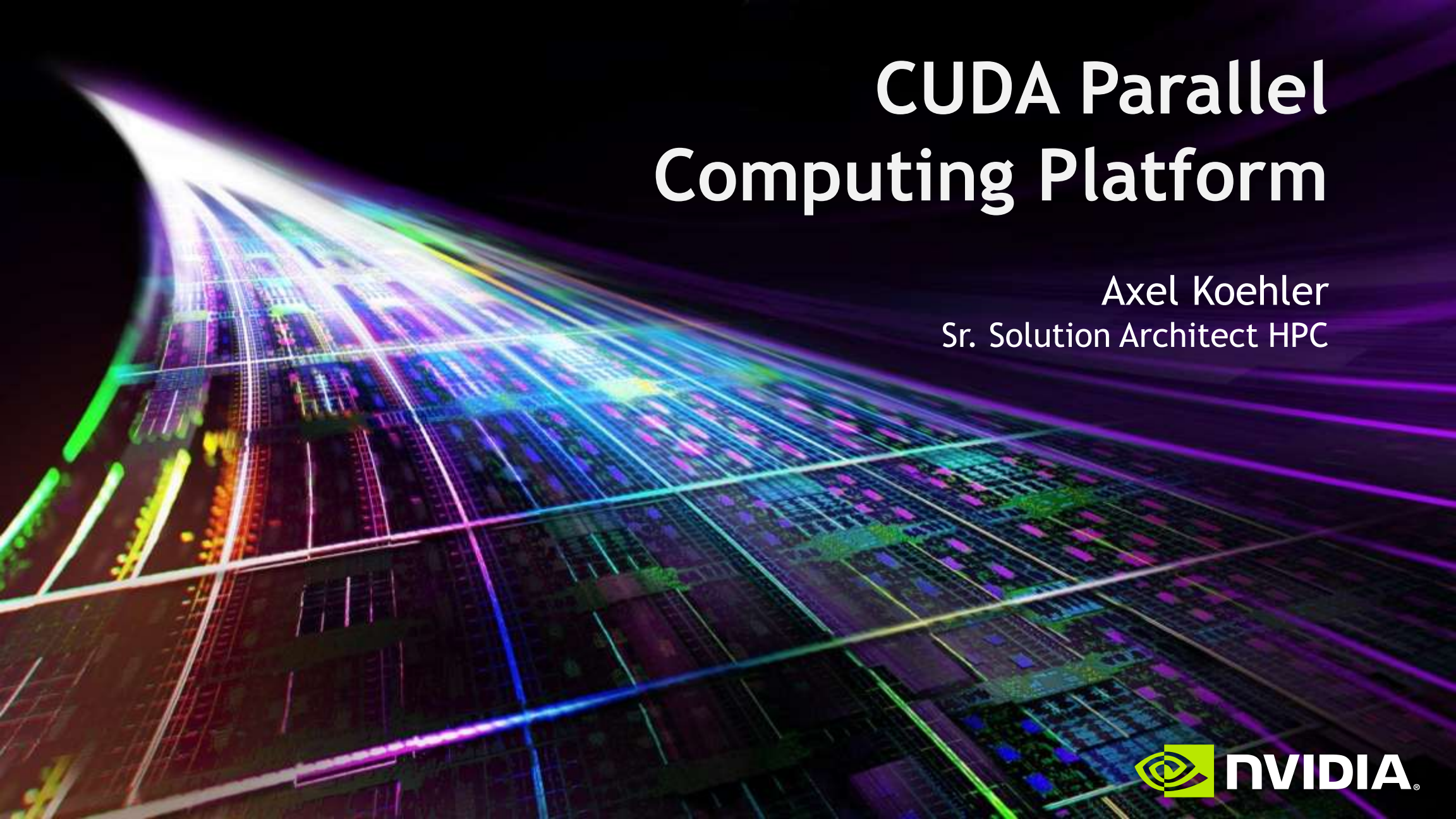


CUDA Parallel Computing Platform

Axel Koehler
Sr. Solution Architect HPC

NVIDIA

CUDA is a powerful development platform for parallel computing

Parallel Computing Platform

Multiple Programming Approaches

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

Development Environment

Parallel Nsight IDE
Linux, Mac and Windows
GPU Debugging and Profiling

CUDA-GDB debugger
NVIDIA Visual Profiler

Third Party Tools
DDT, TotalView,
Vampir, …

Compiler

Open Compiler Tools

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

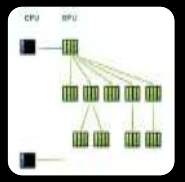
CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

OpenACC Compiler

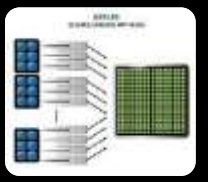
Hardware Capabilities

SMX

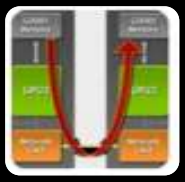
DynamicParallelism



HyperQ



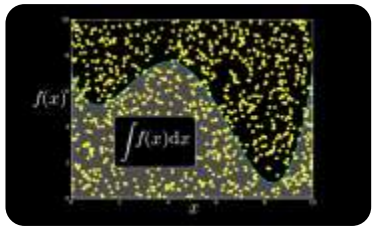
GPUDirect



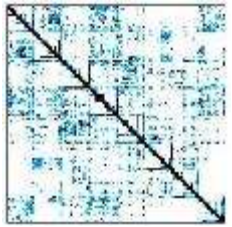
GPU Accelerated Libraries

"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

Matrix Algebra on GPU and Multicore

NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

CUSP

Sparse Linear Algebra

libjacket

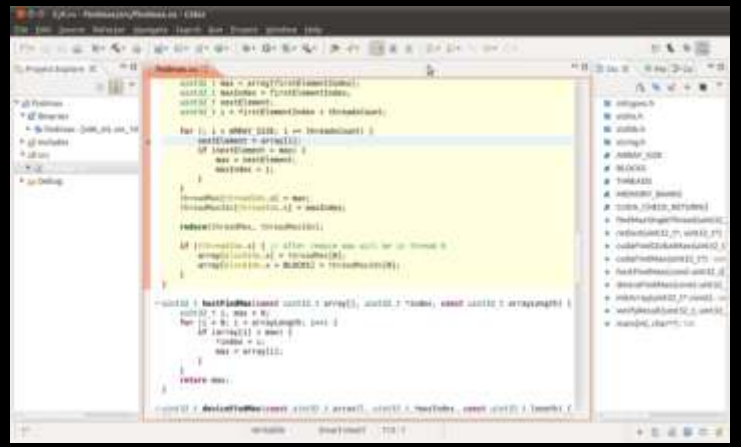
Building-block Algorithms for CUDA

Thrust

C++ STL Features for CUDA

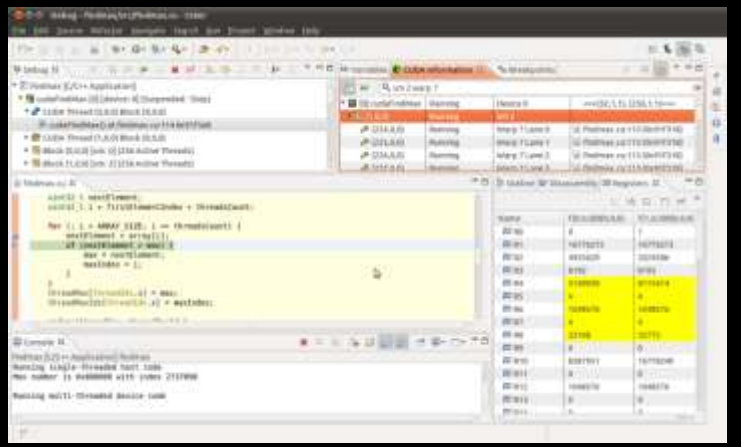
NVIDIA® Nsight™ Eclipse Edition

Power of GPU Computing + Productivity of Eclipse



CUDA-Aware Editor

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



Nsight Debugger

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

Nsight Profiler

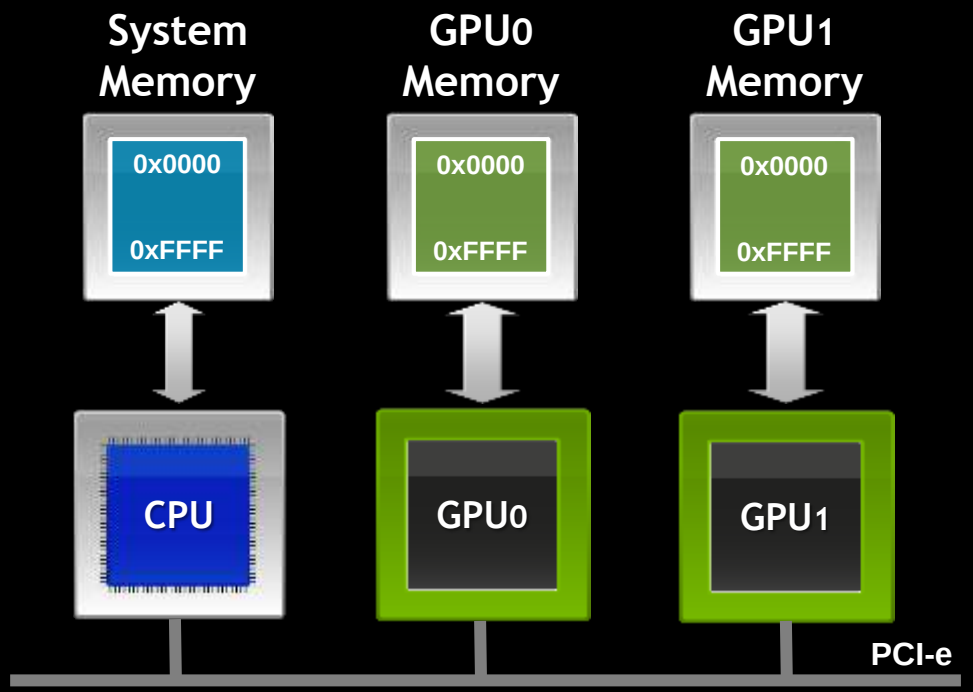
- Quickly identifies performance issues
- Integrated expert system
- Automated analysis
- Source line correlation

CUDA exposes several hardware features that are not available with other APIs

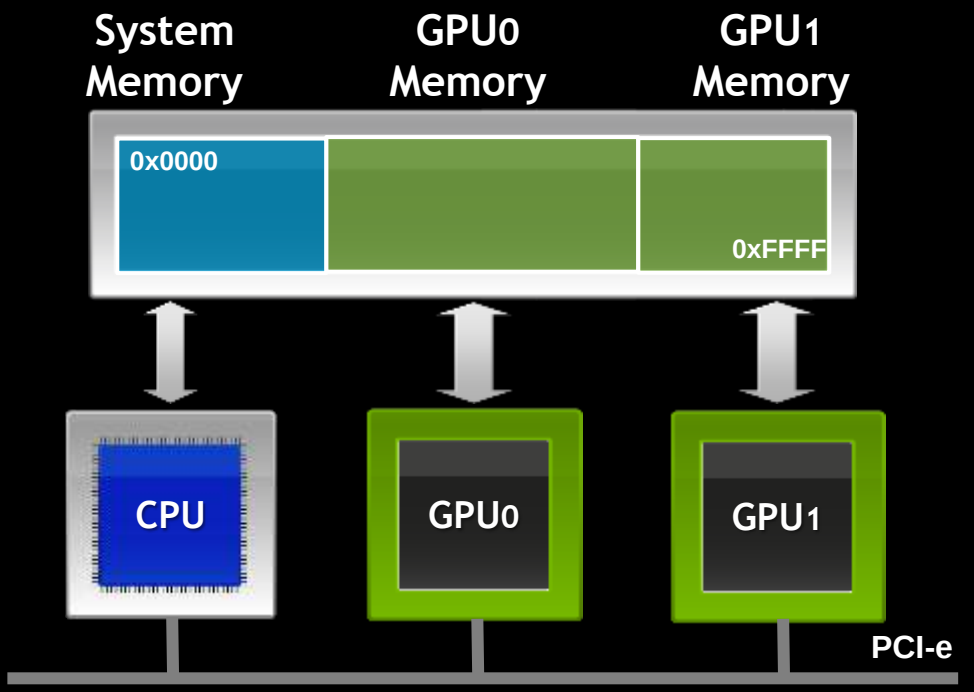
Unified Virtual Addressing

Easier to Program with Single Address Space

No UVA: Multiple Memory Spaces



UVA : Single Address Space



GPU-aware MPI

- **Support GPU to GPU communication through standard MPI interfaces without exposing low level details to the programmer (make MPI implementations aware of GPU pointers)**
  - **e.g. enable MPI_Send, MPI_Recv from/to GPU memory**
  - **Made possible by Unified Virtual Addressing (UVA) in CUDA 4.0**
  - **MVAPICH2, OpenMPI, IBM Platform MPI**

Code without MPI integration

At Sender:

```
cudaMemcpy(s_buf, s_device, size, cudaMemcpyDeviceToHost);
MPI_Send(s_buf, size, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
```

At Receiver:

```
MPI_Recv(r_buf, size, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &req);
cudaMemcpy(r_device, r_buf, size, cudaMemcpyHostToDevice);
```

Code with MPI integration

At Sender:

```
MPI_Send(s_device, size, …);
```

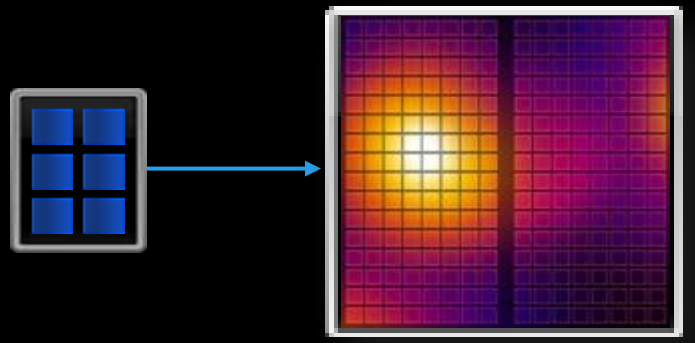
At Receiver:

```
MPI_Recv(r_device, size, …);
```

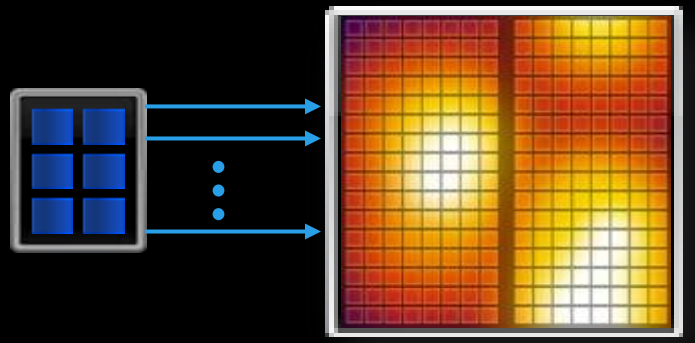
Hyper-Q / Dynamic Parallelism

Hyper-Q

FERMI
1 Work Queue

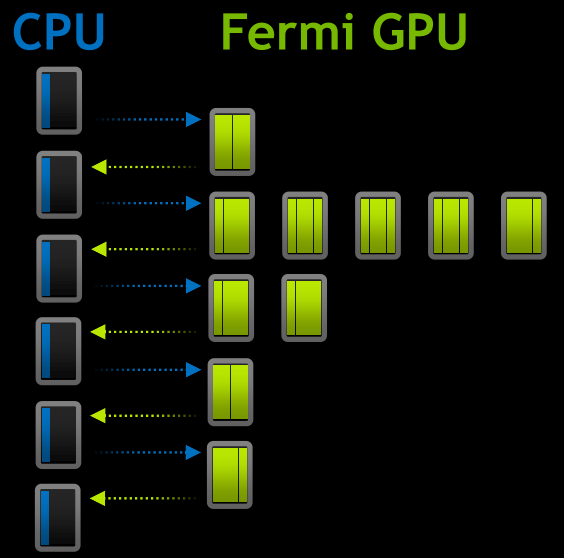


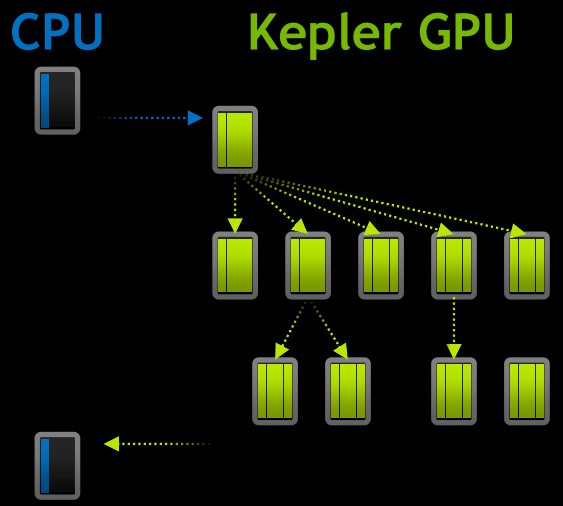
KEPLER
32 Concurrent Work Queues



Dynamic Parallelism

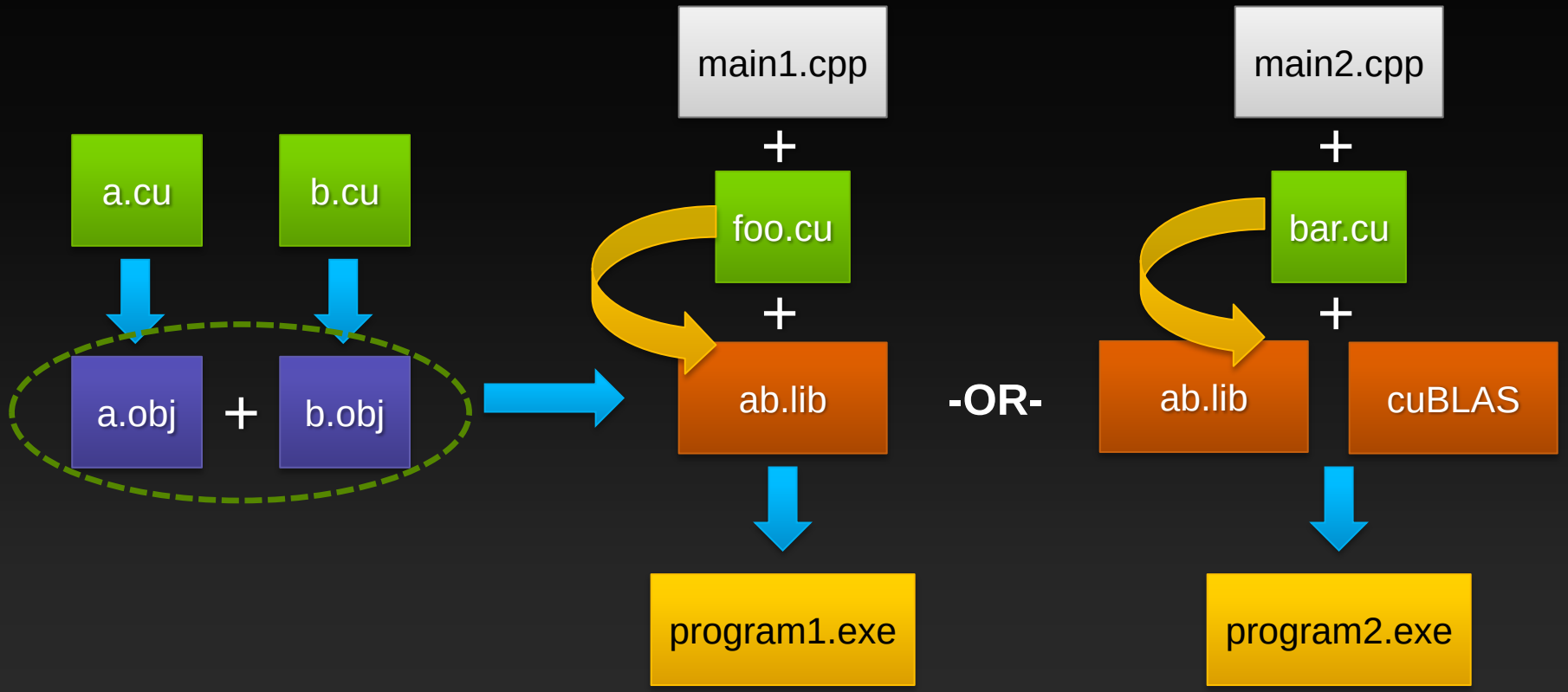
Less Back-Forth, Simpler Code





GPU Callable Libraries

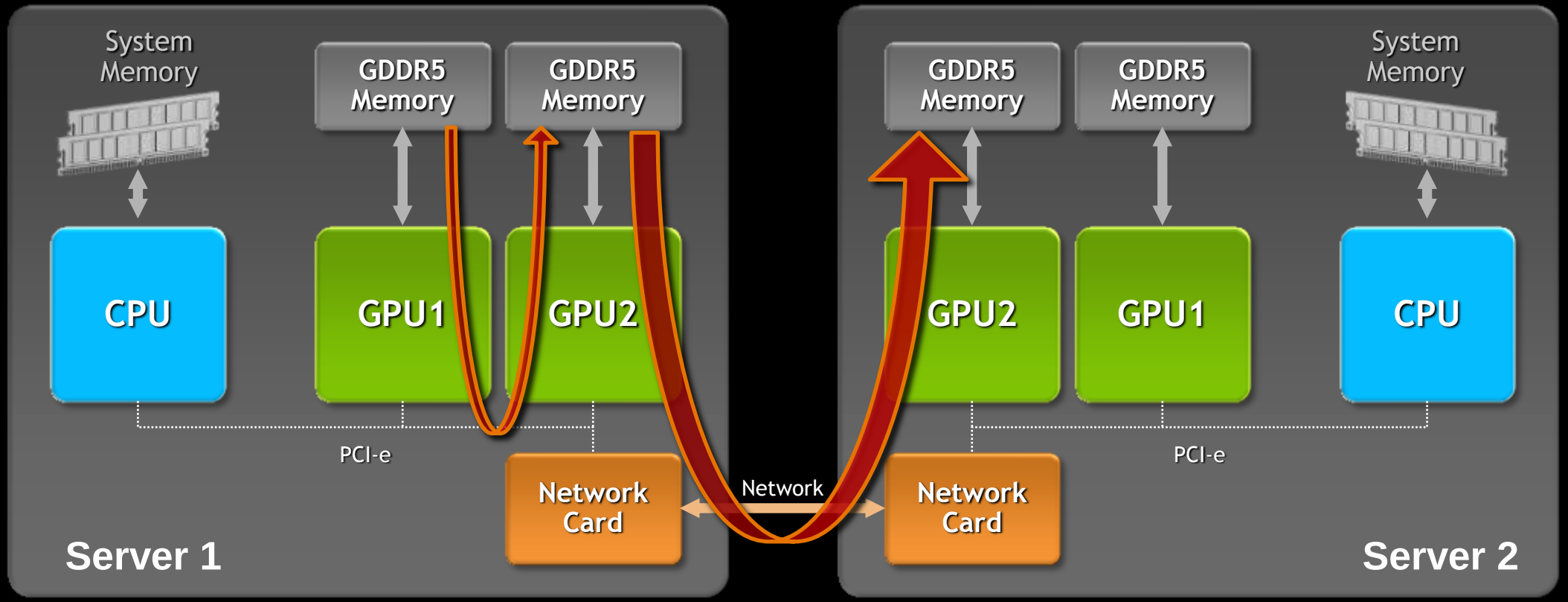
Direct Access to BLAS and other Libraries from GPU Code



Source files compiled separately to create independent object files

Linker creates GPU Callable Libraries and links with CUDA code

Kepler Enables Full NVIDIA GPUDirect™ RDMA



- GPUDirect RDMA is a general approach and can also be used in conjunction with other PCIe devices (eg. flash memory devices)

CUDA opens up to new languages and new hardware platforms

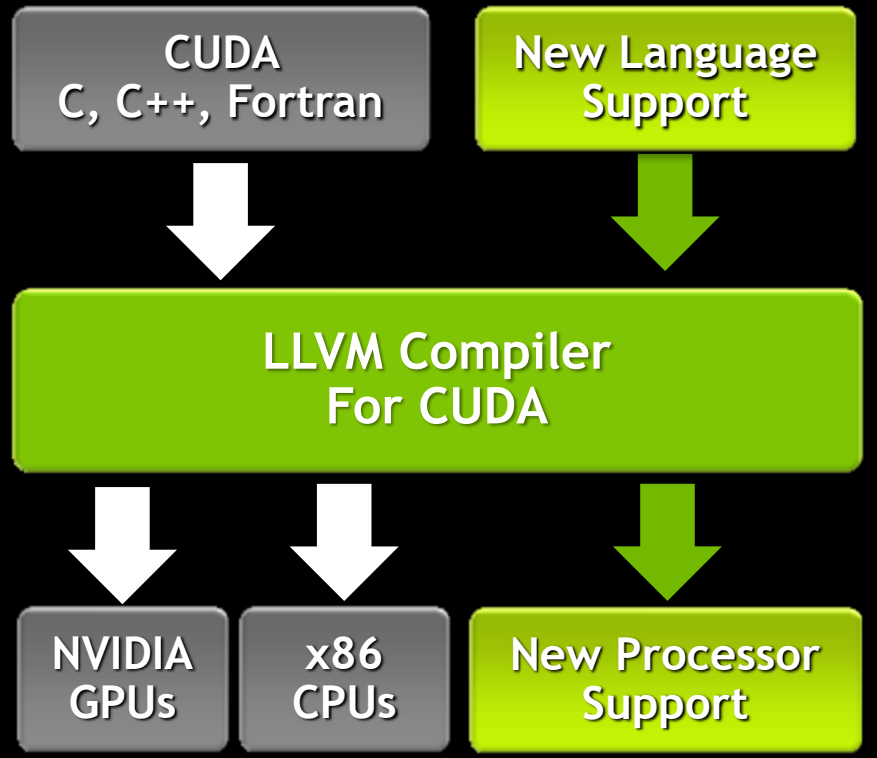
CUDA Compiler Contributed to Open Source LLVM

Developers want to build front-ends for

Java, Python, R, DSLs

Target other processors like

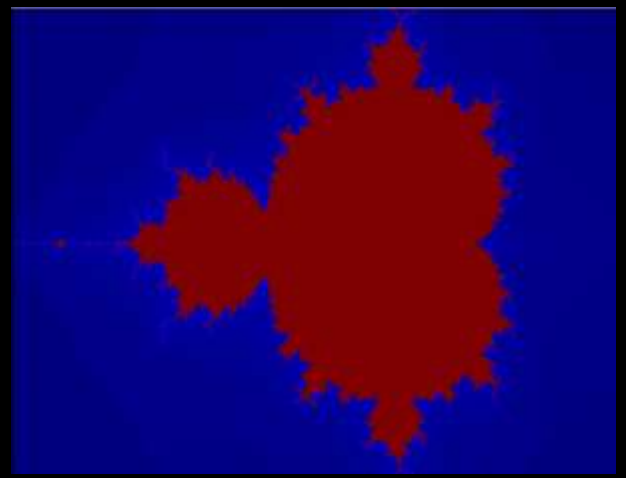
ARM, FPGA, GPUs, x86



Enabling More Programming Languages

CUDA Python

NumbaPro

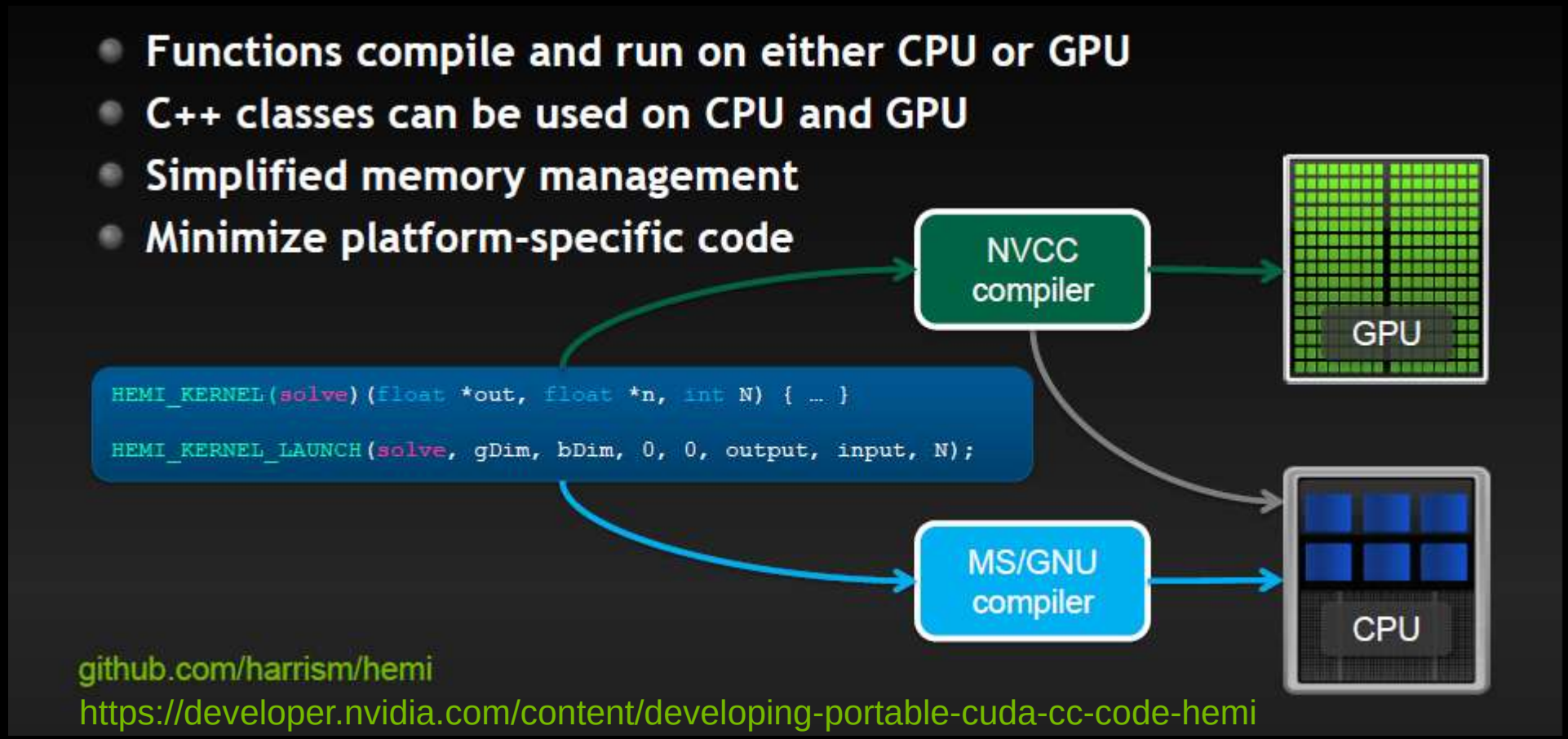


```
@cuda.jit(restype=uint8, argtypes=[f8, f8, uint32], device=True)
def mandel(x, y, max_iters):
  zr, zi = 0.0, 0.0
  for i in range(max_iters):
    newzr = (zr*zr-zi*zi)+x
    zi = 2*zr*zi+y
    zr = newzr
    if (zr*zr+zi*zi) >= 4:
       return i
  return 255

@cuda.jit(argtypes=[uint8[:,:], f8, f8, f8, f8, uint32])
def mandel_kernel(img, xmin, xmax ymin, ymax, iters):
  x, y = cuda.grid(2)
  if x < img.shape[0] and y < img.shape[1]:
    img[y, x] = mandel(min_x+x*((max_x-min_x)/img.shape[0]),
                       min_y+y*((max_y-min_y)/img.shape[1]), iters)

gimage = np.zeros((1024, 1024), dtype = np.uint8)
d_image = cuda.to_device(gimage)
mandel_kernel[(32,32), (32,32)](d_image, -2.0, 1.0, -1.0, 1.0, 20)
d_image.to_host()
```

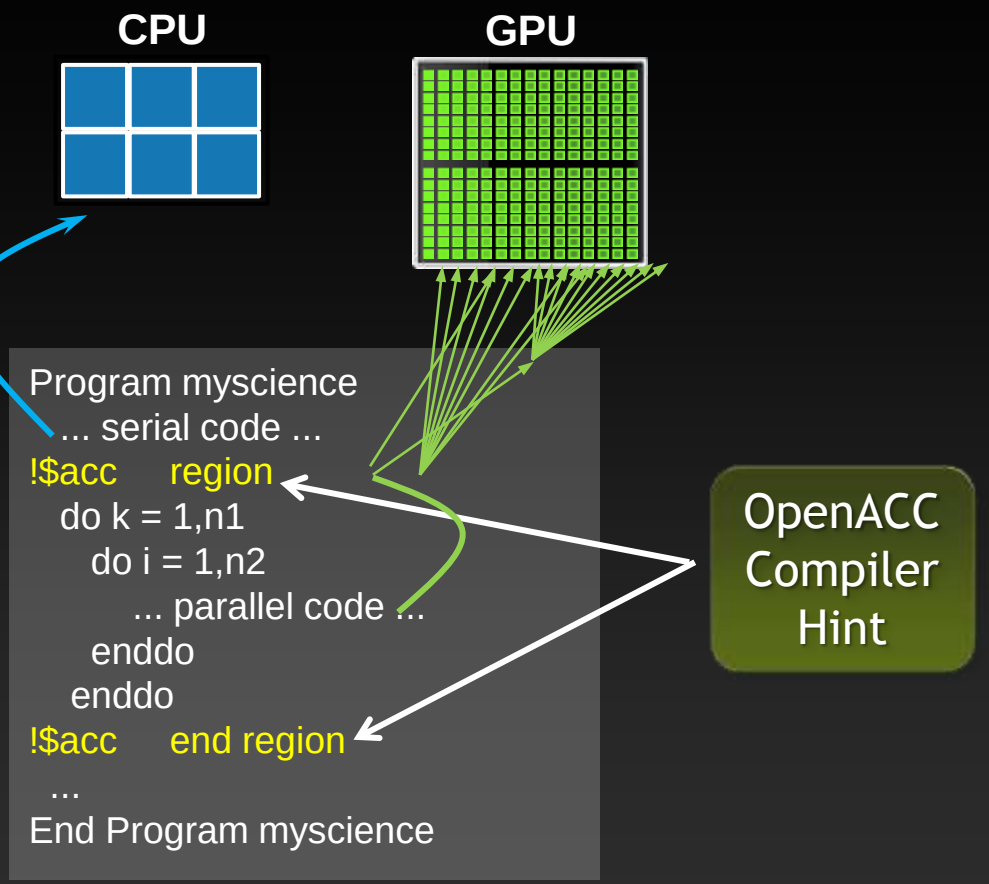
CUDA Programming, Python Syntax

Platform Independent Libraries on Top of CUDA

- **HEMI: Fully Portable C++ GPU Computing**



OpenACC Directives



Easy, Open, Powerful

- Simple Compiler hints
- Works on multicore CPUs & many core GPUs
- Compiler Parallelizes code
- Future Integration into OpenMP standard planned

http://www.openacc.org

NVIDIA CRAY PGI CAPS

Summary

- CUDA is a powerful development platform for parallel computing
  - Smart compilers, Libraries of common routines , Integrated development environments (IDEs) , Profiling, correctness-checking, and debugging
- CUDA exposes several hardware features that are not available via other APIs
- Open Compiler Architecture and Compiler SDK play a very important role to broaden the GPU platform (see next presentation)


Thank you.
Questions?
Axel Koehler
akoehler@nvidia.com
NVIDIA