PTask:
Operating System Abstractions
To Manage GPUs as Compute Devices

Christopher J. Rossbach Jon Currey Mark Silberstein

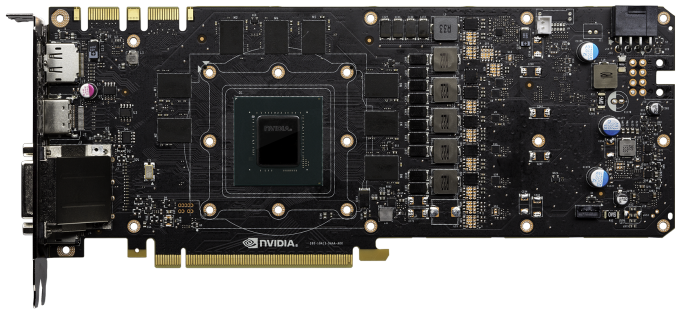
SOSP, 2011

Soyoung Park

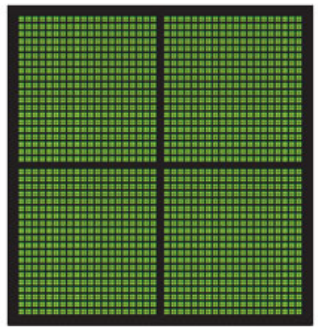
INDEX

Introduction

- **What is GPU and Why we need it?**
  - GPU(Graphics Processing Unit):
    - A programmable logic chip (processor) specialized for display functions.
    - Renders images, animations and video for the computer's screen
    - Thousands of cores to process parallel workloads efficiently
  - GPUs have surpassed CPUs as a source of high-density computing devices.
    - accompanied by emergence of general purpose GPU (GPGPU) frameworks such as DirectX, CUDA, and OpenCL



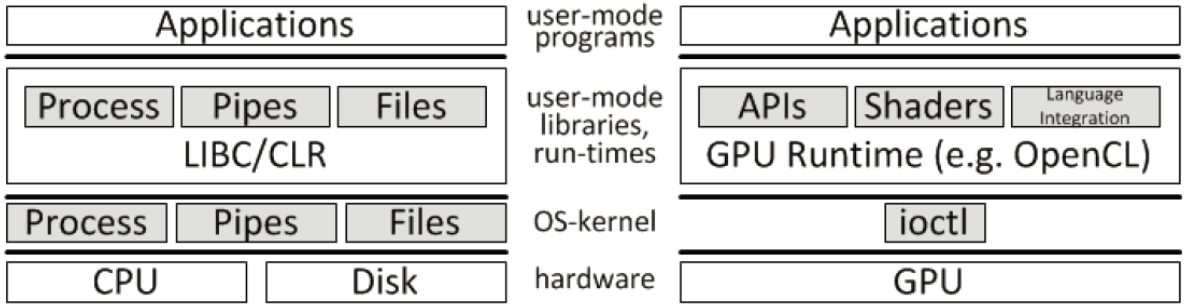
CPU
MULTIPLE CORES



GPU
THOUSANDS OF CORES

Introduction

- **Limited use of GPUs as compute engine**
  - Compute-intensive interactive applications need GPU for their computation
  - kernel-level abstractions for GPU are limited, compared to that of CPU



  - using the GPU from kernel mode driver is not currently supported.
  - OS leaves resource management for GPUs to vendor-supplied drivers and user-mode run-times.

  - OSes need to manage GPUs as a computational devices, like CPU.

Introduction

- **Kernel-level abstractions for managing interactive, high compute devices(especially GPUs).**
  - General Principles:
    - Expose enough hardware detail of the devices for programmers to take advantage of their processing capabilities.
    - Hide inconveniences like memory incoherency between the CPU and GPU, or problems related to performance.
    - Traditional OS guarantees such as fairness and isolation.
    - Provide abstractions that allow programmers to write code that is both modular and performant.

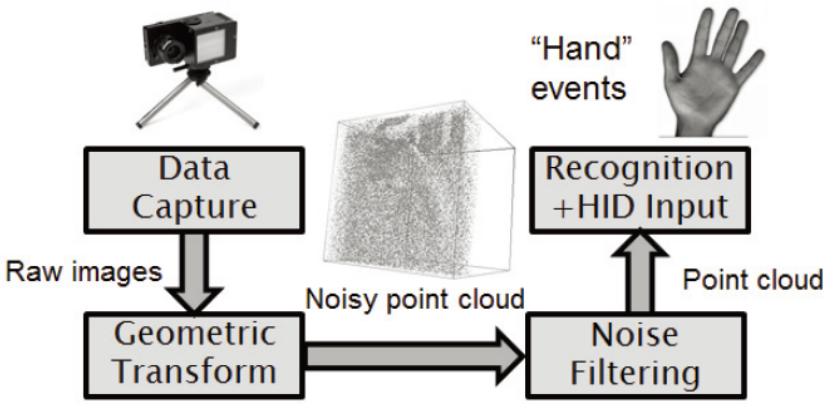
- **Ptask API**
  - provide a dataflow programming model:
    - programmer writes code to manage a graph-structured computation.
    - programmer expresses only *where* data must move, but not *how* or *when*.

Motivation

- **Interactive applications with GPGPU**
  - examples:
    - gesture-based interfaces
    - neural interfaces
    - encrypting file systems
    - realtime audio/visual interfaces (e.g. speech recognition)

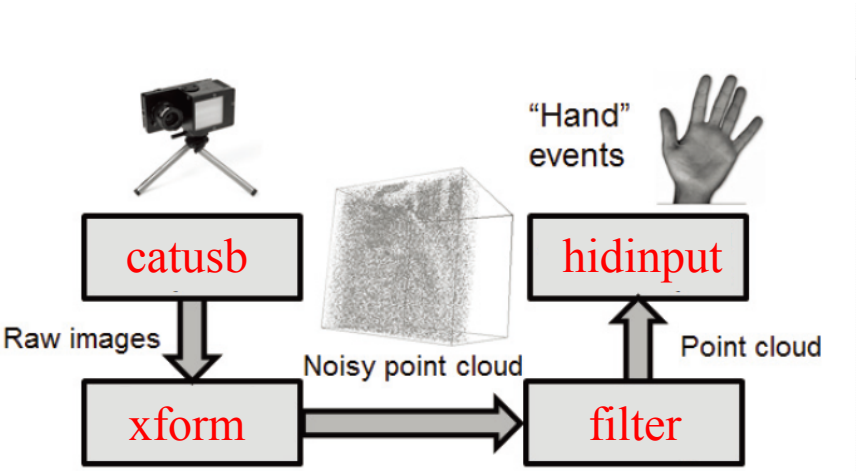
  - These tasks are computationally demanding, have real-time performance and latency constraints, and feature many data-independent phases of computation.

  - Case study:
    - Gestural recognition system that turns user's hand motions into OS input events.



Motivation

- **Case study:**
  - Gestural recognition system that turns user's hand motions into OS input events.
  - components:



    - **catusb** - captures image data from cameras connected on a USB bus
    - **xform** - geometric transformations from each data to a single point cloud
    - **filter** – noise filtering on the point cloud data. data parallel.
    - **hidinput** - detects gestures in a point cloud and sends them to the OS as human interface device (HID) input
  - The system is modular: communication between components needed.
  - data parallelism in **xform** and **filter** argue for GPU acceleration.

Motivation

- **Problem: data movement**
  - (CUDA-based implementation)
  - GPGPU frameworks requires the main memory data to be transferred to the device before the computation and then back to the host to be read .
  - Running the pipeline suffers from excessive data movement:
    - across the user-kernel boundary and
    - from main memory to GPU memory.
  - system spends far more time marshaling data structures and migrating data than it does actually computing on the GPU.

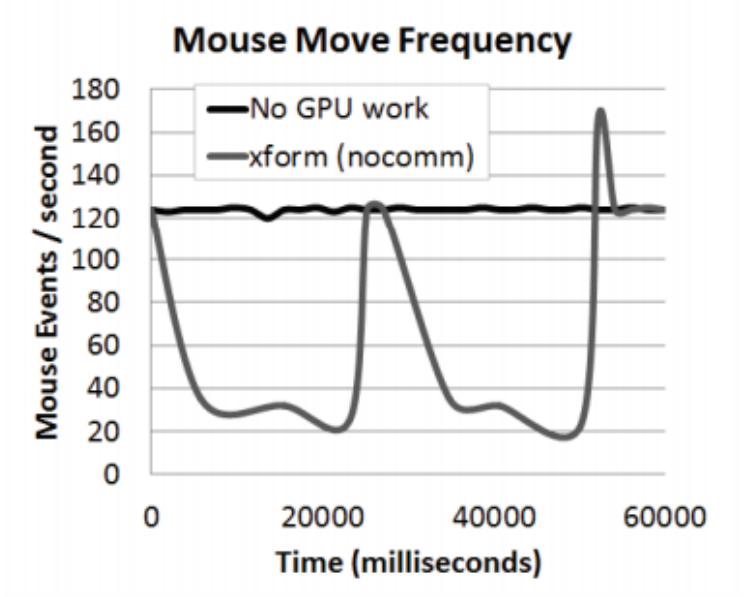
Motivation

- **Problem: No easy fix for data movement**
  - Some GPGPU frameworks offers solutions for data migration between GPU and CPU,
    - e.g. CUDA streams: asynchronous buffer copy
  - But they are not easy to use.
    - A programmer must understand OS-level issues like memory mapping.
    - a static knowledge of which transfers can be overlapped with which computations.
    - Moreover, they are effective only with available communication to perform that is independent of the current computation
  - Optimizing data movement remains important.

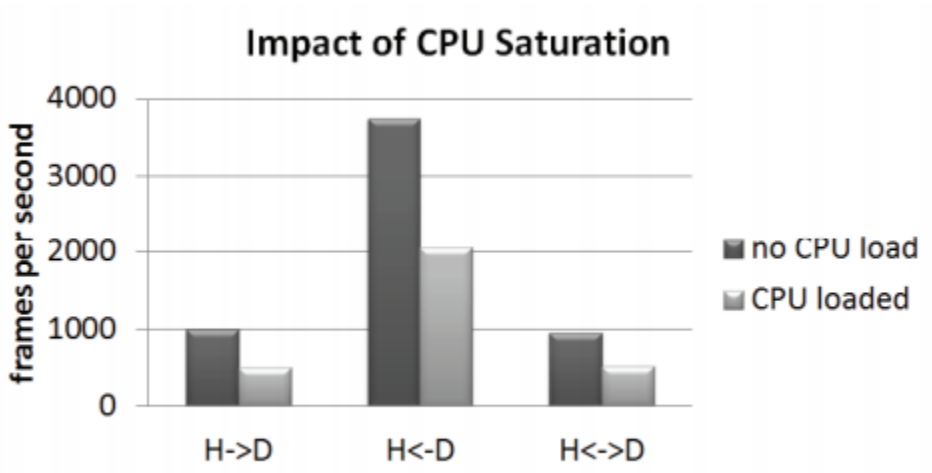
Motivation

- **Problem: Scheduling problem**
  - OS does not treat GPUs as a shared computational resource, like a CPU, but rather as an I/O device.
    - OS does not guarantee fairness and performance for systems that use GPUs for computation
  - GPU work causes system pauses.
    - Significant GPU-work at high frame rates causes the system to be unresponsive for seconds at a time.
    - Since in-progress I/O requests cannot be canceled once begun, GPUs are not preemptible.
    - Windows relies on cancelation to prioritize its own work, its priority mechanism fails.
  - CPU work interferes with GPU throughput.
    - CPU-bound process have impact on the frame rate of a shader program.

Motivation



GPU work causes system pauses.



CPU work interferes with GPU throughput.

Design

- **PTask: New OS abstractions to support GPU programming**
  - PTask (Paraellel Task) API
  - consists of interfaces + runtime library support
  - Support a dataflow programming model
    - Assemble individual tasks into a directed acyclic graph (DAG)
    - **Vertices (ptasks)**: executable code including shader program on GPU, code fragments on other accelerators, and callbacks on CPUs.
    - **Edges**: data flow, connecting inputs and outputs of each vertex.

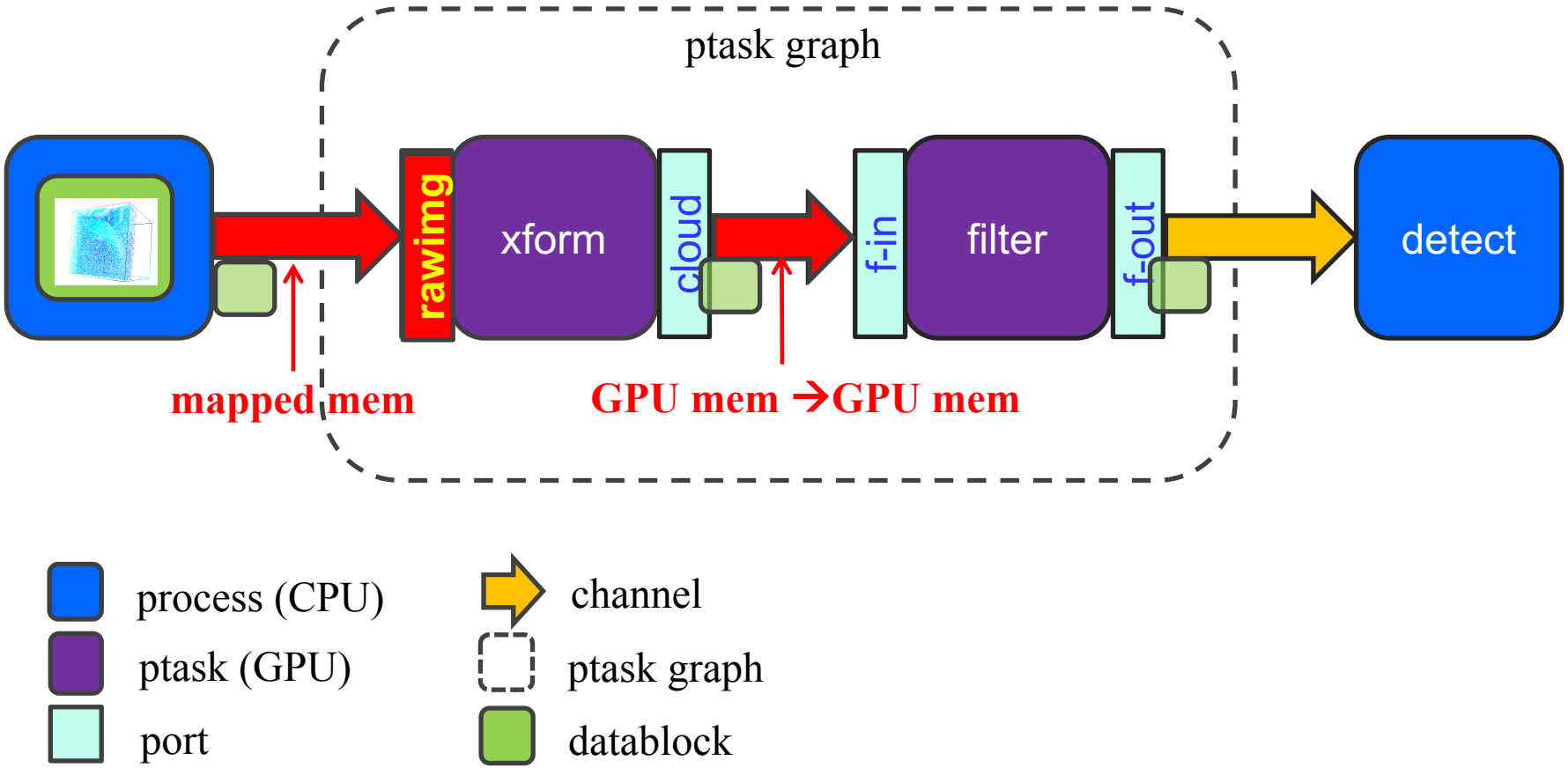
- **Design goals:**
  - 1. Bring GPUs under the purview of a single resource manager.
  - 2. Provide a simple programming model for accelerators.
  - 3. Allow code to be both modular and fast.

PTask API

- **PTask abstractions**
  - PTask.
    - Analogous to the traditional OS process, runs on GPU
    - List of input/output resources (e.g. stdin, stdout…)
  - Ports.
    - Can be mapped to ptask input/outputs
    - A data source or sink
  - Channels.
    - Similar to pipes, connect arbitrary ports
    - Specialize to eliminate double-buffering
  - Graph.
    - collection of ptasks, connected channels with ports
  - Datablocks and Template.
    - A unit of dataflow along an edge in a graph
    - A templete provides meta-data describing data blocks, and help map the raw data to hardware threads on GPU.

PTask API

- **Dataflow through the Graph**



Slide reference: Slides for "PTask: Operating System Abstractions to Manage GPUs as Compute Devices", SOSP October 25, 2011

Implementation

- **PTask Scheduling**
  - Challenges:
    - GPU hardware cannot currently be preempted or context-switched.
    - True integration with the process scheduler is not currently possible due to lack of an OS-facing interface to control the GPU in Windows
    - with multiple GPUs, data locality becomes the primary determinant of performance. (Parallel execution not recommended)
  - Four scheduling modes
    - First-available
    - FIFO
    - priority
    - data-aware

Implementation

- **First Available**
  - Every ptask is assigned a manager thread, competing for available accelerators.
  - Ready ptasks are not queued: when ready ptasks outnumber available accelerators, access is arbitrated by locks on the accelerator data structures.

- **FIFO**
  - first Availble + queue

Implementation

- **Priority mode**
  - Sort the queue based on each ptask's priority
  - Schedular thread computes effective priority value, enhanced by static priority and proxy priority
  - proxy priority:
    - OS priority of the thread managing its invocation and data flows.
    - avoid proxy laundering
    - enables a ptask's manager thread to assume the priority of a requesting process.

- **Data-aware mode**
  - same effective priority system that the priority policy uses,
  - consider the memory spaces where a ptask's inputs are currently up-to-date.

Conclusions

- **OS abstractions for GPUs are critical**
  - Enable fairness & priority
  - OS can use the GPU

- **Dataflow: a good fit abstraction**
  - system manages data movement
  - performance benefits significant

Conclusions

- **Contributions**

  - Provides quantitative evidence that modern OS abstractions are insufficient to support a class of “interactive” applications that use GPU’s

  - Provides a design for OS abstractions to support a wide range of GPU computations with traditional OS guarantees like fairness and isolation.

  - Provides a prototype of the PTask API and a GPU-accelerated gestural interface

  - Demonstrates a prototype of GPU-aware scheduling in the Linux kernel that forces GPU-using applications to respect kernel scheduling priorities.

Thank you.