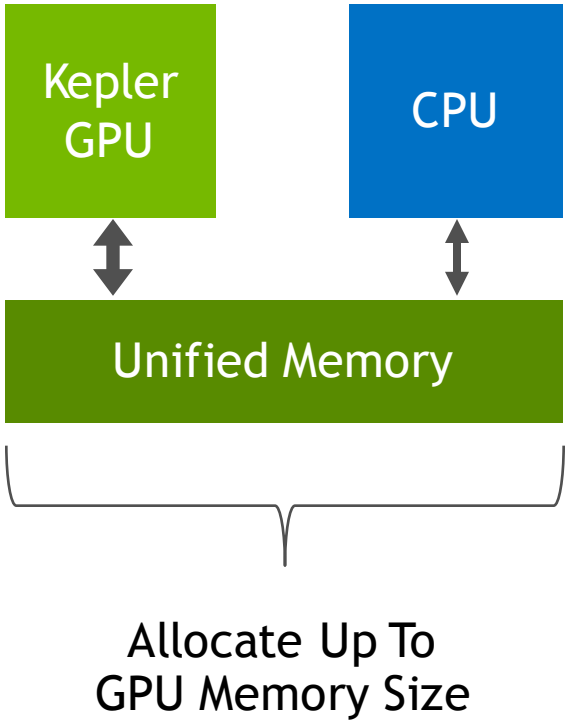
UNIFIED MEMORY

UNIFIED MEMORY

Dramatically Lower Developer Effort

CUDA 6+



Simpler Programming & Memory Model

- Single allocation, single pointer, accessible anywhere
- Eliminate need for *explicit copy*
- Greatly simplifies code porting

Performance Through Data Locality

- Migrate data to accessing processor
- Guarantee global coherence
- Still allows explicit hand tuning

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);
}
```

GREAT PERFORMANCE WITH UNIFIED MEMORY

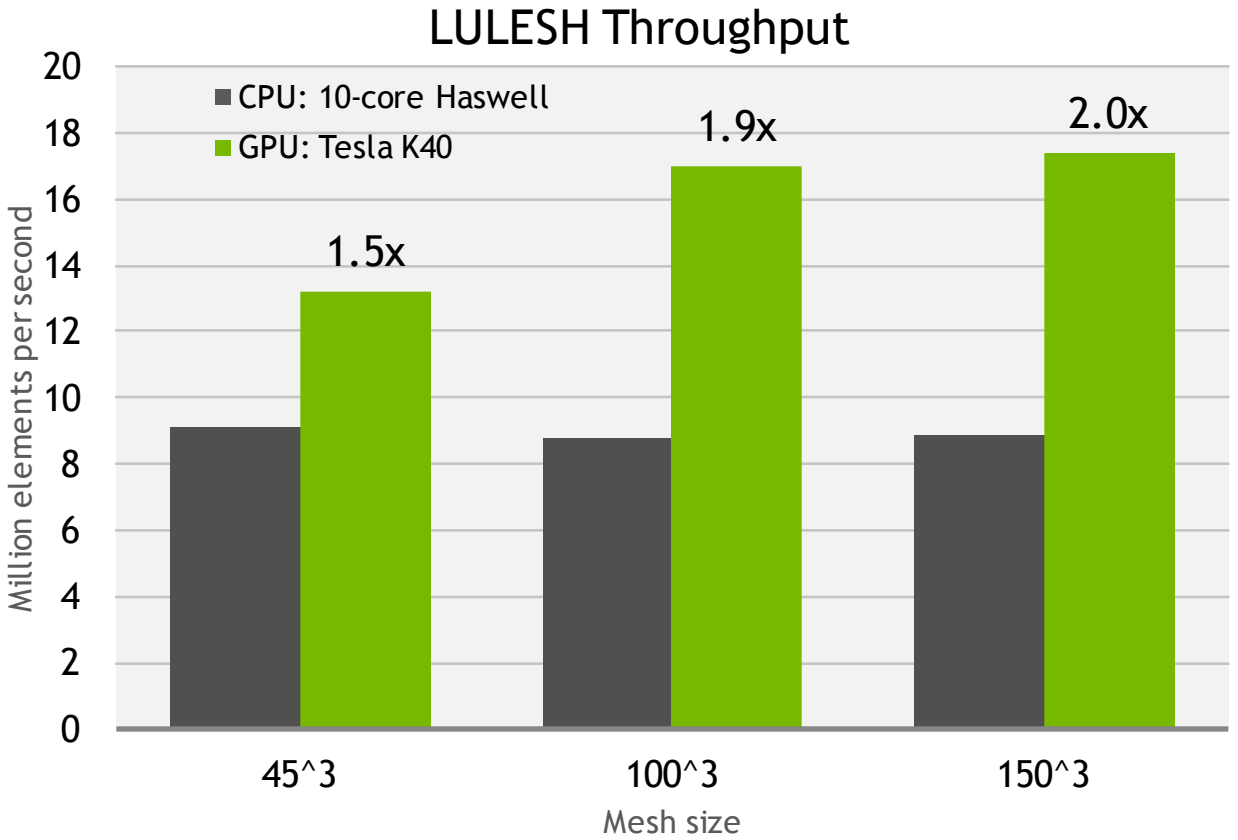
RAJA: Portable C++ Framework for parallel-for style programming

RAJA uses Unified Memory for heterogeneous array allocations

Parallel forall loops run on device

"Excellent performance considering this is a "generic" version of LULESH with no architecture-specific tuning."

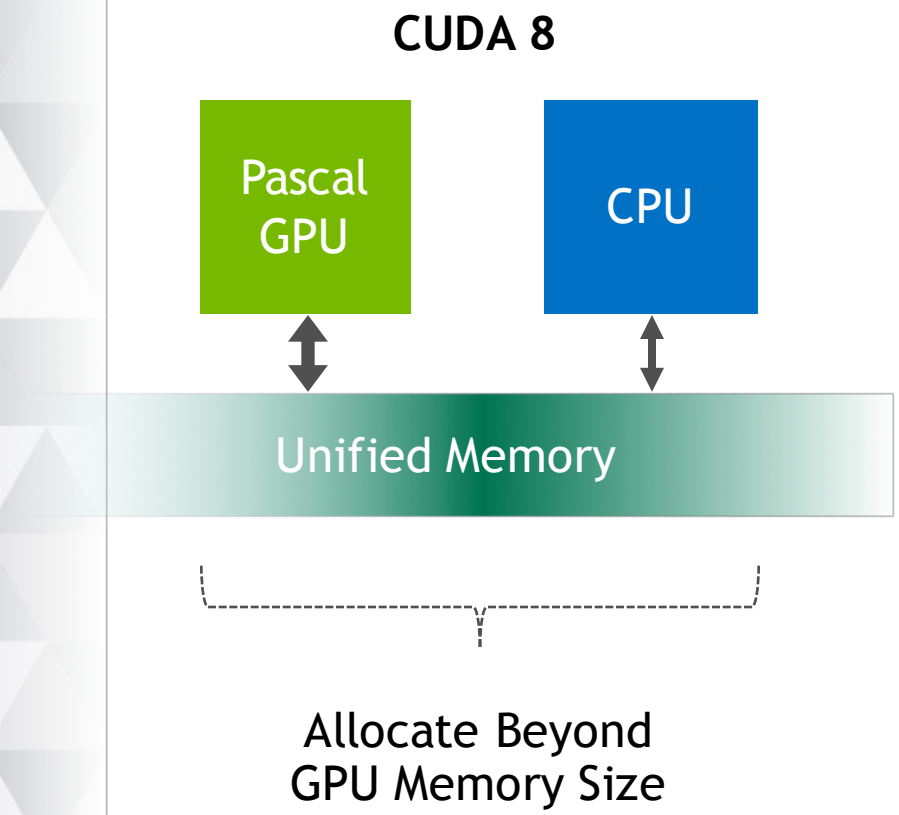
-Jeff Keasler, LLNL



GPU: NVIDIA Tesla K40, CPU: Intel Haswell E5-2650 v3 @ 2.30GHz, single socket 10-core

CUDA 8: UNIFIED MEMORY

Large datasets, simple programming, High Performance



Enable Large Data Models	Oversubscribe GPU memory Allocate up to system memory size
Simpler Data Accesss	CPU/GPU Data coherence Unified memory atomic operations
Tune Unified Memory Performance	Usage hints via cudaMemAdvise API Explicit prefetching API

UNIFIED MEMORY EXAMPLE

On-Demand Paging

```
__global__
void setValue(int *ptr, int index, int val)
{
  ptr[index] = val;
}


void foo(int size) {
  char *data;
  cudaMallocManaged(&data, size);

  memset(data, 0, size);

  setValue<<<...>>>(data, size/2, 5);
  cudaDeviceSynchronize();

  useData(data);

  cudaFree(data);
}
```

Unified Memory allocation

Access all values on CPU

Access one value on GPU

HOW UNIFIED MEMORY WORKS IN CUDA 6

Servicing CPU page faults

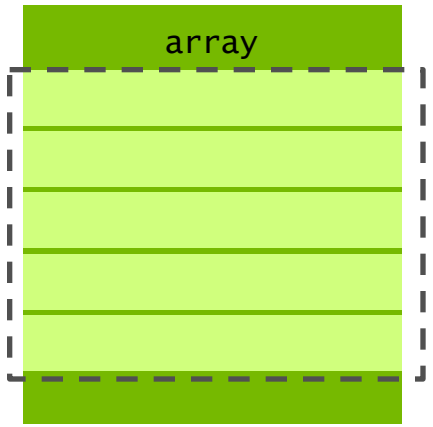
GPU Code

```
__global__
void setValue(char *ptr, int index, char val)
{
  ptr[index] = val;
}
```

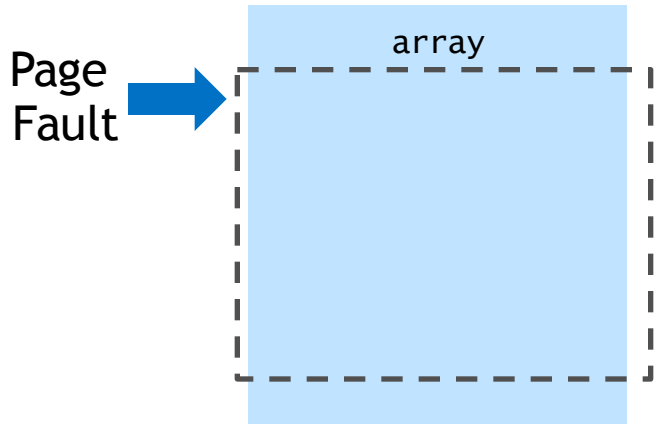
CPU Code

```
cudaMallocManaged(&array, size);
memset(array, size);
setValue<<<...>>>(array, size/2, 5);
```

GPU Memory Mapping



CPU Memory Mapping



Page Fault

Interconnect

HOW UNIFIED MEMORY WORKS ON PASCAL

Servicing CPU *and* GPU Page Faults

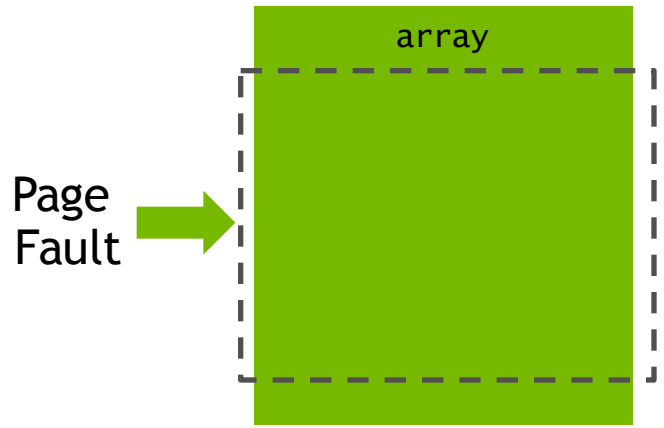
GPU Code

```
__global__
Void setValue(char *ptr, int index, char val)
{
  ptr[index] = val;
}
```

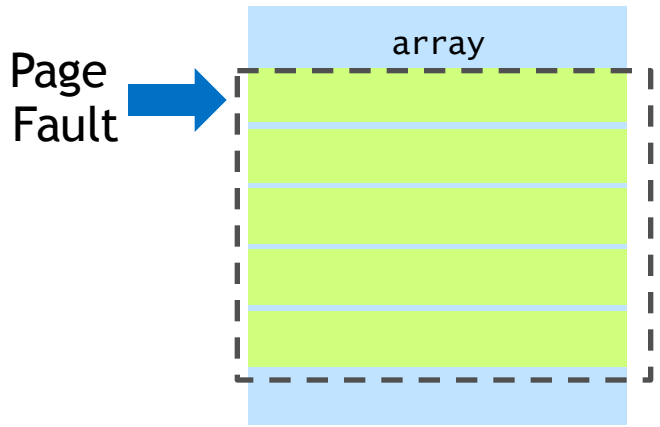
CPU Code

```
cudaMallocManaged(&array, size);
memset(array, size);
setValue<<<...>>>(array, size/2, 5);
```

GPU Memory Mapping

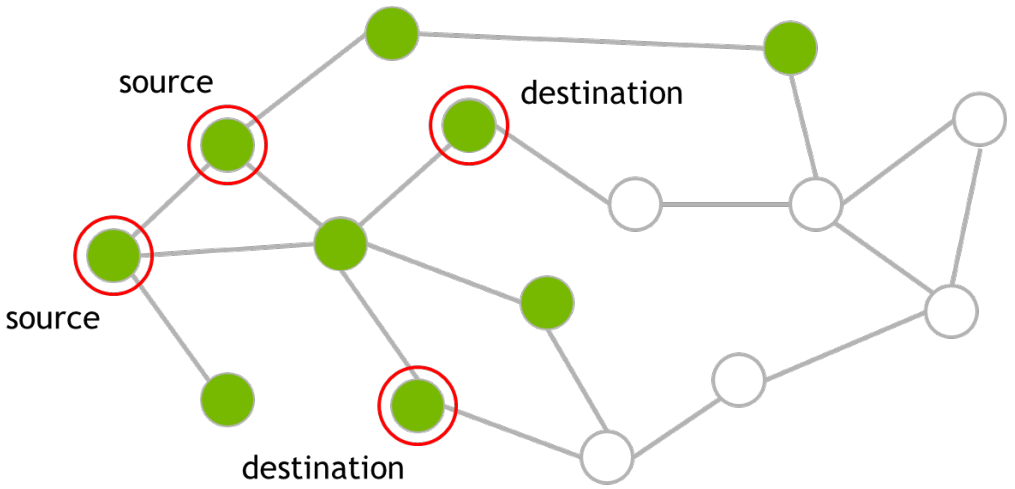


CPU Memory Mapping

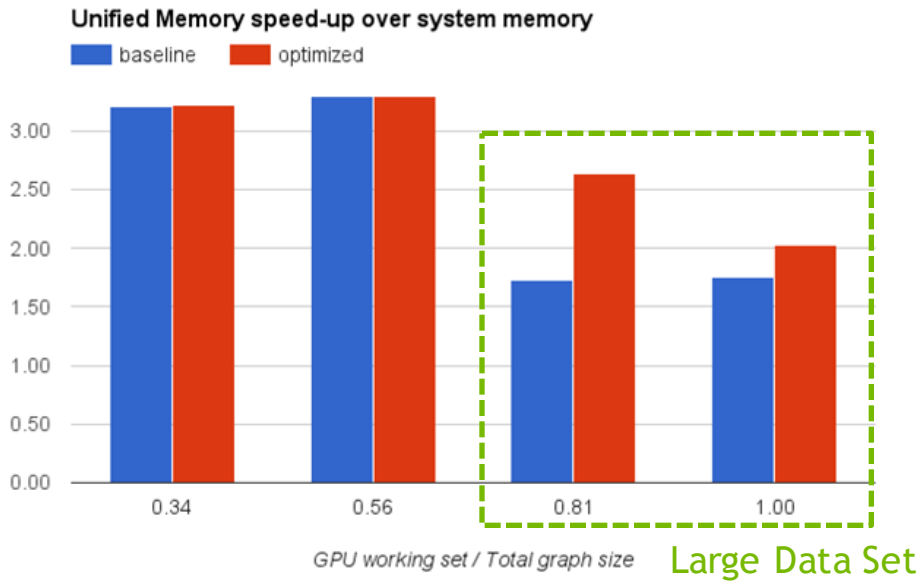


USE CASE: ON-DEMAND PAGING

Graph Algorithms



Performance over GPU directly accessing host memory (zero-copy)



Baseline: migrate on first touch
Optimized: best placement in memory

UNIFIED MEMORY ON PASCAL

GPU memory oversubscription

```
void foo() {
  // Assume GPU has 16 GB memory
  // Allocate 32 GB
  char *data;
  size_t size = 32*1024*1024*1024;
  cudaMallocManaged(&data, size);
}
```

32 GB allocation

Pascal supports allocations where only a subset of pages reside on GPU. Pages can be migrated to the GPU when “hot”.

Fails on Kepler/Maxwell

GPU OVERSUBSCRIPTION

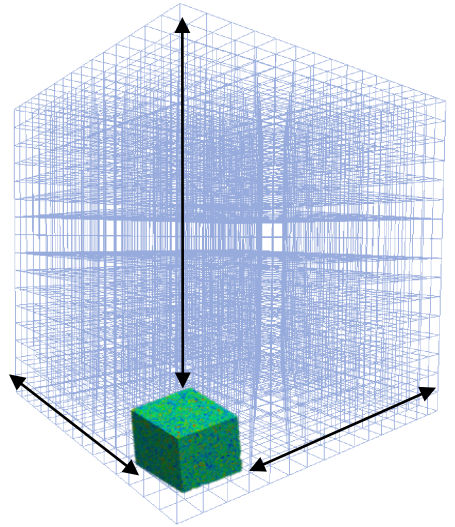
Now possible with Pascal

Many domains would benefit from GPU memory oversubscription:

Combustion – many species to solve for

Quantum chemistry – larger systems

Ray tracing - larger scenes to render



UNIFIED MEMORY ON PASCAL

Concurrent CPU/GPU access to managed memory

```
__global__ void mykernel(char *data) {
  data[1] = 'g';
}

void foo() {
  char *data;
  cudaMallocManaged(&data, 2);

  mykernel<<<...>>>(data);
  // no synchronize here
  data[0] = 'c';

  cudaFree(data);
}
```

OK on Pascal: just a page fault

Concurrent CPU access to 'data' on previous GPUs caused a fatal segmentation fault

UNIFIED MEMORY ON PASCAL

System-Wide Atomics

```
__global__ void mykernel(int *addr) {
  atomicAdd(addr, 10);
}

void foo() {
  int *addr;
  cudaMallocManaged(addr, 4);
  *addr = 0;

  mykernel<<<...>>>(addr);
  __sync_fetch_and_add(addr, 10);
}
```

Pascal enables system-wide atomics

- Direct support of atomics over NVLink
- Software-assisted over PCIe

System-wide atomics not available on Kepler / Maxwell

PERFORMANCE TUNING ON PASCAL

Explicit Memory Hints and Prefetching

Advise runtime on known memory access behaviors with `cudaMemAdvise()`

- `cudaMemAdviseSetReadMostly`: Specify read duplication
- `cudaMemAdviseSetPreferredLocation`: suggest best location
- `cudaMemAdviseSetAccessedBy`: initialize a mapping

Explicit prefetching with `cudaMemPrefetchAsync(ptr, length, destDevice, stream)`

- Unified Memory alternative to cudaMemcpyAsync
- Asynchronous operation that follows CUDA stream semantics

To Learn More:
S6216 “The Future of Unified Memory” by Nikolay Sakharnykh
Tuesday, 4pm