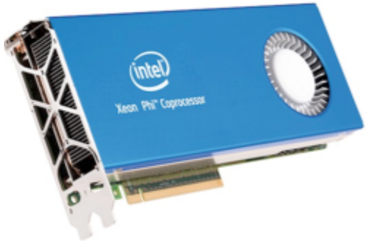
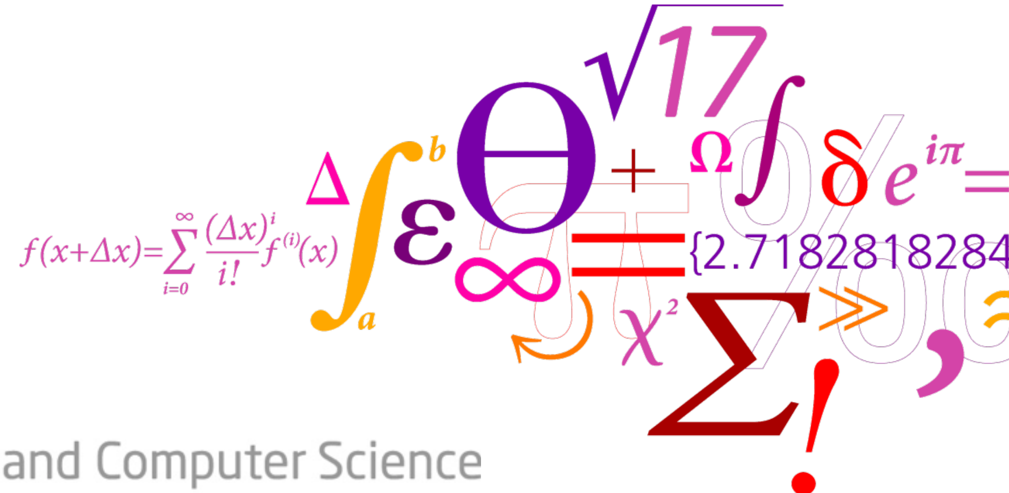


Introduction to GPU computing

Computional Tools for Data Science (Fall 2014)

Hans Henrik Brandenborg Sørensen
DTU Computing Center
<hhbs@dtu.dk>



GPUlab
DTU Informatics

DTU Compute
Department of Applied Mathematics and Computer Science

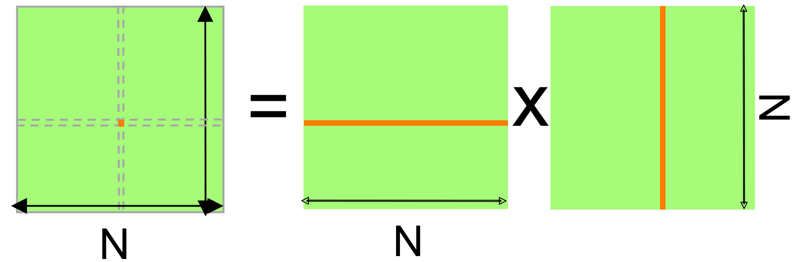
Outline

1. Some basics on computing
2. GPUs / CUDA programming model
3. Numbapro
4. Exercise 1+2+3

- This is a very brief introduction to GPU computing, which assumes that the student has studied some of the details beforehand.

Computing basics

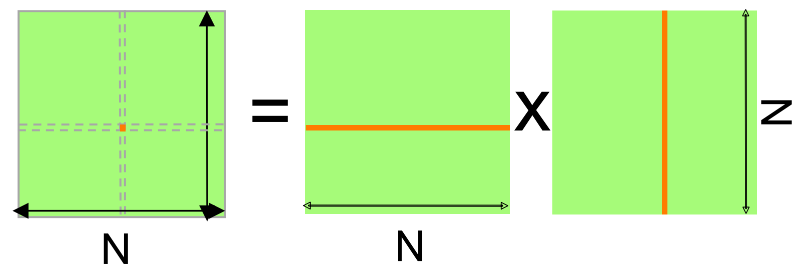
- Running time of scientific applications:
  - # flops * time per flop
  - # bytes moved / bandwidth
  - # messages * latency



flops = O(N^3), # bytes = O(N^2)

Computing basics

- Running time of scientific applications:
  - # flops * time per flop
  - # bytes moved / bandwidth
  - # messages * latency

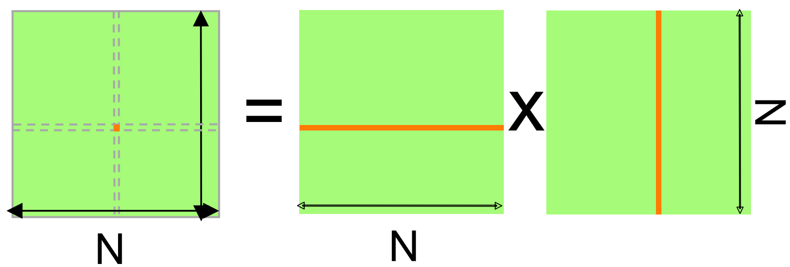


flops = O(N^3), # bytes = O(N^2)

- What is a flop?
  - In theory: +, -, *, /, √
  - In practice: *+ = 1 cycle, / = ~8 cycles, √ = ~10 cycles,..

Computing basics

- Running time of scientific applications:
  - # flops * time per flop
  - # bytes moved / bandwidth
  - # messages * latency



flops = O(N^3), # bytes = O(N^2)

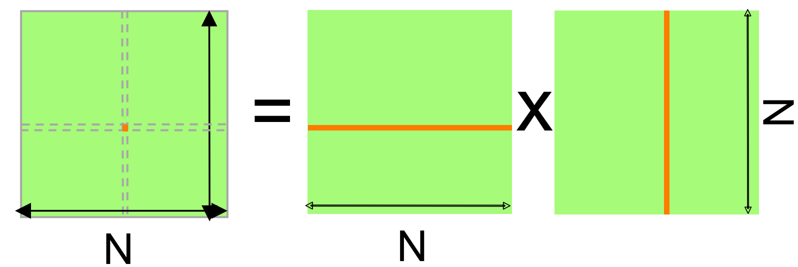
- What is a flop?
  - In theory: +, -, *, /, √
  - In practice: *+ = 1 cycle, / = ~8 cycles, √ = ~10 cycles,..
- What is a memory access?
  - Pages of 4K: disk drive ➔ mem (> ms)
  - Cache line of 64B: mem ➔ L3 ➔ L2 ➔ L1 (12-100 cycles)
  - Word of 8 bit – 512 bit: L1 ➔ Register (4 cycles)

Computing basics

- Running time of scientific applications:
  - # f
  - # bytes moved / bandwidth
  - # messages / latency

➢ Compute bound

➢ Memory bound



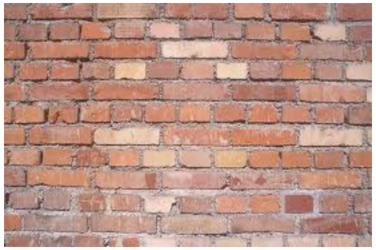
flops = O(N^3), # bytes = O(N^2)

- What is a flop?
  - In theory: +, -, *, /, √
  - In practice: *+ = 1 cycle, / = ~8 cycles, √ = ~10 cycles,..
- What is a memory access?
  - Pages of 4K: disk drive ➔ mem (> ms)
  - Cache line of 64B: mem ➔ L3 ➔ L2 ➔ L1 (12-100 cycles)
  - Word of 8 bit – 512 bit: L1 ➔ Register (4 cycles)

Computing basics

- Peak performance
  - doubles every ~2 years…
- Maximum memory bandwidth
  - doubles every ~3-4 years…
- Anyone notice a problem with these rates?
- "Flops-to-spare"
  - Determining factor in application performance is <u>likely to be memory access patterns rather than flop count</u>
  - Most important tuning techniques are related to reducing memory movement (flops are well "hidden")

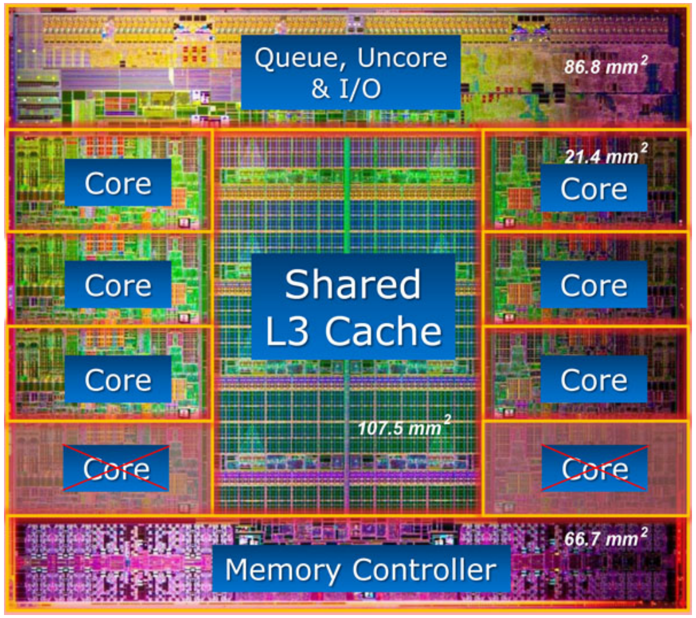
"The memory wall"

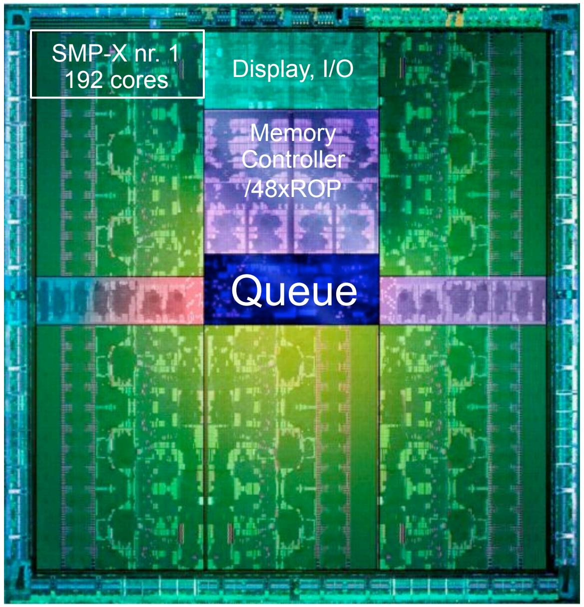


Data sciences (big data)

- We are typically inherently memory bound!

CPUs and GPUs (rough comp.)





Intel Core i7 3960X

129.6 Gflop dp peak performance

6 cores

15MB L3 cache (25% of die).

51.2 GB/s bandwidth

130 watt

NVIDIA Kepler K20X

1.31 Tflop dp peak performance

2680 cores (FP32), 896 cores (FP64)

1536KB L2 cache (2% of die)

250 GB/s bandwidth

233 watt

General purpose CPU

- Usual tasks of the CPU:
  - To run desktop applications
    - Lightly threaded
    - Lots of branches
    - Lots of (indirect) memory accesses

	`vim`	`ls`
Conditional branches	13.6%	12.5%
Memory accesses	45.7%	45.7%
Vector instructions	1.1%	0.2%

  - Modern branch predictors > 90% accuracy
- CPUs are general purpose by construction
- HPC: Can use CPUs for any problem / code

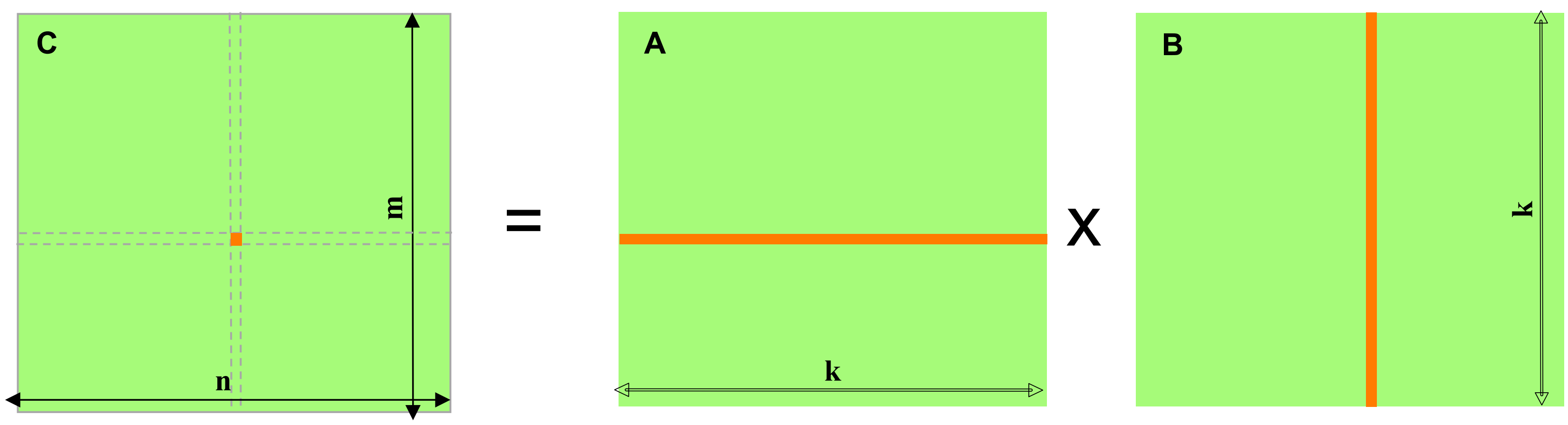
”General purpose” GPU

- Usual task of the GPU:
  - Compute a pixel => requires floating point operations
  - Compute many pixels => massively parallel flops
- It turns out to be a very efficient way to do computing – for particular problems
- Since ~2001 GPUs have been programmable

“General purpose” in the sense “not only pixels”

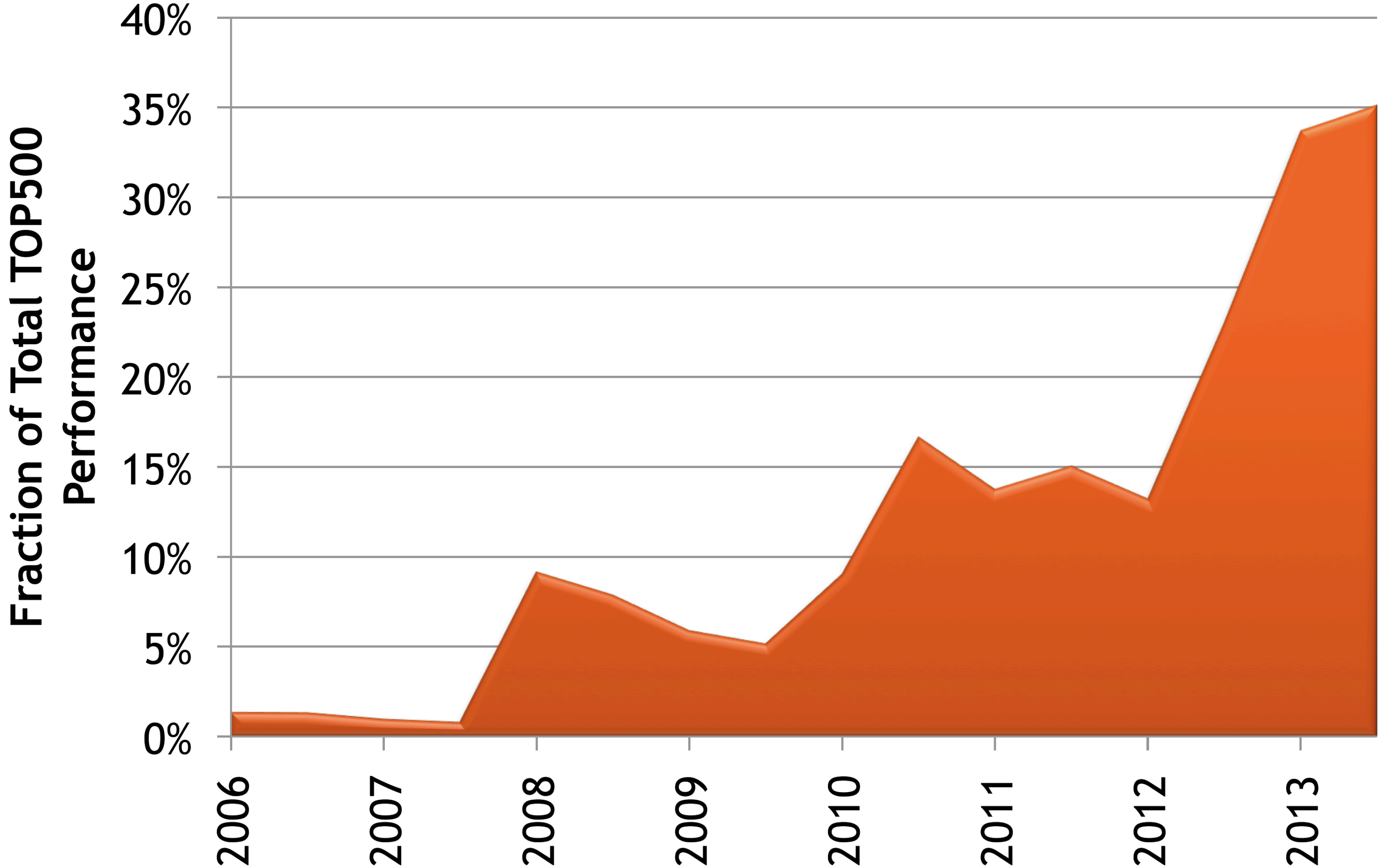
Which problems fit the GPU?

- Problems that use many flops but ”little data”
- Problems that have a high degree of parallelism
- One kind of problem is “matrix computations”:
  - E.g., matrix-vector and matrix-matrix multiplication



- So-to-say: “the heart” of scientific computing

GPU performance share Top 500



Source: Supercomputing 13, BOF, "Highlights of the 42nd Top500 List at SC'13", Denver, November, 2013

When to consider using GPUs?

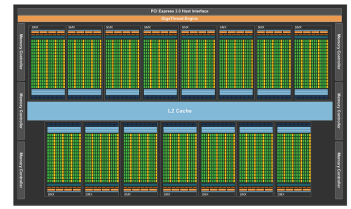
Commodity

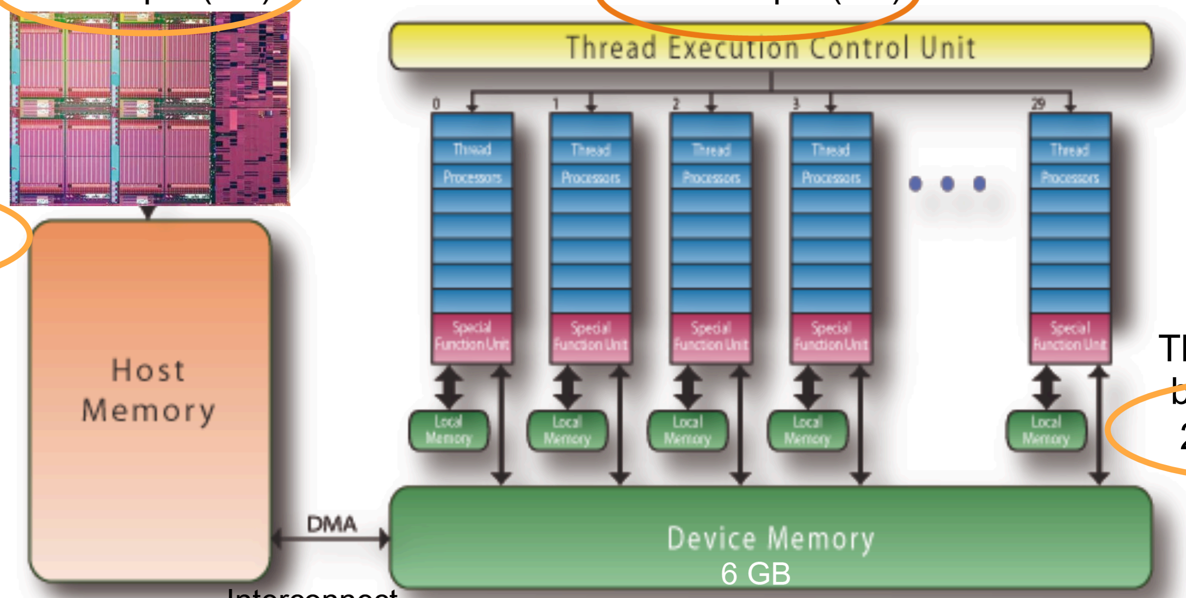
Intel Xeon
8 cores
3 GHz
8*4 ops/cycle
96 Gflop/s (DP)

2 x **Accelerator (GPU)**

Nvidia K20X “Kepler”
2688 “Cuda cores”
.732 GHz
2688*2/3 ops/cycle
1.31 Tflop/s (DP)

192 Cuda cores/SMX
2688 “Cuda cores”





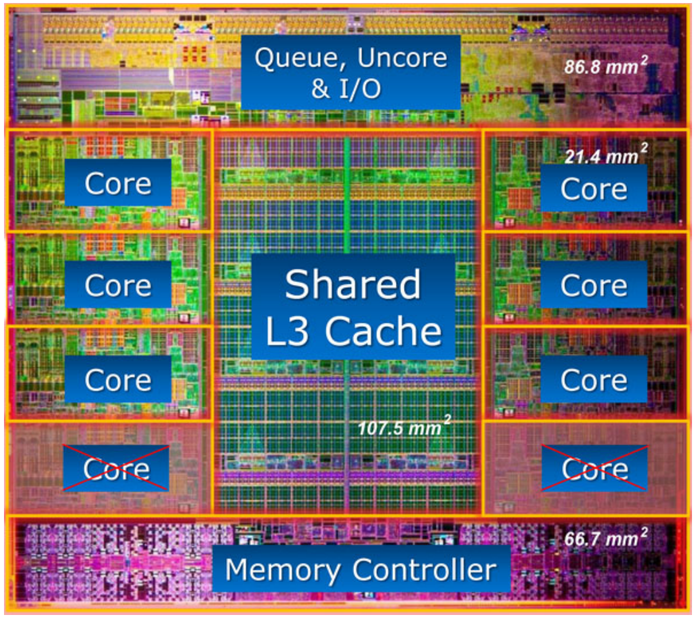
Theoretical bandwidth
32 Gb/s

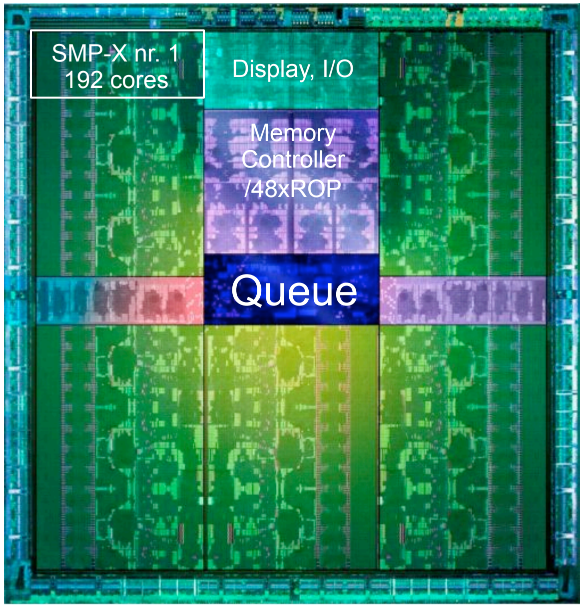
Theoretical bandwidth
250 Gb/s

Interconnect
PCI-X 16 lane
64 Gb/s (8 GB/s)
1 GW/s

- **Example:** Matmult - large N
  - Tcpu ≈ 1e-9*N^3/96 s (+ 1e-9*N^2/32 s)
  - Tgpu ≈ 1e-9*N^3/1310 s + 1e-9*N^2/6 s

Which speed-ups can you expect?





Intel Core i7 3960X	x10	**NVIDIA Kepler K20X**
129.6 Gflop dp peak performance		*1.31 Tflop dp peak performance*
6 cores	x112	2680 cores (FP32), 896 cores (FP64)
15MB L3 cache (25% of die).		1536KB L2 cache (2% of die)
51.2 GB/s bandwidth	x5	250 GB/s bandwidth
130 watt		233 watt

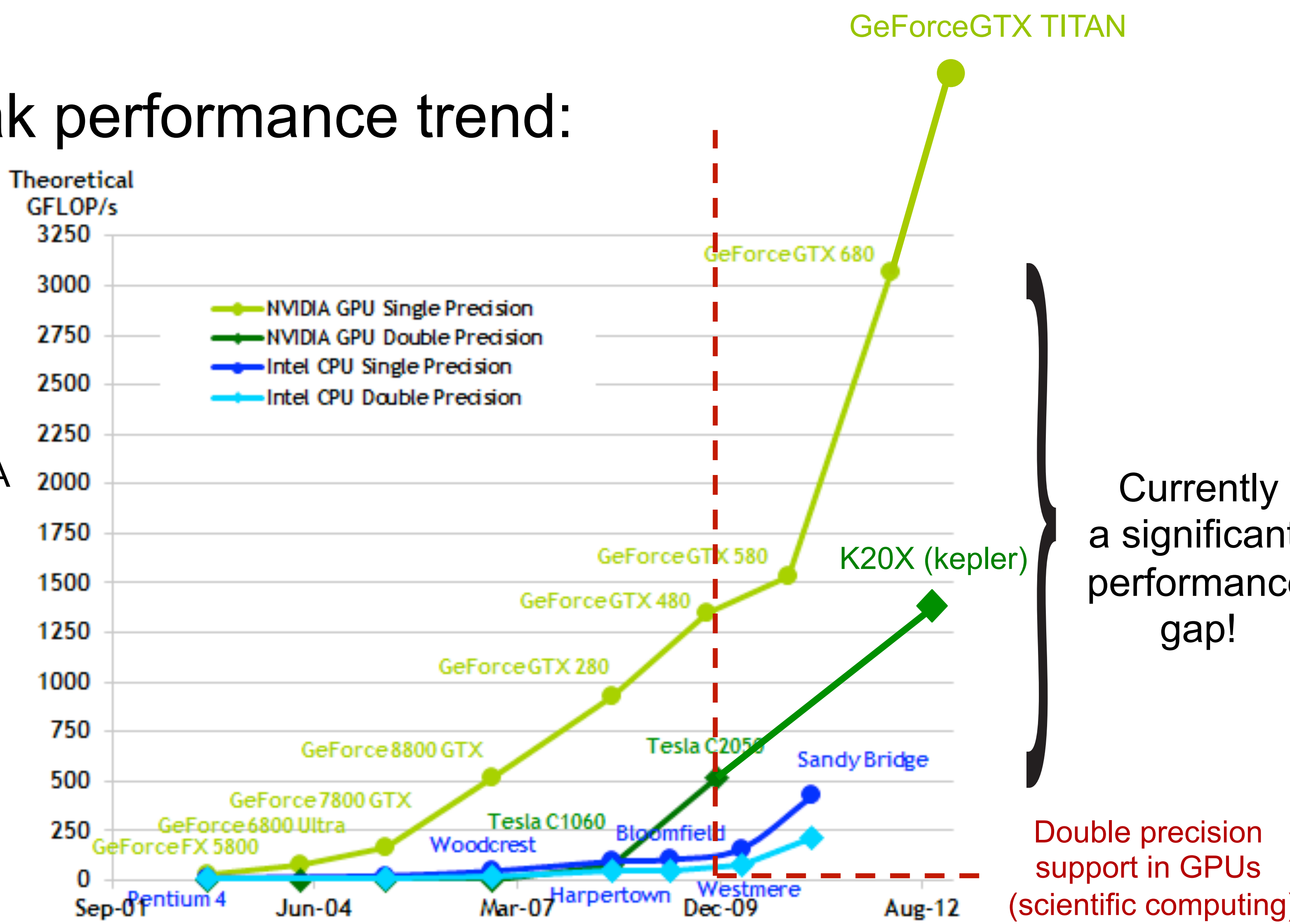
Remember Amdahl!

Trends for GPU algorithms

- Synchronization-reducing algorithms
  - Embarrassingly parallel techniques
- Communication-reducing algorithms
  - Use methods which have lower bound on communication
- Mixed precision methods
  - >3x speed of ops and 2x speed for data movement
- Autotuning
  - Today’s machines are too complicated, build “smarts” into software to adapt to the hardware

CPU vs. GPU trends

- Peak performance trend:

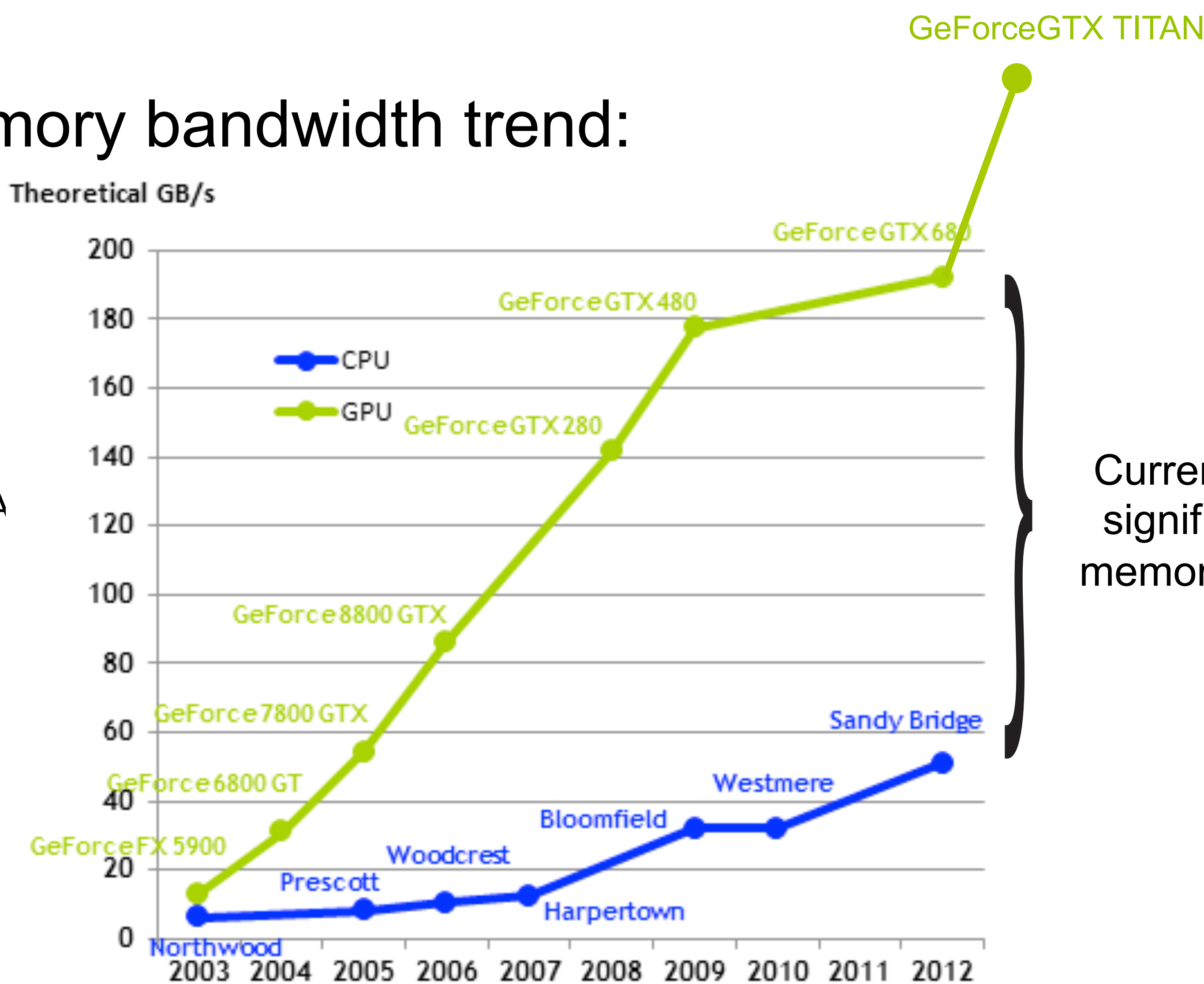


Source: Nvidia, "CUDA programming guide"

Currently a significant performance gap!

CPU vs. GPU trends

- Memory bandwidth trend:



Source: Nvidia, "CUDA programming guide"

Currently a significant memory gab!

CUDA terminology

Fine

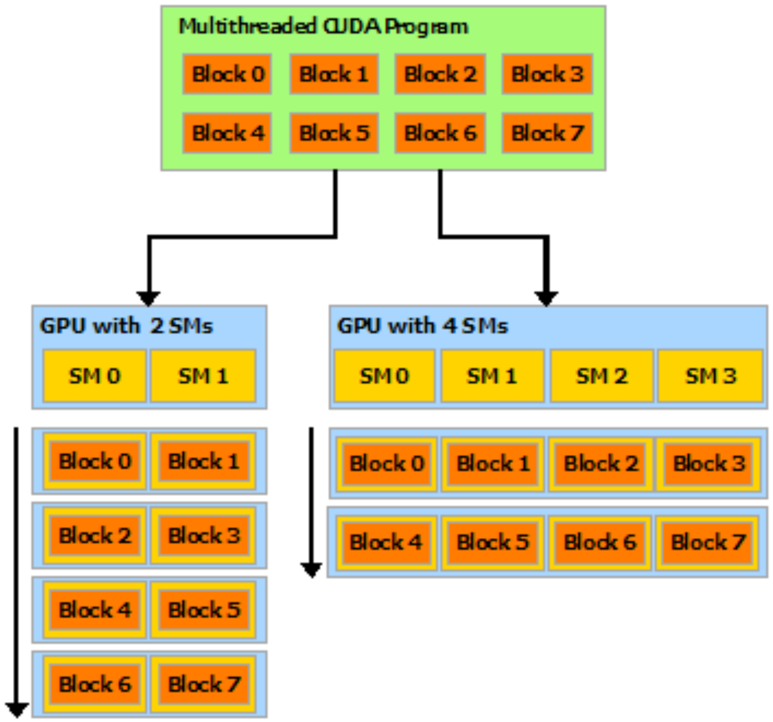
Granularity

Coarse

- Thread ("Unit of parallelism" in CUDA)
  - Concurrent code and associated state executed on the CUDA device in parallel with other threads. Independent control flow.
- Warp ("Unit of execution")
  - A group of threads in same block that are executed physically in parallel – currently 32 threads.
- Block ("Unit of resource assignment")
  - A virtual group of threads executed together that can cooperate and share data.
- Grid ("Task unit")
  - A virtual group of thread blocks that must all finish before the invoked kernel is completed.

Scalable execution model

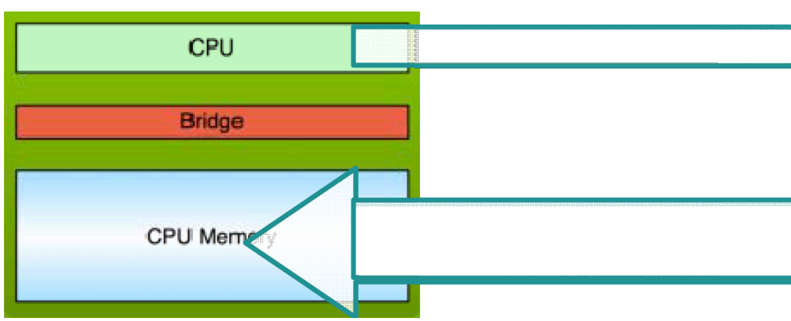
- Why are blocks scheduled in no particular order?

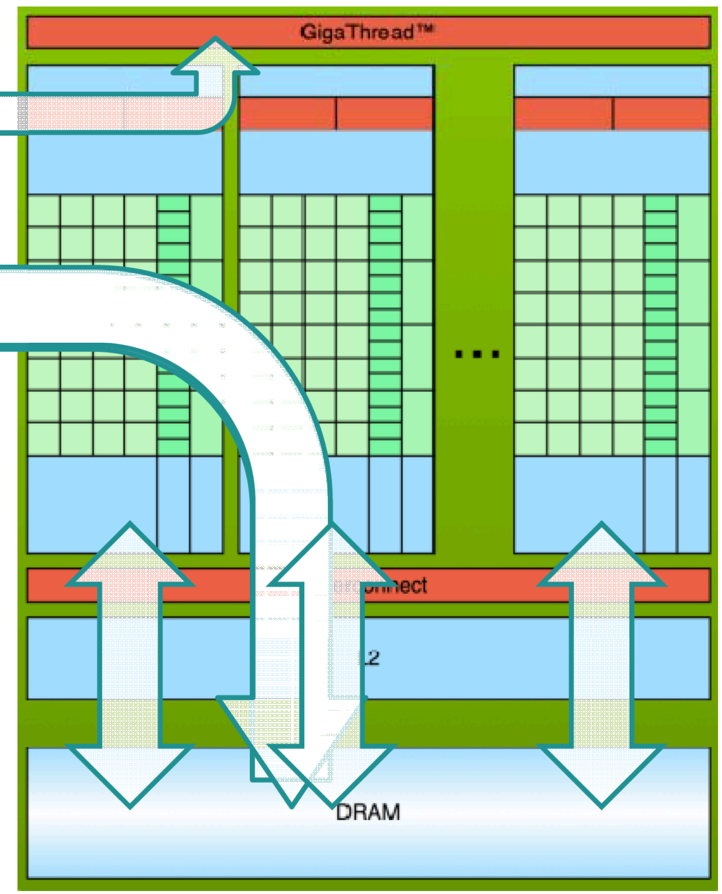


A GPU with more SMs will automatically execute the program in less time than a GPU with fewer SMs.

- Independence among blocks provides the basis for scalability across present and future GPUs!

CUDA processing flow



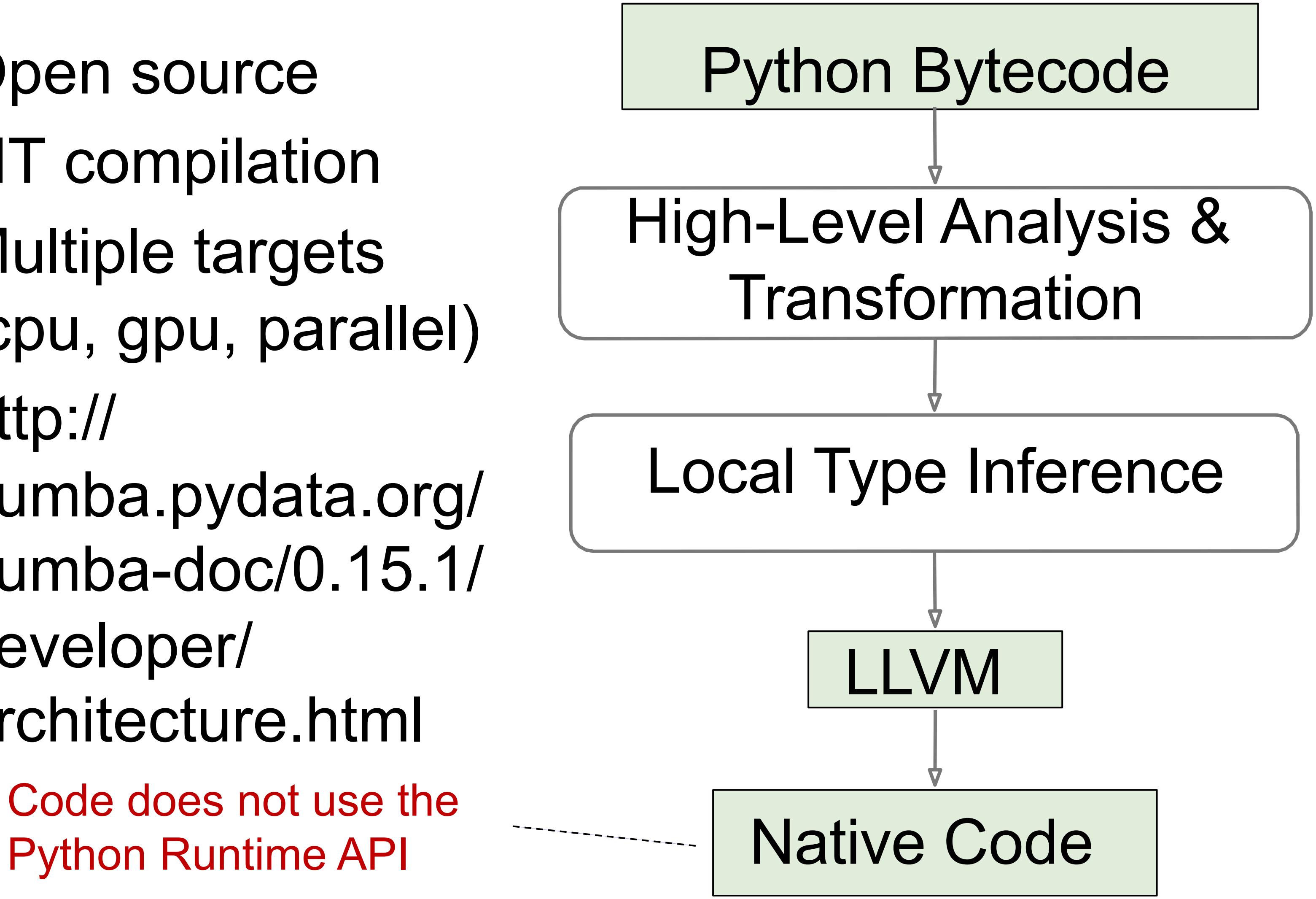


- Typical steps
  1. Copy input data from CPU memory to GPU memory
  2. Invoke GPU kernel and execute, caching data on chip for performance
  3. Copy results from GPU memory to CPU memory

Source: Timothy Lanfear, Nvidia, 2009

Numbapro compiler

- Open source
- JIT compilation
- Multiple targets (cpu, gpu, parallel)
- http://numba.pydata.org/numba-doc/0.15.1/developer/architecture.html



CUDA-Python

- Kernel definition
  - ```
    @cuda.jit('void(float64[:])')
    def kernelname(array):
    ```
- Kernel invocation
  - Syntax: `kernelname[griddim, blockdim](…)`
  - Launches a fixed number of threads
  - All threads execute the same code
  - Each thread has an ID
    - To decide what data to read or write, to decide control flow
- Commonly thousands of threads are launched
  - Rule-of-thumb 25.000 – 100.000 threads are needed.
    ```

Hints for exercises

- Numbapro
  - `from numbapro import *`
- JIT for both CPU and GPU
  - `@jit('float64(float64, int64, int64)', target="cpu")`
  - `@cuda.jit('void(float64, int64, float64[:])')`
  - `@cuda.jit('float64(float64)', device=True, inline=True)`
- CUDA device context and name
  - ```
    gpu = numba.cuda.get_current_device()
    print("Device: %s\n" % gpu.name)
    ```
    ```

Hints for exercises

- Global thread ID inside kernel
  - `idx = cuda.grid(1)`
  - `idx, idy = cuda.grid(2)`
- Block and grid size inside kernel
  - `cuda.blockDim.x` (and .y, .z)
  - `cuda.gridDim.x` (and .y)
    - http://numba.pydata.org/numba-doc/0.15.1/CUDAJit.html#thread-identity-by-cuda-intrinsics
- Rule-of-thumb numbers
  - `blockdim = 256`
  - `griddim = 1024`

Hints for exercises

- Transfer data CPU←→GPU
  - `A_d = cuda.to_device(A)`
  - `x_d.copy_to_device(y_d)`
  - `x_d.copy_to_host(x)`
- Wait for kernels
  - `cuda.synchronize()`
- CUDA libraries
  - `import numbapro.cudalib.cublas as cublas`
  - `blas = cublas.Blas()`
  - `blas.gemv('T', n, n, 1.0, A, x, 0.0, y)`

Advertisement

GPU-LABoratory



Research and education in Graphics Processing Units in Denmark

Established in August 2010 and is a unique national competence center and hardware laboratory at DTU Informatics.

- Development of efficient algorithms
- High-performance scientific computing
- Performance profiling and prediction
- Software development
- Education

GPUlab
DTU Informatics

http://gpulab.imm.dtu.dk

NVIDIA CUDA RESEARCH CENTER

NVIDIA CUDA TEACHING CENTER

Advertisement

PhD stud. DTU Compute

02614 Teaching assistent

DANSIS Graduate Prize of 2013

Some completed and ongoing GPULab projects – Jan 2014:

MSc: Max la Cour Christensen, M. and Eskildsen, K. L. *Nonlinear Multigrid for Efficient Reservoir Simulation*. 2012.

BSc: Mieritz, Andreas. *GPU-Acceleration of Linear Algebra using OpenCL*. 2012.

Special course: Leo Emil Sokoler and Oscar Borries, *Conjugate Gradients on GPU using CUDA*, 2012.

MSc: Høstergaard, Gade-Nielsen, Nicolai Fog, *Implementation and evaluation of fast computational methods for high-resolution ODF problems on multi-core and many-core systems*, 2010

PhD: Stefan L. Glimberg, *Designing Scientific Software for Heterogeneous Computing*, 2010-2013

PhD: Nicolai Fog Gade-Nielsen, *Scientific GPU Computing for Dynamical Optimization*, 2010-2014

PhD: Oscar Borries, *Large-Scale Computational Electromagnetics for Reflector Antenna Analysis*, 2011-

PostDoc DTU Physics

Ind. PostDoc DTU Compute

Some available projects:

-Acceleration of Wind Turbine Vortex Simulation (in collaboration with DTU RISØ).

-Large-scale 3D image reconstruction using GPU acceleration (in collaboration with University of Antwerp).

-Computational Electromagnetics for Reflector Antennas using Accelerators (in collaboration with TICRA).

-Fast Large-scale Banded Solver on the GPU (in collaboration with RESON A/S).

-Sparse matrix computations in genome-wide association studies (in collaboration with GenoKey).

-Your own project idea!