Santa Clara University
School of Engineering

Introduction Programming Massively Parallel Processors

Dr. Maria Pantoja

Motivation

Text & Notes

References:

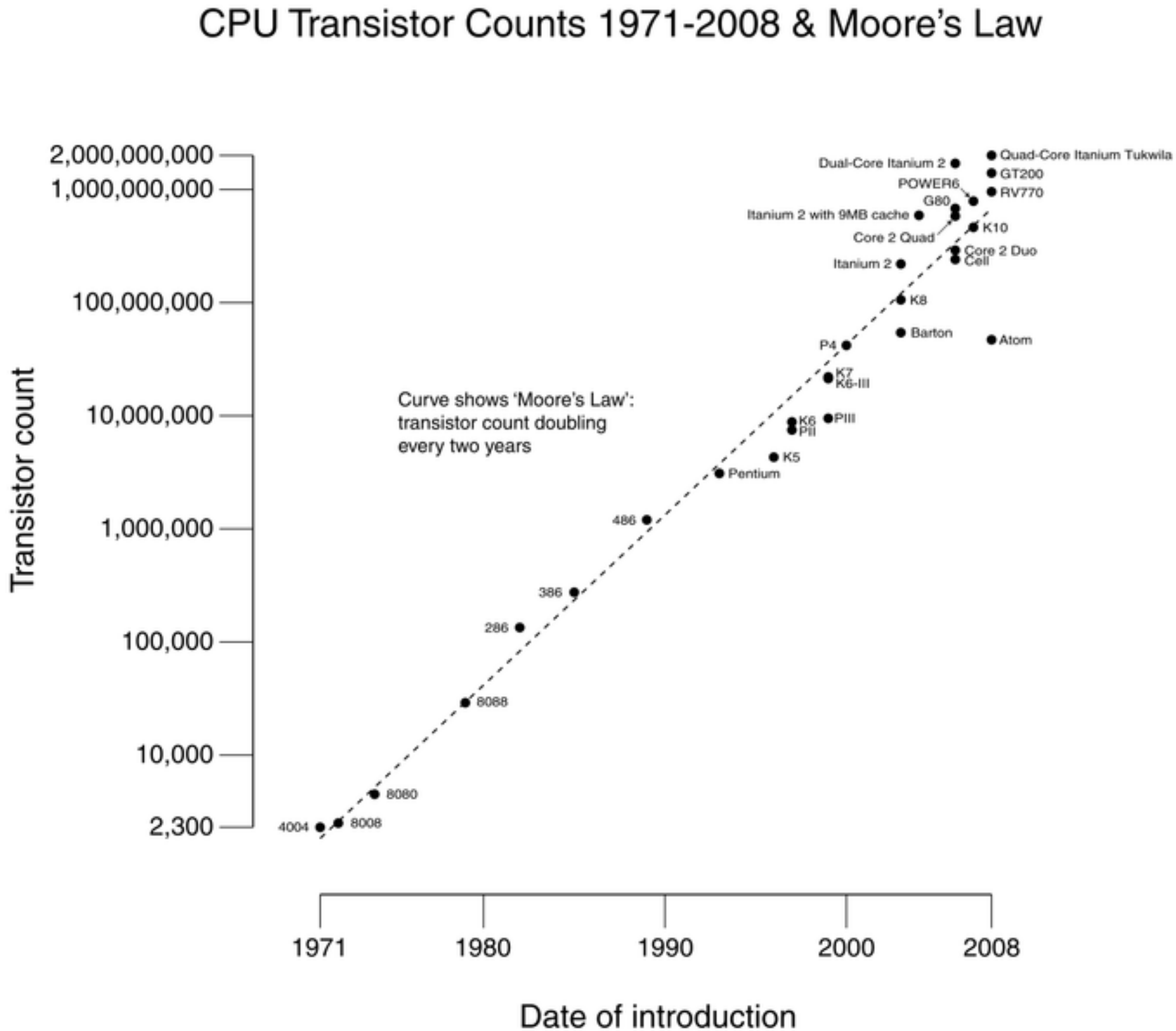
- NVIDIA. *The NVIDIA CUDA Programming Guide*.
- NVIDIA. CUDA Reference Manual.
- CUDA by Example An Introduction to General-Purpose GPU programming. 2010
- Kirk & Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. 2012.

Moore’s Law (paraphrased)

“The number of transistors on an integrated circuit doubles every two years.”

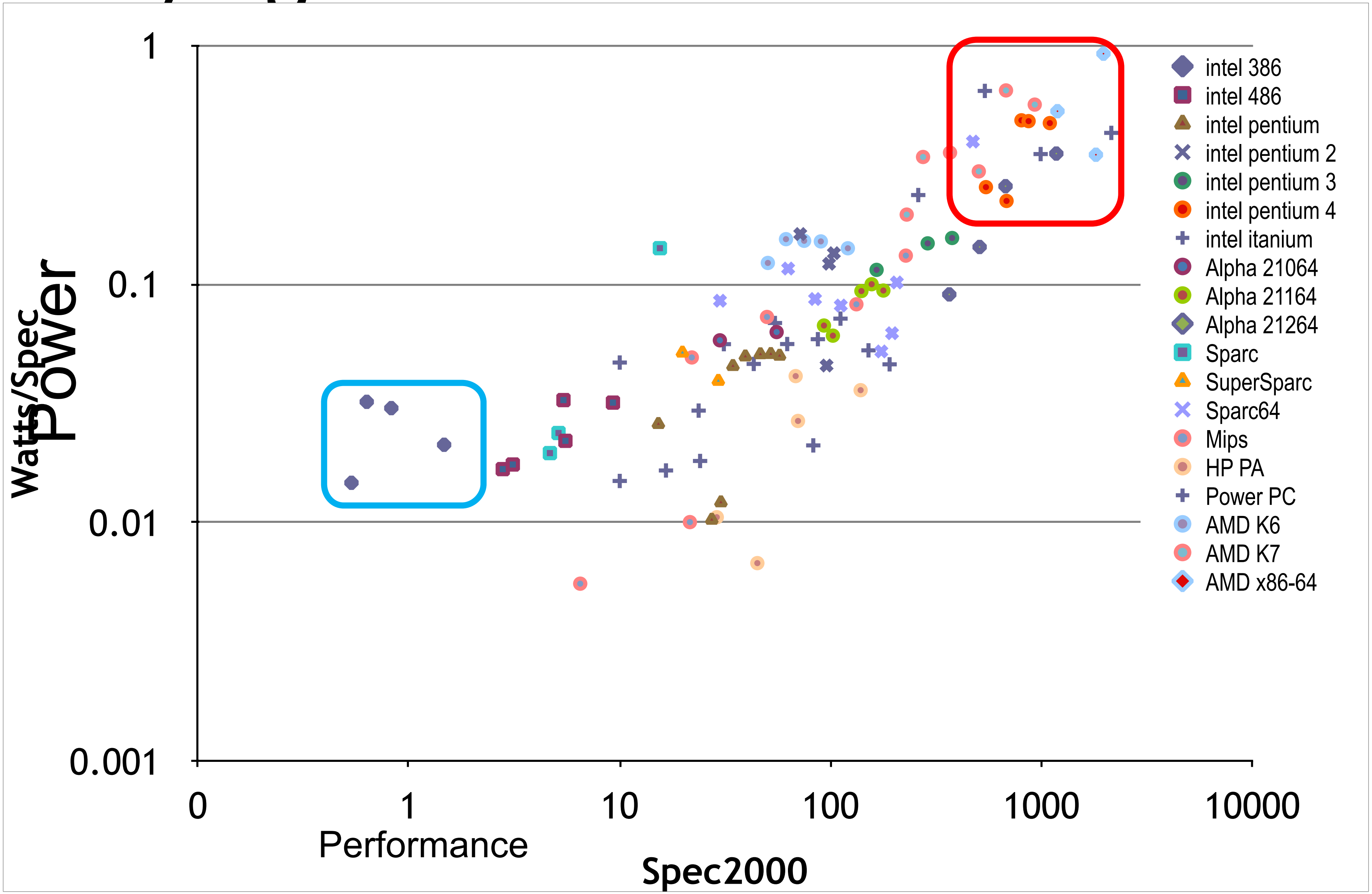
– Gordon E. Moore

Moore's Law (Visualized)



edit: Wikipedia

Buying Performance with Power



(courtesy Mark Horowitz and Kevin Skadron)

Serial Performance Scaling is Over

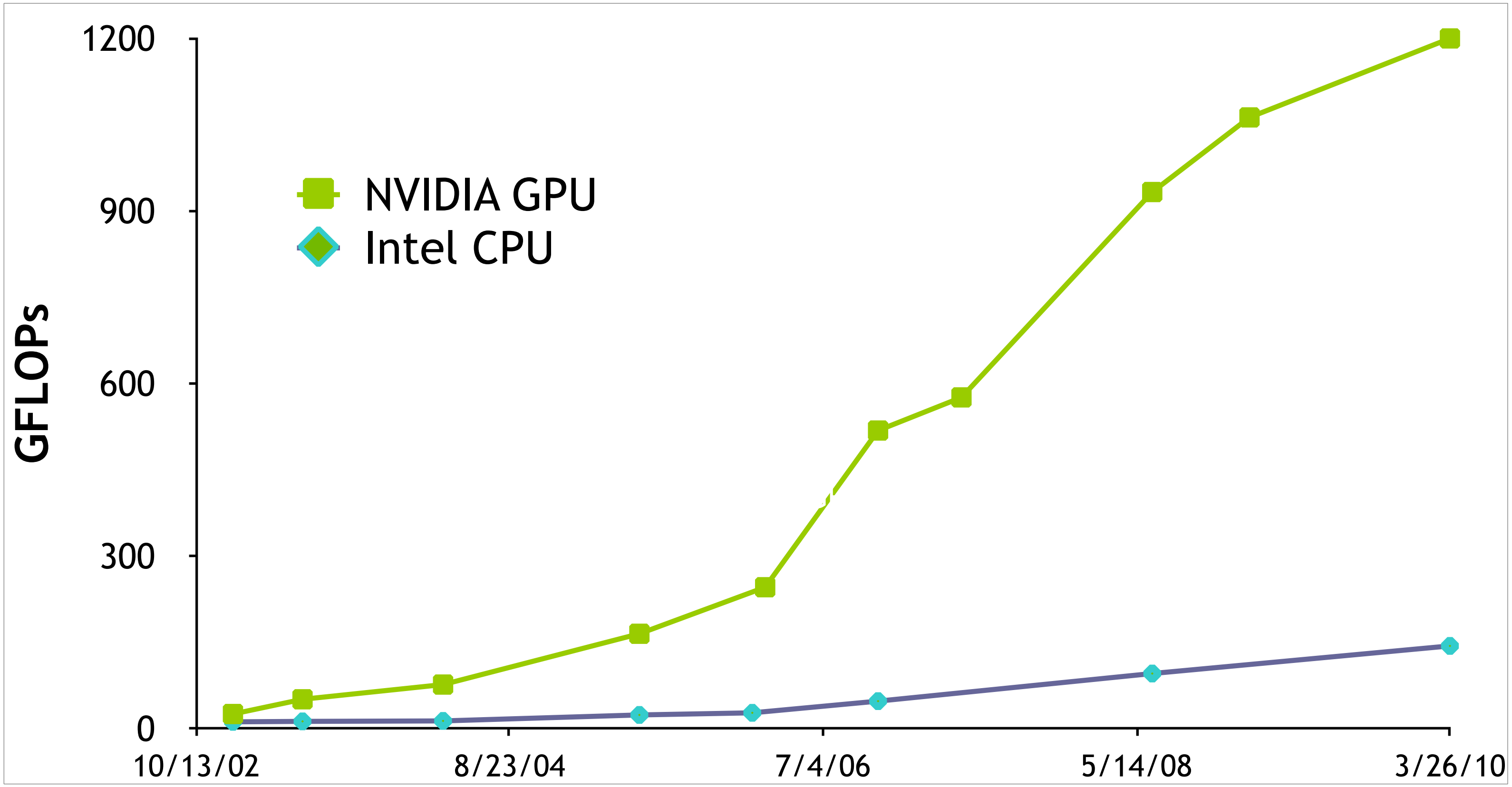
- Cannot continue to scale processor frequencies
  - no 10 GHz chips
- Cannot continue to increase power consumption
  - can’t melt chip
- Can continue to increase transistor density
  - as per Moore’s Law

How to Use Transistors?

- Instruction-level parallelism
  - out-of-order execution, speculation, …
  - vanishing opportunities in power-constrained world
- Data-level parallelism
  - vector units, SIMD execution, …
  - increasing … SSE, AVX, Cell SPE, Clearspeed, GPU
- Thread-level parallelism
  - increasing … multithreading, multicore, manycore
  - Intel Core2, AMD Phenom, Sun Niagara, STI Cell, NVIDIA Fermi, …

Why Massively Parallel Processing?

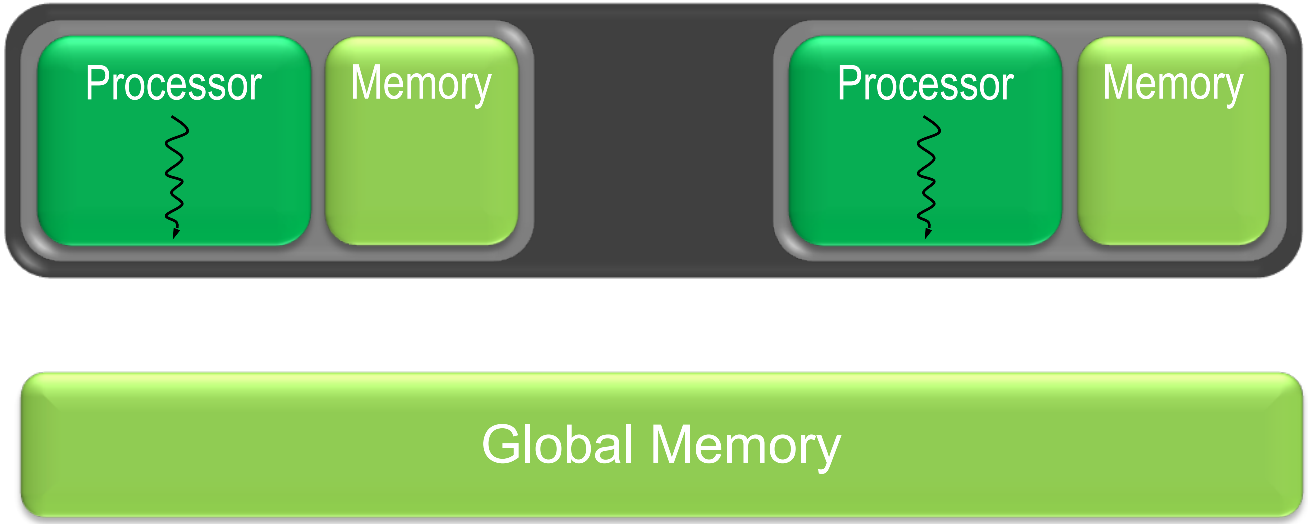
- A quiet revolution and potential build-up
  - Computation: TFLOPs vs. 100 GFLOPs



The "New" Moore's Law

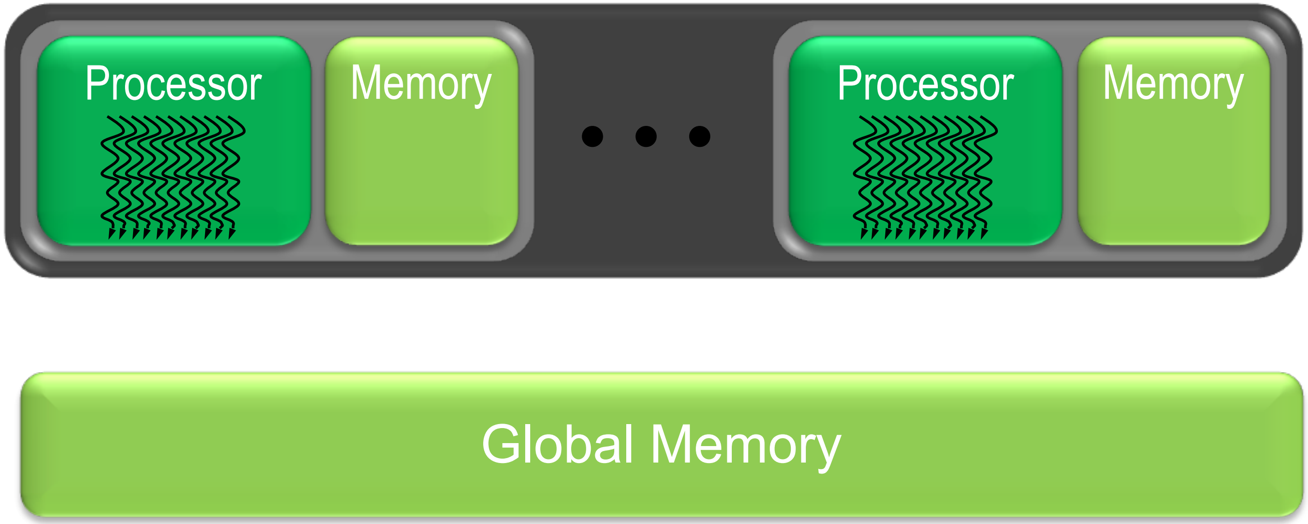
- Computers no longer get faster, just wider
- You *must* re-think your algorithms to be parallel !
- Data-parallel computing is most scalable solution
  - Otherwise: refactor code for 2 cores
  - You will always have more data than cores – build the computation around the data

Generic Multicore Chip



- Handful of processors each supporting ~1 hardware thread
- On-chip memory near processors (cache, RAM, or both)
- Shared global memory space (external DRAM)

Generic Manycore Chip



- Many processors each supporting many hardware threads
- On-chip memory near processors (cache, RAM, or both)
- Shared global memory space (external DRAM)

Enter the GPU

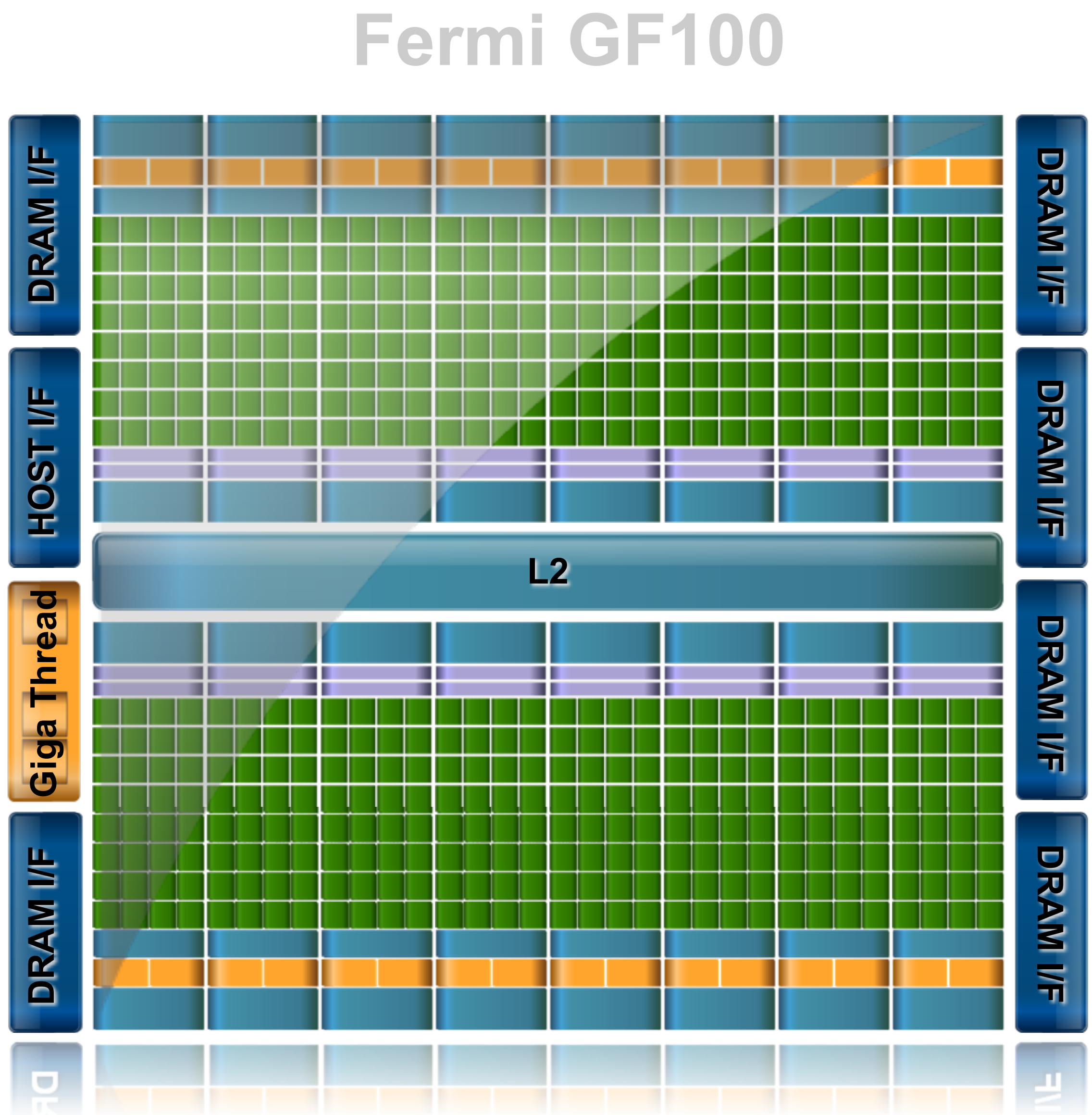
- Massive economies of scale

- Massively parallel

Why is this different from a CPU?

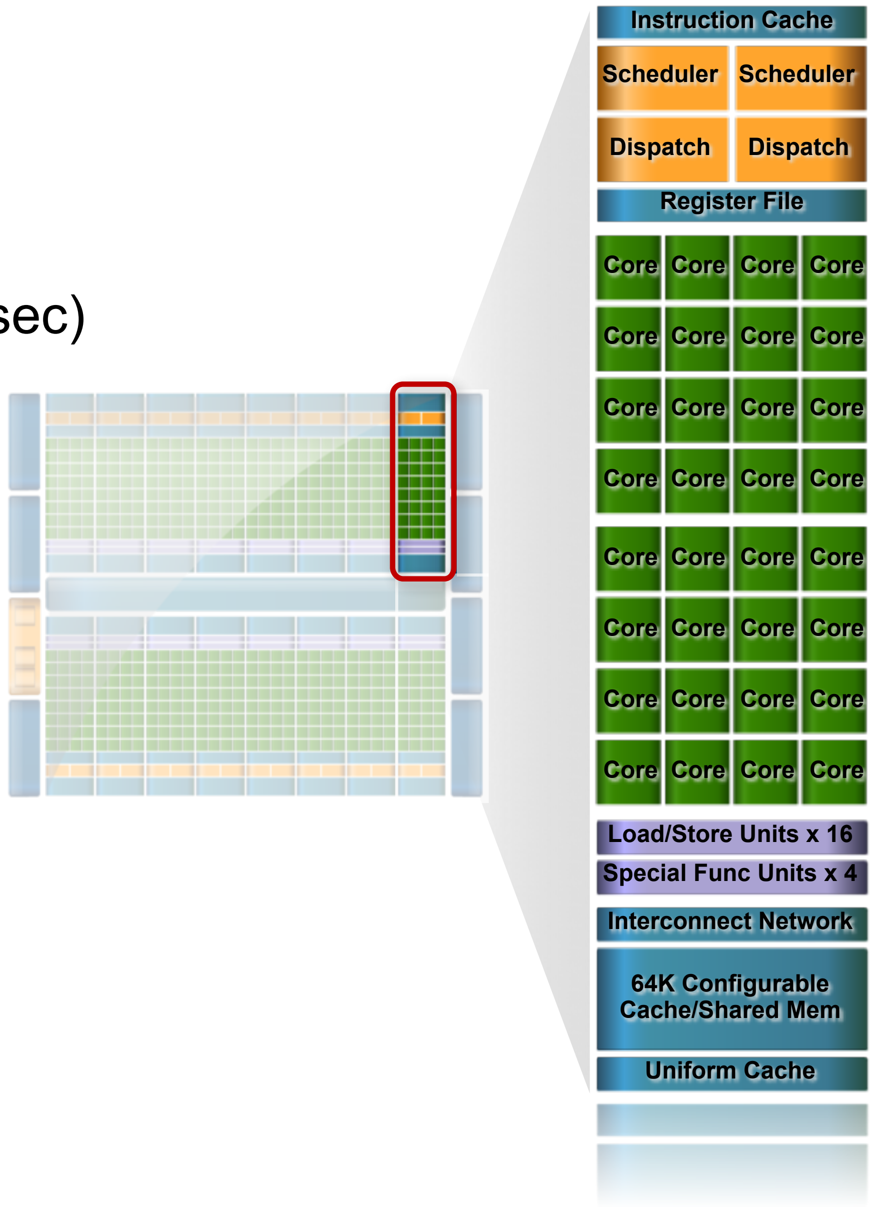
- Different goals produce different designs
  - GPU assumes work load is highly parallel
  - CPU must be good at everything, parallel or not
- CPU: minimize latency experienced by 1 thread
  - big on-chip caches
  - sophisticated control logic
- GPU: maximize throughput of all threads
  - # threads in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

NVIDIA GPU Architecture



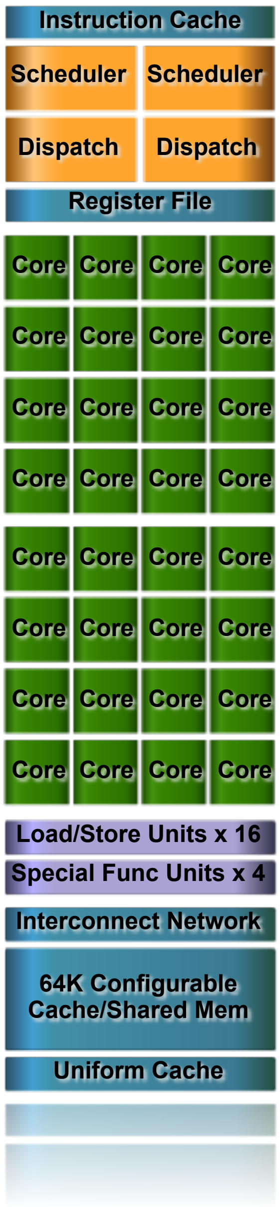
SM Multiprocessor

- 32 CUDA Cores per SM (512 total)
- Direct load/store to memory
  - Usual linear sequence of bytes
  - High bandwidth (Hundreds GB/sec)
- 64KB of fast, on-chip RAM
  - Software or hardware-managed
  - Shared amongst CUDA cores
  - Enables thread communication



Key Architectural Ideas

- SIMT (Single Instruction Multiple Thread) execution
  - threads run in groups of 32 called warps
  - threads in a warp share instruction unit (IU)
  - HW automatically handles divergence
- Hardware multithreading
  - 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



NVIDIA GPU Architecture

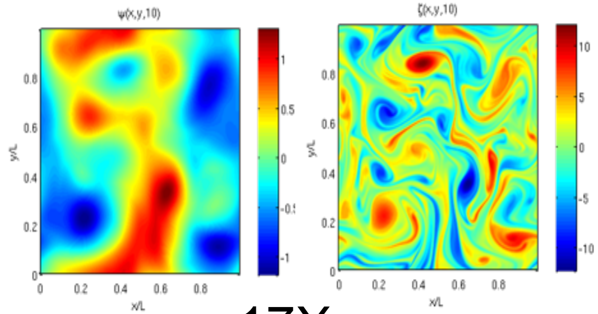
Kepler

CUDA

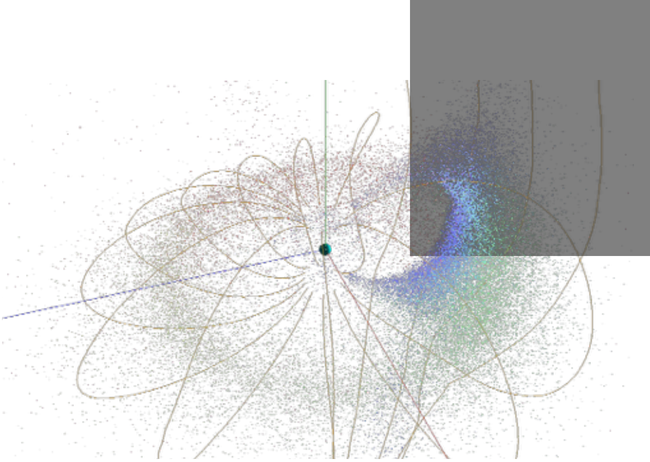
- Scalable parallel programming model
- Minimal extensions to familiar C/C++ environment
- Heterogeneous serial-parallel computing

Motivation

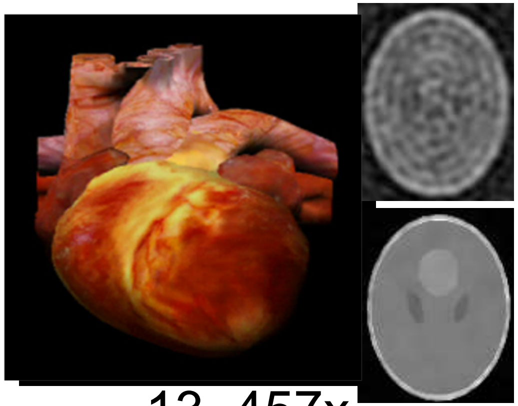
45X



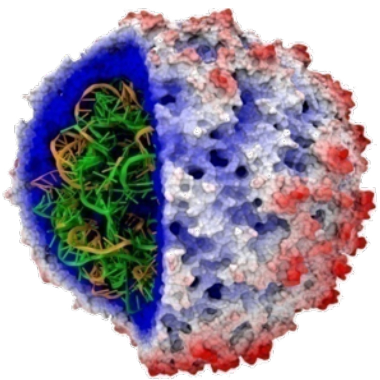
17X



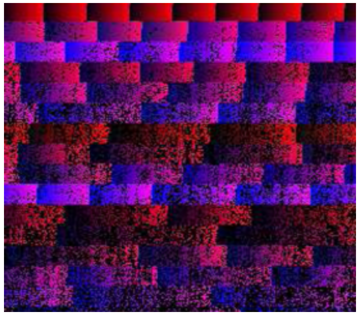
100X



13–457x



110-240X

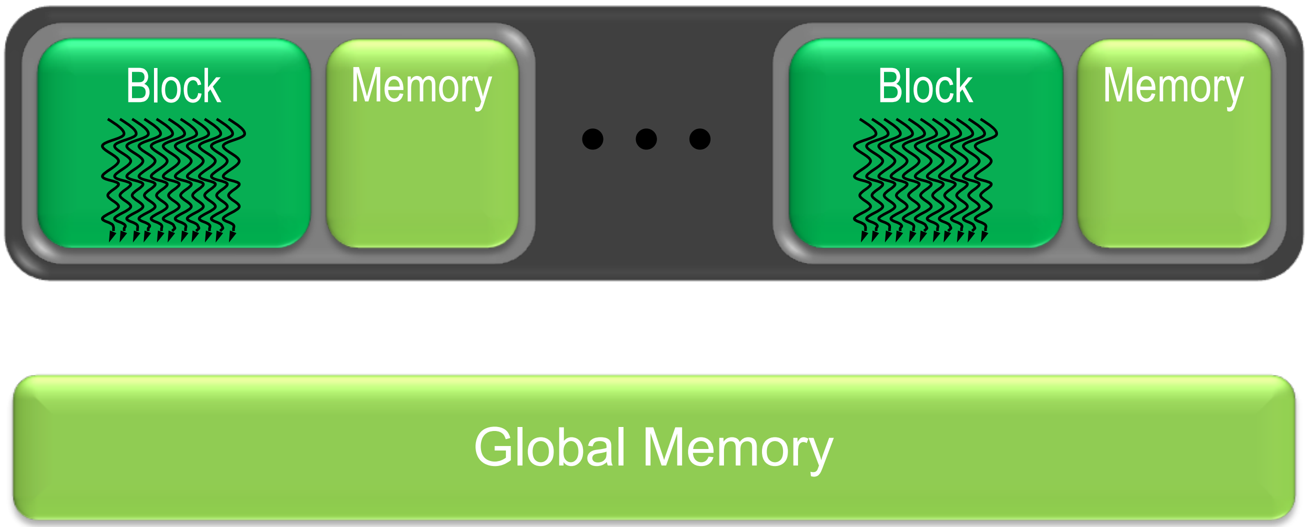


35X

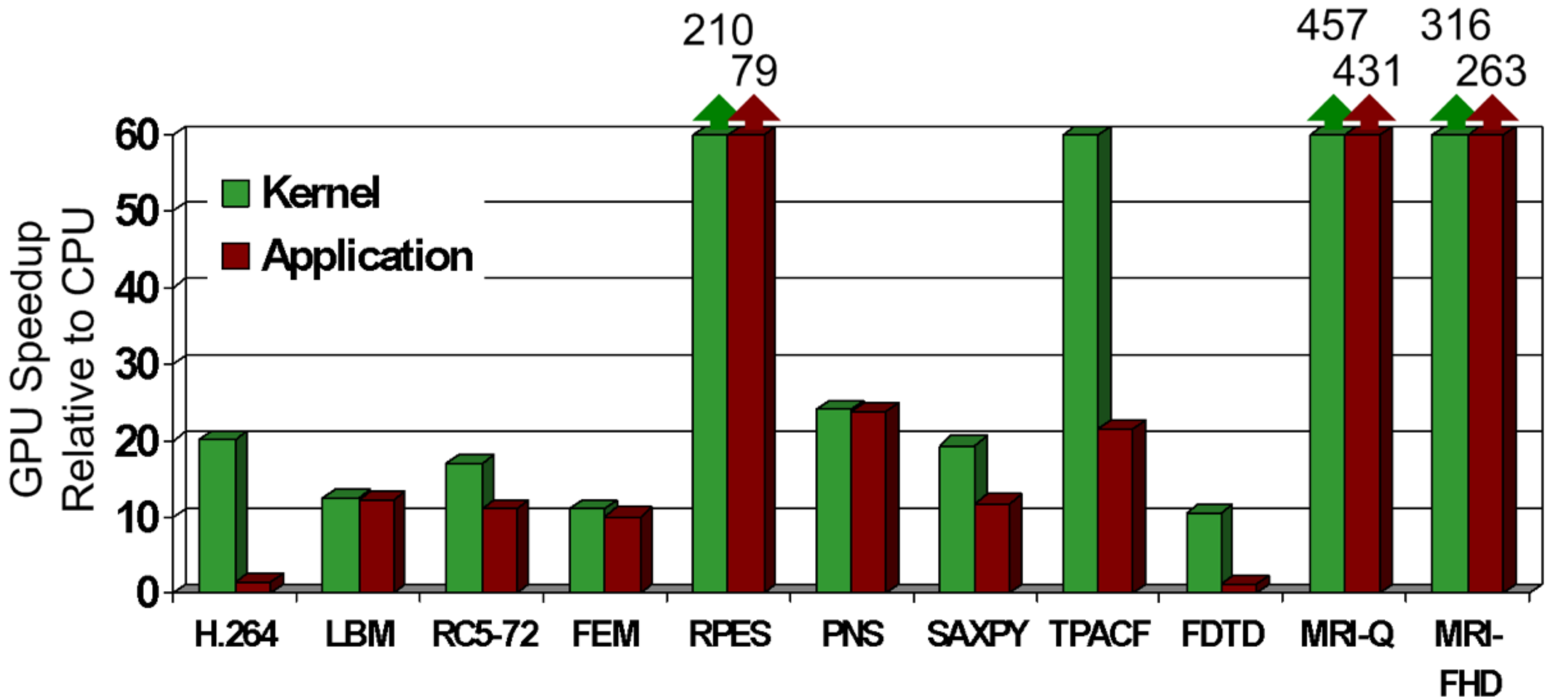
CUDA: Scalable parallel programming

- Augment C/C++ with minimalist abstractions
  - let programmers focus on parallel algorithms
  - *not* mechanics of a parallel programming language
- Provide straightforward mapping onto hardware
  - good fit to GPU architecture
  - maps well to multi-core CPUs too
- Scale to 100s of cores & 10,000s of parallel threads
  - GPU threads are lightweight — create / switch is free
  - GPU needs 1000s of threads for full utilization

CUDA Model of Parallelism



Speedup of Applications



- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in a kernel is typical, as long as the kernel can occupy enough parallel threads
- 25× to 400× speedup if the function's data requirements and control flow suit the GPU and the application is optimized

Final Thoughts

- Parallel hardware is here to stay
- GPUs are massively parallel manycore processors
  - easily available and fully programmable
- Parallelism & scalability are crucial for success
- This presents many important research challenges
  - not to speak of the educational challenges

Course Equipment

- Your own PCs with a CUDA-enabled GPU
- Design Center
  - NVIDIA GeForce GTX 570 boards

    Fermi Architecture GPUs
  - Little Fe cluster
  - Alcatraz1 and 2 with

  Kepler

    •K20 comp cap 3.5

    •GTX 690 comp cap 3.0

Fermi

GTX 480

Course Equipment

- Design Center
  - Remote Login Instructions:

  http://www.scu.edu/engineering/centers/scudc/Terminal-Services.cfm

  -Using putty

  http://www.cse.scu.edu/~sfigueira/10/remote.html

  (linux.scudc.scu.edu)

  -Login to alcatraz 2:

Your Own Computer

- Nvidia CUDA getting started guide for MS Windows
  - CUDA enabled GPU
  - MS Windows XP, vista , or 7 (better 7)
  - Device driver
  - Cuda Software (nvcc compiler)
  - MS Visual Studio Express edition (free for students) 2005 or leter (better 2010)

Your Own Computer

- Verify you have a CUDA enabled GPU
- List of CUDA enabled GPUs:

  http://www.nvidia.com/object/cuda_gpus.html

Your Own Computer

- Download CUDA software:
  - Driver.
  - CUDA toolkit:

    Tools needed to compile and build CUDA applications
  - SDK

    Sample projects that have all necessary project configuration and build in files to perform one click builds using MS Visual Studio

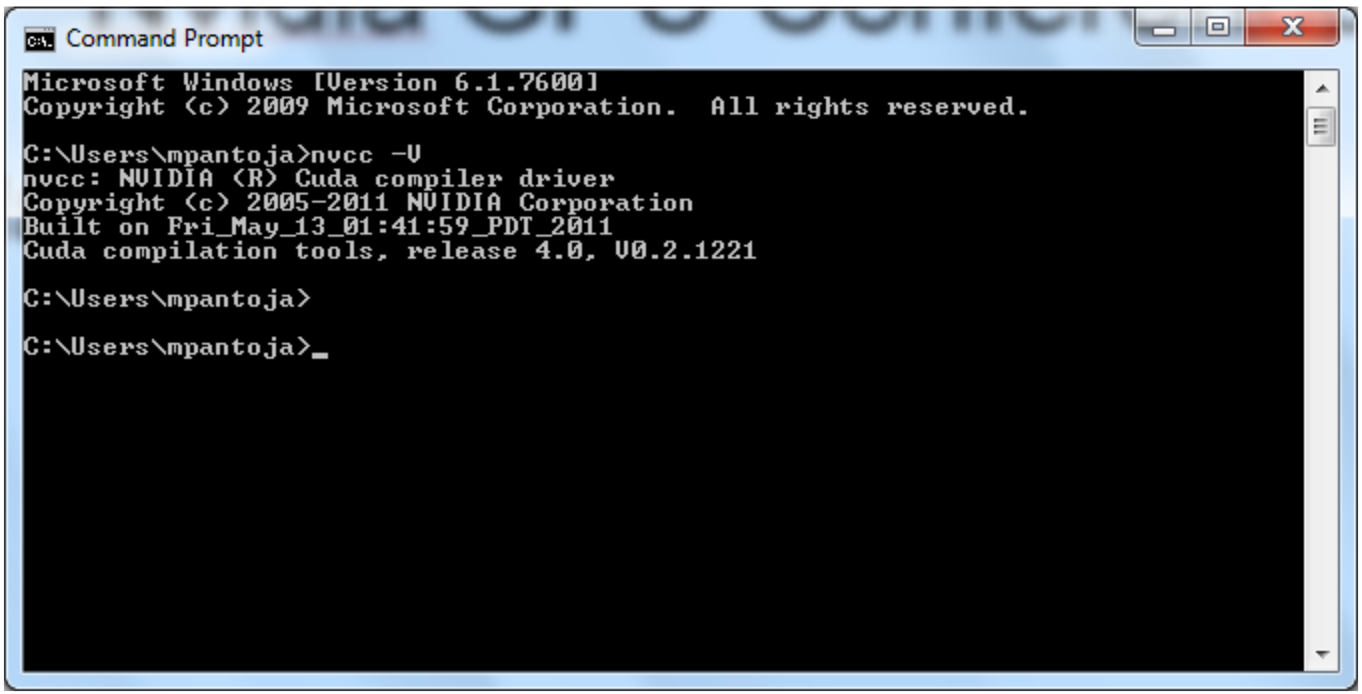
    http://developer.nvidia.com/cuda-toolkit-41

Your Own Computer

- Verify the installation

Command prompt window:

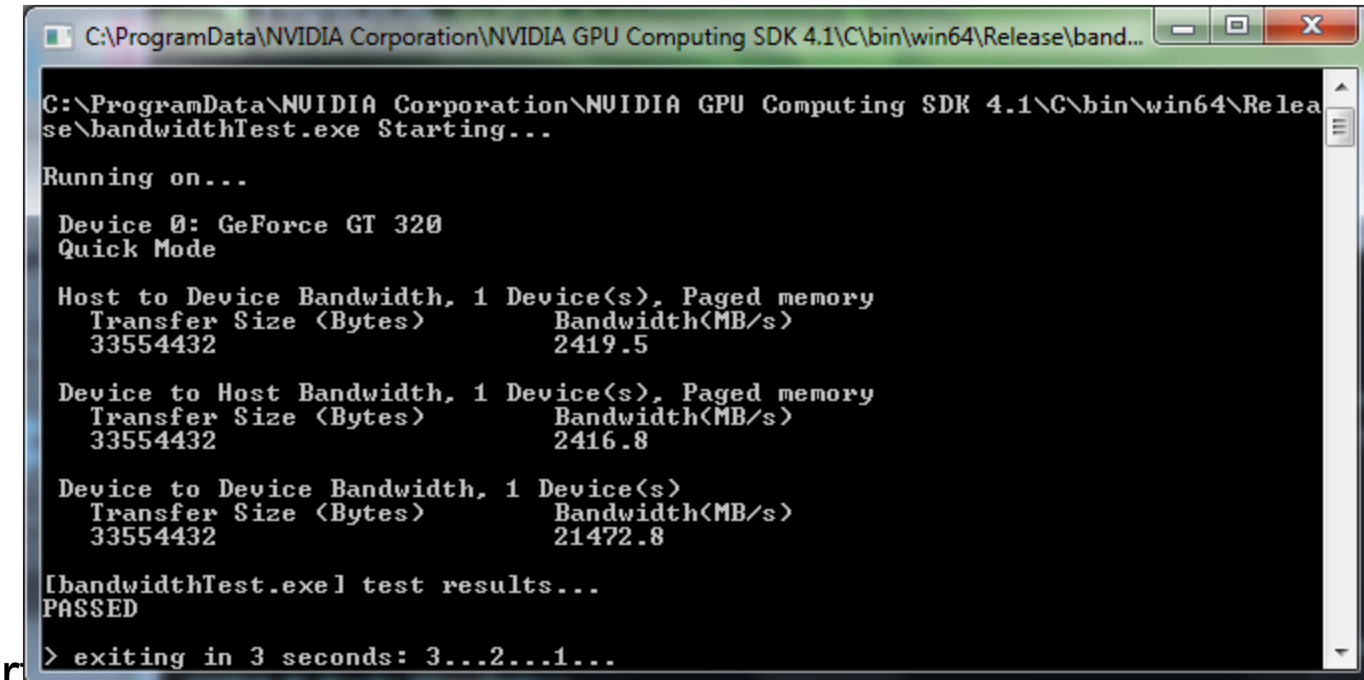
Start->all programs->accessories->command prompt



Your Own Computer

- Verify the installation

Bandwidthtest: sdk\C\src\bandwidthtest



Run par … ect

Cuda and MSVC++ 2010

1. Open MS VC++ 2010.

2. Create new project:

File->new->project, select empty project

3. To get cuda syntax highlighting:

   tools->options->test editor->file extensions

   type *.cu in input box and click on apply

4. Right click on project, select build Customizations, click on the cuda 4.1 select box

5. Right click on project->properties->configuration properties->linker->input

   add cudart.lib to the list of additional dependencies

6. Add new item to project->C++ file but name the file with *.cu extension

Your Own Computer

Compile new CUDA Project

sdk\C\src\bandwidthtest

Solution should be placed on debugging folder