

The background features a complex, three-dimensional grid of glowing cubes. The cubes are primarily green and cyan, with some transitioning into blue and purple towards the right side. The grid is curved and recedes into the distance, creating a sense of depth and perspective. The lighting is dramatic, with bright highlights and deep shadows, giving the grid a metallic or crystalline appearance.

Introducing CUDA 5

CUDA 5

Application Acceleration Made Easier

Dynamic Parallelism

Spawn new parallel work from within GPU code on K20

GPU Callable Libraries

Libraries and plug-ins for GPU code

GPUDirect™

RDMA between GPUs and PCIe devices

New Nsight™ Eclipse Edition

Develop, Debug, and Optimize... all in one tool!

What is Dynamic Parallelism?

The ability to launch new kernels from the GPU

- Dynamically - based on run-time data
- Simultaneously - from multiple threads at once
- Independently - each thread can launch a different grid



Fermi: Only CPU can generate GPU work

Kepler: GPU can generate work for itself

Dynamic Parallelism

Accelerate More of Your Application on the GPU

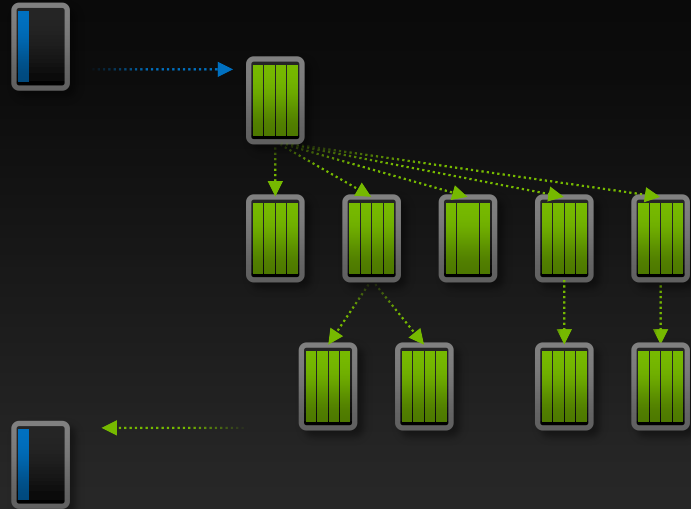
CPU

Fermi GPU

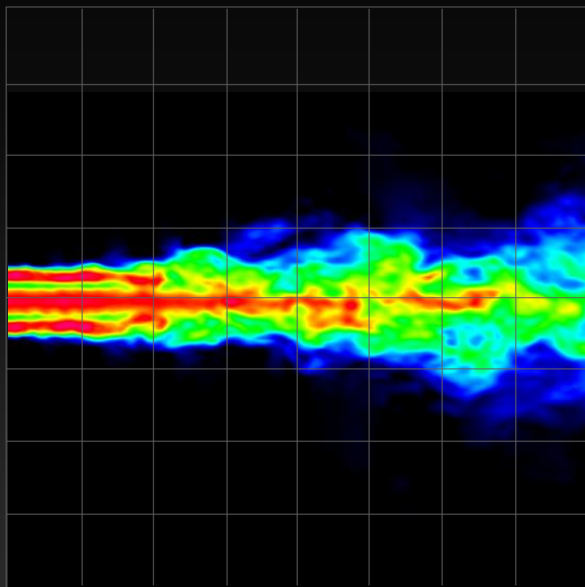


CPU

Kepler GPU

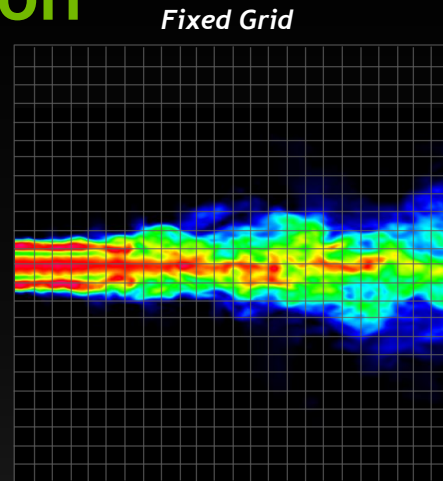


Dynamic Work Generation

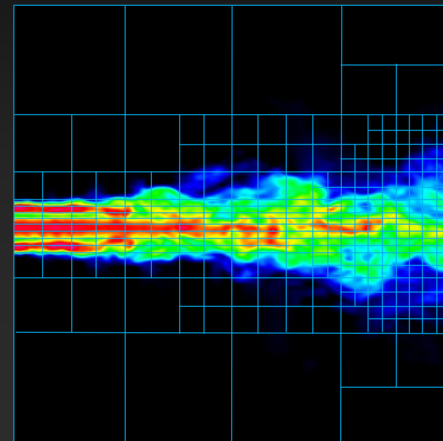


Initial Grid

*Statically assign conservative
worst-case grid*



*Dynamically assign performance
where accuracy is required*



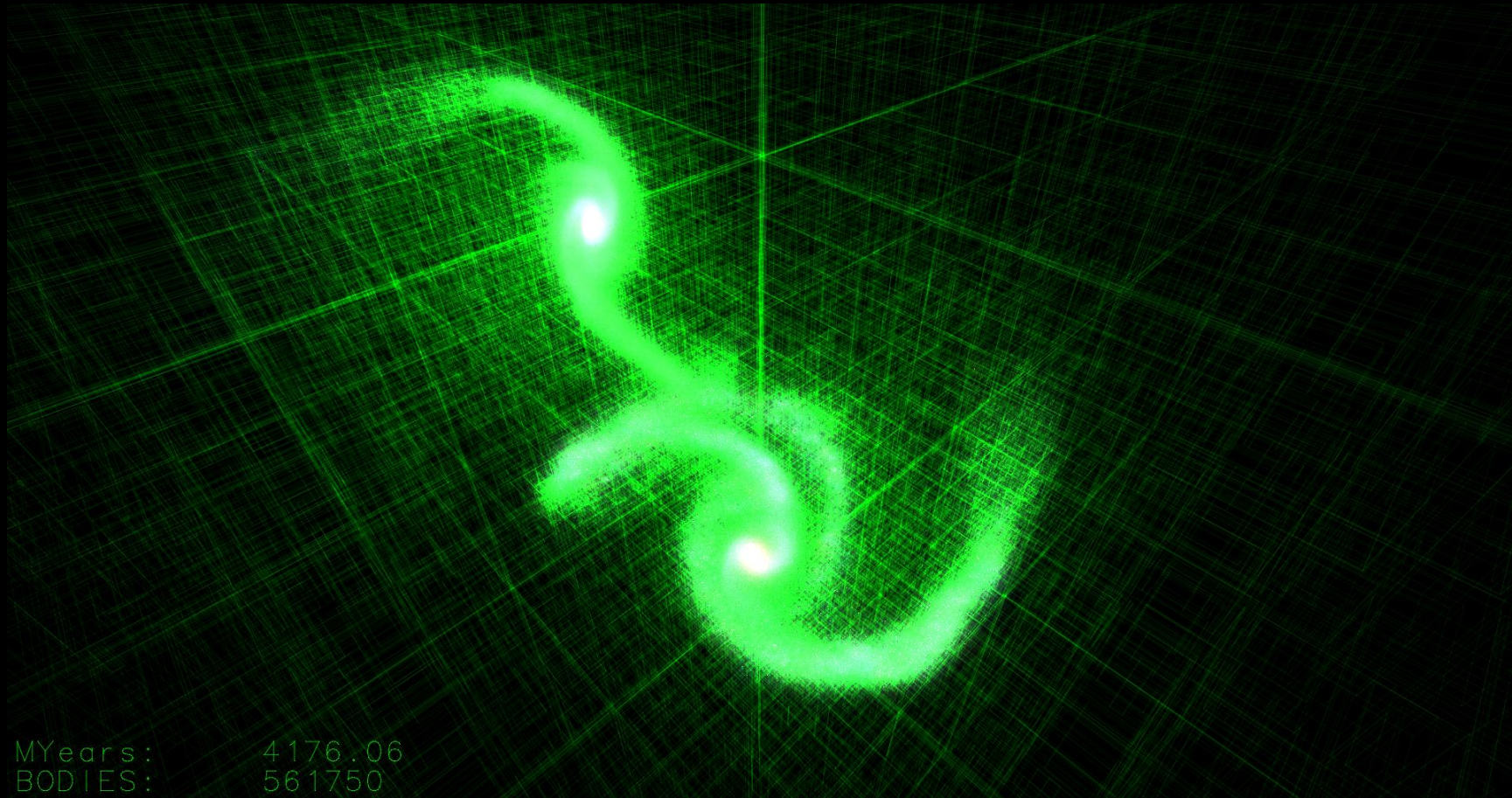
Dynamic Grid

Mapping Compute to the Problem



MYears: 4176.06
BODIES: 561750

Mapping Compute to the Problem

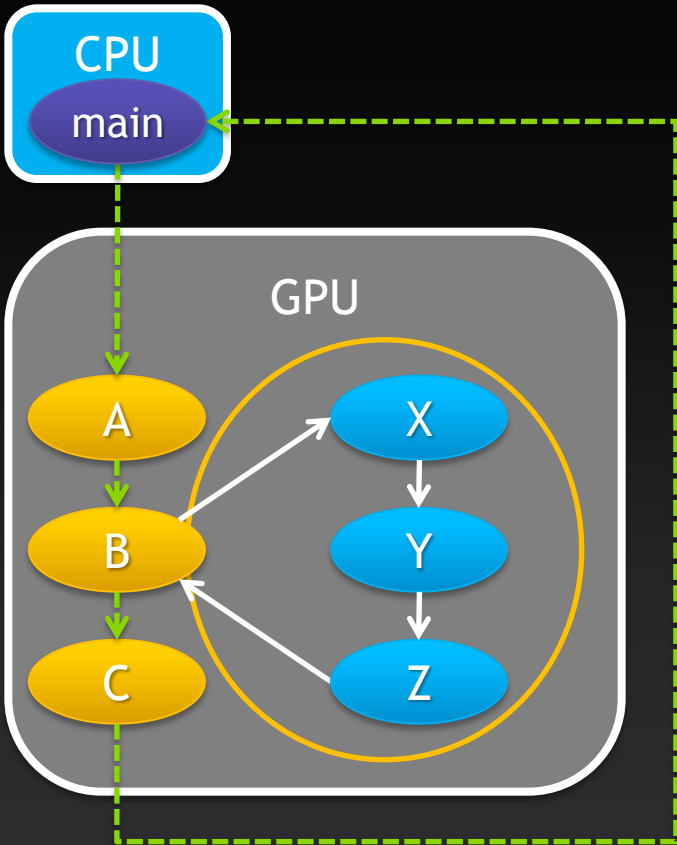


MYears: 4176.06
BODIES: 561750

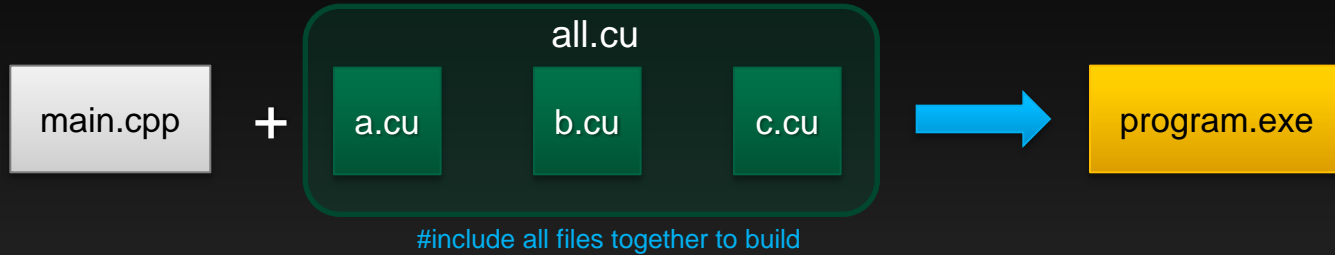
Familiar Syntax and Programming Model

```
int main() {  
    float *data;  
    setup(data);  
  
    A <<< ... >>> (data);  
    B <<< ... >>> (data);  
    C <<< ... >>> (data);  
  
    cudaDeviceSynchronize();  
    return 0;  
}
```

```
__global__ void B(float *data)  
{  
    do_stuff(data);  
  
    X <<< ... >>> (data);  
    Y <<< ... >>> (data);  
    Z <<< ... >>> (data);  
    cudaDeviceSynchronize();  
  
    do_more_stuff(data);  
}
```

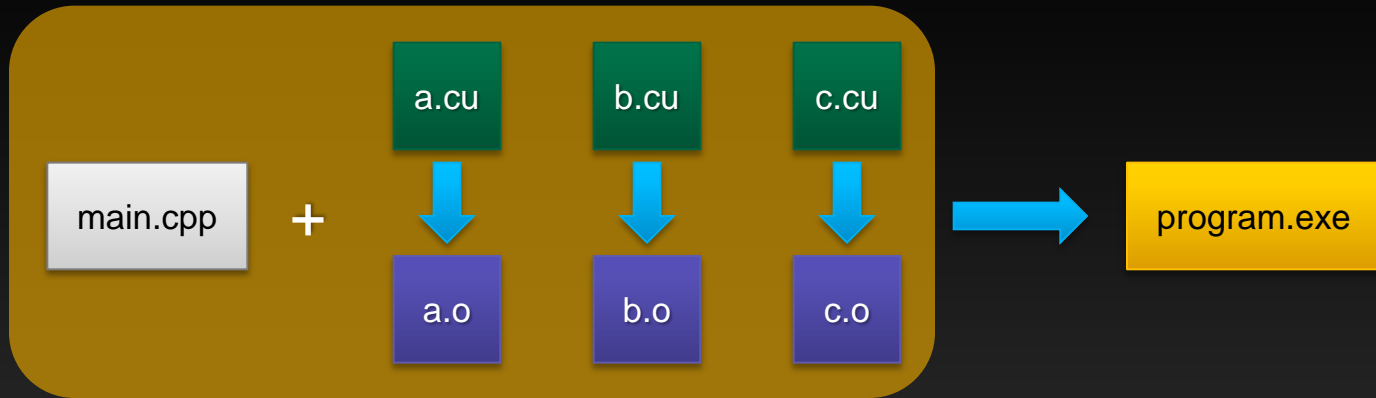


Before CUDA 5: Whole-Program Compilation



Earlier CUDA releases required a single source file for each kernel
Linking with external code was not supported

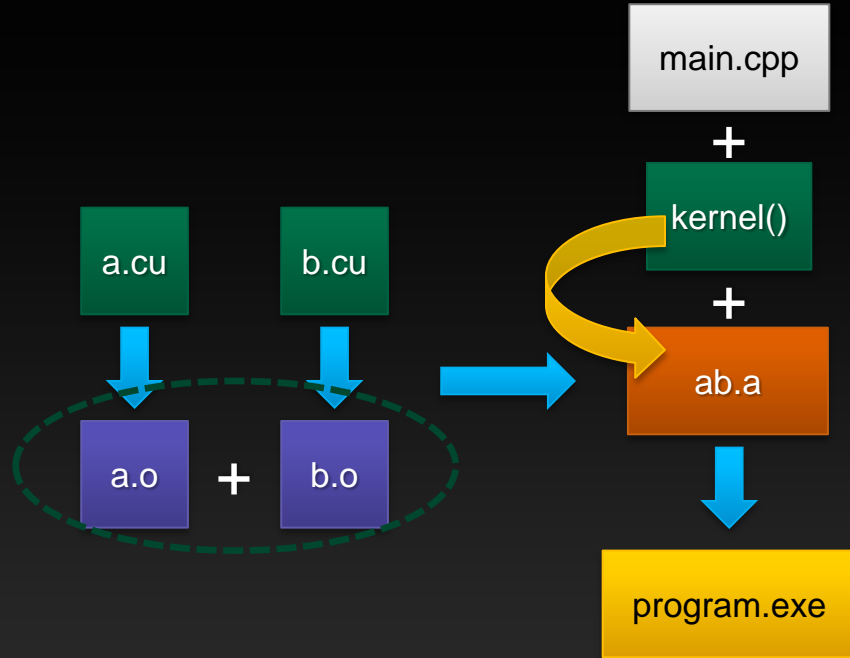
CUDA 5: Separate Compilation & Linking



Separate compilation allows building independent object files

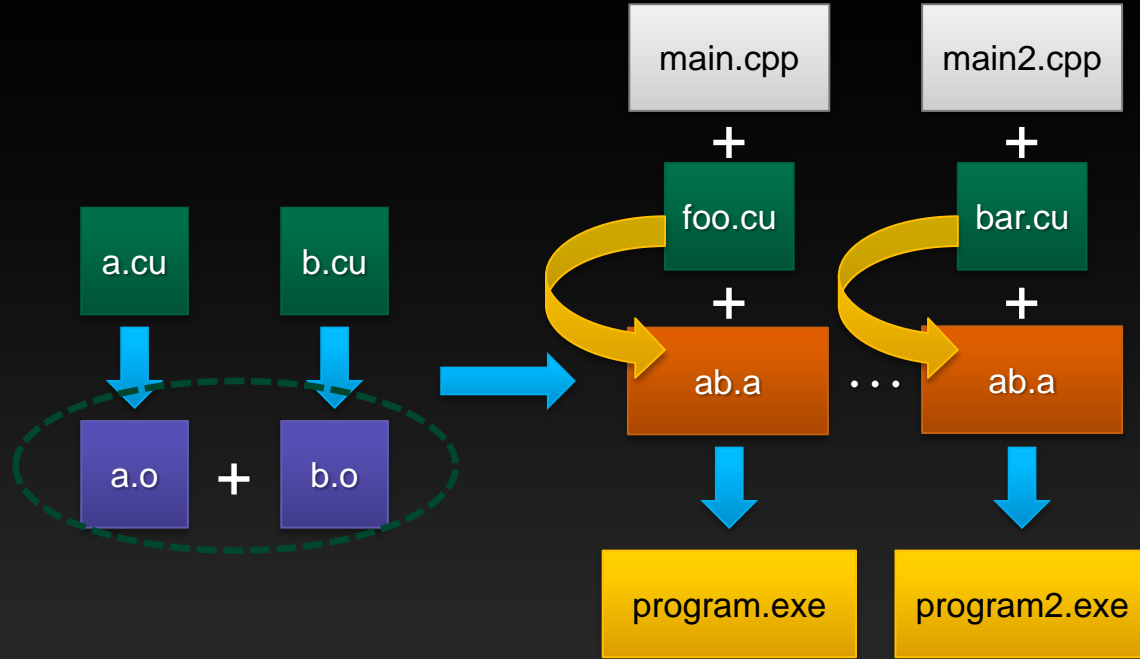
CUDA 5 can link multiple object files into one program

CUDA 5: GPU Callable Libraries



Can combine object files into static libraries
Link and externally call *device* code

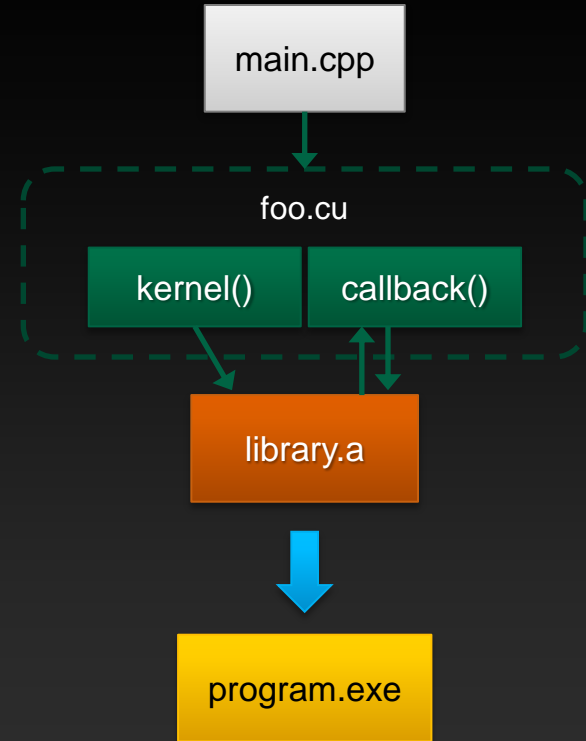
CUDA 5: GPU Callable Libraries



Facilitates code reuse, reduces compile time

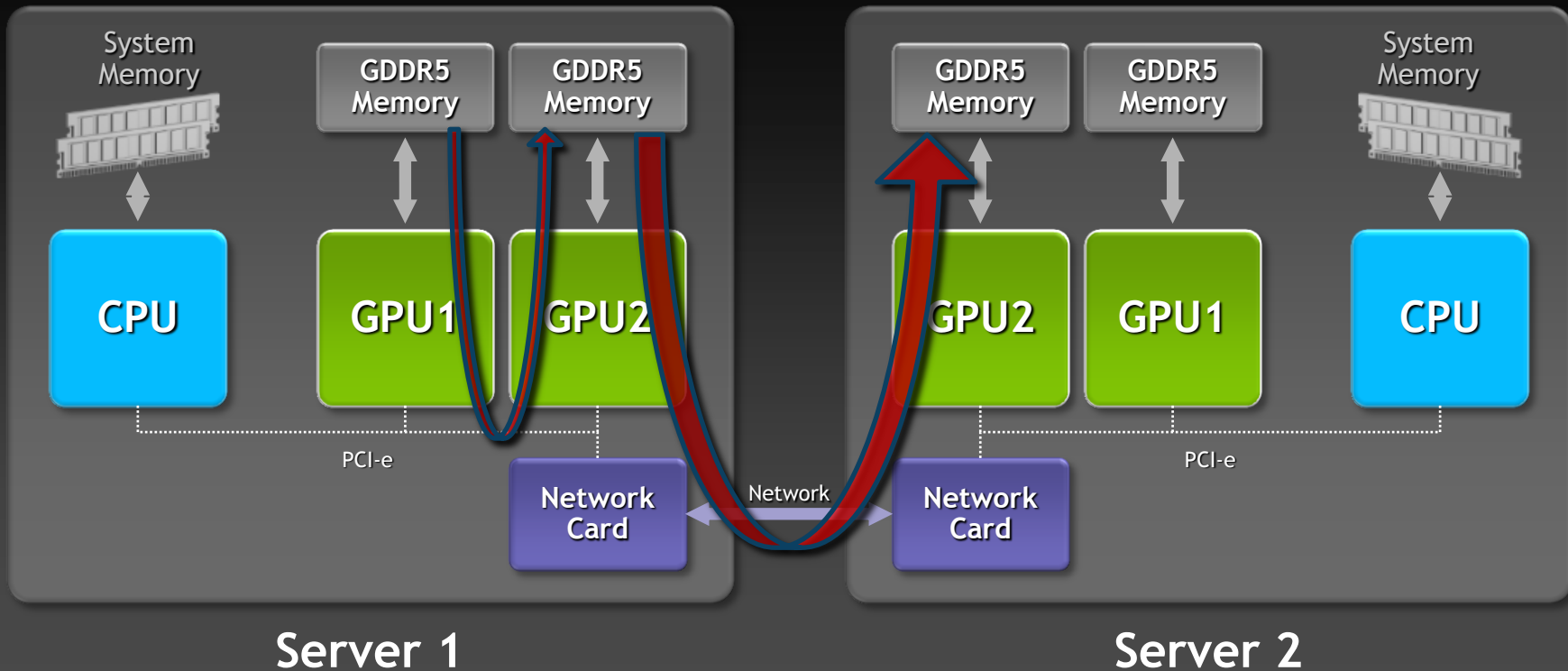
CUDA 5: Callbacks

Enables closed-source device libraries to call user-defined device callback functions



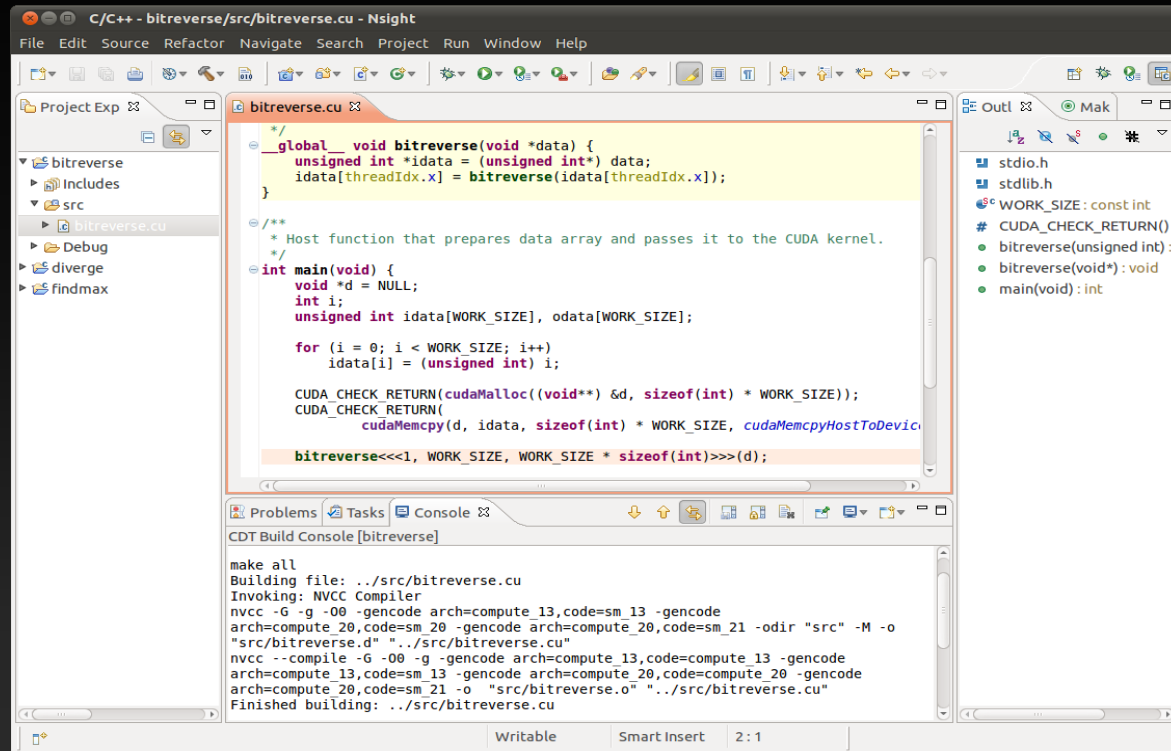
NVIDIA® GPUDirect™ Support for RDMA

Direct Communication Between GPUs and PCIe devices



NVIDIA® Nsight,™ Eclipse Edition

CUDA aware editor

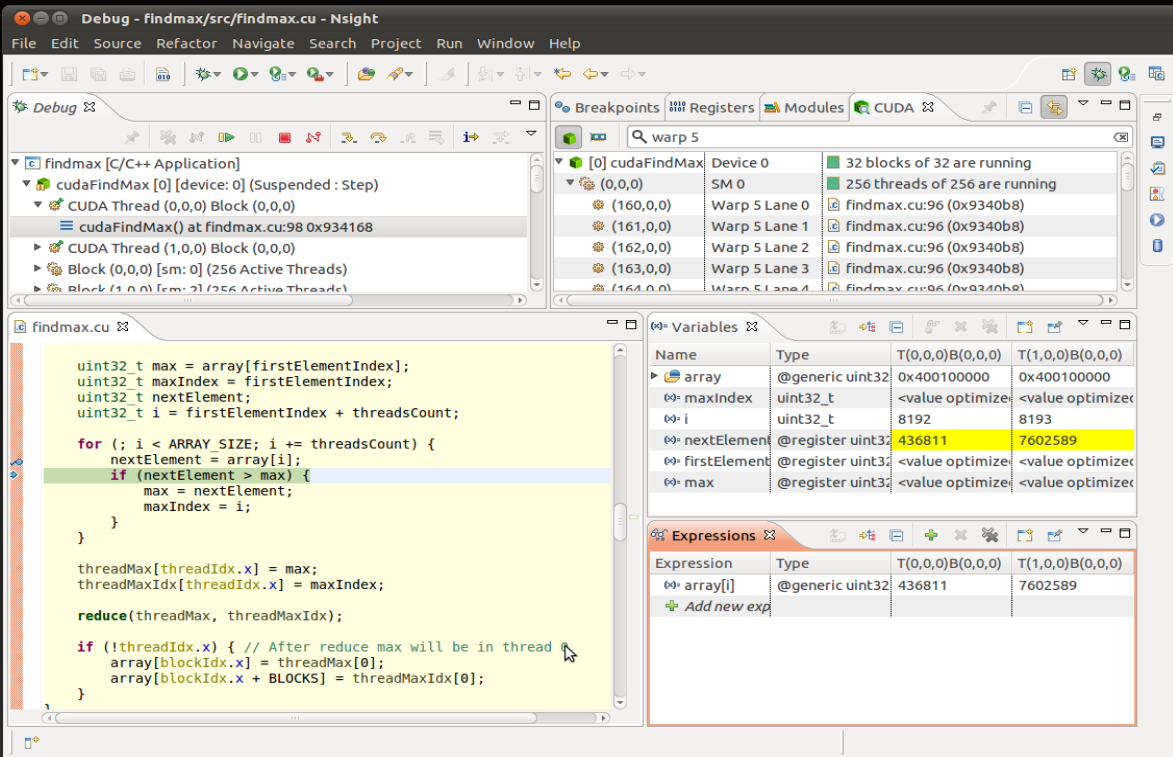


- Integrated CUDA samples makes it quick and easy to get started
- Easily port CPU loops to CUDA kernels with automatic code refactoring
- Semantic highlighting of CUDA code makes it easy to differentiate GPU code from CPU code
- Generate code faster with CUDA aware auto code completion and inline help
- Hyperlink navigation enables faster code browsing
- Supports automatic makefile generation

NVIDIA® Nsight,™ Eclipse Edition

Nsight Debugger

- Seamless and simultaneous debugging of both CPU and GPU code
- View program variables across several CUDA threads
- Examine execution state and mapping of the kernels and GPUs
- View, navigate and filter to selectively track execution across threads
- Set breakpoints and single-step execution at both source-code and assembly levels
- Includes CUDA-MEMCHECK to help detect memory errors



NVIDIA® Nsight™, Eclipse Edition

Nsight Profiler

Profile - Timeline for session diverge - Nsight

File Edit Navigate Search Project Run Window Help

*diverge

Process: 21240
Thread: 950560576
Runtime API: cuda...
Driver API: cudaMemcpyAsync

[0] GeForce GTX 280

Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
63.5% [4] Vec1of...
10.6% [4] Vec50(i...
10.6% [4] Vec1of...
9.6% [4] VecThen(...
5.8% [4] Vec32of...
0.0% [4] VecEmpt...
Streams
Stream 1

Properties [0] GeForce GTX 280

Name	Value
Duration	705.345 ms
Compute Utilization	89.6%
Compute/Memcpy	4,337.261
Overlap	
Kernel/Memcpy	0%
Attributes	
Compute Capability	1.3
Maximums	
Multiprocessor	
Multiprocessors	30
Clock Rate	1.296 GHz
Concurrent Kernel	false
Max IPC	0
Threads per Warp	32
Memory	
Global Memory Bandwidth	125.12 GB/s

Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput	gst 64B	gst 32B	gld 128B	gld 64B
Memcpy HtoD [async]	64.463 ms	48.287 µs	n/a	n/a	n/a	n/a	n/a	256 KB	5.06 GB/s	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	64.599 ms	47.68 µs	n/a	n/a	n/a	n/a	n/a	256 KB	5.12 GB/s	n/a	n/a	n/a	n/a
VecEmpty(void)	64.677 ms	1.728 µs	[1,1,1]	[1,1,1]	0	0	0	n/a	n/a	0	0	0	0
VecThen(int*, int*, int*, int)	64.721 ms	6.015 ms	[1,1,1]	[1,1,1]	11	44	0	n/a	n/a	0	2048	0	0
Vec50(int*, int*, int*, int)	70.74 ms	3.01 ms	[1,1,1]	[1,1,1]	11	44	0	n/a	n/a	0	1025	0	0
Vec1of32(int*, int*, int*, int)	73.754 ms	3.01 ms	[1,1,1]	[1,1,1]	11	44	0	n/a	n/a	0	1025	0	0

- Easily identify performance bottlenecks using a unified CPU and GPU trace of application activity
- Expert analysis system pin-points potential optimization opportunities
- Highlights potential performance problems at specific source-lines within application kernels
- Close integration with Nsight editor and builder for fast edit-build-profile optimization cycle
- Integrates with the new nvprof command-line profiler to enable visualization of profile data collected on headless compute nodes

New Online CUDA Resource Center

All the Latest, All in One Place

Profiling CUDA applications

Nsight must be running and at least one project must exist. Profiler cannot be used when debugging session is in progress.

Nsight Eclipse Edition profiling features are based on the NVIDIA Visual Profiler (nvvp) code. These two tools provide same features and have same user interface.

1. In the **Project Explorer** view, select project you want to profile. Make sure the project executable is compiled and no error markers are shown on the project.
2. On the main window toolbar press the **Profile** button.
3. Press **Yes** when Nsight prompts to switch to the **Profile** perspective.

Nsight will switch to the **Profile** perspective and will display application execution timeline.

Figure 3. Profiling CUDA Application

Name	Value
Duration	108.105 ms
Session	Timeline
Timeline	76.114 ms
Kernel	7.584 μs
Utilization	100%
Iterations	1

Analysis Results:

- Low Compute Utilization [7.584 μs / 108.105 ms = 0%]
The multiprocessors of one or more GPUs are mostly idle.
- Low Compute / Memory Efficiency [7.584 μs / 8.292 μs = 0.913]
The amount of time performing compute is low relative to the amount of time required for memory.
- Low Memory/Compute Overlap [0 ms / 7.584 μs = 0%]
The percentage of time when memory is being performed in parallel with compute is low.
- Inefficient Memory Size

- Programming Guides
- API Reference
- Library Manuals
- Code Samples
- Tools Manuals
- Platform Specs

Over 1600 Files!

<http://docs.nvidia.com>

Summary

1

Dynamic Parallelism accelerates more of your application on Kepler GPUs

2

GPU Callable Libraries enable 3rd party ecosystem and customization via callbacks and plug-ins

3

GPUDirect support for RDMA enables high performance, low-latency communication between GPUs and other PCIe devices

4

New Nsight™ Eclipse Edition increases programmer productivity for world's most popular parallel programming platform

www.nvidia.com/getcuda

Thanks!