Supporting Dynamic GPU Computing Result Reuse in the Cloud

Husheng Zhou, Yangchun Fu and Cong Liu
{husheng.zhou, yangchun.fu, cong}@utdallas.edu

The University of Texas at Dallas
Department of Computer Science

Graphics Processing Units

2008



250 cores
900 Gflops
(Tesla)

2010



500 cores
1345 Gflops
(Fermi)

2012



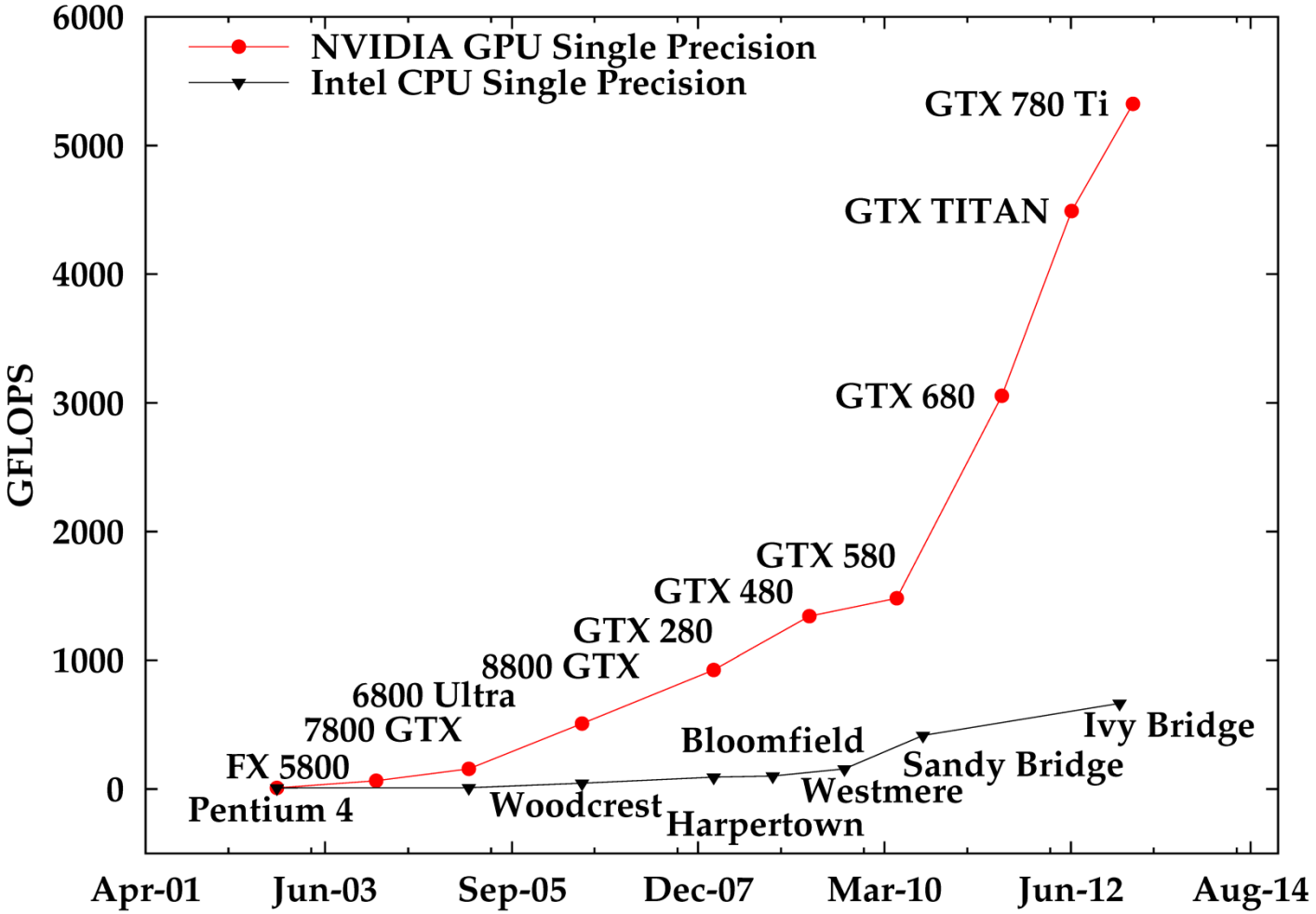
1500 cores
3200 Gflops
(Kepler)

2015



2048 cores
4612 Gflops
(Maxwell)

GPU vs CPU Peak Performance



General Purpose Computing on GPU in t he cloud



Cloud Gaming



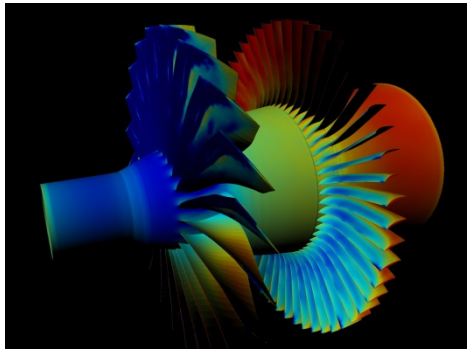
Big Data



Computer Vision



Supercomputing

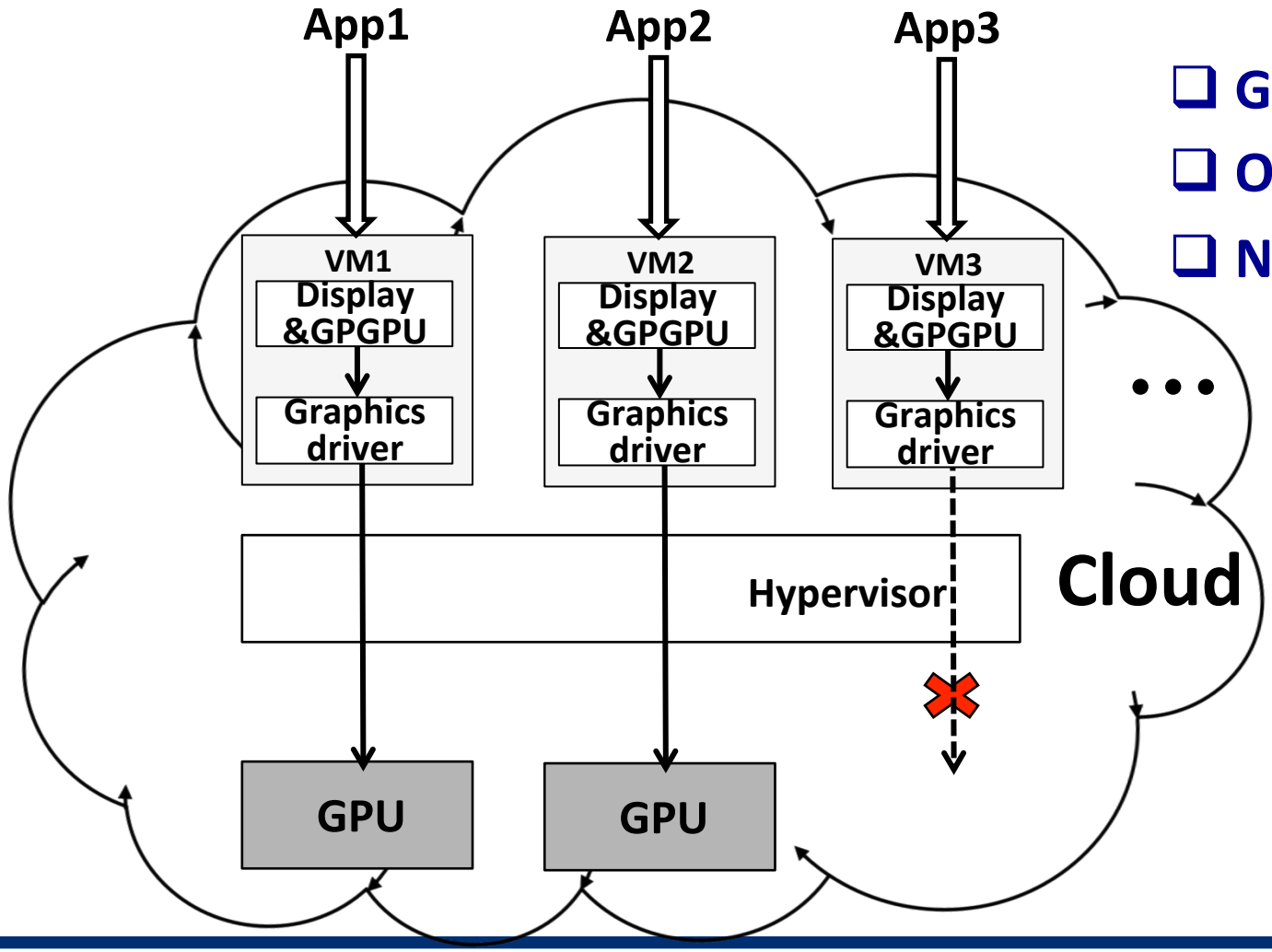


Scientific Simulation



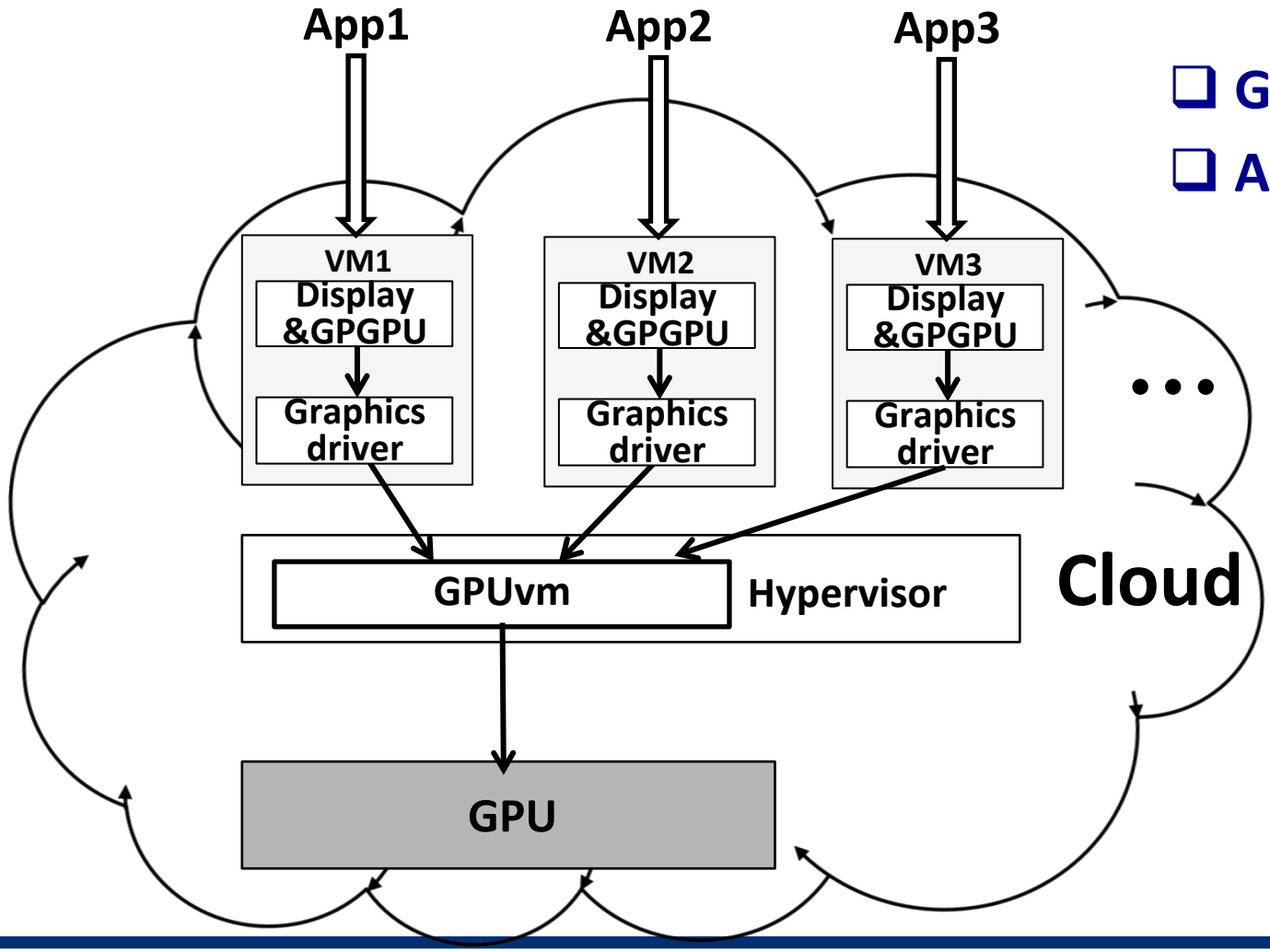
Economic Modeling

GPU in the cloud



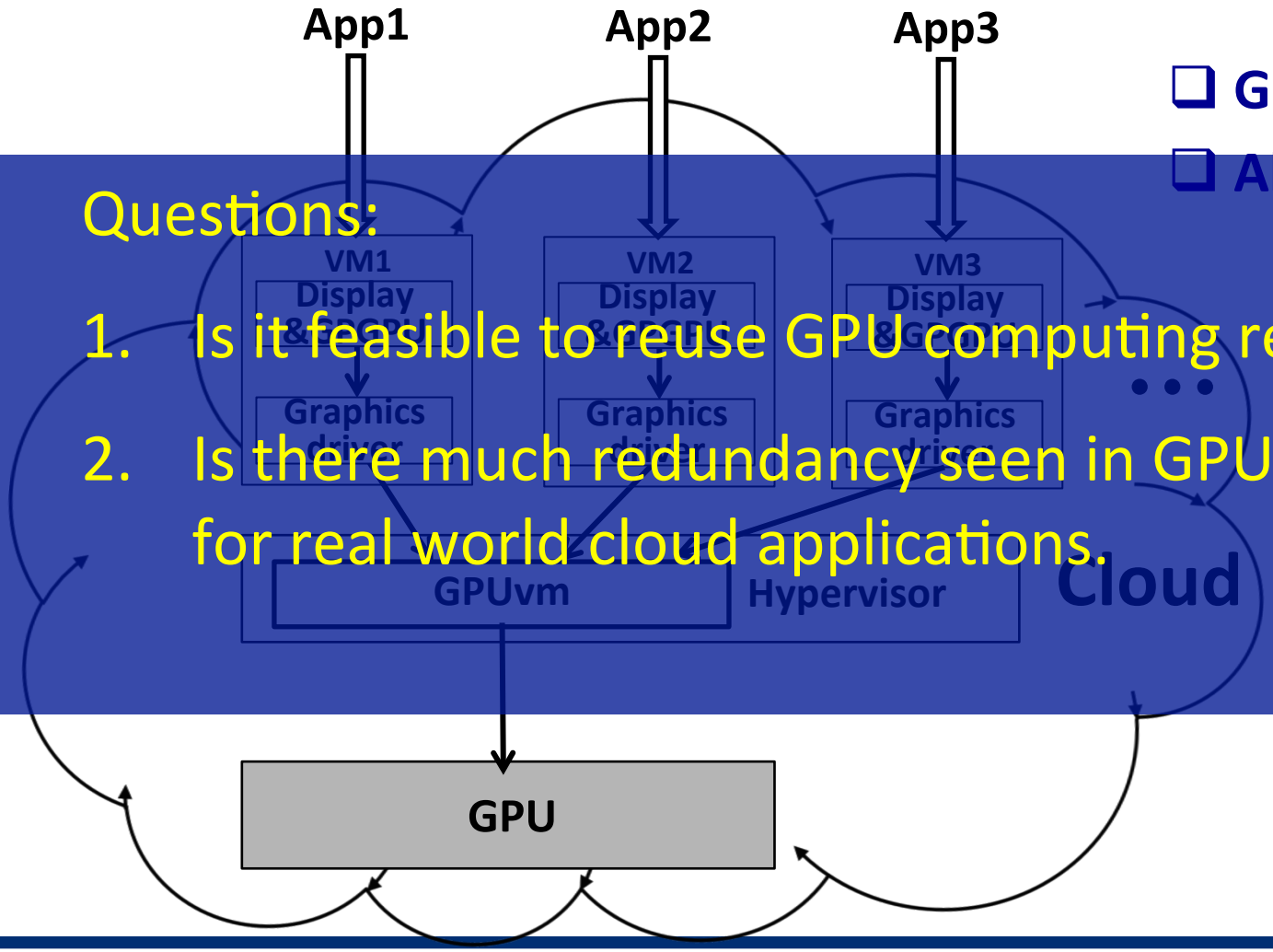
- GPU passthrough
- One GPU to a VM
- No Multiplexing

GPU in the cloud



- GPU virtualization
- Allow GPU sharing

GPU in the cloud



- GPU virtualization
- Allow GPU sharing

Questions:

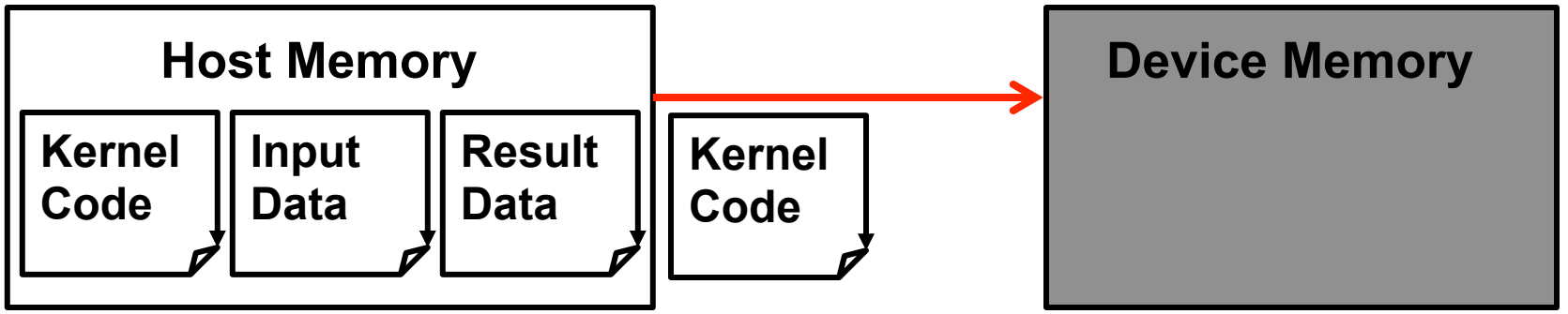
1. Is it feasible to reuse GPU computing results?
2. Is there much redundancy seen in GPU computations for real world cloud applications.

GPU Computing



CMD_HtoD



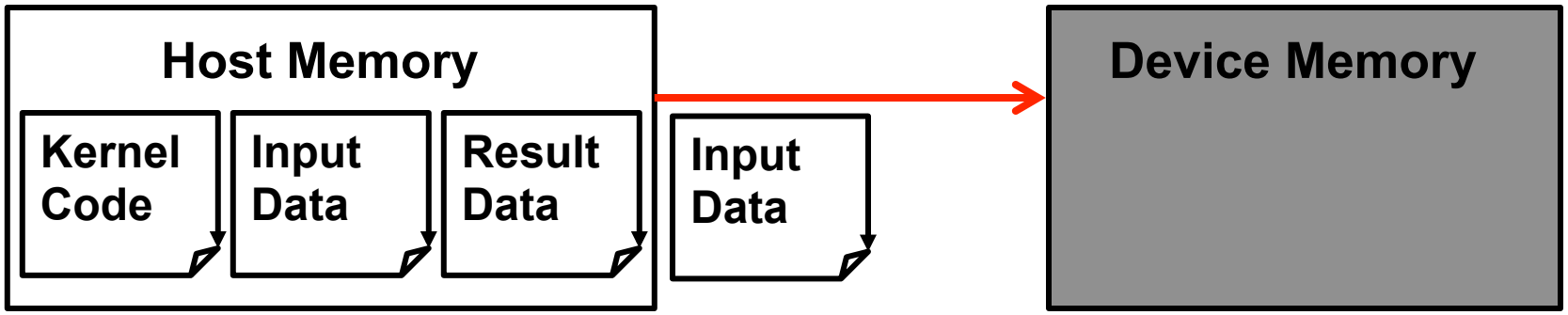


GPU Computing



CMD_HtoD



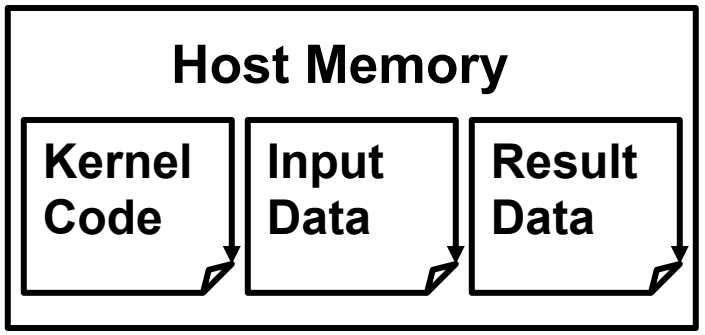


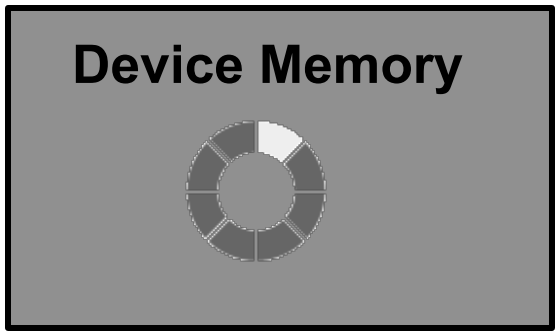
GPU Computing



CMD_Launch





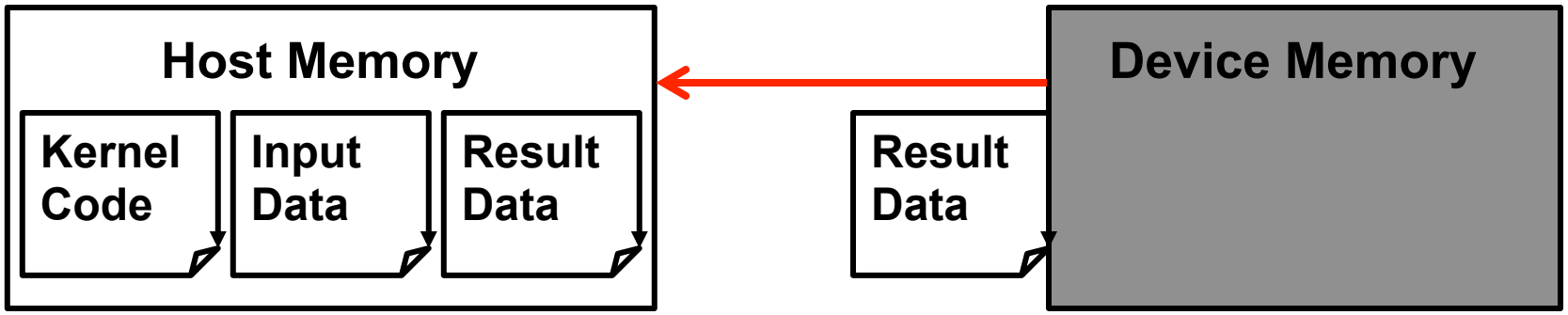


GPU Computing

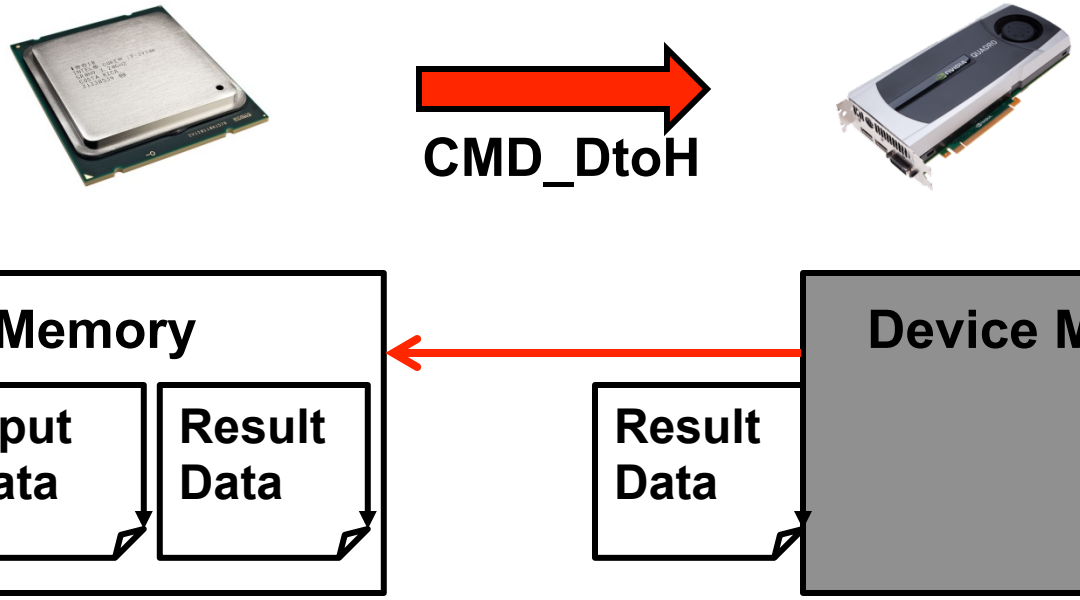






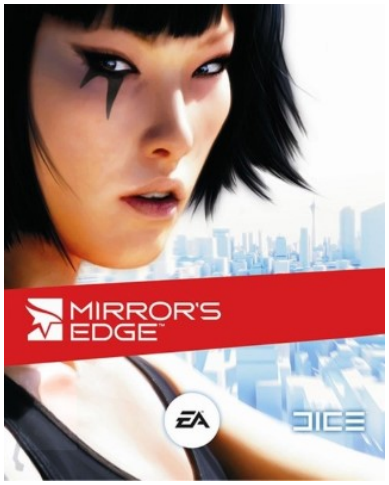


GPU Computing



The "blackbox" feature of GPU processing makes GPU a natural choice for implementing the result reuse idea.

Case Study





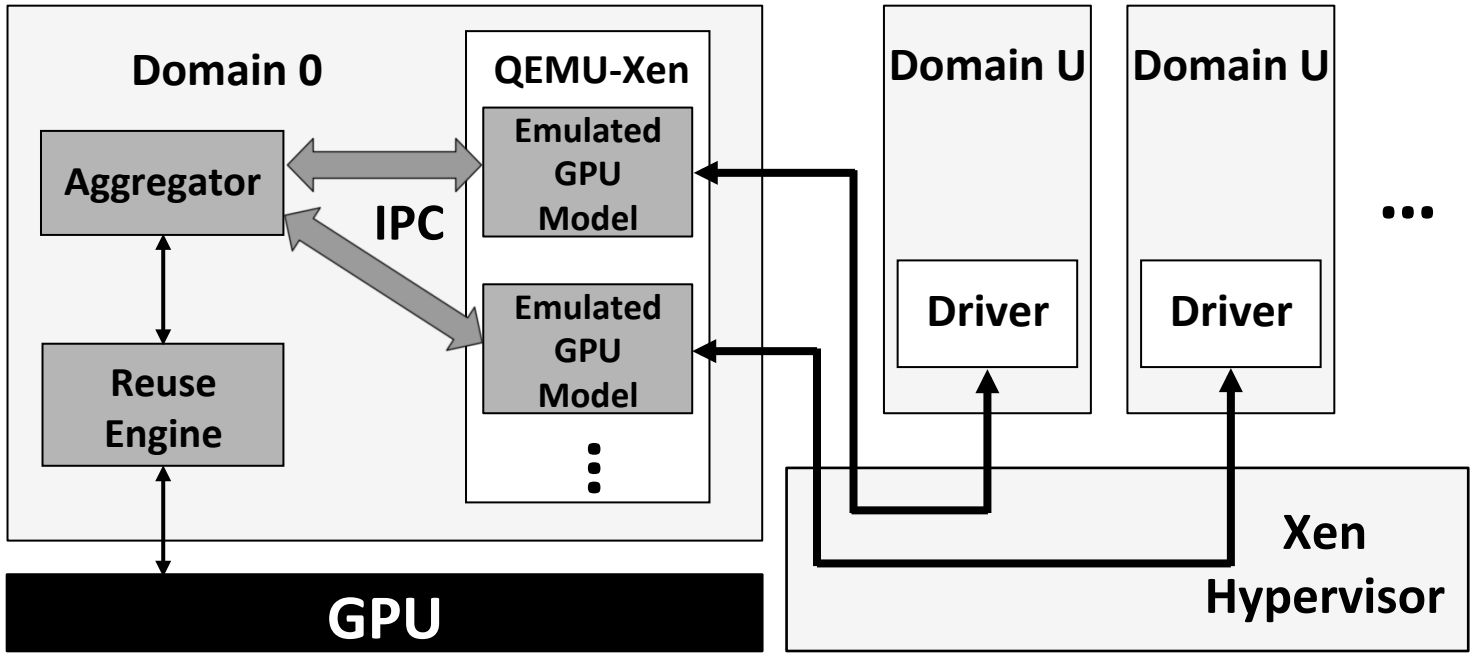
- Two runs, each for 10 minutes
- Collect GPU computing trace

	#Launch	Uniq	Exe	Redundant	Identica
R1 (10 min)	173,766	24	22.31s	24.71%	47.12%
R2 (10 min)	123,175	19	17.10s	25.63%	66.47%

Objective

- **Goal:**
  - Transparently allow GPU **result memoization and reuse in the cloud**
- **Approach:**
  - Intercepting GPU commands at hypervisor level
  - Cache and reuse previous computed GPU results
- **Related Work:**
  - hardware-based task level memoization. (ISCA'14, Arnau et. al)
  - Pattern-based approximation (ASPLOS'14, Samadi et.al)

GRU(Gpu Result re-Use) Ecosystem

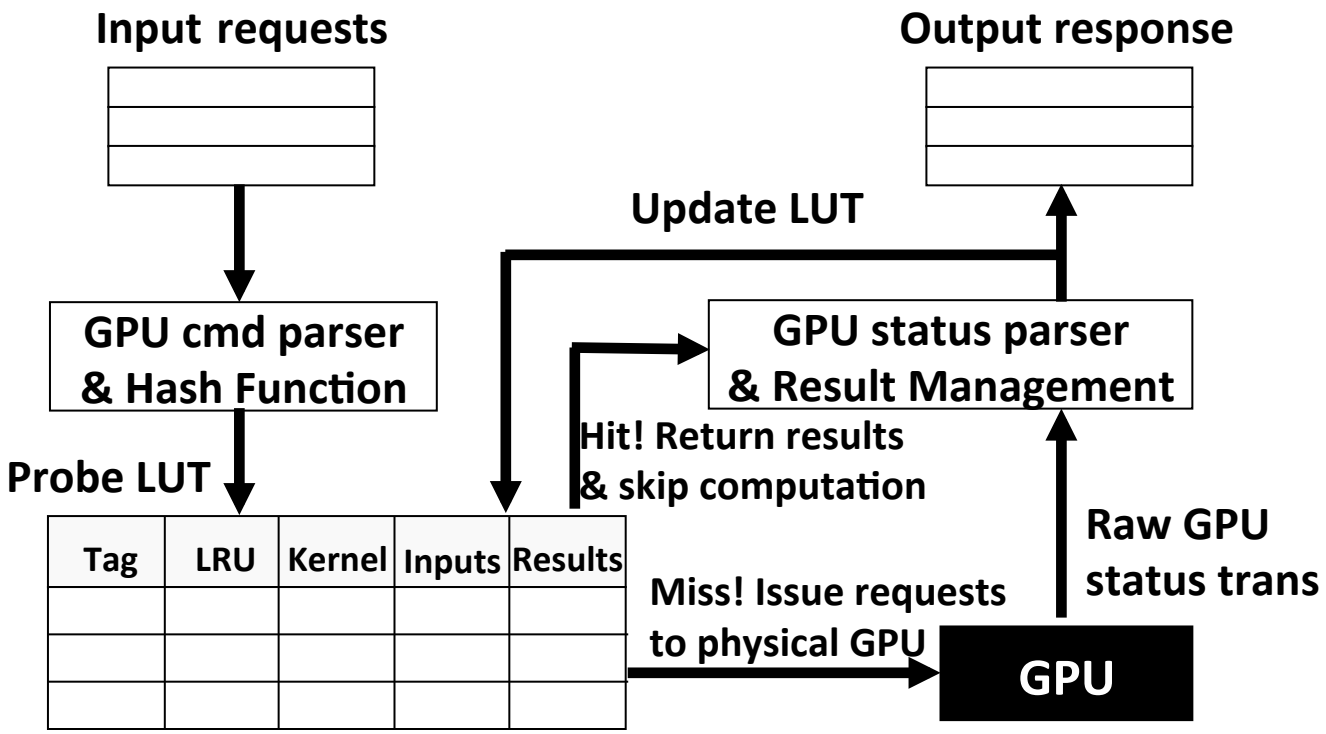


Qemu-Xen : collects the MMIO operations and forwards GPU commands

Aggregator : interprets the parameters of GPU requests and manages the GPU memory;

Reuse engine : uses LUT to identify cached results and stores computation results in a "shadow result memory"

Workflow


Input requests
Output response
Update LUT
GPU cmd parser
& Hash Function
GPU status parser
& Result Management
Hit! Return results
& skip computation
Probe LUT
Raw GPU
status trans
Tag
LRU
Kernel
Inputs
Results
Miss! Issue requests
to physical GPU
GPU

Experiment Setup

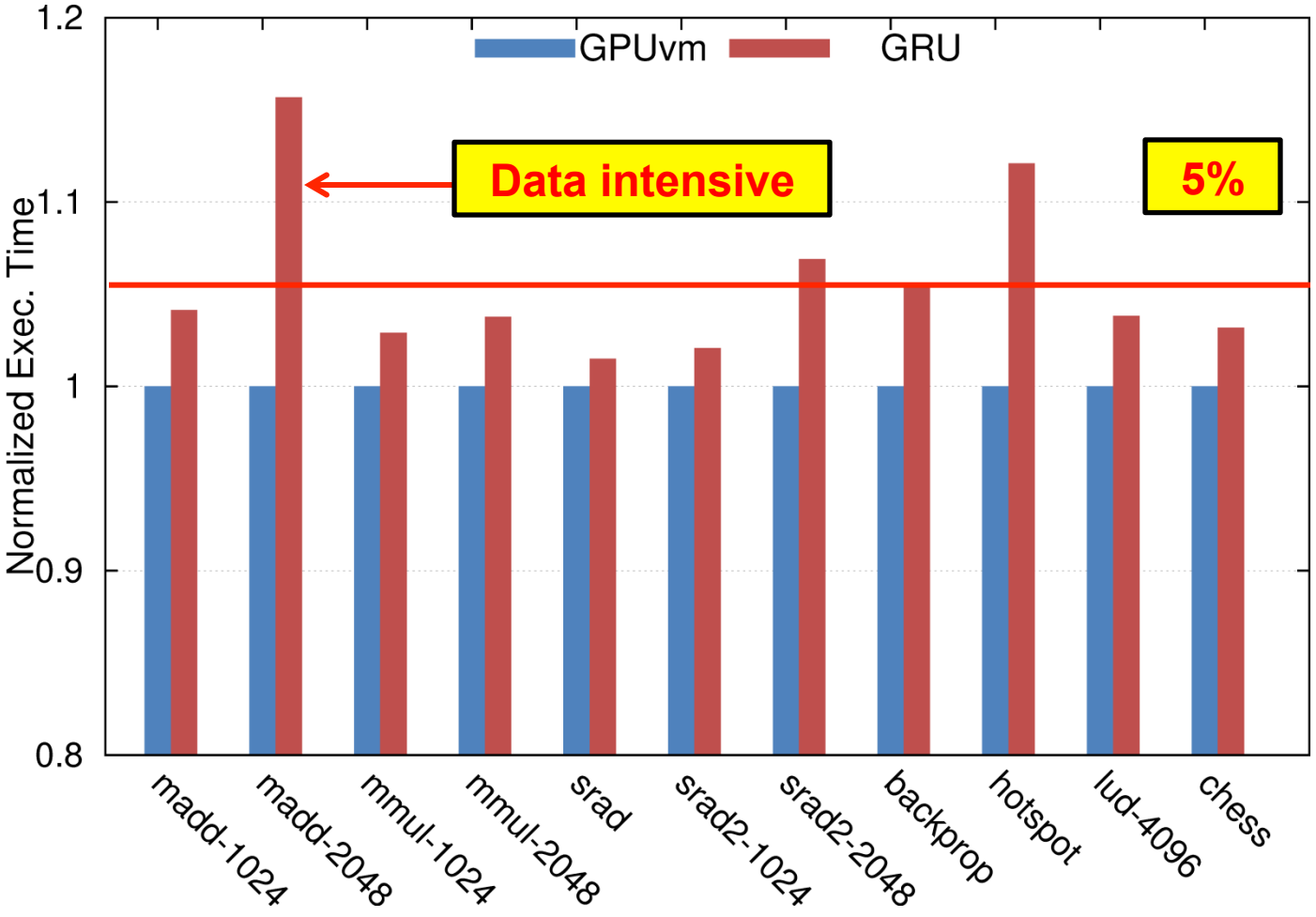
- **Overhead**
- **Throughput performance**

Device	Intel Core i7 – 4790K	NVIDIA Quadro 6000
Memory (B/W)	16 GB DDR3 (12.8 GB/s)	6 GB GDDR5 (133.9 GB/s)
Avg Perf.	177 GFlops	1,345 GFlops
PCIe 3.0 x16	15.76 GB/s	
Dom0	Ubuntu 12.04 - 3.6.5 + Gdev	
DomU	Ubuntu 12.04 - 3.6.5 + Gdev + CUDA tool kit 4.2	
Hypervisor	Xen-4.2 + xen tool + GRU	

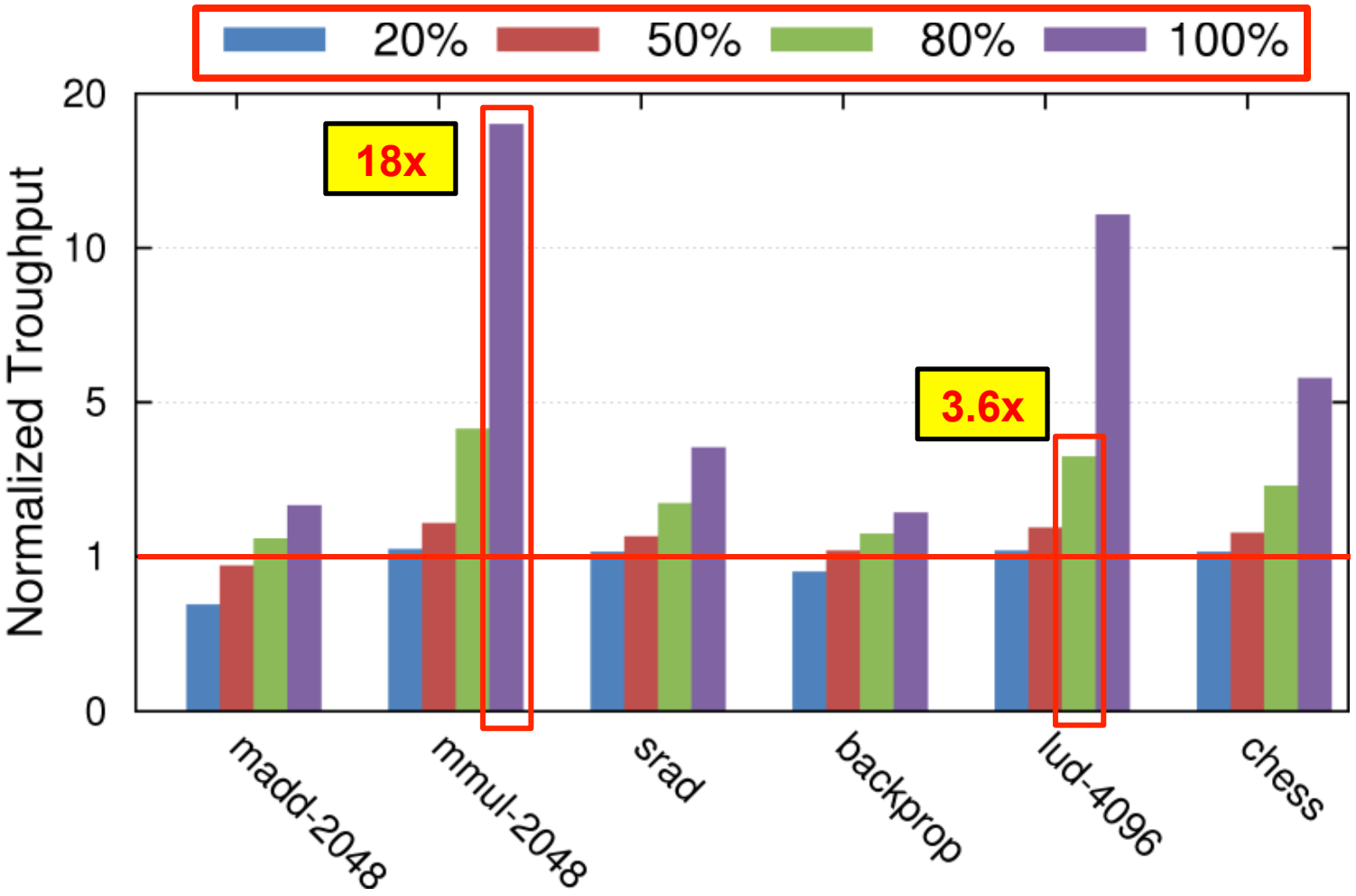
Benchmarks

Application	Description
chess	Chinese Chess game with naive AI
madd	Matrix addition
mmul	Matrix multiplication
srad	Speckle reducing anisotropic diffusion
srad2	Another version of srad with pseudo-inputs
backprop	Back propagation
hotspot	Physics simulation
lud	LU Decomposition

Overhead


GPUvm
GRU
Data intensive
5%
Normalized Exec. Time
1.2
1.1
1
0.9
0.8
madd-1024
madd-2048
mmul-1024
mmul-2048
srad
srad2-1024
srad2-2048
backprop
hotspot
lud-4096
chess

Speedup


20%
50%
80%
100%
Normalized Troughput
20
10
5
1
0
18x
3.6x
madd-2048
mmul-2048
srad
backprop
lud-4096
chess

Conclusion and Ongoing work

- **We present the first system solution for the idea of GPU result memoization and result reuse in the cloud**
- **Stabilize and generalize GRU**
  - Support complicated applications with dependent kernels
- **Support partial result reuse**
  - Introducing reuse-measuring metrics
  - Compiler-assisted approach
- **Enlarge reusability**
  - Approximate computing
- **Scheduling:**
  - Reuse-oriented scheduling

Ongoing Evaluations

- **We are evaluating some popular GPGPU cloud applications**
  - Deep-learning for image recognition
  - Motifs detection tool for DNA
  - GPU accelerated MapReduce and database query

Discussion

Thanks

Discussion

- How to identify partial reusable result
- Reduce overhead for data hashing
- Representative cloud applications