gem5-gpu

Extending gem5 for GPGPUs

Jason Power, **Marc Orr**, Joel Hestness, Mark Hill, David Wood

(powerjg/morr)@cs.wisc.edu

gem5-gpu

- gem5 + GPGPU-Sim (v3.0.1)

- Flexible memory system
- Models CPU-GPU interactions
- Full-system simulation

Outline

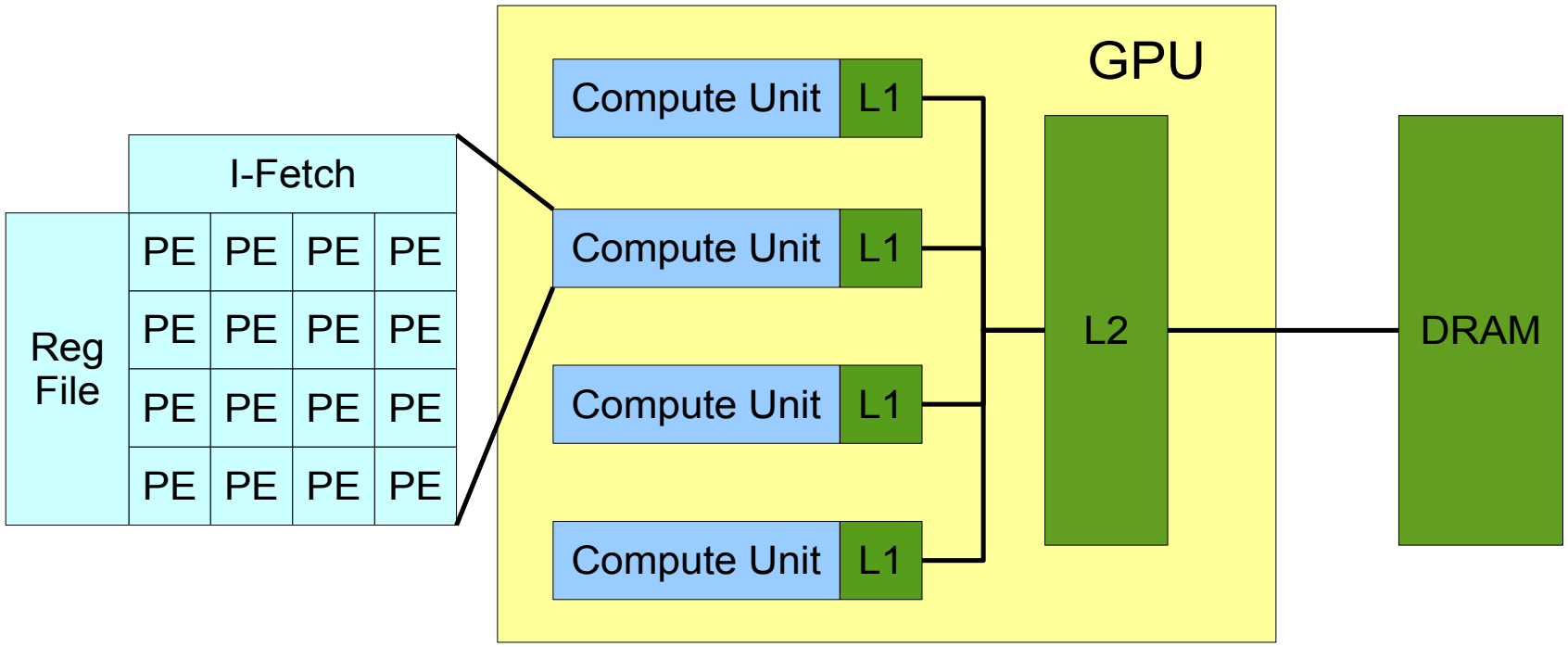
- **Background**
  - Motivation
  - GPGPU architecture
  - GPGPU code
- Implementation
- Brief case study
- Conclusions

Motivation

- More general purpose GPU computing
- Tighter integration

- Current simulators
  - gem5: CPU and memory model
  - GPGPU-Sim: GPU only timing model

GPGPU Architecture



CUDA Example

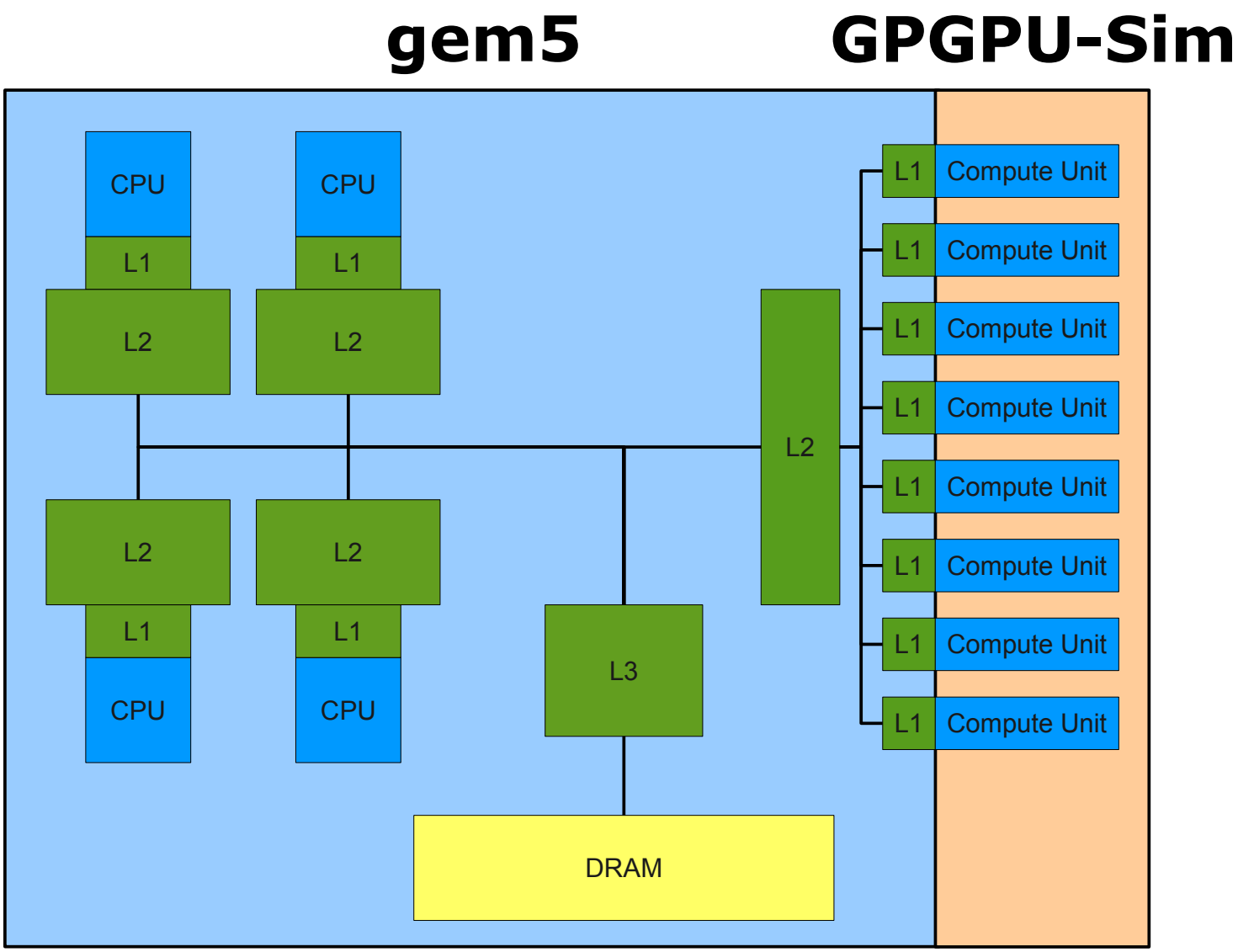
```
int *host_data = malloc();
int *device_data = cudaMalloc();
init_data_CPU(host_data);

//copy host data to device
cudaMemcpy(host_data, device_data);

//launch kernel
kernel(device_data);

//copy device data back to host
cudaMemcpy(device_data, host_data);
process_results_CPU(host_data);
```

```
kernel(int *data) {
 int tid=threadIdx.x;
 data[tid]=tid;
}
```

Outline

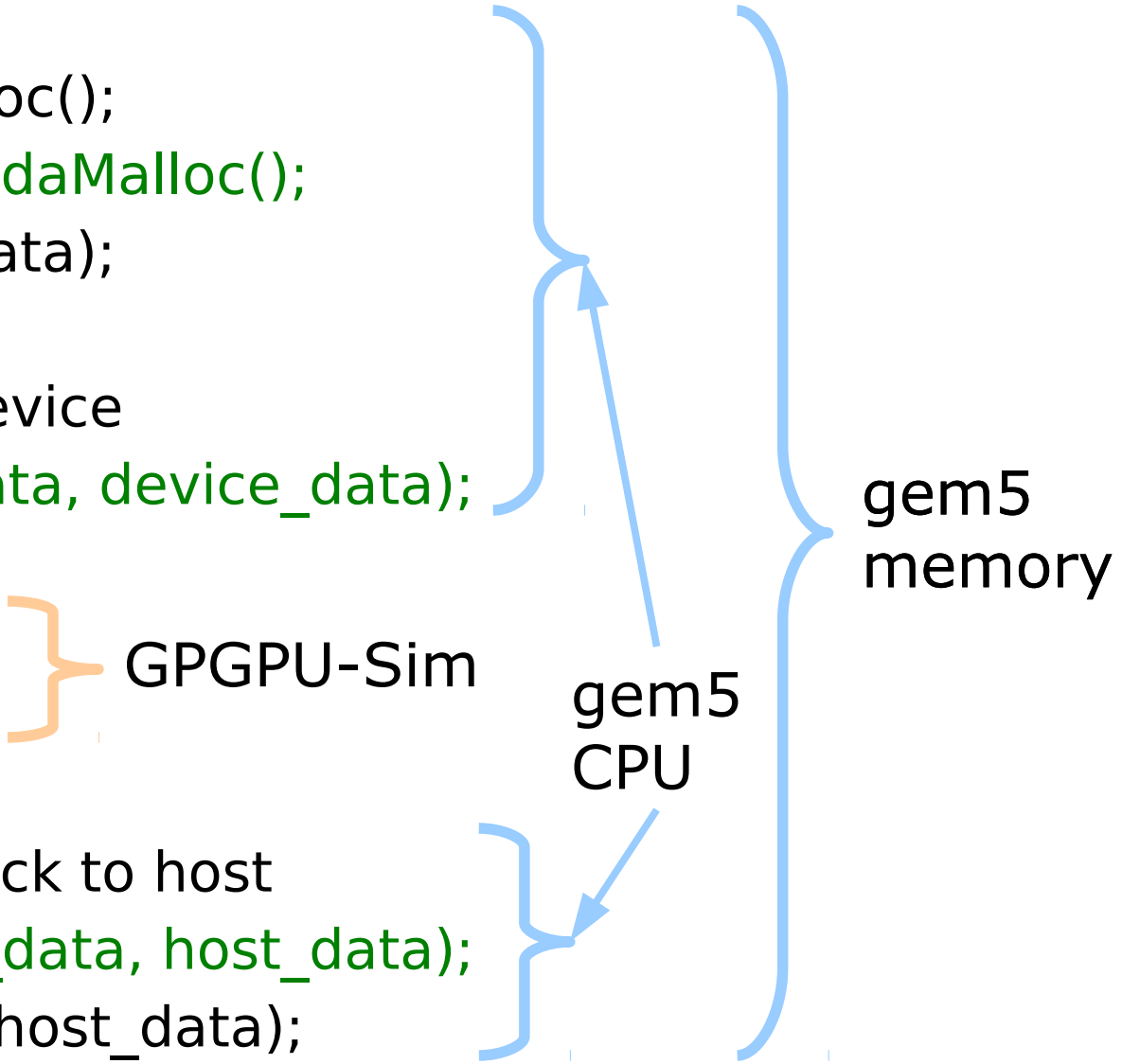
- Background
- **Implementation**
  - System overview
  - Implementation
- Brief case study
- Conclusions

System Overview



CUDA Example

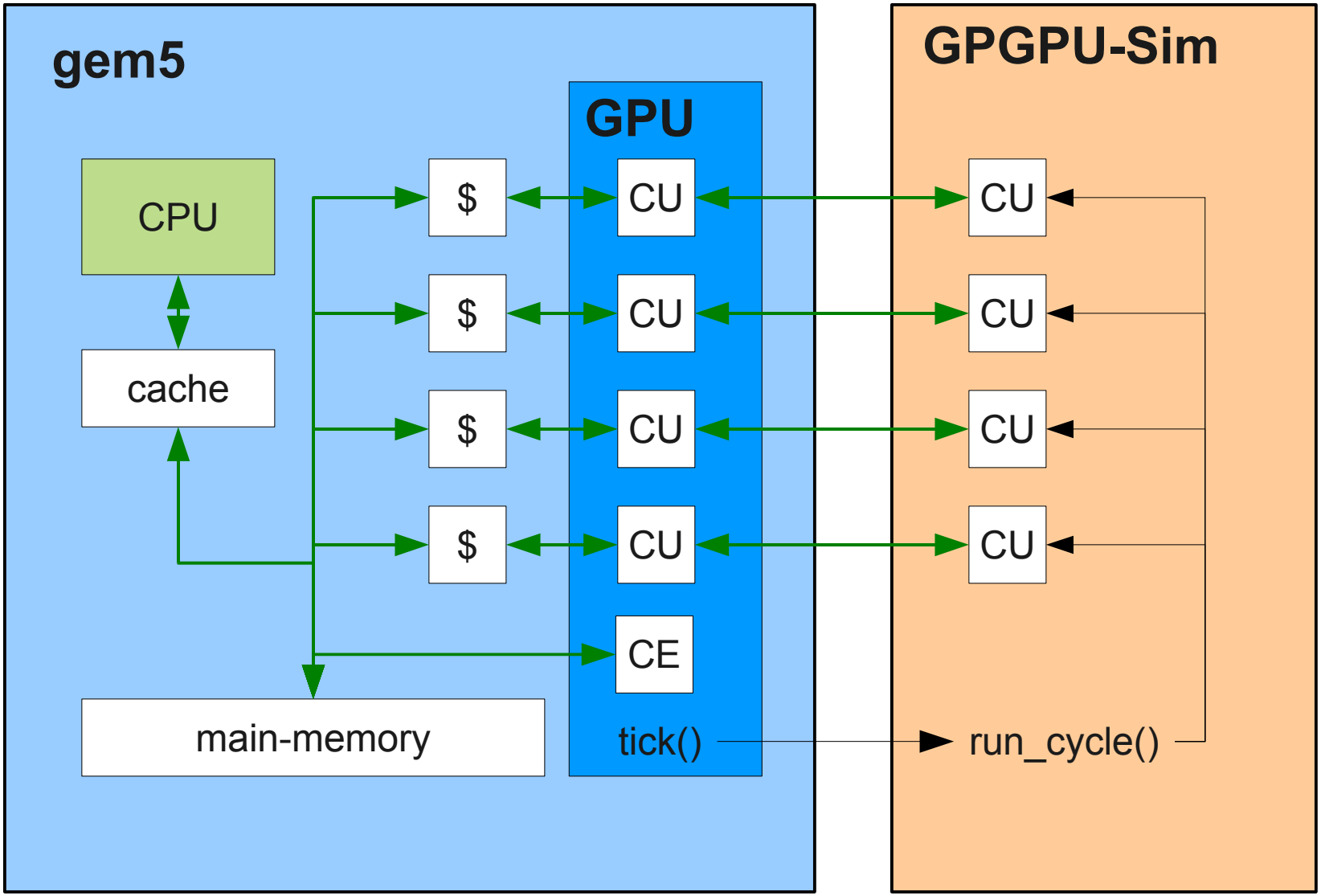
```
int *host_data = malloc();
int *device_data = cudaMalloc();
init_data_CPU(host_data);

//copy host data to device
cudaMemcpy(host_data, device_data);

//launch kernel
kernel(device_data);

//copy device data back to host
cudaMemcpy(device_data, host_data);
process_results_CPU(host_data);
```



Implementation

- Minimally modified GPGPU-Sim
  - A few bug fixes (64-bit addresses, etc)
  - Support event-driven simulation
  - Memory is execute-at-execute
- Wrapper around GPGPU-Sim objects
  - Graphics processing unit
  - Compute unit
- Additional GPU interfaces
  - Coalescer and memory unit
  - Copy Engine
  - GPU TLB

Integration



Flexible Memory System

- Complex cache topologies
  - All Ruby protocols work out of the box
  - Fully-coherent, heterogeneous caches
  - Partially incoherent caches
  - Split physical memory
- Detailed memory unit and coalescer
  - Decoupled from core and cache models
  - Very high bandwidth
  - Pluggable coalescer

Outline

- Background
- Implementation
- **Brief case study**
  - Unified memory example
  - Results
- Conclusions

Conventional Memory

```
int *host_data = malloc();
int *device_data = cudaMalloc();
init_data_CPU(host_data);

//copy host data to device
cudaMemcpy(host_data, device_data);

//launch kernel
kernel(device_data);

//copy device data back to host
cudaMemcpy(device_data, host_data);
process_results_CPU(host_data);
```

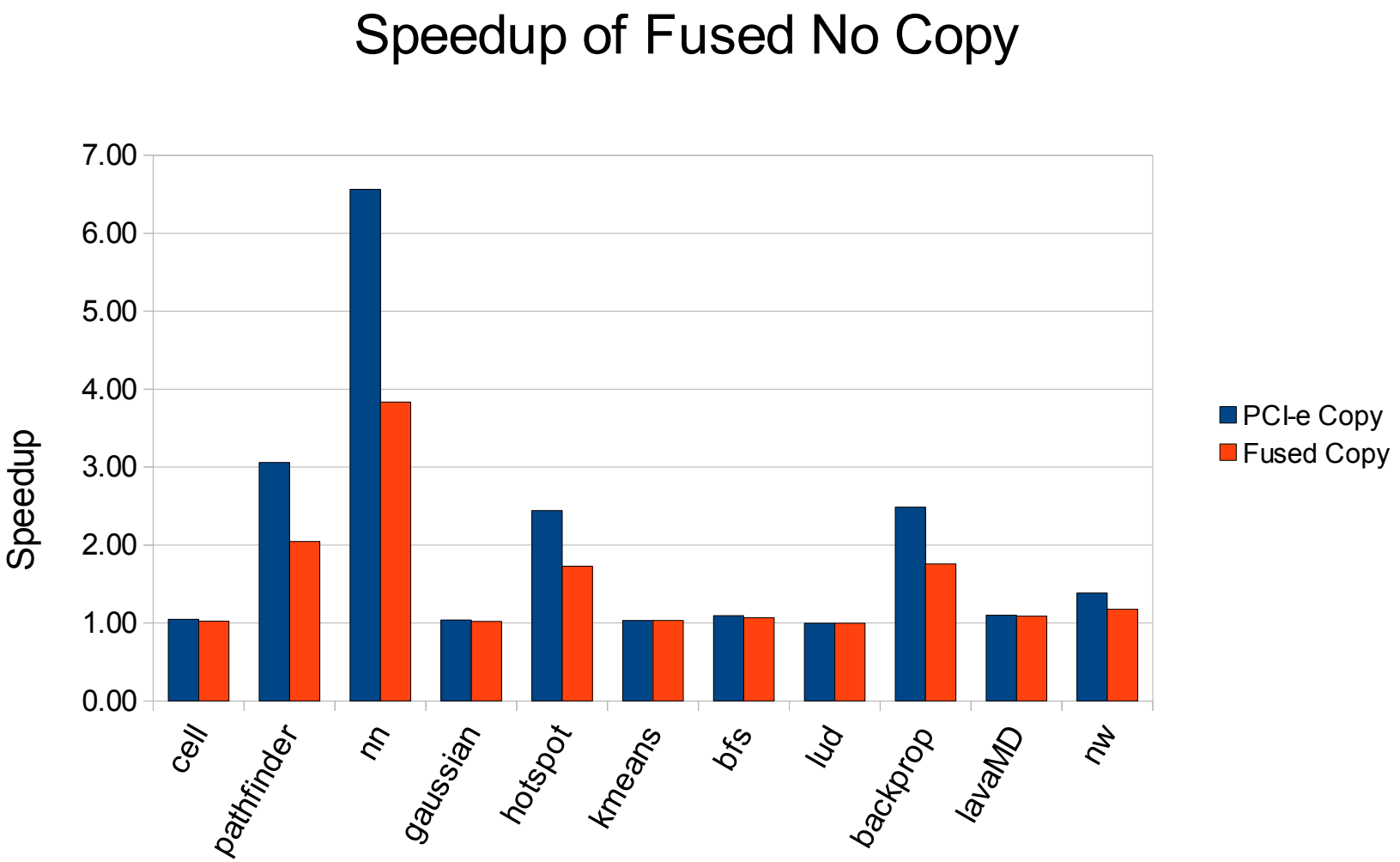
No Copy Memory

```
int *host_data = malloc();
int *device_data = cudaMalloc();
init_data_CPU(host_data);

//copy host data to device
cudaMemcpy(host_data, device_data);

//launch kernel
kernel(host_data);

//copy device data back to host
cudaMemcpy(device_data, host_data);
process_results_CPU(host_data);
```

System Configuration

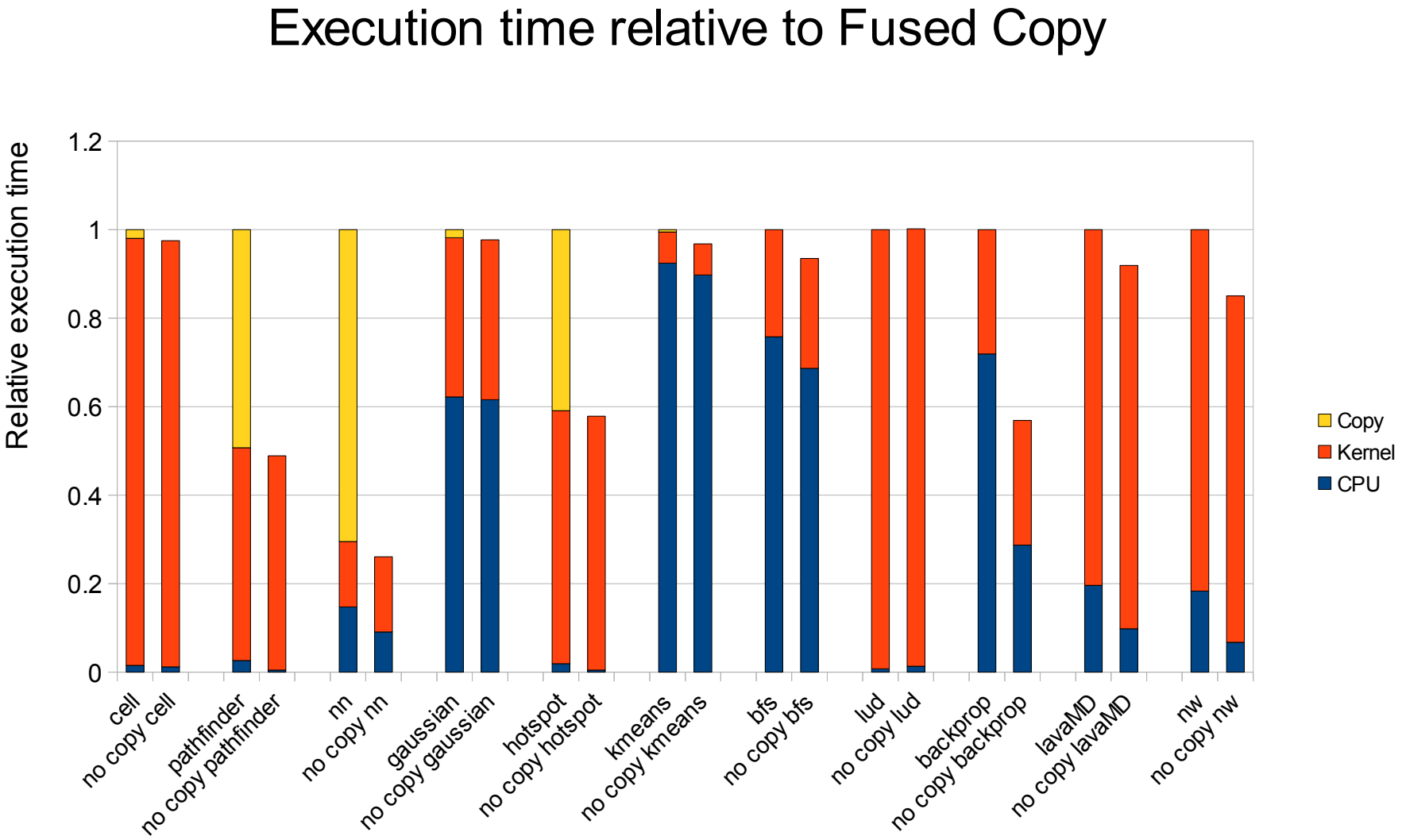
- CPU
  - 1 core (Simple Timing), 2 GHz
  - L1 data: 64 kB/2-way
  - L1 instruction: 32 kB/2-way
  - L2: 2MB/8-way/64 byte lines
- Memory
  - 2 GB @ 90 GB/s
- GPU
  - 16 CUs @ 1.4 GHz
  - L1 64 kB/64-way/128 byte lines
  - L2 1MB/512-way/128 byte lines
  - 48 kB of Shared Memory, 30 cycle latency
  - 32768 registers/CU (16 banks)

Case Study: Eliminating Copies



PCI-e Copy BW: 1.5 GB/s
Fused Copy BW: 3 GB/s

Fused execution time



Fused Copy BW: 3 GB/s

Outline

- Background
- Implementation
- Brief case study
- **Conclusions**

Features (Today)

- Open Source GPGPU Core model
- Runs unmodified source
  - Supports most of CUDA language
- Many cache configurations
  - Incoherent (NVIDIA Fermi-like)
  - Fully coherent
  - Split physical memory
- Support for full-system mode
- Rodinia benchmarks (copy and no copy)
- Checkpointing

Experiences / Take-Aways

- Challenges
  - bandwidth configuration
  - asymmetric cache hierarchies
  - hard-coded cache line sizes

- Should there be a standard interface for accelerators in gem5?

Backup

Features (Tomorrow)

- Accurate timing for address translations
- OpenCL
- More flexible checkpointing
- Deeper integration into GPU pipeline