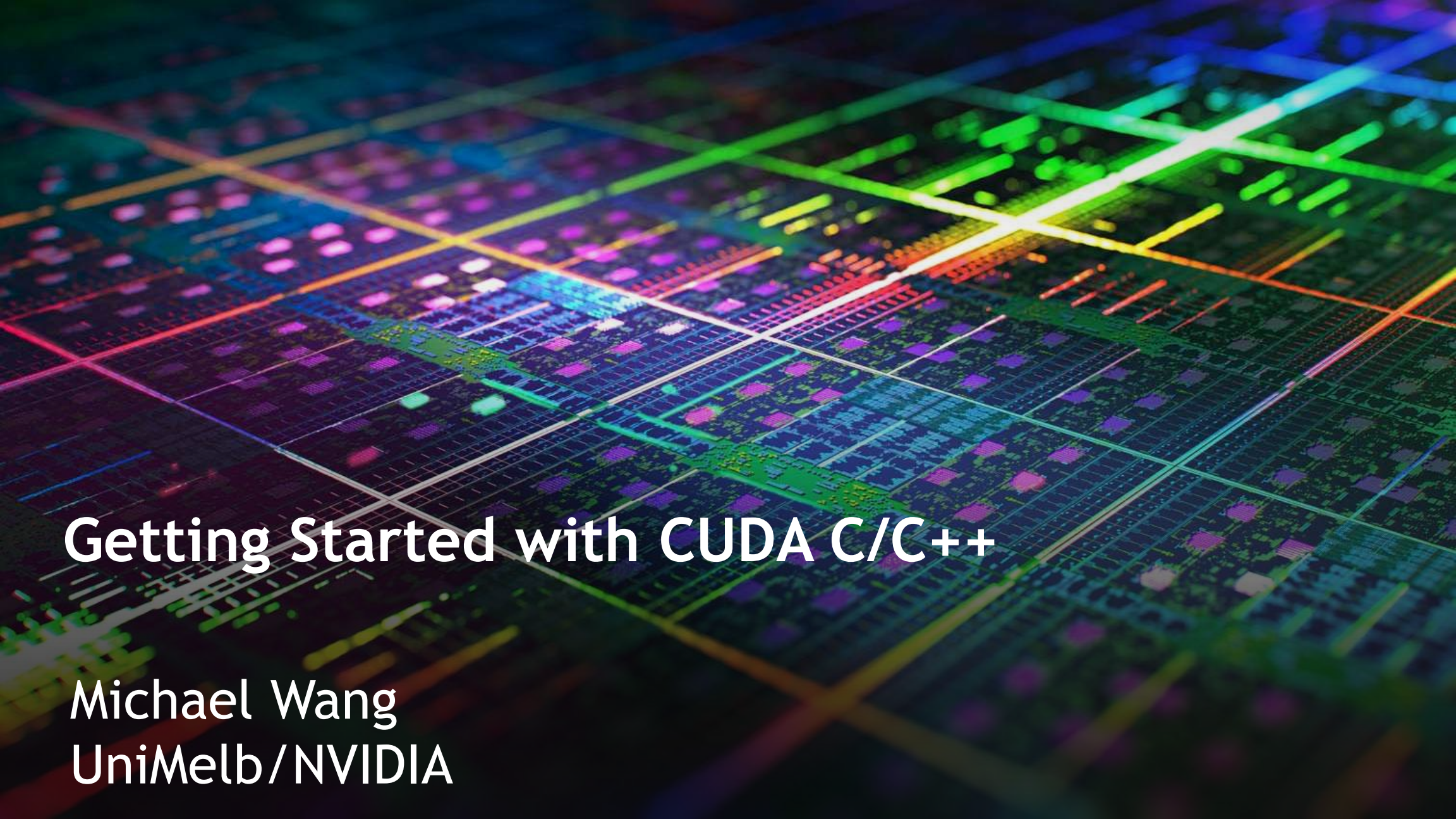
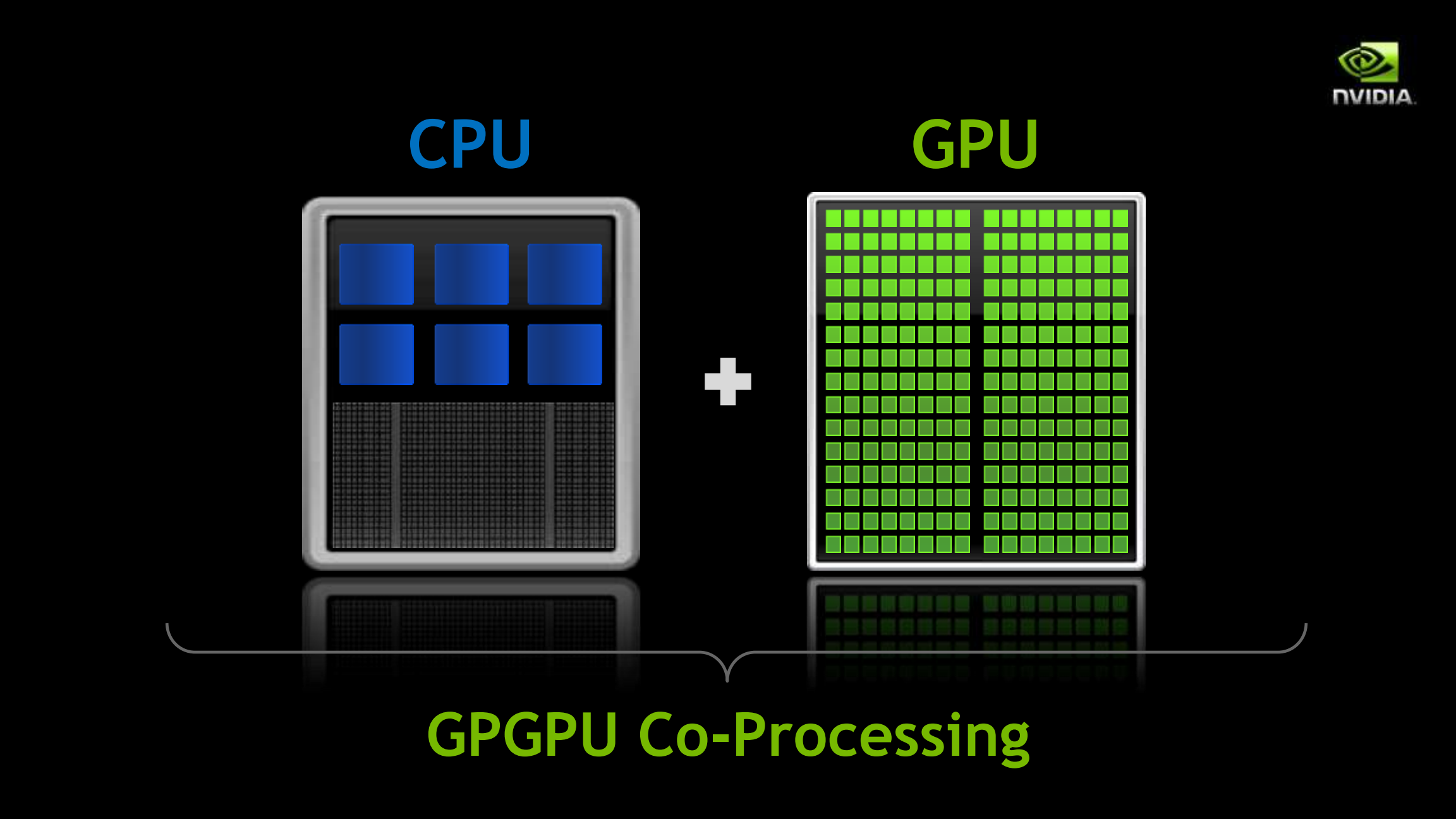


Getting Started with CUDA C/C++

Michael Wang
UniMelb/NVIDIA

Thank You to Our Sponsors


CPU
GPU
GPGPU Co-Processing

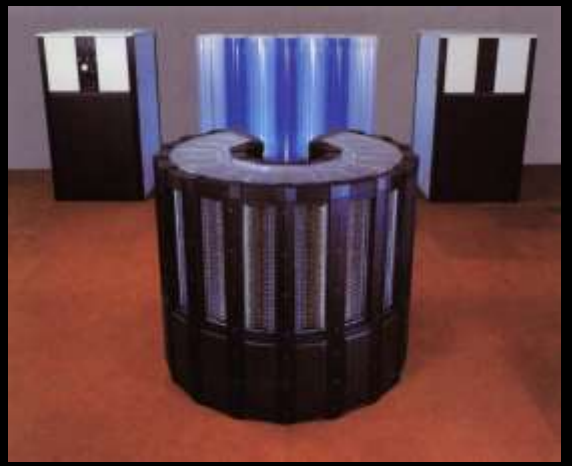
Once upon a time...

Past Massively Parallel Supercomputers

Thinking Machine

Goodyear MPP

MasPar

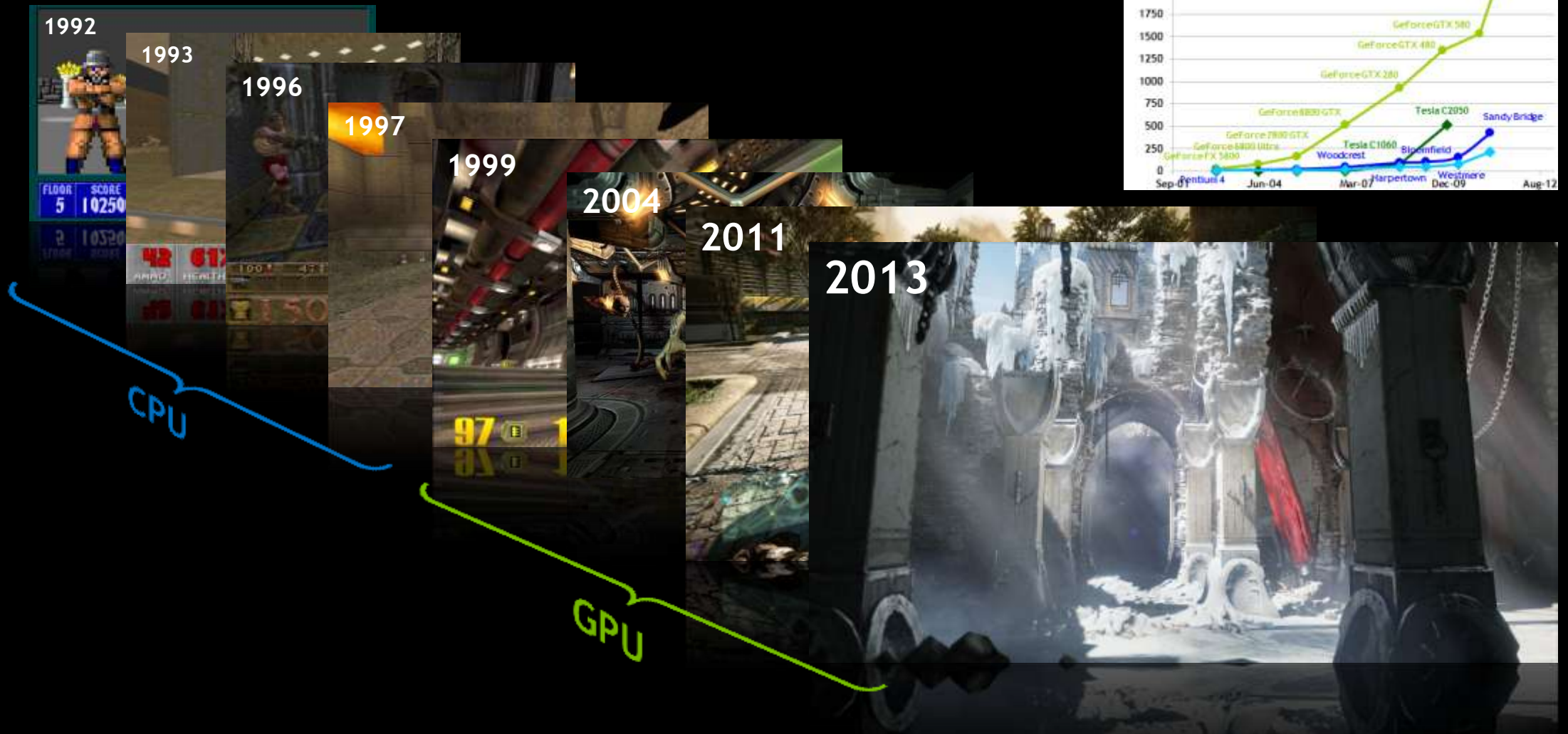


Cray 2

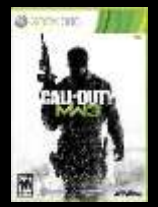
Today

1.31 TFLOPS on DGEMM

Made Possible by Graphics



Leveraging The Power of Graphics



3D Application or Game

3D Engine

CPU

GPU

Fixed Function Pipeline

GPU Front End

Primitive Assembly

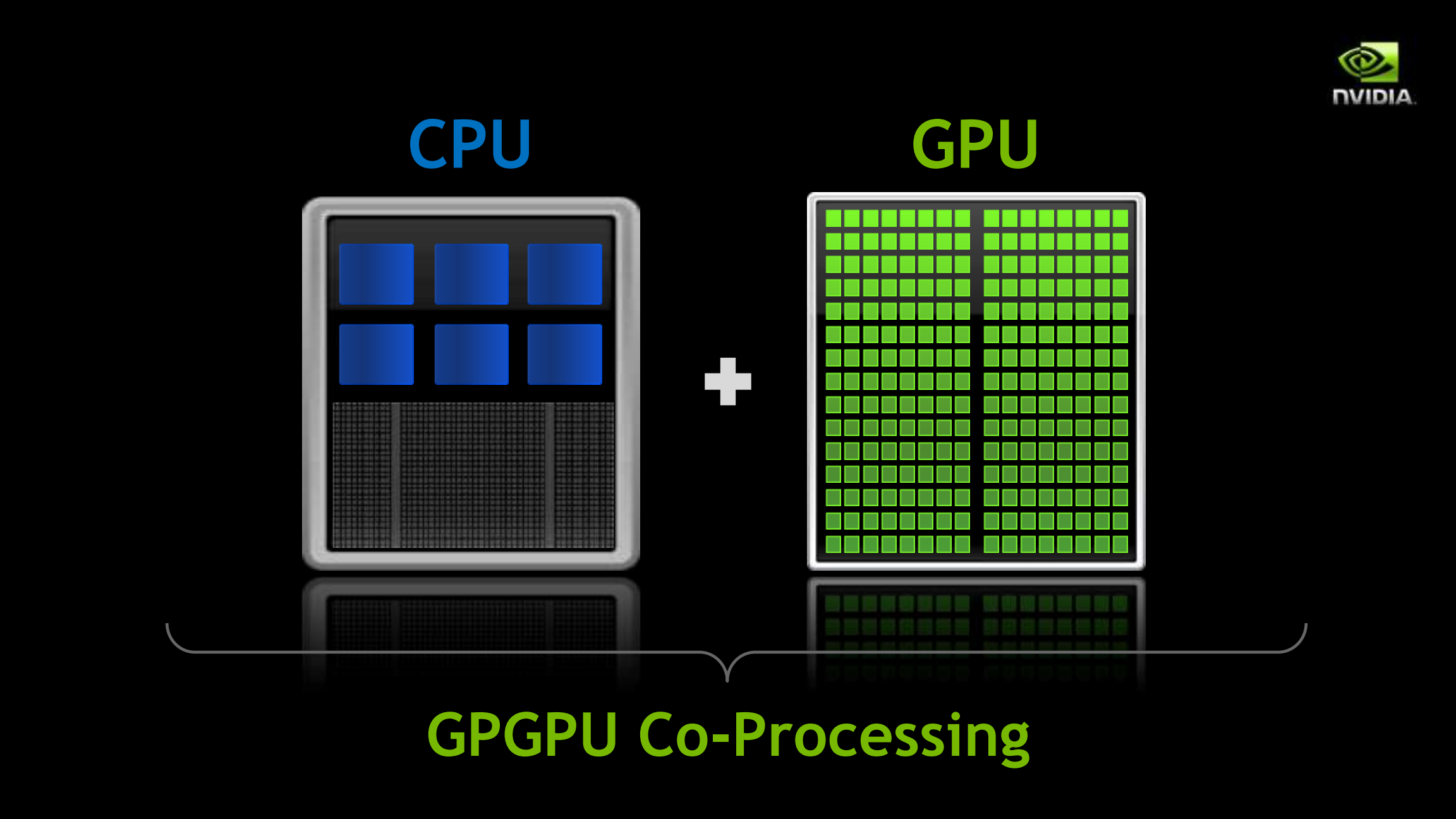
Rasterisation & Interpolation

Raster Operations

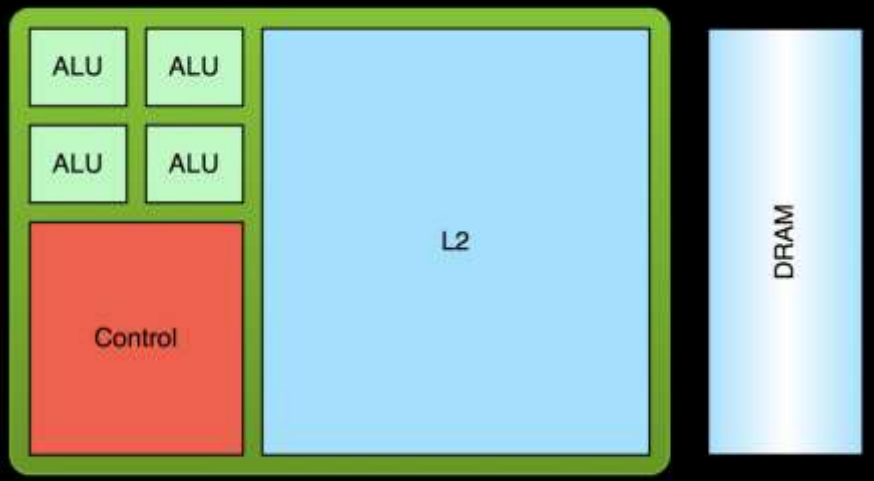
Frame Buffer

Fully Programmable Pipeline

GENERAL PURPOSE

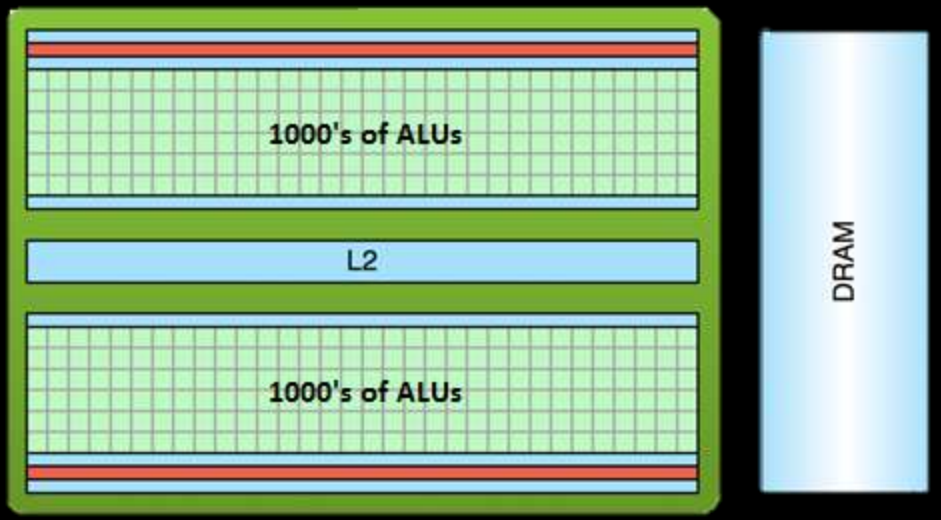

CPU
GPU
GPGPU Co-Processing

Low Latency vs High Throughput



CPU

- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution



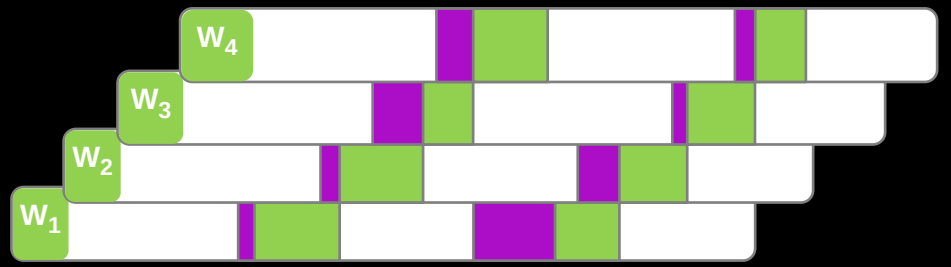
GPU

- Optimized for data-parallel, throughput computation
- Architecture tolerant of memory latency
- More transistors dedicated to computation

Low Latency or High Throughput?

- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other thread warps

GPU Stream Multiprocessor – High Throughput Processor



CPU core – Low Latency Processor

Computation Thread/Warp

Processing

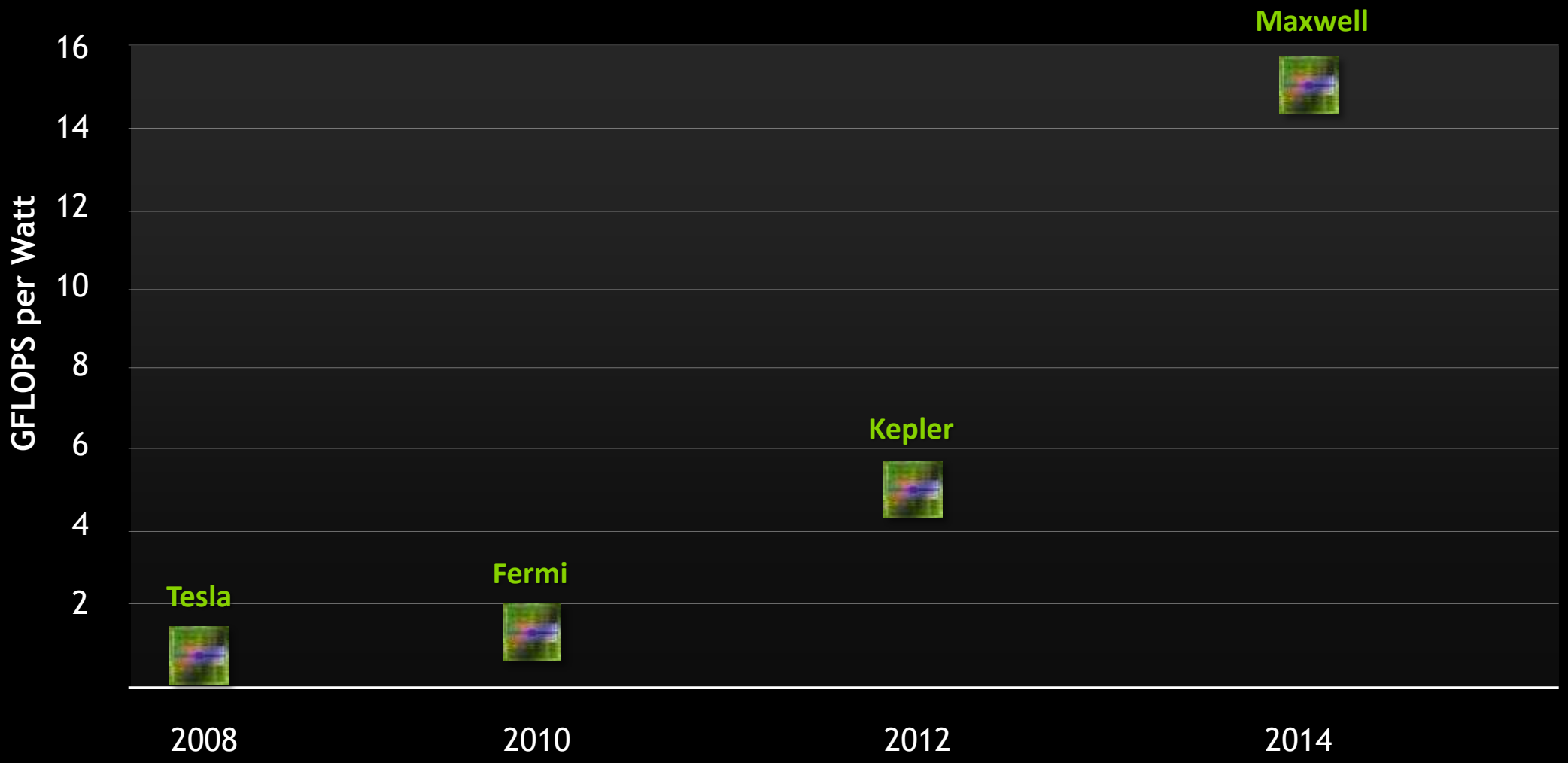
Waiting for data

Ready to be processed

Context switch

K20X: 3x Faster Than Fermi

GPUs: Two Year Heart Beat

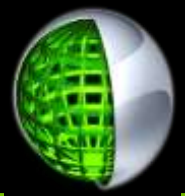


CUDA Parallel Computing Platform

Programming Approaches

- Libraries — "Drop-in" Acceleration
- OpenACC Directives — Easily Accelerate Apps
- Programming Languages — Maximum Flexibility

Development Environment



Nsight IDE
Linux, Mac and Windows
GPU Debugging and Profiling

CUDA-GDB debugger
NVIDIA Visual Profiler

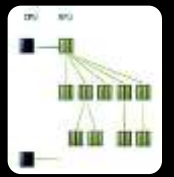
Open Compiler Tool Chain

Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

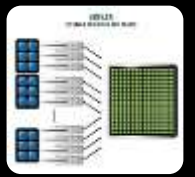
Hardware Capabilities

SMX

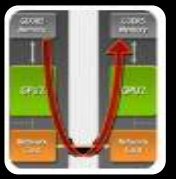
Dynamic Parallelism



HyperQ



GPUDirect



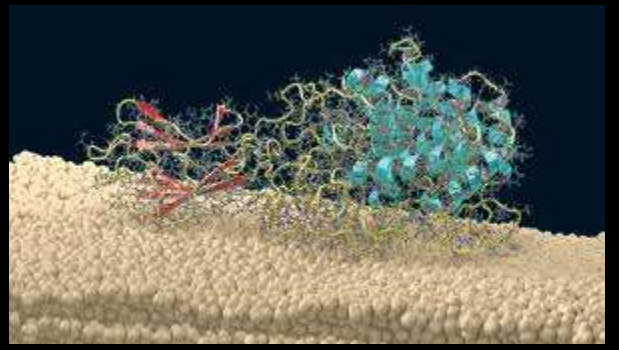
Getting Started with CUDA

GPU Accelerated Science Applications

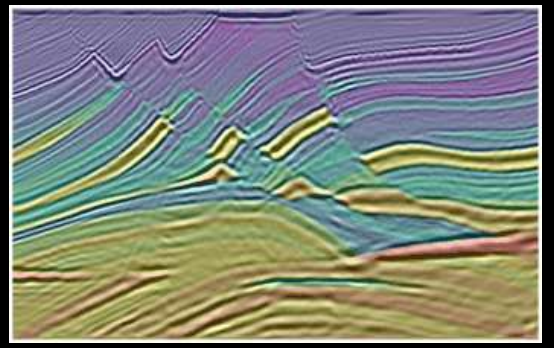
Over 110+ Accelerated science apps in our catalog. Just a few:

AMBER

GROMACS



LAMMPS



Tsunami RTM

www.nvidia.com/teslaapps

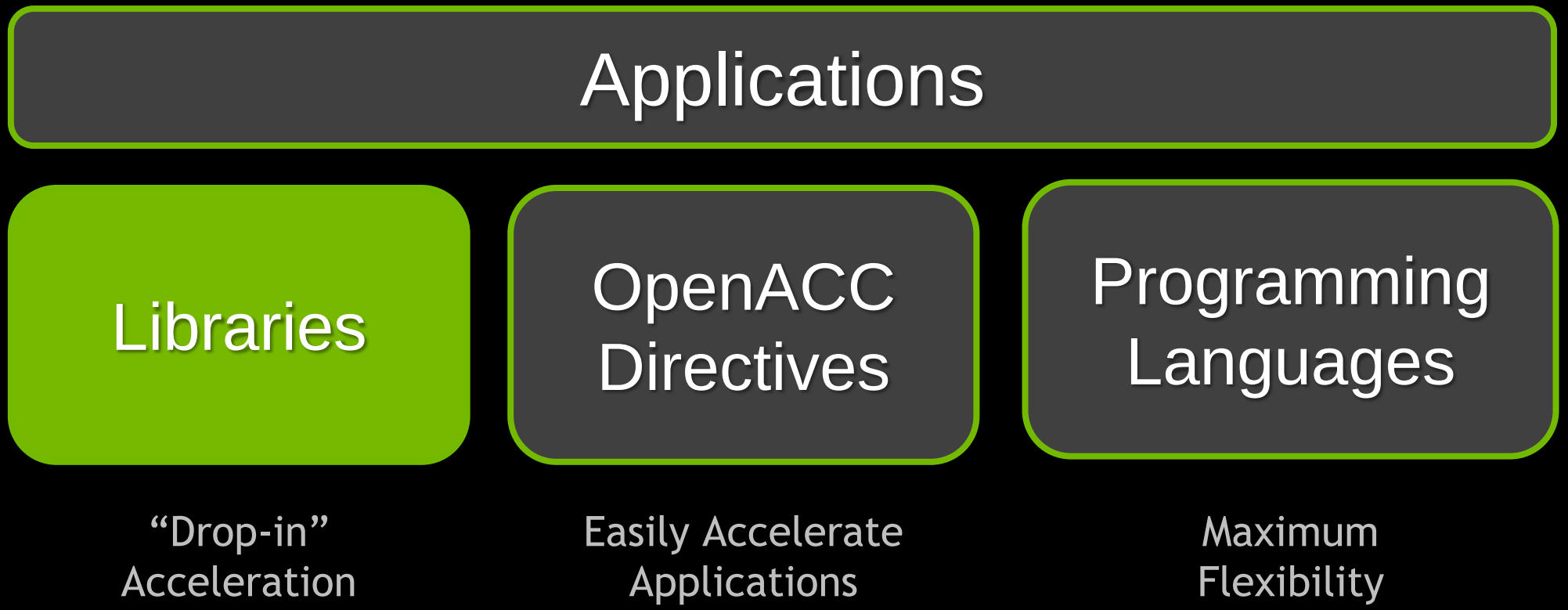
NWChem

GPU Accelerated Workstation Applications

Fifty accelerated workstation apps in our catalog. Just a few:

www.nvidia.com/object/gpu-accelerated-applications.html

3 Ways to Accelerate Applications



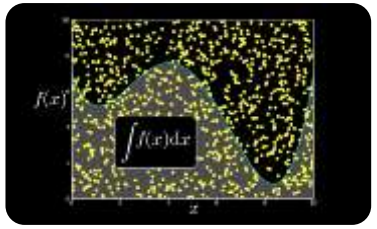
Why should you use libraries?

- No need to reinvent the wheel
  - Implement complex algorithms
  - Deal with details of the platform
- High Performance
  - Layers of optimizations
  - In depth knowledge of architecture
- Low Maintenance
  - Rigorous testing/quality-assurance
  - Have someone to file bugs against

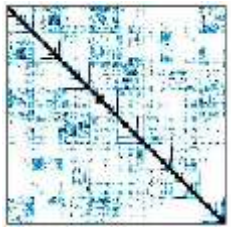
GPU Accelerated Libraries

"Drop-in" Acceleration for your Applications

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

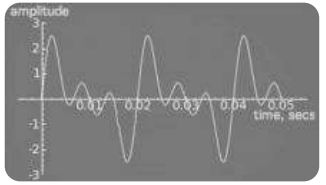
CULA tools

GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

Thrust

C++ STL Features for CUDA

Explore the CUDA (Libraries) Ecosystem

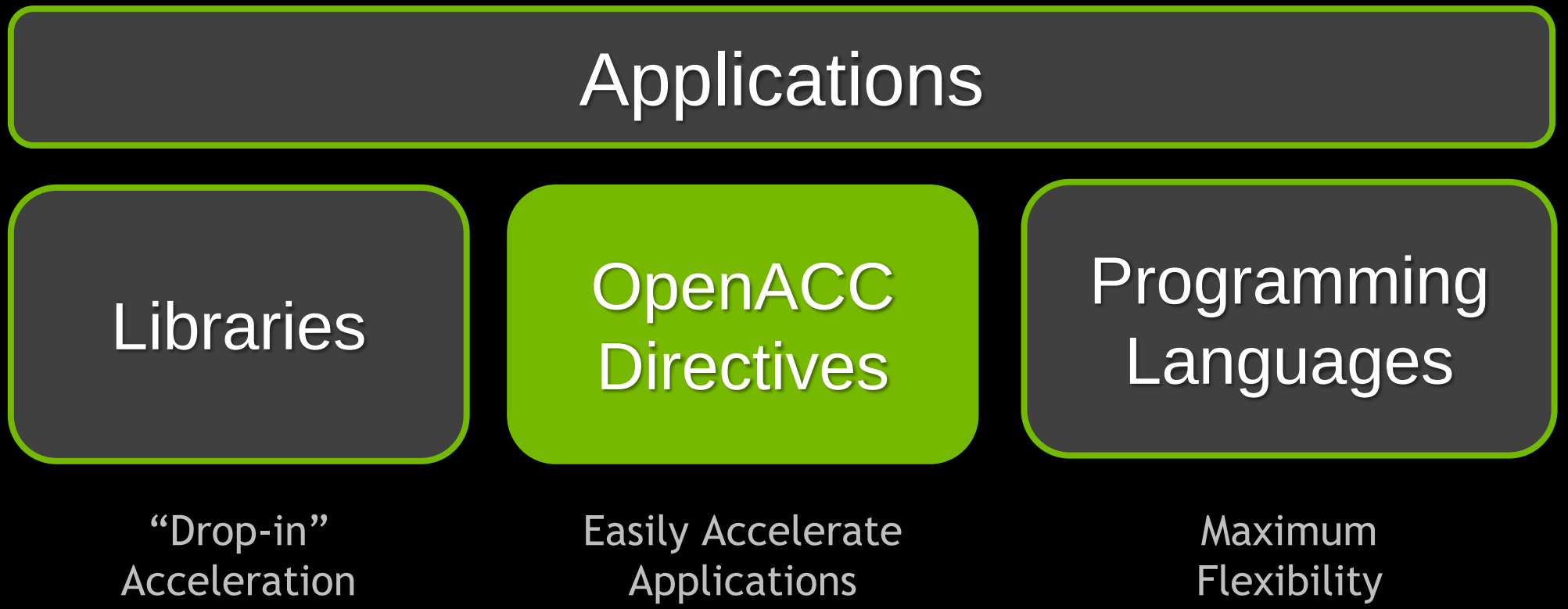
- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

  developer.nvidia.com/cuda-tools-ecosystem

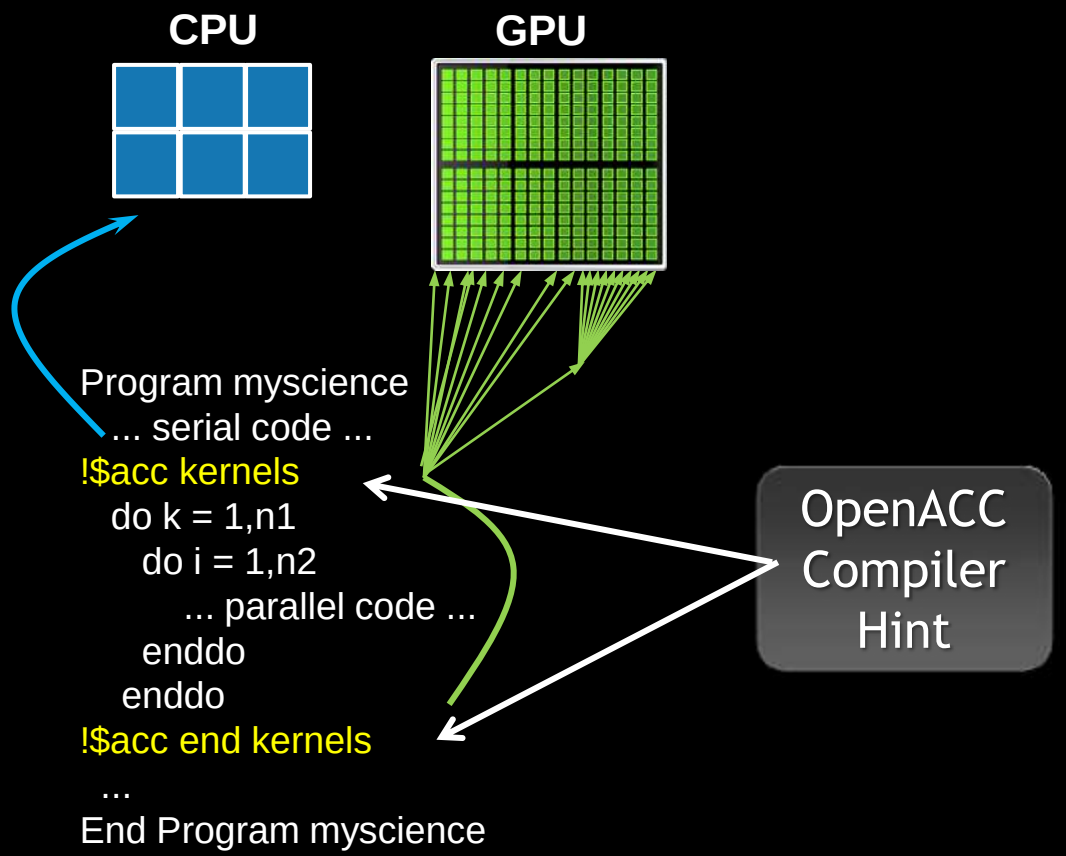
- Watch GTC On Demand Talks

  www.gputechconf.com/gtcnew/on-demand-gtc.php

3 Ways to Accelerate Applications



OpenACC Directives



Your original Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

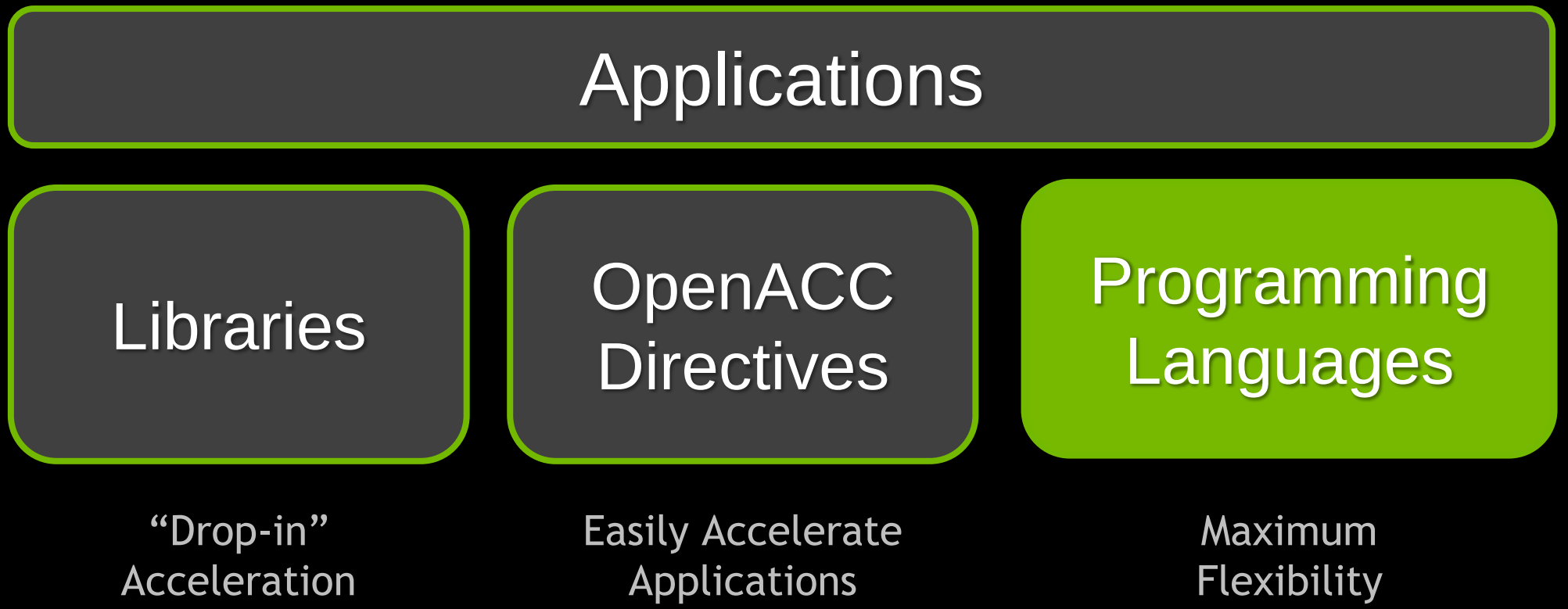
Works on many-core GPUs & multicore CPUs

Come to Paul's talk after lunch

OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online
- openacc.org
- OpenACC 2.0 Specification recently announced
  - Available for public comment
- Implementations available from PGI, Cray, and CAPS

3 Ways to Accelerate Applications



GPU Programming Languages

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	OpenACC, CUDA Fortran
C	OpenACC, CUDA C
C++	OpenACC, Thrust, CUDA C++
Come to Amr's talk after lunch — Python	PyCUDA, Copperhead, NumbaPro, …Anaconda
F#	Alea.cuBase

Programming a CUDA Language

- CUDA C/C++
  - Based on industry-standard C/C++
  - Small set of extensions to enable heterogeneous programming
  - Straightforward APIs to manage devices, memory etc.

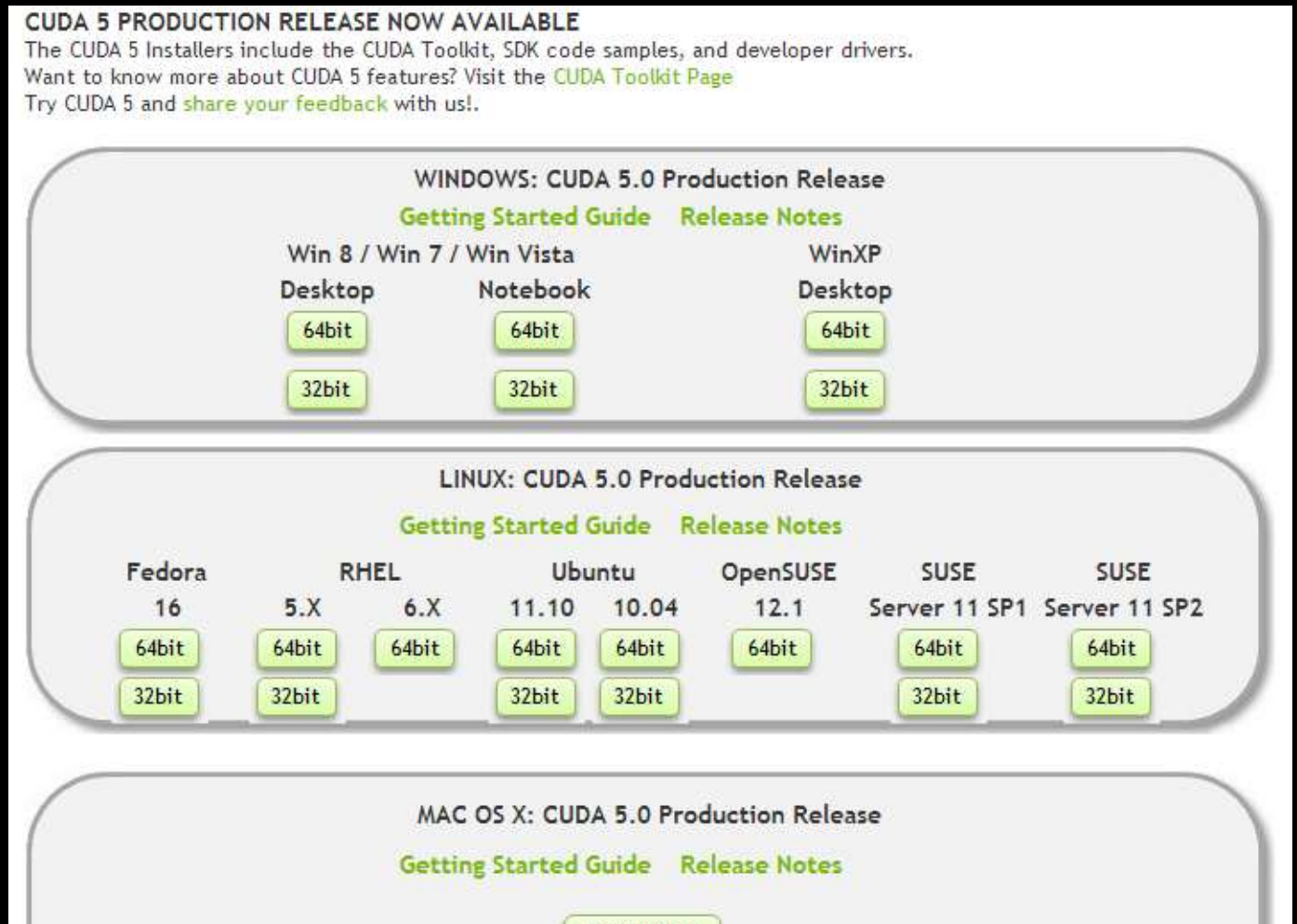
- We run a half-day to full day CUDA EASY Workshop that introduces CUDA C/C++ and OpenACC (for C)
  - Fully interactive, with hands-on demos
  - More details on this at the end!

Prerequisites

- You (probably) need experience with C or C++
  - Our CUDA EASY Workshops assume ZERO actual C/C++ experience, but a working understanding of programming (e.g. MATLAB, R, Mathematica)
- You don't need GPU experience
- You don't need parallel programming experience
- You don't need graphics experience

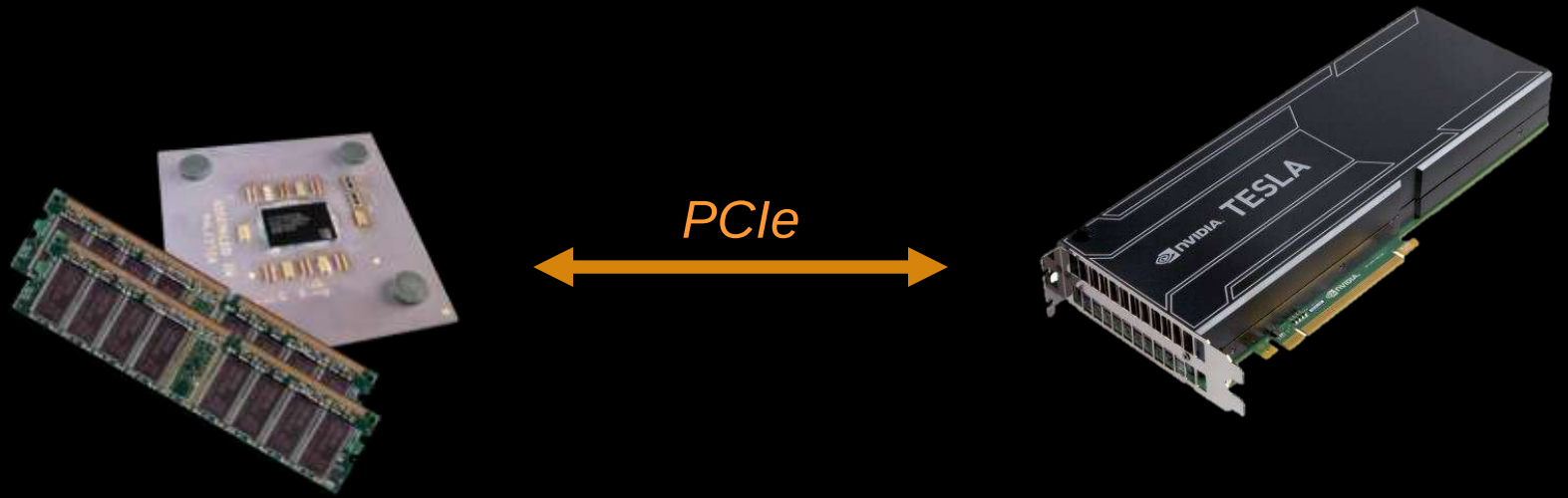
CUDA 5 Toolkit and SDK

www.nvidia.com/getcuda



Heterogeneous Computing

- Terminology:
  - *Host* The CPU and its memory (host memory)
  - *Device* The GPU and its memory (device memory)



Vector Addition - CUDA C

Standard C Code

```
void vecAddCPU (int n,
                float *a,
                float *b,
                float *c)
{

  for (int i = 0; i < n; ++i)
    c[i] = a[i] + b[i];
}
...
// Perform vecAddCPU on 1M elements
vecAddCPU (4096*256, a, b, c);
```

CUDA C Code

```
__global__
void vecAddGPU (int n,
                float *a,
                float *b,
                float *c)
{
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) c[i] = a[i] + b[i];
}
...
// Perform vecAddGPU on 1M elements
vecAddGPU<<<4096,256>>>(n, a, b, c);
```

Parallelism on a GPU - CUDA Blocks

- A function which runs on a GPU is called a "kernel"
- Each parallel invocation of a function running on the GPU is called a "block"

Grid0

- A block can identify itself by reading blockIdx.x
  - In the above example, gridDim.x = N

Parallelism on a GPU - CUDA Threads

- Each block is then broken up into “threads”

Block0


threadIdx.x = 0


threadIdx.x = 1


threadIdx.x = 2

. . .

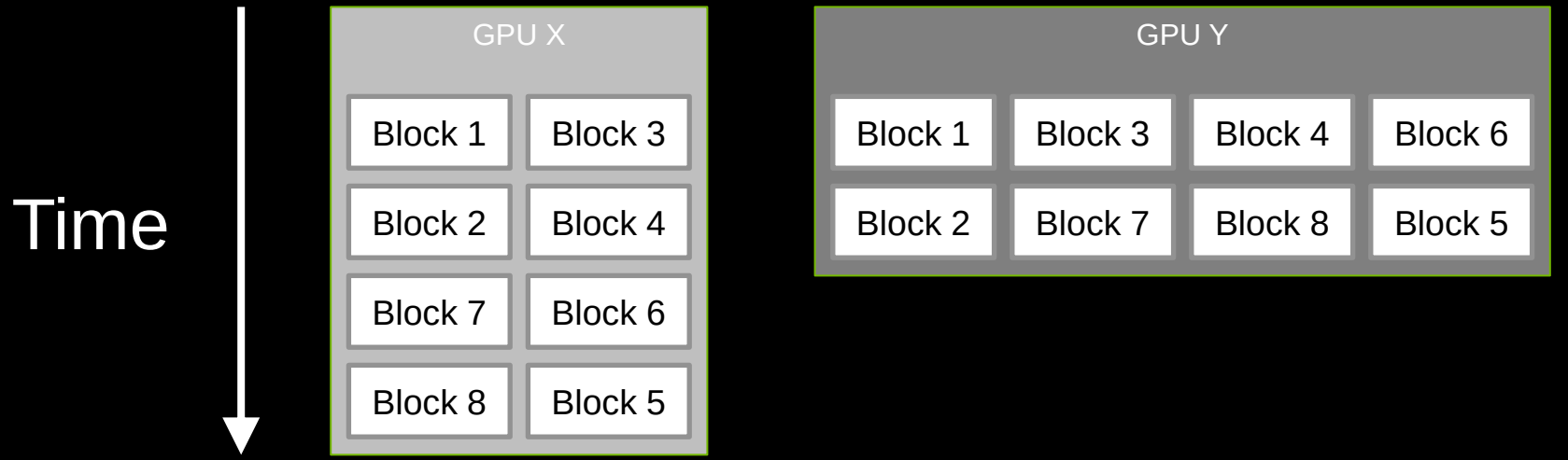

threadIdx.x = M - 1

- A thread can identify itself by reading threadIdx.x
- The total number of threads per block can be read with blockDim.x
  - In the above example blockDim.x = M

Why threads and blocks?

- Threads within a block can
  - Communicate very quickly (share memory)
  - Synchronize (wait for all threads to catch up)

Why threads and blocks?



- **Why break up into blocks?**
  - **A block cannot be broken up among multiple SMs (streaming multiprocessors), and you want to keep all SMs busy.**
  - **Allows the HW to scale the number of blocks running in parallel based on GPU capability**

Controlling Parallelism in Kernel

```
vecAddGPU<<<1,1024>>>(a, b, c, n)
```

Triple chevron contains the "kernel launch parameters"

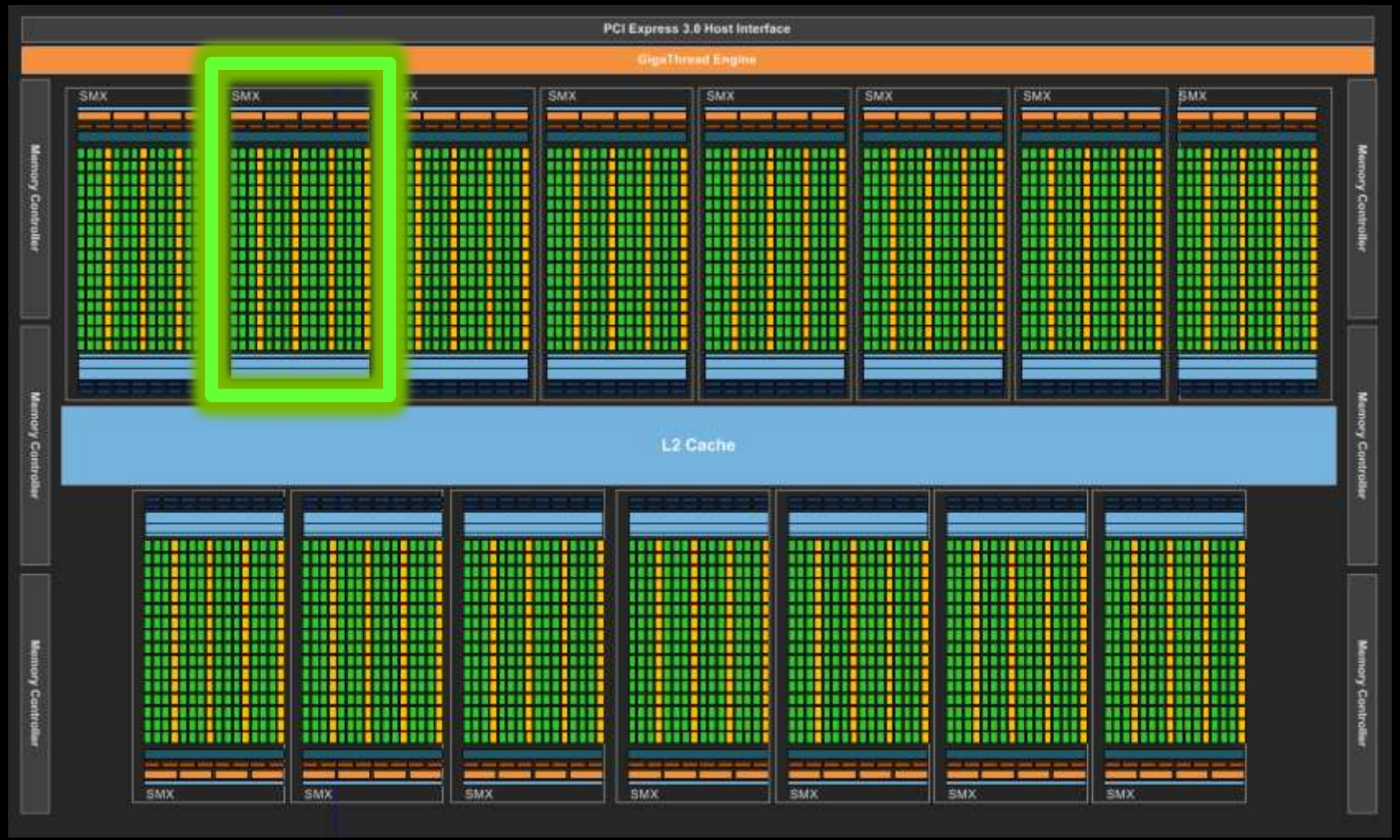
2nd parameter defines number of threads per block (hamsters per cage) to populate

1st parameter defines number of blocks (hamster cages) to use

There is a hardware limit to these. GPU architecture dependent

```
vecAddGPU<<<1024,1024>>>(a, b, c, n)
vecAddGPU<<<2048,512>>>(a, b, c, n)
vecAddGPU<<<4096,256>>>(a, b, c, n)
```

Kepler GK110 Block Diagram



VecAdd CPU Function


```
void vecAddCPU (int n, float *a, float *b, float *c) {

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

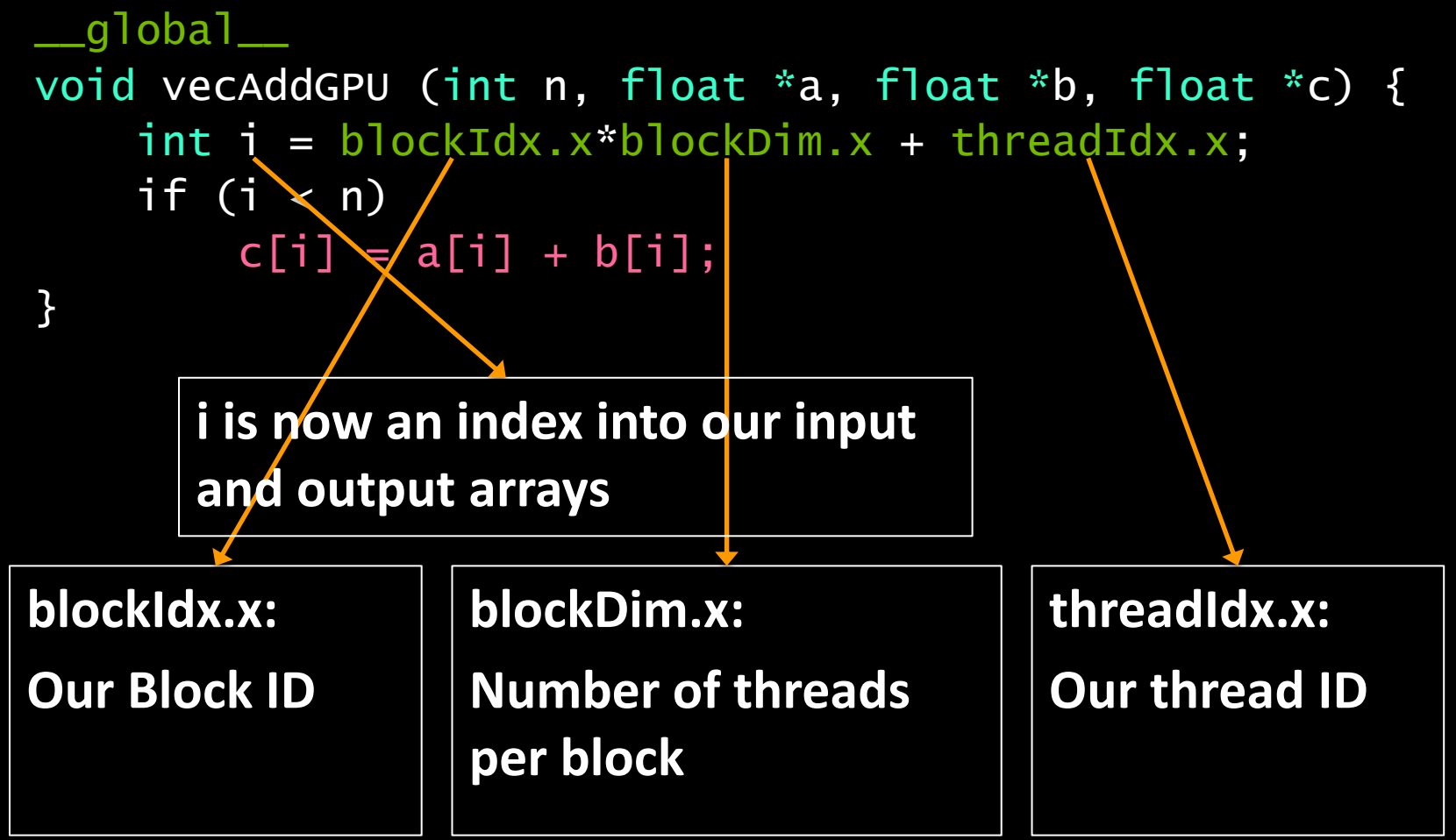
VecAdd GPU Kernel


```
__global__
void vecAddGPU (int n, float *a, float *b, float *c) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}
```

VecAdd GPU Kernel

```
__global__
void vecAddGPU (int n, float *a, float *b, float *c) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}
```

VecAdd GPU Kernel



VecAdd GPU Kernel - Data Accesses


```
__global__
void vecAddGPU (int n, float *a, float *b, float *c) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}
```

- **Let's work with 30 data elements**
  - **Broken into 3 blocks, with 10 threads per block**
- So, blockDim.x = 10

VecAdd GPU Kernel - Data Accesses


```
__global__
void vecAddGPU (int n, float *a, float *b, float *c) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}
```

10 threads (hamsters) each with a different

- For blockIdx.x = 0
  - i = 0 * 10 + threadIdx.x = {0,1,2,3,4,5,6,7,8,9}
- For blockIdx.x = 1
  - i = 1 * 10 + threadIdx.x = {10,11,12,13,14,15,16,17,18,19}
- For blockIdx.x = 2
  - i = 2 * 10 + threadIdx.x = {20,21,22,23,24,25,26,27,28,29}

VecAdd More Efficiently… with Thrust

High-level

Productive

Parallel analogue to STL

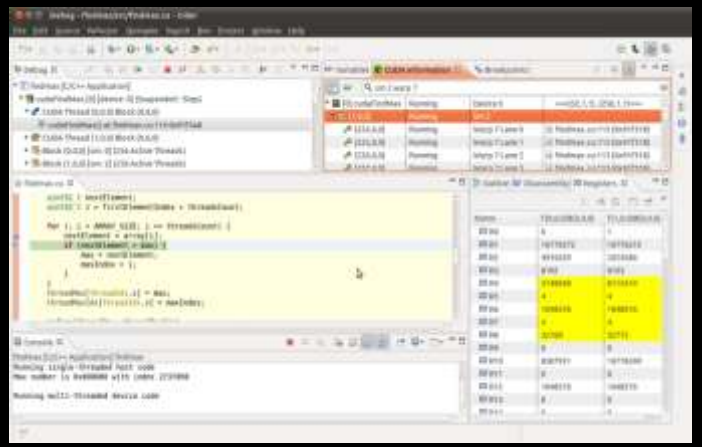
Performance-portable

Come to Luke’s talk after lunch

How to get started?

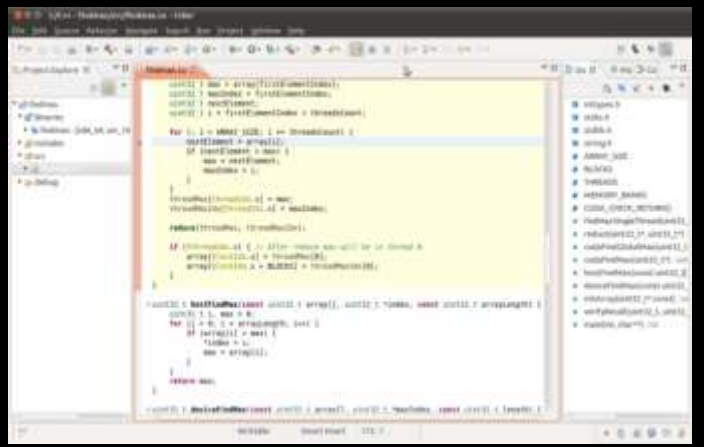
- Profile your existing CPU code
  - Look for the time hogs!
  - Is it a good candidate?
    - Nested loops, lots of parallelism, data independence..
- Don’t have to port the whole code at once!
  - Port in steps
  - Analyze, Parallelize, Optimize, Deploy!

NVIDIA® Nsight™ Eclipse Edition for Linux and MacOS



CUDA-Aware Editor

- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs



Nsight Debugger

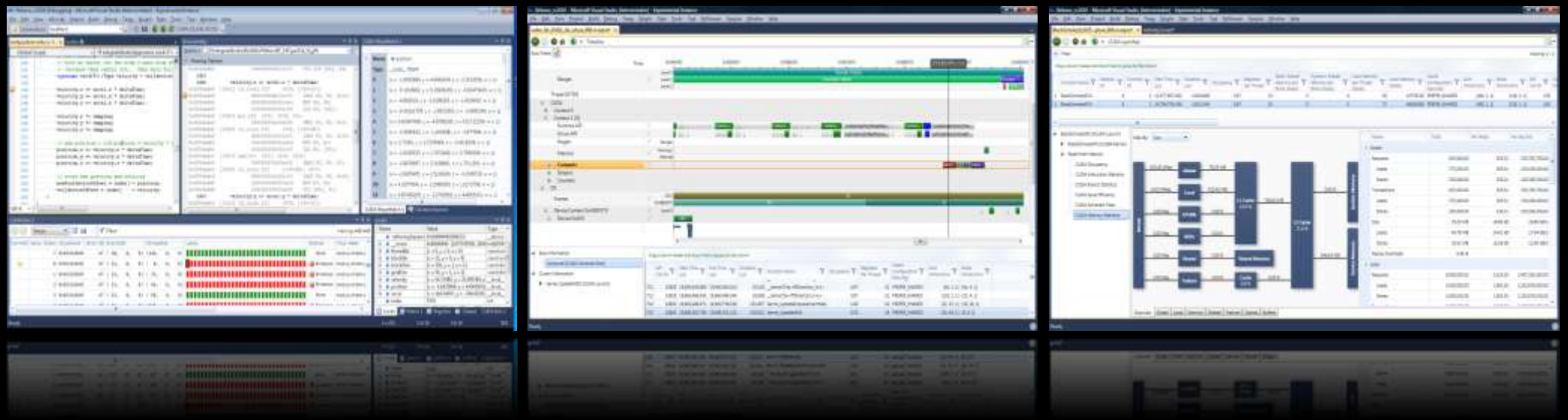
- Simultaneously debug CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging

Nsight Profiler

- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

developer.nvidia.com/nsight

NVIDIA® Nsight™, Visual Studio Ed.



CUDA Debugger

- Debug CUDA kernels directly on GPU hardware
- Examine thousands of threads executing in parallel
- Use on-target conditional breakpoints to locate errors

CUDA Memory Checker

- Enables precise error detection

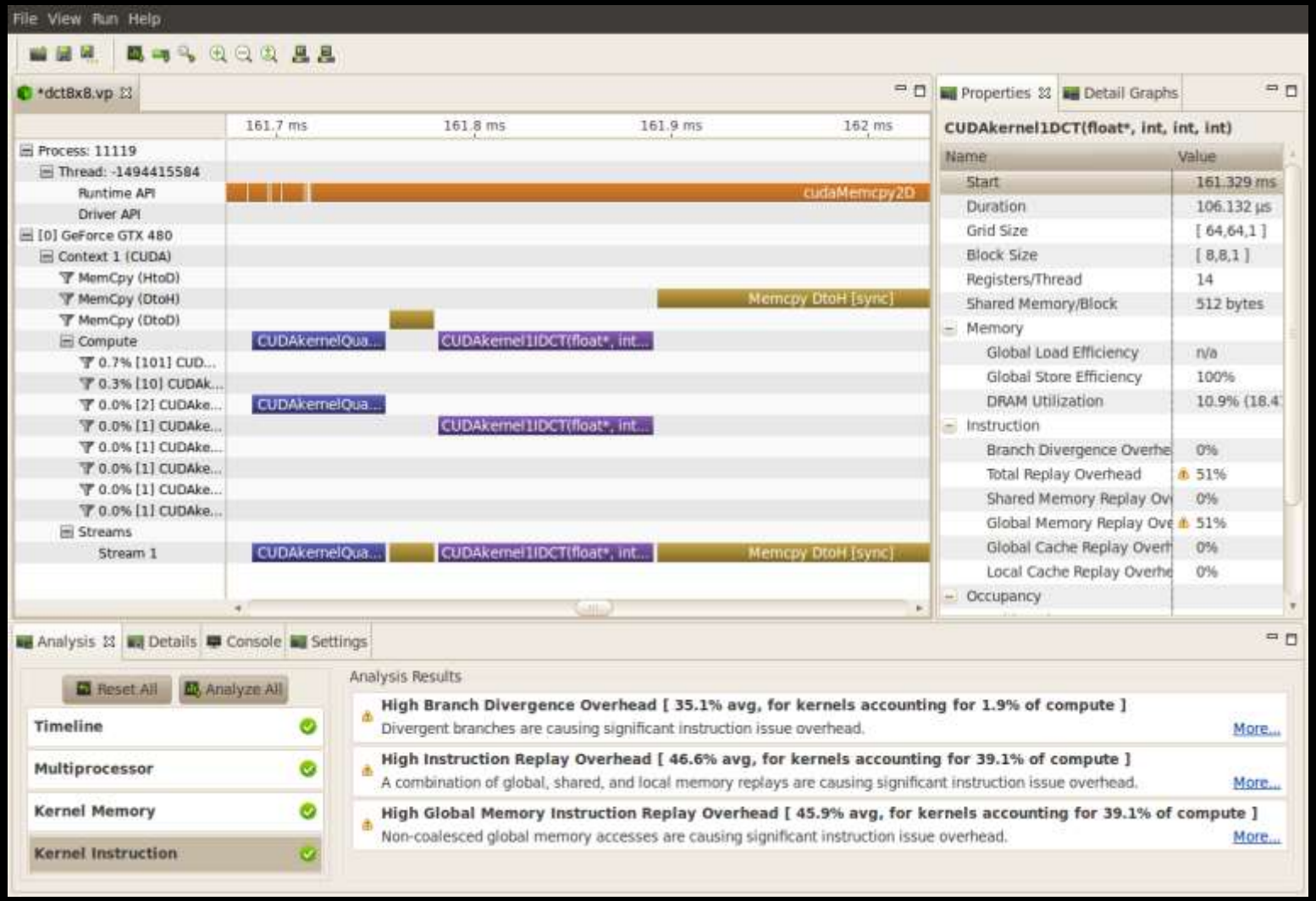
System Trace

- Review CUDA activities across CPU and GPU
- Perform deep kernel analysis to detect factors limiting maximum performance

CUDA Profiler

- Advanced experiments to measure memory utilization, instruction throughput and stalls

NVIDIA Visual Profiler



Links to get started

- Get CUDA: www.nvidia.com/getcuda
- Nsight IDE: www.nvidia.com/nsight
- Programming Guide/Best Practices…
  - docs.nvidia.com
- Questions:
  - NVIDIA Developer forums devtalk.nvidia.com
  - Search or ask on www.stackoverflow.com/tags/cuda
- General: www.nvidia.com/cudazone

Local Meetup Group

Monthly get-togethers

Talks / Guest Speakers

Prizes

www.meetup.com/Melbourne-GPU-Users/

m.wang@unimelb.edu.au

Last of all, and BEST of all…

Sign up for the next CUDA EASY Workshop, happening at:

Swinburne University

Monash University

University of Melbourne

…your Uni…

…your company…

FREE!

m.wang@unimelb.edu.au
mignonep@unimelb.edu.au