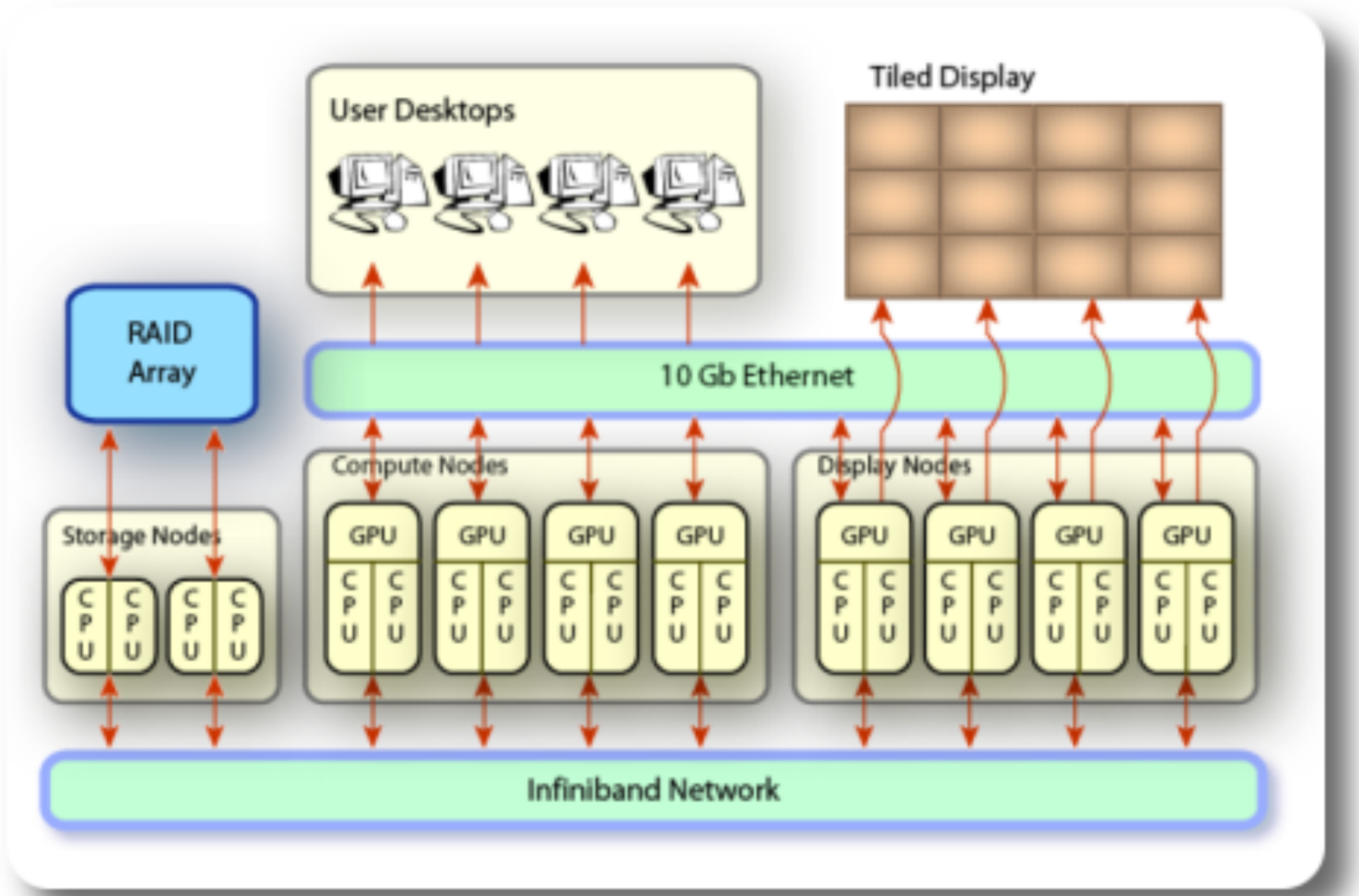
Introduction to GPU Programming Languages

Maryland CPU/GPU Cluster Infrastructure



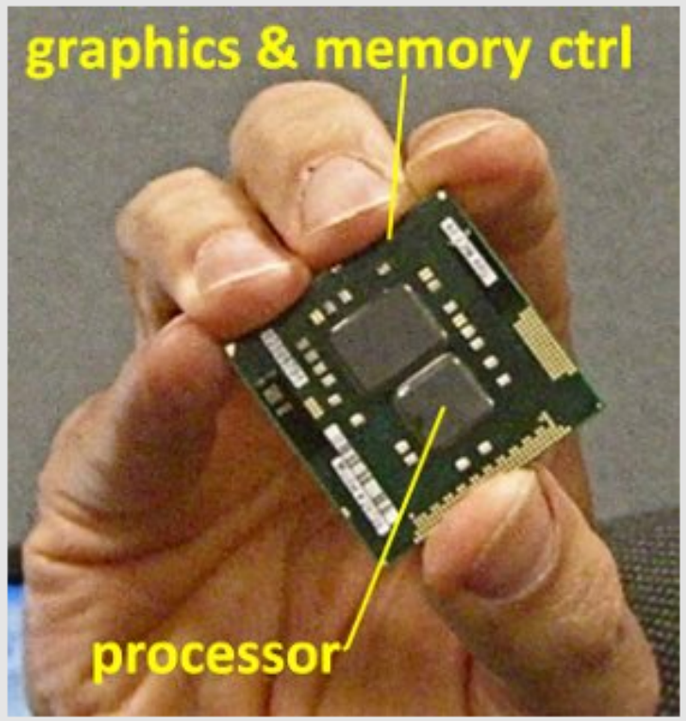
http://www.umiacs.umd.edu/
research/GPU/facilities.html

Intel's Response to NVIDIA GPUs

iSGTW INTERNATIONAL SCIENCE GRID THIS WEEK

Feature - GPU-based cheap supercomputing coming to an end

Nvidia's CUDA has been hailed as "Supercomputing for the Masses," and with good reason – amazing speedups ranging from 10x through hundreds have been reported on scientific / technical code. CUDA has become a darling of academic computing and a major player in DARPA's Exascale program, but performance alone does not account for that popularity: price clinches the deal. For all that computing power, they're incredibly cheap. As Sharon Glotzer of UMich noted, "Today you can get two gigaflops for $500. That is ridiculous." It is indeed. And it's only possible because CUDA is subsidized by sinking the fixed costs of its development into the high volumes of Nvidia's mass market low-end GPUs.



Intel's Sandy Bridge architecture places the processor and GPU on the same chip.

Image courtesy Greg Pfister.

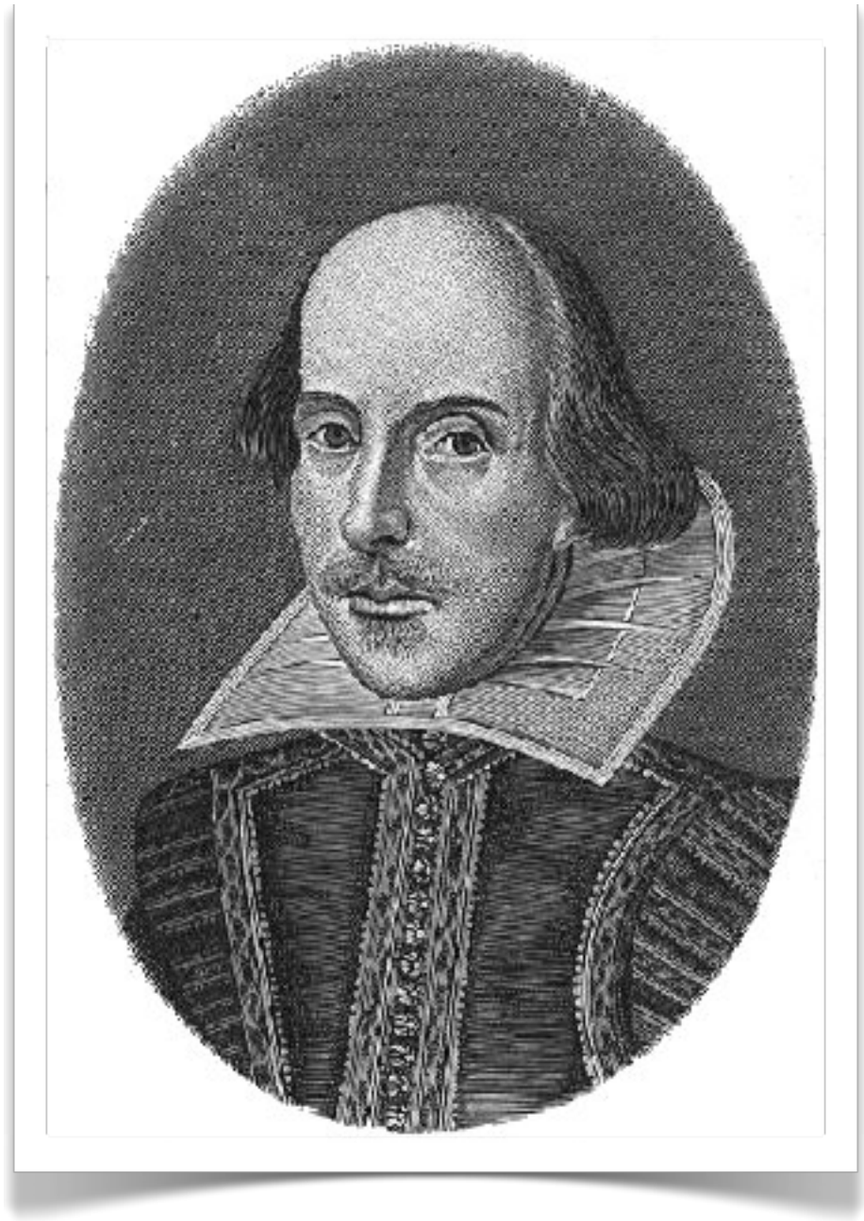
Unfortunately, that subsidy won't last forever; its end is now visible. Intel has now started pounding the marketing drums on something long predicted: integration of Intel's graphics onto the same die as its next generation "Sandy Bridge" processor chip, due out in mid-2011.

Probably not coincidentally, mid-2011 is when AMD's Llano processor will see daylight. It incorporates enough graphics-related processing to be an apparently decent DX11 GPU, although to my knowledge the architecture hasn't been disclosed in detail.

Just prior to this Fall's IDF (Intel Developer Forum), Anandtech received an early demo part of Sandy Bridge and checked out the graphics, among other things. Their net is that

To Accelerate Or Not To Accelerate

- Pro:
  - They make your code run faster.
- Cons:
  - ~~They're expensive.~~
  - They're hard to program.
  - ~~Your code may not be cross-platform.~~



When is GPUs appropriate?

- Applications
  - Traditional GPU Applications: Gaming, image processing
    - i.e., manipulating image pixels, oftentimes the same operation on each pixel
  - Scientific and Engineering Problems: physical modeling, matrix algebra, sorting, etc.

- Data parallel algorithms:
  - Large data arrays
  - Single Instruction, Multiple Data (SIMD) parallelism
  - Floating point computations

Parallel Hardware Landscape: Instruction and Data Streams

- Flynn's Classification: Hardware dimensions of memory and control

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: GPUs
	Multiple	MISD: No Examples Today	MIMD: Intel Nalhelm

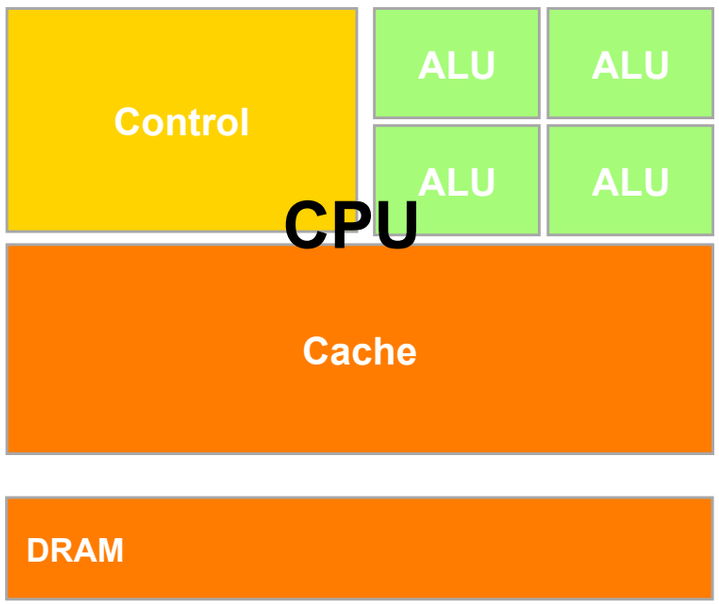
Single Instruction, Multiple Data (SIMD)

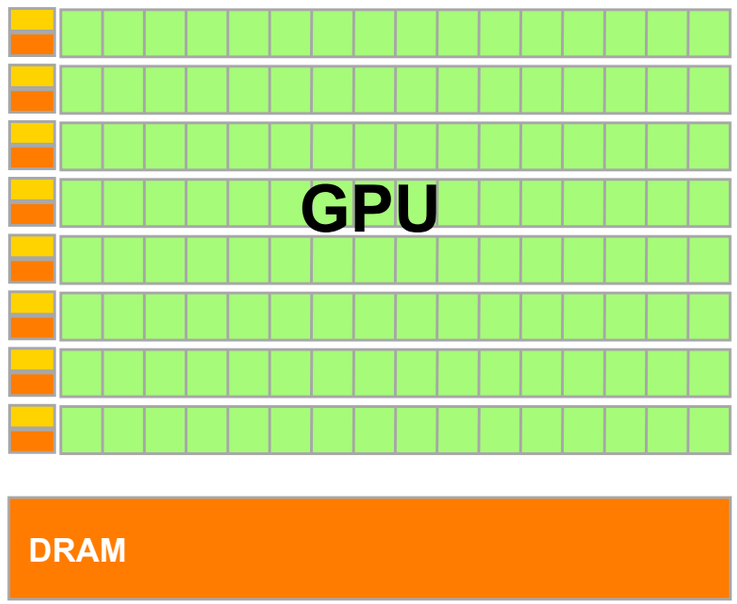
Example Instruction:
A[i][j] = A[i][j]++;

- Element-wise operations on vectors or matrices of data.
- Multiple processors: All processors execute the same set of instructions at the same time
  - Each with data at a different address location.
- Advantages:
  - Simplifies synchronization
  - Reduces instruction control hardware; One program, many results.
  - Works best for highly data-parallel applications (e.g., matrix operations, monte carlo calculations)

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

CPU vs. GPU Hardware Design Philosophies

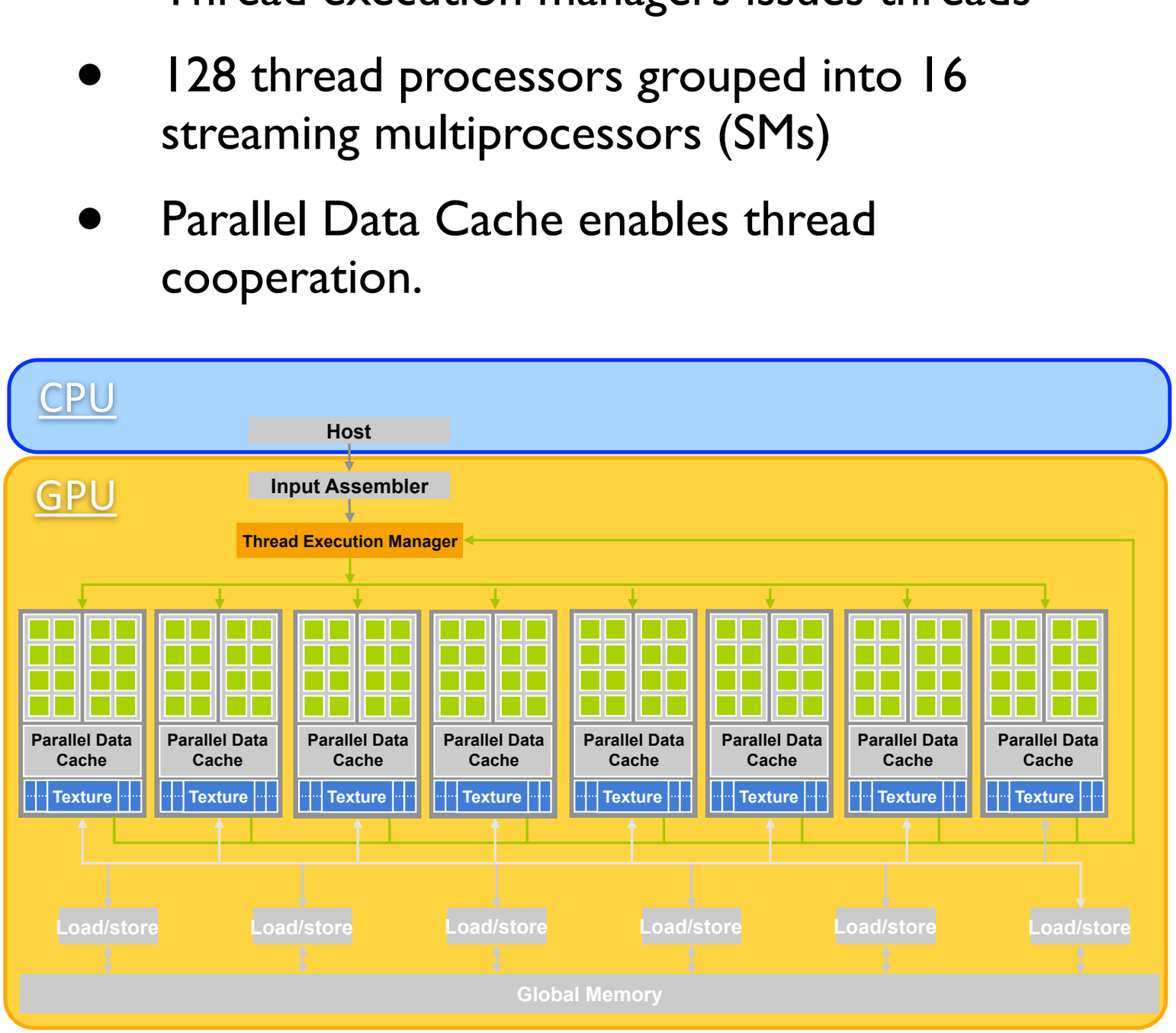


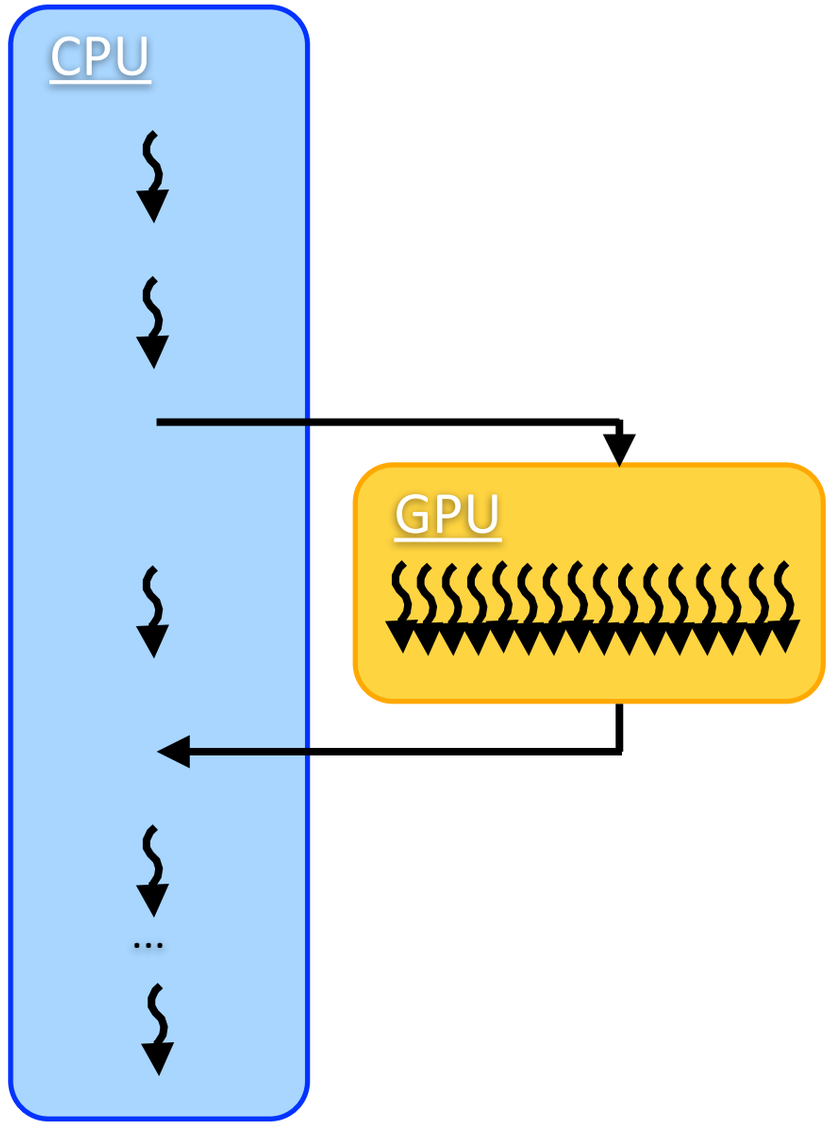


	Dual socket, AMD 2.3 GHz 12-core	NVIDIA Tesla S2050
Peak DP FLOPs	220.8 GFLOPs DP	2060.8 GFLOPs DP (9.3x)
Peak SP FLOPS	441.6 GFLOPs SP	4121.6 GFLOPs SP (9.3x)
Peak RAM BW	25 GB/sec	576 GB/sec (23x)
Peak PCIe BW	N/A	16 GB/sec
Needs x86 server to attach to?	No	Yes
Power/Heat	~450 W	~900 W + ~400 W (~2.9x)
Code portable?	Yes	No (CUDA) Yes (PGI, OpenCL)

CUDA-capable GPU Hardware Architecture

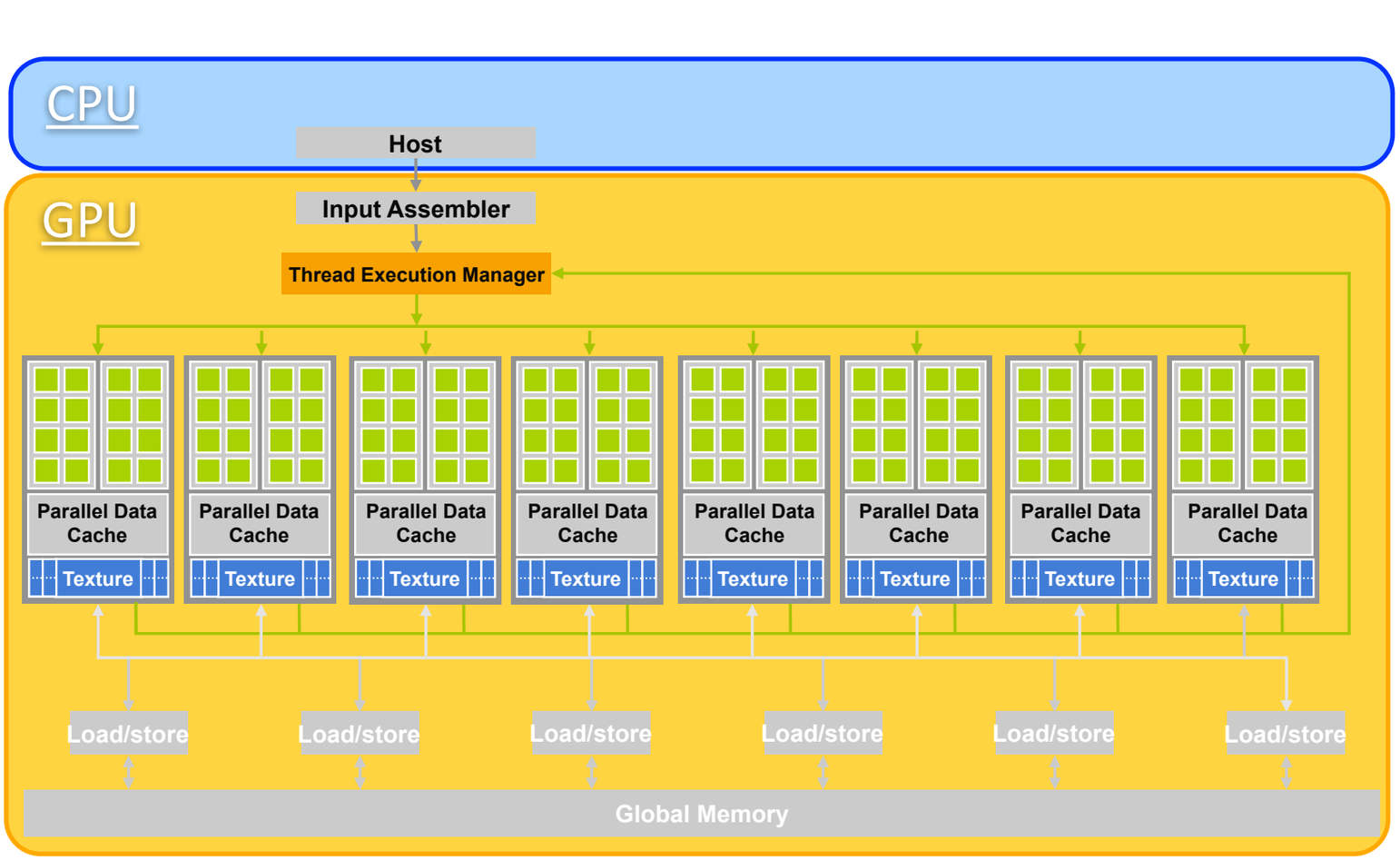
- Processors execute computing threads
- Thread execution managers issues threads
- 128 thread processors grouped into 16 streaming multiprocessors (SMs)
- Parallel Data Cache enables thread cooperation.

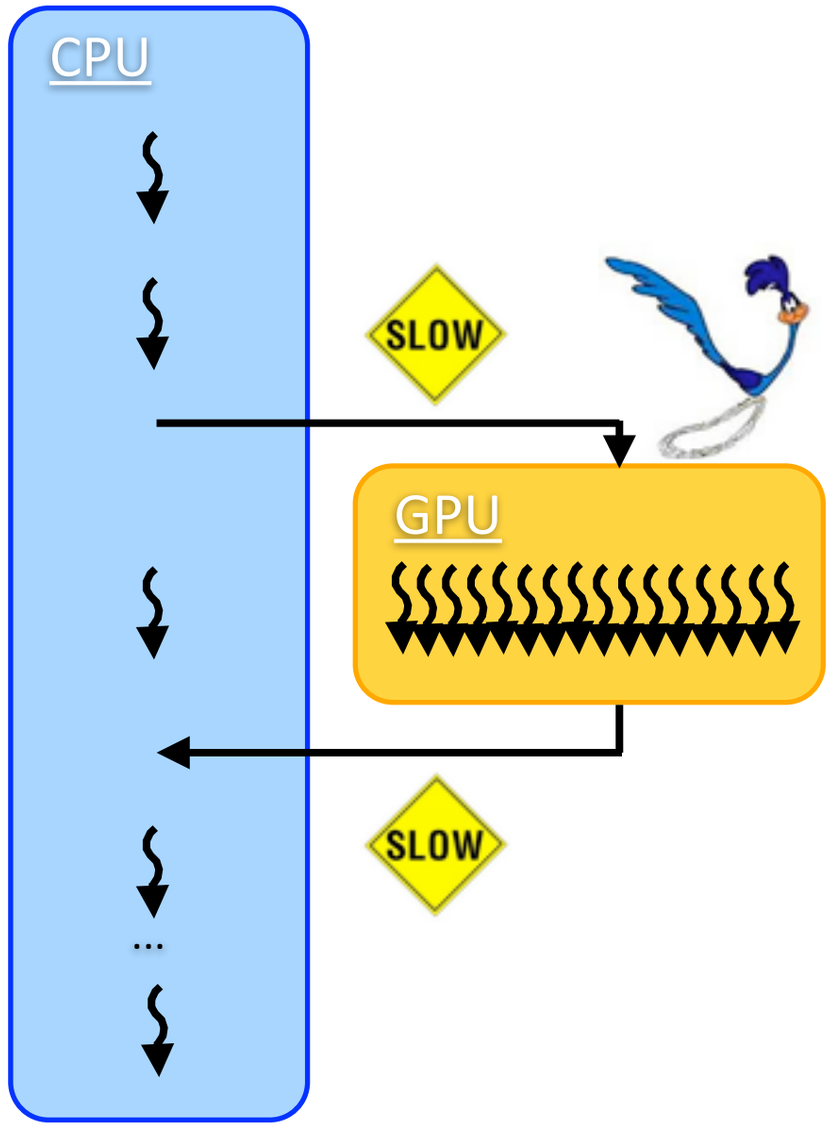




CUDA-capable GPU Hardware Architecture

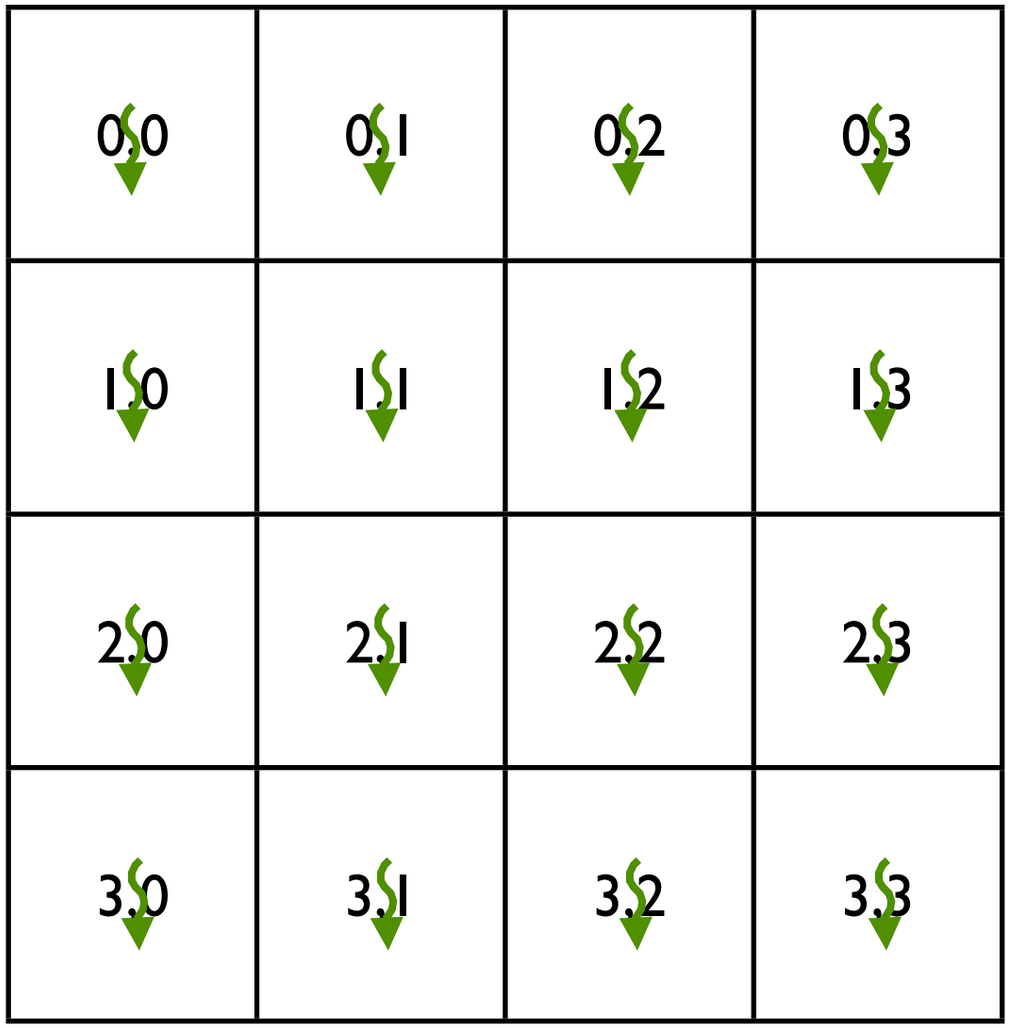
- Processors execute computing threads
- Thread execution managers issues threads
- 128 thread processors grouped into 16 streaming multiprocessors (SMs)
- Parallel Data Cache enables thread cooperation.





Single Instruction, Multiple Threads (SIMT)

- A version of SIMD used in GPUs.
- GPUs use a thread model to achieve a high parallel performance and hide memory latency.
- On a GPU, 10,000s of threads are mapped on to available processors that all execute the same set of instructions (on different data addresses).



Is it hard to program on a GPU?

- In the olden days – (pre-2006) – programming GPUs meant either:
  - using a graphics standard like OpenGL (which is mostly meant for rendering), or
  - getting fairly deep into the graphics rendering pipeline.
- To use a GPU to do general purpose number crunching, you had to make your number crunching pretend to be graphics.
- This is hard. Why bother?

How to Program on a GPU Today

- Proprietary programming language or extensions
  - NVIDIA: CUDA (C/C++)
  - AMD/ATI: StreamSDK/Brook+ (C/C++)
- OpenCL (Open Computing Language): an industry standard for doing number crunching on GPUs.
- Portland Group Inc (PGI) Fortran and C compilers with accelerator directives; PGI CUDA Fortran (Fortran 90 equivalent of NVIDIA's CUDA C).
- OpenMP version 4.0 may include directives for accelerators.