

Introduction to CUDA C/C++

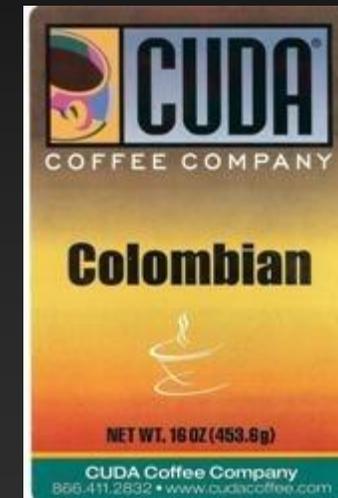
Mark Ebersole, NVIDIA
CUDA Educator



What is CUDA?



- Programming language?
- Compiler?
- Classic car?
- Beer?
- Coffee?



CUDA Parallel Computing Platform

www.nvidia.com/getcuda



Programming Approaches

Libraries

“Drop-in” Acceleration

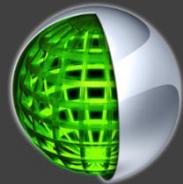
OpenACC Directives

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



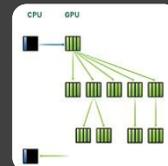
Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

Hardware Capabilities

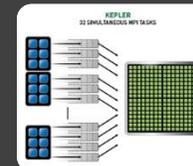
SMX



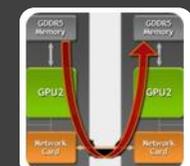
Dynamic Parallelism



HyperQ



GPUDirect



Getting Started with CUDA



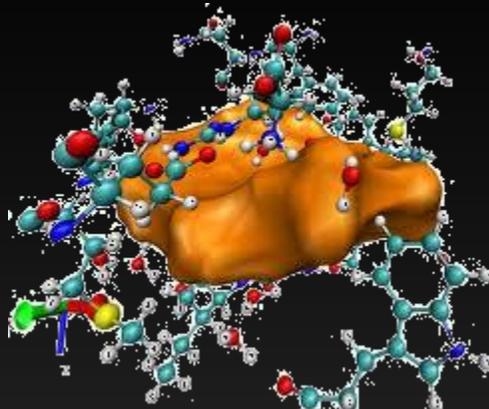
GPU Accelerated Science Applications



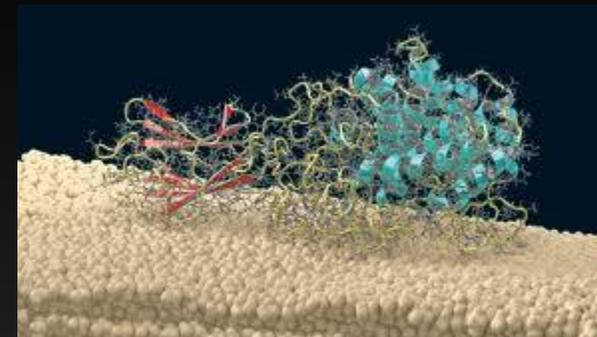
Over 145+ Accelerated science apps in our catalog. Just a few:



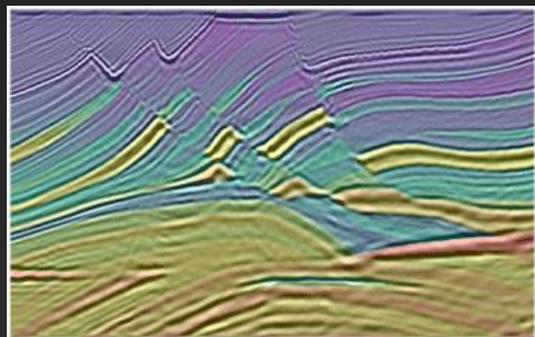
AMBER



GROMACS

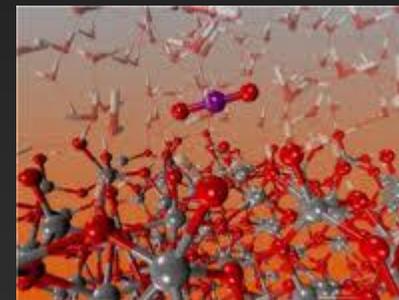


LAMMPS



Tsunami RTM

www.nvidia.com/teslaapps



NWChem

GPU Accelerated Workstation Applications



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

AUTODESK AUTOCAD 2011

A 3D CAD model of a mechanical component, possibly a turbine or engine part, rendered with a semi-transparent green wireframe overlay to show internal structure.

Maximize your productivity.
[learn more >](#)

AUTODESK INVENTOR 2012

A 3D CAD model of a complex mechanical assembly, possibly a turbine or engine, rendered in a yellowish-gold color scheme.

Maximize your design potential.
[learn more >](#)

DASSAULT SYSTEMES CATIA

A 3D CAD model of a sleek, modern car, rendered in a dark blue color with red highlights on the rear lights.

The proven combination for perfect designs. [learn more >](#)

DASSAULT SYSTEMES SOLIDWORKS

A 3D CAD model of a car wheel and suspension system, rendered in a dark blue color with red highlights on the brake caliper.

Examine every aspect of your model.
[learn more >](#)

SIEMENS NX

A 3D CAD model of a complex mechanical assembly, possibly a turbine or engine, rendered in a dark blue color with red highlights on the internal components.

The right solutions. The right decision.
[learn more >](#)

PTC CREO PARAMETRIC 2.0

A 3D CAD model of chess pieces, rendered in a white color with red highlights on the pieces, set against a red and white checkered board.

Experience a new level of performance
[learn more >](#)

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

3 Ways to Accelerate Applications



Applications

Libraries

“Drop-in”
Acceleration

OpenACC
Directives

Easily Accelerate
Applications

Programming
Languages

Maximum
Flexibility

3 Ways to Accelerate Applications



Applications

Libraries

“Drop-in”
Acceleration

OpenACC
Directives

Easily Accelerate
Applications

Programming
Languages

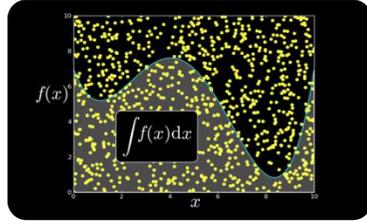
Maximum
Flexibility

GPU Accelerated Libraries

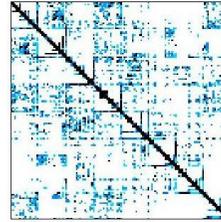
“Drop-in” Acceleration for your Applications



NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE



NVIDIA NPP



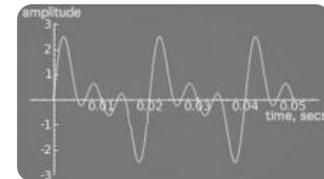
Vector Signal
Image Processing



GPU Accelerated
Linear Algebra



Matrix Algebra on
GPU and Multicore



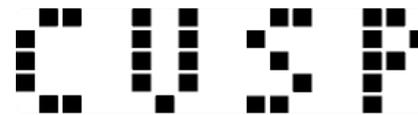
NVIDIA cuFFT



IMSL Library



ArrayFire Matrix
Computations



Sparse Linear
Algebra



C++ STL Features
for CUDA



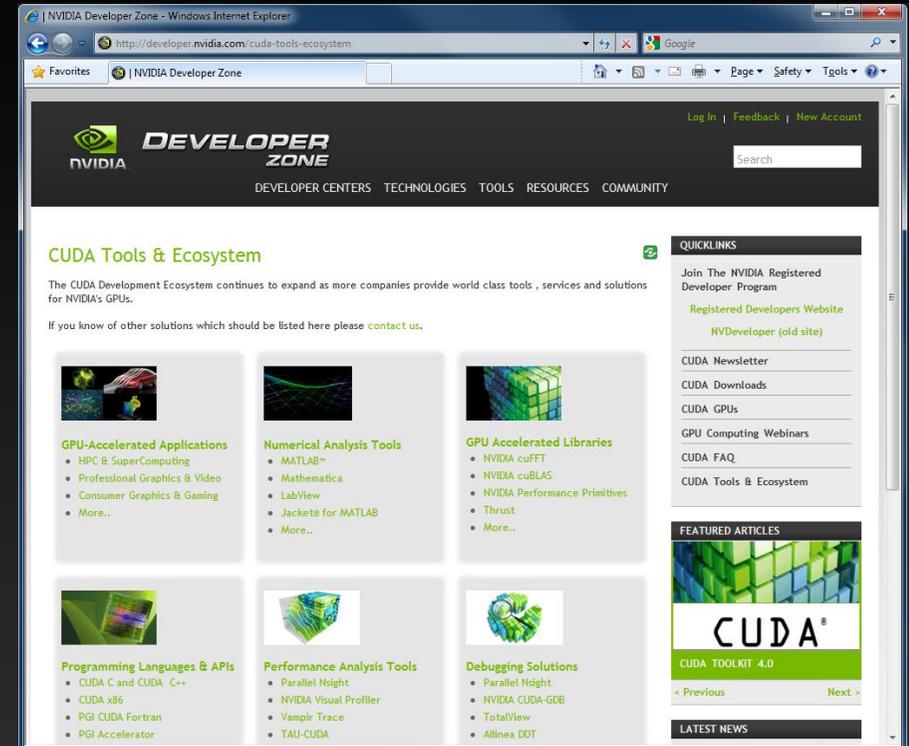
Explore the CUDA (Libraries) Ecosystem



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

developer.nvidia.com/cuda-tools-ecosystem

- Watch past GTC library talks



3 Ways to Accelerate Applications



Applications

Libraries

OpenACC
Directives

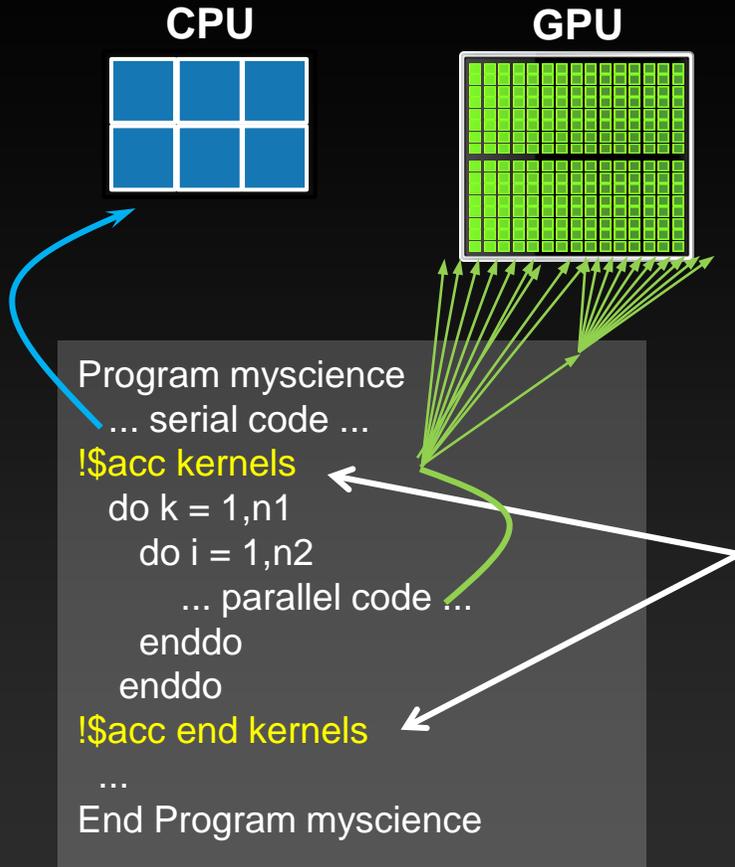
Programming
Languages

“Drop-in”
Acceleration

Easily Accelerate
Applications

Maximum
Flexibility

OpenACC Directives



Your original
Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs &
multicore CPUs

OpenACC Specification and Website



- Full OpenACC 1.0 Specification available online

www.openacc.org

- OpenACC 2.0 Specification just announced
- Implementations available now from PGI, Cray (beta), and CAPS

The OpenACC™ API QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

CAPS

CRAY
THE SUPERCOMPUTER COMPANY

NVIDIA.

PGI

Version 1.0, November 2011

Start Now with OpenACC Directives



Sign up for a **free trial** of the directives compiler now!

Free trial license to PGI Accelerator

Tools for quick ramp

www.nvidia.com/gpudirectives



TESLA

NVIDIA Home > Products > High Performance Computing > OpenACC GPU Directives

GPU COMPUTING SOLUTIONS

- Main
- What is GPU Computing?
- Why Choose Tesla
- Industry Software Solutions
- Tesla Workstation Solutions
- Tesla Data Center Solutions
- Tesla Bio Workbench
- Where to Buy
- Contact US
- Sign up for Tesla Alerts
- Fermi GPU Computing Architecture

SOFTWARE AND HARDWARE INFO

- Tesla Product Literature
- Tesla Software Features
- Software Development Tools
- CUDA Training and Consulting Services
- GPU Cloud Computing Service Providers
- OpenACC GPU Directives

Accelerate Your Scientific Code with OpenACC
The Open Standard for GPU Accelerator Directives

Thousands of cores working for you.
Based on the [OpenACC](#) standard, GPU directives are the easy, proven way to accelerate your scientific or industrial code. With GPU directives, you can accelerate your code by simply inserting compiler hints into your code and the compiler will automatically map compute-intensive portions of your code to the GPU. Here's an example of how easy a single directive hint can accelerate the calculation of pi. With GPU directives, you can get started and see results in the same afternoon.

```
#include <stdio.h>
#define N 10000
int main(void) {
    double pi = 0.0f; long i;
    #pragma acc region for
    for (i=0; i<N; i++)
    {
        double t= (double)((i+0.5)/N);
        pi +=4.0/(1.0+t*t);
    }
    printf("pi=%f\n",pi/N);
    return 0;
}
```

"I have written micron (written in Fortran 90) properties of two and dimensional magnetic directives approach error perform my computation which resulted in a speedup (more than 20 computation." [Learn more](#)

Professor M. Amin Kay
University of Houston

"The PGI compiler is not just how powerful it is software we are writing times faster on the NV are very pleased and excited future uses. It's like on supercomputer." [Learn more](#)

Dr. Kerry Black
University of Melbourne

By starting with a free, 30-day trial of PGI directives today, you are working on the technology that is the foundation of the OpenACC directives standard. OpenACC is:

3 Ways to Accelerate Applications



Applications

Libraries

“Drop-in”
Acceleration

OpenACC
Directives

Easily Accelerate
Applications

Programming
Languages

Maximum
Flexibility

GPU Programming Languages



Numerical analytics ▶

MATLAB, Mathematica, LabVIEW

Fortran ▶

OpenACC, CUDA Fortran

C ▶

OpenACC, CUDA C

C++ ▶

Thrust, CUDA C++

Python ▶

PyCUDA, Copperhead

C# ▶

GPU.NET

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.
- This session introduces CUDA C/C++

Prerequisites



- You (probably) need experience with C or C++
- 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



CUDA 5 PRODUCTION RELEASE NOW AVAILABLE

The CUDA 5 Installers include the CUDA Toolkit, SDK code samples, and developer drivers.

Want to know more about CUDA 5 features? Visit the [CUDA Toolkit Page](#)

Try CUDA 5 and [share your feedback](#) with us!.

WINDOWS: CUDA 5.0 Production Release

[Getting Started Guide](#) [Release Notes](#)

Win 8 / Win 7 / Win Vista

WinXP

Desktop

Notebook

Desktop

64bit

64bit

64bit

32bit

32bit

32bit

LINUX: CUDA 5.0 Production Release

[Getting Started Guide](#) [Release Notes](#)

Fedora

RHEL

Ubuntu

OpenSUSE

SUSE

SUSE

16

5.X

6.X

11.10

10.04

12.1

Server 11 SP1

Server 11 SP2

64bit

64bit

64bit

64bit

64bit

64bit

64bit

64bit

32bit

32bit

32bit

32bit

32bit

32bit

MAC OS X: CUDA 5.0 Production Release

[Getting Started Guide](#) [Release Notes](#)

DOWNLOAD

CONCEPTS

Heterogeneous Computing

Blocks

Threads

Indexing

SAXPY



Standard C Code

```
void saxpy(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

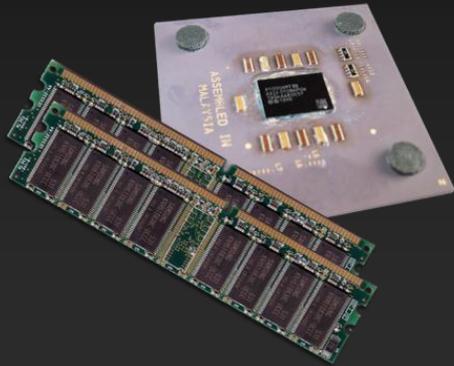
int N = 1<<20;

// Perform SAXPY on 1M elements
saxpy(N, 2.0f, x, y);
```

Heterogeneous Computing



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



Host



Device

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”

Parallelism on a GPU - CUDA Blocks



el”

the GPU is called a

= BLOCK

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



blockIdx.x = 0



blockIdx.x = 1



blockIdx.x = 2

...



blockIdx.x = N-1

- A block can identify itself by reading **blockIdx.x**

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”

Grid1



blockIdx.x = 0



blockIdx.x = 1



blockIdx.x = 2

...



blockIdx.x = W-1

- A block can identify itself by reading **blockIdx.x**

Parallelism on a GPU - CUDA Threads



- Each block is then broken up into “**threads**”

Parallelism on a GPU – CUDA Threads



“threads”

= THREAD

threadIdx.x

block can be read with blockDim.x

Parallelism on a GPU - CUDA Threads



Block

- Each block is then broken up into “threads”



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 $\text{blockDim.x} = M$

Why threads and blocks?



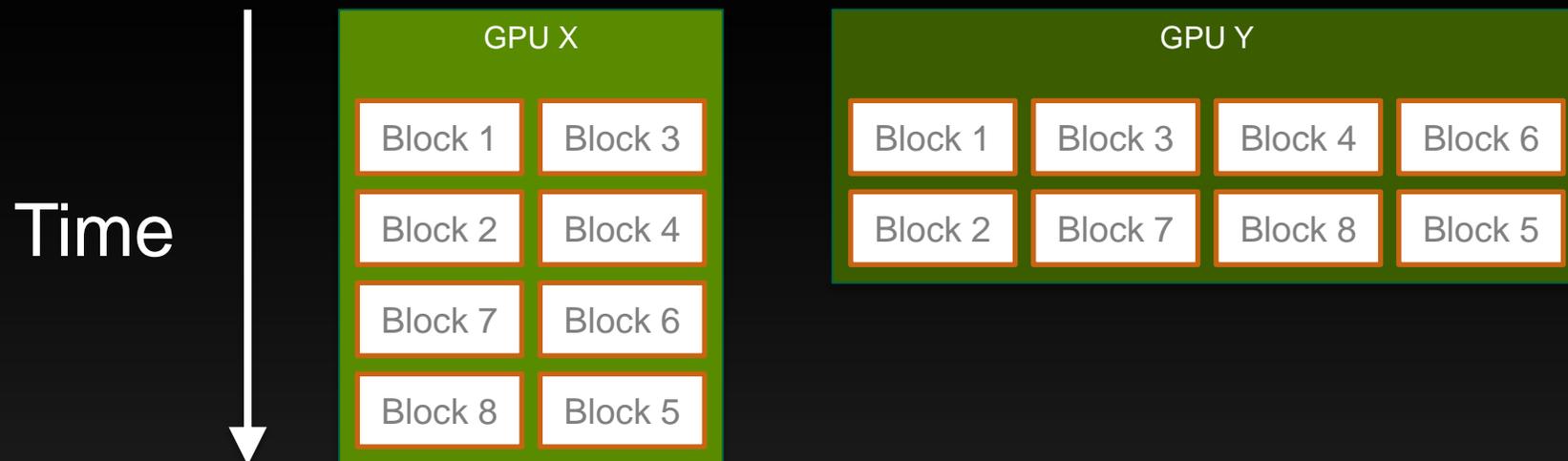
● Threads within a block can

- Communicate very quickly (share memory)
- Synchronize (wait for all threads to catch up)

● Why 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

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

Kepler GK110 Block Diagram



Hello Parallelism!



hello_parallelism Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Common Properties
- Configuration Properties
 - General
 - Debugging
 - VC++ Directories
 - CUDA C/C++
 - Common
 - Device
 - Host
 - Command Line

C interleaved in PTXAS Output	No
Code Generation	compute_20,sm_21
Generate GPU Debug Information	Yes (-G)
Generate Line Number Information	No
Max Used Register	0
Verbose PTXAS Output	No

Hello Parallelism!



kernel.cu x

(Global Scope)

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>

__global__ void hello()
{
    printf("Hello Parallelism from thread %d in block %d\n", threadIdx.x, blockIdx.x);
}

int main()
{
    hello<<<1,1>>>();
    cudaDeviceSynchronize();

    return 0;
}
```



Output

Show output from: Build

```
1>----- Build started: Project: hello_parallelism, Configuration: Debug Win32 -----
1>Build started 10/2/2012 2:21:54 PM.
1>InitializeBuildStatus:
1> Creating "Debug\hello_parallelism.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>AddCudaCompilePropsDeps:
1>Skipping target "AddCudaCompilePropsDeps" because all output files are up-to-date with respect to the input files.
1>CudaBuild:
1> Compiling CUDA source file kernel.cu...
1>
1> C:\Users\mebersole\Documents\code\Hello\hello_parallelism>"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\nvcc.exe" -ge
1> kernel.cu
1> tmpxft_00001ff0_00000000-2_kernel.cudafe1.gpu
1> tmpxft_00001ff0_00000000-7_kernel.cudafe2.gpu
1> kernel.cu
1> tmpxft_00001ff0_00000000-2_kernel.cudafe1.cpp
1> tmpxft_00001ff0_00000000-12_kernel.ii
1>ManifestResourceCompile:
1> All outputs are up-to-date.
1>Manifest:
1> All outputs are up-to-date.
1>LinkEmbedManifest:
1> All outputs are up-to-date.
1> hello_parallelism.vcxproj -> C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug\hello_parallelism.exe
1>PostBuildEvent:
1> copy "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\cuda*.dll" "C:\Users\mebersole\Documents\code\Hello\hello_parallel
1> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\cuda32_50_27.dll
1> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\cuda64_50_27.dll
1> 2 file(s) copied.
1>FinalizeBuildStatus:
1> Deleting file "Debug\hello_parallelism.unsuccessfulbuild".
1> Touching "Debug\hello_parallelism.lastbuildstate".
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:06.08
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Hello Parallelism!



```
C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug>hello_parallelism.exe  
Hello Parallelism from thread 0 in block 0  
  
C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug>█
```

Hello Parallelism!



```
int main()
{
    hello<<<1,18>>>();
    cudaDeviceSynchronize();

    return 0;
}
```

Hello Parallelism



```
C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug>hello_parallelism.exe
Hello Parallelism from thread 0 in block 0
Hello Parallelism from thread 16 in block 0
Hello Parallelism from thread 1 in block 0
Hello Parallelism from thread 2 in block 0
Hello Parallelism from thread 17 in block 0
Hello Parallelism from thread 3 in block 0
Hello Parallelism from thread 4 in block 0
Hello Parallelism from thread 5 in block 0
Hello Parallelism from thread 6 in block 0
Hello Parallelism from thread 7 in block 0
Hello Parallelism from thread 8 in block 0
Hello Parallelism from thread 9 in block 0
Hello Parallelism from thread 10 in block 0
Hello Parallelism from thread 11 in block 0
Hello Parallelism from thread 12 in block 0
Hello Parallelism from thread 13 in block 0
Hello Parallelism from thread 14 in block 0
Hello Parallelism from thread 15 in block 0

C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug>
```

Hello Parallelism!



```
int main()
{
    hello<<<2,18>>>();
    cudaDeviceSynchronize();

    return 0;
}
```

Hello

```
C:\Users\mebersole\Documents\code\Hello\hello_parallelism\Debug>hello_parallelism.exe
Hello Parallelism from thread 0 in block 0
Hello Parallelism from thread 1 in block 0
Hello Parallelism from thread 2 in block 0
Hello Parallelism from thread 16 in block 0
Hello Parallelism from thread 3 in block 0
Hello Parallelism from thread 4 in block 0
Hello Parallelism from thread 17 in block 0
Hello Parallelism from thread 5 in block 0
Hello Parallelism from thread 6 in block 0
Hello Parallelism from thread 7 in block 0
Hello Parallelism from thread 8 in block 0
Hello Parallelism from thread 9 in block 0
Hello Parallelism from thread 10 in block 0
Hello Parallelism from thread 11 in block 0
Hello Parallelism from thread 12 in block 0
Hello Parallelism from thread 13 in block 0
Hello Parallelism from thread 14 in block 0
Hello Parallelism from thread 15 in block 0
Hello Parallelism from thread 0 in block 1
Hello Parallelism from thread 16 in block 1
Hello Parallelism from thread 1 in block 1
Hello Parallelism from thread 17 in block 1
Hello Parallelism from thread 2 in block 1
Hello Parallelism from thread 3 in block 1
Hello Parallelism from thread 4 in block 1
Hello Parallelism from thread 5 in block 1
Hello Parallelism from thread 6 in block 1
Hello Parallelism from thread 7 in block 1
Hello Parallelism from thread 8 in block 1
Hello Parallelism from thread 9 in block 1
Hello Parallelism from thread 10 in block 1
Hello Parallelism from thread 11 in block 1
Hello Parallelism from thread 12 in block 1
Hello Parallelism from thread 13 in block 1
Hello Parallelism from thread 14 in block 1
Hello Parallelism from thread 15 in block 1
```



SAXPY CPU



```
void saxpy_cpu(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
```

SAXPY kernel



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

SAXPY kernel



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

blockIdx.x:
Our Block ID

blockDim.x:
Number of threads per
block

threadIdx.x:
Our thread ID

SAXPY kernel



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

i is now an index into our input and output arrays

SAXPY kernel - with data



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

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

SAXPY kernel - with data



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

10 threads (hamsters)
each with a different i

- For $\text{blockIdx.x} = 0$

- $i = 0 * 10 + \text{threadIdx.x} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- For $\text{blockIdx.x} = 1$

- $i = 1 * 10 + \text{threadIdx.x} = \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$

- For $\text{blockIdx.x} = 2$

- $i = 2 * 10 + \text{threadIdx.x} = \{20, 21, 22, 23, 24, 25, 26, 27, 28, 29\}$

Calling saxpy_gpu: main()



Standard C Code

```
#define N (2048 * 512)
int main(void) {
    float *x, *y;    // host copies

    int size = N * sizeof(float);

    // Alloc space for host copies of
    // x & y and setup input values
    x = (float *)malloc(size);
    random_floats(x, N);
    y = (float *)malloc(size);
    random_floats(y, N);
```

Parallel C Code

```
#define N (2048 * 512)
int main(void) {
    float *x, *y;    // host copies
    float *d_x, *d_y; // device copies
    int size = N * sizeof(float);

    // Alloc space for device copies
    cudaMalloc((void **)&d_x, size);
    cudaMalloc((void **)&d_y, size);

    // Alloc space for host copies of
    // x & y and setup input values
    x = (float *)malloc(size);
    random_floats(x, N);
    y = (float *)malloc(size);
    random_floats(y, N);
```

Calling saxpy_gpu: main()



Standard C Code

```
// Launch saxpy on CPU
saxpy_cpu(N, 2.0f, x, y);

// Cleanup

free(x); free(y);
return 0;
}
```

Parallel C Code

```
// Copy input to device
cudaMemcpy(d_x, &x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, &y, size, cudaMemcpyHostToDevice);

// Launch saxpy kernel on GPU
saxpy_gpu<<<4096,256>>>(N, 2.0f, d_x, d_y);

// Copy result back to host
cudaMemcpy(&y, d_y, size, cudaMemcpyDeviceToHost);

// Cleanup
cudaFree(d_x); cudaFree(d_y);
free(x); free(y);
return 0;
}
```

CUDA C++: Develop Generic Parallel Code



CUDA C++ features enable sophisticated and flexible applications and middleware

Class hierarchies

__device__ methods

Templates

Operator overloading

Functors (function objects)

Device-side new/delete

More...

```
template <typename T>
struct Functor {
    __device__ Functor(_a) : a(_a) {}
    __device__ T operator(T x) { return a*x; }
    T a;
}

template <typename T, typename Oper>
__global__ void kernel(T *output, int n) {
    Oper op(3.7);
    output = new T[n]; // dynamic allocation
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        output[i] = op(i); // apply functor
}
```

Thrust C++ Template Library



Serial C++ Code

with STL and Boost

```
int N = 1<<20;
std::vector<float> x(N), y(N);

...

// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
               2.0f * _1 + _2);
```

www.boost.org/libs/lambda

Parallel C++ Code

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);

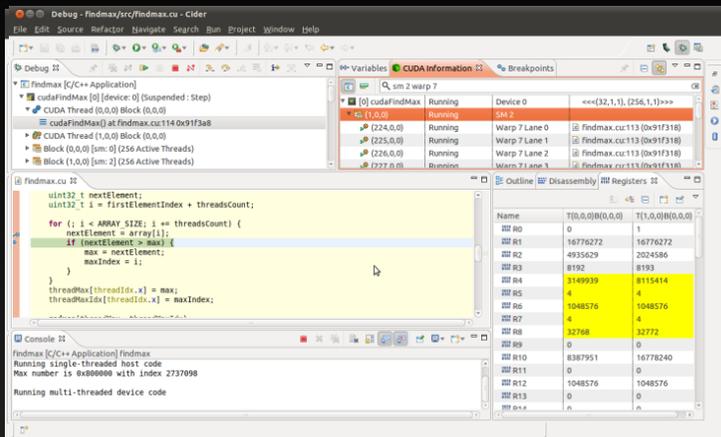
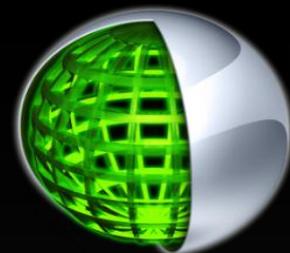
...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;

// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                  d_y.begin(), d_y.begin(),
                  2.0f * _1 + _2);
```

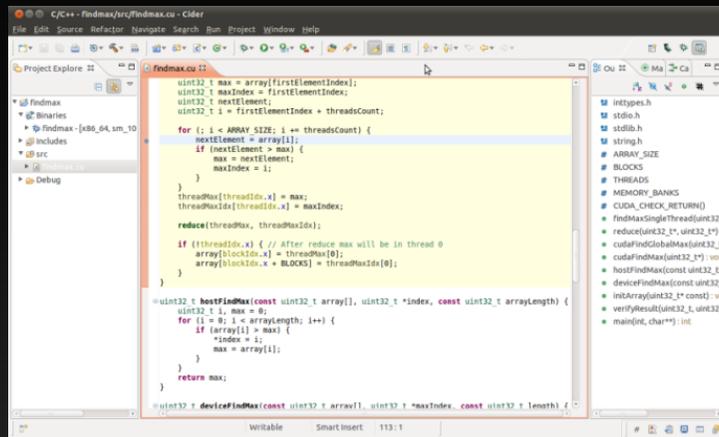
thrust.github.com

NVIDIA[®] Nsight[™], Eclipse Edition for Linux and MacOS



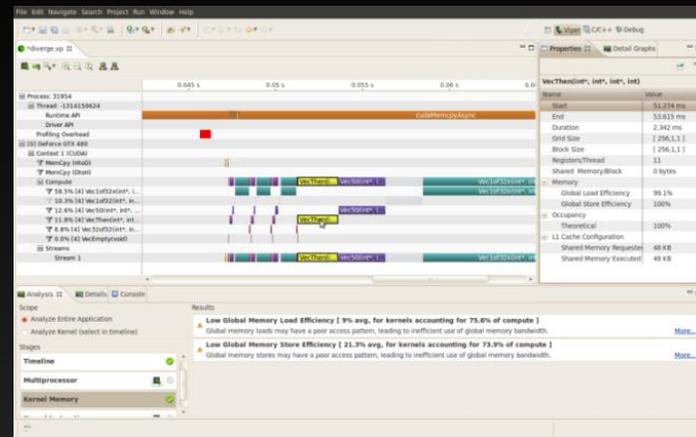
CUDA-Aware Editor

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



Nsight Debugger

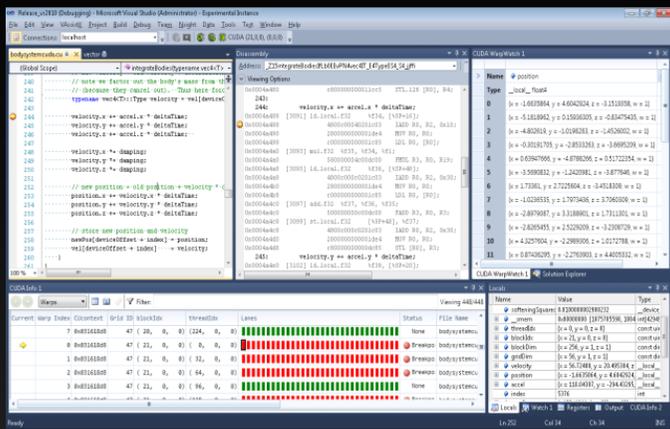
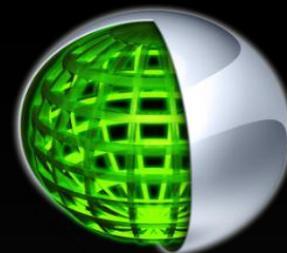
- Simultaneously debug CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging



Nsight Profiler

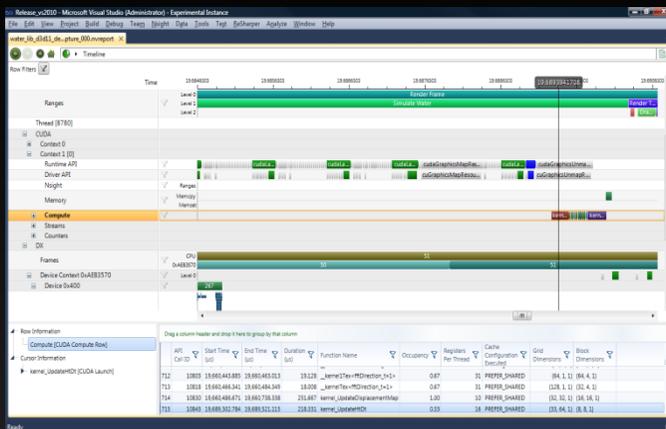
- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

NVIDIA[®] Nsight[™] Visual Studio Ed.



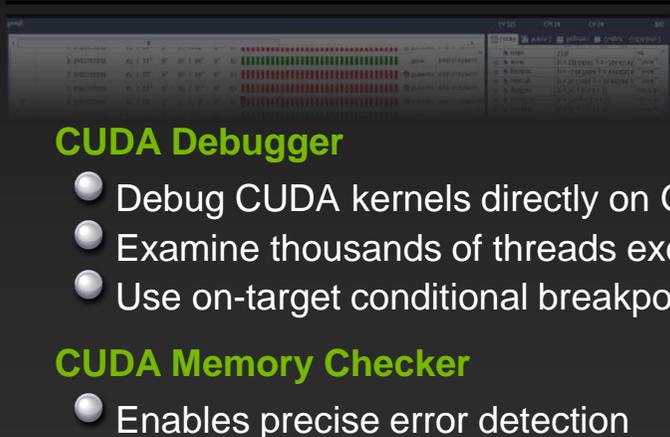
CUDA Debugger

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



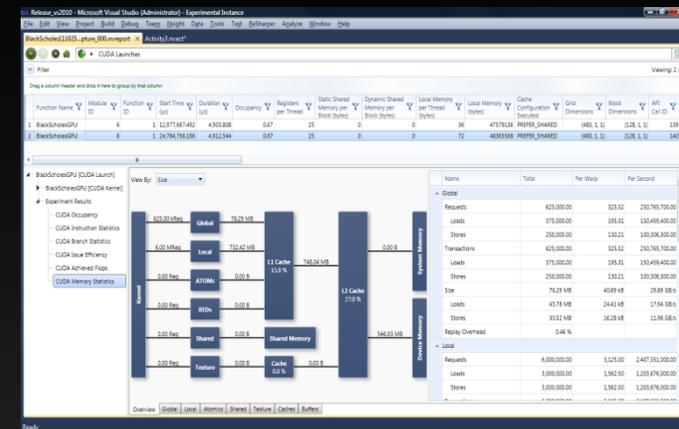
System Trace

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



CUDA Memory Checker

- Enables precise error detection



CUDA Profiler

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

NVIDIA Visual Profiler



The screenshot displays the NVIDIA Visual Profiler interface for a process named *dct8x8.vp. The main window shows a performance timeline with various components like Runtime API, Driver API, and Compute. The Compute section is expanded to show several CUDA kernels, including CUDAKernelQua... and CUDAKernel1DCT(float*, int...). The right-hand pane shows the properties for the selected CUDAKernel1DCT(float*, int, int, int) kernel, including its start time, duration, grid size, block size, and various efficiency metrics.

Name	Value
Start	161.329 ms
Duration	106.132 μ s
Grid Size	[64,64,1]
Block Size	[8,8,1]
Registers/Thread	14
Shared Memory/Block	512 bytes
Memory	
Global Load Efficiency	n/a
Global Store Efficiency	100%
DRAM Utilization	10.9% (18.4%)
Instruction	
Branch Divergence Overhead	0%
Total Replay Overhead	51%
Shared Memory Replay Overhead	0%
Global Memory Replay Overhead	51%
Global Cache Replay Overhead	0%
Local Cache Replay Overhead	0%
Occupancy	

Analysis Results

- High Branch Divergence Overhead [35.1% avg, for kernels accounting for 1.9% of compute]**
Divergent branches are causing significant instruction issue overhead. [More...](#)
- High Instruction Replay Overhead [46.6% avg, for kernels accounting for 39.1% of compute]**
A combination of global, shared, and local memory replays are causing significant instruction issue overhead. [More...](#)
- High Global Memory Instruction Replay Overhead [45.9% avg, for kernels accounting for 39.1% of compute]**
Non-coalesced global memory accesses are causing significant instruction issue overhead. [More...](#)

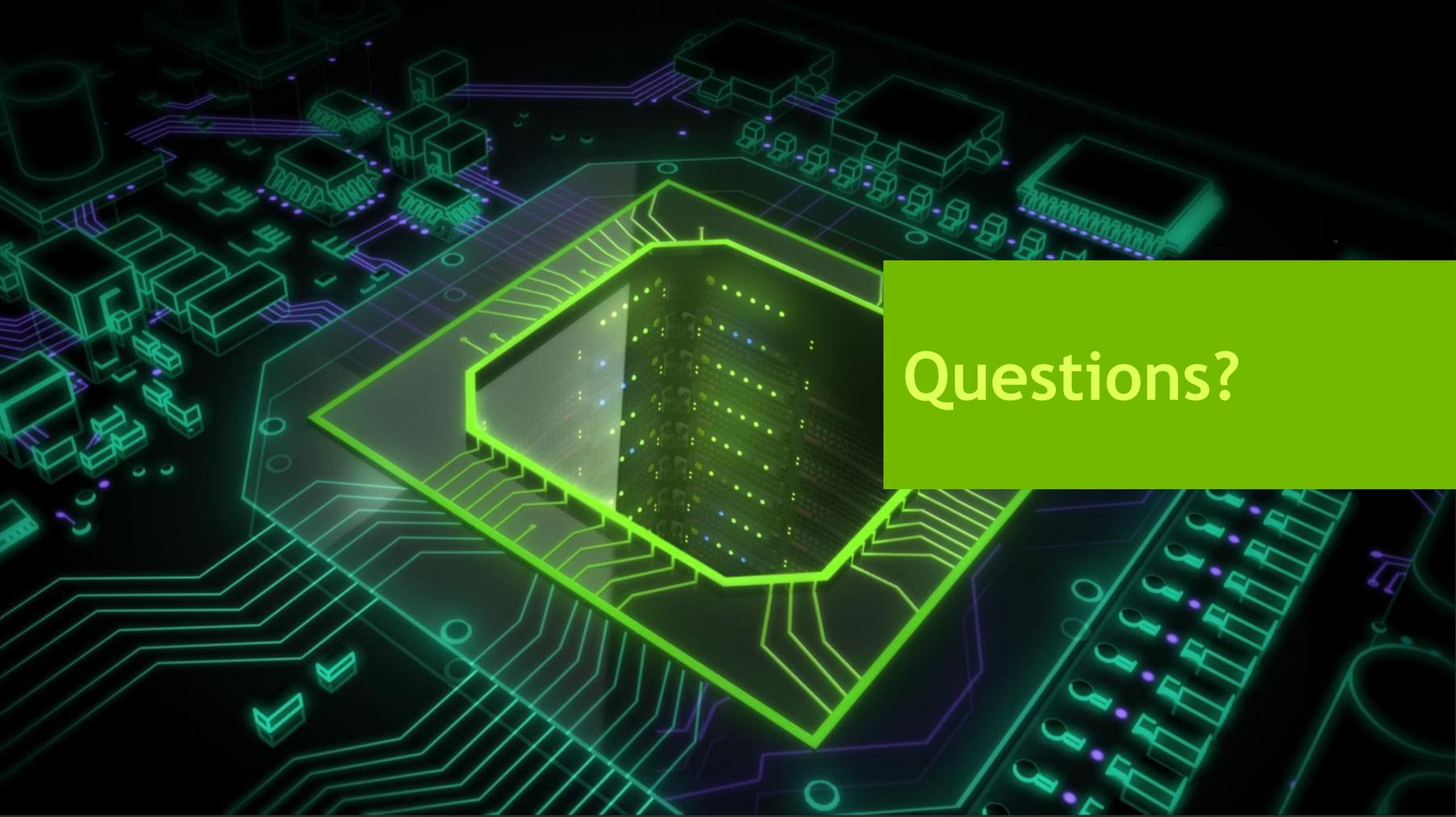
nvprof - CUDA 5.0 Toolkit



- **Textual reports**
 - Summary of GPU and CPU activity
 - Trace of GPU and CPU activity
 - Event collection
- **Headless profile collection**
 - Use nvprof on headless node to collect data
 - Visualize timeline with Visual Profiler

Links to get started

- Get CUDA: www.nvidia.com/getcuda
- Nsight: 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



Questions?