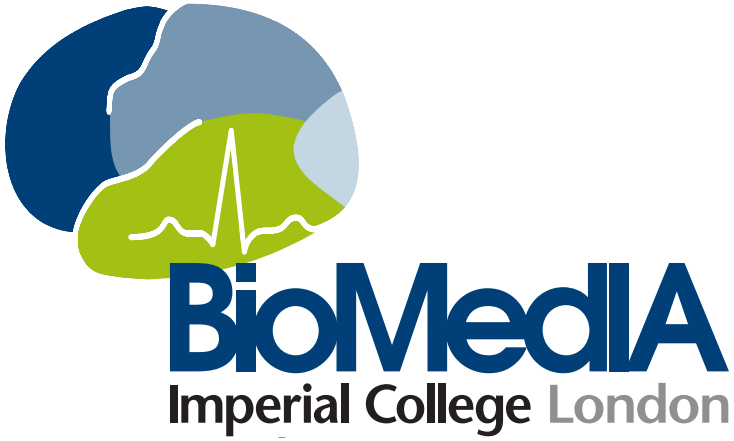
GPU programming

Dr. Bernhard Kainz



Overview



- About myself
- Motivation
- GPU hardware and system architecture
- GPU programming languages
- GPU programming paradigms
- Pitfalls and best practice
- Reduction and tiling examples
- State-of-the-art applications

This week

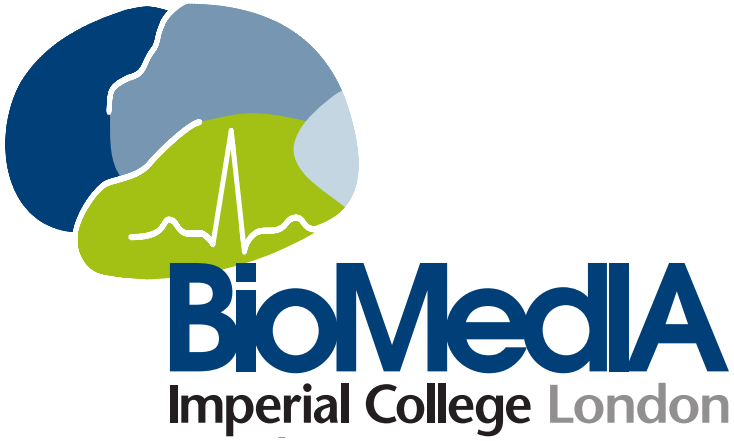
Next week

About myself



- Born, raised, and educated in Austria
- PhD in interactive medical image analysis and visualisation
- Marie-Curie Fellow, Imperial College London, UK
- Senior research fellow King‘s College London
- Lecturer in high-performance medical image analysis at DOC
- > 10 years GPU programming experience

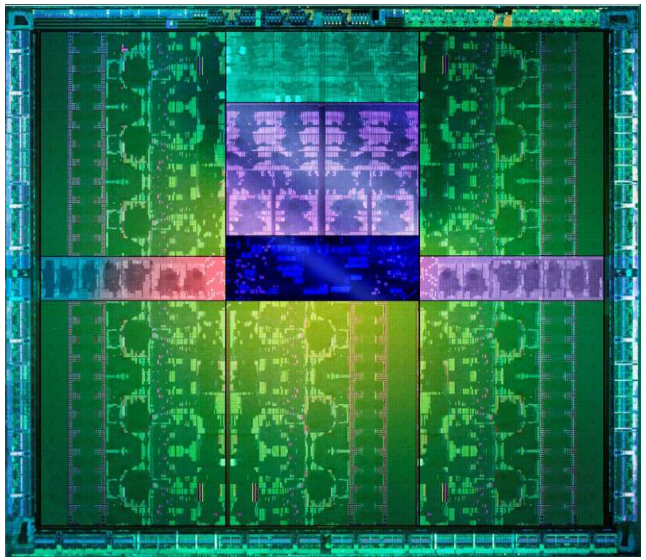
History



GPUs







GPU = graphics processing unit

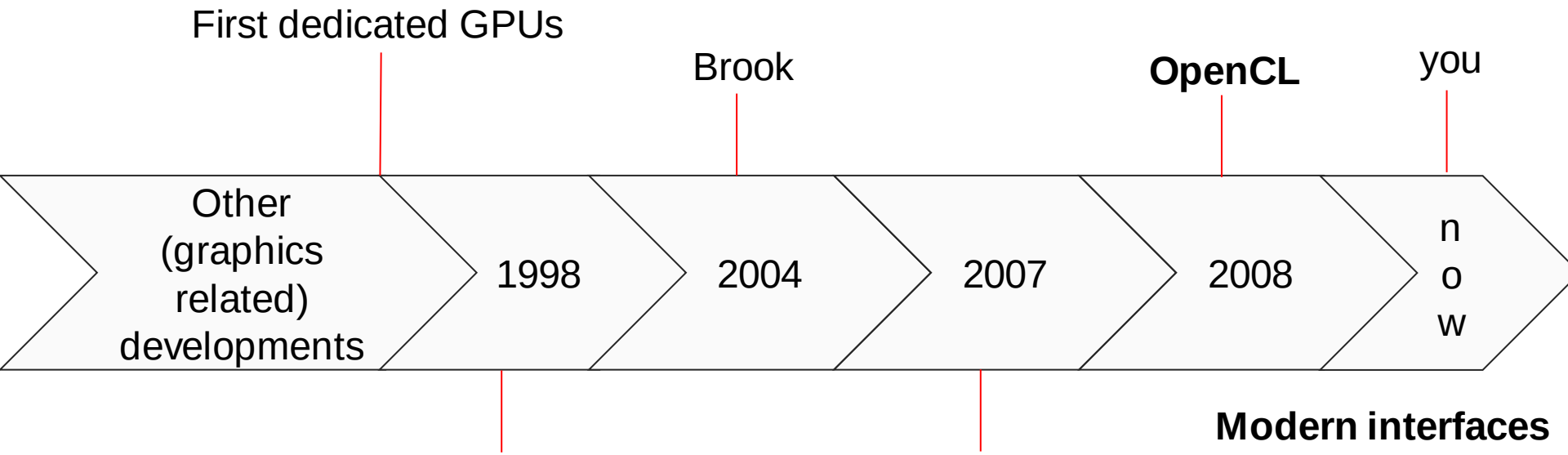
GPGPU = General Purpose Computation on Graphics Processing Units

CUDA = Compute Unified Device Architecture

OpenCL = Open Computing Language

Images: www.geforce.co.uk

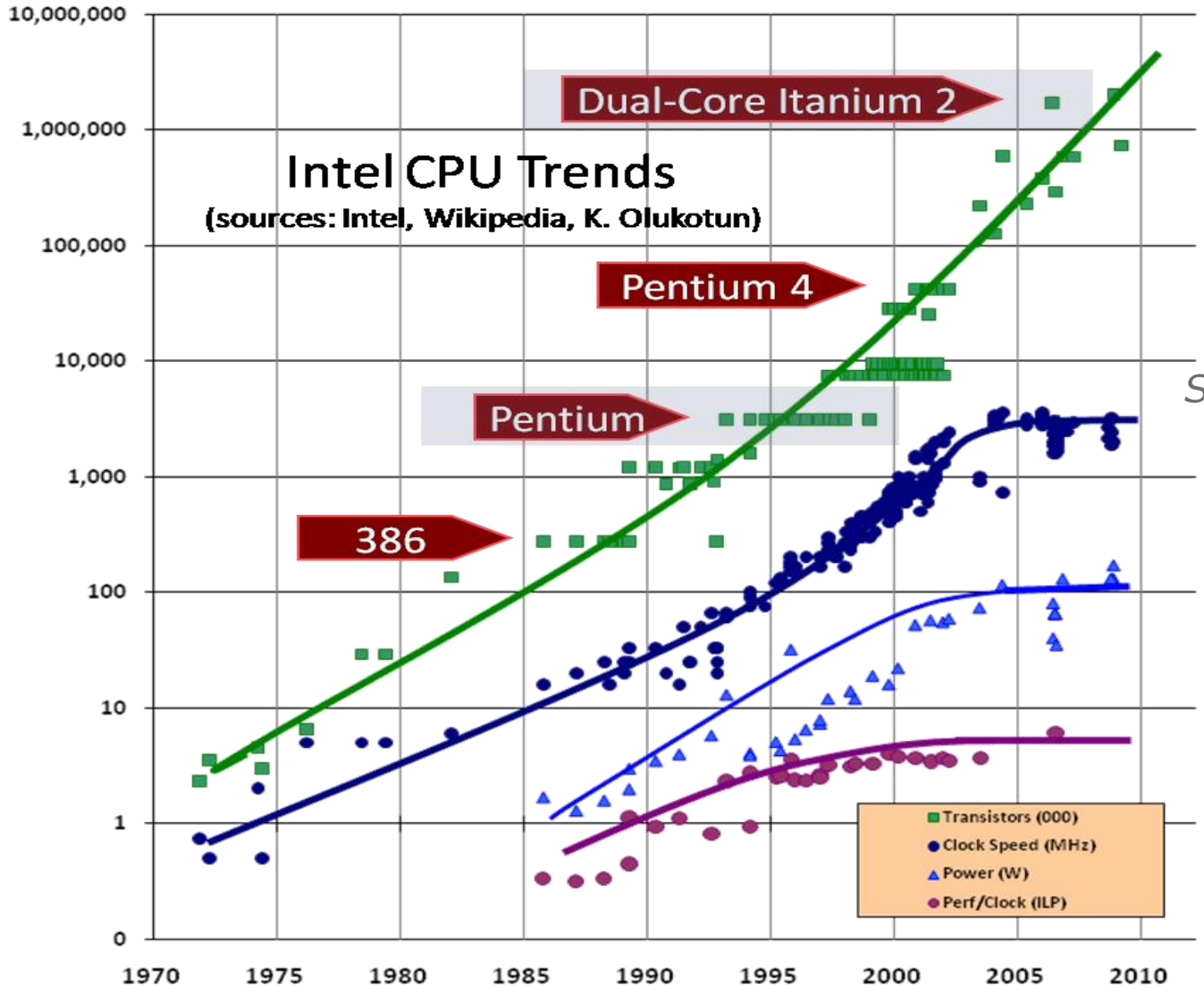
History


First dedicated GPUs
Brook
OpenCL
you
Other (graphics related) developments
1998
2004
2007
2008
now
programmable shader
CUDA
Modern interfaces to CUDA and OpenCL (python, Matlab, etc.)



http://www.computerhistory.org/timeline/graphics-games/

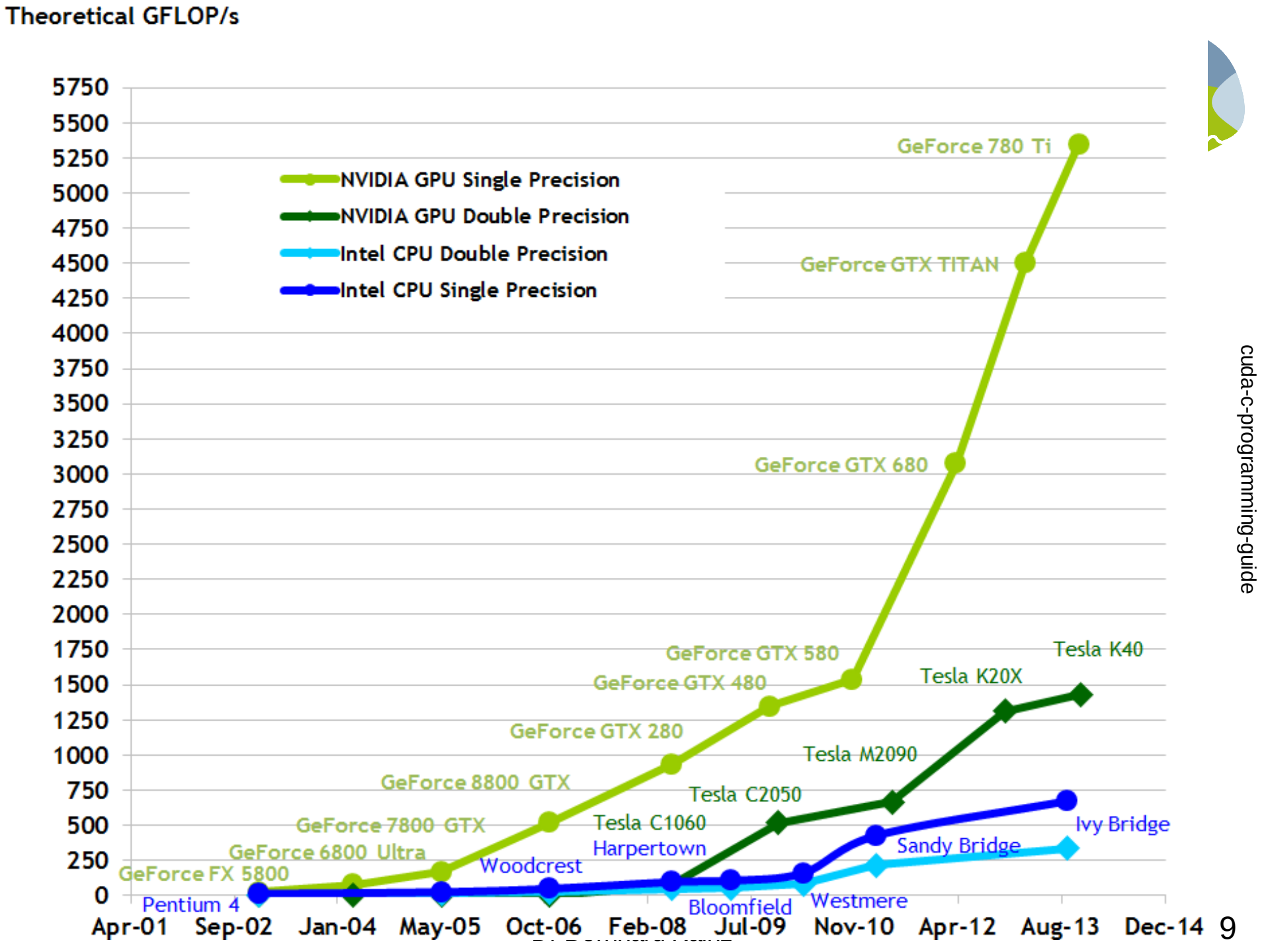
Why GPUs became popular for computing

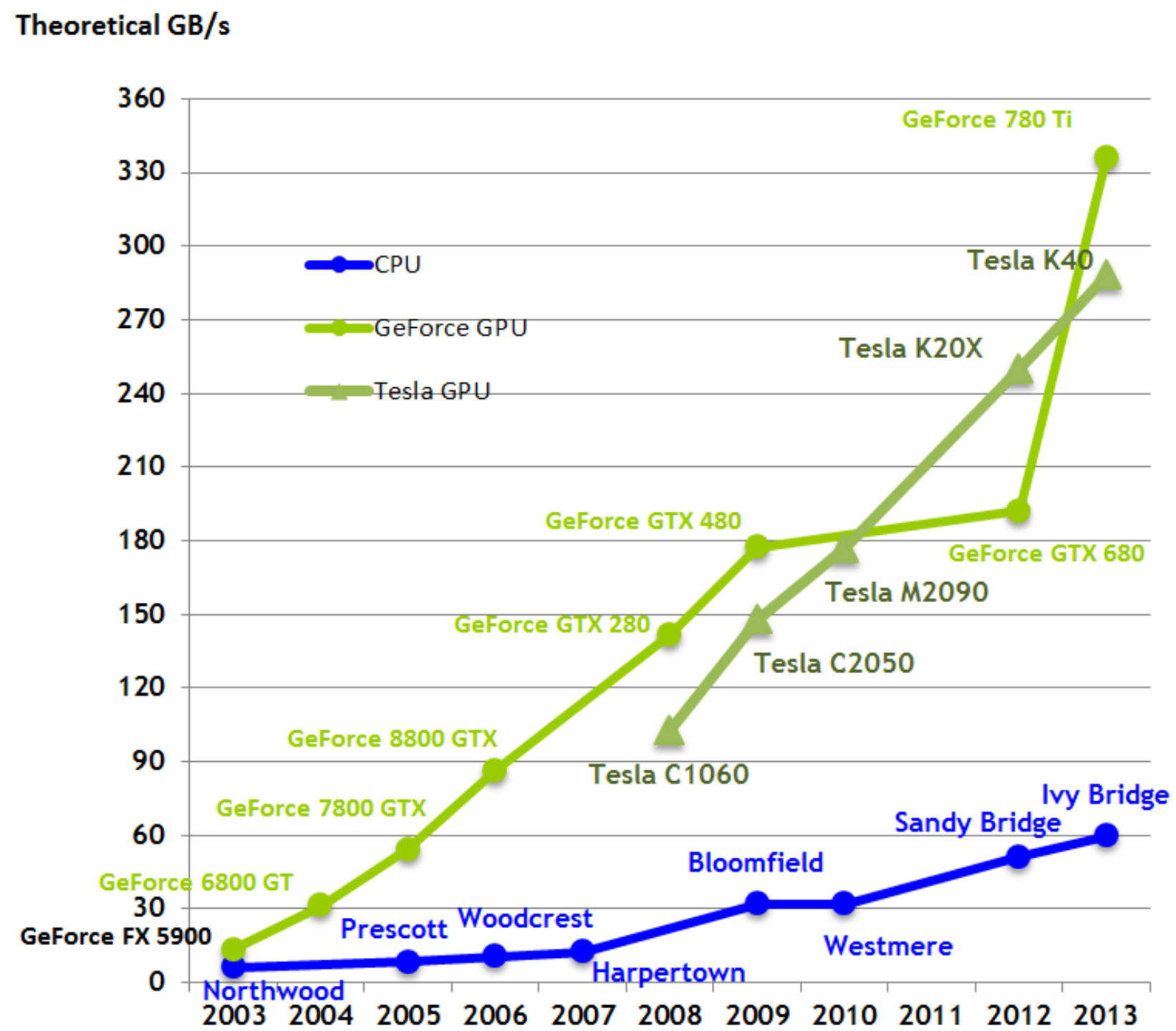


Sandy Bridge

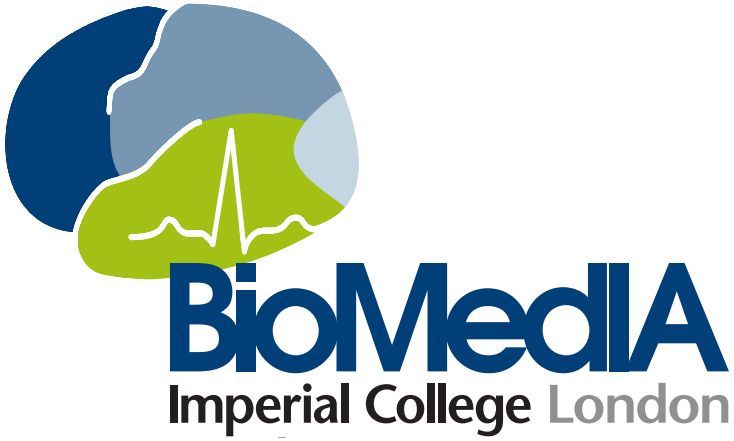
Haswell

© HerbSutter
„The free lunch is over"

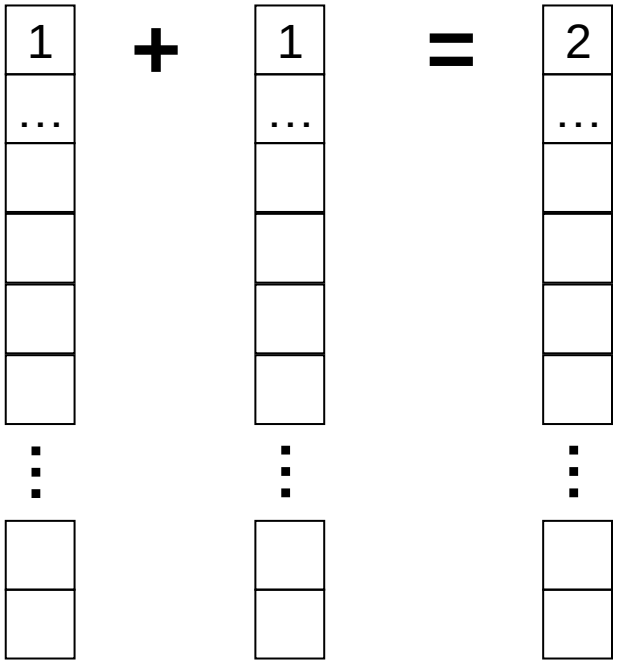

Theoretical GFLOP/s
5750
5500
5250
5000
4750
4500
4250
4000
3750
3500
3250
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Double Precision
Intel CPU Single Precision
GeForce 780 Ti
GeForce GTX TITAN
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla K40
Tesla K20X
Tesla M2090
Tesla C2050
Tesla C1060
Harpertown
Woodcrest
Pentium 4
Bloomfield
Westmere
Sandy Bridge
Ivy Bridge
Apr-01
Sep-02
Jan-04
May-05
Oct-06
Feb-08
Jul-09
Nov-10
Apr-12
Aug-13
Dec-14


Theoretical GB/s
360
330
300
270
240
210
180
150
120
90
60
30
0
CPU
GeForce GPU
Tesla GPU
GeForce 780 Ti
Tesla K40
Tesla K20X
GeForce GTX 480
GeForce GTX 680
Tesla M2090
GeForce GTX 280
Tesla C2050
GeForce 8800 GTX
Tesla C1060
GeForce 7800 GTX
Ivy Bridge
Sandy Bridge
Bloomfield
GeForce 6800 GT
GeForce FX 5900
Prescott
Woodcrest
Westmere
Harpertown
Northwood
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Motivation



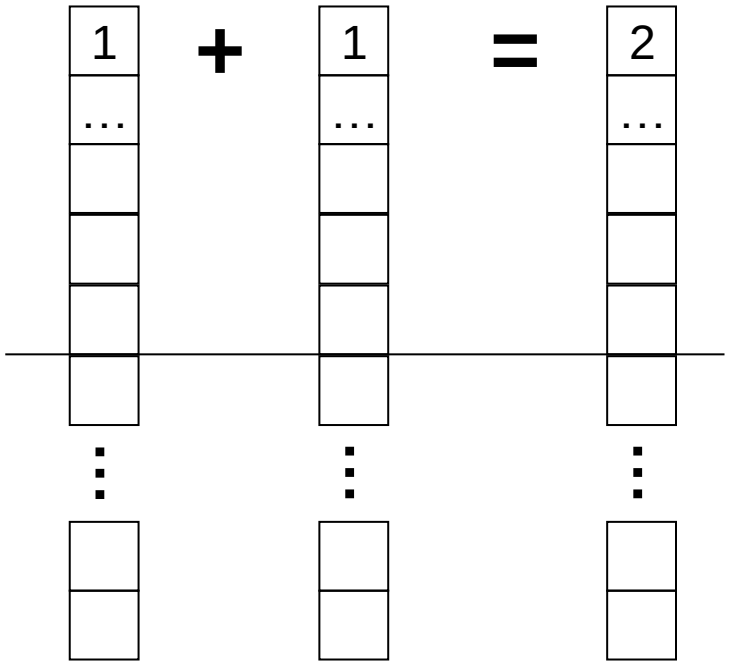
parallelisation



Thread 0

```
for (int i = 0; i < N; ++i)
    c[i] = a[i] + b[i];
```

parallelisation



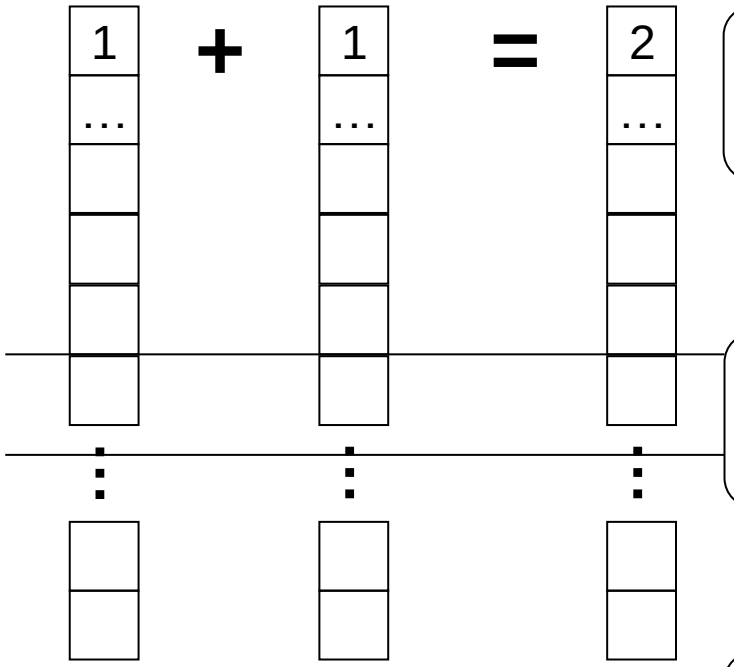
Thread 0

```
for (int i = 0; i < N/2; ++i)
    c[i] = a[i] + b[i];
```

Thread 1

```
for (int i = N/2; i < N; ++i)
    c[i] = a[i] + b[i];
```

parallelisation



Thread 0

```
for (int i = 0; i < N/3; ++i)
    c[i] = a[i] + b[i];
```

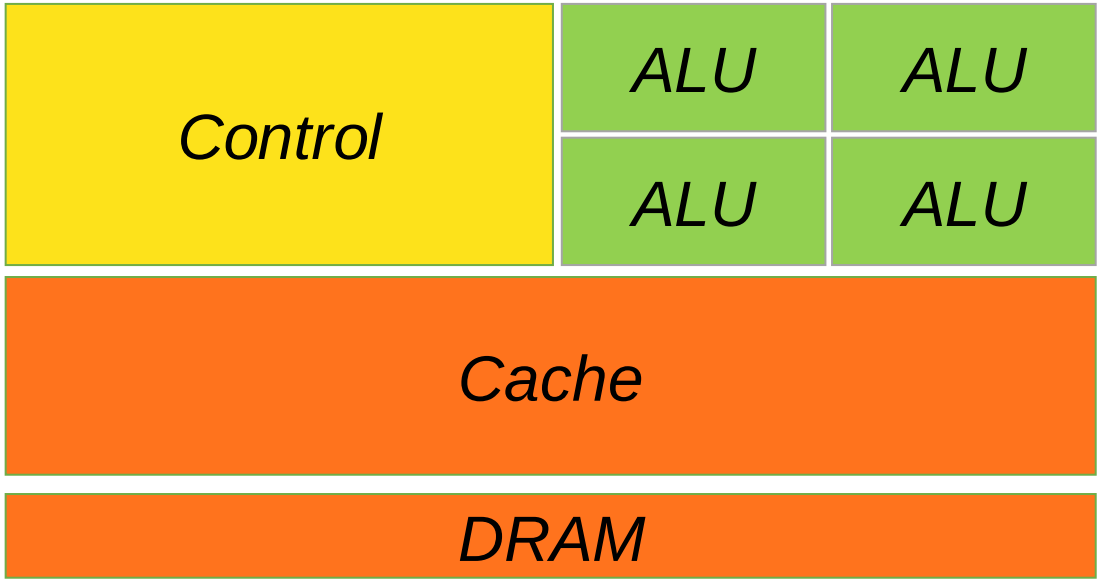
Thread 1

```
for (int i = N/3; i < 2*N/3; ++i)
    c[i] = a[i] + b[i];
```

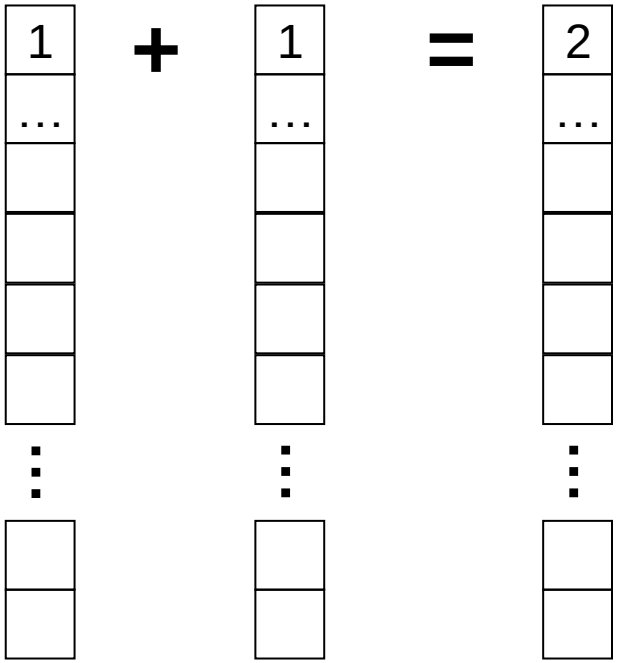
Thread 2

```
for (int i = 2*N/3; i < N; ++i)
    c[i] = a[i] + b[i];
```

multi-core CPU



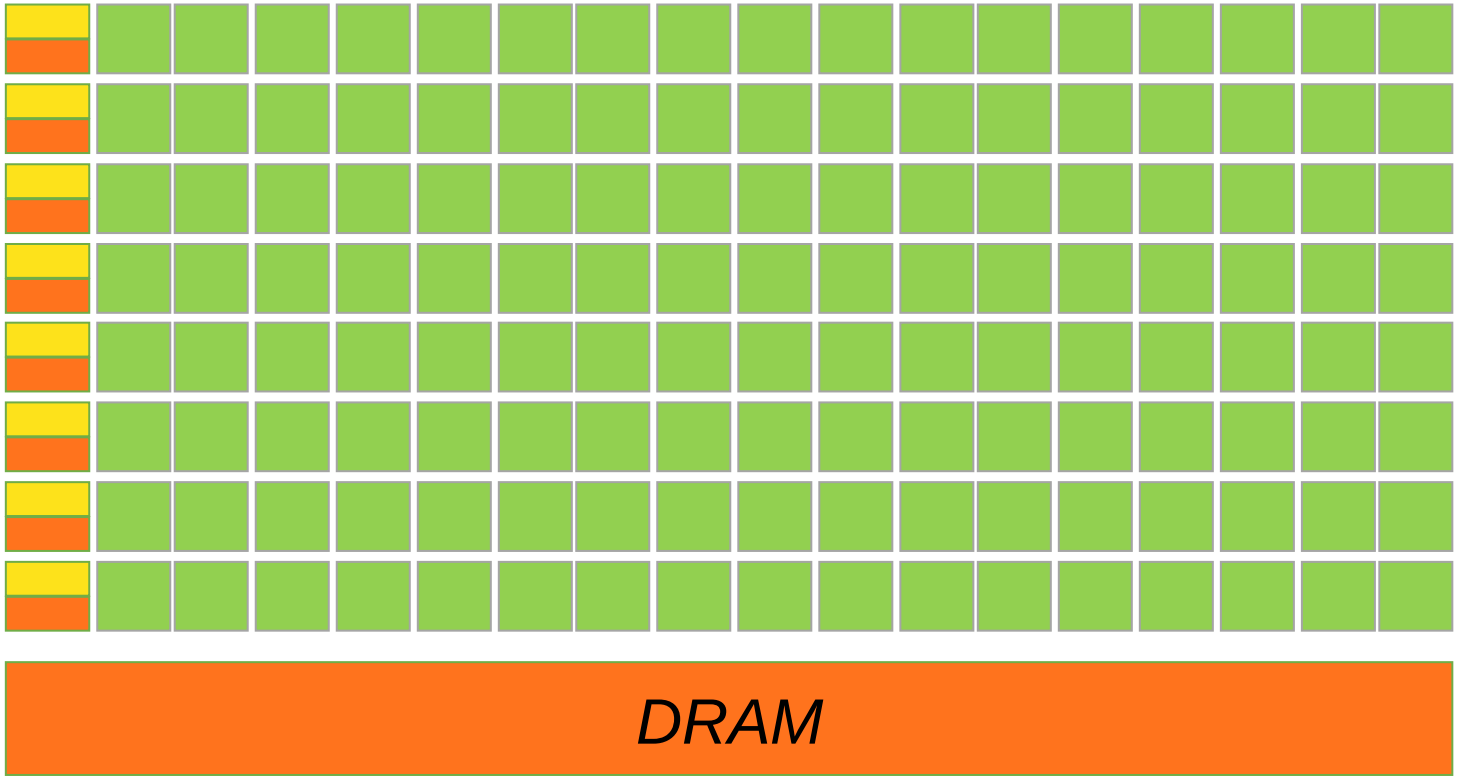
parallelisation




```
c[0] = a[0] + b[0];
c[1] = a[1] + b[1];
c[2] = a[2] + b[2];
c[3] = a[3] + b[3];
c[4] = a[4] + b[4];
c[5] = a[5] + b[5];
```

```
c[N-1] = a[N-1] + b[N-1];
c[N] = a[N] + b[N];
```

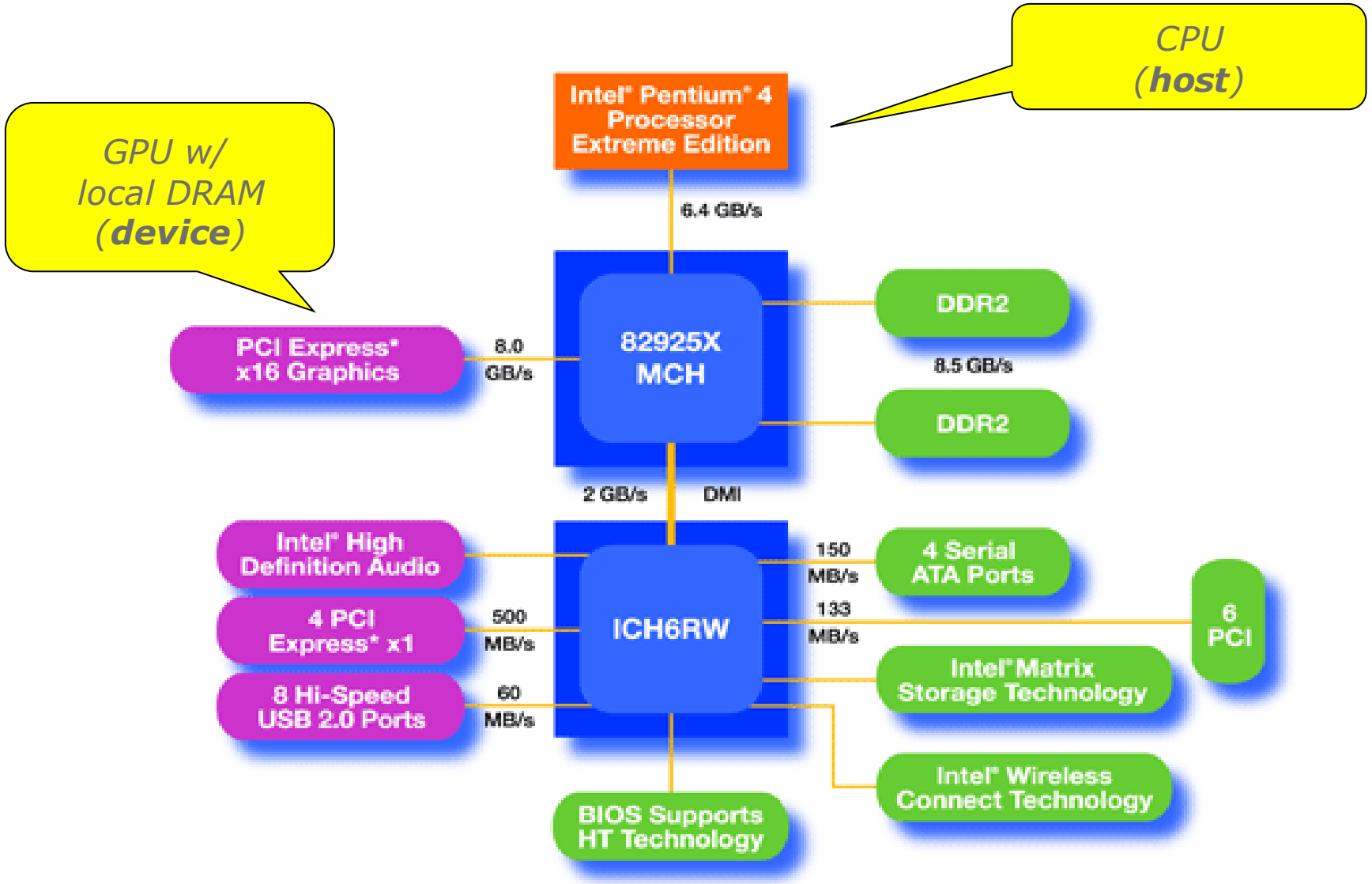
multi-core GPU



Terminology



Host vs. device



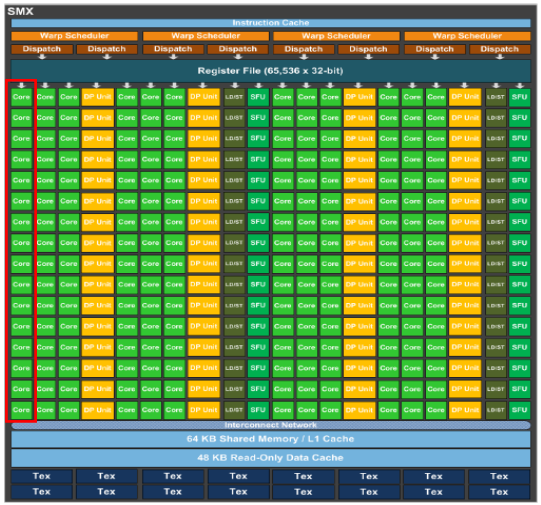


current schematic Nvidia Maxwell architecture

Streaming Multiprocessors (SM,SMX)



- single-instruction, multiple-data (SIMD) hardware
- 32 threads form a warp
- Each thread within a warp must execute the same instruction (or be deactivated)
- 1 instruction → 32 values computed
- handle more warps than cores to hide latency



Differences CPU-GPU

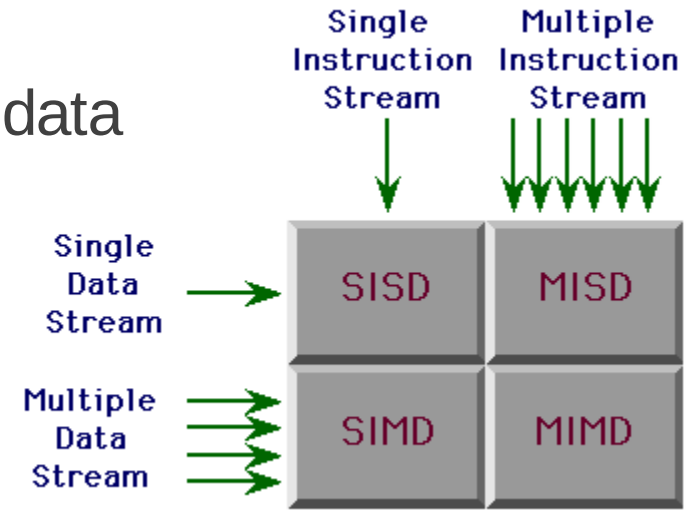


- Threading resources
  - Host currently ~32 concurrent threads
  - Device: smallest executable unit of parallelism: “Warp”: 32 thread
  - 768-1024 active threads per multiprocessor
  - Device with 30 multiprocessors: > 30.000 active threads
  - Devices can hold billions of threads
- Threads
  - Host: heavyweight entities, context switch expensive
  - Device: lightweight threads
  - If the GPU processor must wait for one warp of threads, it simply begins executing work on another Warp.
- Memory
  - Host: equally accessible to all code
  - Device: divided virtually and physically into different types

Flynn‘s Taxonomy



- **SISD**: single-instruction, single-data (single core CPU)
- **MIMD**: multiple-instruction, multiple-data (multi core CPU)
- **SIMD**: single-instruction, multiple-data (data-based parallelism)
- **MISD**: multiple-instruction, single-data (fault-tolerant computers)

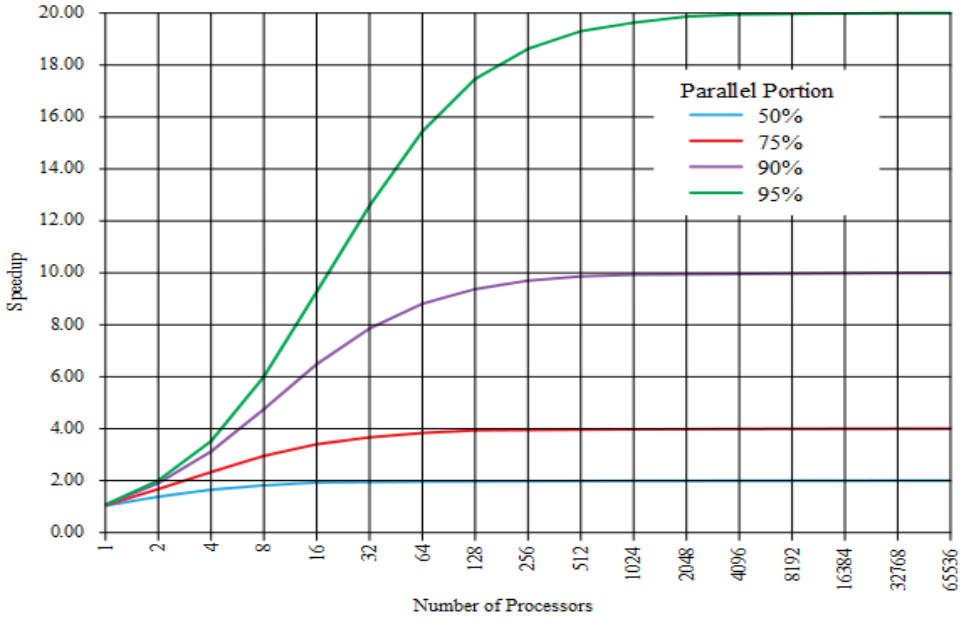


Amdahl's Law

- Sequential vs. parallel
- Performance benefit

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

- P: parallelizable part of code
- N: # of processors

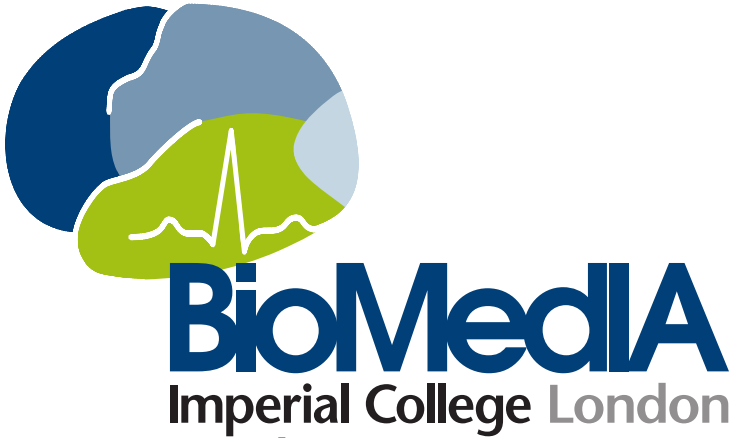


SM Warp Scheduling



- SM hardware implements zero overhead Warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution on a prioritized scheduling policy
  - All threads in a Warp execute the same instruction when selected
- Currently: ready-queue and memory access score-boarding
- Thread and warp scheduling are active topics of research!

Programming GPUs



Programming languages



- OpenCL (Open Computing Language):
- **OpenCL is an open, royalty-free, standard for cross-platform, parallel programming of modern processors**
- An Apple initiative approved by Intel, Nvidia, AMD, etc.
- Specified by the Khronos group (same as OpenGL)
- It intends to unify the access to heterogeneous hardware accelerators
  - CPUs (Intel i7, …)
  - GPUs (Nvidia GTX & Tesla, AMD/ATI 58xx, …)
- **What's the difference to other languages?**
  - Portability over Nvidia, ATI, S3… platforms + CPUs
  - Slow or no implementation of new/special hardware features

Programming languages



- CUDA:
- **“Compute Unified Device Architecture”**
  - Nvidia GPUs only!
  - Open source announcement
  - Does not provide CPU fallback
  - NVIDIA CUDA Forums – 26,893 topics
    - AMD OpenCL Forums – 4,038 topics
  - Stackoverflow CUDA Tag – 1,709 tags
    - Stackoverflow OpenCL Tag – 564 tags
  - Raw math libraries in NVIDIA CUDA
    - CUBLAS, CUFFT, CULA, Magma
- new hardware features immediately available!

Installation



- Download and install the newest driver for your GPU!
- OpenCL: get SDK from Nvidia or AMD
- CUDA: https://developer.nvidia.com/cuda-downloads
- CUDA nvcc complier -> easy access via CMake and .cu files
- OpenCL -> no special compiler, runtime evaluation
- Integrated Intel something graphics -> No No No!



Writing parallel code



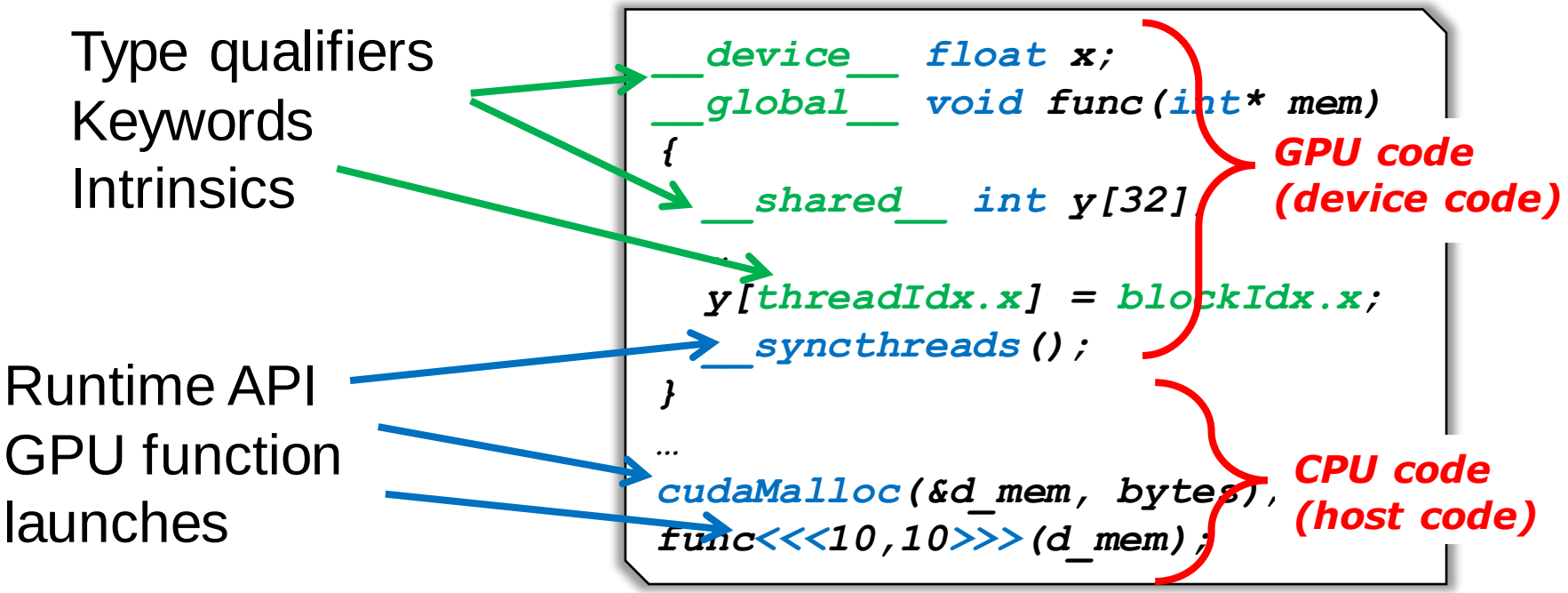
- Current GPUs have > 3000 cores (GTX TITAN, Tesla K80 etc.)
- Need more threads than cores (warp scheduler)
- Writing different code for 10000 threads / 300 warps?

→Single-program, multiple-data (SPMD = SIMDI) model

- Write one program that is executed by all threads

CUDA C

CUDA C is C (C++) with additional keywords to control parallel execution



Kernel



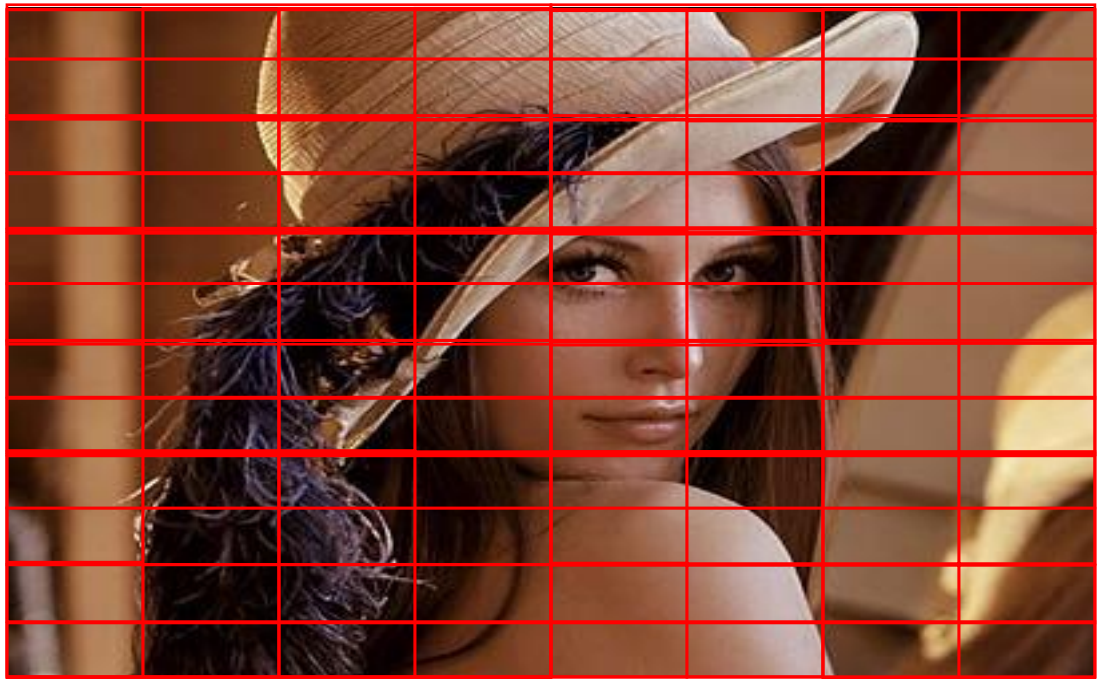
- A function that is executed on the GPU
- Each started thread is executing the same function

```
__global__ void myfunction(float *input, float* output)
{
  *output = *input;
}
```

- Indicated by **__global__**
- must have return value **void**

Parallel Kernel

- Kernel is split up in blocks of threads



Launching a kernel



- A function that is executed on the GPU
- Each started thread is executing the same function

```
dim3 blockSize(32,32,1);
dim3 gridSize((iSpaceX + blockSize.x - 1)/blockSize.x,
              (iSpaceY + blockSize.y - 1)/blockSize.y), 1)
myfunction<<<gridSize, blockSize>>>(input, output);
```

- Indicated by **__global__**
- must have return value **void**

Distinguishing between threads



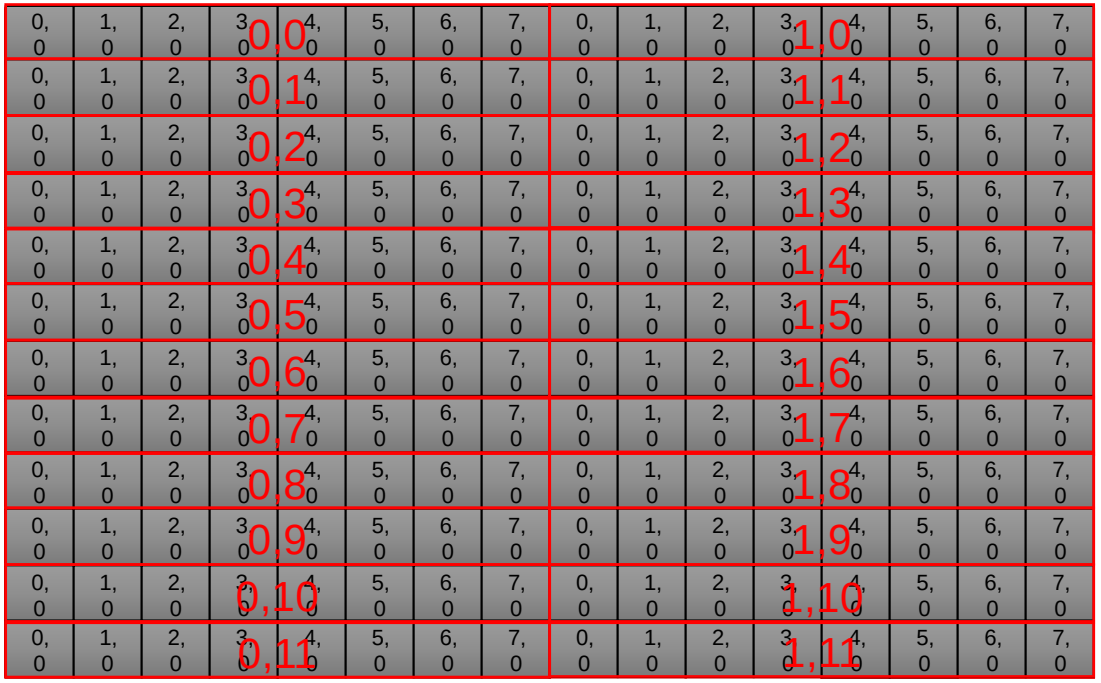
- using **threadIdx** and **blockIdx** execution paths are chosen
- with **blockDim** and **gridDim** number of threads can be determined

```
__global__ void myfunction(float *input, float* output)
{
  uint bid = blockIdx.x + blockIdx.y * gridDim.x;
  uint tid = bId * (blockDim.x * blockDim.y) +
             (threadIdx.y * blockDim.x) + threadIdx.x;
  output[tid] = input[tid];
}
```

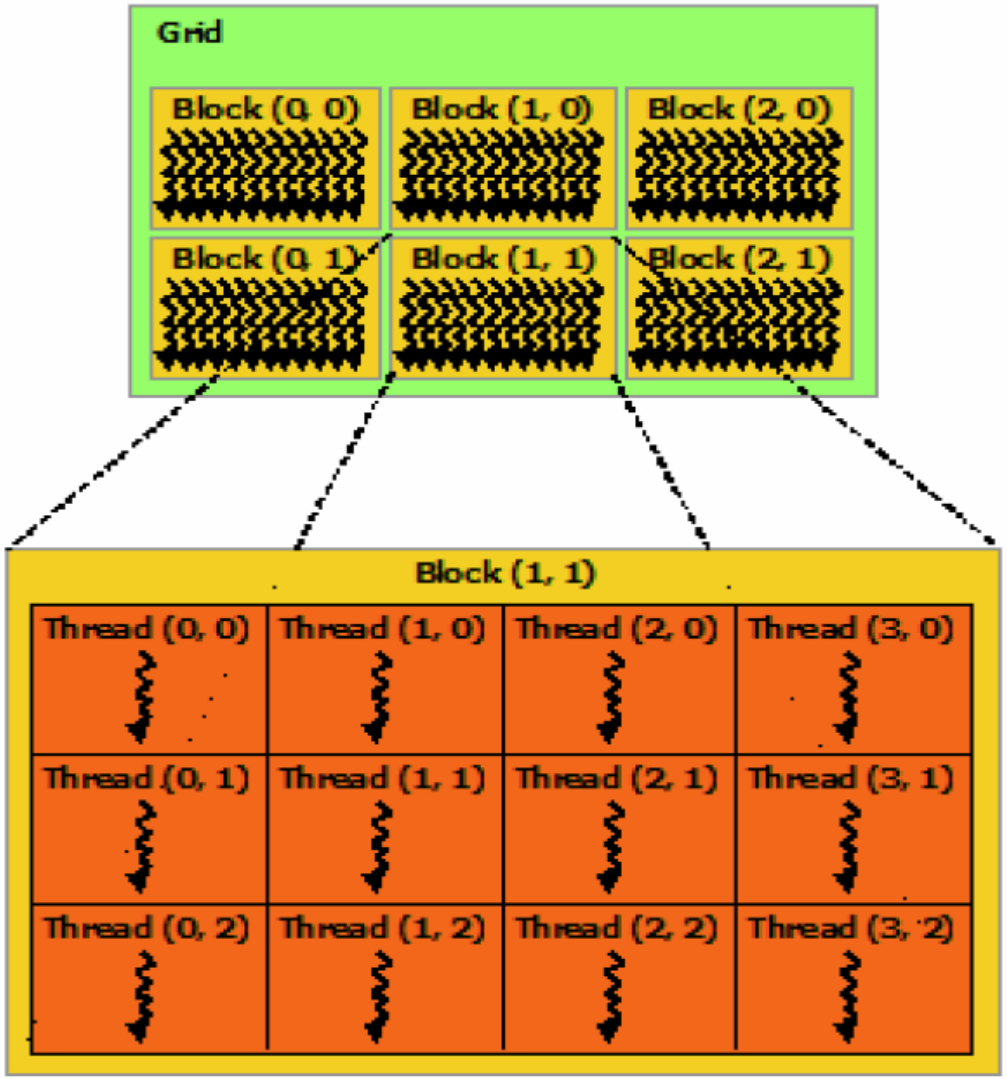
Distinguishing between threads



- blockId and threadId



Grids, Blocks, Threads



Blocks



- Threads within one block…
  - are executed together
  - can be synchronized
  - can communicate efficiently
  - share the same local cache

    → can work on a goal cooperatively

Blocks

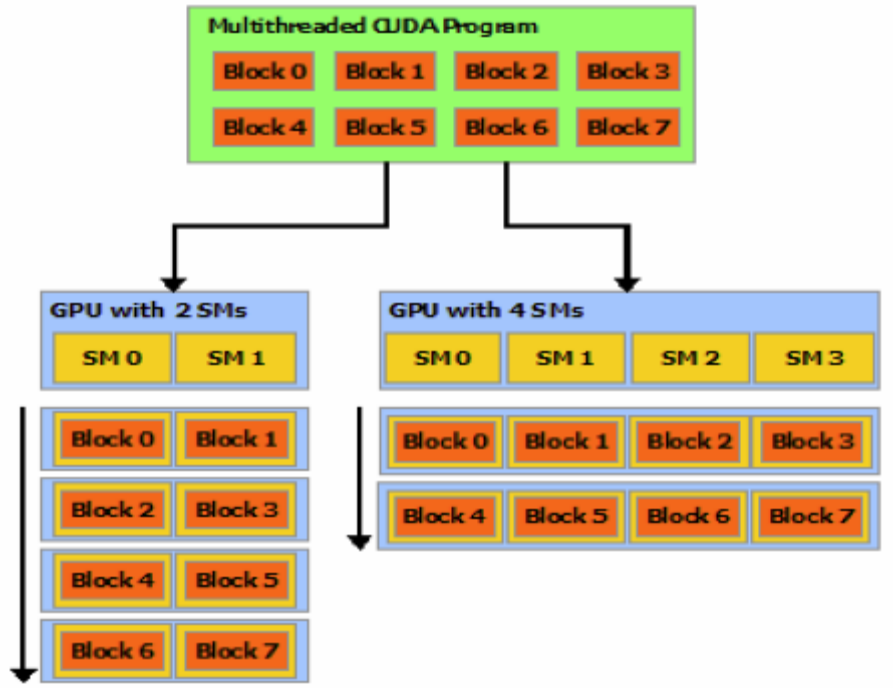


- Threads of different blocks…
  - may be executed one after another
  - cannot synchronize on each other
  - can only communicate inefficiently

  → should work independently of other blocks

Block Scheduling

- Block queue feeds multiprocessors
- Number of available multiprocessors determines number of concurrently executed blocks



Blocks to warps



- On each multiprocessor each block is split up in warps Threads with the lowest id map to the first warp

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0	9,0	10,0	11,0	12,0	13,0	14,0	15,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1	11,1	12,1	13,1	14,1	15,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2	10,2	11,2	12,2	13,2	14,2	15,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3	11,3	12,3	13,3	14,3	15,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4	10,4	11,4	12,4	13,4	14,4	15,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5	11,5	12,5	13,5	14,5	15,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6	9,6	10,6	11,6	12,6	13,6	14,6	15,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7	9,7	10,7	11,7	12,7	13,7	14,7	15,7

warp 0

warp 1

warp 2

warp 3

Where to start



- CUDA programming guide: https://docs.nvidia.com/cuda/cuda-c-programming-guide/
- OpenCL
  http://www.nvidia.com/content/cudazone/download/opencl/nvidia_opencl_programmingguide.pdf
  http://developer.amd.com/tools-and-sdks/opencl-zone/

GPU programming

Dr. Bernhard Kainz

