MMUs for GPGPUs

Supporting x86-64 Address Translation for 100s of GPU Lanes

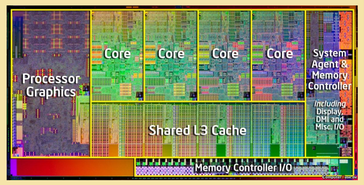
Jason Power, Mark Hill, David Wood

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Affiliates Meeting 2013

Motivation

- CPUs and GPUs are physically integrated
- Still logically separate
  - Explicit memory copies
  - Difficult to share data-structures (linked lists, trees, etc.)



- Want cache coherence and ***shared virtual address space***

```
void main() {
   int *d_in, *d_out;
   int *h_in, *h_out;

   // allocate input array on host
   h_in = new int[1024];
   h_in = ... // Initial host array

   // allocate output array on host
   h_out = new int[1024];

   // Allocate input/output array on device
   d_in = cudaMalloc(sizeof(int)*1024);
   d_out = cudaMalloc(sizeof(int)*1024);

   // copy input array from host to device
   cudaMemcpy(d_in, h_in, sizeof(int)*4, HtD);

   vectoryCopy<<<1,1024>>>(d_in, d_out);

   // copy the output array from device
   cudaMemcpy(h_out, d_out, sizeof(int)

   // continue host computation with re
   ... h_out

   // Free memory on device
   cudaFree(d_in); cudaFree(d_out);
   // Free memory on host
   delete[] h_in; delete[] h_out;
}
```

Separate virtual address space

Shared virtual address space

```
void main() {
   int *h_in, *h_out;

   // allocate input array on host
   h_in = new int[1024];
   h_in = ... // Initial host array

   // allocate output array on host
   h_out = new int[1024];

   vectoryCopy<<<1,1024>>>(h_in, h_out);

   // continue host computation with result
   ... h_out

   // Free memory on host
   delete[] h_in; delete[] h_out;
}
```

- GPU now a drop-in replacement

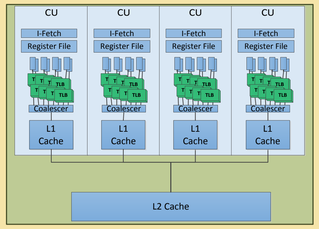
Problem

- To implement a shared virtual address space: Need ***a memory management unit (MMU)***
  - Low overhead
  - Compatible with CPU (x86-64)
  - Support for 4KB pages
  - Page faults, TLB flushes, TLB shootdown, etc.

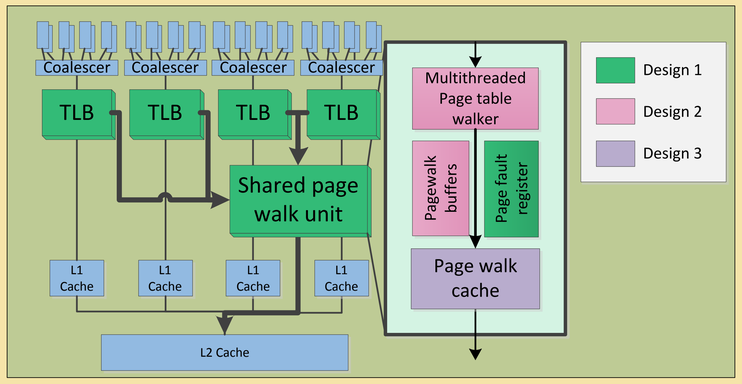
Data-driven Design

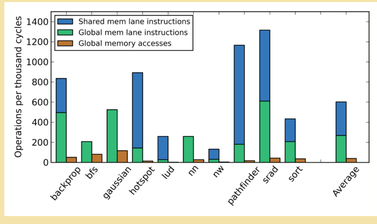
Design 0: Naïve CPU-like

- Place TLB at each lane (CUDA core)
- High complexity
- High power

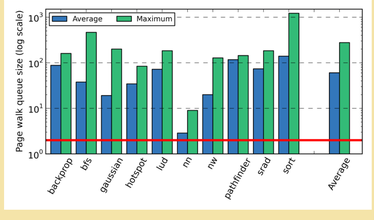


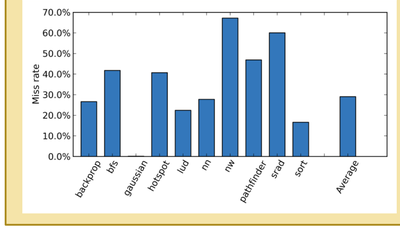
Proof of concept design





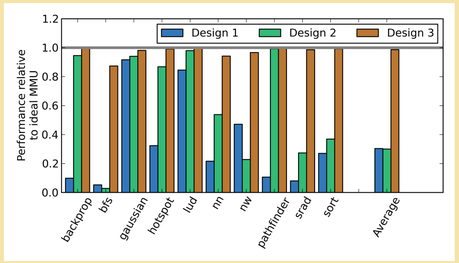
- Coalescer/shared mem effectively filter
- **D1:** *Move TLB behind coalescer*
- Need to satisfy high TLB miss bandwidth
- **D2:** *+Highly-threaded page table walker*



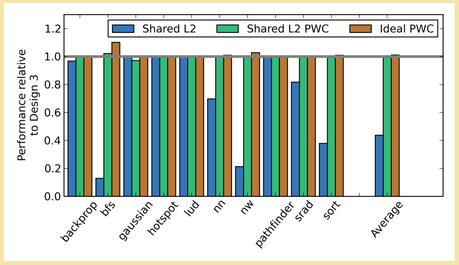


- Need to reduce TLB miss latency
- **D3:** *+Page walk cache*

Results



- Ideal MMU:
  - Infinite caches
  - Minimal miss latency
- D3 within 1% of ideal



- Shared L2 & PWC
  - Can reduce size of L1 TLB
- Ideal PWC
  - Average 1%
  - Max 10%

	Per-CU L1 TLB entries	Highly-threaded page table walker	Page walk cache size	Shared L2 TLB entries
Ideal MMU	Infinite	Infinite	Infinite	None
Design 0	N/A: Per-lane MMUs		None	None
Design 1	128	Per-CU walkers	None	None
Design 2	128	Yes (32-way)	None	None
Design 3	64	Yes (32-way)	8 KB	None
Shared L2	64	Yes (32-way)	None	1024
Shared L2 & PWC	32	Yes (32-way)	8 KB	512
Ideal PWC	64	Yes (32-way)	Infinite	None

Conclusions

- Non-exotic GPU MMU design
  - L1 TLB behind coalescer and shared memory
  - Shared highly-threaded page table walker
  - Shared page walk cache
- Full compatibility with x86-64
- Simplifies heterogeneous programming opening the doors to novel applications

MMUs for GPGPUs

Supporting x86-64 Address Translation for 100s of GPU Lanes

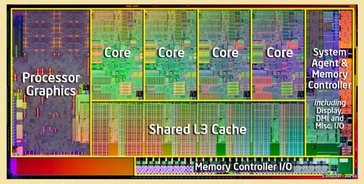
Jason Power, Mark Hill, David Wood

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Affiliates Meeting 2013

Motivation

- CPUs and GPUs are physically integrated
- Still logically separate
  - Explicit memory copies
  - Difficult to share data-structures (linked lists, trees, etc.)



- Want cache coherence and ***shared virtual address space***

```
void main() {
   int *d_in, *d_out;
   int *h_in, *h_out;

   // allocate input array on host
   h_in = new int[1024];
   h_in = ... // Initial host array

   // allocate output array on host
   h_out = new int[1024];

   // Allocate input/output array on device
   d_in = cudaMalloc(sizeof(int)*1024);
   d_out = cudaMalloc(sizeof(int)*1024);

   // copy input array from host to device
   cudaMemcpy(d_in, h_in, sizeof(int)*4, HtD);

   vectoryCopy<<<1,1024>>>(d_in, d_out);

   // copy the output array from device
   cudaMemcpy(h_out, d_out, sizeof(int)

   // continue host computation with re
   ... h_out

   // Free memory on device
   cudaFree(d_in); cudaFree(d_out);
   // Free memory on host
   delete[] h_in; delete[] h_out;
}
```

Separate virtual address space

Shared virtual address space

```
void main() {
   int *h_in, *h_out;

   // allocate input array on host
   h_in = new int[1024];
   h_in = ... // Initial host array

   // allocate output array on host
   h_out = new int[1024];

   vectoryCopy<<<1,1024>>>(h_in, h_out);

   // continue host computation with result
   ... h_out

   // Free memory on host
   delete[] h_in; delete[] h_out;
}
```

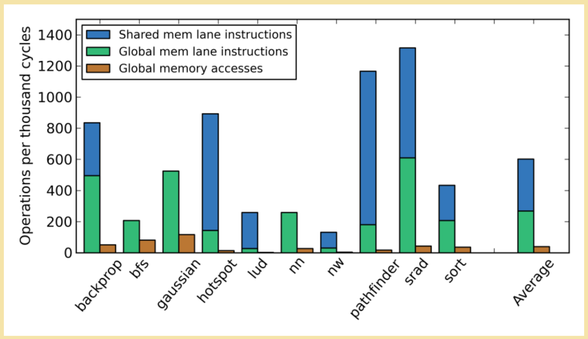
- GPU now a drop-in replacement

Problem

- To implement a shared virtual address space: Need ***a memory management unit (MMU)***
  - Low overhead
  - Compatible with CPU (x86-64)
  - Support for 4KB pages
  - Page faults, TLB flushes, TLB shootdown, etc.

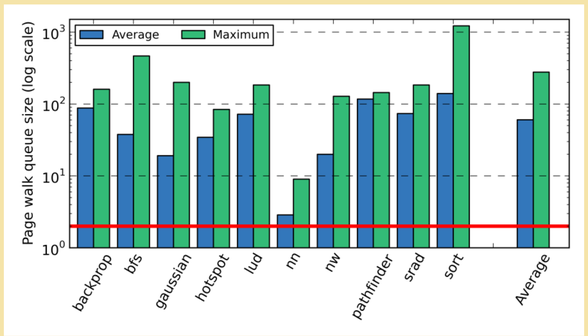
Data-driven design

D1: TLBs after the coalescer



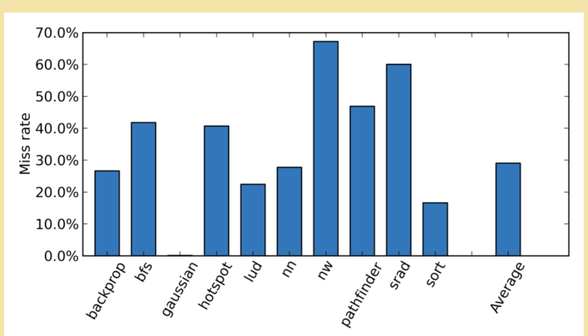
- Coalescer/shared mem effectively filter requests

D2: +Highly-threaded page table walker



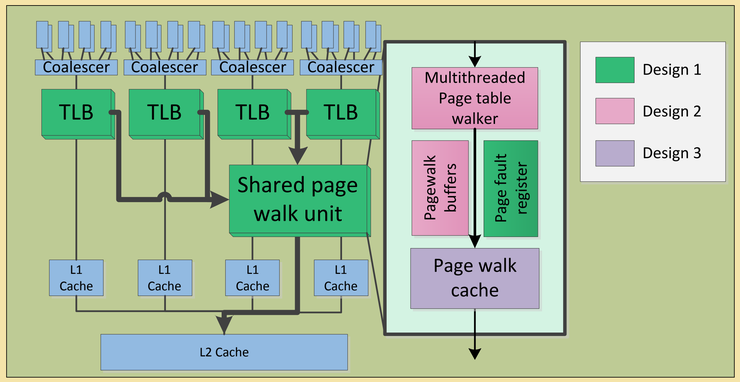
- Need to satisfy high TLB miss bandwidth

D3: +Page walk cache

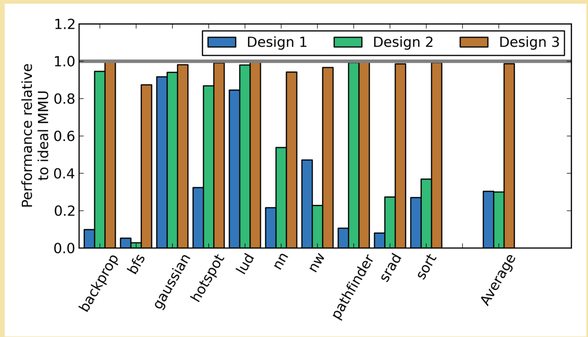


- Need to reduce TLB miss latency

Proof-of-concept MMU



Results



- D3 within 1% of infeasible ideal MMU

	Per-CU L1 TLB entries	Highly-threaded page table walker	Page walk cache size		Per-CU L1 TLB entries	Highly-threaded page table walker	Page walk cache size
Ideal MMU	Infinite	Infinite	Infinite	Design 2	128	Yes (32-way)	None
Design 1	128	Per-CU walkers	None	Design 3	64	Yes (32-way)	8 KB

Conclusions

- Non-exotic GPU MMU design
  - L1 TLB behind coalescer and shared memory
  - Shared highly-threaded page table walker
  - Shared page walk cache
- Full compatibility with x86-64
- Simplifies heterogeneous programming opening the doors to novel applications