



Information Coding / Computer Graphics, ISY, LiTH

Timeline for CPUs

80's: CPU and system same speed. Zero wait states.

1993: CPUs faster than the rest of the system. Rapid raise of frequency.

Late 90's to present: Multi-CPU systems, multi-core CPUs.

CPUs are still improving, but going for higher frequency is not as obvious as before.



Information Coding / Computer Graphics, ISY, LiTH

Meanwhile, at the graphics dept

80's: Hardware sprites. Push pixels with low-level code.

1993: Textures 3D games: Wolf3D, Doom.

Early 90's: Professional 3D boards.

1996: 3dfx Voodoo1!

2001: Programmable shaders.

2006: G80, unified architecture. CUDA

2009: OpenCL.

2010: Fermi architecture

2012: Kepler architecture



Information Coding / Computer Graphics, ISY, LiTH

	1995	2005	
CPU Frequency (GHz)	.1	3.2	32x
Memory Frequency (GHz)	.03	1.2	40x
Bus Bandwidth (GB/sec)	.1	4	40x
Hard Disk Size (GB)	.5	200	400x



Information Coding / Computer Graphics, ISY, LiTH

	1995	2005	
CPU Frequency (GHz)	.1	3.2	32x
Memory Frequency (GHz)	.03	1.2	40x
Bus Bandwidth (GB/sec)	.1	4	40x
Hard Disk Size (GB)	.5	200	400x
Pixel Fill Rate (GPixels/sec)	.0004	3.3	8250x
Vertex Rate (GVerts/sec)	.0005	.35	700x
Graphics flops (GFlops/sec)	.001	40	40000x
Graphics Bandwidth (GB/sec)	.3	19	63x
Frame Buffer Size (MB)	2	256	128x



How about 2005-2011?

	2005	2011	
CPU Frequency (GHz)	3.2	3.8	1.18x
Memory Frequency (GHz)	1.2	2.0	1.67x
Bus Bandwidth (GB/sec)	4	31	7.75x
Hard Disk Size (GB)	200	4000	20x

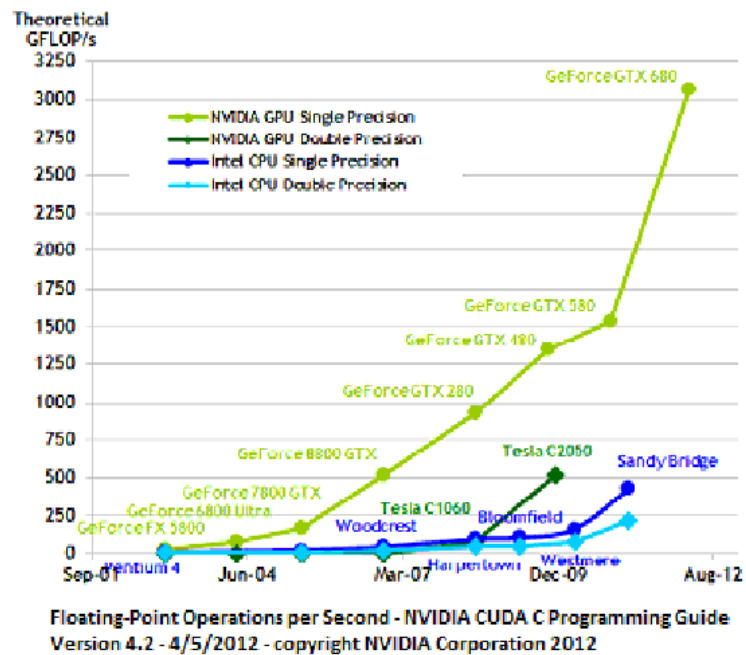


	2005	2011	
CPU Frequency (GHz)	3.2	3.8	1.18x <small>x cores?</small>
Memory Frequency (GHz)	1.2	2.0	1.67x
Bus Bandwidth (GB/sec)	4	31	7.75x
Hard Disk Size (GB)	200	4000	20x
Pixel Fill Rate (GPixels/sec)	3.3	59	18x
Vertex Rate (GVerts/sec)	.35	?	?
Graphics flops (GFlops/sec)	40	2488	62x
Graphics Bandwidth (GB/sec)	19	327.7	17x
Frame Buffer Size (MB)	256	3000	12x



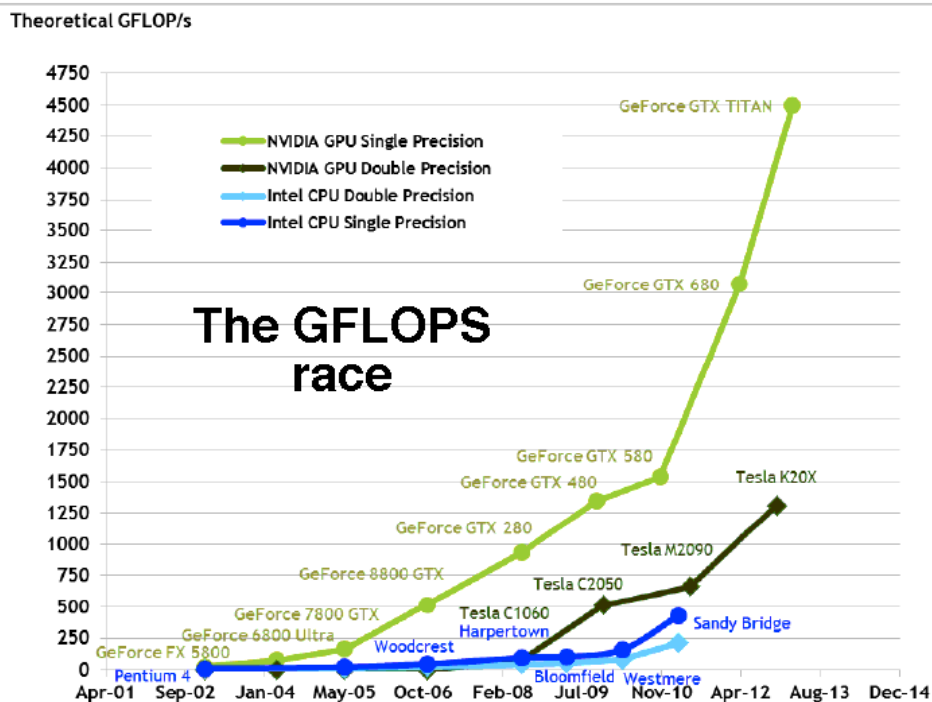
Information Coding / Computer Graphics, ISY, LiTH

The GFLOPS race



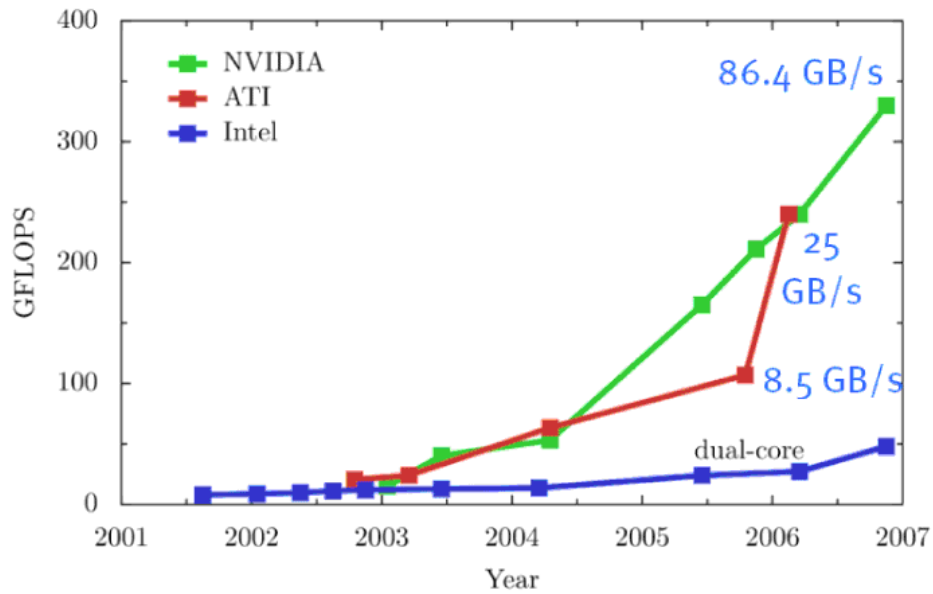
Information Coding / Computer Graphics, ISY, LiTH

The GFLOPS race

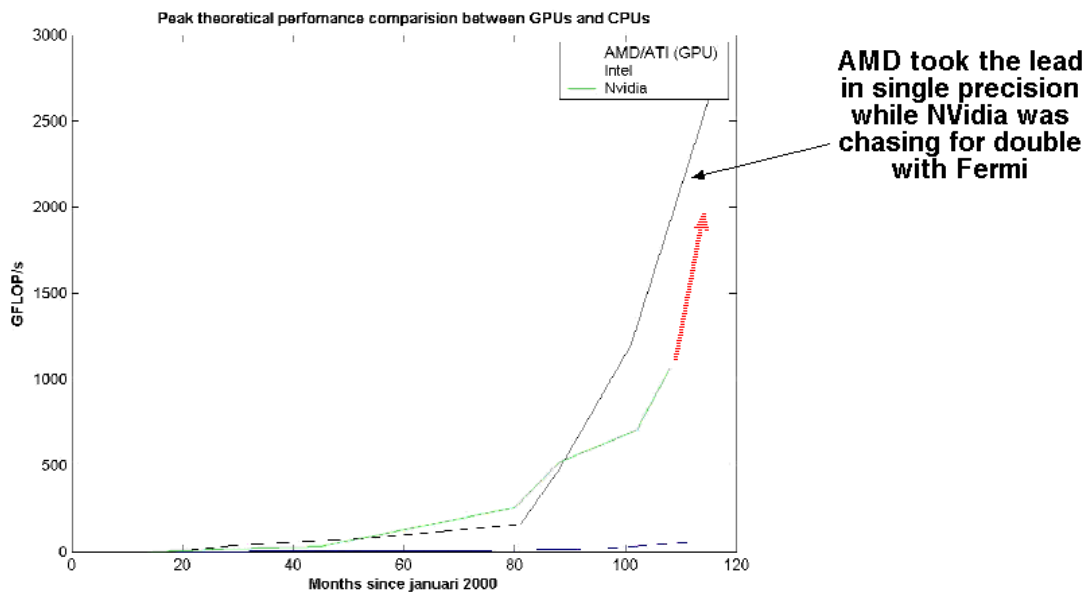




Another graph, including ATI/AMD

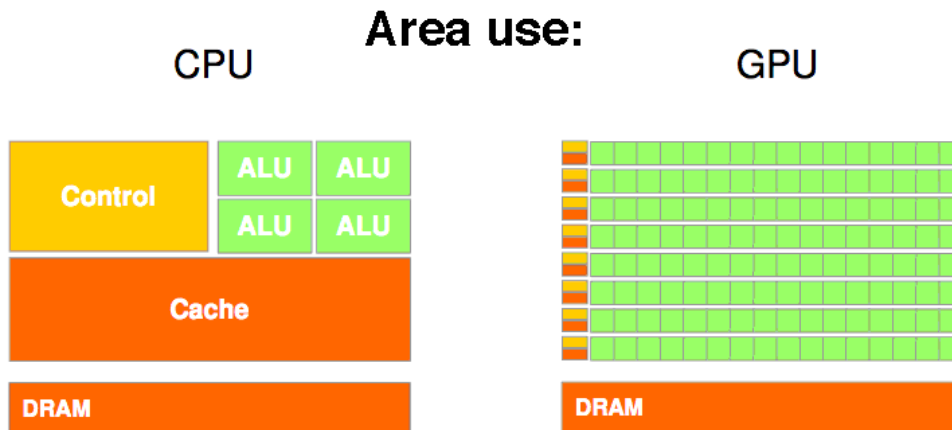


Another graph, including ATI/AMD





How is this possible?



But in particular: SIMD architecture



Flynn's taxonomy

SISD Single instruction, single data Old single-core	MISD Multiple instructions, single data Multiple computations for redundancy
SIMD Single instruction, multiple data GPUs, vector processors	MIMD Multiple instructions, multiple data. Multi-core CPUs

SIMT = single instruction, multiple thread



Information Coding / Computer Graphics, ISY, LiTH

Important implications of SIMD/SIMT

SIMD: "if" statements - all or none

**SIMT: Thread focused. Fast thread switching.
Hides memory latency!**



Information Coding / Computer Graphics, ISY, LiTH

Why did GPUs get so much performance?

**Early problem with large amounts of data. (Complex geometry,
millions of output pixels.)**

Graphics pipeline designed for parallelism!

Hiding memory latency by parallelism

**Volume. 3D graphics boards central component in game
industry. Everybody wants one!**

**New games need new impressive features. Many important
advancements started as game features.**



Must process many pixels fast!

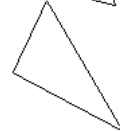


Early GPUs could draw textured, shaded triangles much faster than the CPU.

Must do matrix multiplication and divisions fast.



Next generation could transform vertices and normalize vectors.



Must have programmable parts.

This was added to make Phong shading and bump mapping.

Must work in floating-point!

This was for light effects, HDR.



So a GPU should

- process vertices, many in parallel, applying the same transformations on each
- process pixels (fragments) in parallel, applying the same color/light/texture calculations on each

SIMD friendly problem!

Less control, control many calculations instead of one



A different kind of threads

SIMD threads, all run the same program

Group-wise, they execute in parallel, SIMD-style

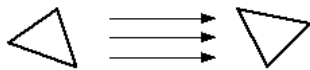
Shader threads calculate one pixel or one vertex

CUDA/OpenCL threads may calculate anything, but typically one part of the output - in order



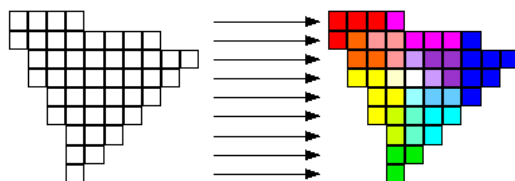
The main tasks in rendering graphics:

One thread per vertex
Same operations, same kernel, different data



CUDA and OpenCL generalize this to any kind of data, and possibility to access any part of memory.

One thread per pixel (fragment)
Same operations, same kernel, different data





The 3D pipeline in the GPU

Low-level operations from vertices to pixel data

