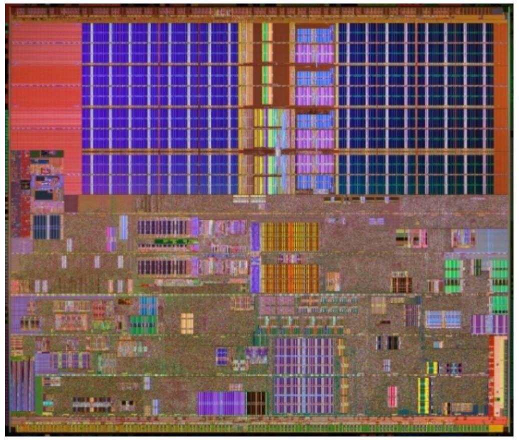


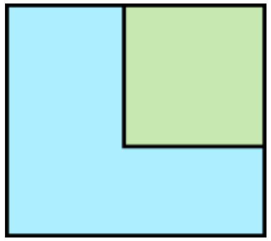
Pentium III





Pentium IV

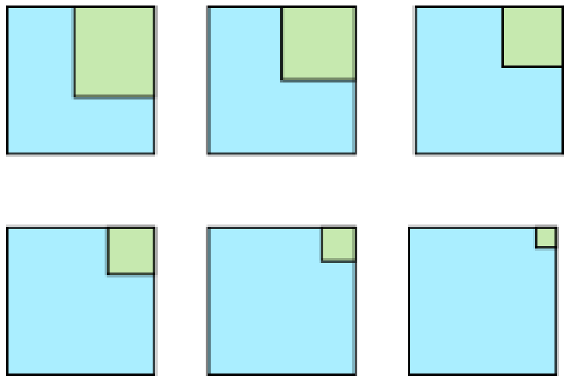




Q: What can you observe? Why?

Extrapolation of Single Core CPU

If we extrapolate the trend, in a few generations, Pentium will look like:

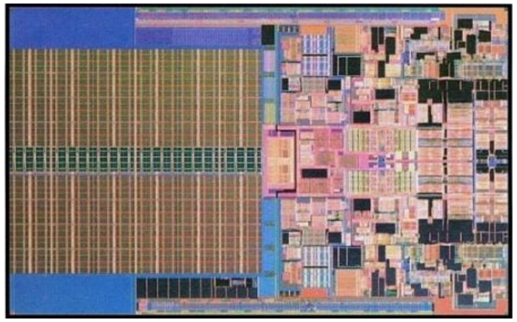


Of course, we know it did not happen.

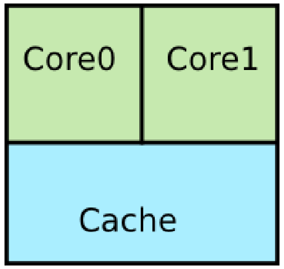
Q: What happened instead? Why?

Evolution of Multi-core CPUs

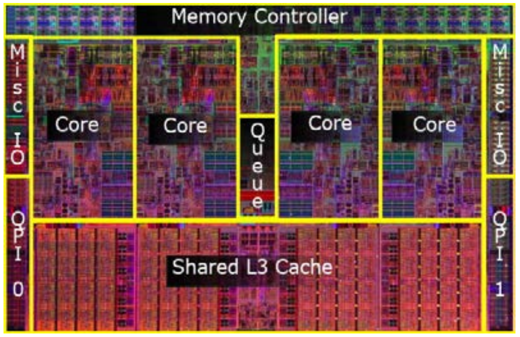
Penryn

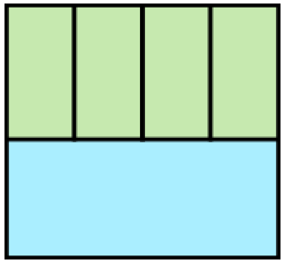


Chip area breakdown

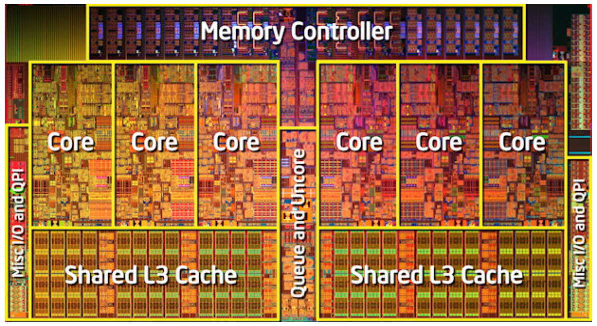


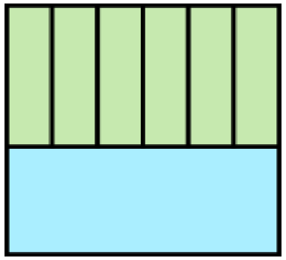
Bloomfield



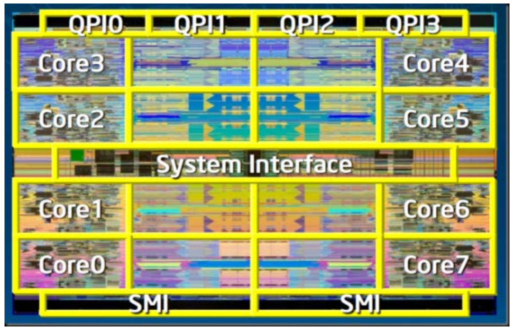


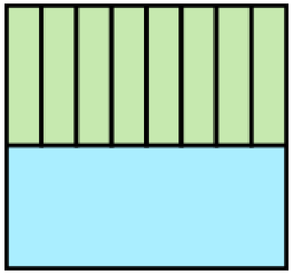
Gulftown





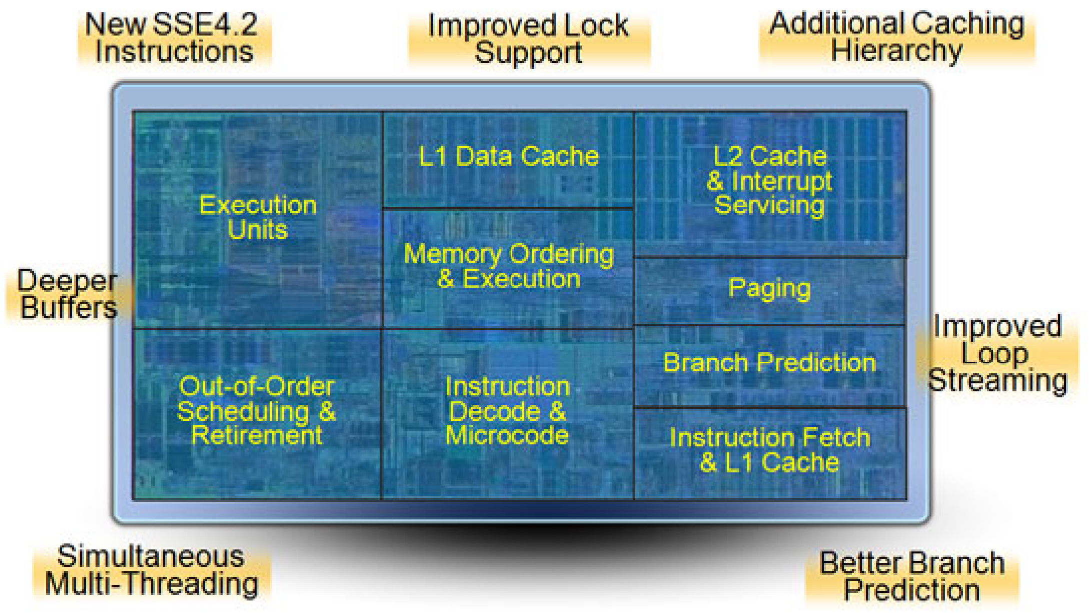
Beckton

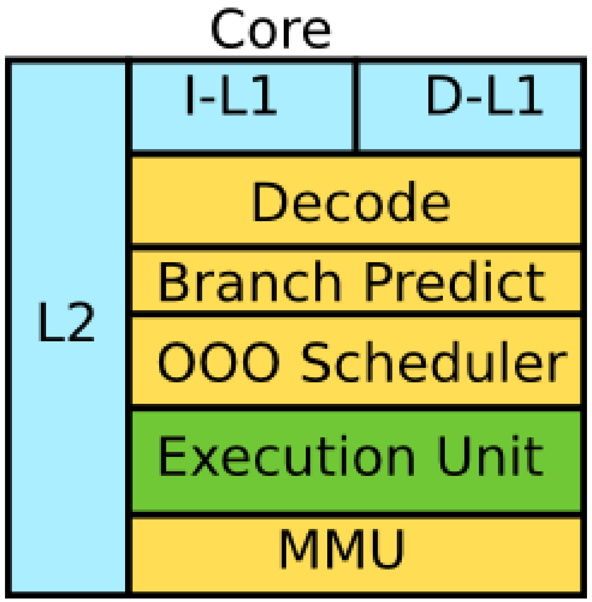




Q: What can you observe? Why?

Let's Take a Closer Look

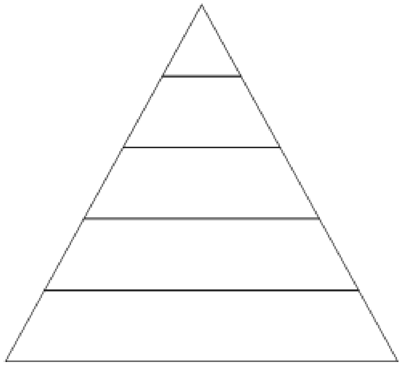

New SSE4.2 Instructions
Improved Lock Support
Additional Caching Hierarchy
Execution Units
L1 Data Cache
L2 Cache & Interrupt Servicing
Memory Ordering & Execution
Paging
Deeper Buffers
Improved Loop Streaming
Out-of-Order Scheduling & Retirement
Instruction Decode & Microcode
Branch Prediction
Instruction Fetch & L1 Cache
Simultaneous Multi-Threading
Better Branch Prediction



Less than 10% of total chip area is used for the real execution.

Q: Why?

The Memory Hierarchy



	Size (Byte)	Energy (pJ)	Delay (cycles)	Bandwidth (GB/s)
Reg	1K	10	1	1000
L1	32K	20	5	100
L2	256K	100	10	100
L3	8M	200	50	100
Off-chip	4G	2000	100	10

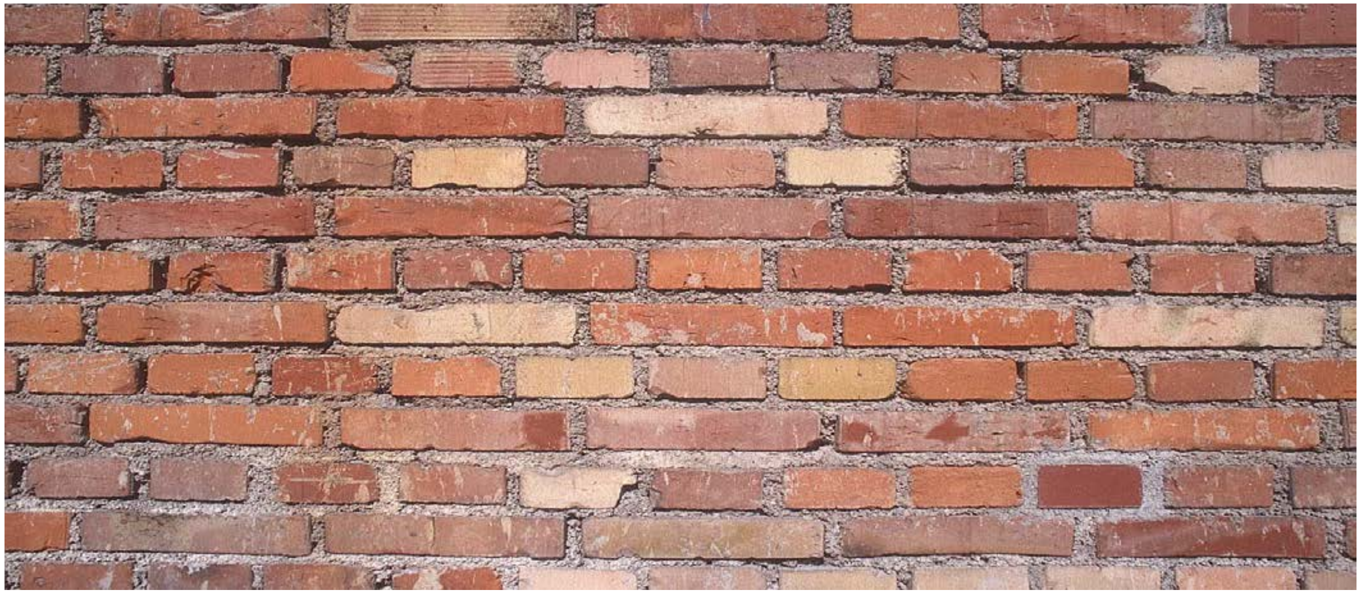
Notes on Energy at 45nm:
64-bit Int ADD takes about 1 pJ.
64-bit FP FMA (fused multiply-add) takes about 200 pJ.

It seems we can not further increase the computational density.

The Brick Wall -- UC Berkeley's View

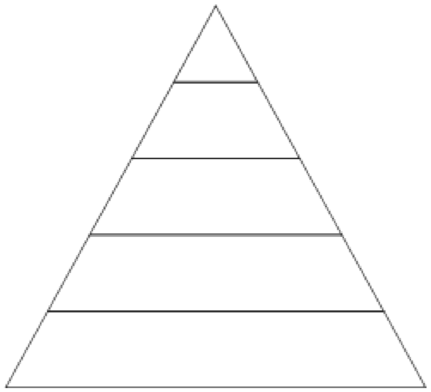
Power Wall: power expensive, transistors free
Memory Wall: Memory slow, multiplies fast
ILP Wall: diminishing returns on more ILP HW

Power Wall + Memory Wall + ILP Wall = Brick Wall



David Patterson, "Computer Architecture is Back - The Berkeley View of the Parallel Computing Research Landscape", Stanford EE Computer Systems Colloquium, Jan 2007, link

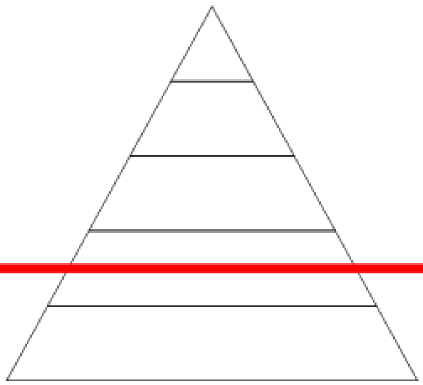
How to Break the Brick Wall?



	Size (Byte)	Energy (pJ)	Delay (cycles)	Bandwidth (GB/s)
Reg	1K	10	1	1000
L1	32K	20	5	100
L2	256K	100	10	100
L3	8M	200	50	100
Off-chip	4G	2000	100	10

Hint: how to exploit the parallelism inside the application?

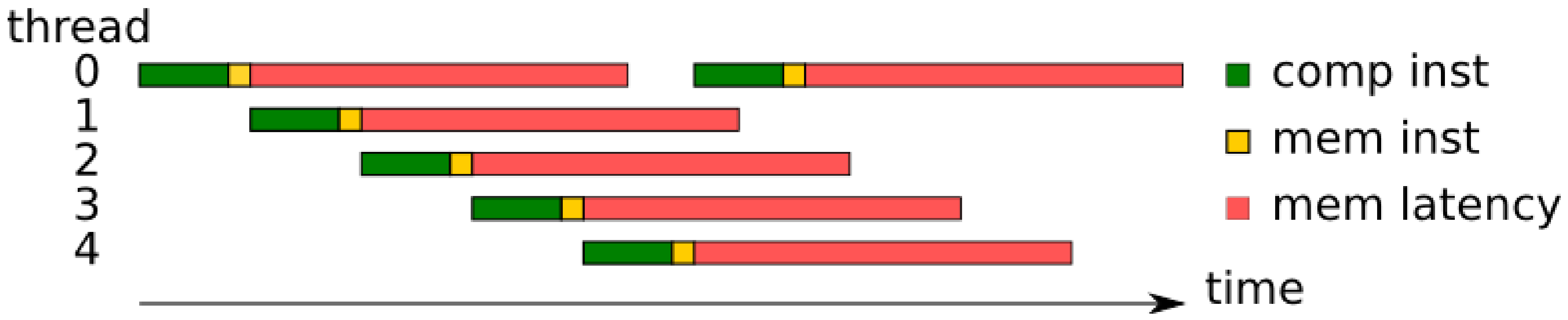
Step 1: Trade Latency with Throughput



	Size (Byte)	Energy (pJ)	Delay (cycles)	Bandwidth (GB/s)
Reg	~~1K~~ 16K	~~10~~ 20	~~1~~ 2~3	1000
L1	32K	20	5	100
L2	256K	100	10	100
~~L3~~	~~8M~~	~~200~~	~~50~~	~~100~~
Off-chip	4G	2000	100	10

Hind the memory latency through fine-grained interleaved threading.

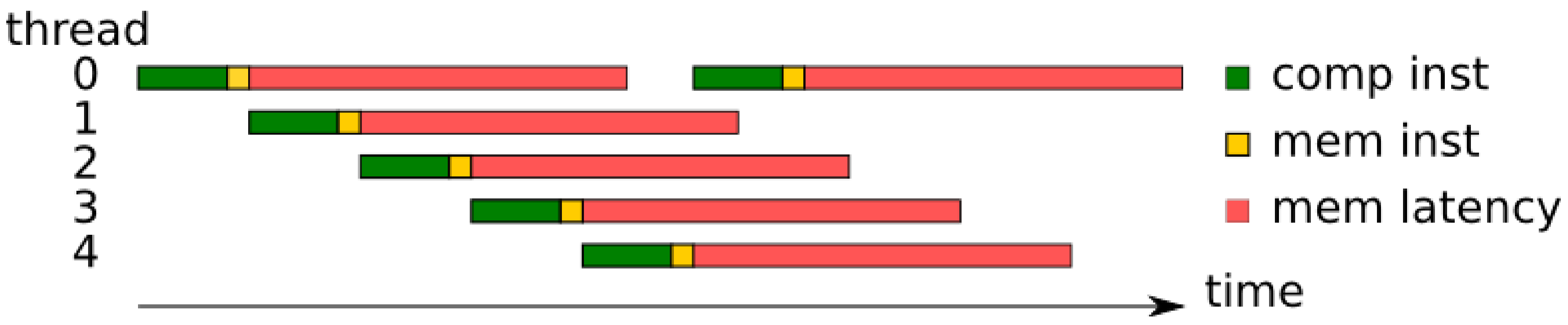
Interleaved Multi-threading



The granularity of interleaved multi-threading:

- 100 cycles: hide off-chip memory latency
- 10 cycles: + hide cache latency
- 1 cycle: + hide branch latency, instruction dependency

Interleaved Multi-threading



The granularity of interleaved multi-threading:

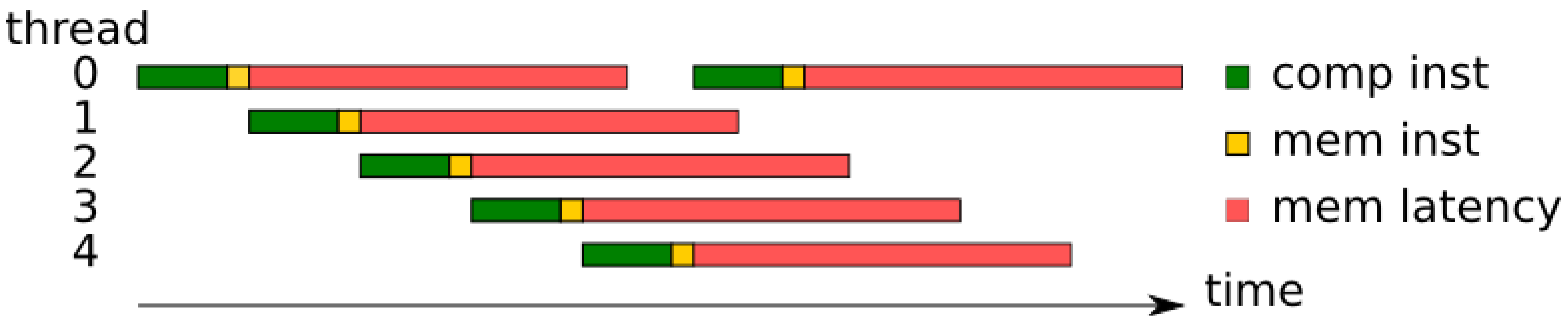
- 100 cycles: hide off-chip memory latency
- 10 cycles: + hide cache latency
- 1 cycle: + hide branch latency, instruction dependency

Fine-grained interleaved multi-threading:

Pros: ?

Cons: ?

Interleaved Multi-threading



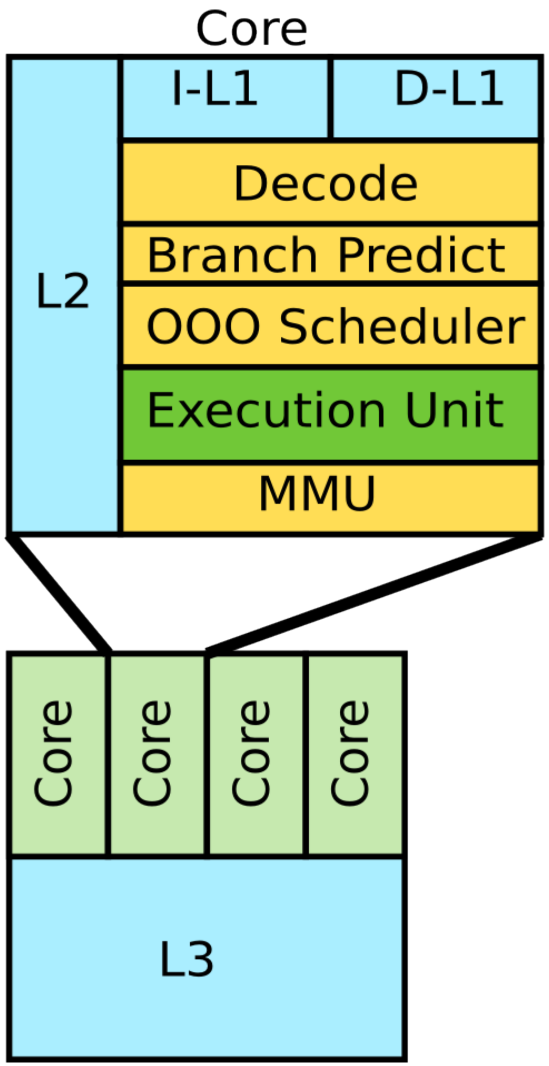
The granularity of interleaved multi-threading:

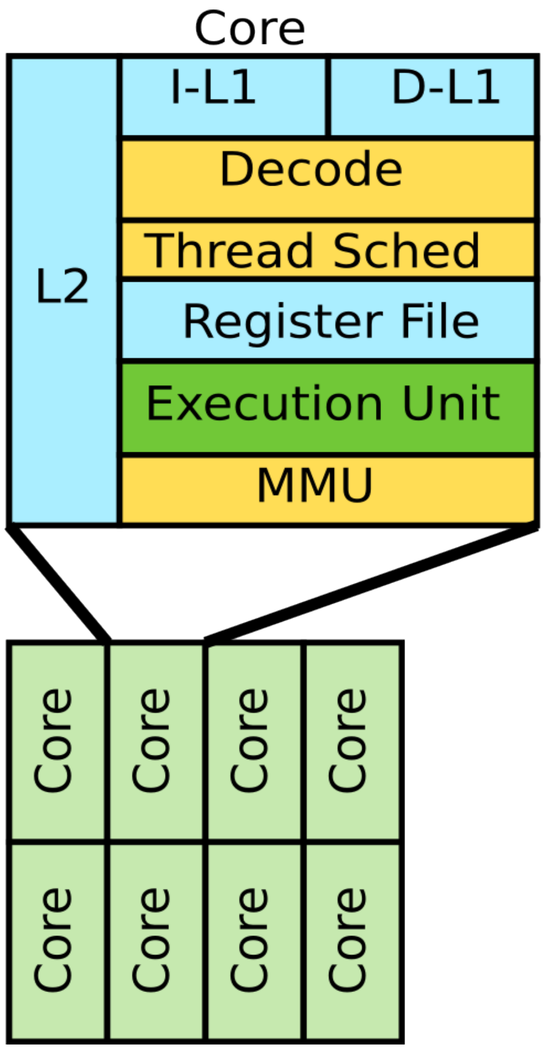
- 100 cycles: hide off-chip memory latency
- 10 cycles: + hide cache latency
- 1 cycle: + hide branch latency, instruction dependency

Fine-grained interleaved multi-threading:
Pros: remove branch predictor, OOO scheduler, large cache
Cons: register pressure, etc.

Fine-Grained Interleaved Threading

Without and with fine-grained interleaved threading



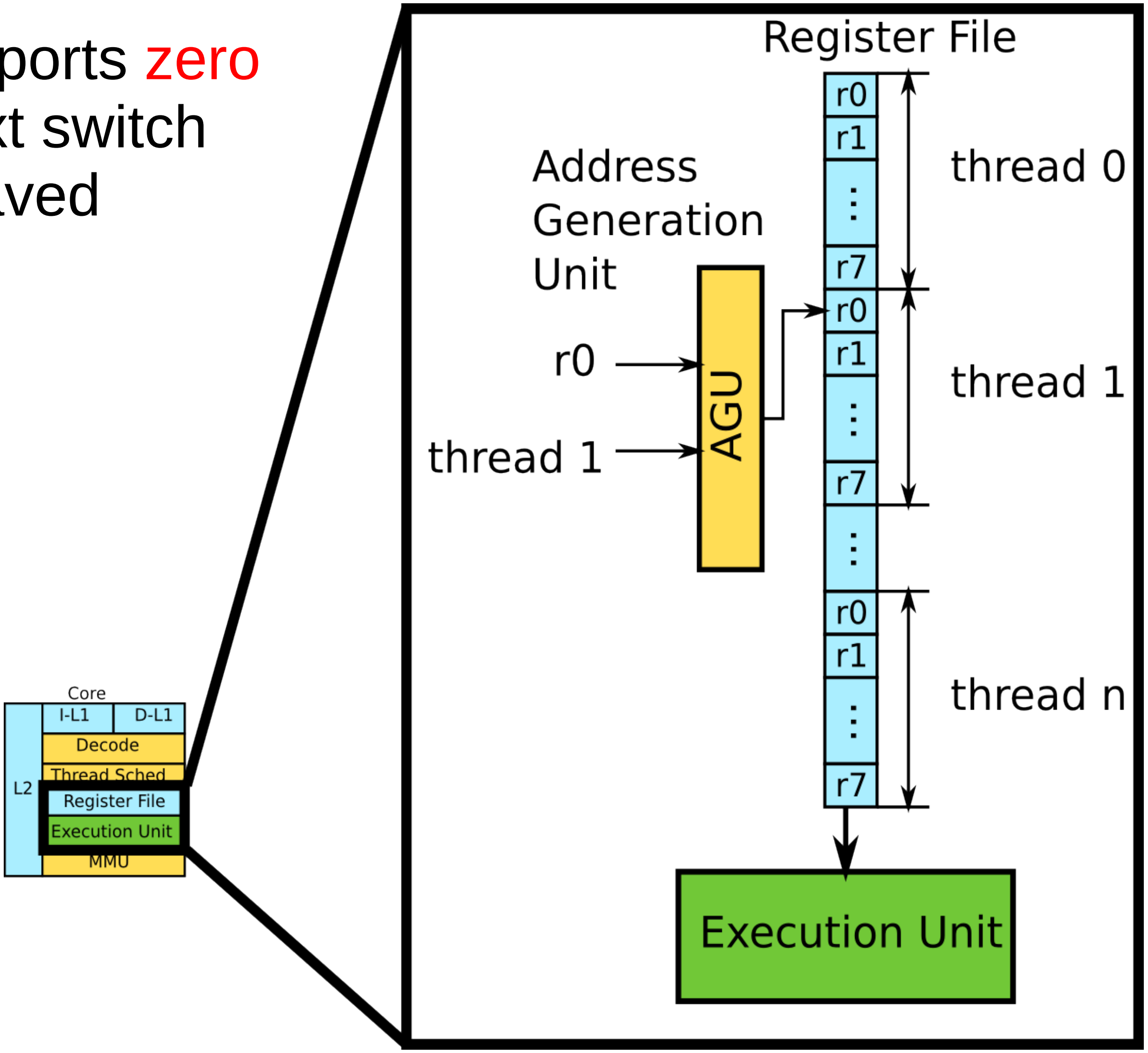


Pros:
reduce cache size,
no branch predictor,
no OOO scheduler

Cons:
register pressure,
thread scheduler,
require huge parallelism

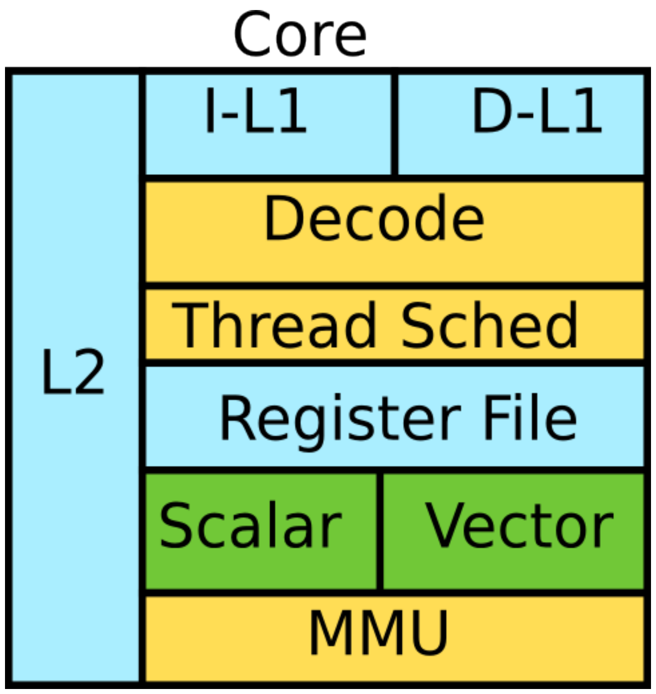
HW Support

Register file supports zero overhead context switch between interleaved threads.



Can We Make Further Improvement?

- Reducing large cache gives 2x computational density.
- Q: Can we make further improvements?

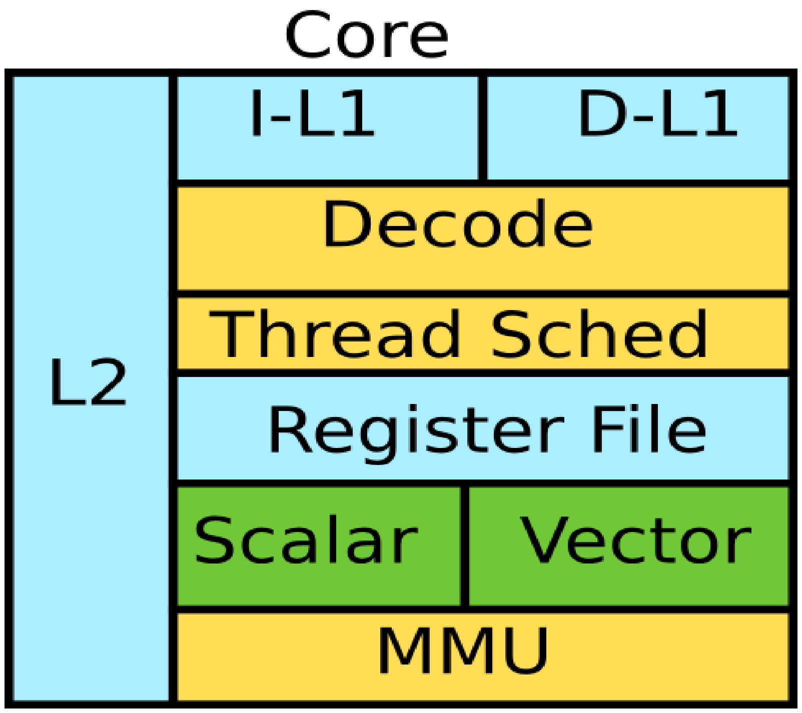


Hint:
We have only utilized thread level parallelism (TLP) so far.

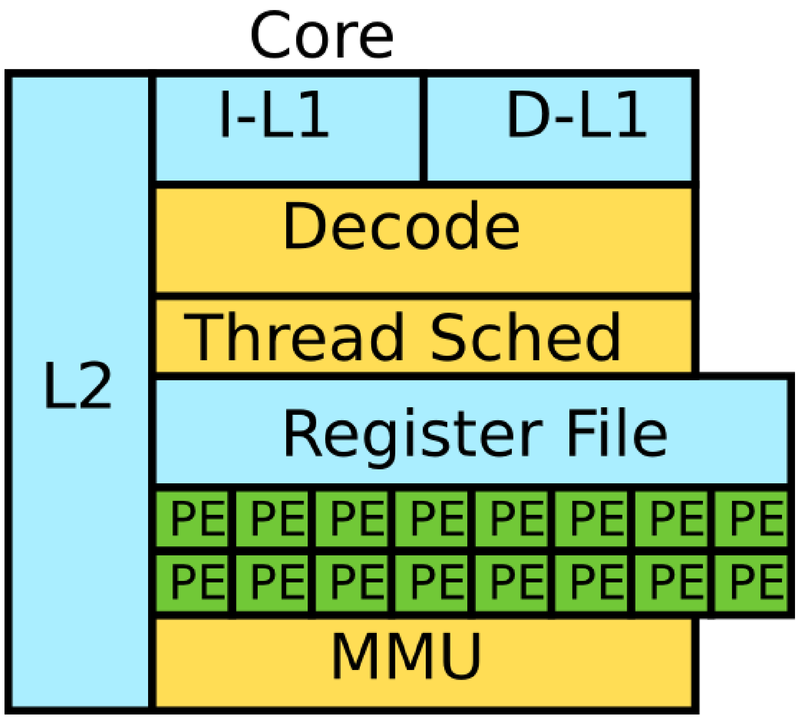
Step 2: Single Instruction Multiple Data

GPU uses wide SIMD: 8/16/24/... processing elements (PEs)
CPU uses short SIMD: usually has vector width of 4.

SSE has 4 data lanes

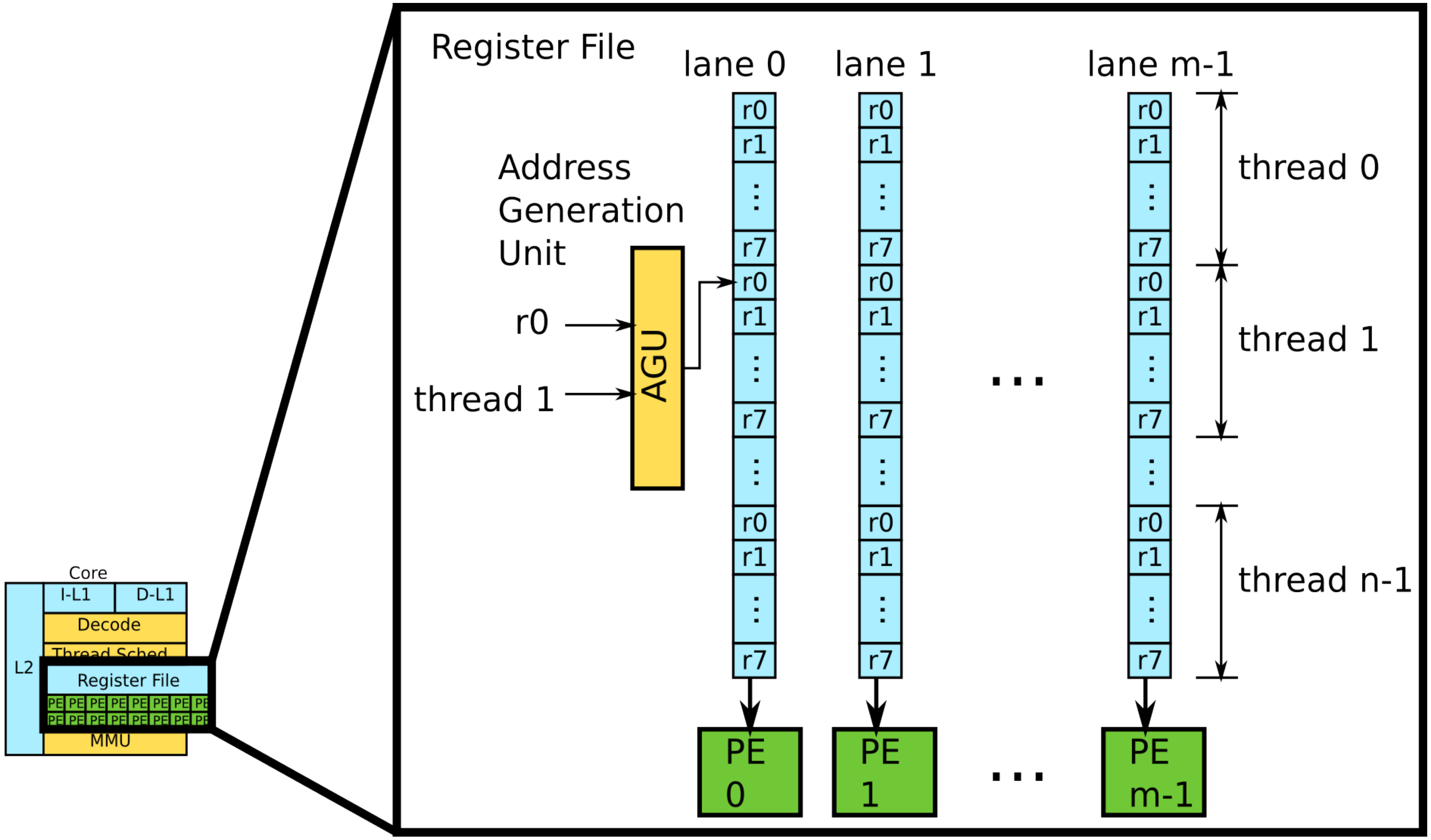


GPU has 8/16/24/... data lanes



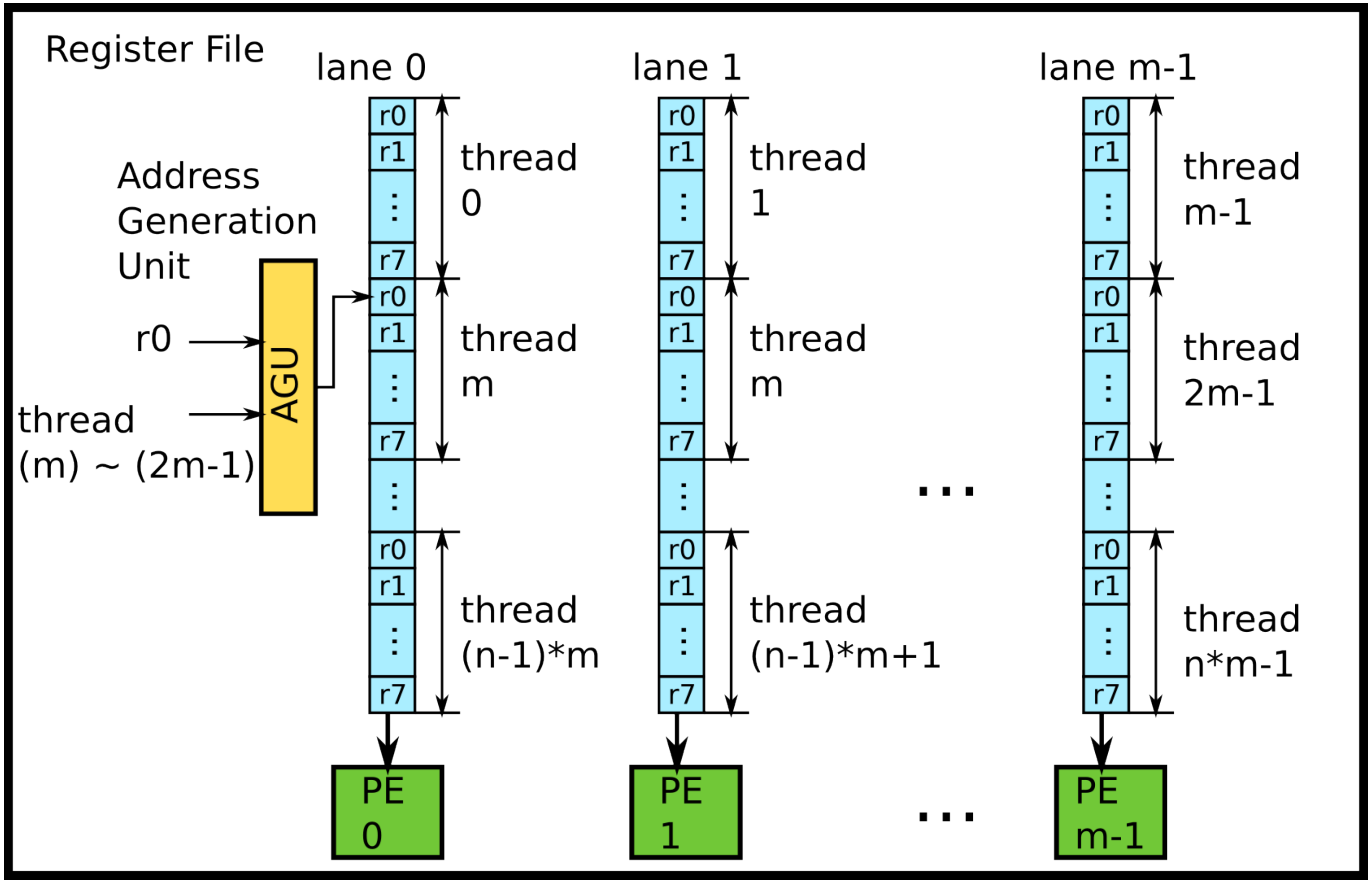
Hardware Support

Supporting interleaved threading + SIMD execution



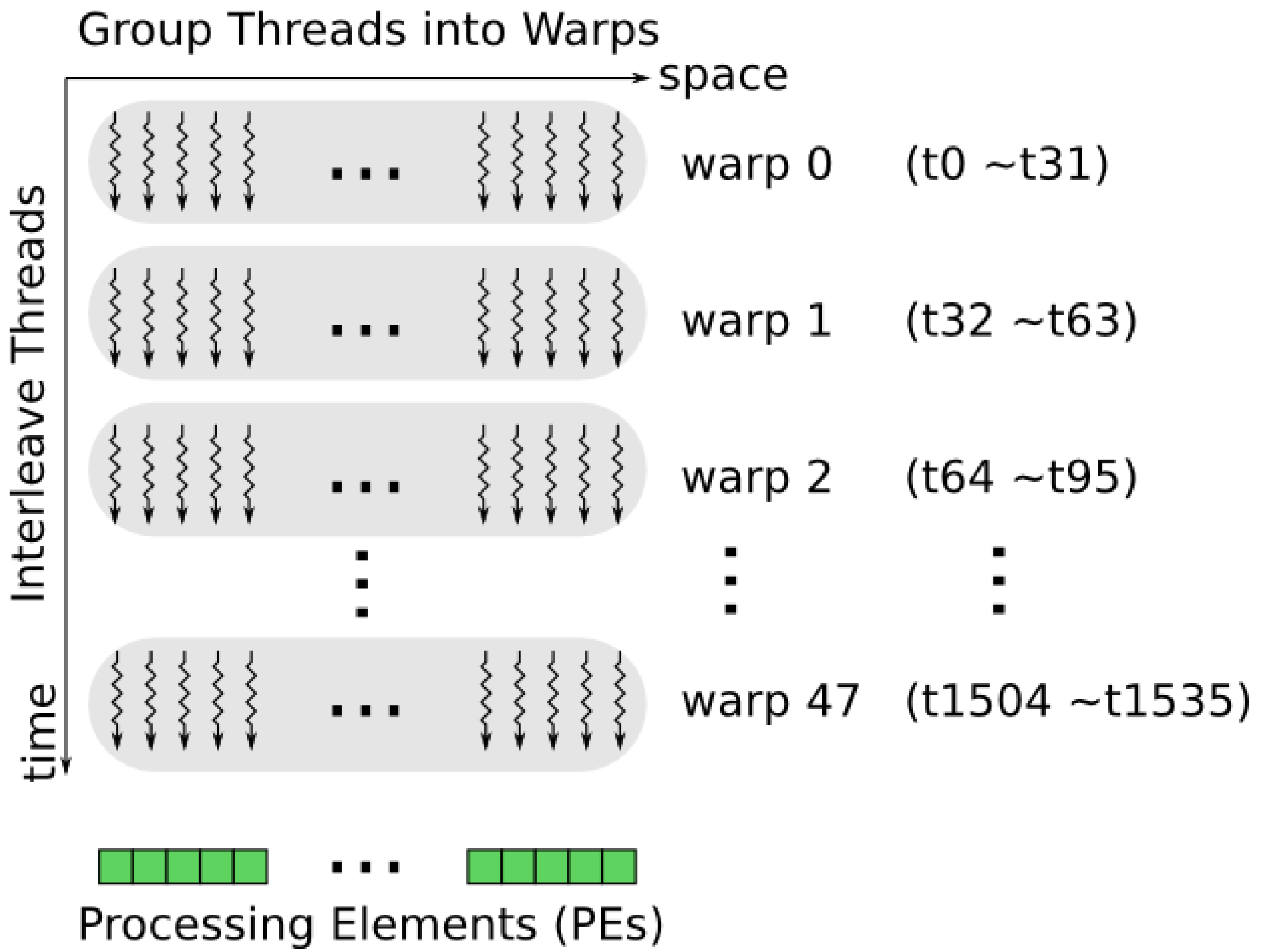
Single Instruction Multiple Thread (SIMT)

Hide vector width using scalar threads.

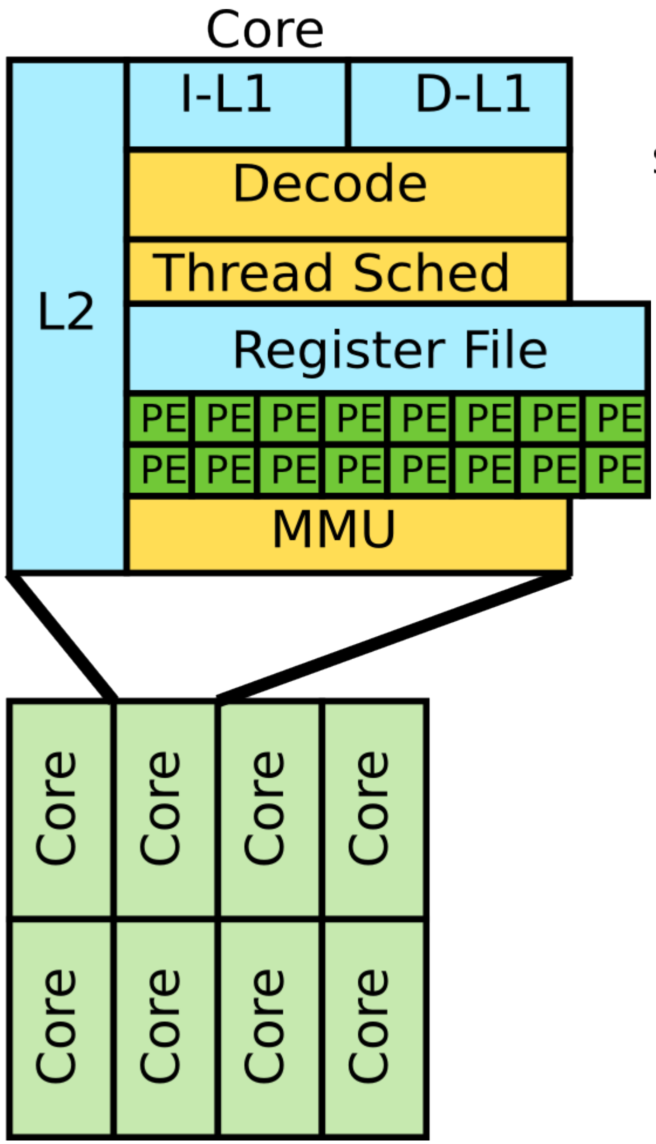


Example of SIMT Execution

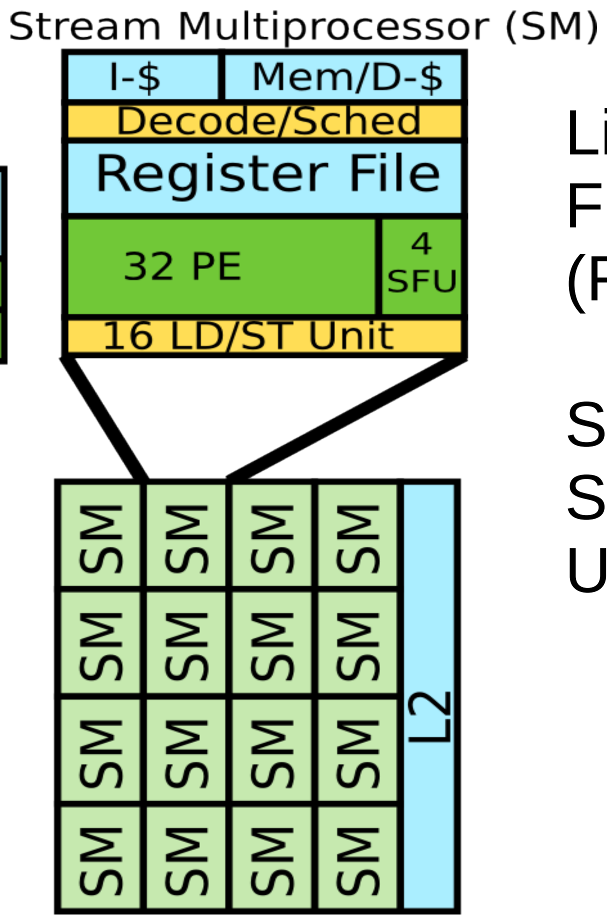
Assume 32 threads are grouped into one warp.



Step 3: Simple Core



The Stream Multiprocessor (SM) is a light weight core compared to IA core.



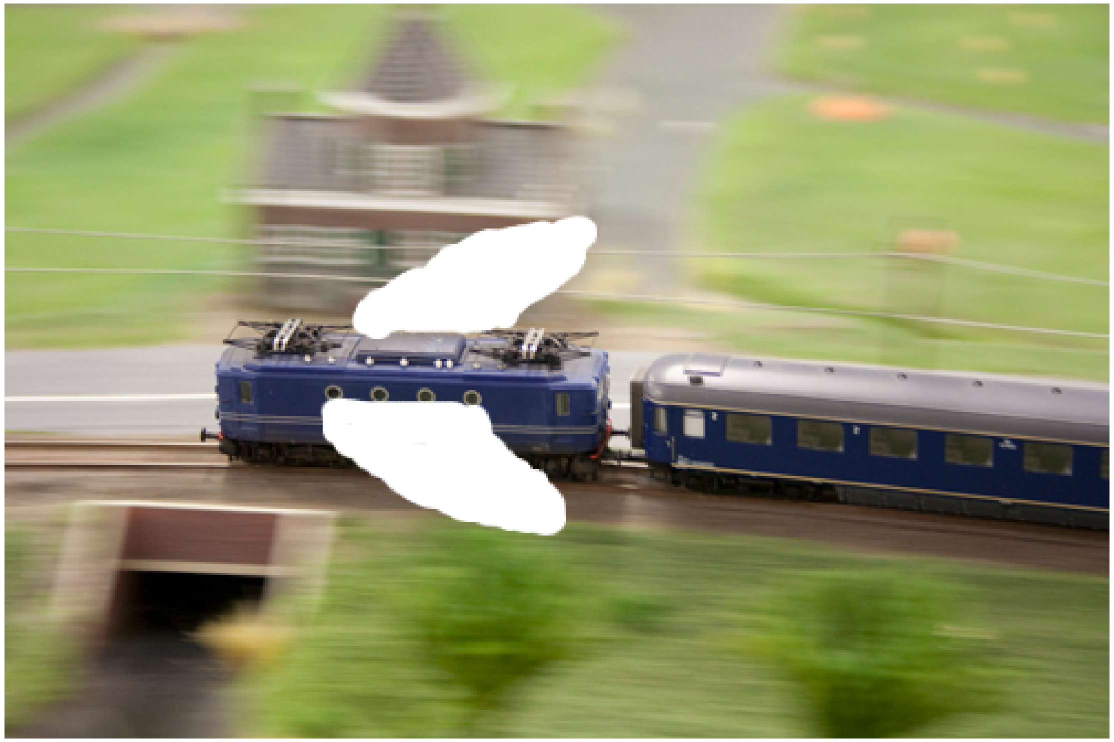
Light weight PE: Fused Multiply Add (FMA)

SFU: Special Function Unit

NVIDIA's Motivation of Simple Core

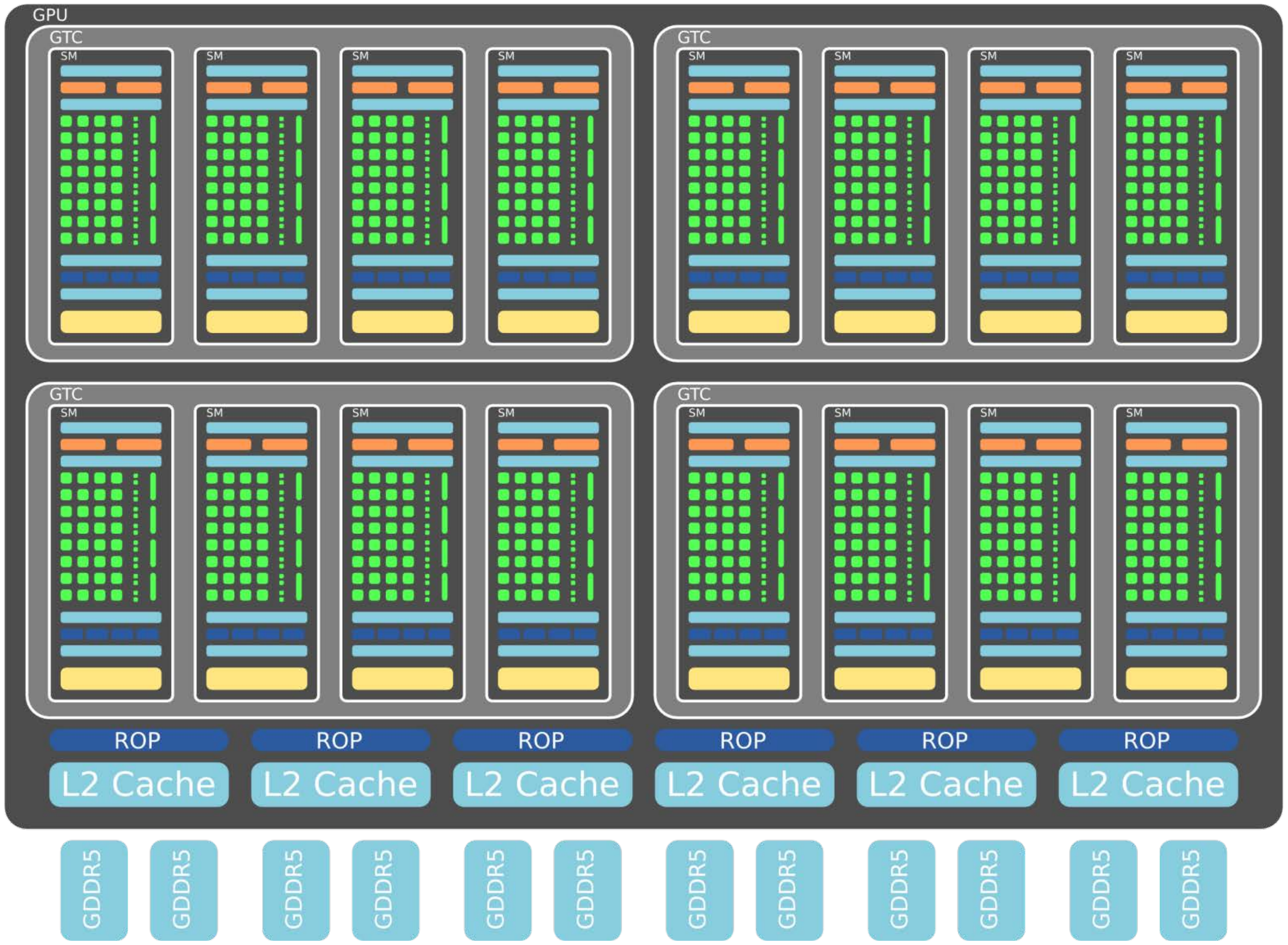
"This [multiple IA-core] approach is analogous to trying to build an airplane by putting wings on a train."

--Bill Dally, NVIDIA



Review: How Do We Reach Here?

NVIDIA Fermi, 512 Processing Elements (PEs)



Throughput Oriented Architectures

1. Fine-grained interleaved threading (~2x comp density)
2. SIMD/SIMT (>10x comp density)
3. Simple core (~2x comp density)

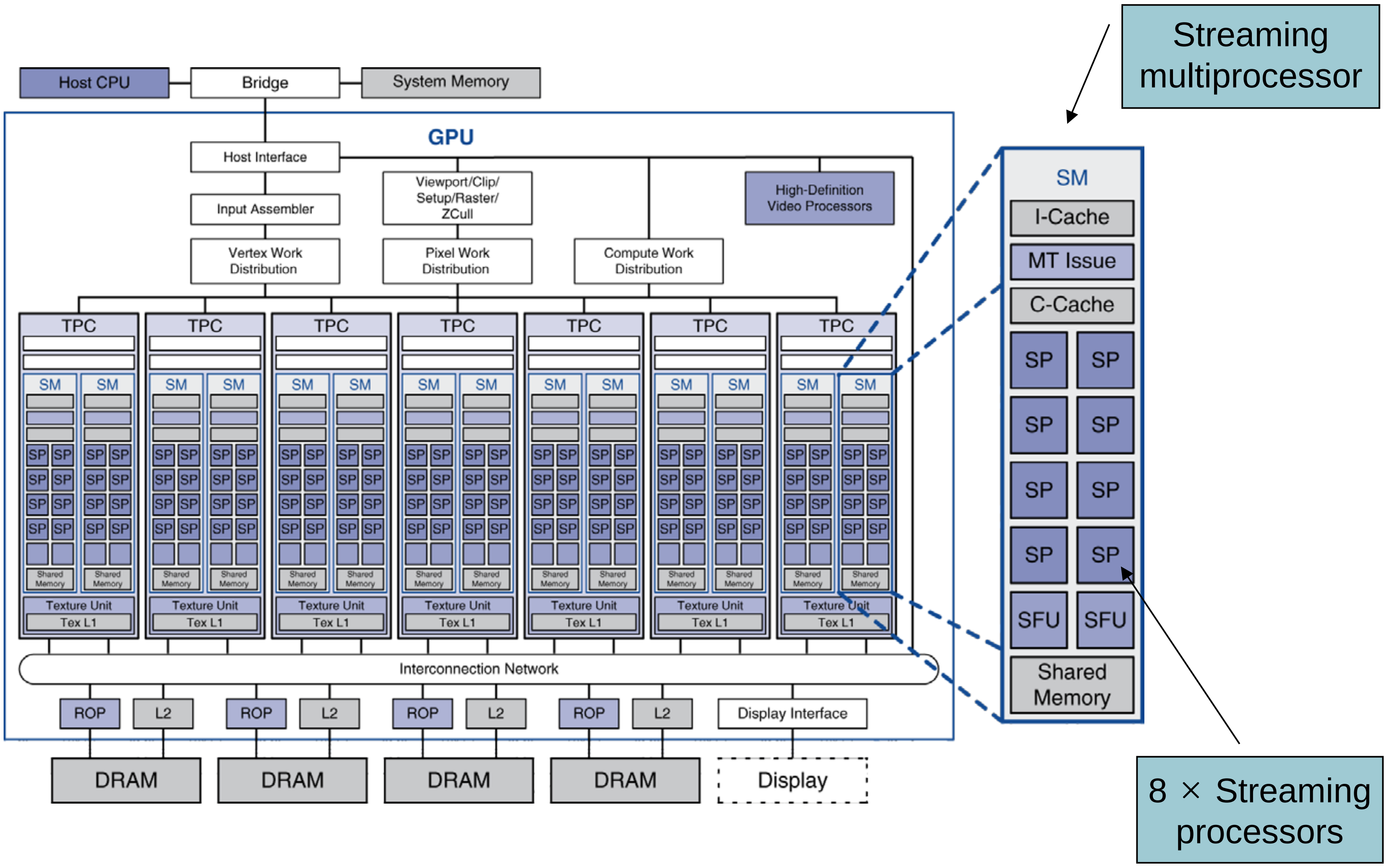
Key architectural features of throughput oriented processor.

ref: Michael Garland. David B. Kirk, "Understanding throughput-oriented architectures", CACM 2010. (link)

NVIDIA Tesla

- Tesla-based GPUs chips are offered with between 1 and 16 nodes, which NVIDIA calls multiprocessors
  - GeForce 8800 GTX has 16 multiprocessors and a clock rate of 1.35 GHz
  - Each multiprocessors contains 8 multithreaded single-precision floating-point units and integer units, which NVIDIA calls streaming processors.

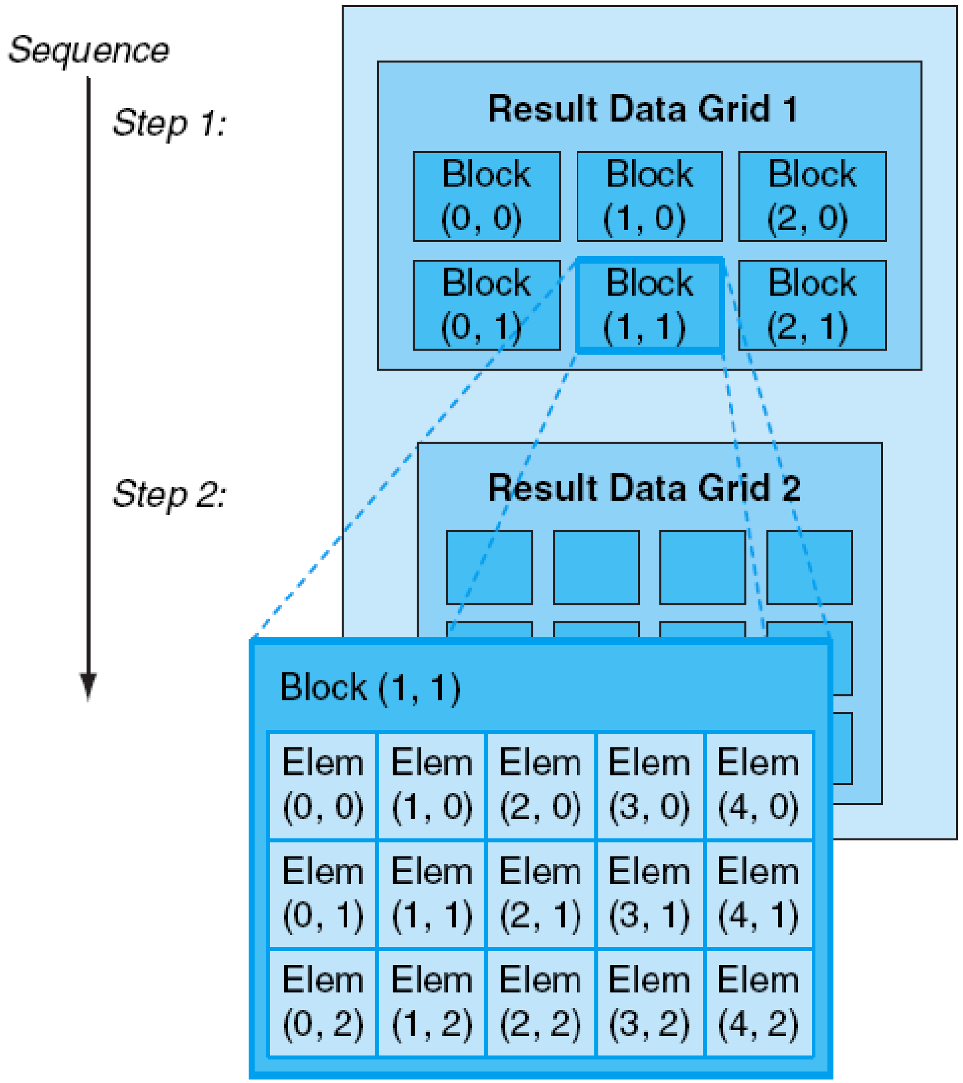
NVIDIA Tesla



Data parallel problem

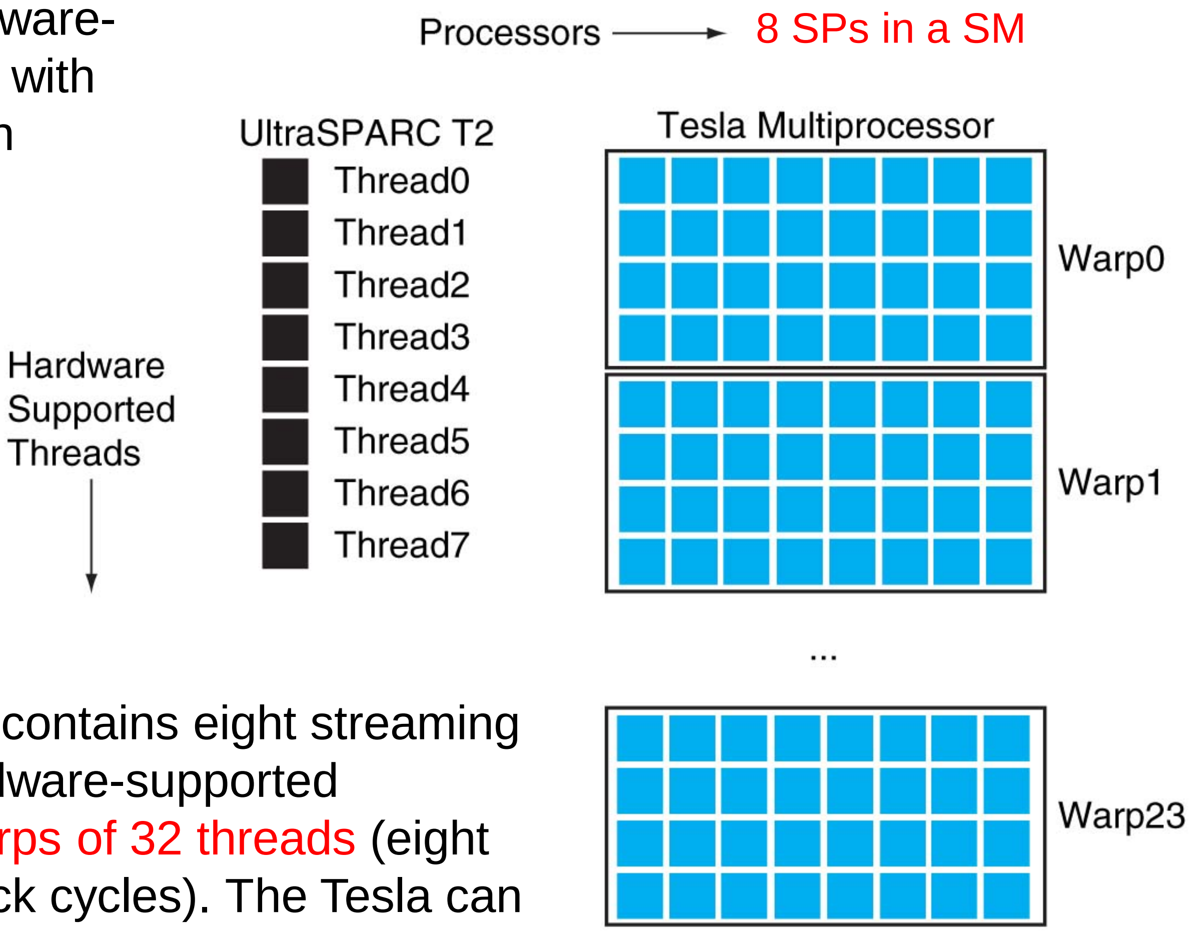
- Decompose the problem into many small problems that can be solved in parallel.
- Each multiprocessor computes a block, and each thread computes an element

- Grid: a set of blocks
- Block: a set of threads
- Element: a thread



NVIDIA Tesla & Sun T2

The T2 core is a single processor and uses hardware-supported multithreading with eight threads. The T2 can switch every clock cycle.



The Tesla multiprocessor contains eight streaming processors and uses hardware-supported multithreading with 24 warps of 32 threads (eight processors times four clock cycles). The Tesla can switch only every two or four clock cycles.

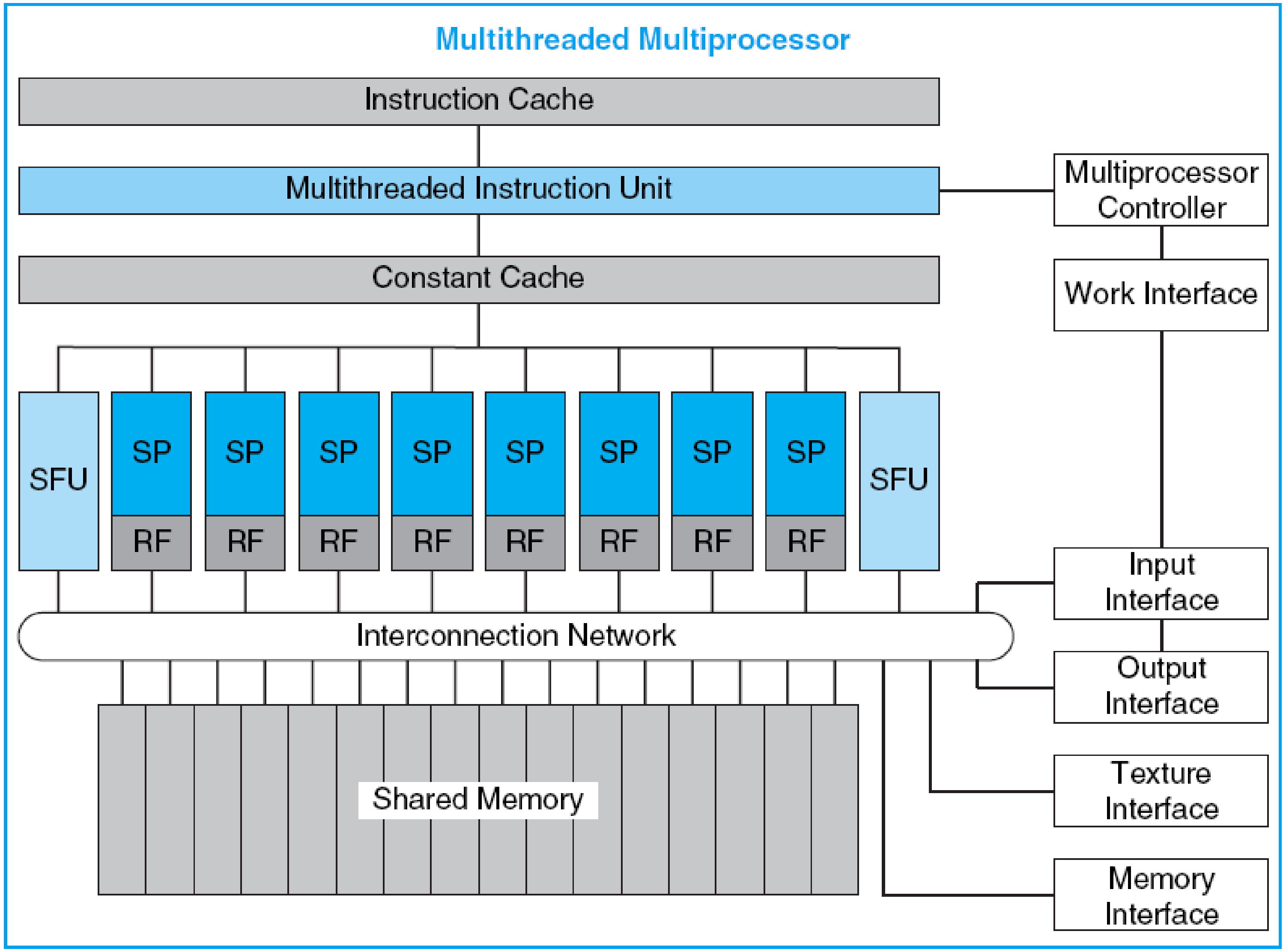
SIMT

- The best performance comes when all 32 threads of a warp execute together in a SIMD-like fashion, which Tesla architecture calls single-instruction multiple-thread (SIMT).
  - SIMT dynamically discovers
    - which threads can execute the same instruction together, and
    - which threads are idle that cycle.
  - The minimum unit of switching warps is two clock cycles across eight streaming processors (SPs).
  - With this restriction, the hardware is much simpler
- Each GPU thread has its own private registers, private per-thread memory, program counter, and thread execution state
  - Can execute an independent code path

Streaming Processor (SP)

- Each SP is hardware multithreaded, supporting up to 64 threads
  - Limited by the number of registers
  - Its 1024 registers are partitioned among the assigned threads
- Programs declares their register demand
  - Typically 16 ~ 64 scalar 32-bit registers per thread
  - Each SP supports 32 threads when 32 registers are required per thread, which corresponds to 256 threads per multiprocessor.
- Each SP core execute an instruction for 4 individual threads of a warp using 4 clocks.

Tesla Multiprocessor



SFU: special function unit

CUDA Paradigm

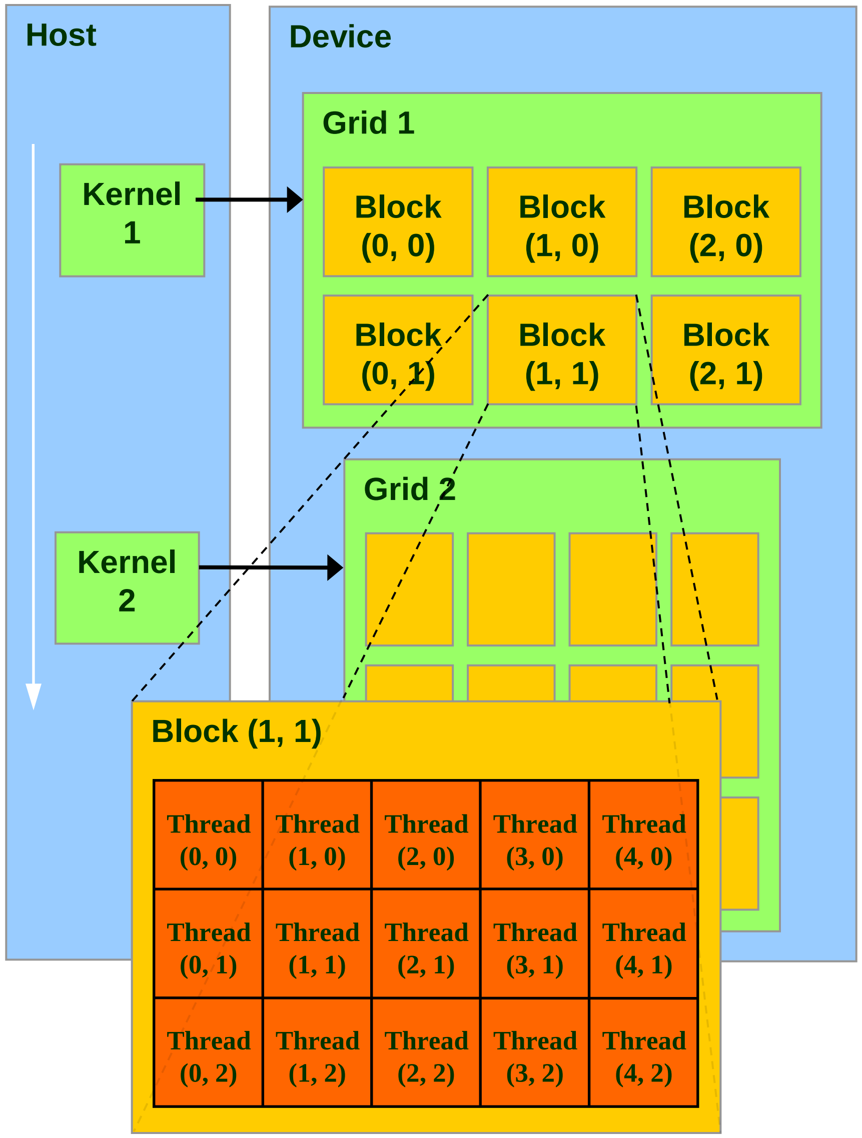
- CUDA provides three key abstractions
  - A hierarchy of thread groups
  - Shared memories
  - Barrier synchronization
- The programmer writes a serial program that calls parallel kernels
- The programmer organizes these threads into a hierarchy of thread blocks and grids of thread blocks.
- A kernel: a program or function for one thread, designed to be executed by many threads.

CUDA Paradigm

- CUDA provides three key abstractions
  - A hierarchy of thread groups
  - Shared memories
  - Barrier synchronization
- The programmer writes a serial program that calls parallel kernels
- The programmer organizes these threads into a hierarchy of thread blocks and grids of thread blocks.
- A kernel: a program or function for one thread, designed to be executed by many threads.

Grids and Blocks

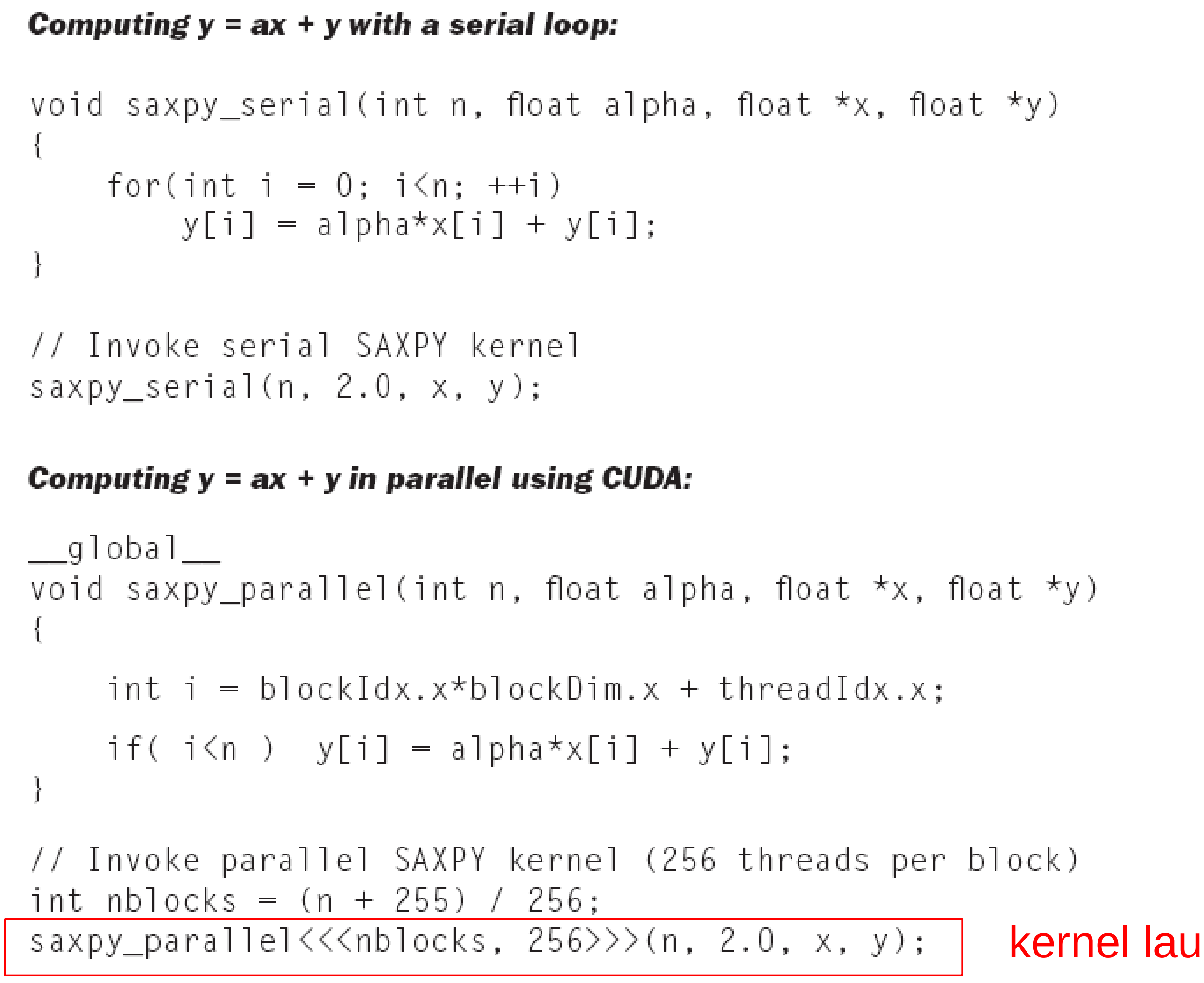
- A kernel is executed as a grid of thread blocks
  - All threads share data memory space
- A thread block is a batch of threads that can cooperate with each other by:
  - Synchronizing their execution
    - For hazard-free shared memory accesses
  - Efficiently sharing data through a low latency shared memory
- Two threads from two different blocks cannot cooperate



CUDA Paradigm

- When invoking a kernel, the programmer specifies the number of threads per block and the number of blocks comprising the grid.
- Each thread is given a unique thread ID number and each thread block is given a unique block ID number.
- _global_ declaration specifier indicates that the procedure is a kernal entry point
- CUDA program launch parallel kernels with the extended function call syntax
  - kernel_specifier <<<dimGrid, dimBlock>>>
    (.., param list,);

C & CUDA Codes for SAXPY



Computing y = ax + y with a serial loop:

```
void saxpy_serial(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i<n; ++i)
        y[i] = alpha*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Computing y = ax + y in parallel using CUDA:

```
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    if( i<n )  y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

kernel launching

FIGURE A.3.4 Sequential code (top) in C versus parallel code (bottom) in CUDA for SAXPY

CUDA programming

- Parallel execution and thread management are automatic.
  - All thread creation, scheduling, and termination is handled for the programmer by the underlying system.
  - A Tesla architecture GPU performs all thread management directly in hardware.
  - The thread of a block execute concurrently and may synchronize at a synchronization barrier by calling _syncthreads() intrinsic.
  - Threads in a block, which should reside on the same multiprocessor, may communicate with each other by writing and reading per-block shared memory at a synchronization barrier.

CUDA programming

- A CUDA program is a unified C/C++ program for a heterogeneous CPU and GPU system.
- It execute on the CPU and dispatches parallel work to the GPU.
  - This work consists of a data transfer from main memory and a thread dispatch.
  - A thread is a piece of the program for the GPU.
- Programmers specify the number of threads in a thread block and the number of thread blocks they wish to start executing on the GPU.
  - All the threads in the thread block are scheduled to run on the same multiprocessor so they all share the same local memory.

CUDA programming

    - Thus, they can communicate via loads and stores instead of messages.
- The CUDA compiler allocates registers to each thread, under the constraints that the registers per thread time thread per thread block does not exceed the 8192 registers per multiprocessors.
- A thread block can be up to 512 threads
    - Each group of 32 threads in a thread block is packed into warps.