Modern CUDA Features

November 2014 SciNet TechTalk

Scott Northrup
SciNet
www.scinethpc.ca

November 19, 2014

Outline

Outline

GPU Programming

GPGPU Languages

- OpenGL, DirectX (Graphics only)
- OpenCL (1.0, 1.1, 2.0) Open Standard
- CUDA (NVIDIA proprietary)
- OpenACC
- OpenMP 4.0

CUDA
What is it?

Compute Unified Device Architecture

- parallel computing platform and programming model created by NVIDIA
- Language Bindings
  - C/C++ `nvcc` compiler (works with gcc/intel)
  - Fortran (PGI compiler)
  - Others (pyCUDA,jCUDA, etc.)
- CUDA Versions (V1.0 - 6.5)
- Hardware Compute Capability (1.0 - 5.2)
- Platforms
  - x86_64 - Windows, OSX, Linux
  - ARM v7,v9 - Linux
  - POWER 8,9 - Linux

NVIDIA GPGPU History - CUDA

CUDA Software Releases

- CUDA Toolkit 1.0 (June 2007)
- CUDA Toolkit 2.0 (Aug 2008)
- CUDA Toolkit 3.0 (March 2010)
- CUDA Toolkit 4.0 (May 2011)
- CUDA Toolkit 5.0 (Oct 2012)
- CUDA Toolkit 6.0 (April 2014)
- CUDA Toolkit 6.5 (August 2014)

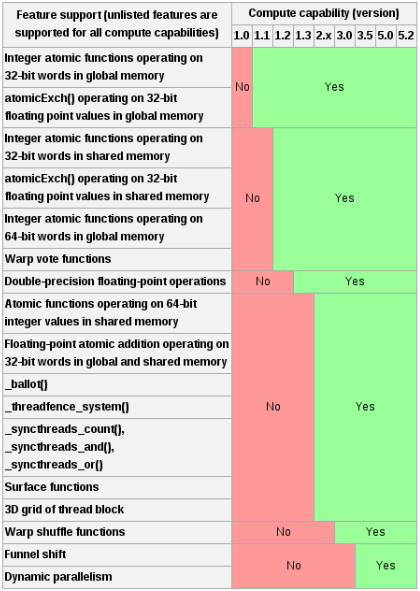
NVIDIA GPGPU History - Hardware

GPU Generations - Compute Capability

CC	Arch	Chip	Tesla Card
1.0-1.3	Tesla	GT200	C1060, S1070, M1060
2.0-2.1	Kepler	GF110	C20[50/70] M20[50/70/75/90]
3.0-3.5	Fermi	GK110/210	K20,K20x,K40,K80
5.0-5.2	Maxwell	GM204	GeForce GTX 980*
?	Pascal	(2016)	?
?	Volta	(2017)	?

*NOTE: non-Tesla commodity graphics card

Compute Capability



Feature support (unlisted features are supported for all compute capabilities)	Compute capability (version)								
	1.0	1.1	1.2	1.3	2.x	3.0	3.5	5.0	5.2
Integer atomic functions operating on 32-bit words in global memory	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
atomicExch() operating on 32-bit floating point values in global memory	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integer atomic functions operating on 32-bit words in shared memory	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
atomicExch() operating on 32-bit floating point values in shared memory	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Integer atomic functions operating on 64-bit words in global memory	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Warp vote functions	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Double-precision floating-point operations	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Atomic functions operating on 64-bit integer values in shared memory	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Floating-point atomic addition operating on 32-bit words in global and shared memory	No	No	No	No	Yes	Yes	Yes	Yes	Yes
_ballot()	No	No	No	No	Yes	Yes	Yes	Yes	Yes
_threadfence_system()	No	No	No	No	Yes	Yes	Yes	Yes	Yes
_syncthreads_count(), _syncthreads_and(), _syncthreads_or()	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Surface functions	No	No	No	No	Yes	Yes	Yes	Yes	Yes
3D grid of thread block	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Warp shuffle functions	No	No	No	No	No	Yes	Yes	Yes	Yes
Funnel shift	No	No	No	No	No	No	Yes	Yes	Yes
Dynamic parallelism	No	No	No	No	No	No	Yes	Yes	Yes

http://en.wikipedia.org/wiki/CUDA

Compute Canada GPGPU Resources

GPU Systems

- Westgrid: Parallel
  - 60 nodes (3x NVIDIA M2070)
- SharcNet: Monk
  - 54 nodes (2x NVIDIA M2070)
- SciNet: Gravity, ARC
  - 49 nodes (2x NVIDIA M2090)
  - 8 nodes (2x NVIDIA M2070)
  - 1 node (1x NVIDIA K20) - arcX
- Calcul Quebec: Guillimin
  - 50 nodes (2x NVIDIA K20)
- Calcul Quebec: Helios
  - 15 nodes (8x NVIDIA K20)

Outline

What's in CUDA 5 and 6

New in 5.x

- Dynamic Parallelism (CC $>$ 3.0)
- GPU Object Linking (non-monolithic compilation)
- GPU Direct (RDMA on IB)
- Arch: IBM Power 8, ARM v7 (5.5)
- nvprof Profiler

What's in CUDA 5 and 6

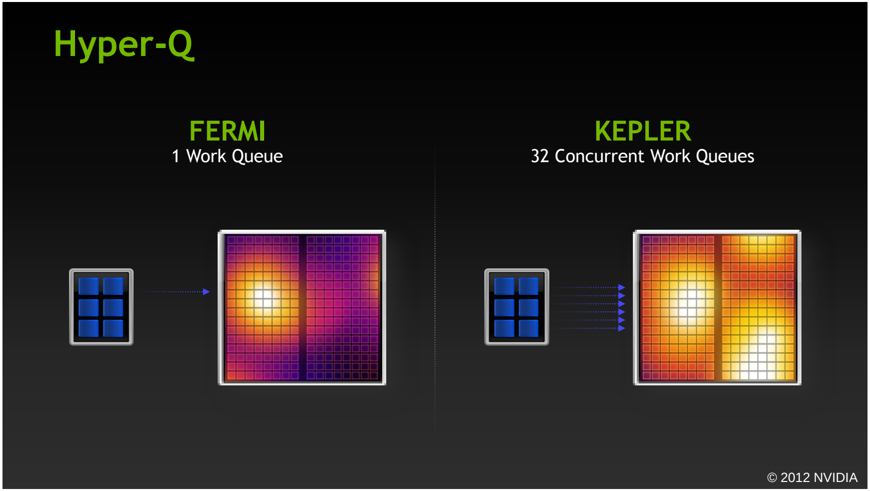
New in 5.x

- Dynamic Parallelism (CC $>$ 3.0)
- GPU Object Linking (non-monolithic compilation)
- GPU Direct (RDMA on IB)
- Arch: IBM Power 8, ARM v7 (5.5)
- nvprof Profiler

New in 6.x

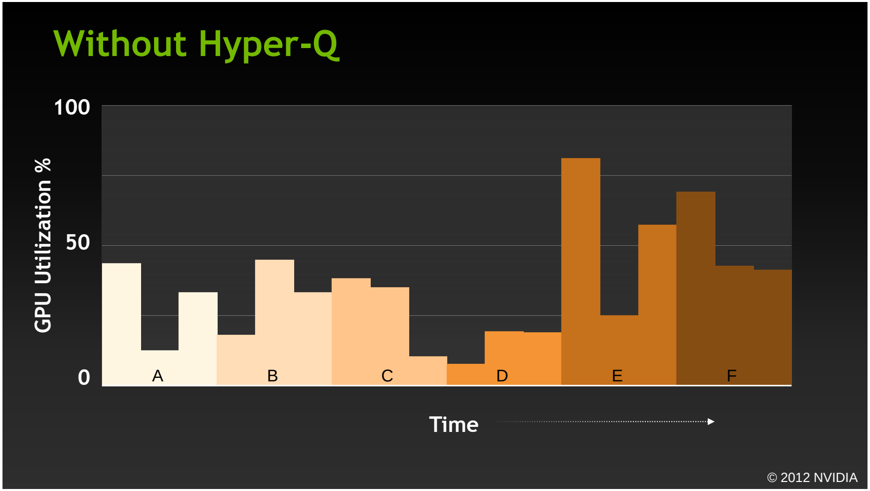
- Unified Memory (CC $>$ 3.0)
- Extended (XT) libraries (multi-GPU cuBLAS, cuFFT)
- GPU Direct (RDMA with MPI)
- Occupancy Calculator APIs (6.5)
- Visual Profiler Updates (6.5)
- Arch: ARM 64 (6.5)

Dynamic Parallelism



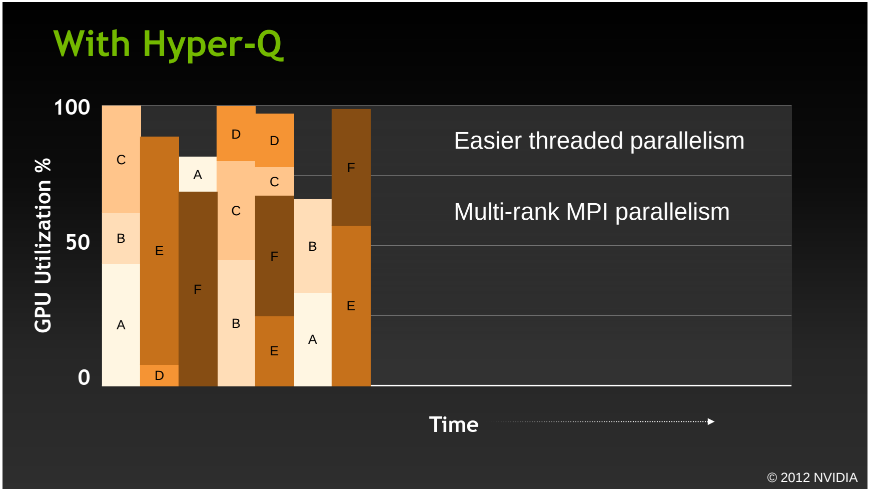
Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



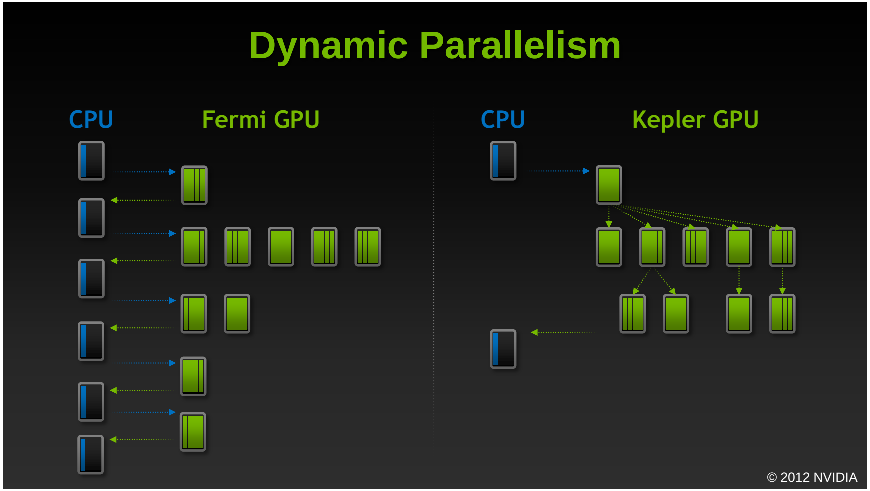
Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



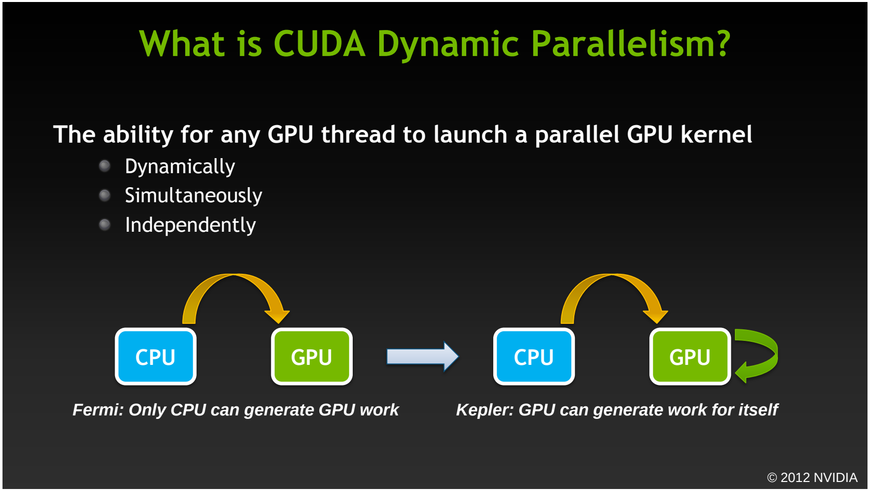
Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



What is CUDA Dynamic Parallelism?

The ability for any GPU thread to launch a parallel GPU kernel

- Dynamically
- Simultaneously
- Independently

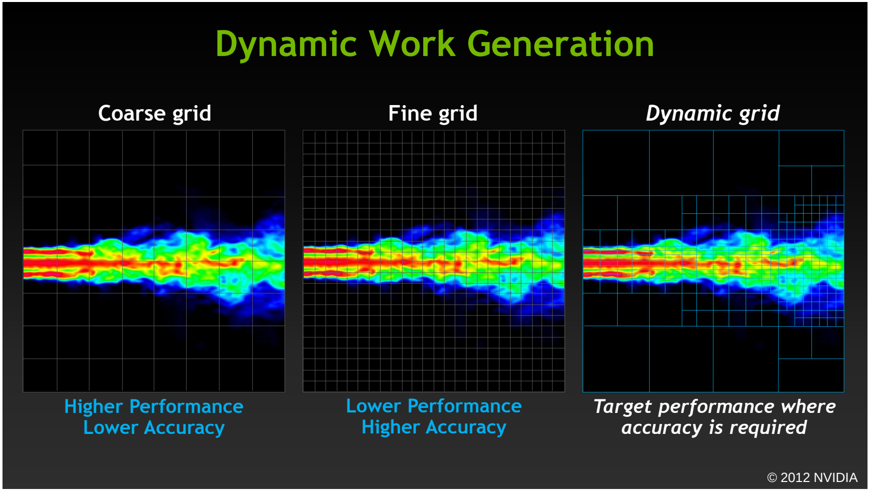
CPU
GPU
CPU
GPU

Fermi: Only CPU can generate GPU work

Kepler: GPU can generate work for itself

Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



Dynamic Work Generation

Coarse grid

Fine grid

Dynamic grid

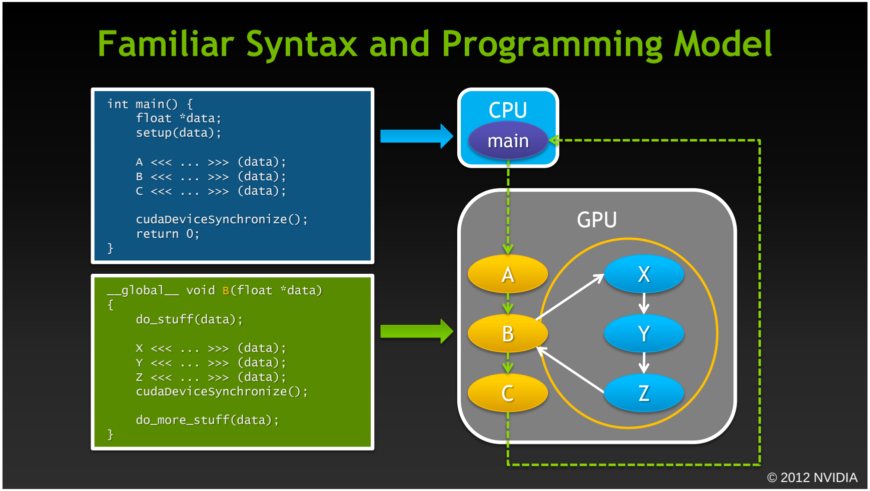
Higher Performance
Lower Accuracy

Lower Performance
Higher Accuracy

Target performance where accuracy is required

Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



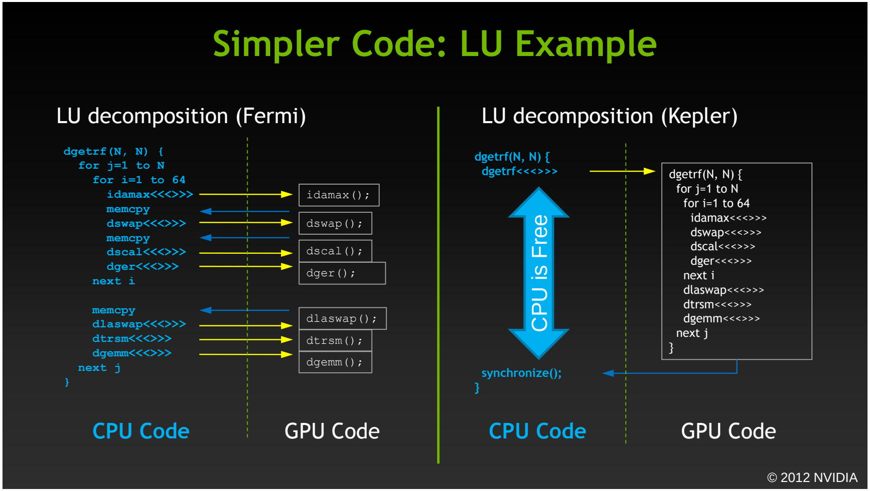
Familiar Syntax and Programming Model

```
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);
}
```

Images: Mark Ebersole, NVIDIA

Dynamic Parallelism



Simpler Code: LU Example

LU decomposition (Fermi)

```
dgetrf(N, N) {
  for j=1 to N
    for i=1 to 64
      idamax<<<>>>
      memcpy
      dswap<<<>>>
      memcpy
      dscal<<<>>>
      dger<<<>>>
    next i

    memcpy
    dlaswap<<<>>>
    dtrsm<<<>>>
    dgemm<<<>>>
  next j
}
```

```
idamax();
dswap();
dscal();
dger();
dlaswap();
dtrsm();
dgemm();
```

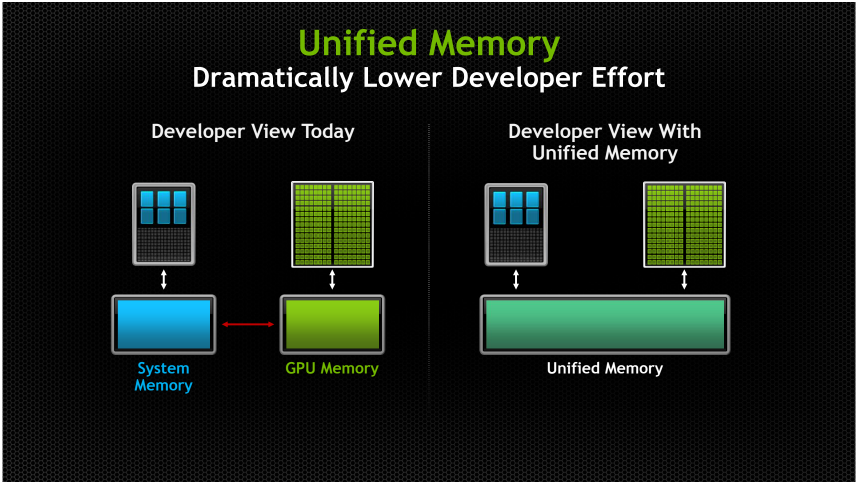
LU decomposition (Kepler)

```
dgetrf(N, N) {
 dgetrf<<<>>>

 synchronize();
}
```

```
dgetrf(N, N) {
  for j=1 to N
    for i=1 to 64
      idamax<<<>>>
      dswap<<<>>>
      dscal<<<>>>
      dger<<<>>>
    next i
    dlaswap<<<>>>
    dtrsm<<<>>>
    dgemm<<<>>>
  next j
}
```

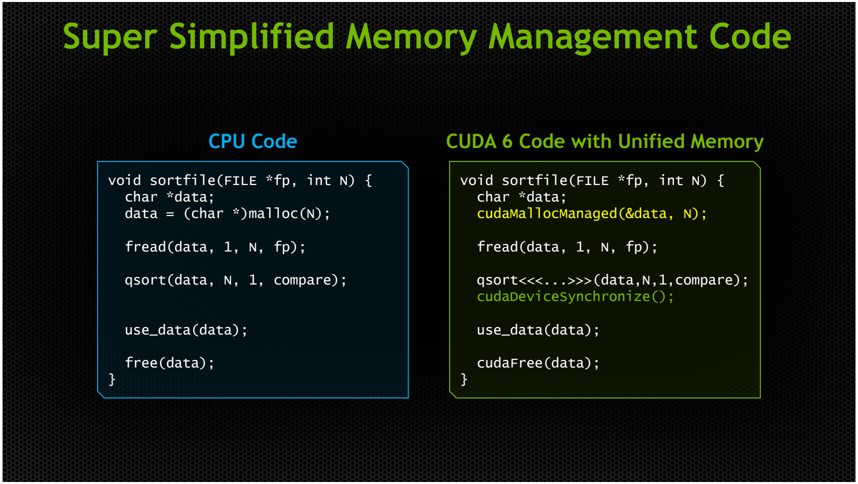
Images: Mark Ebersole, NVIDIA

Unified Memory



Images: Mark Harris, NVIDIA

Unified Memory



Super Simplified Memory Management Code

CPU Code

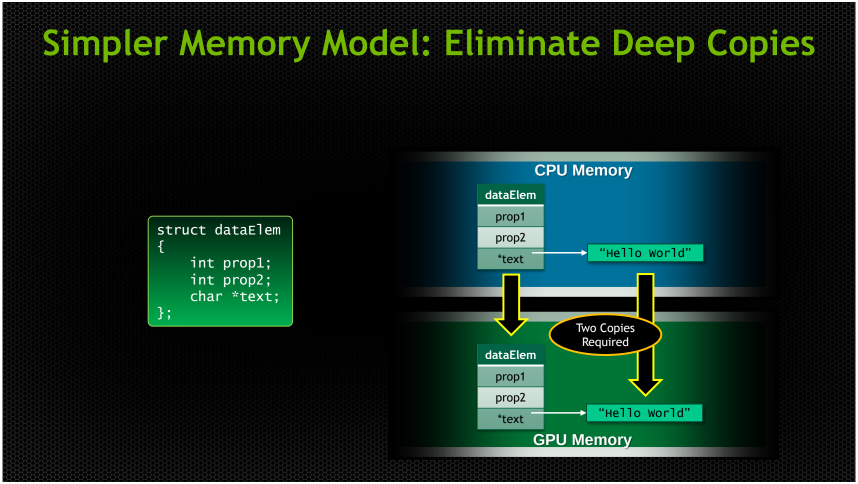
```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA 6 Code with Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

Images: Mark Harris, NVIDIA

Unified Memory

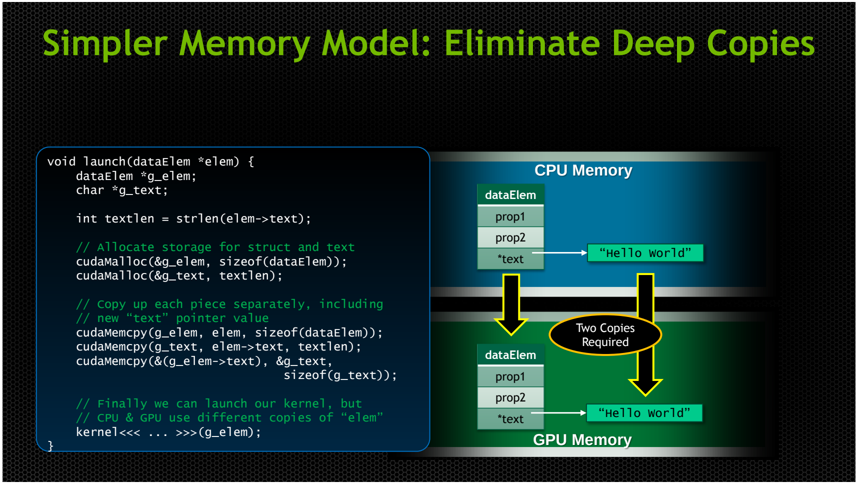


Simpler Memory Model: Eliminate Deep Copies

```
struct dataElem
{
    int prop1;
    int prop2;
    char *text;
};
```

Images: Mark Harris, NVIDIA

Unified Memory

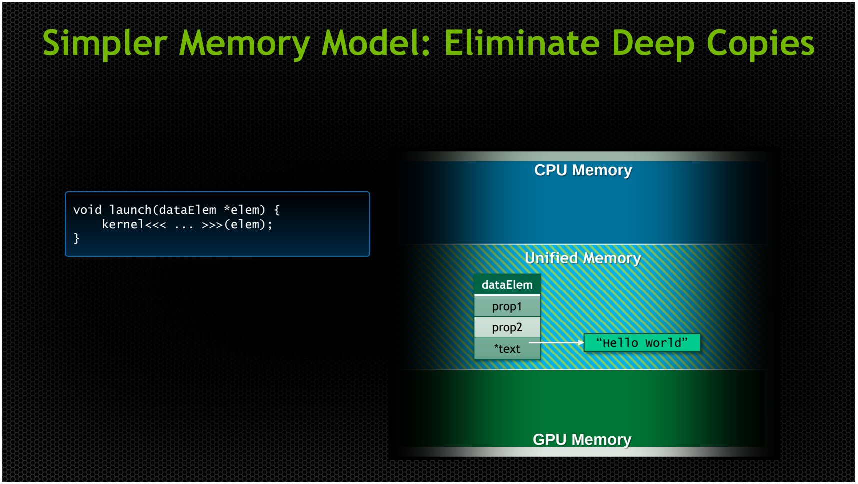


Simpler Memory Model: Eliminate Deep Copies

```
void launch(dataElem *elem) {
    dataElem *g_elem;
    char *g_text;

    int textlen = strlen(elem->text);

    // Allocate storage for struct and text
    cudaMalloc(&g_elem, sizeof(dataElem));
    cudaMalloc(&g_text, textlen);

    // Copy up each piece separately, including
    // new “text” pointer value
    cudaMemcpy(g_elem, elem, sizeof(dataElem));
    cudaMemcpy(g_text, elem->text, textlen);
    cudaMemcpy(&(g_elem->text), &g_text,
                                 sizeof(g_text));

    // Finally we can launch our kernel, but
    // CPU & GPU use different copies of “elem”
    kernel<<< ... >>>(g_elem);
}
```

Images: Mark Harris, NVIDIA

Unified Memory

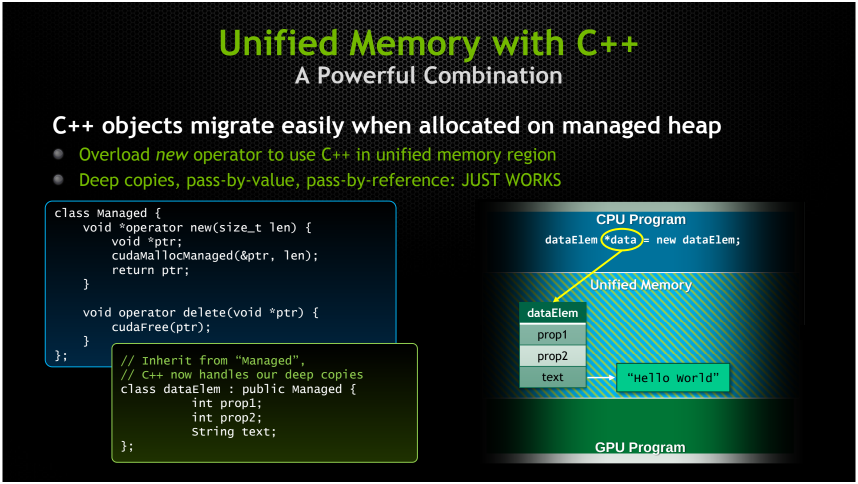


Simpler Memory Model: Eliminate Deep Copies

```
void launch(dataElem *elem) {
    kernel<<< ... >>>(elem);
}
```

Images: Mark Harris, NVIDIA

Unified Memory



Unified Memory with C++
A Powerful Combination

C++ objects migrate easily when allocated on managed heap

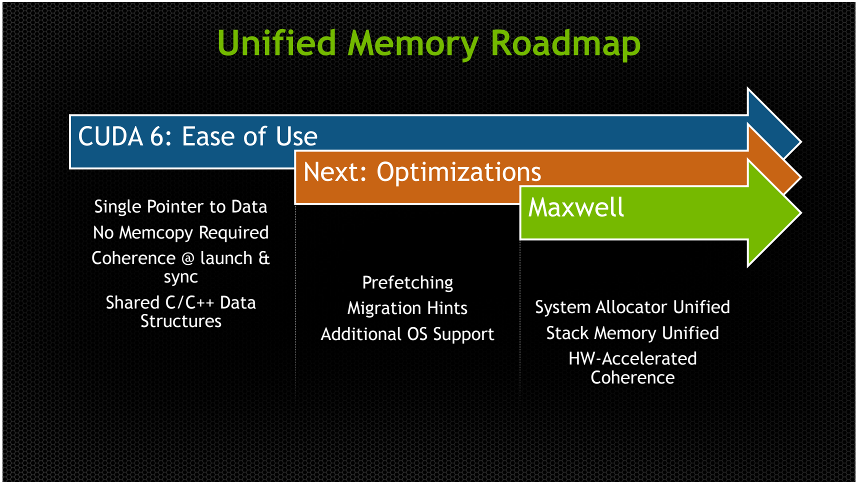
- Overload *new* operator to use C++ in unified memory region
- Deep copies, pass-by-value, pass-by-reference: JUST WORKS

```
class Managed {
    void *operator new(size_t len) {
        void *ptr;
        cudaMallocManaged(&ptr, len);
        return ptr;
    }

    void operator delete(void *ptr) {
        cudaFree(ptr);
    }
};
```

```
// Inherit from “Managed”,
// C++ now handles our deep copies
class dataElem : public Managed {
          int prop1;
          int prop2;
          String text;
};
```

Images: Mark Harris, NVIDIA

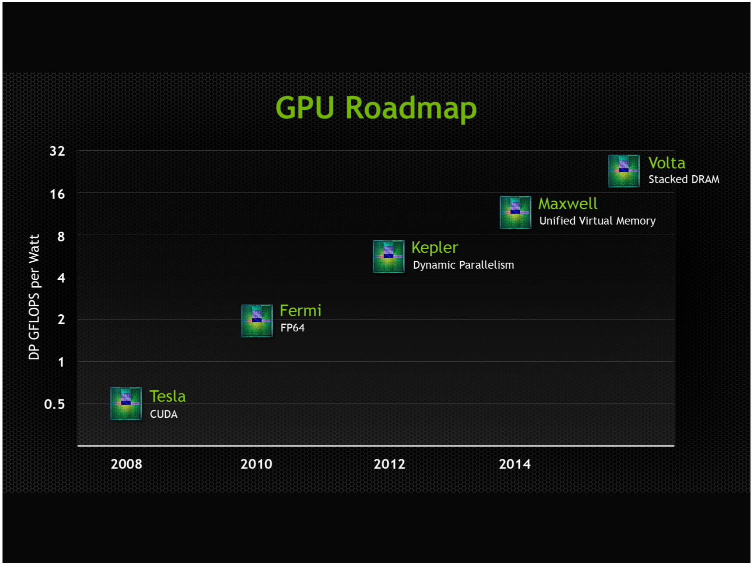
Unified Memory



Images: Mark Harris, NVIDIA

Outline

NVIDIA GPGPU Hardware Road Map

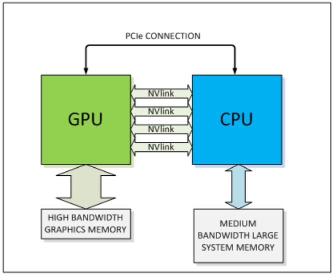


Images: nvidia.com, anandtech.com

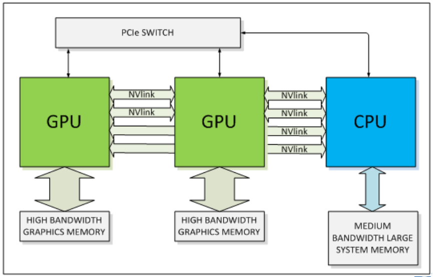
NVIDIA GPGPU Future - Hardware

New Features

- Architectures: x86_64, ARM64, Power 8 & 9
- NVLINK - 5x faster than PCIe Gen3 (Maxwell)
    - Unified Virtual Memory
- 3D Memory - faster, lower power



PCIe CONNECTION
GPU
NVlink
NVlink
NVlink
NVlink
CPU
HIGH BANDWIDTH GRAPHICS MEMORY
MEDIUM BANDWIDTH LARGE SYSTEM MEMORY



PCIe SWITCH
GPU
NVlink
NVlink
GPU
NVlink
NVlink
NVlink
NVlink
CPU
HIGH BANDWIDTH GRAPHICS MEMORY
HIGH BANDWIDTH GRAPHICS MEMORY
MEDIUM BANDWIDTH LARGE SYSTEM MEMORY

Images: nvidia.com, anandtech.com

Upcoming Education

- Online Tutorials: https://nvidia.qwiklab.com/
- SharcNet: CUDA Basics and how to Webinar, Oct. 15th, 12-1pm
- Calcul Quebec (McGill): Introduction to GPU / CUDA, Dec. 4th, 9am-4pm
- SciNet: Introduction to GPGPU with CUDA, March 4th, 2015 10pm-4pm