EASTERN WASHINGTON UNIVERSITY
start something big

GPU Programming Intro.

Computer Science Department
Eastern Washington University
Yun Tian (Tony) Ph.D.

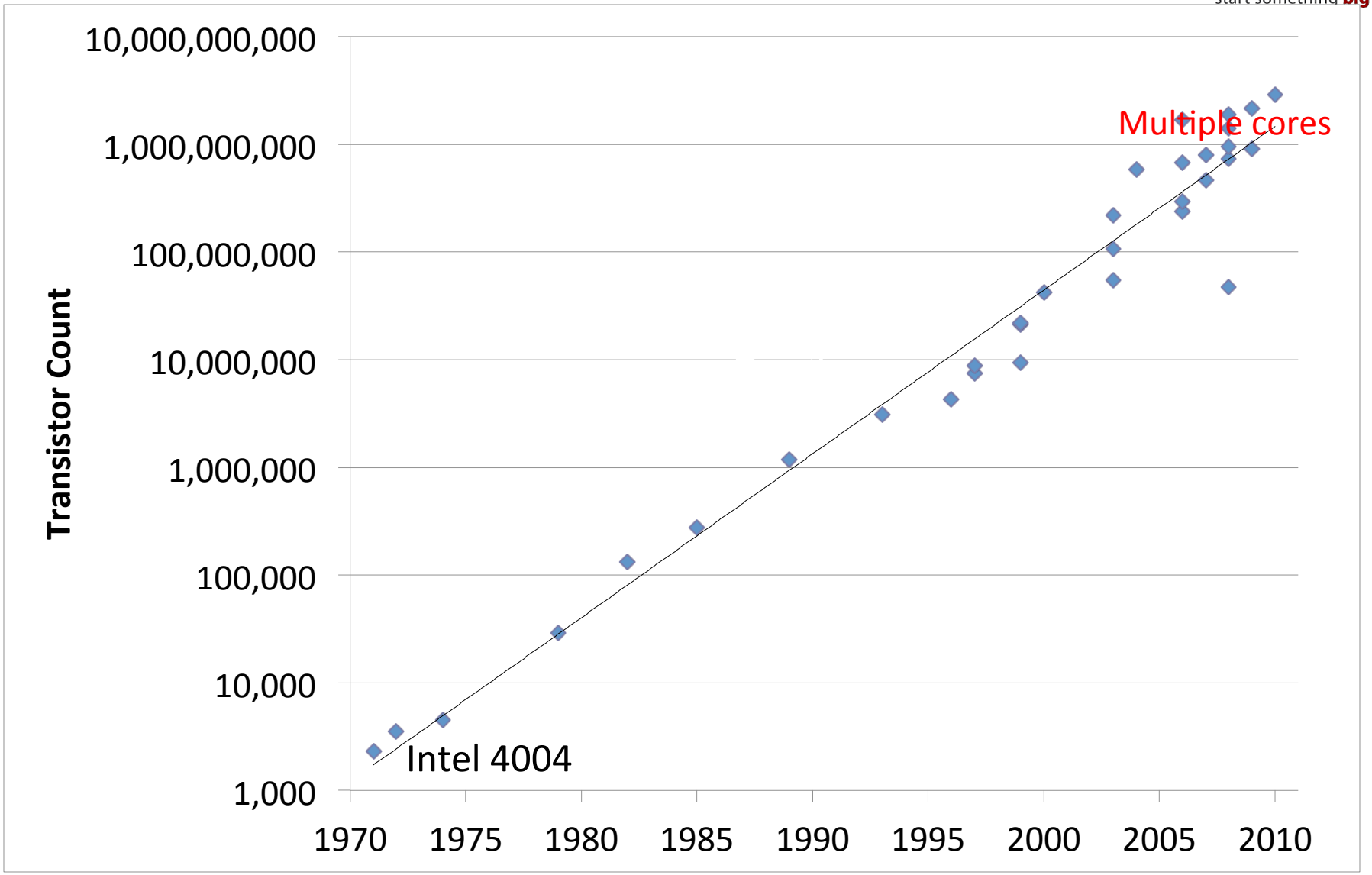
Outline Today

- Why parallel programming?
- Multicore and manycore systems
- GPU Architecture

Moore's Law

- The number of transistors on an integrated circuit doubles every two years."

  – Gordon E. Moore


Transistor Count
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010
Multiple cores
Intel 4004

Serial Performance Scaling is Over

- Two decade ago, we can buy a better(faster) CPU to speed up our sequential program.
- Nowadays, CPU clock speed and heat issues bottleneck a single processor's speed.
  - Clock speed reaches its limit also.
    - No 10GHz chips.
  - Density of transistors reach limit.
  - Heat
    - More power consumption will cause more heat.
    - Heat could burn the whole chip.

Multi-core or Multi-processor

- How to continue to maintain Moore's Law true?
  - Multi-core systems and manycore systems, e.g. Intel i7-4770 (4 cores) and GPUs

Parallel is the Future

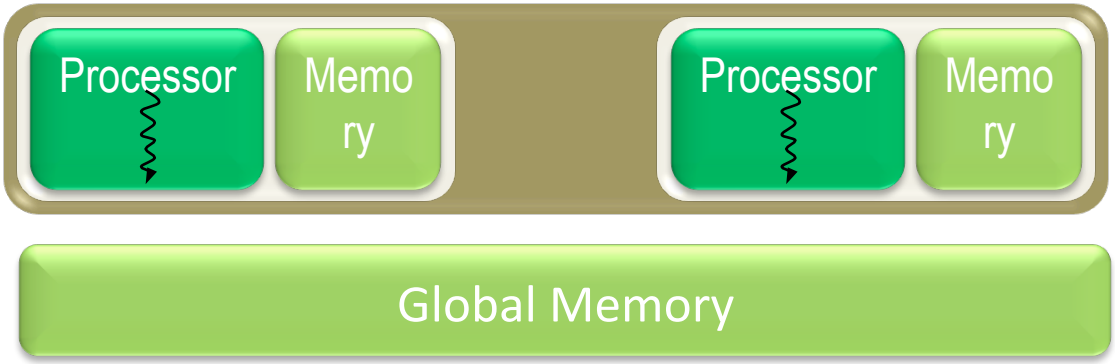
- The tide of commodity technology is rising in a different direction.
  - Computers have stopped getting faster (higher clock speeds), they just get wider (more parallel).
  - If you work on an interesting problem, you must rethink your algorithms to be parallel.
    - Otherwise, if your serial algorithm isn't fast enough today it never will be fast enough.

Parallel is the Future

- Data Parallel is scalable
  - Scalable: if we use more cores/processors, it gets faster to process a same dataset.
  - If your problem is of interesting size the data will always outnumber the cores,
  - So organize the computation around the data.
  - Big Data

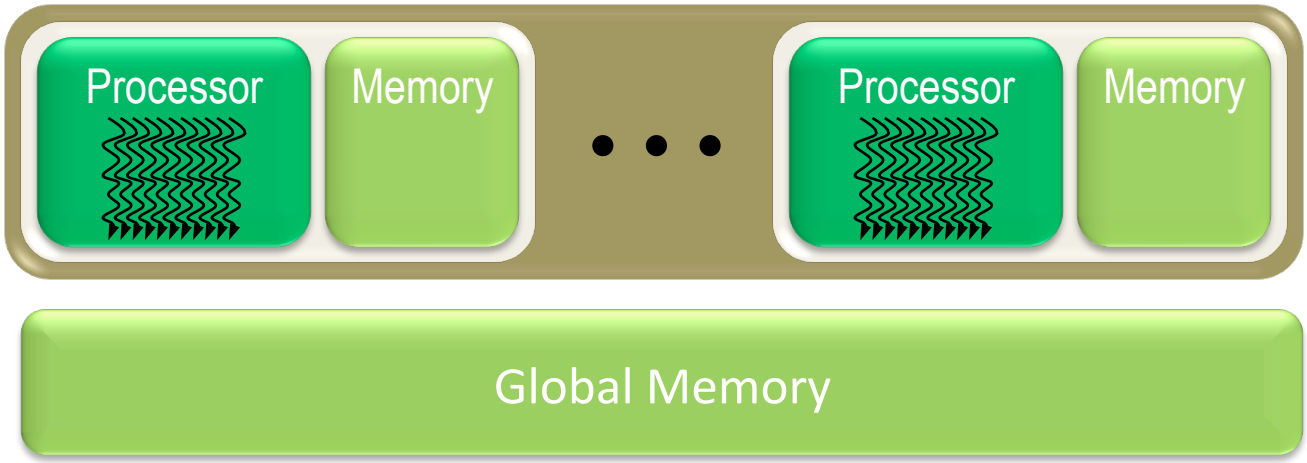
Trend One – Generic Multicore CPU

- One instance of this idea is the current wave of multicore CPUs.
  - Handful of processors each supporting 1 hardware thread.
  - On-chip memory near processors (cache, RAM, or both)
  - Shared global memory space (external DRAM).

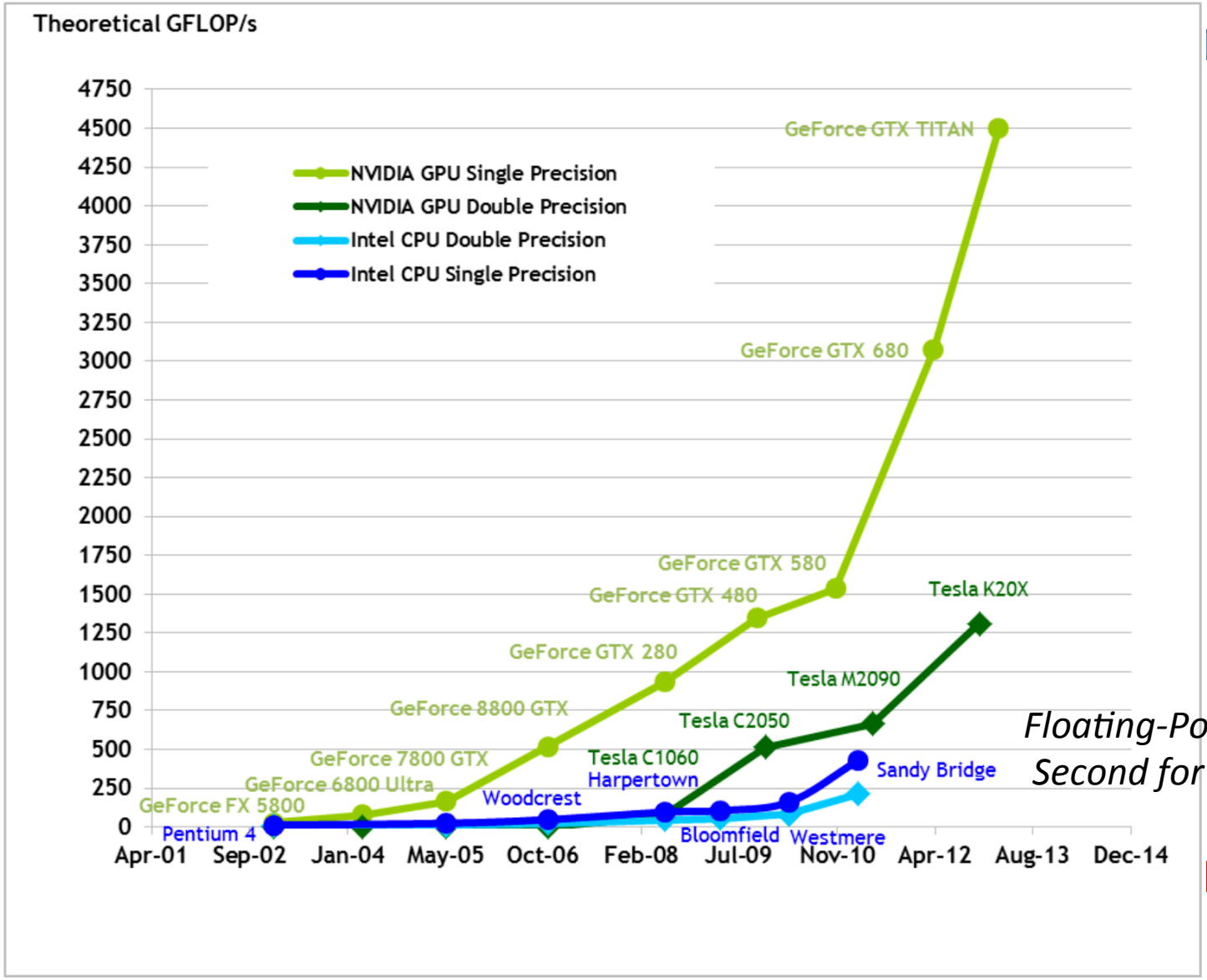


Trend Two – Generic Manycore Chip

- Alternatively, a manycore parallel architecture replicates many processor cores,
- Each hosting several lightweight hardware threads.

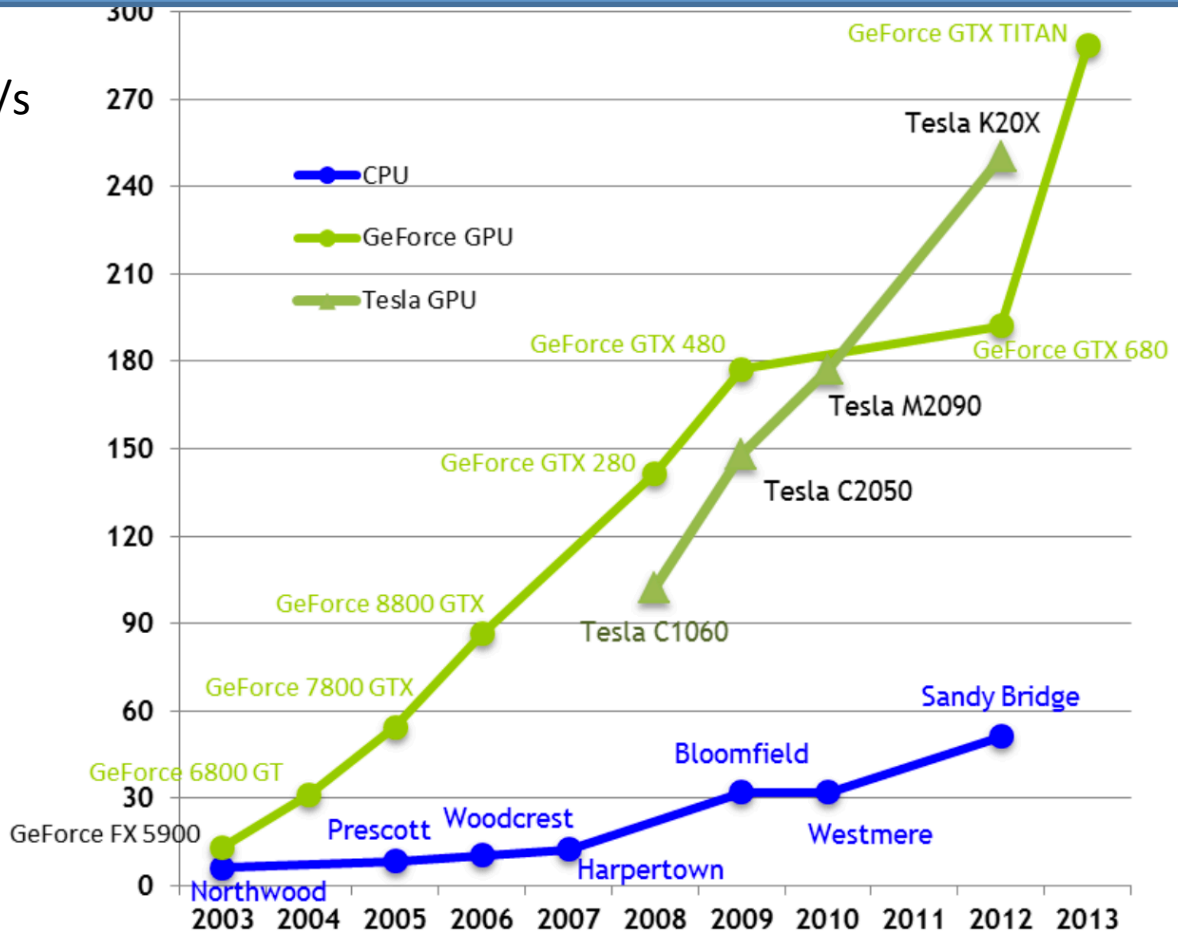


Why Massively Parallel Processing?



Floating-Point Operations per Second for the CPU and GPU

Why Massively Parallel Processing?



Memory Bandwidth for the CPU and GPU

GPU Computing Era

- Lessons from Graphics Pipeline
- Throughput is paramount
  - must paint every pixel within frame time
  - Scalability for more input data
- Create, run, & retire lots of threads very rapidly
  - measured 14.8 Gthread/s in test
- Use multithreading to hide latency
  - 1 stalled thread is OK if 100 are ready to run

CPU differs from GPU

- Different goals produce different designs
  - GPU assumes work load is highly parallel
  - CPU must be good at everything, parallel or not
- CPU: minimize latency experienced by 1 thread
  - big on-chip caches
  - sophisticated control logic
- GPU: maximize execution throughput of all threads
  - # threads in flight limited by resources => lots of resources (registers, bandwidth, etc.)
  - multithreading can hide latency => skip the big caches
  - share control logic across many threads
  - Global Memory channels and arithmetic op. have long latency.

GPU Computing Era

- High throughput computation
  - GeForce GTX 680: over 1T FLOP/s
- High bandwidth memory
  - GeForce GTX 680: over 180 GB/s
- High availability to all
  - 400+ million CUDA-capable GPUs in the wild

GPU Architecture, Kepler GK110

SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

GPU Architecture Idea

- SM(Streaming Multiprocessor at the heart of the NVIDIA GPU architecture.
- The individual scalar cores(green squares) from the last slide are assembled into groups of 32 in an SM.
- SM = Streaming Multiprocessor
- SMX in Kepler Arch. = new Streaming Mulitprocessor
- SP = Streaming Processor (A scalar core)

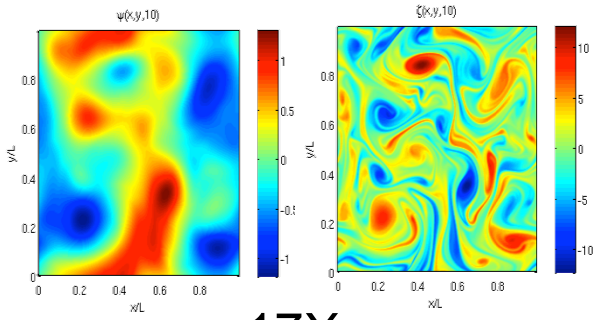
GPU Architecture Idea

- SIMT (Single Instruction Multiple Thread) execution
  - threads run in groups of 32 called warps
    - Warps are the unit of scheduling.
  - threads **in a warp** share instruction unit (IU)
    - Execute a same instruction at one time, like SIMD
  - HW automatically handles divergence
    - First for threads execute if true branches.
    - Then for threads execute if false branches.

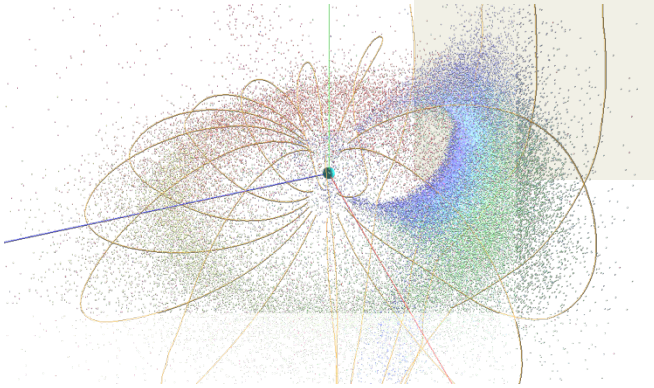
GPU Architecture Idea

- Hardware multithreading (light-weight thread)
  - HW resource allocation & thread scheduling
  - HW relies on threads to hide latency
- Threads have all resources needed to run
  - any warp not waiting for something can run
  - context switching is (basically) free

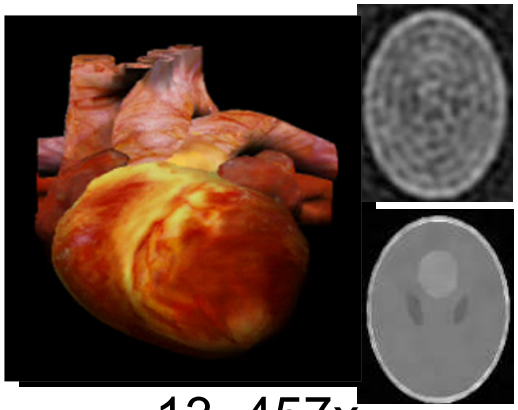
45X



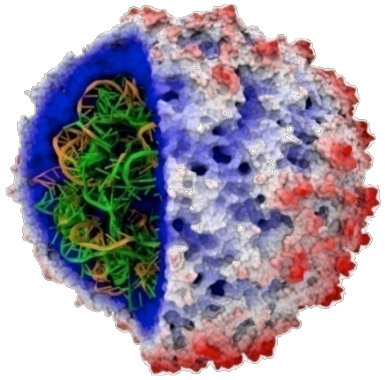
17X



100X

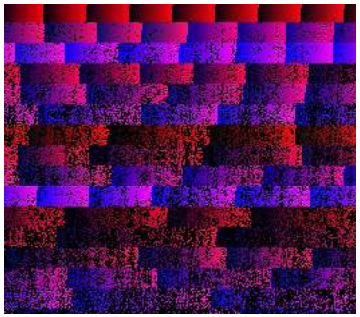


13–457x



110-240X

Motivation



35X

Summary

- CPU: minimize latency
- GPU: maximize throughput
  - High bandwidth
  - Big latency
  - Multithreading hides latency
- GPU architecture ideas
  - SM
  - SP

Next Class

- Chapter 2
- History of GPU hardware and GPU computing
- Auto Scalable GPUs