PDC Summer School 2016

# Short Introduction to GPU programming for Scientific Computing

2015-08-19

Michael Schliephake

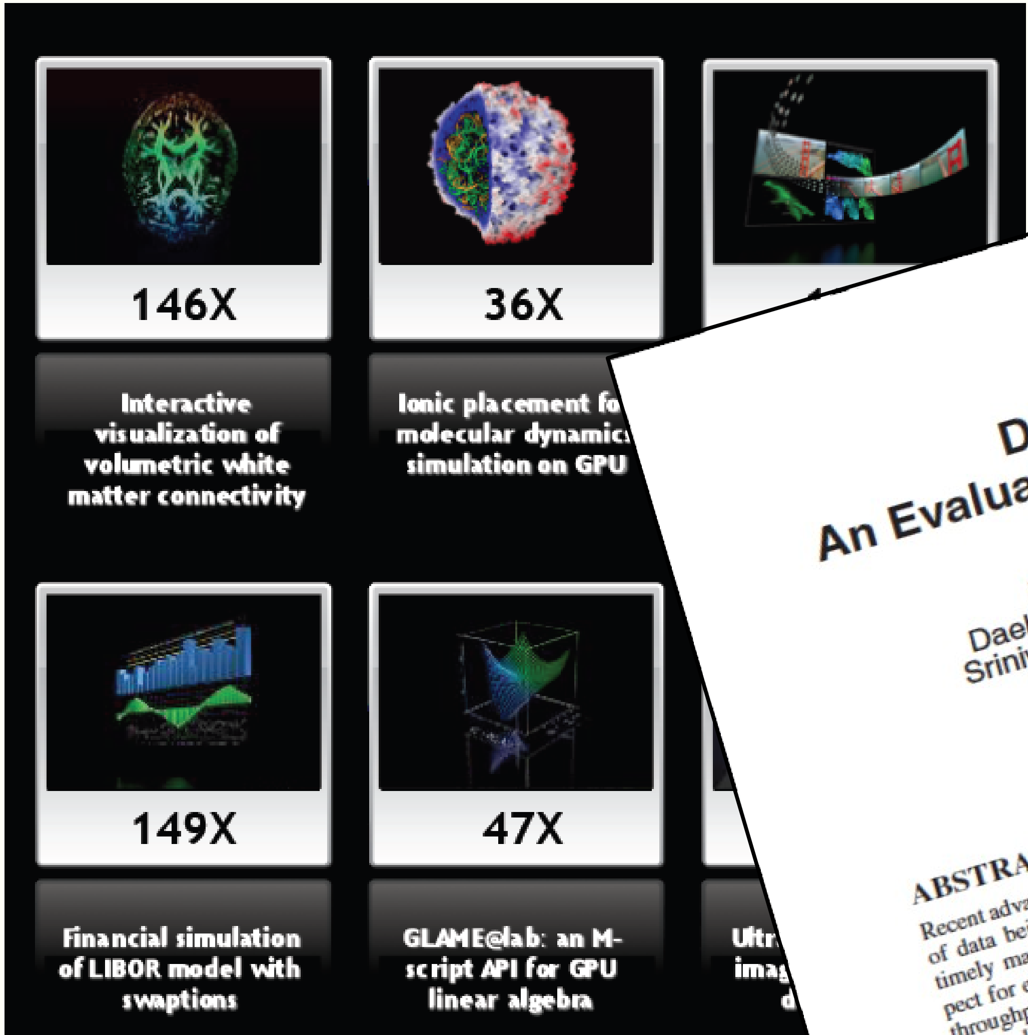
Szilárd Páll

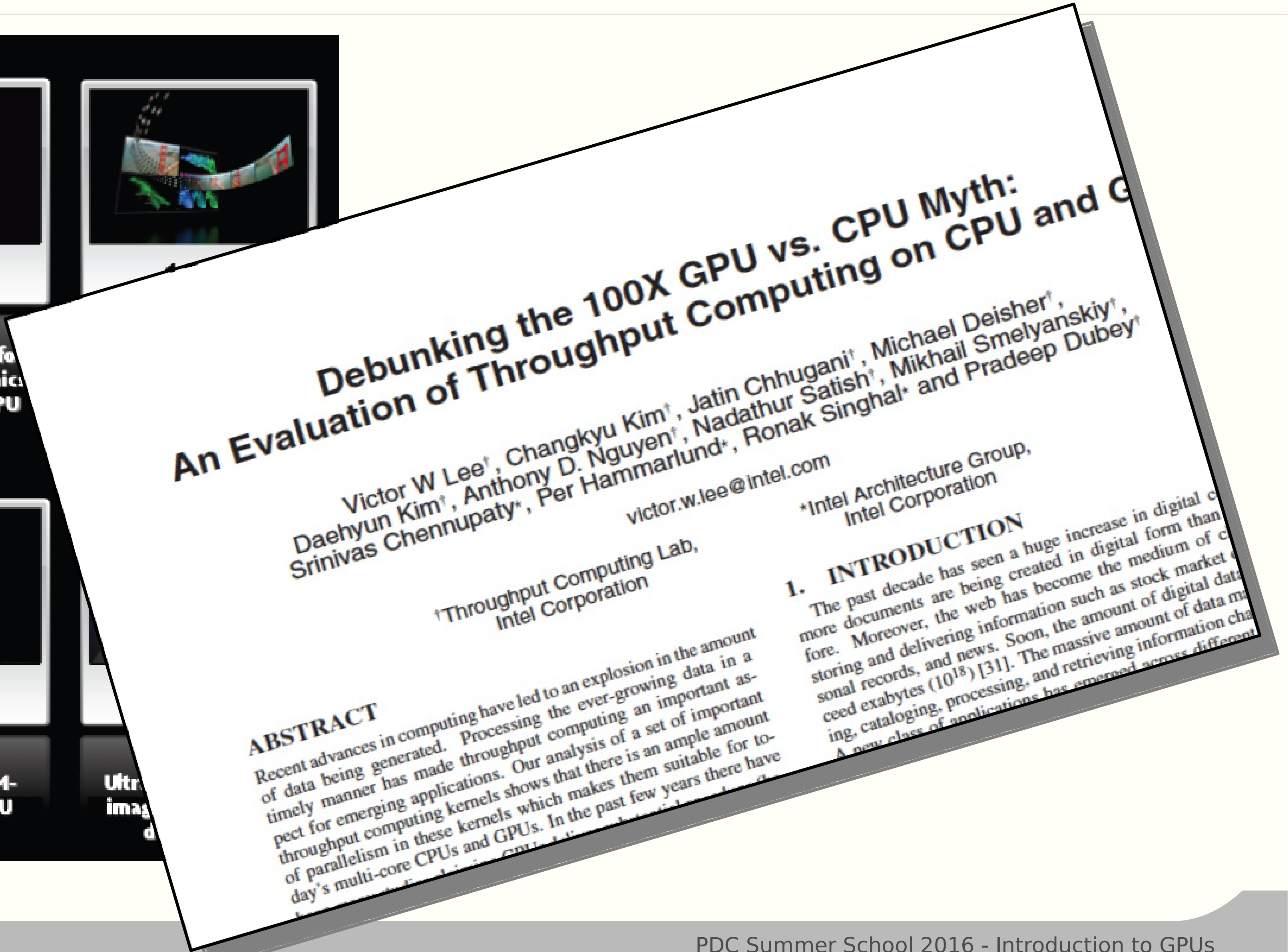
KTH – CSC – HPCViz

# Overview

1. GPU processor characteristics
2. Practical usage scenarios

# The Debate GPU vs. CPU



**Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and G**

Victor W Lee†, Changkyu Kim†, Jatin Chhugani†, Michael Deisher†,
Daehyun Kim†, Anthony D. Nguyen†, Nadathur Satish†, Mikhail Smelyanskiy†,
Srinivas Chennupaty*, Per Hammarlund*, Ronak Singhal* and Pradeep Dubey†

victor.w.lee@intel.com

†Throughput Computing Lab,
Intel Corporation

*Intel Architecture Group,
Intel Corporation

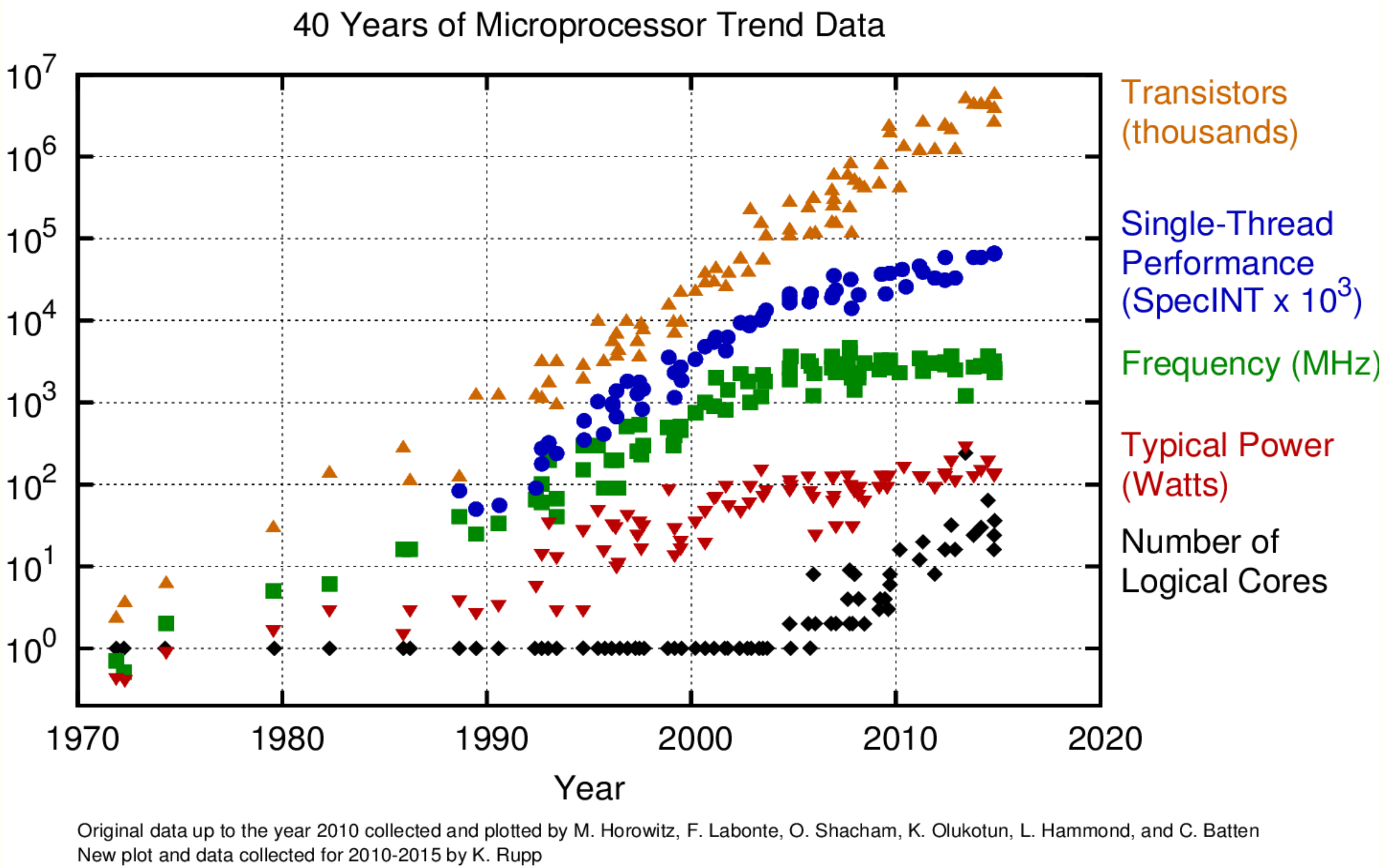
**ABSTRACT**

Recent advances in computing have led to an explosion in the amount of data being generated. Processing the ever-growing data in a timely manner has made throughput computing an important aspect for emerging applications. Our analysis of a set of important throughput computing kernels shows that there is an ample amount of parallelism in these kernels which makes them suitable for today's multi-core CPUs and GPUs. In the past few years there have

**1. INTRODUCTION**

The past decade has seen a huge increase in digital c
more documents are being created in digital form than
fore. Moreover, the web has become the medium of c
storing and delivering information such as stock market c
sonal records, and news. Soon, the amount of digital data
ceed exabytes ($10^{18}$) [31]. The massive amount of data ma
ing, cataloging, processing, and retrieving information cha

# Processor trends



# Modern CPUs

- Complex cores optimized for serial code
  - **Latency optimized**
  - Superscalar, out-of-order speculative execution with branch prediction logic and complicated cache hierarchies and vector units
- Single core performance is leveling off
  - Multicore CPUs everywhere
  - Vector units are getting larger
- Only a small part of the circuitry is doing the actual computation!

# Classical Supercomputers

- Large amount of computing nodes
  - Distributed memory
  - Multicore processors (tens of cores per node)
- Fast network (interconnect)
  - Proprietary or Infiniband (or even Gig-Ethernet)
- Programming model
  - Processes communicate via Message Passing (MPI)
  - Multi-threading inside a node

# Power Wall

- Performance of top supercomputers is limited by power consumption
  - K Computer, 10.51 Pflops, 12.66 MW (current #5)
    - Enough for a small town with ~6000 houses
  - Sunway TaihuLight: 93 Pflops, 15 MW (current #1)
- More efficient computing is needed

- Current trends
  - manycore + wide vector units
  - accelerators

# Top500 June 2016

New trends starting to dominate: manycore, accelerators

Traditional machines: multicore

| Rank | Site | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|---|
| 1 | National Supercomputing Center in Wuxi<br>China | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway<br>NRCPC | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 2 | National Super Computer Center in Guangzhou<br>China | **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P<br>NUDT | 3,120,000 | 33,862.7 | 54,902.4 | 17,808 |
| 3 | DOE/SC/Oak Ridge National Laboratory<br>United States | **Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x<br>Cray Inc. | 560,640 | 17,590.0 | 27,112.5 | 8,209 |
| 4 | DOE/NNSA/LLNL<br>United States | **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom<br>IBM | 1,572,864 | 17,173.2 | 20,132.7 | 7,890 |
| 5 | RIKEN Advanced Institute for Computational Science (AICS)<br>Japan | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect<br>Fujitsu | 705,024 | 10,510.0 | 11,280.4 | 12,660 |

# Accelerators

- Basic idea: move computationally intensive tasks to a specialized unit that is
  - Faster & more efficient for the selected tasks
  - Specialized chips → not suitable or slow for general purpose work
- Coprocessors not a new invention:
  - Intel 8087 or 80387 (floating point), DSPs in sounds cards (e.g. Sound Blaster AWE)
  - Common in consumer electronics: A/V processing

# Accelerators challenges

- Designing specialized hardware is very expensive
  - Somebody has to pay for it!

    → GPUs: gamers
- Programmability challenges
  - General vs specialized
  - Programming model

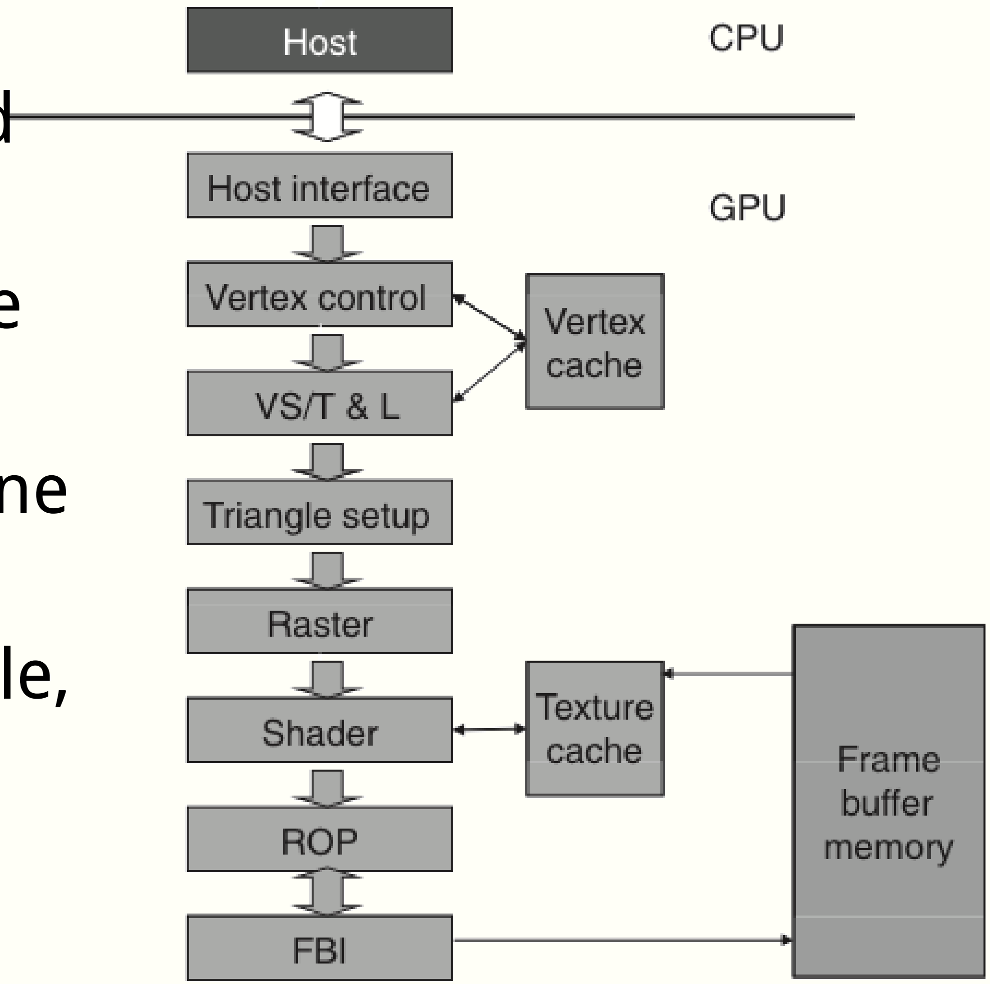
# General Purpose GPU

- GPU is Graphics Processing Unit that you can find inside a display adapter.
  - Optimized for computations needed in graphics rendering
- GPUs have evolved rapidly
- Shader units became programmable
  - Opened the door for GPGPU:

    **G**eneral-**P**urpose computing on **G**raphics **P**rocessing **U**nits

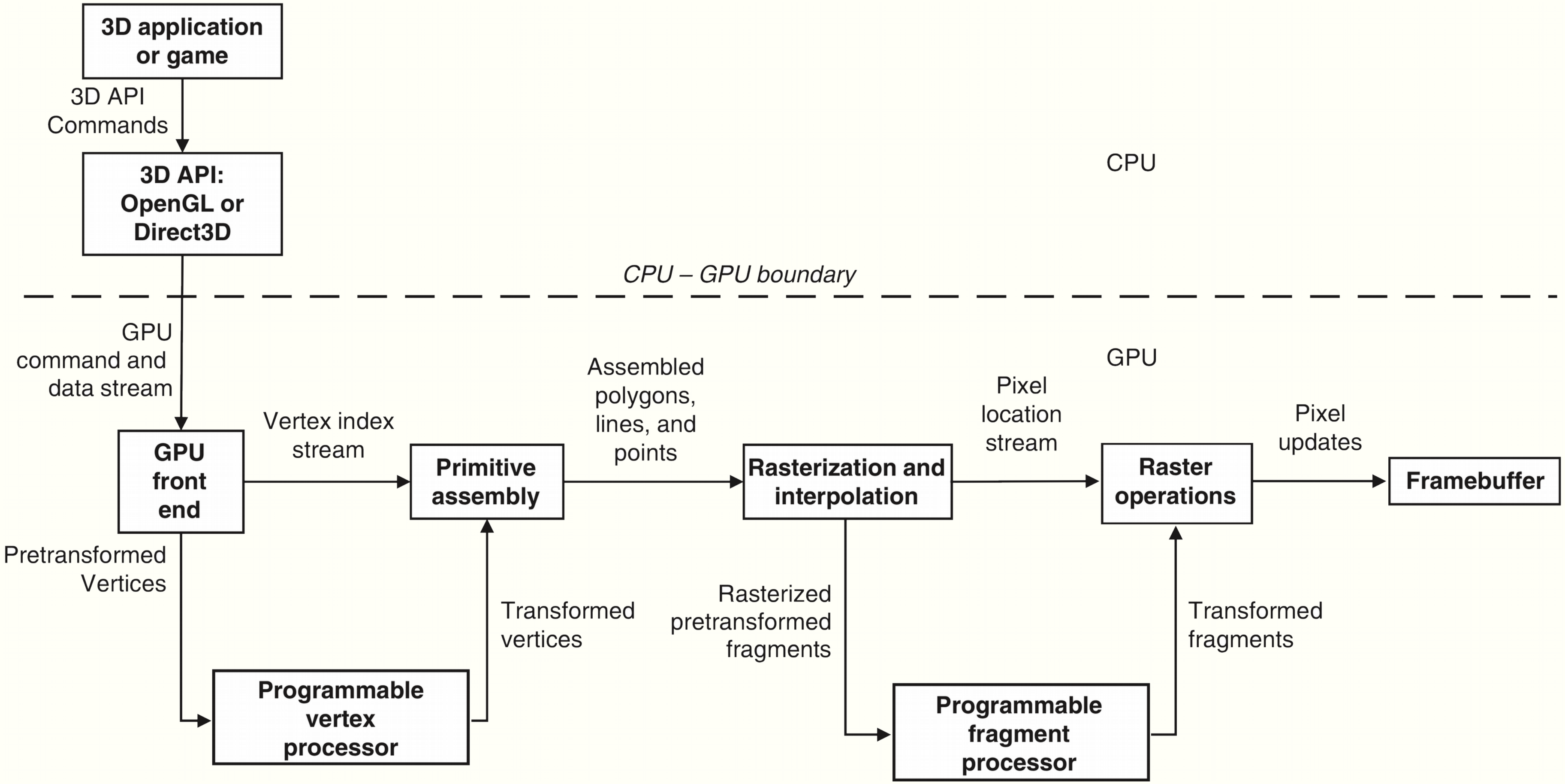
# History
# Programmable Graphics Cards

- First graphics pipelines with fixed functionality
- Transformation to programmable device increased functionality
- Many pixel operations can be done parallel
- Other computations were possible, but cumbersome

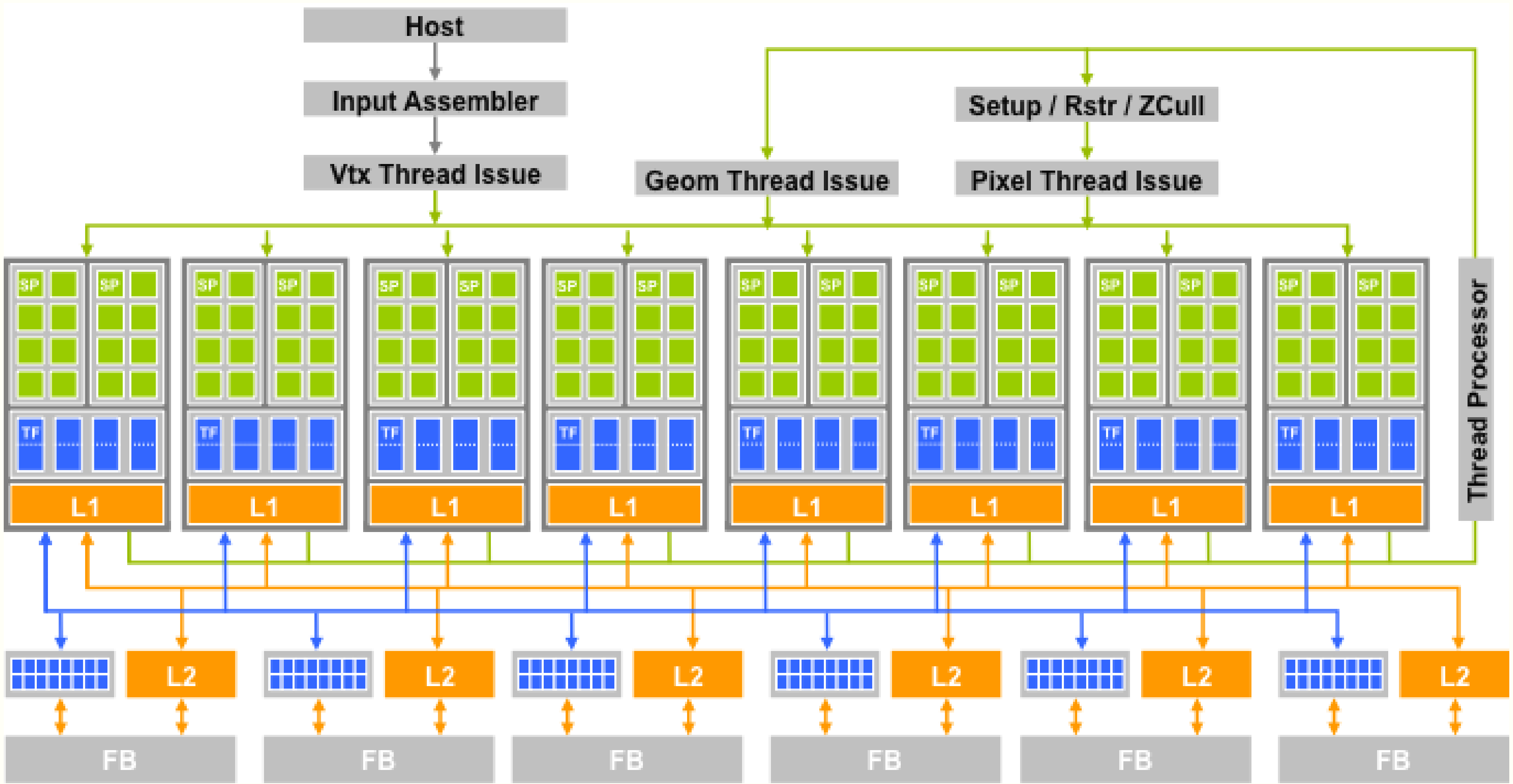



Images: Courtesy David Kirk/NVIDIA and Wen-mei W. Hwu

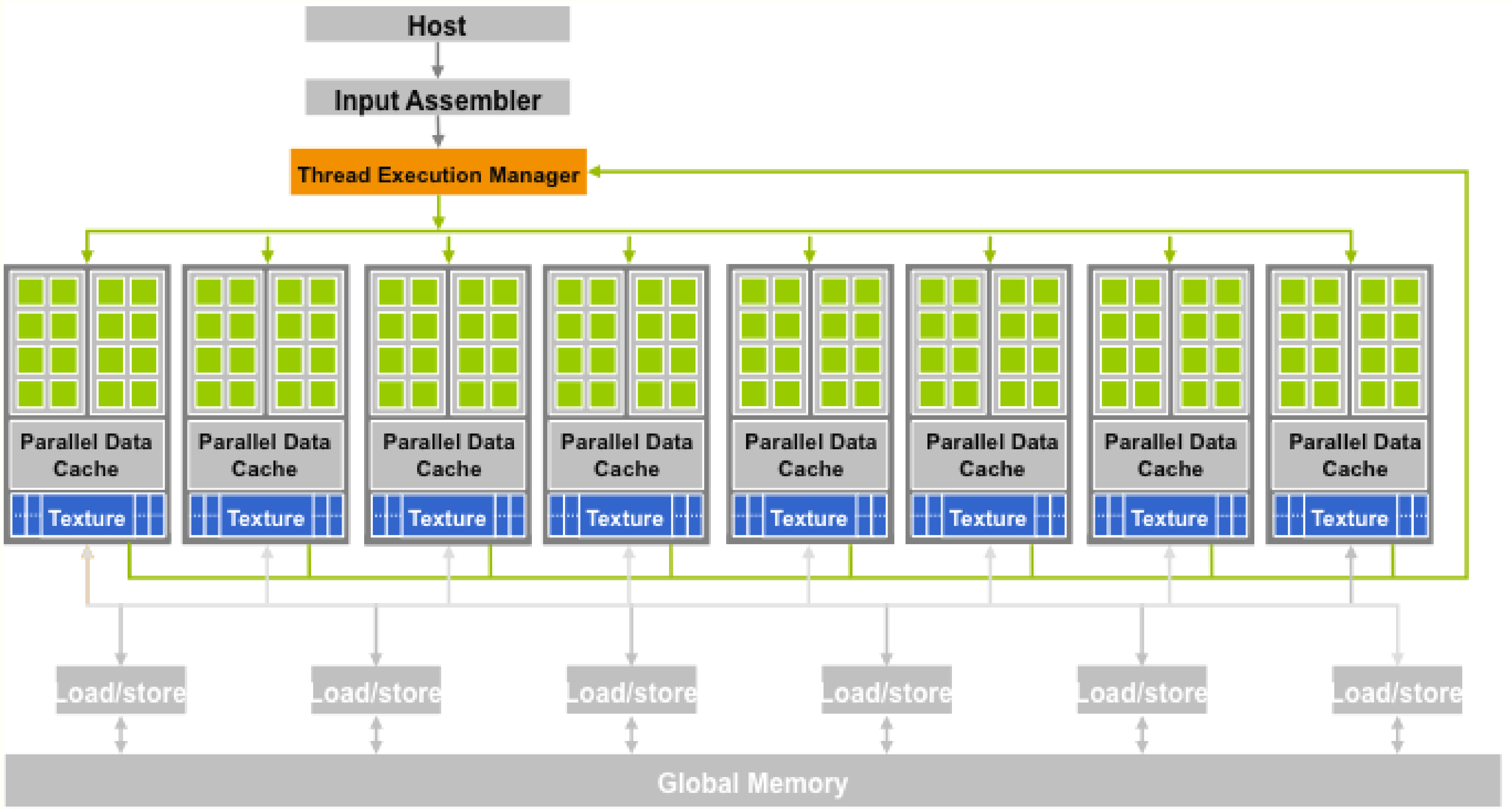
# History
# Programmable Graphics Cards



# History
# G80 in Graphics Mode



# History
# G80 in CUDA Mode

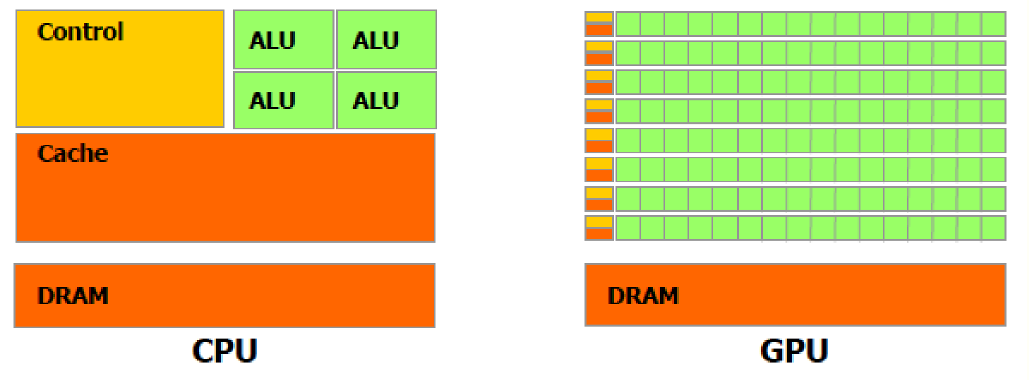


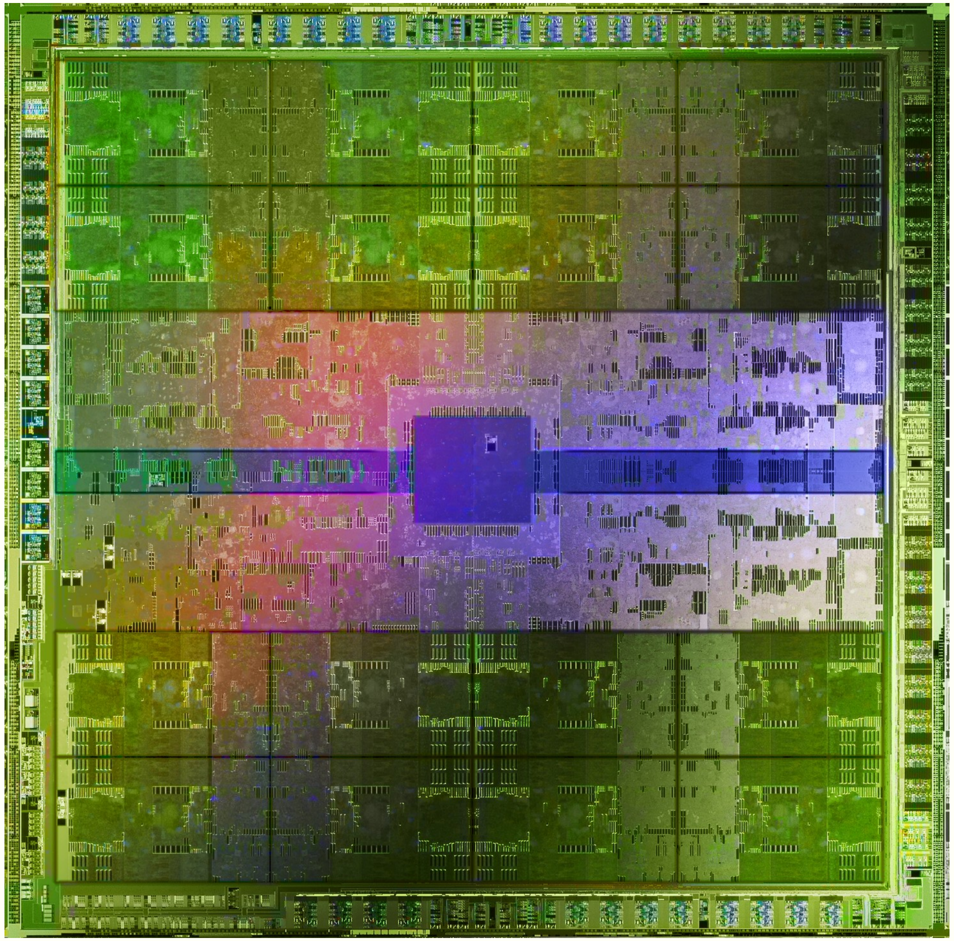
# GPU processor characteristics

- More transistors used for compute
- Many, simple cores
- Smaller caches
- Many memory controllers for high bandwidth

→ **massively parallel**

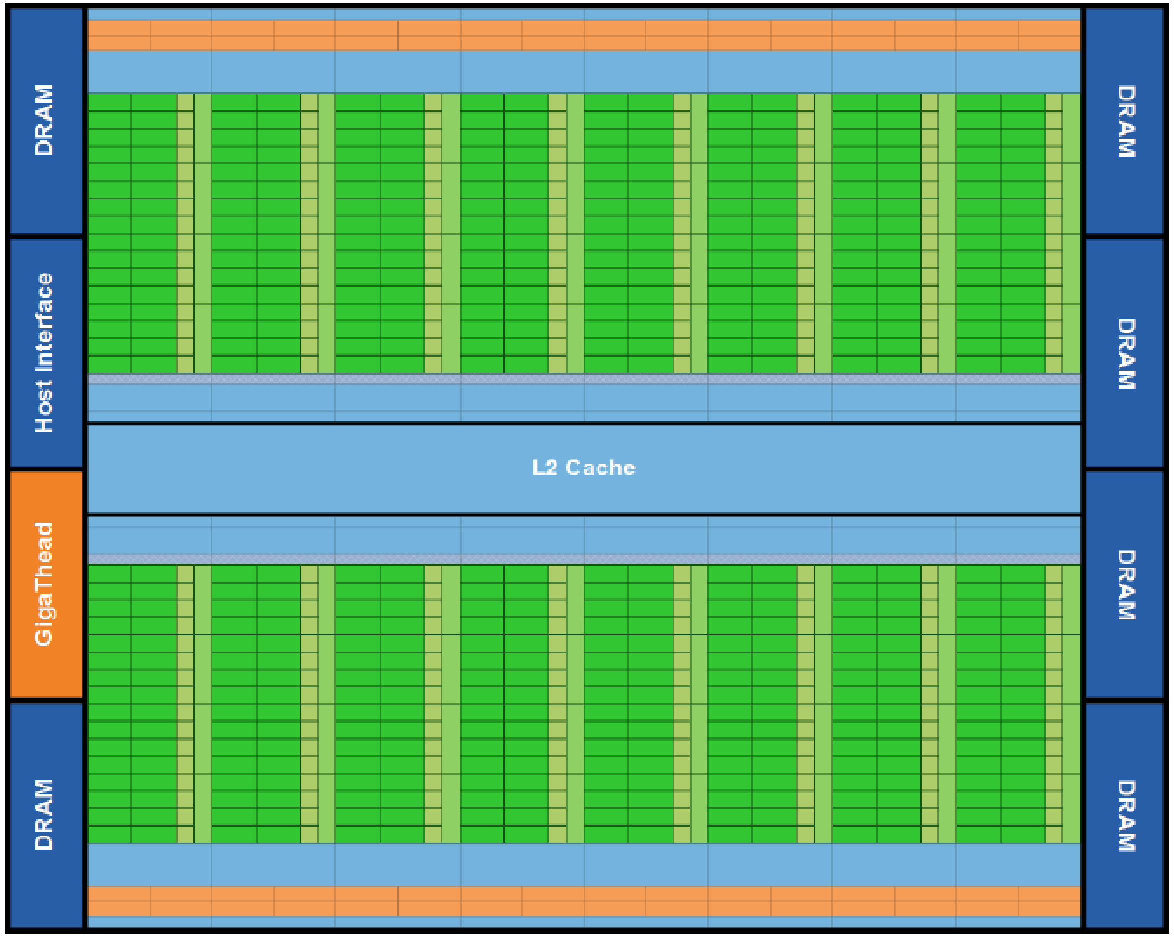
→ **throughput** optimized arch

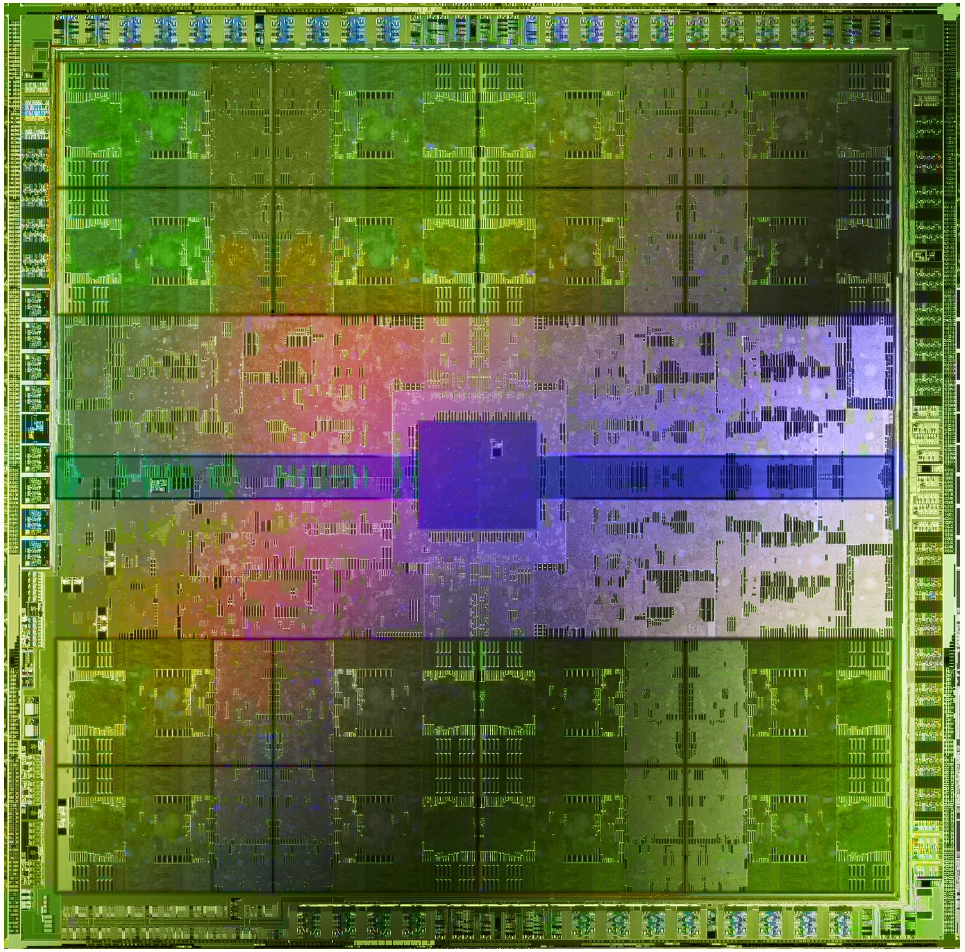


Images: Courtesy of NVIDIA and David Kirk/NVIDIA and Wen-mei W. Hwu

# Fermi Processor



# Fermi to Pascal through Kepler and Maxwell

- More transistors:

  3 → 7 → 8 → 15 billion
- On-chip parallelism:

  512 → 3584 thread proc. (CUDA ”cores”)
- Floating point throughput:

  0.7/1.3 → 1.7/5 → 0.2/6.8 → ~5/10 Gflops SP/DP
- Bandwidth:

  178 → 288 → 288 → 720 Gb/s
- More/faster ”close” memory (registers, caches)
- Better efficiency, flexibility, programmability

# Fermi to Pascal through Kepler and Maxwell

- More transistors:

  3 → 7 → 8 → 15 billion

- On-chip parallelism:

  512 → 3584 thread proc. (CUDA ”cores”)

- Floating point throughput:

  0.7/1.3 → 1.7/5 → 0.2/6.8 → ~5/10 Gflops SP/DP

- Bandwidth:

  178 → 288 → 288 → 720 Gb/s

- More/faster ”close” memory (registers, caches)
- Better efficiency, flexibility, programmability

**Still (just) a GPU/accelerator!**

# Fermi to Pascal through Kepler and Maxwell

- More transistors:

    3 → 7 → 8 → 15 billion

- On-chip parallelism:

    512 → 3584 thread proc. (CUDA ”cores”)

- Floating point throughput:

    0.7/1.3 → 1.7/5 → 0.2/6.8 → ~5/10 Gflops SP/DP

- Bandwidth:

    178 → 288 → 288 → 720 GB/s

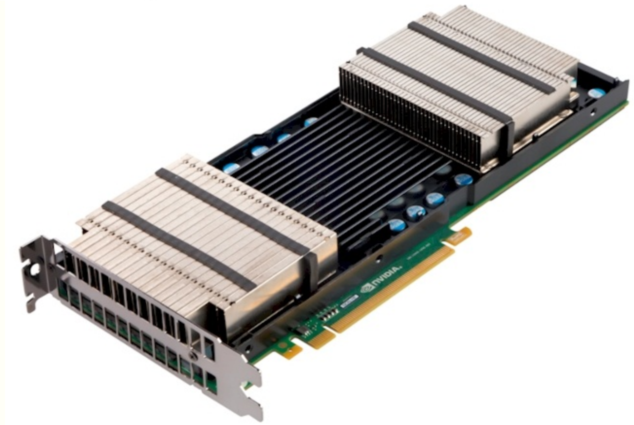
- More/faster ”close” memory (registers, caches)
- Better efficiency, flexibility, programmability

**Still (just) a GPU/accelerator!**

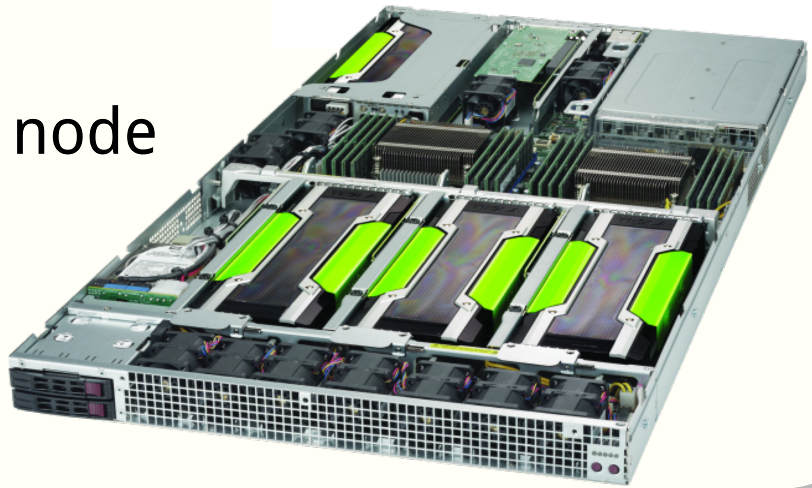
**But a fast one!***

# NVIDIA Tesla K80


- Architecture: GK210
- ALUs: 2x 2496 (2x 13 SM)
- Memory interface: 384 bit
- Memory bandwidth: 288 GB/s
- Peak Performance: 2.91/874 Tflops DP/SP
- RAM: 2x 12 GB



- Tegner @PDC: 9 nodes, 2 CPUs + 1 GPU/ node



# Other GPU Accelerators

- AMD
  - GPUs: consumer (Radeon) & professional (Firepro)
    - First HBM memory
    - up to 14 Tflops SP
  - APUs (integrated CPU+GPU)
  - Programmed with OpenCL, HCC, C++ AMP
- Intel iGPU
  - up to 1.15 Tflops (Skylake)
  - eDRAM

# Other Accelerators non GPU

- Intel Xeon Phi
  - Up to 72 x86 cores, wide vector units
  - ≥3 Tflop/s double, ≥6 Tflop/s single precision
  - MPI + threading/OpenMP programming

- FPGA
- DSP
- Custom ASICs

# Technical aspects of using GPUs

- Data parallel computation with
  - limited need of synchronization
  - limited need of operating system services
  - high arithmetic intensity
  - Predictable memory access patterns
- Problems
  - Amdahl's Law
  - Data transfer
- Approach when there is no ”ideal programming problem”: heterogeneous applications

# How to use GPUs

1. Use existing GPU software
2. Use numerical libraries with GPU support
3. Programming using directives
4. Native GPU code

# Using existing GPU software

- HOOMD, NAMD, GROMACS, GPU-HMMER, GPU-BLAST, LAMMPS, Matlab (Toolbox), ...
- Pros
  - No implementation headaches for end users
- Cons
  - What if my science area/application is not supported?
  - Often include only limited set of functionality
  - GPU versions can be in early development phase, tricky to use efficiently

# Use Libraries with GPU Support

- cuBLAS, cuFFT, cuSPARSE, clBLAS, ViennaCL, MAGMA, PetsC, OpenCV, Torch, etc.*
- Pros
  - Easy to implement in your programs
  - Algorithms in libraries usually efficient
- Cons
  - Speedup limited by Amdahl's law and there is still transfer bottleneck

* CUDA libraries: https://developer.nvidia.com/gpu-accelerated-libraries
OpenCL libraries: http://www.iwocl.org/resources/opencl-libraries-and-toolkits

# Directive-based GPU programming

- OpenACC
  - Backed by Portland Group, CAPS, Cray and NVIDIA
  - Compilers: PGI, Cray, CAPS HMPP (GCC partial support)
- OpenMP 4.0+: general offload directives
  - More than just loop offload
  - GCC, clang/llvm

# Directive Based GPU Code

- OpenMP, OpenACC
- Pros
  - Same code base as CPU version
  - Short time to solution
  - Portability is better due to different backends
- Cons
  - Generated code may not be as fast as hand-tuned CUDA

# Native GPU code

- CUDA, OpenCL
- Pros
  - Good control and best performance
- Cons
  - Requires most time
  - Portability (including performance)

# Native GPU code in a non C/C++ project

- Alternatives:
  - Cuda Fortran (PGI)
  - PyCUDA / PyOpenCL
  - CUDA Python
  - Alea GPU: .NET
  - Julia, R,...

# CUDA is Nvidia specific

- OpenCL is standard
  - Support for a wide range of devices
    - GPUs (from mobile to server), FPGAs, CPUs,...
  - Performance portability is difficult (impossible by some measure)
- On NVIDIA hardware
  - CUDA is generally faster
    - Remains NVIDIA's main programming interface
    - Will evolve faster than OpenCL

# Summary

- Power wall → current trends in computing: many-core & accelerators
- Throughput vs latency architectures
- Accelerators are evolving fast but their use is still challenging
- Programming GPUs
  - CUDA, OpenCL
  - Directive-based approaches