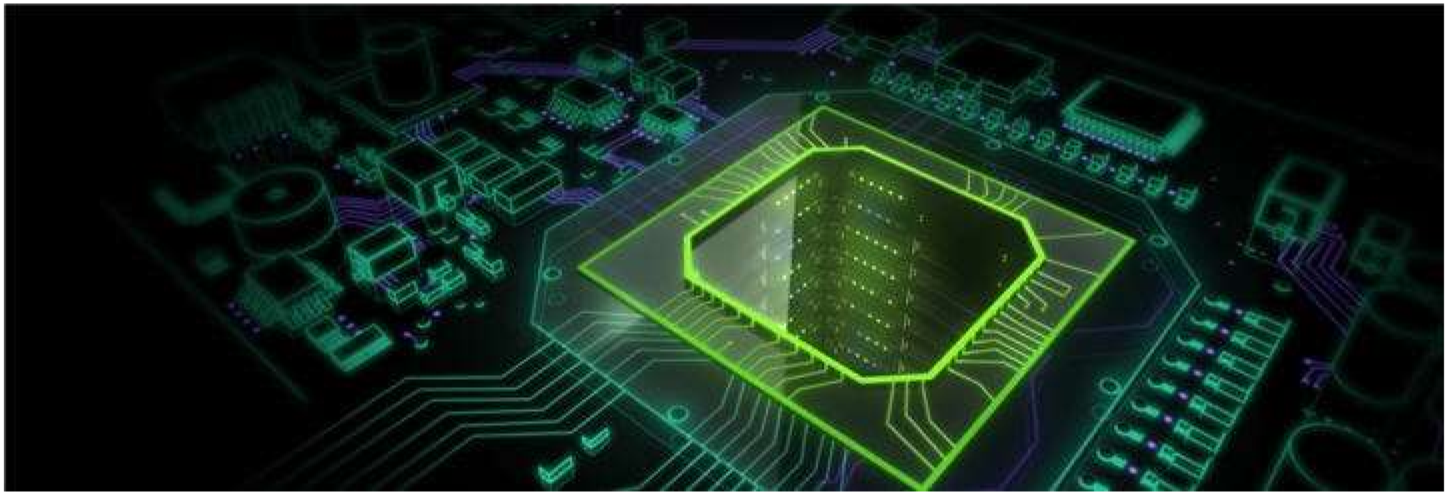
CSCI-GA.3033-004

Graphics Processing Units (GPUs): Architecture and Programming

Lecture 1: Gentle Introduction to GPUs

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Who Am I?

- Mohamed Zahran (aka Z)
- Research interest: Computer architecture/OS/Compilers Interaction
- http://www.mzahran.com
- Office hours: Tu-Th 2:00-3:00 pm
  - Or by appointment
- Office: WWH 320

What we will learn in this course

- Why GPUs
- GPU Architecture
- GPU-CPU Interaction
- GPU programming model
- When do GPUs excel? When not?
- Solving real-life problems using GPUs
  - With the best performance we can get!

But I also want you to:

- Get more than an good grade
- Use what you have learned in *MANY* different contexts
  - GPUs can be used in many non-graphics applications
- Have a feeling for how hardware and software evolve and interact
- To enjoy the course!

The Course Web Page

- Lecture slides
- Reading assignments
- Info about labs, homework assignments, project, and exams.
- Useful links (manuals, tools, book errata, …)
- A link to NYU classes where:
  - you submit assignments
  - you get your grades
  - you ask/answer questions on forums
  - you get announcements from the instructor

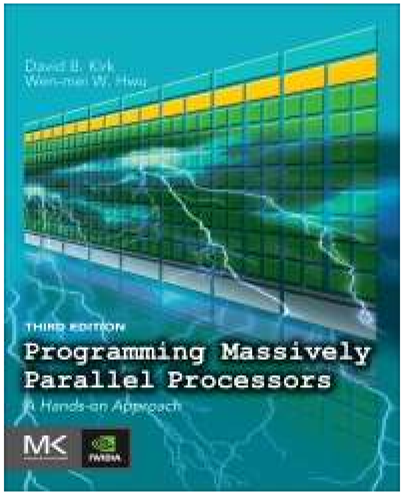
The Textbook

Programming Massively Parallel Processors:
A Hands-on Approach

By

David B. Kirk & Wen-mei W. Hwu

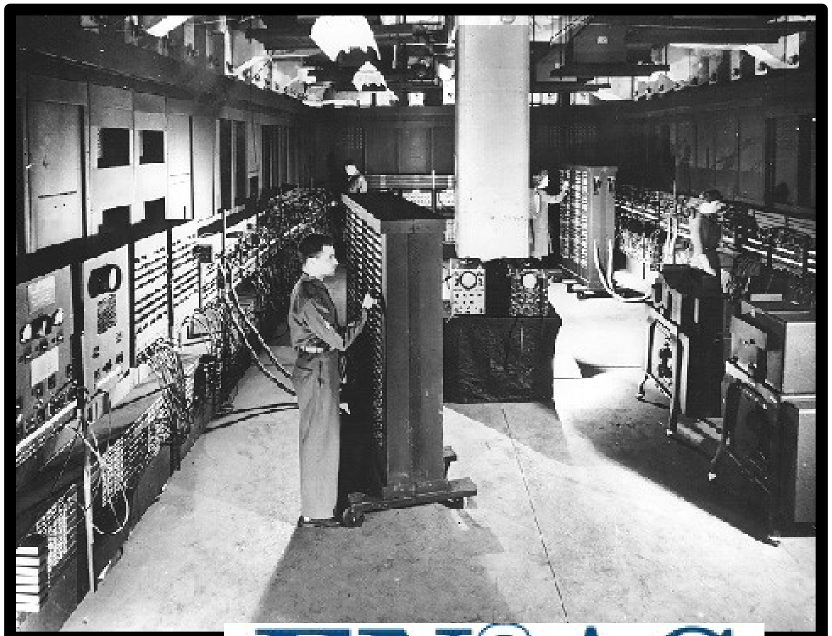
3rd Edition



Grading

- Homework assignments: 15%
- Project: 25%
- Programming assignments: 30%
- Final Exam (open book/notes): 30%

Computer History

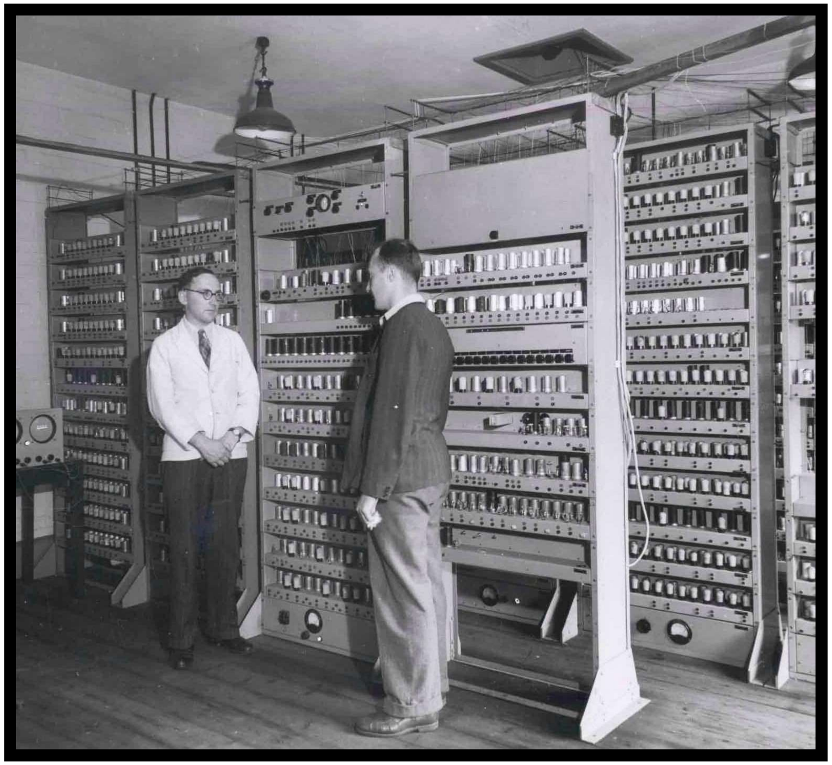


ENIAC

Eckert and Mauchly

- 1^{st} working electronic computer (1946)
- 18,000 Vacuum tubes
- 1,800 instructions/sec
- 3,000 ft^3

Computer History



EDSAC 1 (1949)

http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/

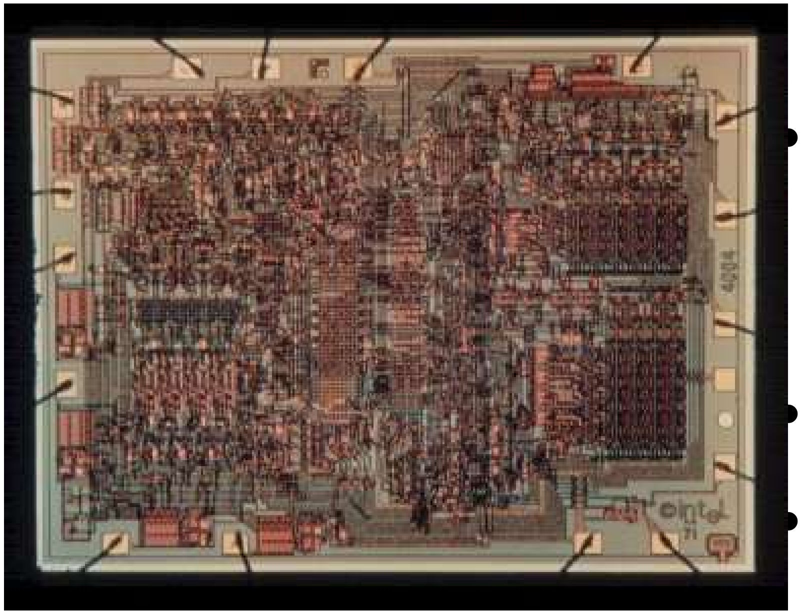
- Maurice Wilkes

1st stored program computer

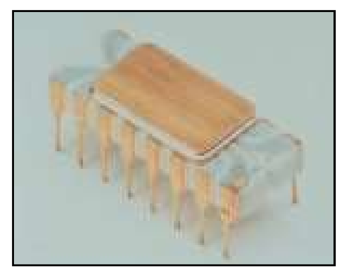
650 instructions/sec

1,400 ft^3

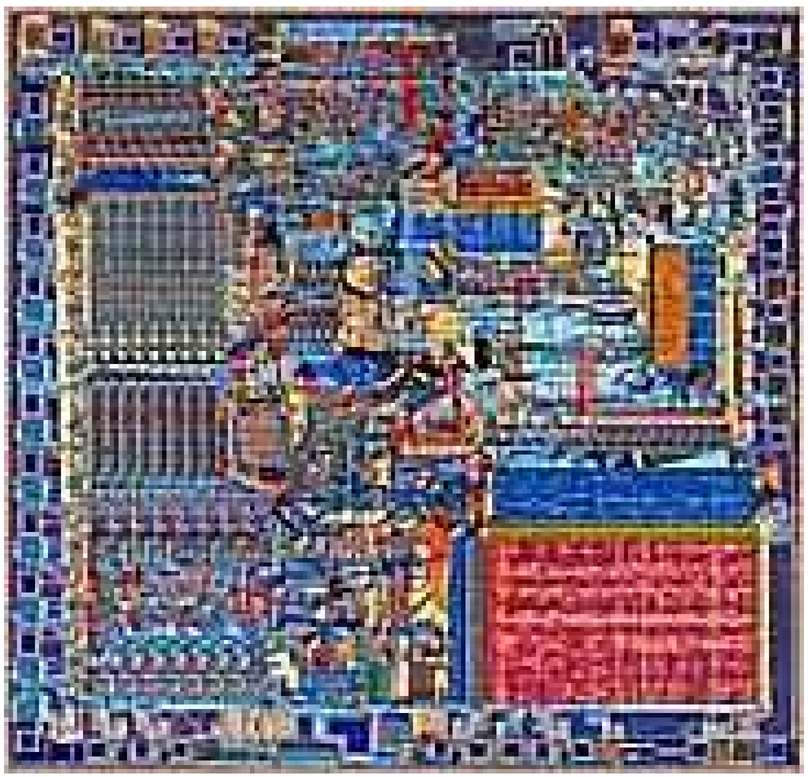
Intel 4004 Die Photo



- Introduced in 1970
  - First microprocessor
- 2,250 transistors
- 12 mm^2
- 108 KHz

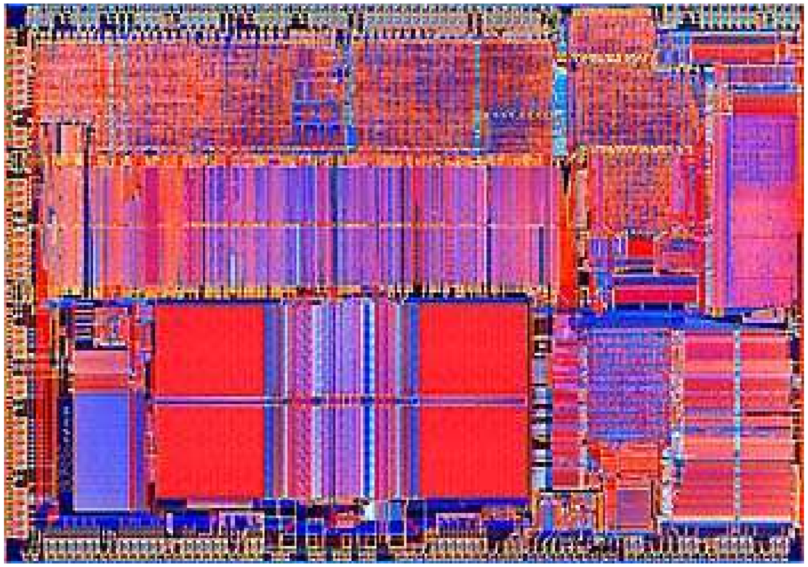


Intel 8086 Die Scan



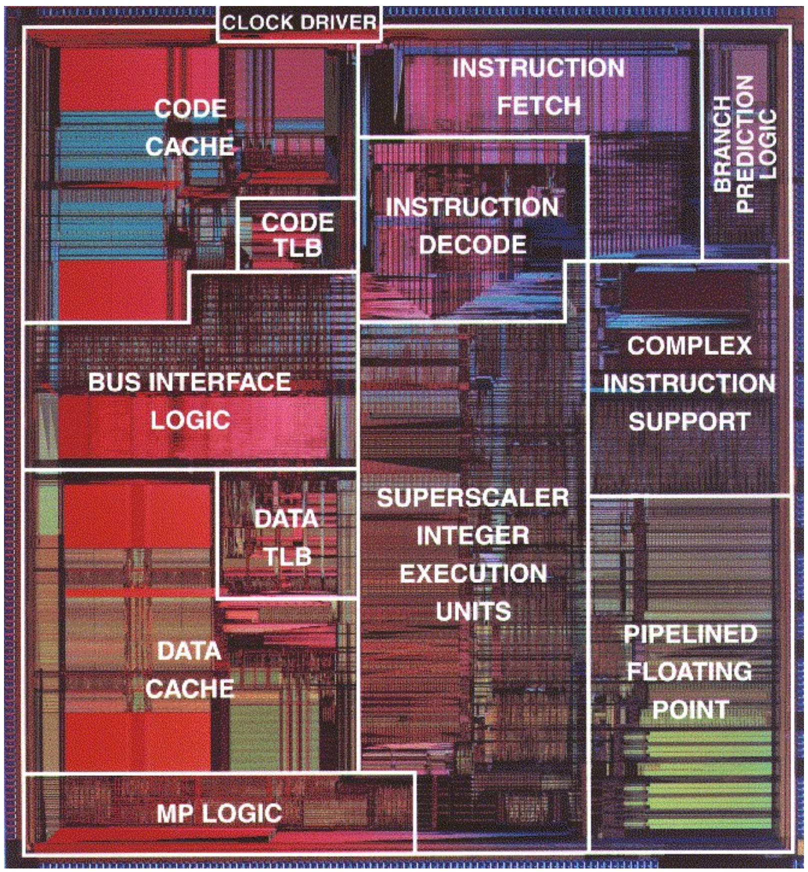
- 29,000 transistors
- 33 mm^2
- 5 MHz
- Introduced in 1979
  - Basic architecture of the IA32 PC

Intel 80486 Die Scan



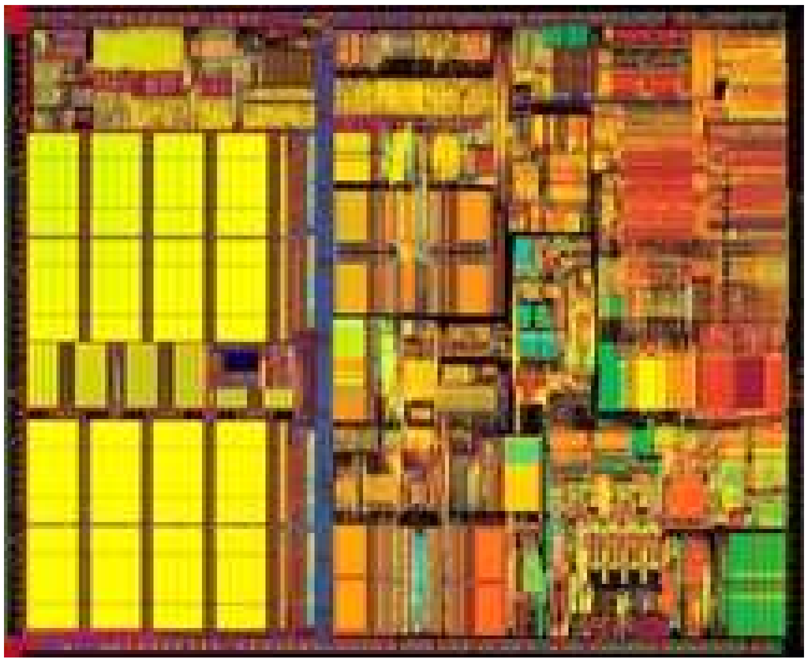
- 1,200,000 transistors
- 81 mm^2
- 25 MHz
- Introduced in 1989
  - 1st pipelined implementation of IA32

Pentium Die Photo



- 3,100,000 transistors
- 296 mm^2
- 60 MHz
- Introduced in 1993
  - 1st superscalar implementation of IA32

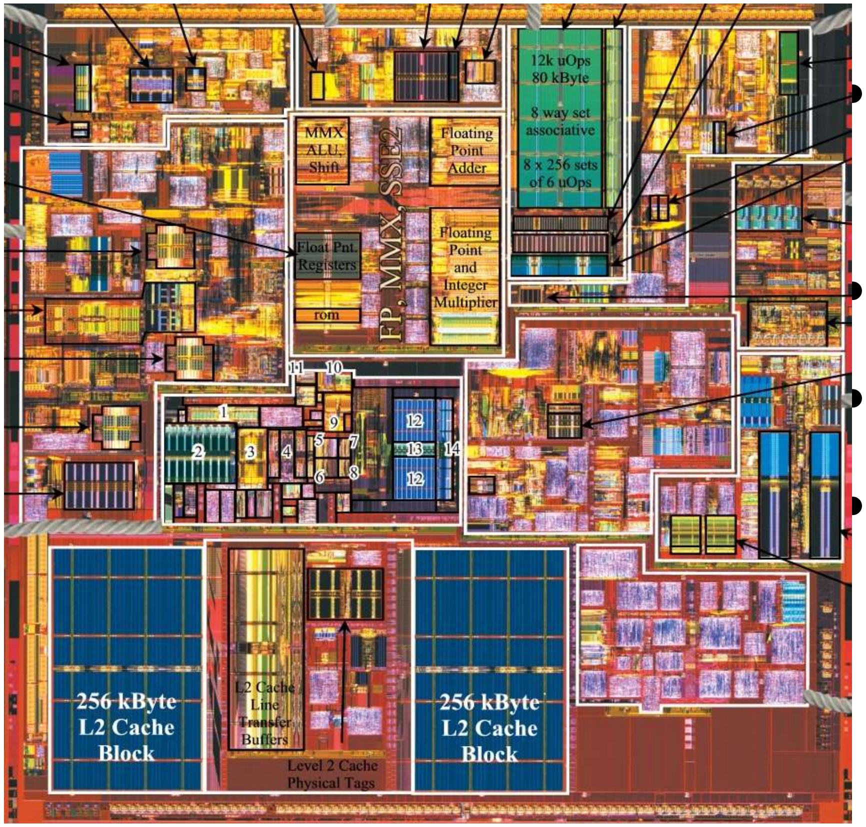
Pentium III



- 9,500,000 transistors
- 125 mm^2
- 450 MHz
- Introduced in 1999

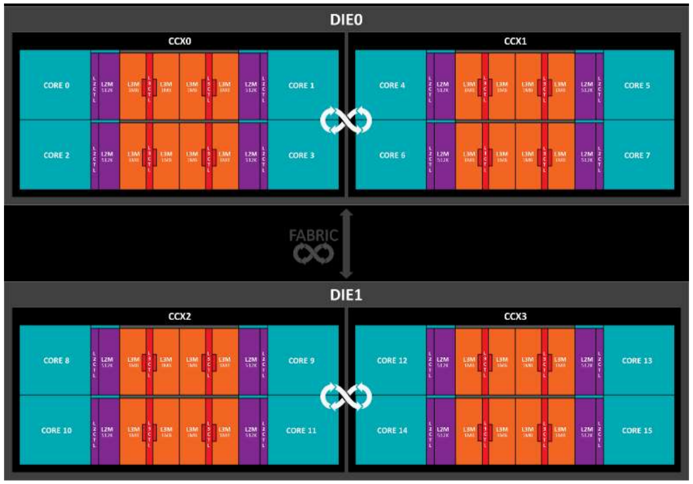
http://www.intel.com/intel/museum/25anniv/hof/hof_main.htm

Pentium 4

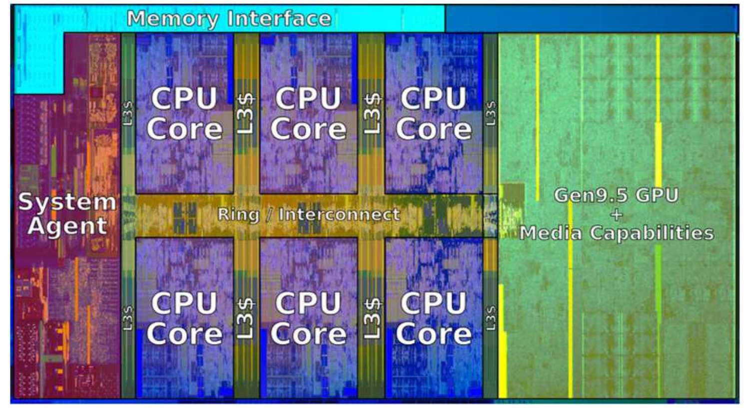


- 55,000,000 transistors
- 146 mm^2
- 3 GHz
- Introduced in 2000

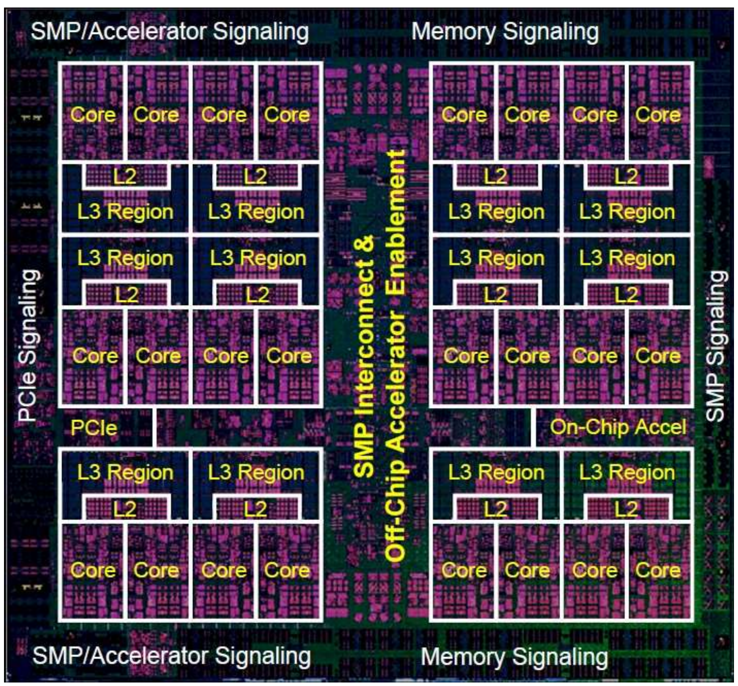
http://www.chip-architect.com



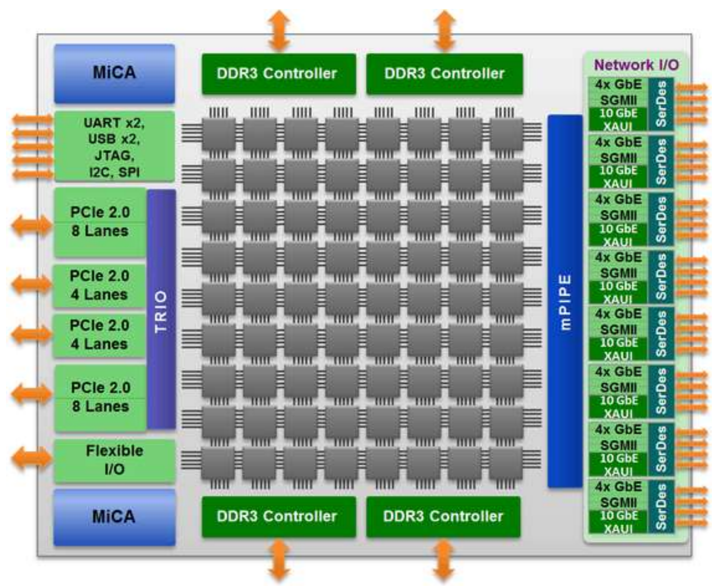
AMD RyZen Threadripper (16 cores)



Intel Core i7 (Coffee Lake)

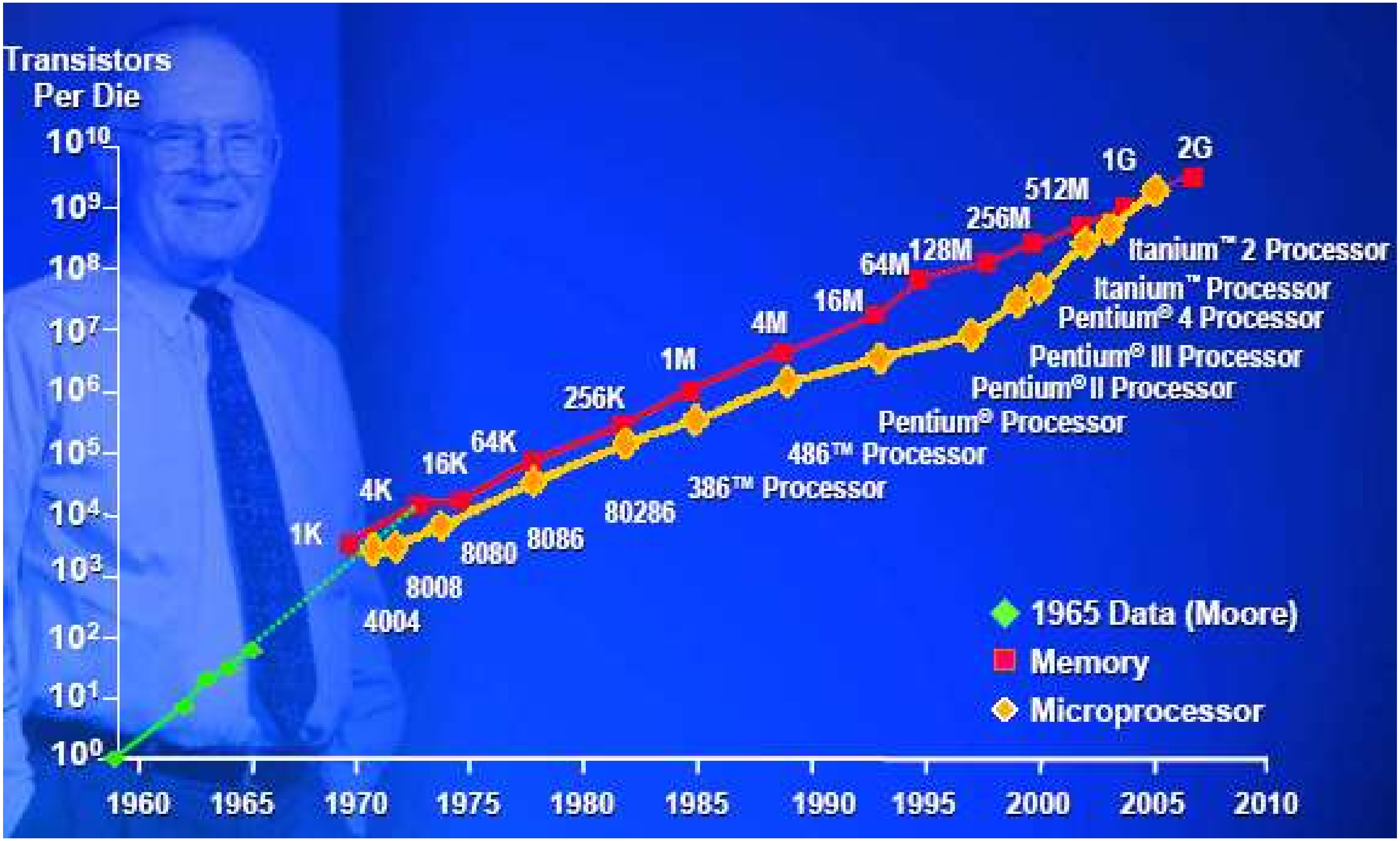


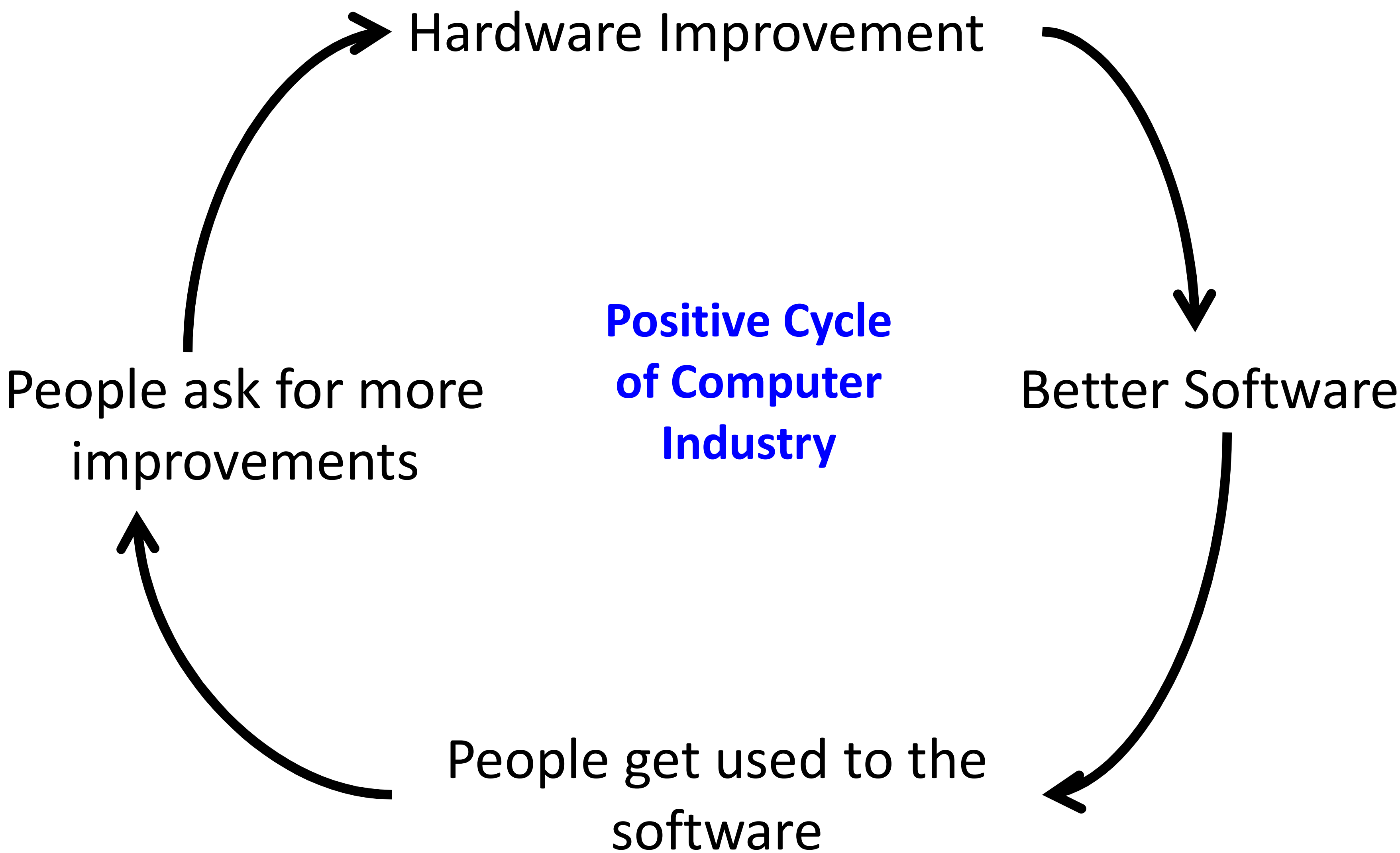
IBM Power 9 (24 cores)



Tilera (72 cores)

The Famous Moore's Law





Software cost dominates hardware cost.

Important Questions

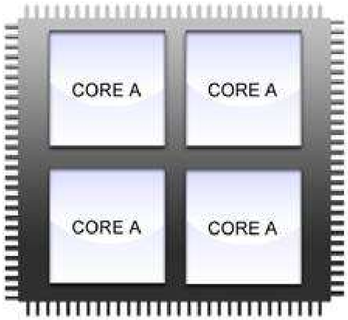
- **How to control software cost?**
  - By reducing redesigning of the software.
- **And how to do that?**
  - By making the application scalable
    - More cores
    - More threads per core
    - More memory
    - Faster interconnect
    - Basically: scalability in the face of hardware growth.
  - By making the application portable
    - Across different instruction sets (x86, ARM, …)
    - From multicore to GPU to FPGA to ….
    - Shared vs distributed memory
    - …

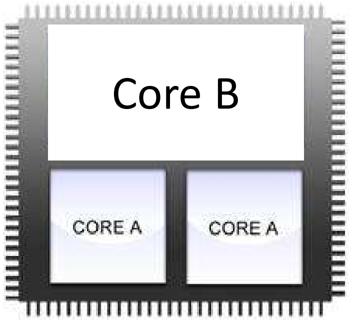
The Status-Quo

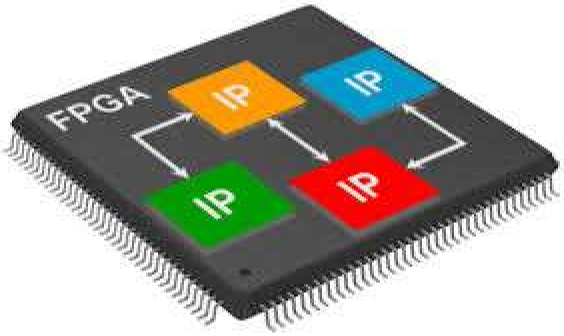
- We moved from single core to multicore
  - for technological reasons
- Free lunch is over for software folks
  - The software will not become faster with every new generation of processors
- Not enough experience in parallel programming
  - Parallel programs of old days were restricted to some elite applications -> very few programmers
  - Now we need parallel programs for many different applications

Not only parallel programming

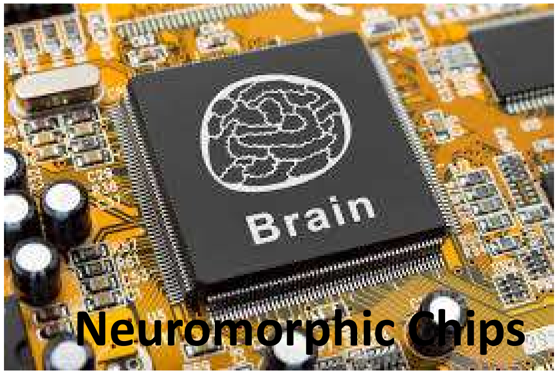
But Heterogeneous parallel programming!

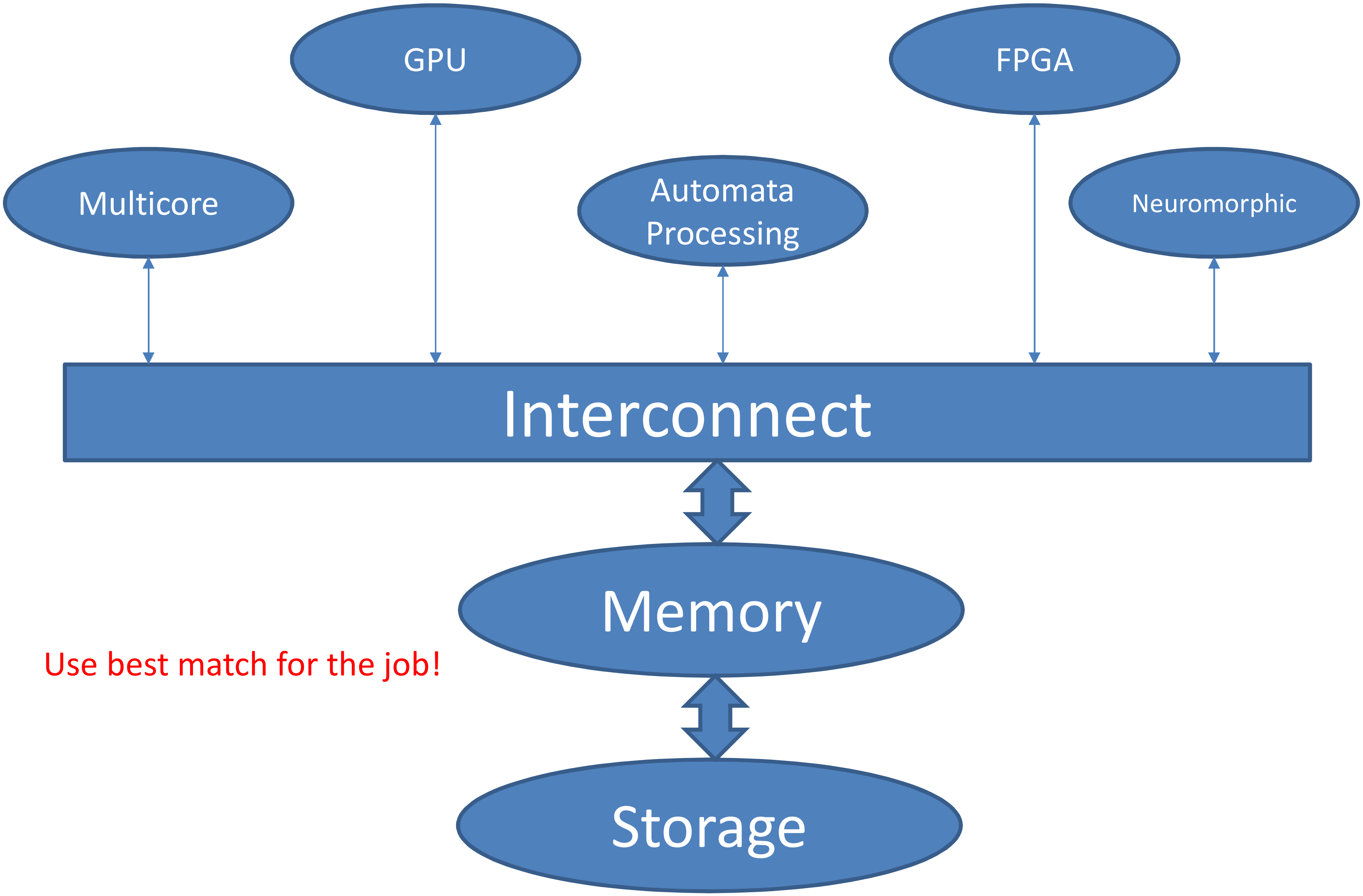






Heterogeneity Everywhere





Use best match for the job!

Software Perspective

Two type of developers

Performance Group

(C/C++, CUDA, OpenCL, ….)

Productivity Group

(Python, Scala, …)

Attempts to Make Parallel Programming Easy

- 1st idea: The right computer language would make parallel programming straightforward
  - **Result so far**: Some languages made parallel programming easier, but none has made it as fast, efficient, and flexible as traditional sequential programming.

Attempts to Make Parallel Programming Easy

- 2nd idea: If you just design the hardware properly, parallel programming would become easy.
  - **Result so far**: no one has yet succeeded!

Attempts to Make Parallel Programming Easy

- 3rd idea: Write software that will automatically parallelize existing sequential programs.
  - **Result so far**: Success here is inversely proportional to the number of cores!

Two Main Goals

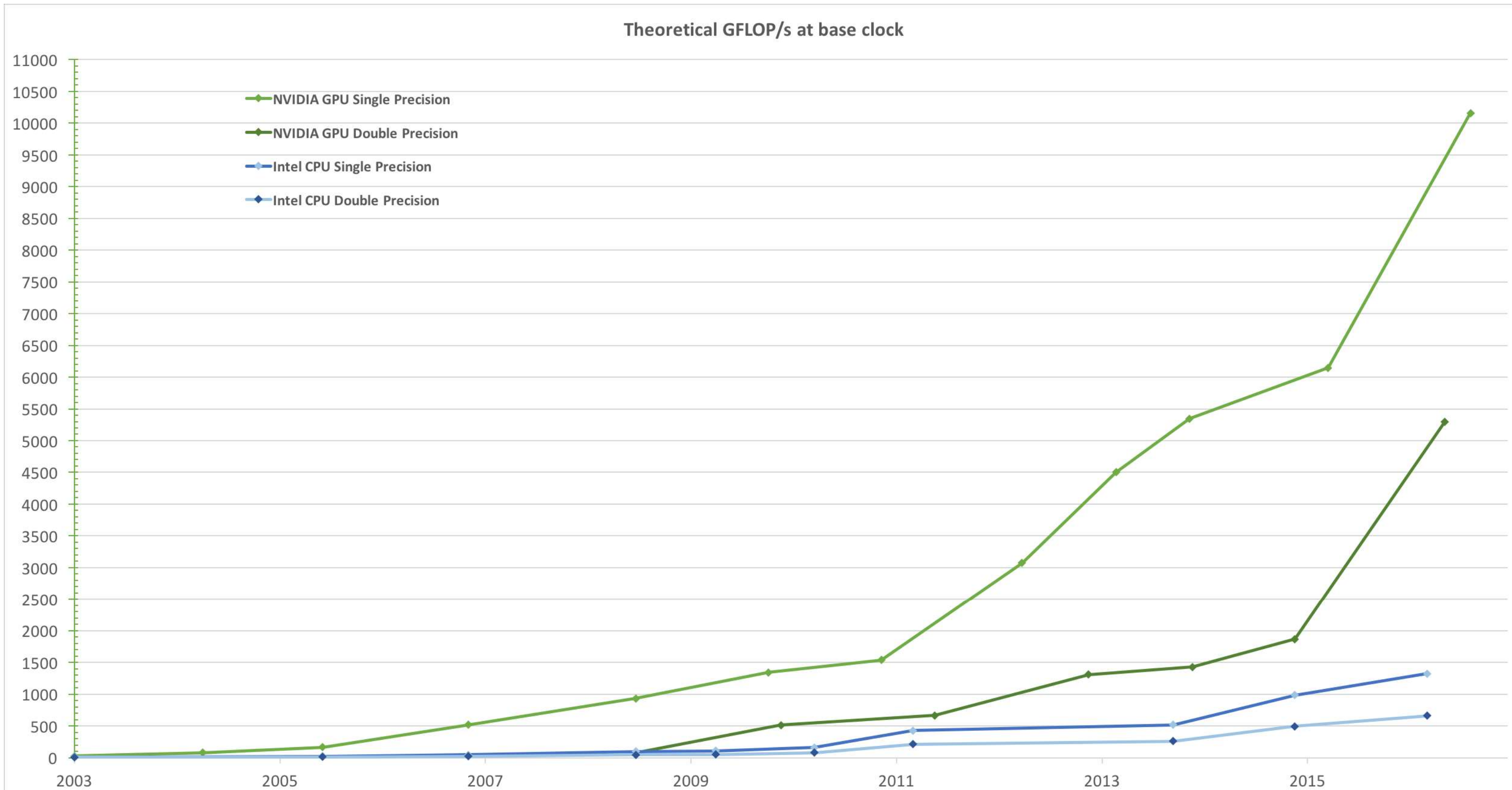
- Maintain execution speed of old sequential programs

- Increase throughput of parallel programs

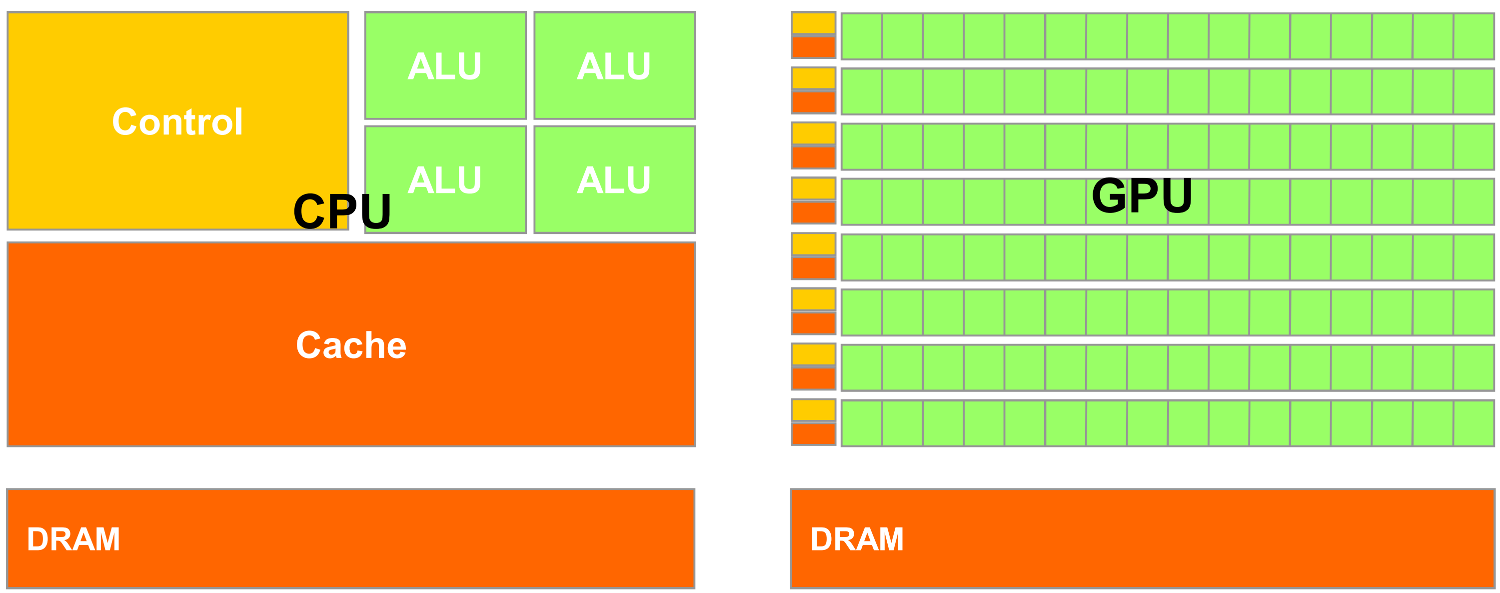
Two Main Goals

- Maintain execution speed of old sequential programs → CPU
- Increase throughput of parallel programs → GPU + CPU

Performance

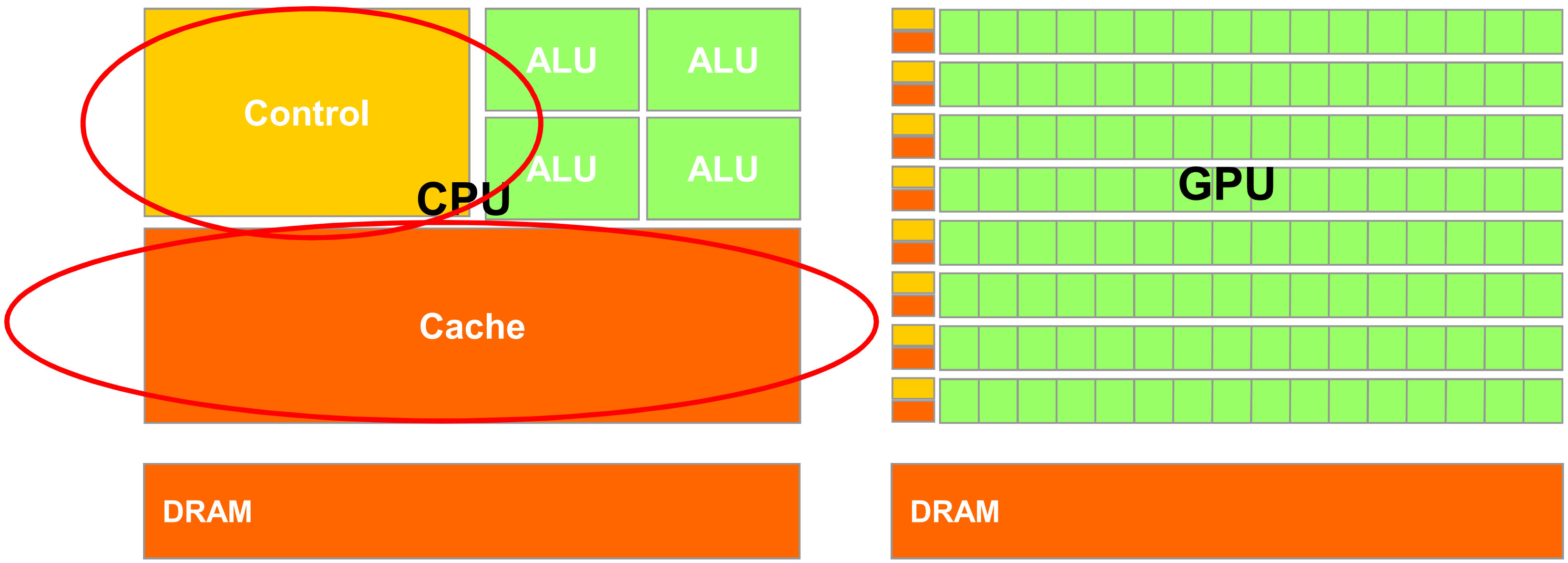


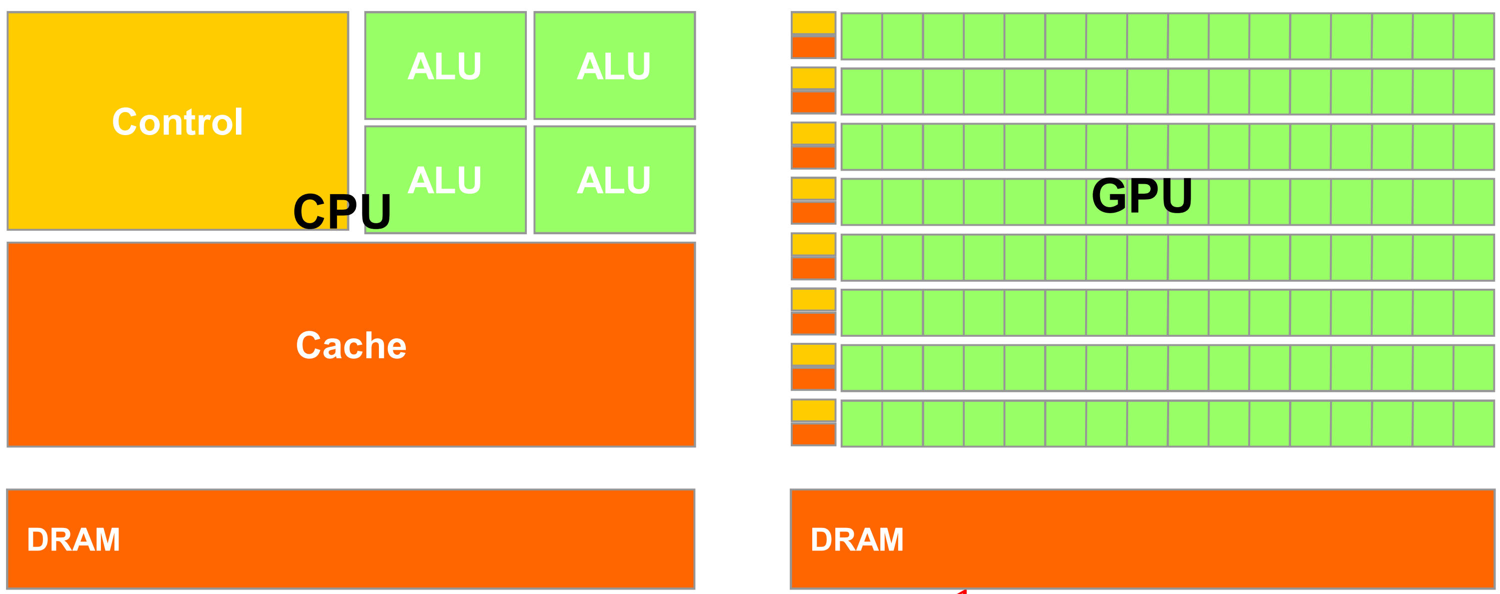
Source: NVIDIA CUDA C Programming Guide



Control
ALU
ALU
ALU
ALU
CPU
Cache
DRAM
GPU
DRAM

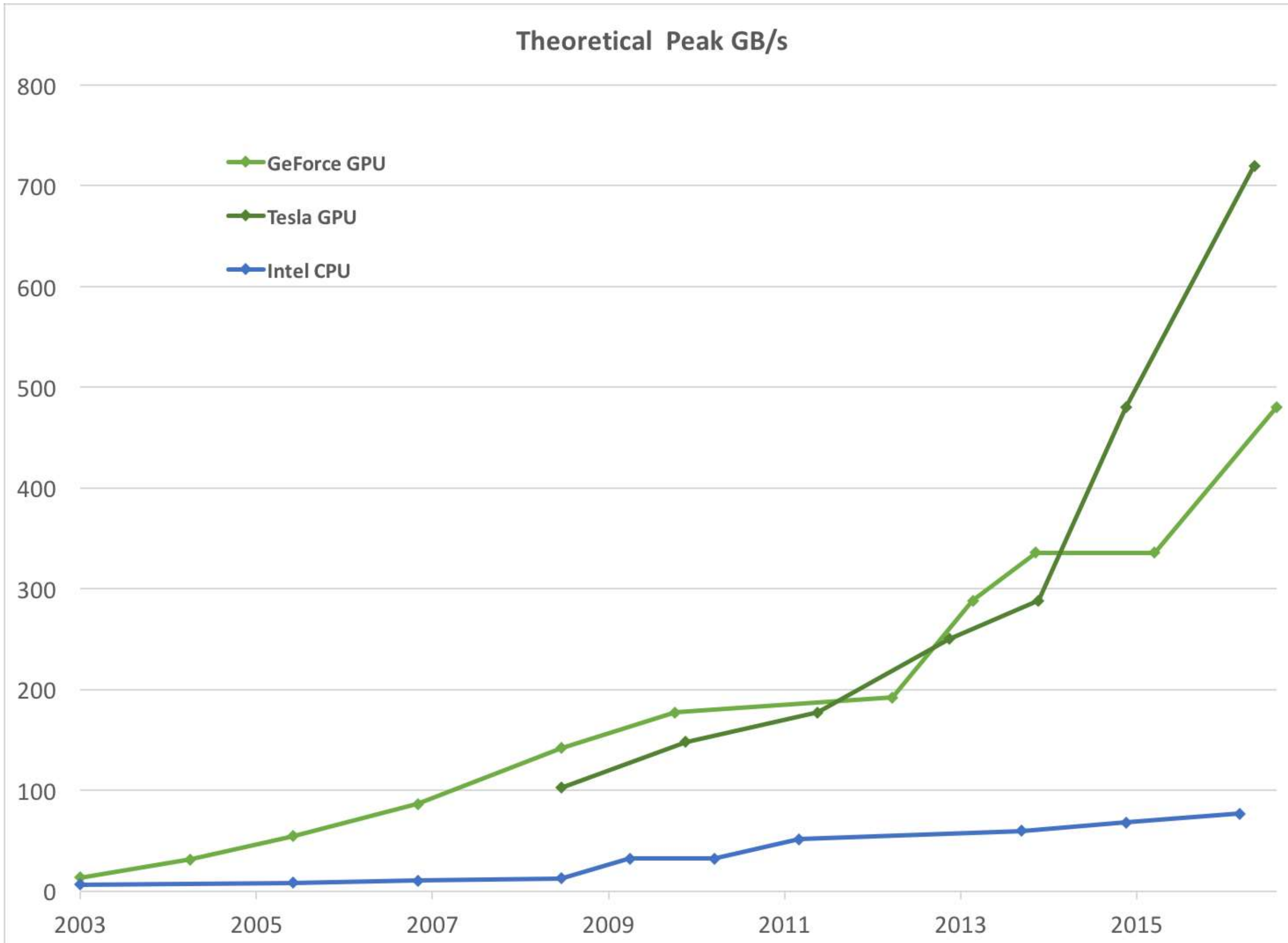
CPU is optimized for sequential code performance





Almost 10x the bandwidth of multicore (relaxed memory model)

Memory Bandwidth

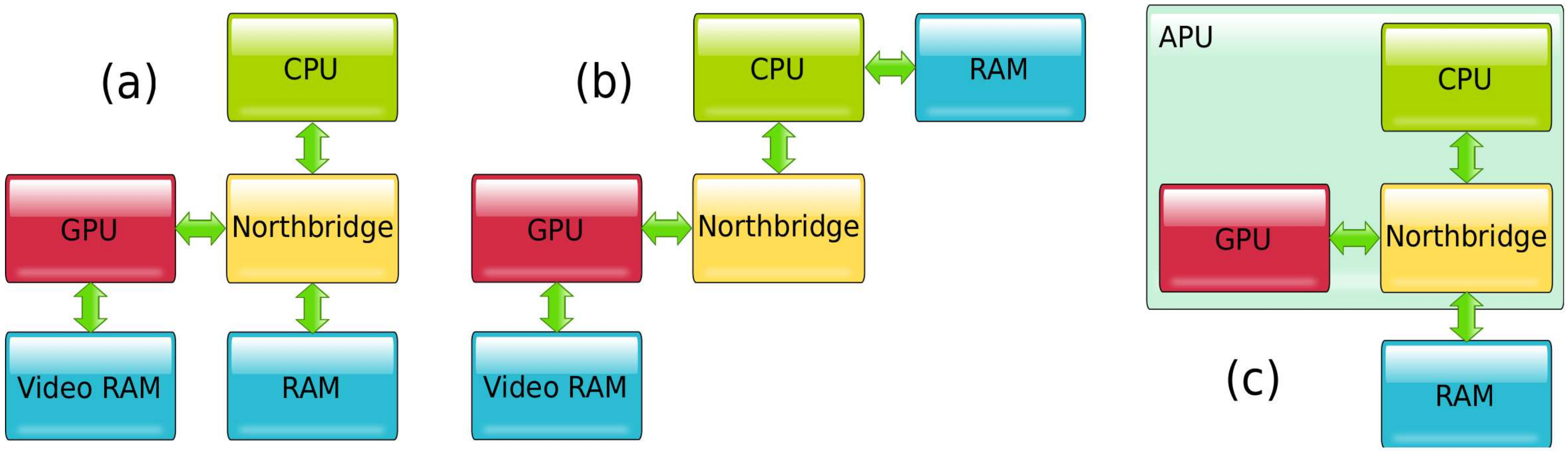


Source: NVIDIA CUDA C Programming Guide

How to Choose A Processor for Your Application?

- Performance
- Very large installation base
- Practical form-factor and easy accessibility
- Support for IEEE floating point standard

Integrated GPU vs Discrete GPU



- (a) and (b) represent discrete GPU solutions, with a CPU-integrated memory controller in (b). Diagram (c) corresponds to integrated CPU-GPU solutions, as the AMD's Accelerated Processing Unit (APU) chips.

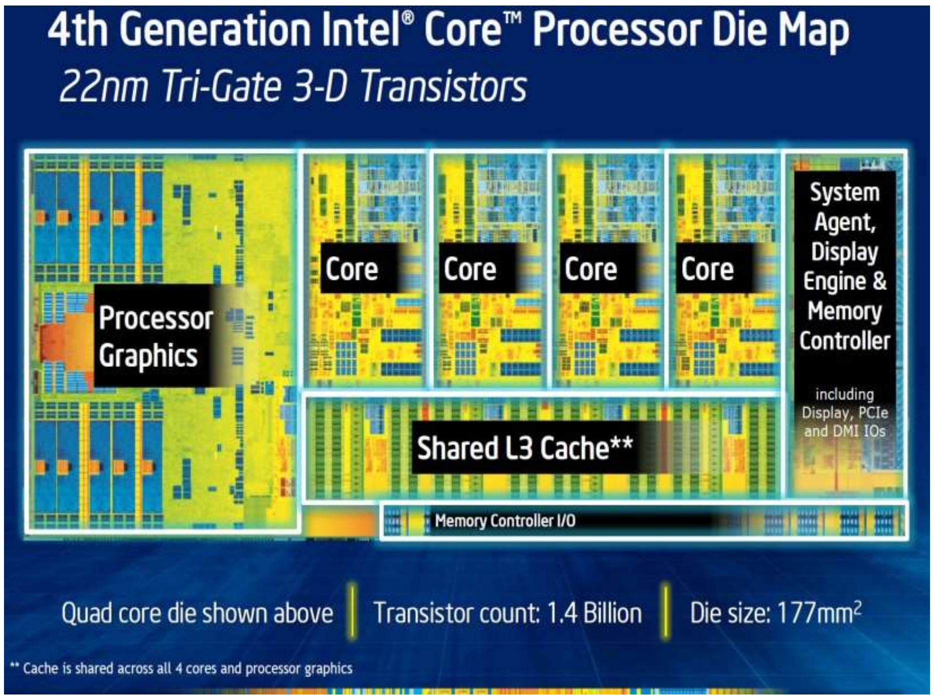
source: *Multicore and GPU Programming: An Integrated Approach by* G. Barlas, 2014

Tradeoff: Low energy vs higher performance

Integrated CPU+GPU processors

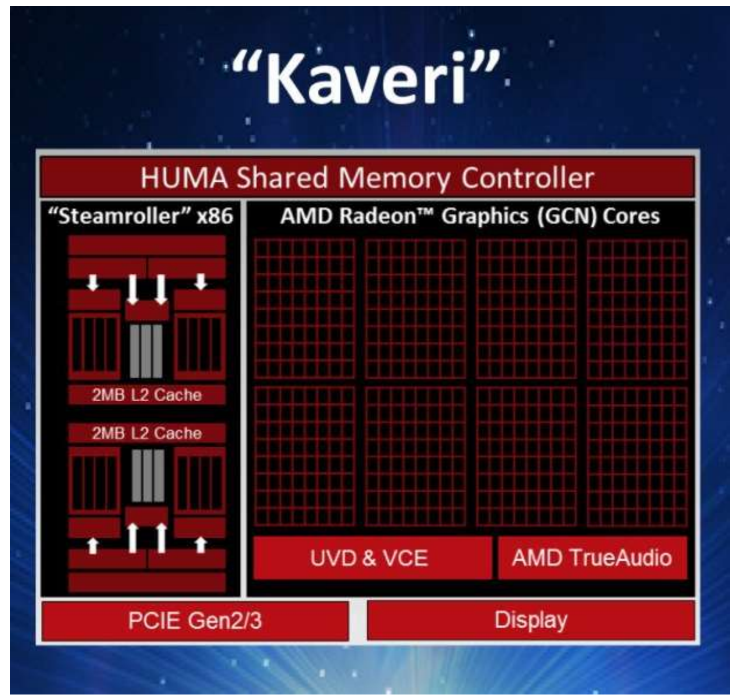
- More than 90% of processors shipping today include a GPU on die
- Low energy use is a key design goal

Intel 4th Generation Core Processor: "Haswell"



4-core GT2 Desktop: **35 W package**
2-core GT2 Ultrabook: **11.5 W package**

AMD Kaveri APU



http://www.geeks3d.com/20140114/amd-kaveri-a10-7850k-a10-7700k-and-a8-7600-apus-announced/

Desktop: **45-95 W package**
Mobile, embedded: **15 W package**

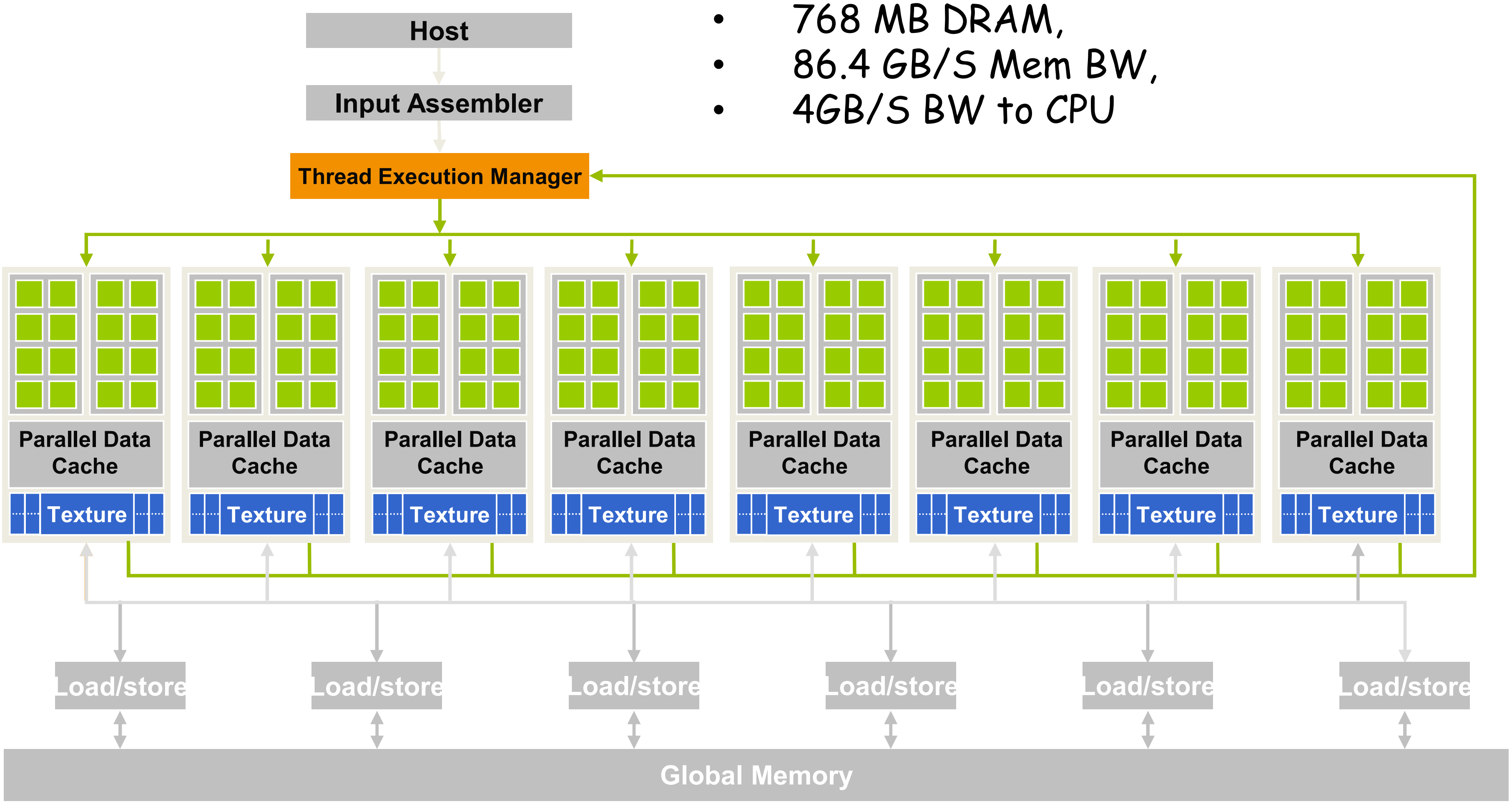
source: Performance and Programmability Trade-offs in the OpenCL 2.0 SVM and Memory Model
by Brian T. Lewis, Intel Labs

Is Any Application Suitable for GPU?

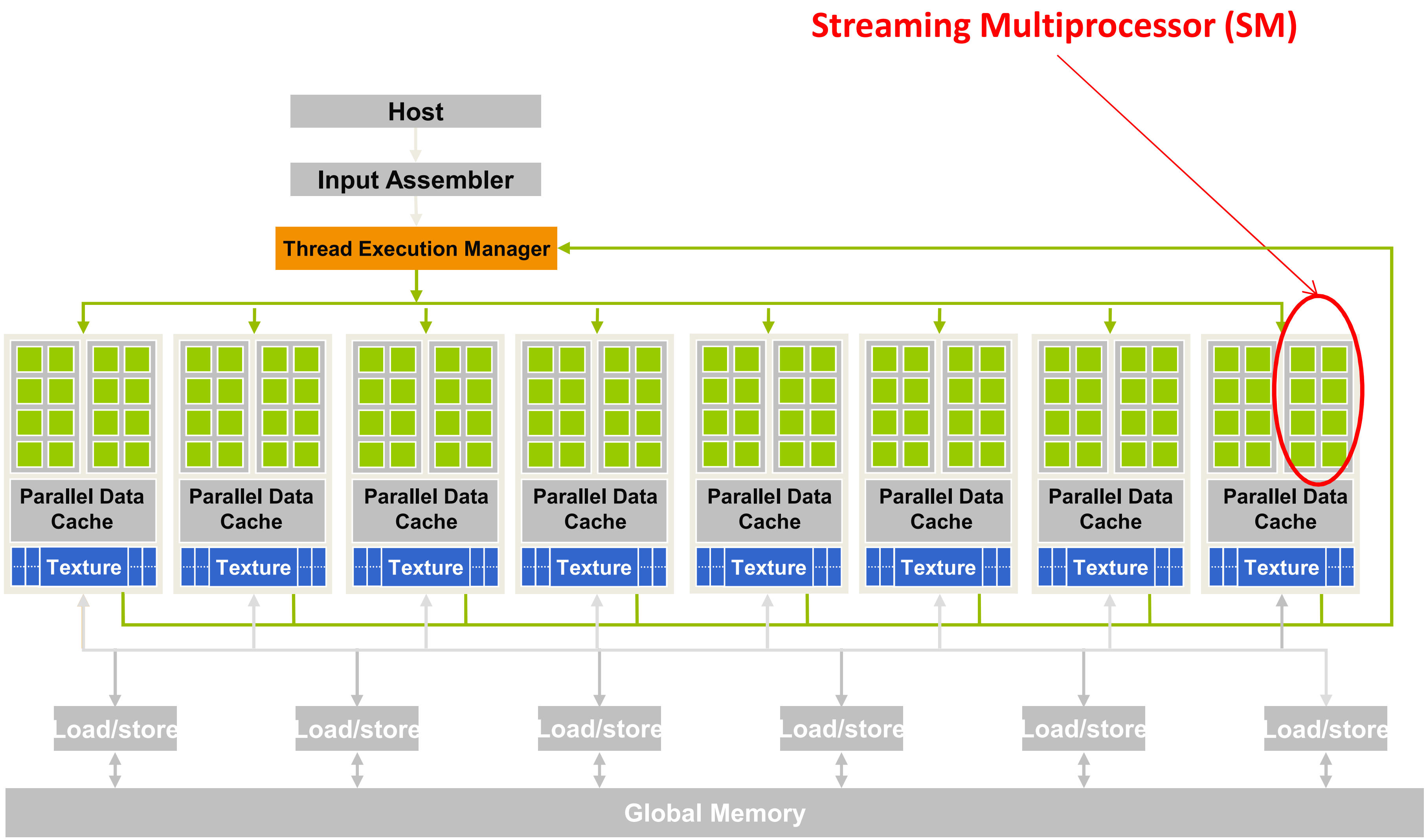
- Heck no!
- You will get the best performance from GPU if your application is:
  - Computation intensive
  - Many independent computations
  - Many similar computations

A Glimpse at A GPGPU: GeForce 8800 (2007)

- 16 highly threaded SM's,
- >128 FPU's, 367 GFLOPS,
- 768 MB DRAM,
- 86.4 GB/S Mem BW,
- 4GB/S BW to CPU



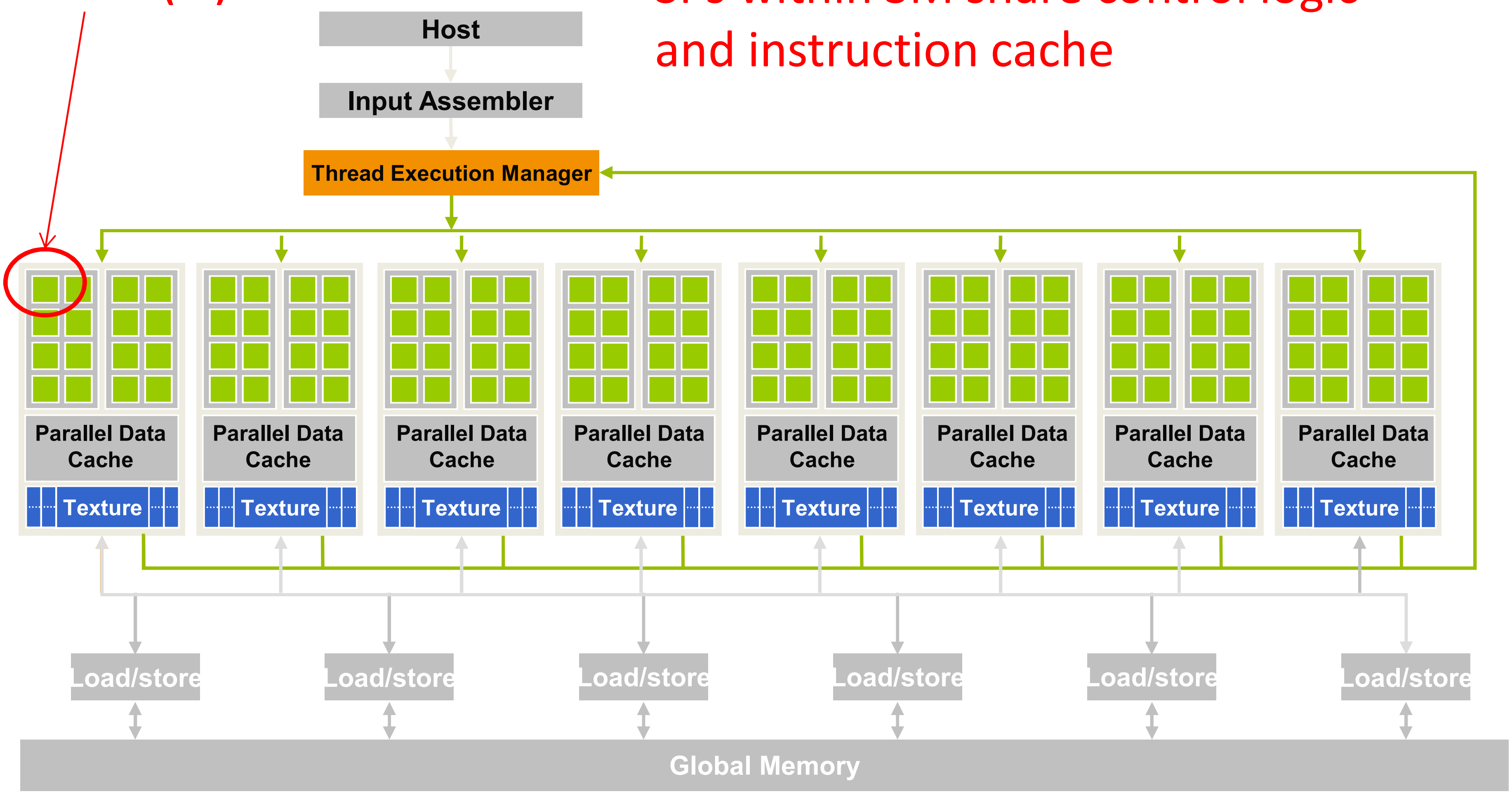
A Glimpse at A GPU



A Glimpse at A Modern GPU

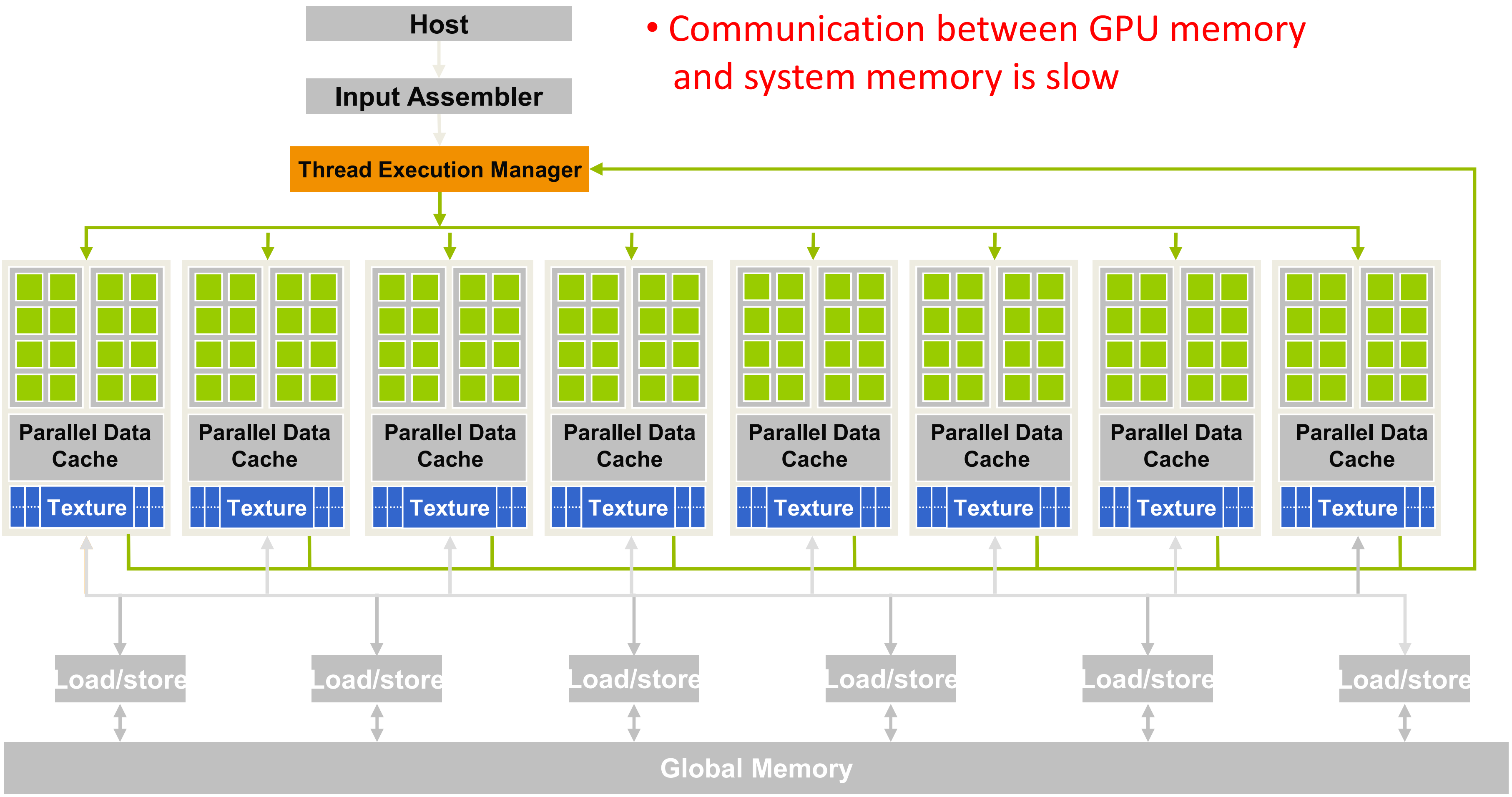
Streaming Processor (SP)

SPs within SM share control logic and instruction cache



A Glimpse at A Modern GPU

- Much higher bandwidth than typical system memory
- A bit slower than typical system memory
- Communication between GPU memory and system memory is slow



Amdahl's Law

Execution Time After Improvement =
Execution Time Unaffected +(Execution Time Affected / Amount of Improvement)

- Example:

  "Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

  How about making it 5 times faster?

Improvement in your application speed depends on the portion that is parallelized

Winning Applications Use Both CPU and GPU

- CPUs for sequential parts where latency matters
  - CPUs can be 10X+ faster than GPUs for sequential code

- GPUs for parallel parts where throughput wins
  - GPUs can be 10X+ faster than CPUs for parallel code

Source: NVIDIA GPU teaching kit

Things to Keep in Mind

- Try to increase the portion of your program that can be parallelized
- Figure out how to get around limited bandwidth of system memory
- When an application is suitable for parallel execution, a good implementation on GPU can achieve more than 100x speedup over sequential implementation.
- You can reach 10x fairly easy, beyond that … stay with us!

Enough for Today

- Some applications are better run on CPU while others on GPU
- If you don't care about performance, parallel programming is easy!
- Main limitations
  - The parallelizable portion of the code
  - The communication overhead between CPU and GPU
  - Memory bandwidth saturation

Welcome … And Have Fun!