King Abdullah University of Science and Technology

KAUST

CS 380 - GPU and GPGPU Programming
Lecture 17: Stream Computing and GPGPU

Markus Hadwiger, KAUST

Reading Assignment #9 (until Nov 4)

Read (required):

- **Brook for GPUs: Stream Computing on Graphics Hardware**
  Ian Buck et al., SIGGRAPH 2004

`http://graphics.stanford.edu/papers/brookgpu/`

Read (optional):

- **The Imagine Stream Processor**
  Ujval Kapasi et al.; IEEE ICCD 2002

`http://cva.stanford.edu/publications/2002/imagine-overview-iccd/`

- **Merrimac: Supercomputing with Streams**
  Bill Dally et al.; SC 2003

`https://dl.acm.org/citation.cfm?doid=1048935.1050187`

Types of Parallelism

Bit-Level Parallelism (70s and 80s)

- Doubling the word size 4, 8, 16, 32-bit (64-bit ~2003)

Instruction-Level Parallelism (mid 80s-90s)

- Instructions are split into stages → multi stage pipeline
- Superscalar execution, …

Data Parallelism

- Multiple processors execute the same instructions on different parts of the data

Task Parallelism

- Multiple processors execute instructions independently

From GPU to GPGPU

1990s Fixed function graphics-pipeline used for more general computations in academia (e.g., rasterization, z-buffer)

2001 Shaders changed the API to access graphics cards

2004 Brook for GPUs changed the terminology

Since then:

- ATI's Stream SDK (originally based on Brook)
- NVIDIA's CUDA (together with Brook developers)
- OpenCL (platform independent)
- GLSL Compute Shaders (platform independent)

Stream Programming Abstraction

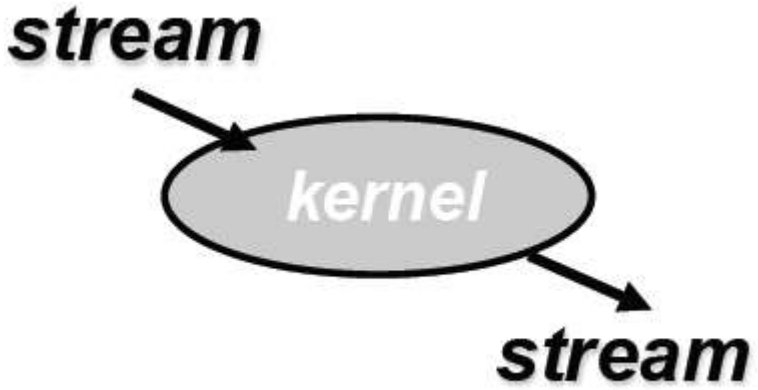
Goal: SW programming model that matches data parallelism

Streams

- Collection of data records
- All data is expressed in streams

Kernels

- Inputs/outputs are streams
- Perform computation on streams
  (each data record is processes independently)
- Can be chained together



Courtesy John Owens

Why Streams?

- Exposing parallelism
  - Data parallelism
  - Task parallelism

```
for(i = 0; i<size; i++)
{
    a[i] = 2*b[i];
}
```

```
for(each a, b)
{
    a = 2*b;
}
```

```
for(i = 0; i<size; i++)
{
    a[i] = a[i+1]*2;
}
```

```
for(each a)
{
    ???
}
```

- Multiple stream elements can be processed in parallel
- Multiple tasks can be processed in parallel
- Predictable memory access pattern
- Optimize for throughput of all elements, not latency of one
- Processing many elements at once allows latency hiding

Thank you.

- John Owens
- Ian Buck et al.
- AMD