Introduction to Parallel and Distributed Processing

Introduction to GPGPUs

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

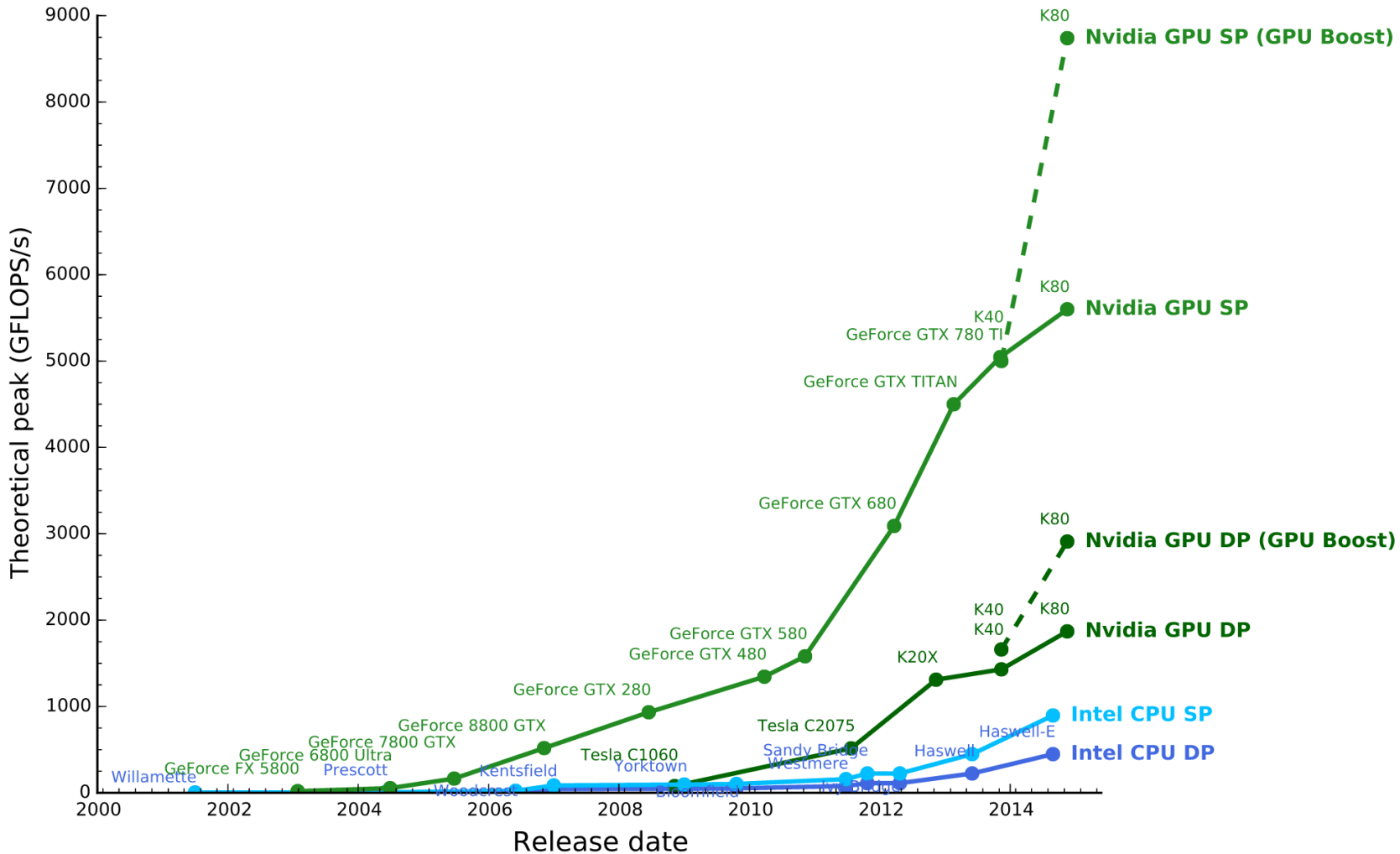
Suggested Reading

- CUDA Zone
  https://developer.nvidia.com/cuda-zone/

Need For Speed

- Parallelism at the processor level evolved along two “trajectories:”
- Multi-core general processors
  (with shared memory programming models)
- Many-core massively parallel accelerators
  (with massive data parallelism)

CPU vs GPU


Theoretical peak (GFLOPS/s)
Release date
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
2000
2002
2004
2006
2008
2010
2012
2014
K80
Nvidia GPU SP (GPU Boost)
K80
Nvidia GPU SP
K40
GeForce GTX 780 TI
GeForce GTX TITAN
GeForce GTX 680
K80
Nvidia GPU DP (GPU Boost)
K40
K40
K80
Nvidia GPU DP
GeForce GTX 580
GeForce GTX 480
K20X
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2075
Tesla C1060
Sandy Bridge
Westmere
Haswell
Haswell-E
Intel CPU SP
Intel CPU DP
Prescott
Kentsfield
Yorktown
Willamette

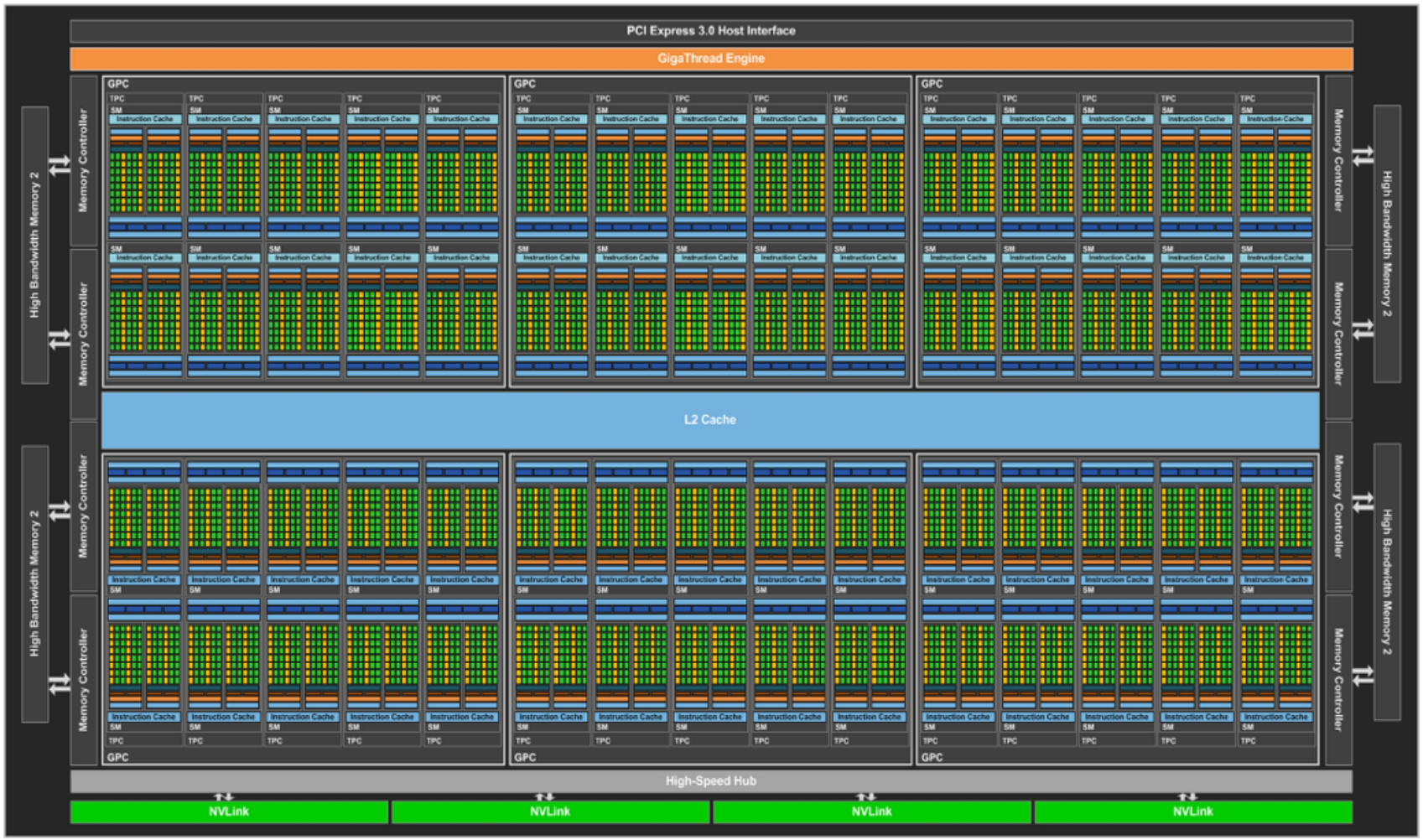
GPGPU Philosophy

- Many applications require pure number crunching and almost real-time response (e.g. deep learning)
- GPUs historically had excellent data processing capability
- GPGPUs pack many “simple” cores in several multiprocessors with no-cost of context switching

GPGPU Example Parallelism Scale

- 56 SM multiprocessors, each SM processor with 64 SP CUDA-cores and 4 TU
- 32 threads form warp and are executed together (SIMD)
- 64 warps per processor or 2048 concurrent threads
- Over 100K concurrent threads on the device
- ~5TFLOPs in DP and ~10TFLOPs in SP

GPGPU Architecture Example



GPGPU Architecture Example

Programming GPGPUs

- Significantly vendor specific: e.g. NVIDIA CUDA
- OpenCL tries to standardize both multi/many-cores
- OpenACC proprietary OpenMP-style parallelization
- OpenMP provides a set of offloading pragmas
- Thrust C++ provides STL-style API on top of CUDA

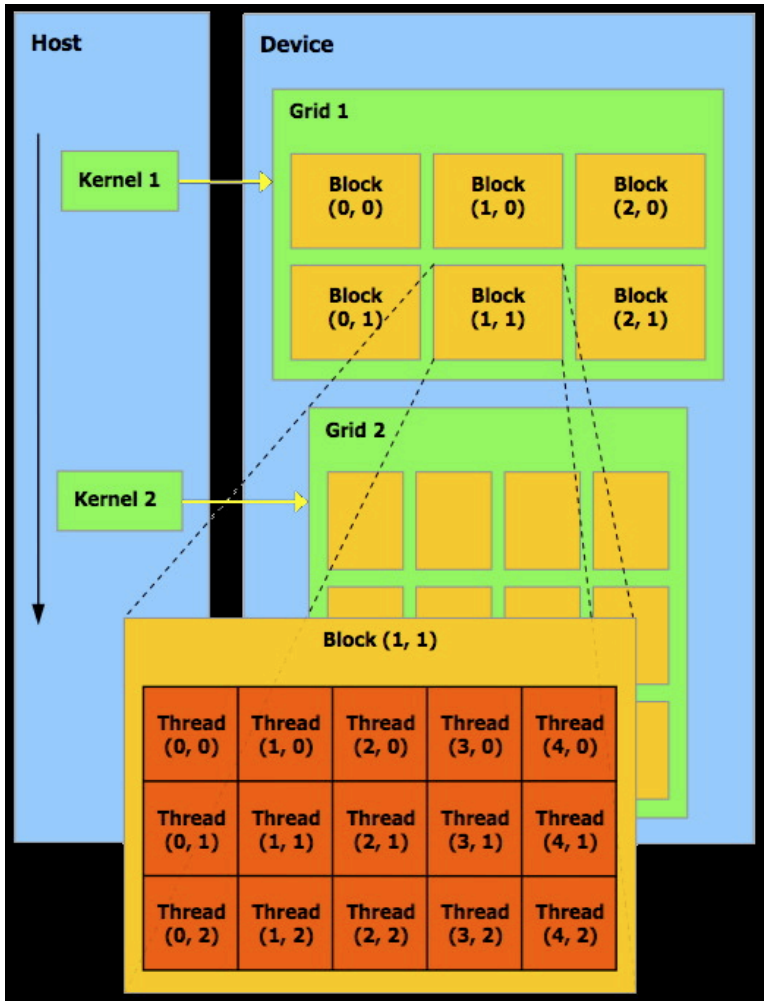
NVIDIA CUDA

- API similar to C/C++ with extra keywords
- User defines “kernels” that will be executed on GPU
- Kernel when launched creates thousands of threads
- Application may have many kernels
- Kernel may require data movement from the main memory (host memory) to GPU memory (device memory)

CUDA Threads

- Kernel can launch more threads than the device can handle
- Threads grouped (logically) into blocks
- Blocks organized into a grid

CUDA Threads



CUDA Execution

- Thread executes scalar instructions on a single core
- Thread can use some number of registers (255)
- Block of threads executes on a single multiprocessor
  no migration, all threads run in parallel,
  limit on threads/block (1024)
- Multiprocessor can execute multiple blocks
  limit on blocks/multiprocessor (16)
- Grid executes on a single device
  concurrently or serially

CUDA Block Abstraction

- Threads within block can communicate, synchronize
- Blocks can be distributed on many multiprocessors
- Can execute in any arbitrary order without pre-emption

For Fun

- Spend some time reviewing details of NVIDIA CUDA and GPGPU architecture.