Efficient CPU↔GPU data transfers

CUDA 6.0 Unified Virtual Memory

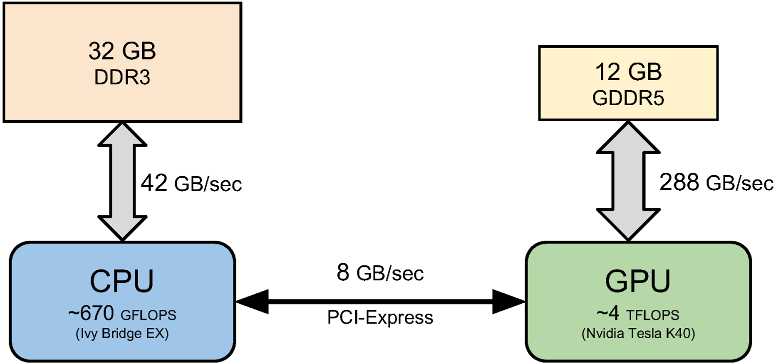
Juraj Kardoš

(University of Lugano)

July 9, 2014

Motivation

- Impact of **data transfers** on overall application performance



When GPU↔CPU memory transfers are performed?

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

Where else?

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

Where else?

- Loading **kernel binary code** (implicitly, by driver)

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

Where else?

- Loading **kernel binary code** (implicitly, by driver)
- Loading **kernel arguments** (transferred into GPU constant memory upon kernel launch, implicitly, by driver)

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

Where else?

- Loading **kernel binary code** (implicitly, by driver)
- Loading **kernel arguments** (transferred into GPU constant memory upon kernel launch, implicitly, by driver)
- Passing return scalar value, e.g. reduction result (remember `__global__` functions are always `void`)

When GPU↔CPU memory transfers are performed?

- When transferring **input/output arrays**

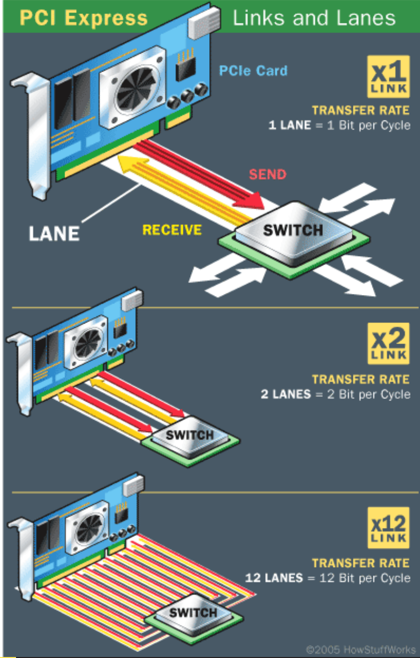
Where else?

- Loading **kernel binary code** (implicitly, by driver)
- Loading **kernel arguments** (transferred into GPU constant memory upon kernel launch, implicitly, by driver)
- Passing return scalar value, e.g. reduction result (remember `__global__` functions are always `void`)
- Initializing `__device__` variables

PCI
EXPRESS®

PCI Express overview

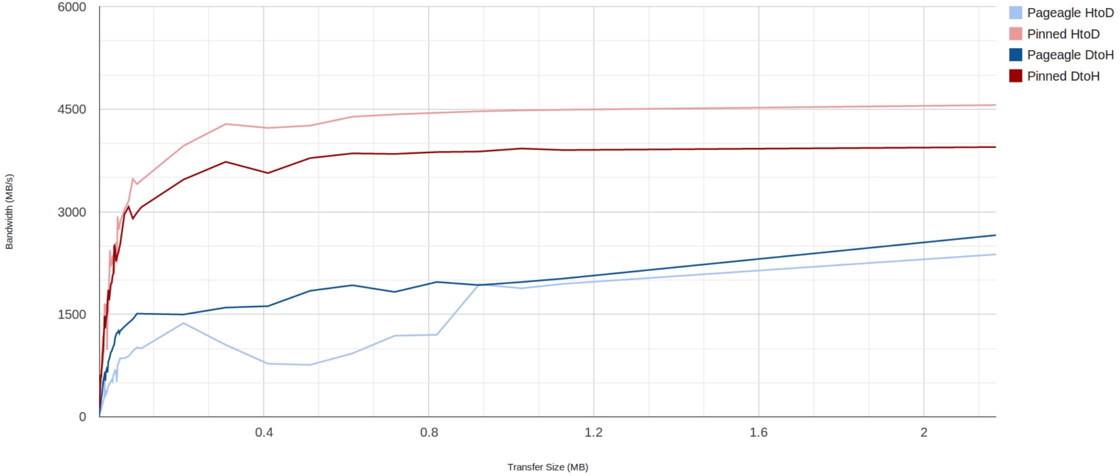
- Computer expansion bus
- Point-to-point connection
- Lane sharing
- Single bus (x1)
  - 500 MB/s per lane (PCI-e v2)
- Multiple lanes (x2, x4, x8, x16, x32)
  - 8 GB/s for a 16 lane bus



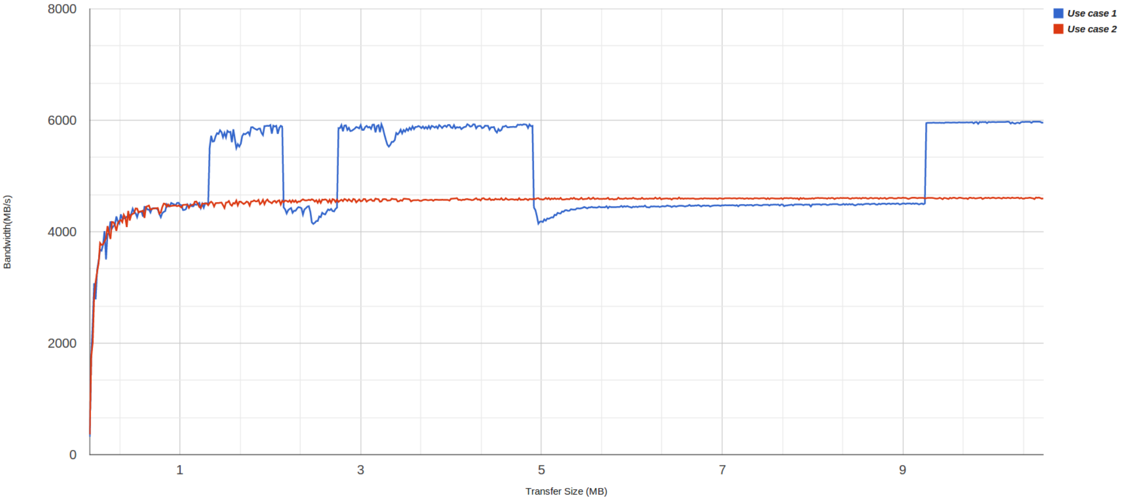
Generations of PCI-Express

PCI Express version	Per Lane Bandwidth	x16 Bandwidth
1.0 (2003)	250 MB/s	4 GB/s
2.0 (2007)	500 MB/s	8 GB/s
3.0 (2010)	984 MB/s	15 GB/s
4.0 (2014-15)	1969 MB/s	31 GB/s

PCI-E Bandwidth Test



Remember PCI-E Lanes?



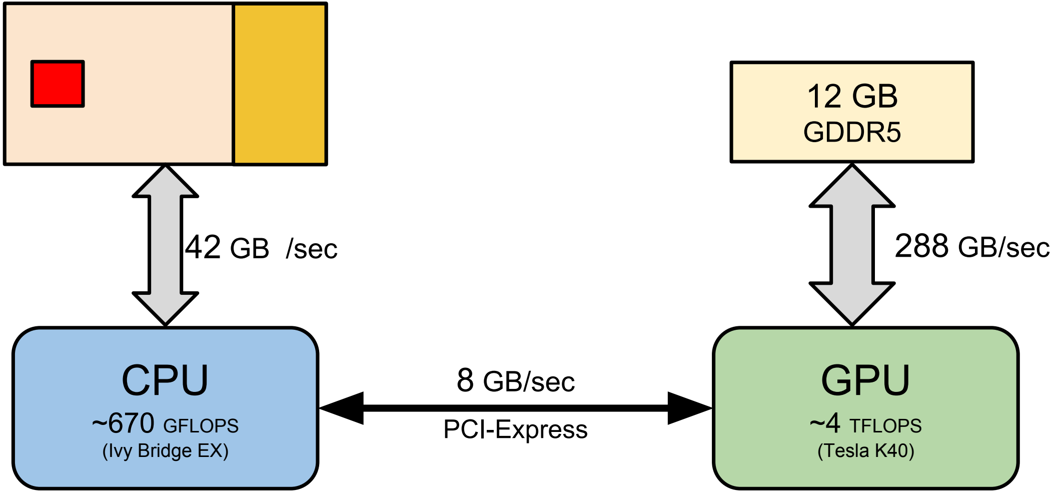
Types of data transfers in CUDA

- Pageable or pinned
- Explicit or implicit (automatic, UVM)
- Synchronous or asynchronous
- Peer to peer (between GPUs of the same host)
- GPUDirect (between GPU and network interface)

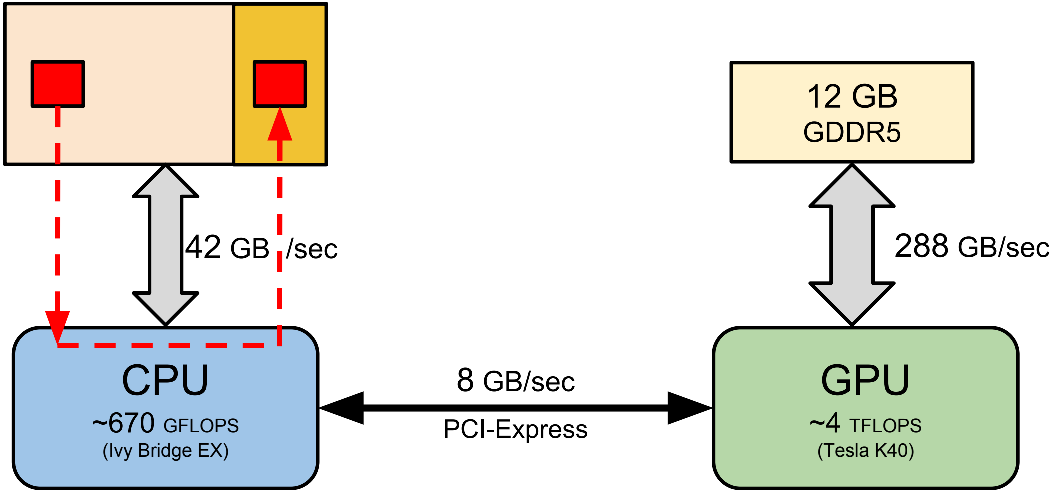
Types of data transfers in CUDA

- **Pageable or pinned**
- Explicit or implicit (automatic, UVM)
- Synchronous or asynchronous
- Peer to peer (between GPUs of the same host)
- GPUDirect (between GPU and network interface)

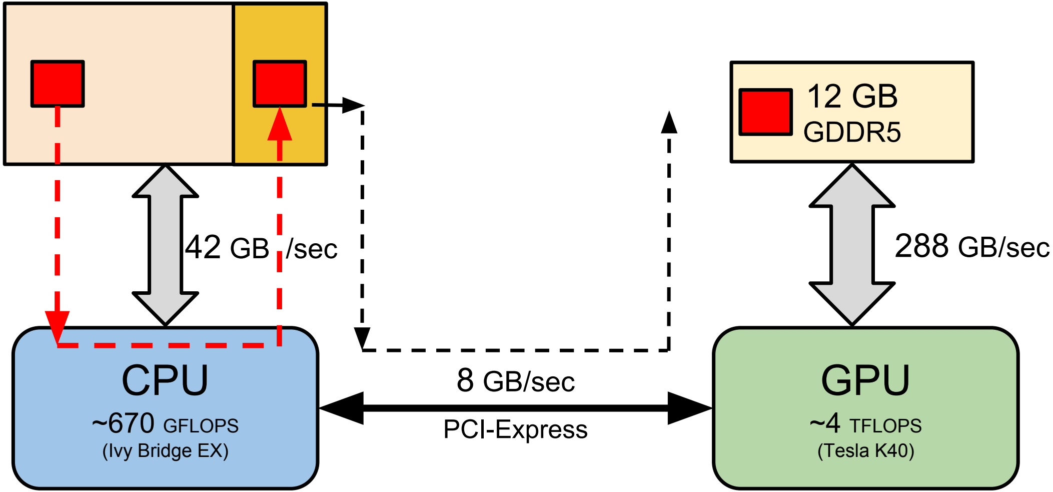
Pageable and pinned memory transfer



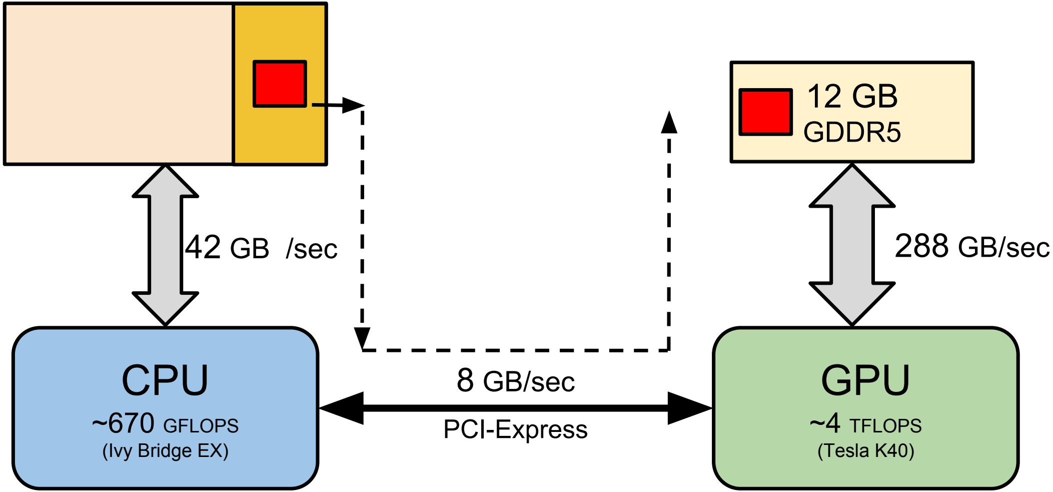
Pageable and pinned memory transfer



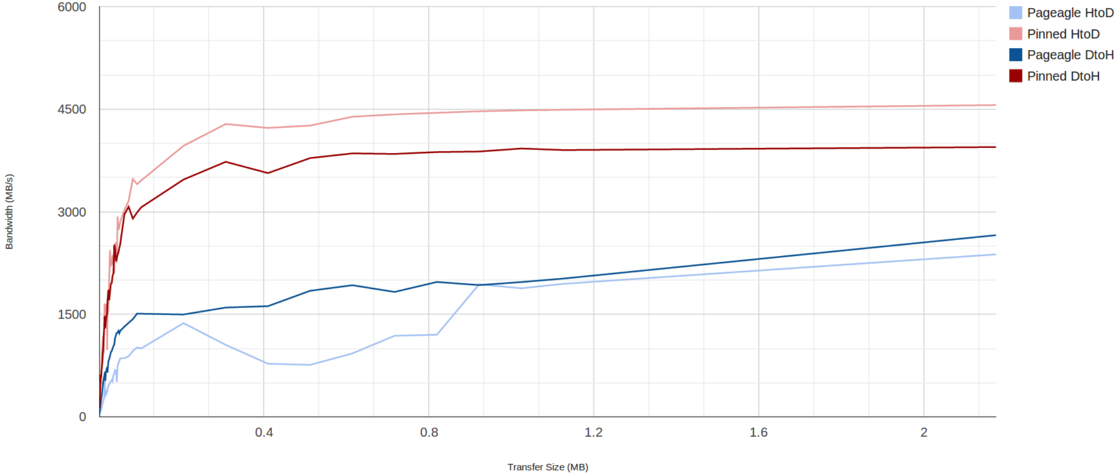
Pageable and pinned memory transfer



Pageable and **pinned** memory transfer



Pageable and pinned memory transfer



Pageable and pinned memory transfer

```
//allocate memory
w0 = (real*)malloc( szarrayb);
cudaMalloc(&w0_dev, szarrayb);

//memcopy
cudaMemcpy(w0_dev, w0, szarrayb, ↩
    cudaMemcpyHostToDevice);

//kernel compute
wave13pt_d<<<...>>>( ..., w0_dev, ...);

//memcopy
cudaMemcpy(w0, w0_dev, szarrayb, ↩
    cudaMemcpyDeviceToHost);
```

Listing 1: Pageable

```
//allocate memory
cudaMallocHost(&w0, szarrayb);
cudaMalloc(&w0_dev, szarrayb);

//memcopy
cudaMemcpy(w0_dev, w0, szarrayb, ↩
    cudaMemcpyHostToDevice);

//kernel compute
wave13pt_d<<<...>>>( ..., w0_dev, ...);

//memcopy
cudaMemcpy(w0, w0_dev, szarrayb, ↩
    cudaMemcpyDeviceToHost);
```

Listing 2: Pinned

Pageable and pinned memory transfer - Summary

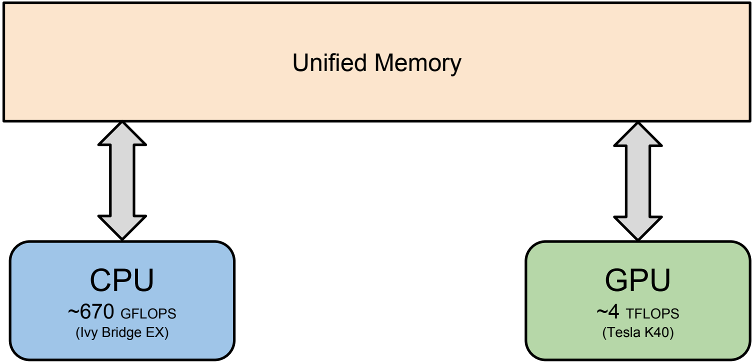
- Pageable memory - user memory space, requires extra mem-copy
- Pinned memory - kernel memory space
- Pinned memory performs better (higher bandwidth)
- Do not over-allocate pinned memory - reduces amount of physical memory available for OS

Types of data transfers in CUDA

- Pageable or pinned
- **Explicit or implicit (UVM)**
- Synchronous or asynchronous
- Peer to peer (between GPUs of the same host)
- GPUDirect (between GPU and network interface)

Unified Memory

- Developer view on memory model
- Still two distinct physical memories on HW level



Unified Memory - Usage

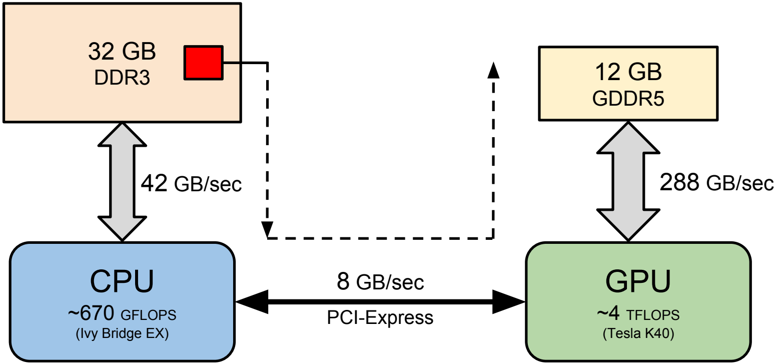
```
//allocate memory
w0 = (real*)malloc( szarrayb);
cudaMalloc(&w0_dev, szarrayb);

//memcopy
cudaMemcpy(w0_dev, w0, szarrayb, ↩
    cudaMemcpyHostToDevice);

//kernel compute
wave13pt_d<<<...>>>( ..., w0_dev, ...);

//memcopy
cudaMemcpy(w0, w0_dev, szarrayb, ↩
    cudaMemcpyDeviceToHost);

//host function
f(w0);
```

Listing 3: Explicit memory

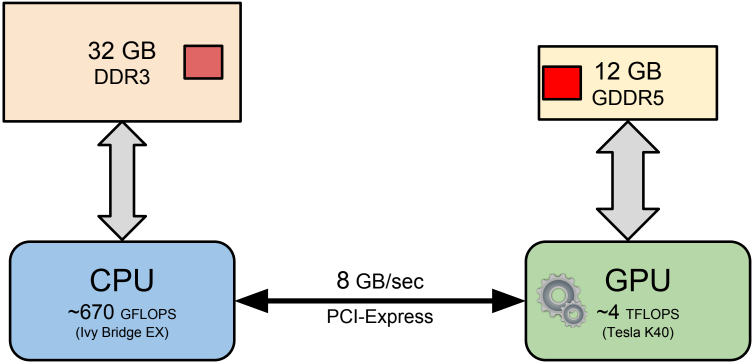
```
//allocate memory
cudaMallocManaged(&w0, szarrayb);

//kernel compute
wave13pt_d<<<...>>>( ..., w0, ...);

//host function
f(w0);
```

Listing 4: UVM

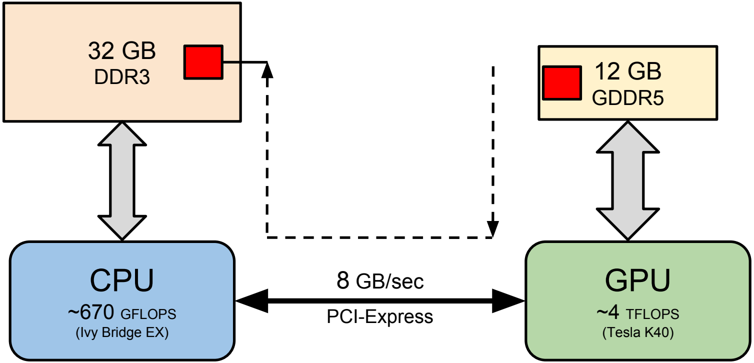
Unified Memory - Use Case



Unified Memory - Use Case

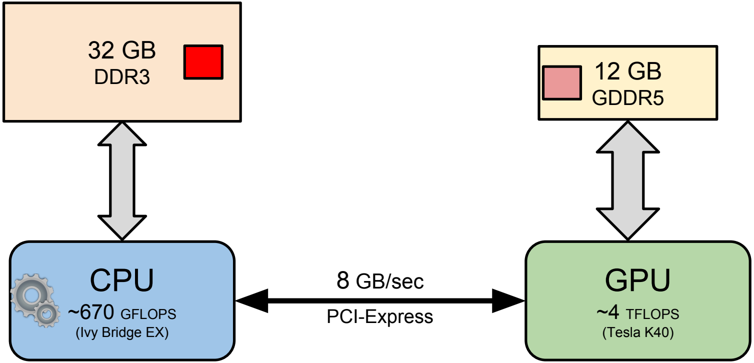


Unified Memory - Use Case

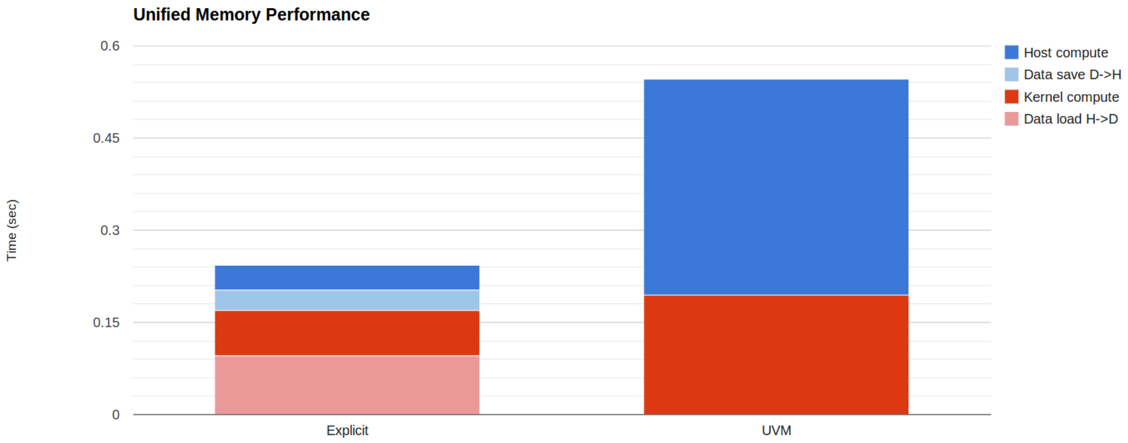


Unified Memory - Use Case

- How does UVM perform when compared to explicit memory movements?



Implicit memory transfers: UVM

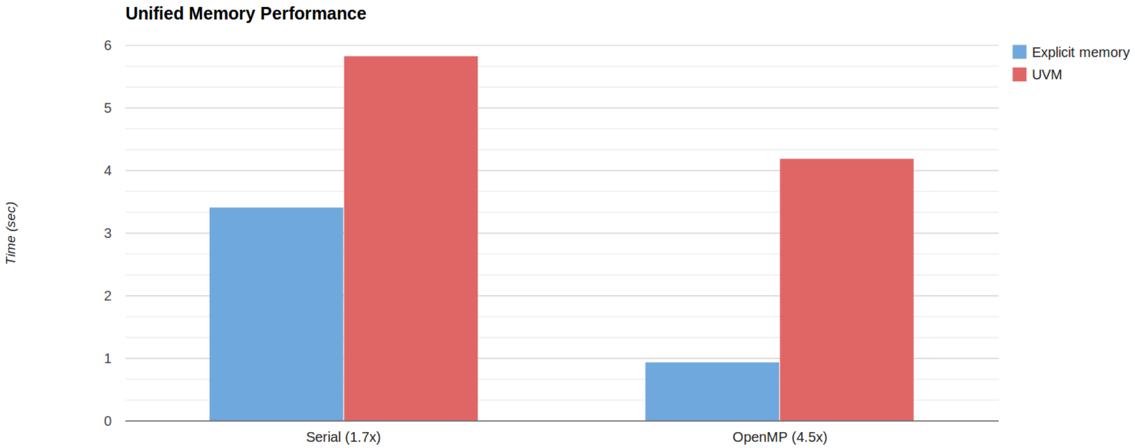


Implicit memory transfers: UVM

- How does UVM perform in case of multi-threading?

Implicit memory transfers: UVM

- UVM Implements CS - threads are serialized, performance degradation



UVM - Summary

- Simplifies programming model, but...

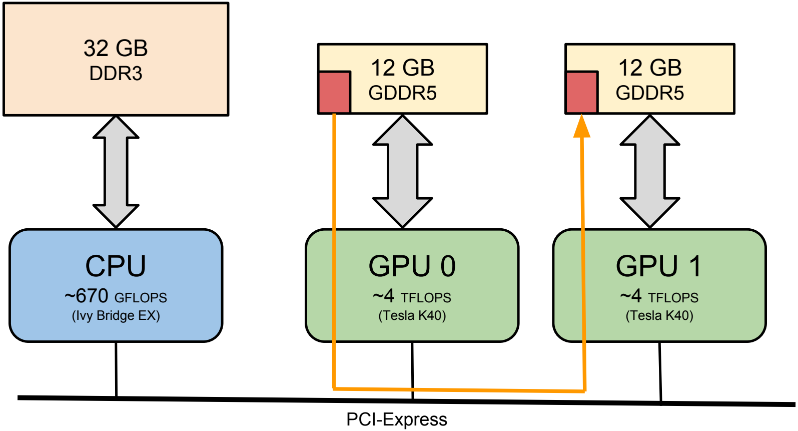
UVM - Summary

- Simplifies programming model, but...
- Performance issue D -> H
- CS in multi-threaded application

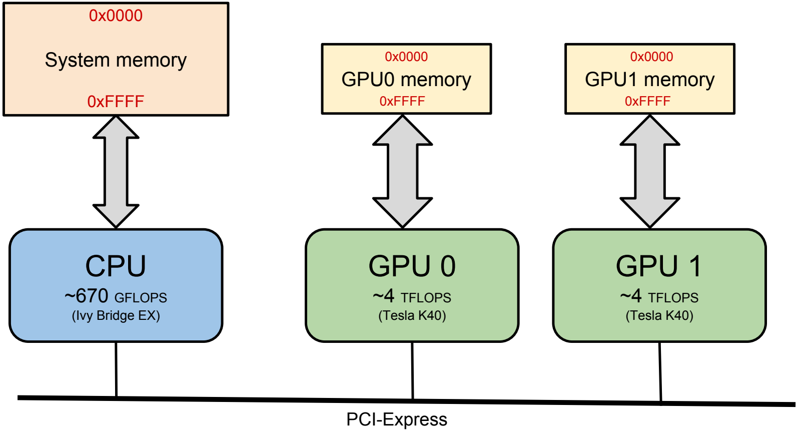
Types of data transfers in CUDA

- Pageable or pinned
- Explicit or implicit (automatic, UVM)
- Synchronous or asynchronous
- **Peer to peer (between GPUs of the same host)**
- GPUDirect (between GPU and network interface)

Peer to peer data transfers overview

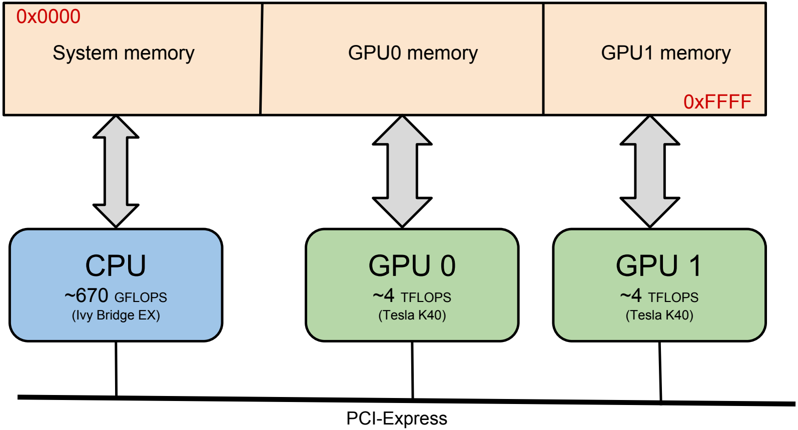


Peer to peer data transfers - Unified Virtual Addressing



Peer to peer data transfers - Unified Virtual Addressing

- UVA maps memories into single address space



P2P Memory Transfer - Usage

```
//allocate memory on gpu0 and gpu1
cudaSetDevice(gpuid_0);
cudaMalloc(&gpu0_buf, buf_size);
cudaSetDevice(gpuid_1);
cudaMalloc(&gpu1_buf, buf_size);

//enable P2P
cudaSetDevice(gpuid_0);
cudaDeviceEnablePeerAccess(gpuid_1, 0);
cudaSetDevice(gpuid_1);
cudaDeviceEnablePeerAccess(gpuid_0, 0);

//P2P copy
cudaMemcpy(gpu0_buf, gpu1_buf, buf_size, cudaMemcpyDefault)
```

Listing 5: P2P

Peer to peer data transfers - Summary

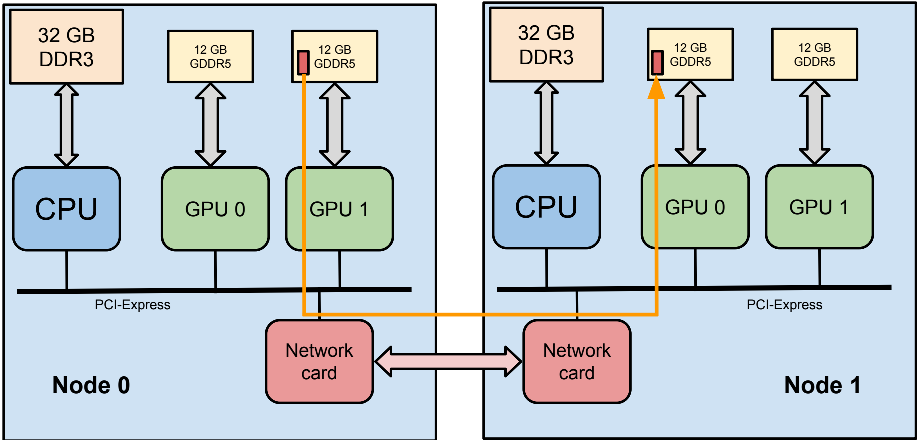
- P2P and UVA can be used to both **simplify** and **accelerate** CUDA programs
- One address space for all CPU and GPU memory
  - Determine physical memory location from pointer value
  - Simplified library interface – `cudaMemcopy()`
- Faster memory copies between GPUs with less host overhead

Types of data transfers in CUDA

- Pageable or pinned
- Explicit or implicit (automatic, UVM)
- Synchronous or asynchronous
- Peer to peer (between GPUs of the same host)
- **GPUDirect (between GPU and network interface)**

GPU direct overview

- Eliminate CPU bandwidth and latency bottlenecks using remote direct memory access transfers between GPUs and other PCIe devices



General recommendations

- PCI-E is efficient only starting from reasonably large data buffer
- UVM simplifies programming model but may result in worse performance
- It's always a good idea to know when underlying runtime routes data though intermediate buffer (additional copying) and avoid that (pinned memory, GPUDirect)
- It's always a good idea to compute something, while data is being transferred (asynchronous)

Control questions

1. How many PCI-E lanes 1 GPU can consume? Suppose you have 40 PCI-E lanes and 4 GPUs. How many lanes there will be available per GPU, if they all are transferring data simultaneously?
2. Given that UVM is slower than explicit copying, what it could still be good for?
3. What is better to use for multi-gpu application: P2P memory transfers, GPUDirect or CUDA-aware MPI?

Control questions: answers

1. 1 GPU usually can use up to 16× lanes. With 4 GPUs in a single system, there will be 8× lanes link per GPU in average, i.e. 2 times less than with single GPU in system. Note this when building your GPU servers.
2. UVM simplifies GPU porting, allowing you omit explicit memory copies during intensive GPU kernels code development.
3. CUDA-aware MPI uses P2P and GPUDirect as underlying engines. Thus, CUDA-aware MPI might better suite MPI applications, while single-node programs could be written in simpler way with CUDA P2P.