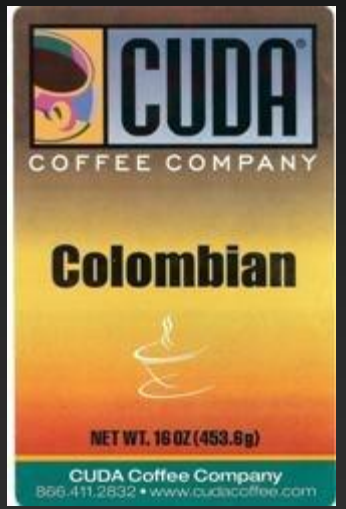
Introduction to CUDA C/C++

Mark Ebersole, NVIDIA
CUDA Educator

NVIDIA

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

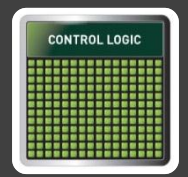
Open Compiler Tool Chain

LLVM COMPILER INFRASTRUCTURE

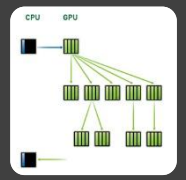
Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

Hardware Capabilities

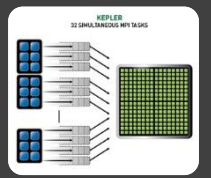
SMX



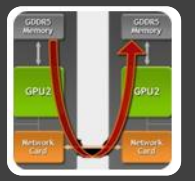
Dynamic Parallelism



HyperQ



GPUDirect

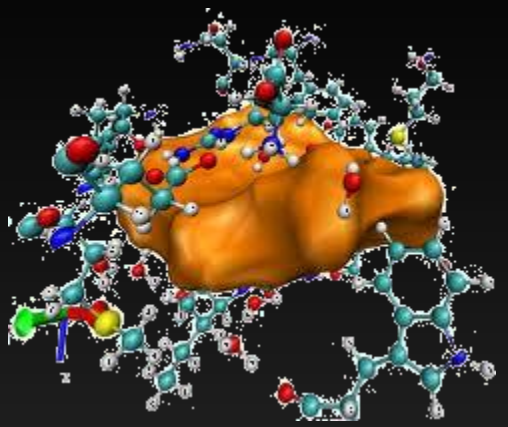


Getting Started with CUDA

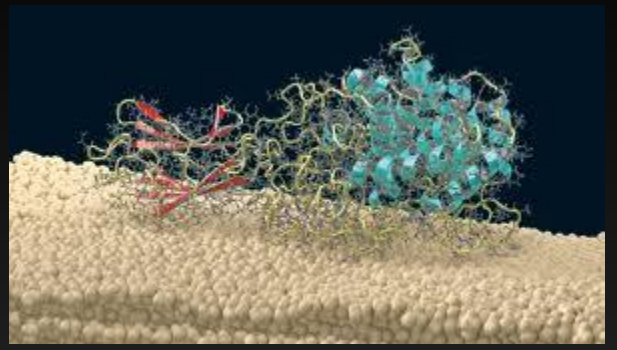
GPU Accelerated Science Applications

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

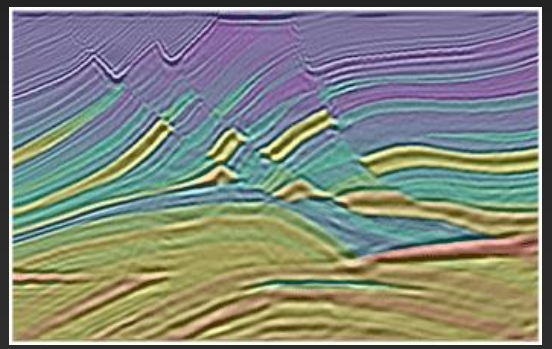
AMBER



GROMACS



LAMMPS



Tsunami RTM

www.nvidia.com/teslaapps

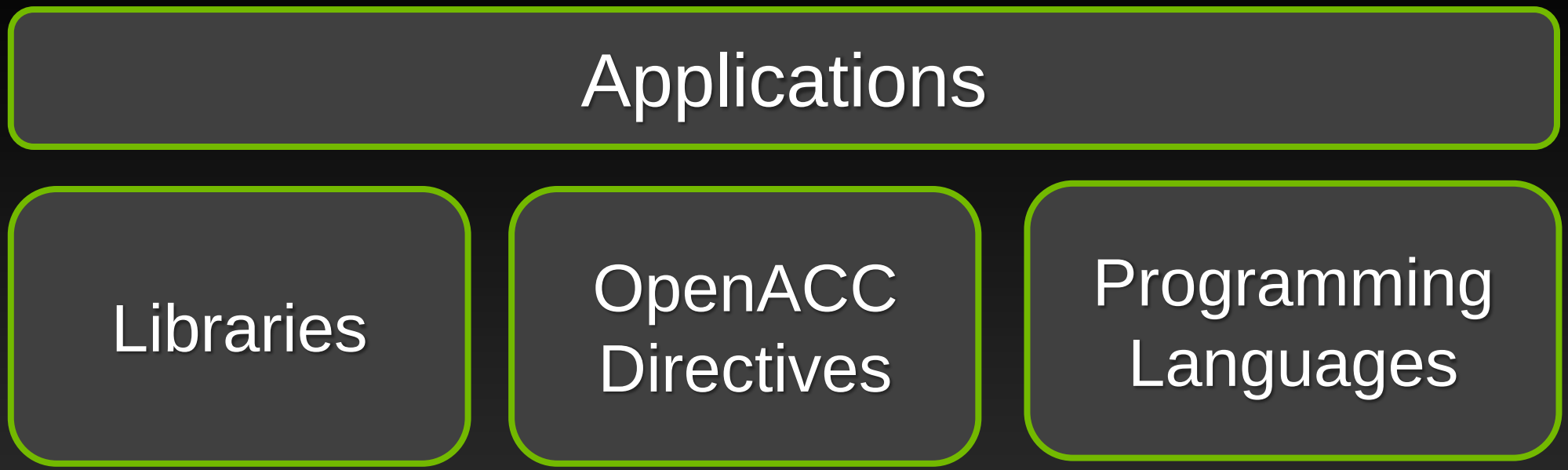
NWChem

GPU Accelerated Workstation Applications

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

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

3 Ways to Accelerate Applications



3 Ways to Accelerate Applications

Applications

Libraries

OpenACC Directives

Programming Languages

"Drop-in" Acceleration

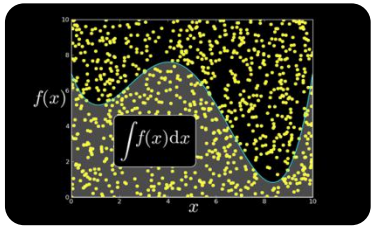
Easily Accelerate Applications

Maximum Flexibility

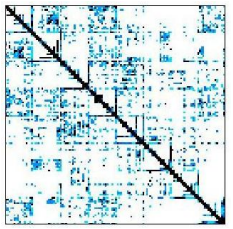
GPU Accelerated Libraries

"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

open source initiative

NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

open source initiative

Thrust

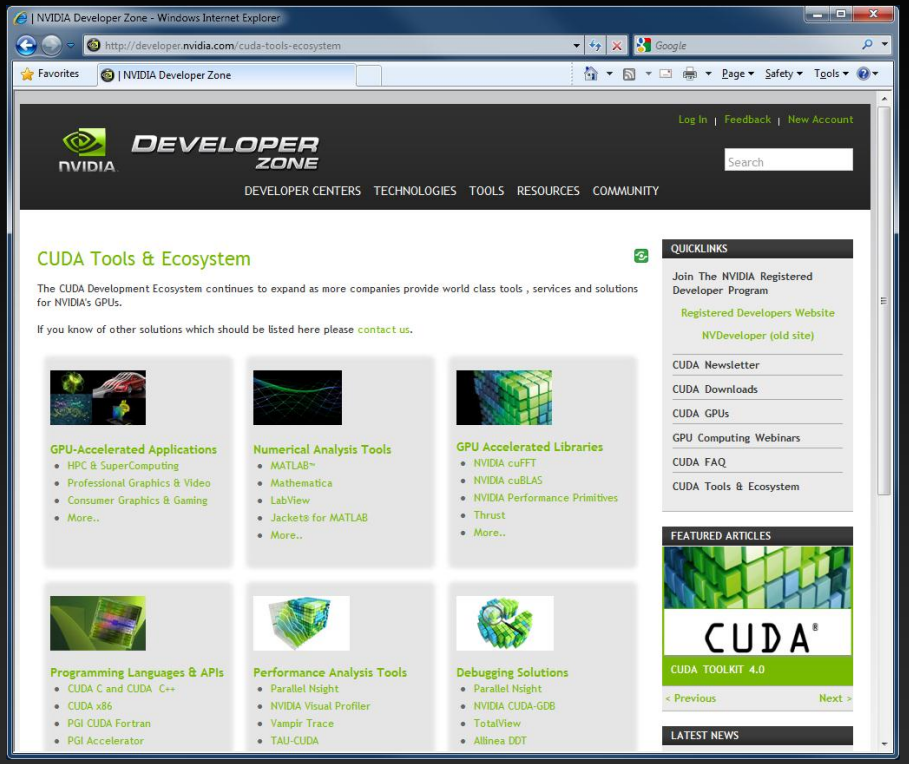
C++ STL Features for CUDA

open source initiative

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

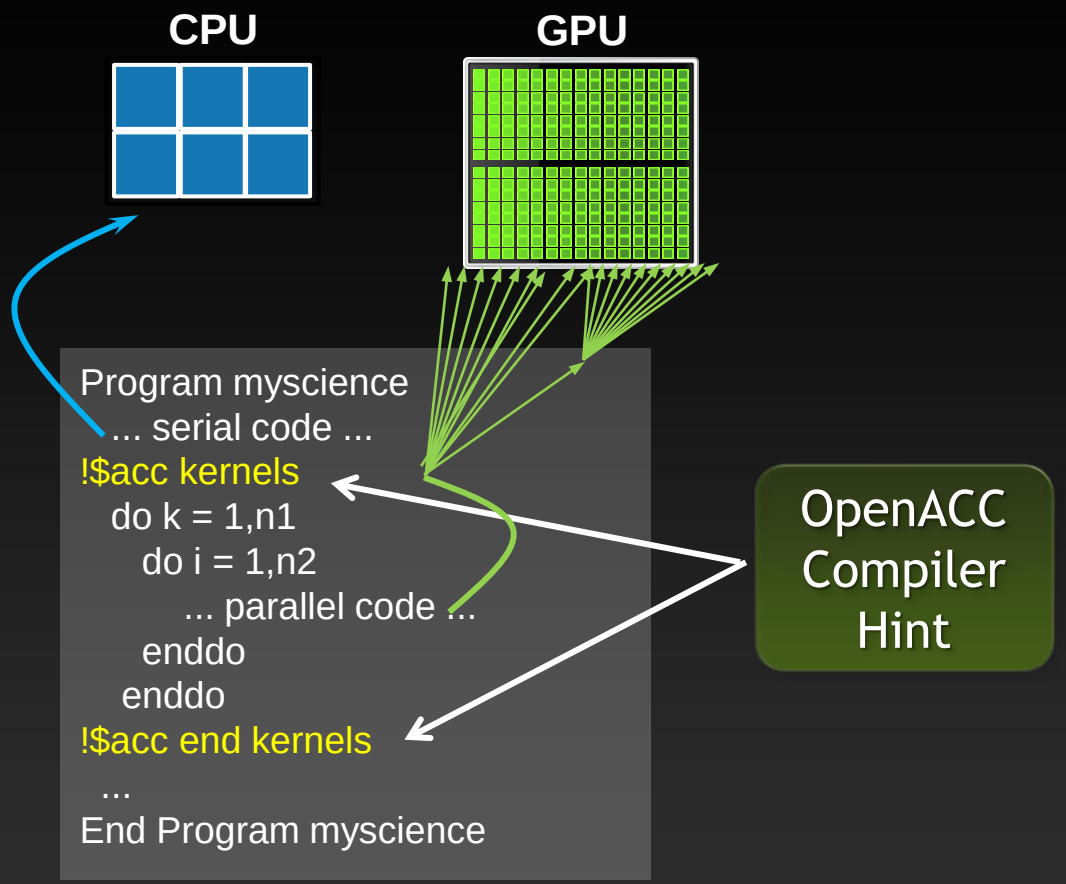
Programming Languages

"Drop-in" Acceleration

Easily Accelerate Applications

Maximum Flexibility

OpenACC Directives



Simple Compiler hints

Compiler Parallelizes code

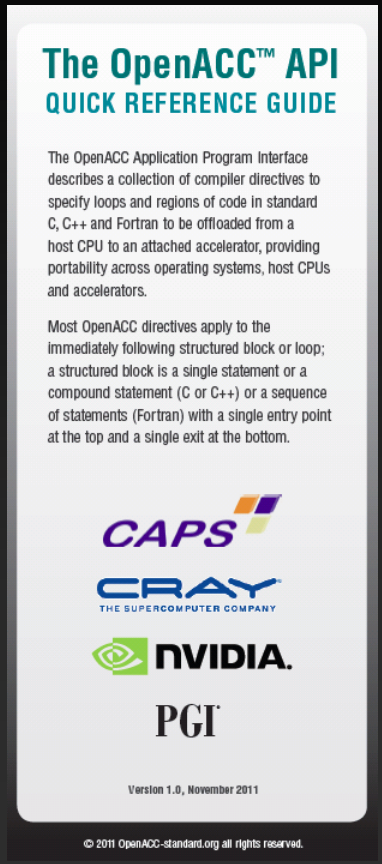
Works on many-core GPUs & multicore CPUs

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



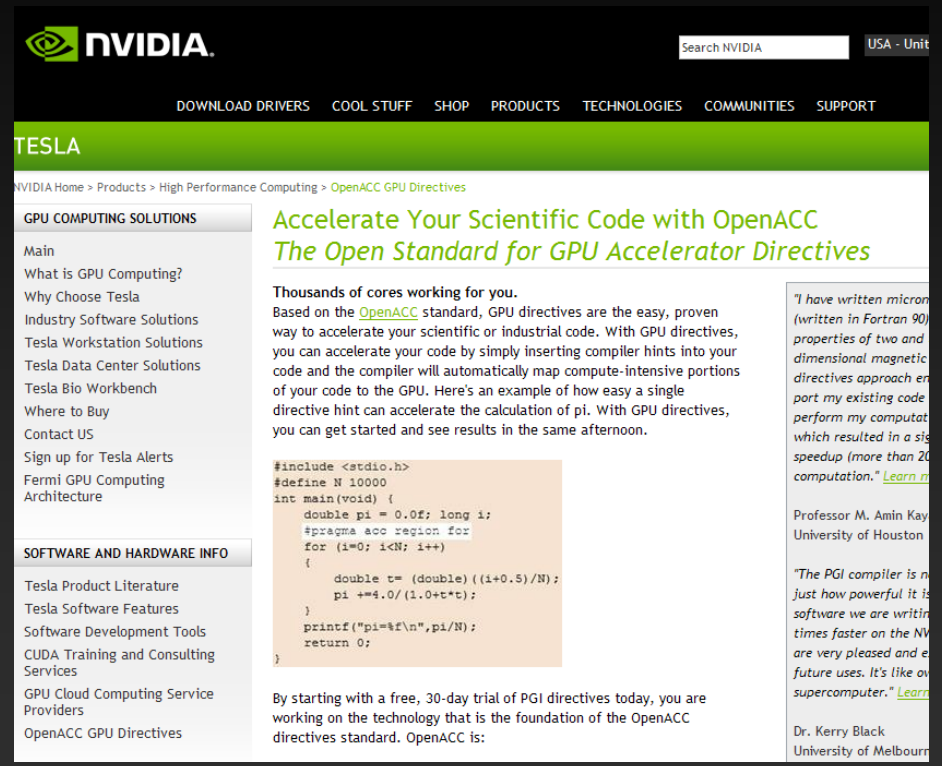
Start Now with OpenACC Directives

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



NVIDIA

PGI®

3 Ways to Accelerate Applications

Applications

Libraries

OpenACC Directives

Programming Languages

"Drop-in" Acceleration

Easily Accelerate Applications

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

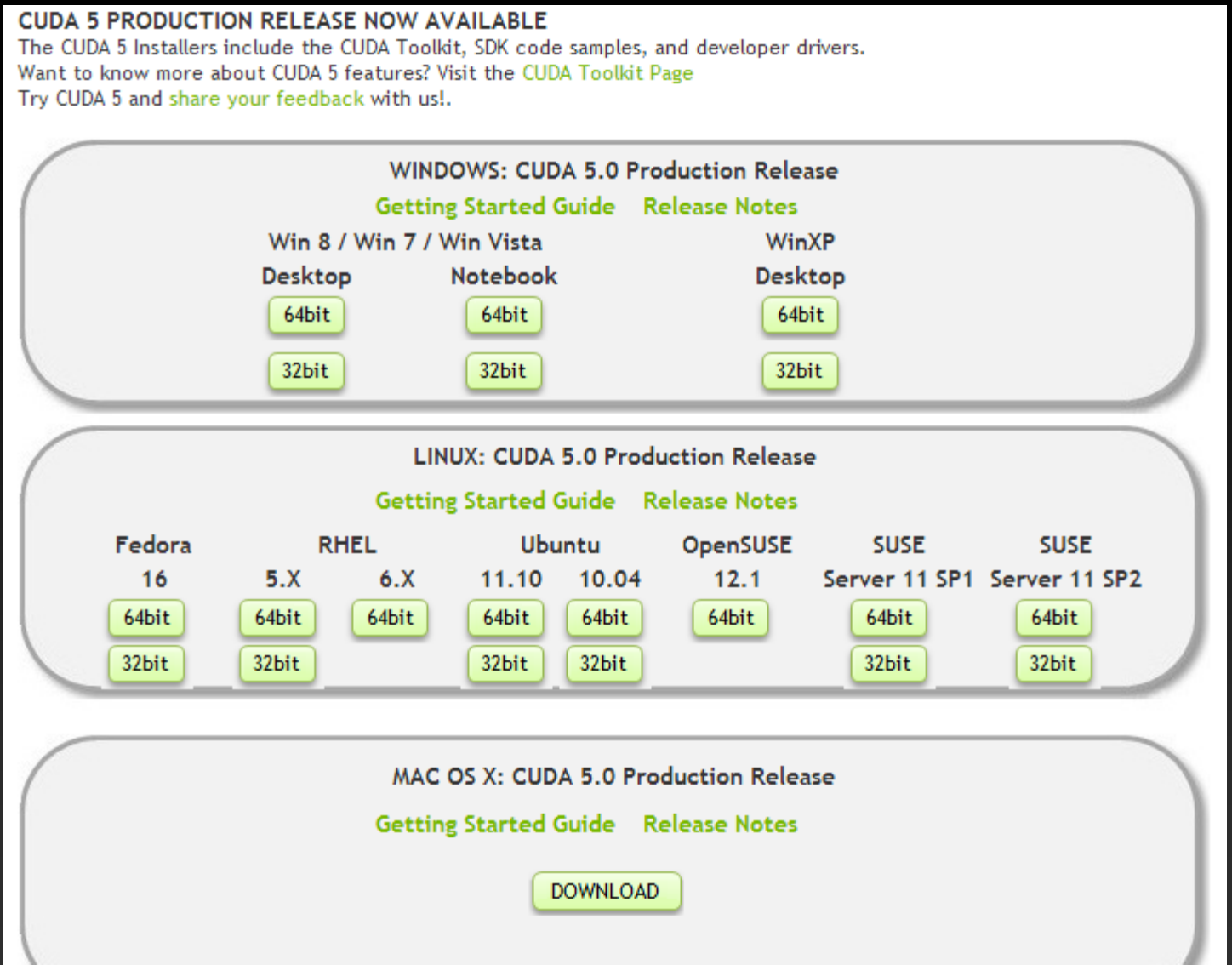
Programming a CUDA Language

- 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



CONCEPTS

- Heterogeneous Computing
- Blocks
- Threads
- Indexing

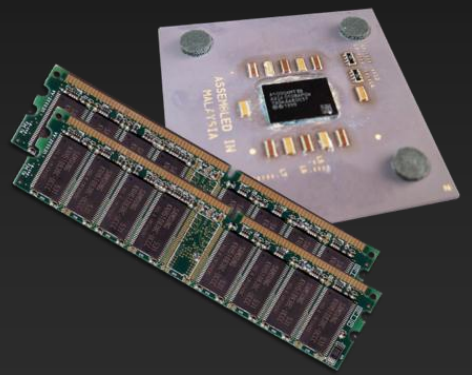
SAXPY

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

- E… …reads”

= THREAD

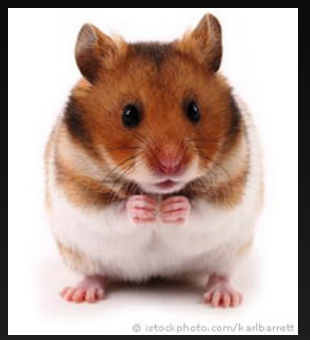
- A… …g threadIdx.x
- T… …k can be read with blockDim.x

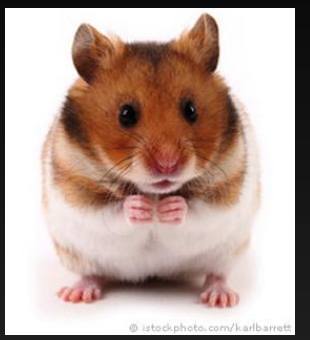
Parallelism on a GPU - CUDA Threads

- Each block is then broken up into “threads”

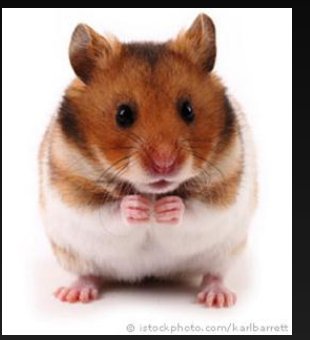
Block


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?

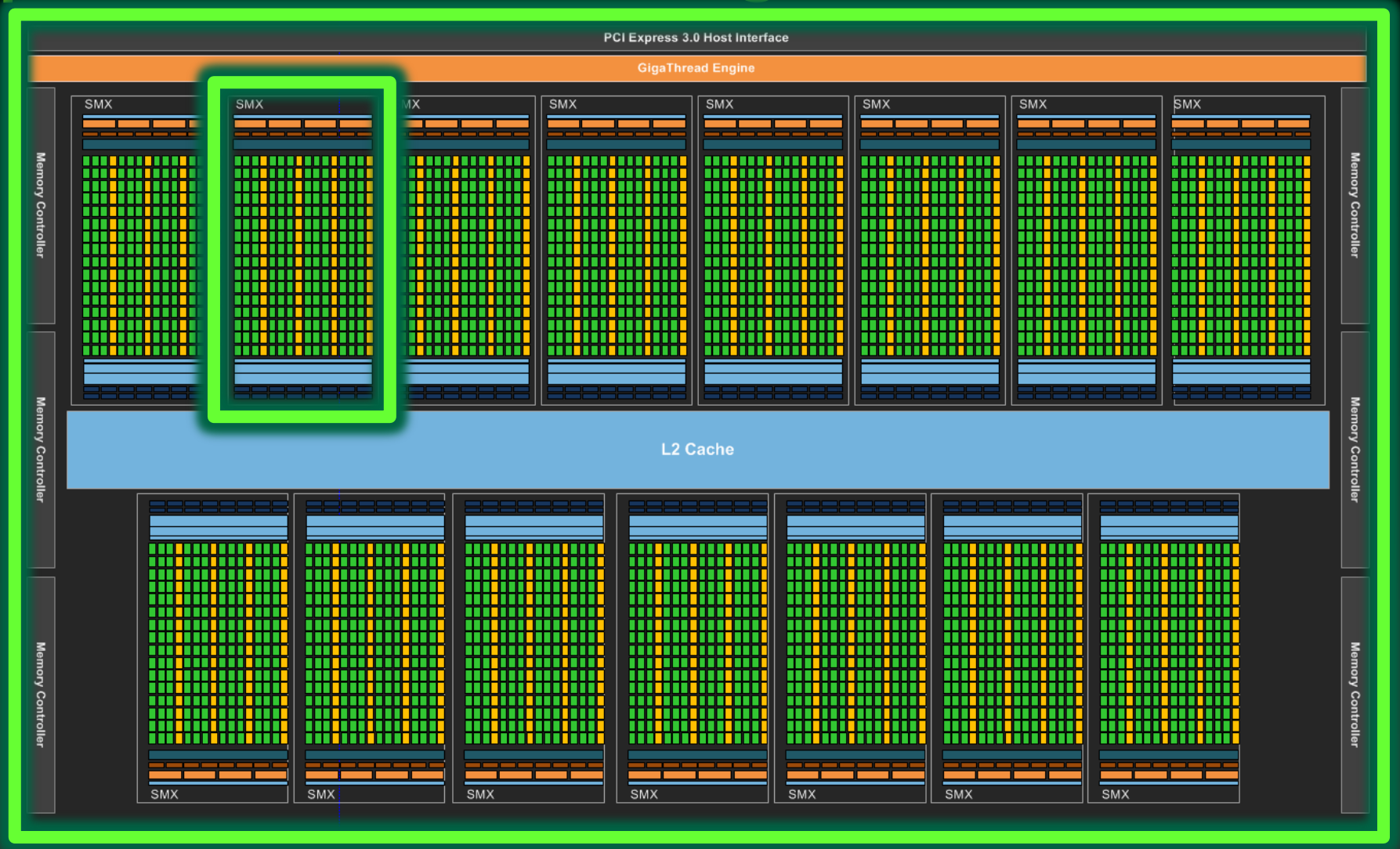
- **Threads within a block can**
  - Communicate very quickly (share memory)
  - Synchronize (wait for all threads to catch up)

- **Why 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

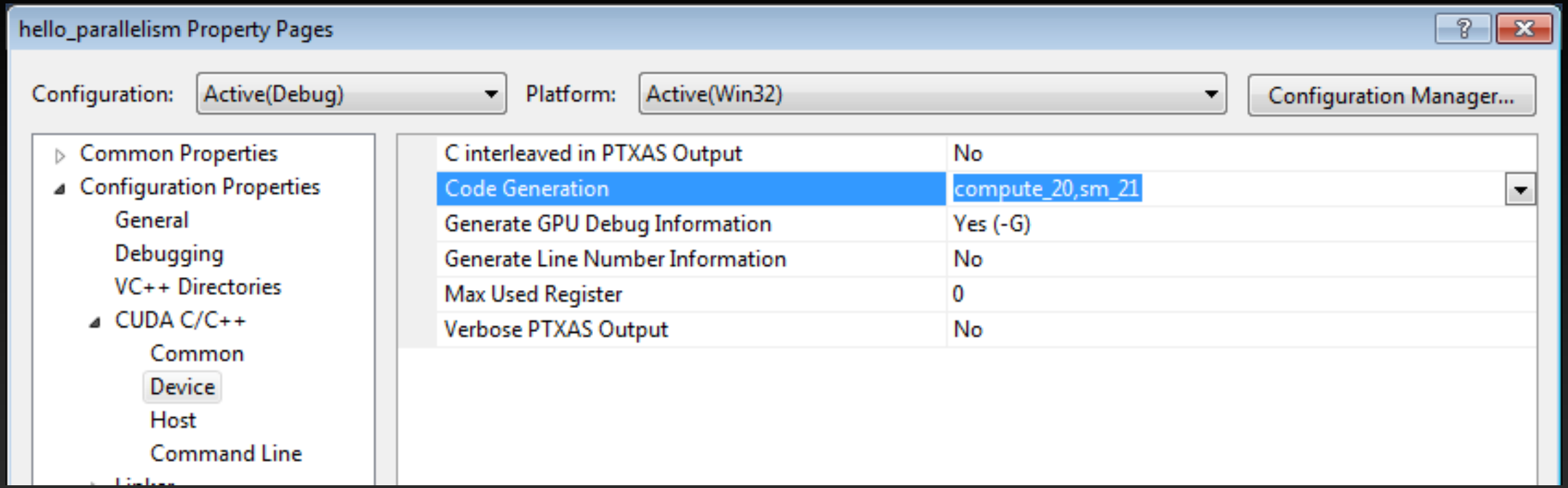
Why threads and blocks?

- **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**

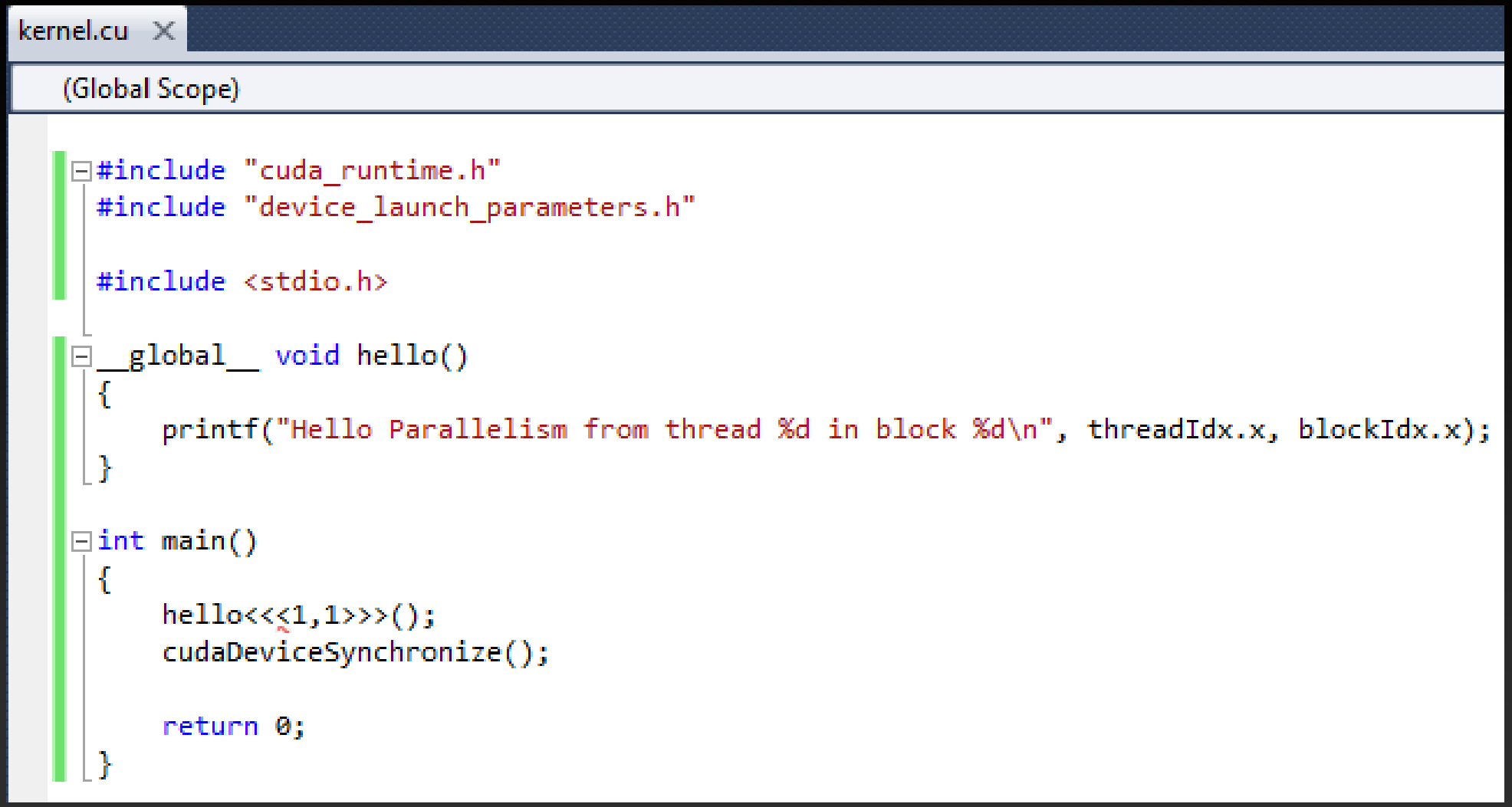
Kepler GK110 Block Diagram

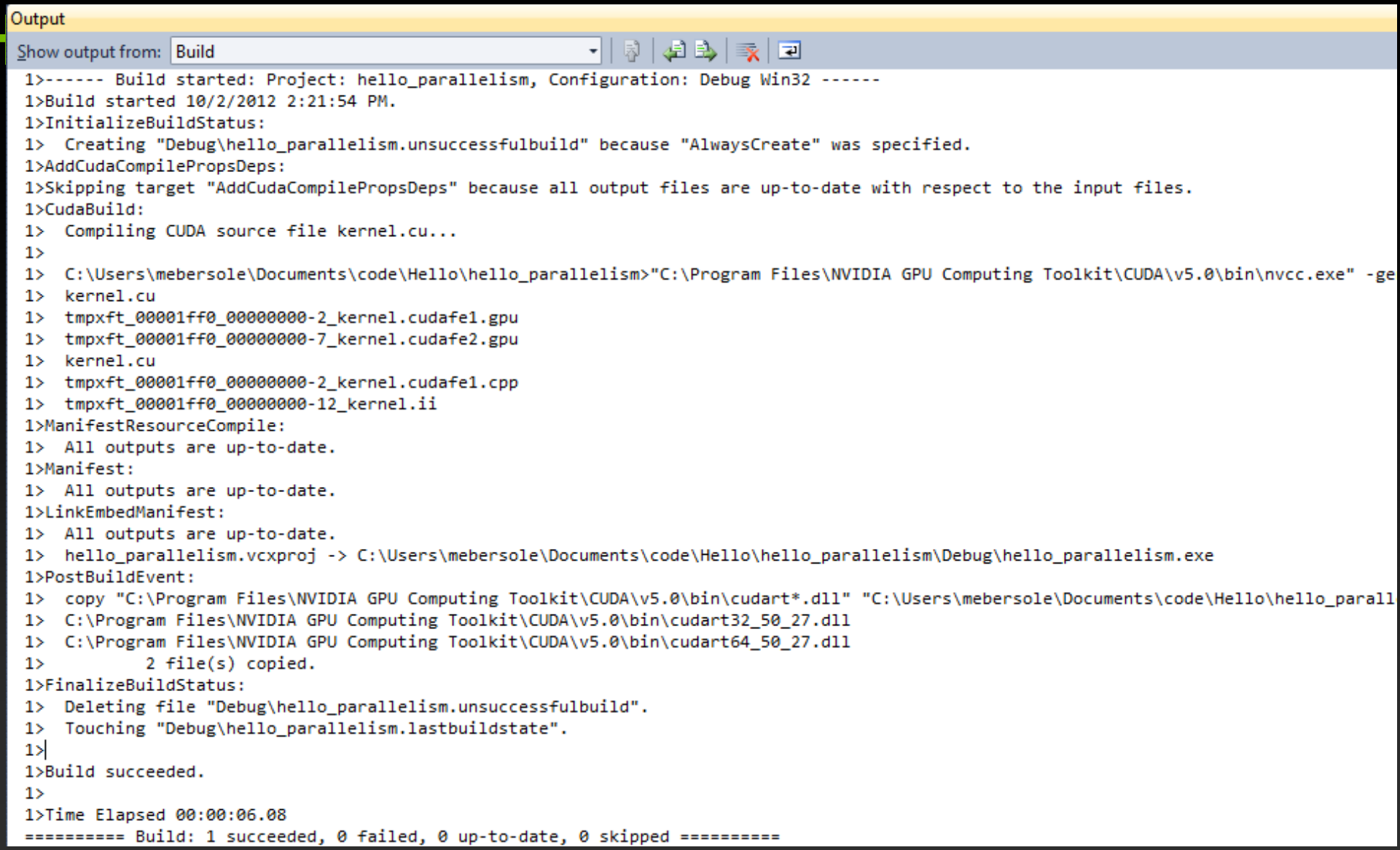


Hello Parallelism!

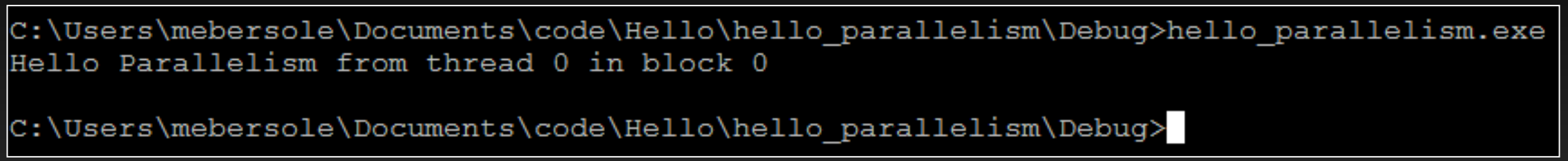


Hello Parallelism!

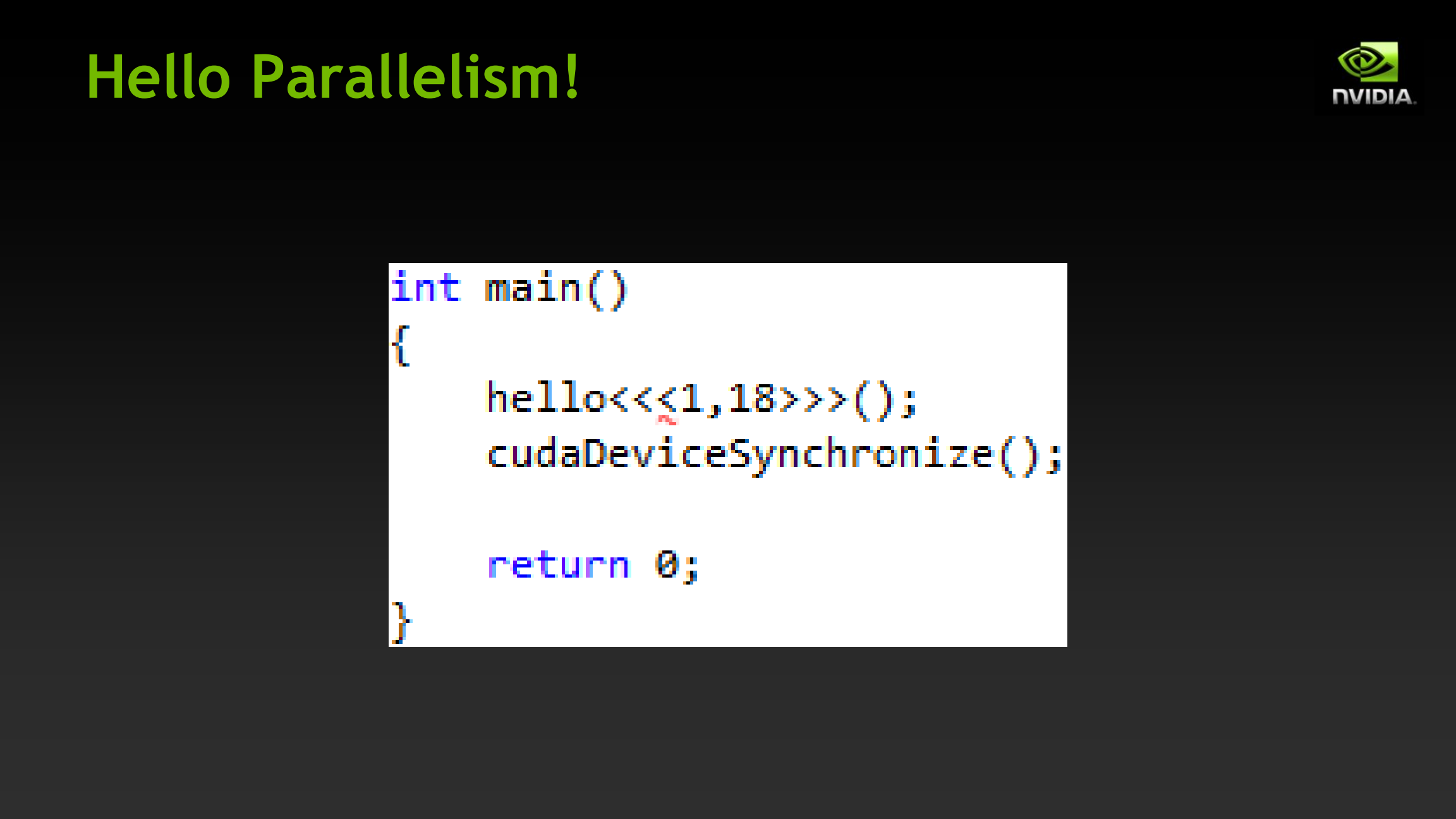



Output
Show output from: Build
1>------ Build started: Project: hello_parallelism, Configuration: Debug Win32 ------
1>Build started 10/2/2012 2:21:54 PM.
1>InitializeBuildStatus:
1> Creating "Debug\hello_parallelism.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>AddCudaCompilePropsDeps:
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\cudart*.dll" "C:\Users\mebersole\Documents\code\Hello\hello_parall
1> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\cudart32_50_27.dll
1> C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.0\bin\cudart64_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!



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 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}

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
}
```

http://developer.nvidia.com/cuda-toolkit

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);
```

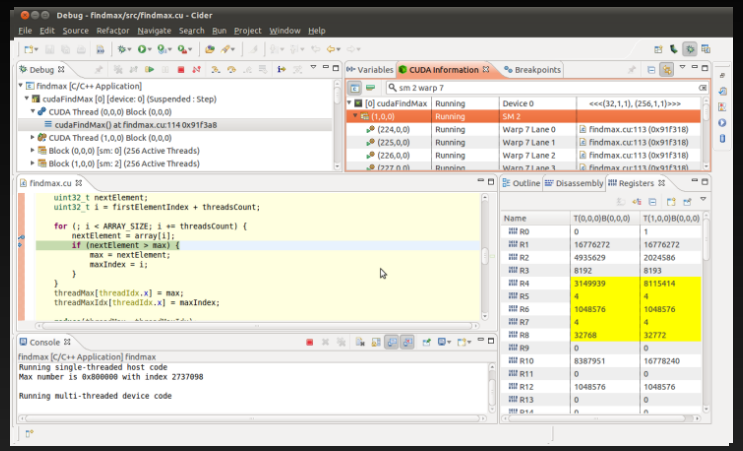
www.boost.org/libs/lambda

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);
```

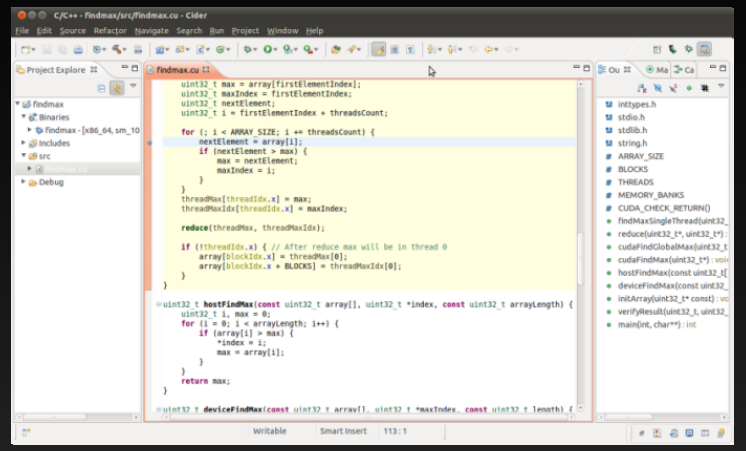
thrust.github.com

NVIDIA® Nsight™ Eclipse Edition for Linux and MacOS



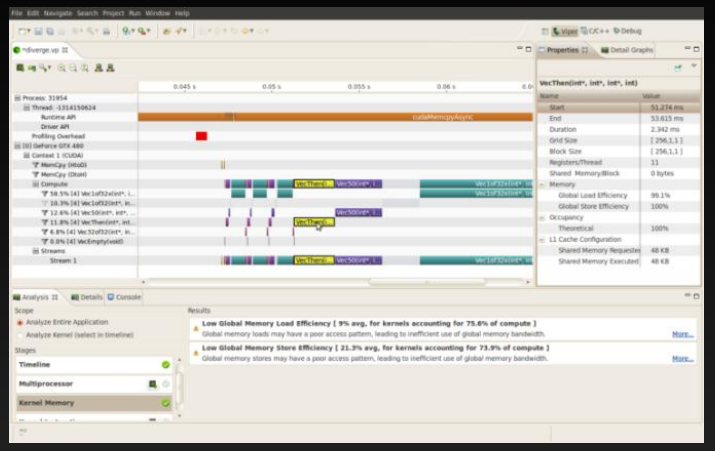
CUDA-Aware Editor

- 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

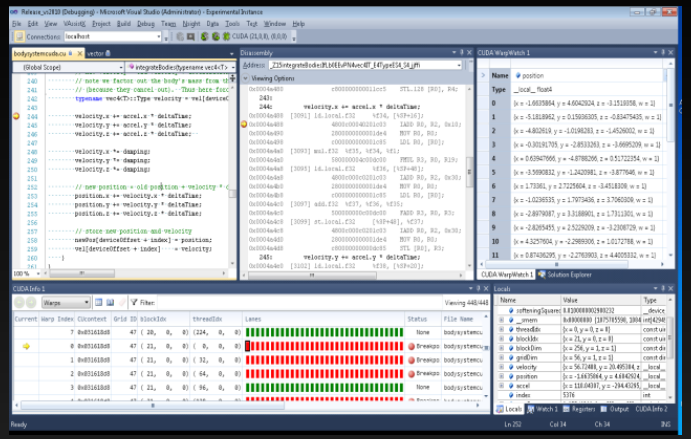


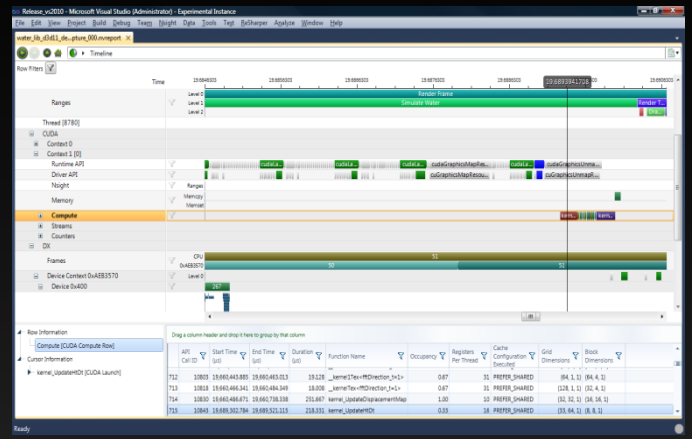
Nsight Profiler

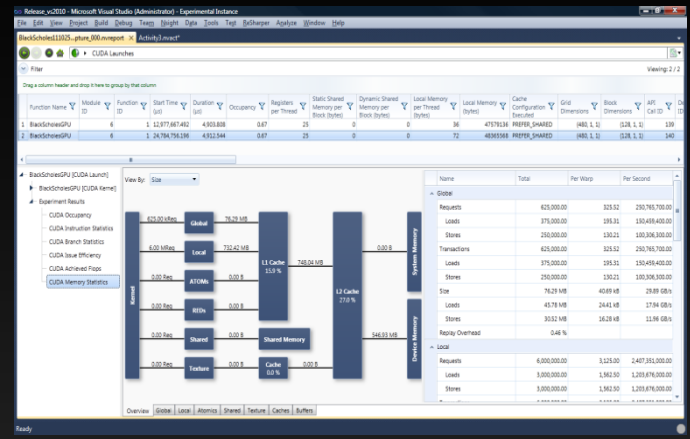
- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

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

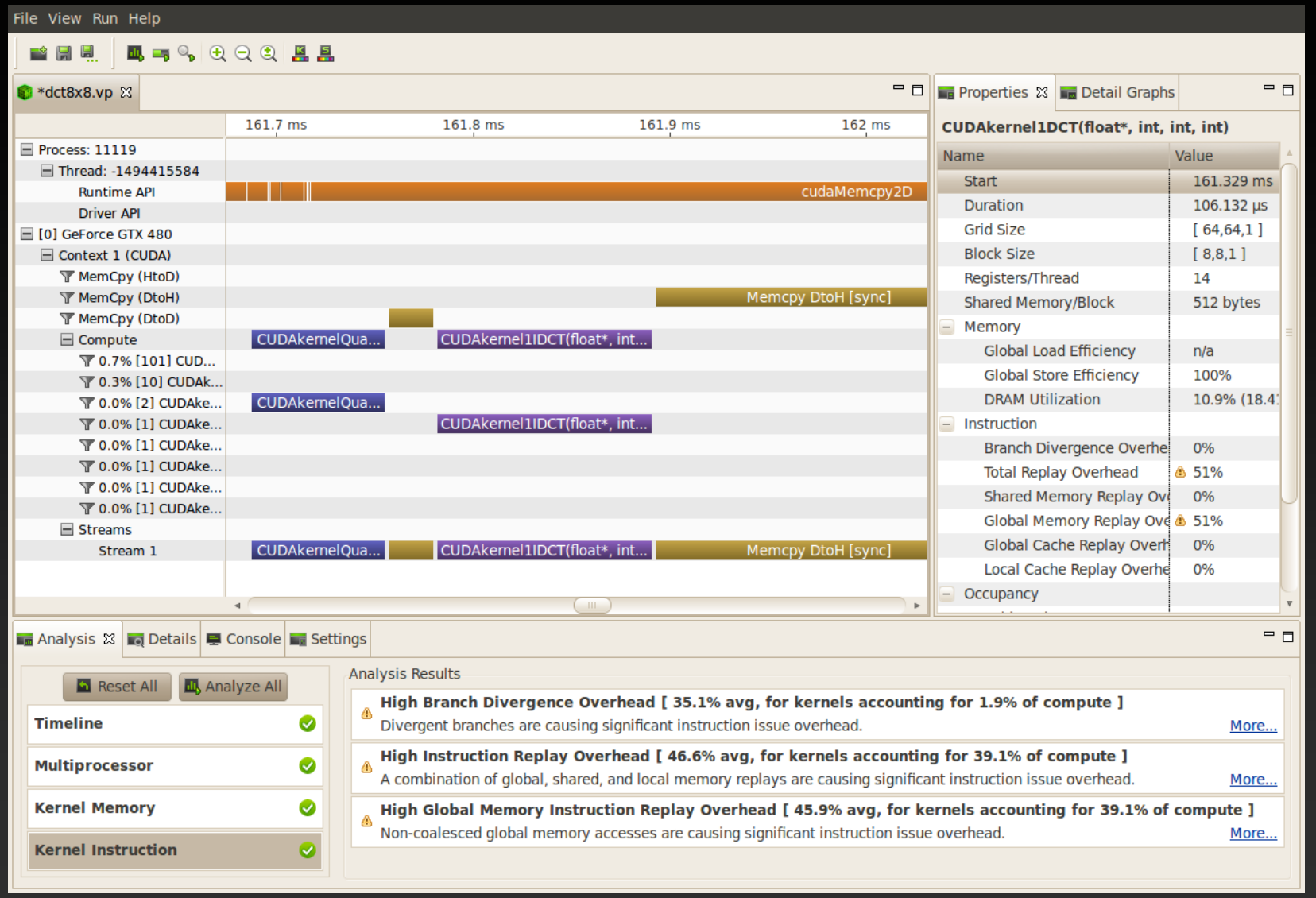
System Trace

- 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

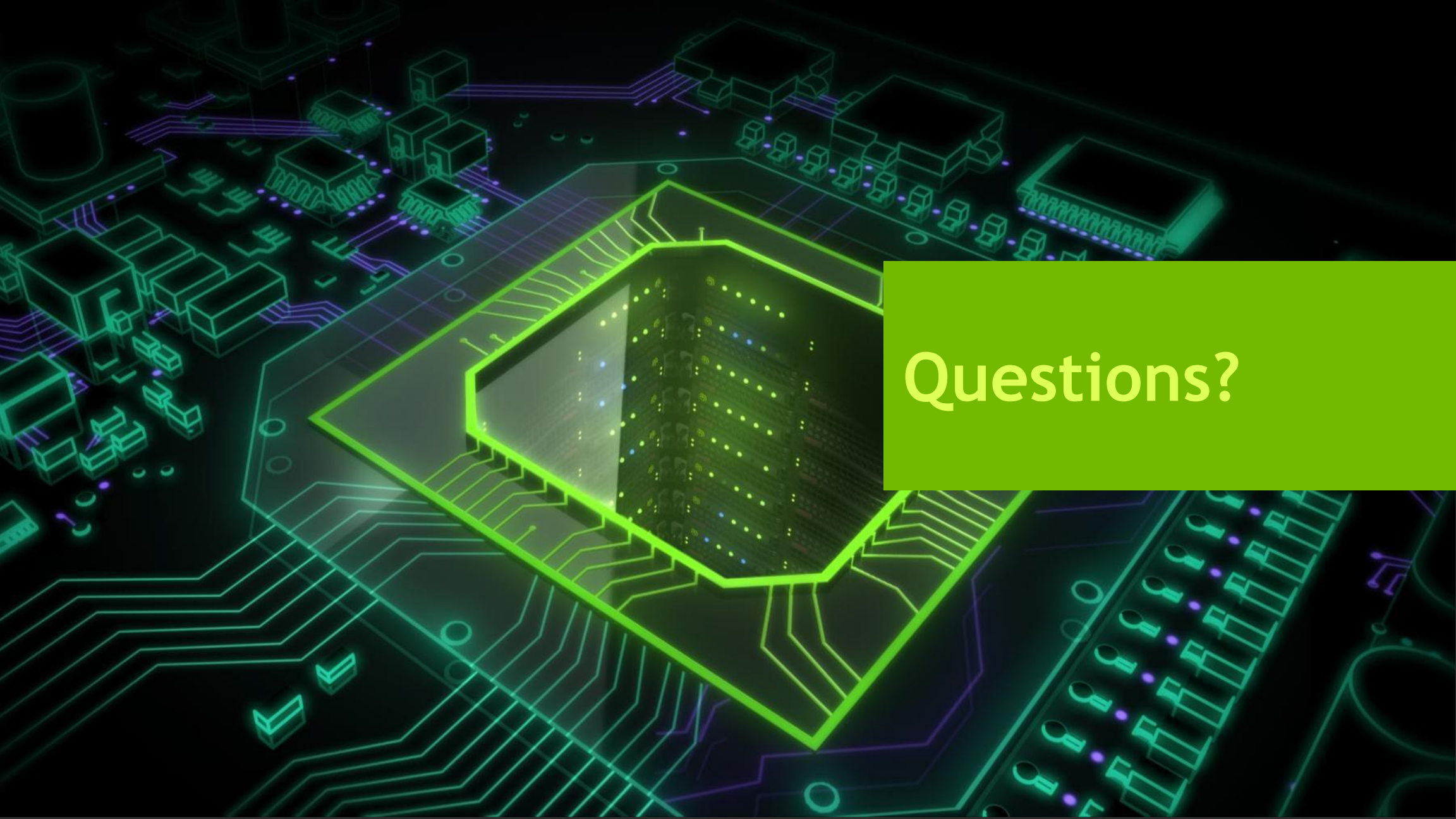


nvprof - CUDA 5.0 Toolkit

- 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?