GPGPU Programming
&
Erlang

Kevin A. Smith

What is GPGPU Programming?

- Using the graphics processor for non-graphical programming
- Writing algorithms for the GPU instead of the host processor

Why?

- Ridiculous amount of parallelism
- Very high throughput
- Economical

Common Uses

- Medical Imaging
- Oil & Gas Exploration
- Data Mining
- Algorithmic Trading

Moving Into The Mainstream

- **PacketShader: A GPU-Accelerated software router** (Han, Jang, Park & Moon)
- **SSLShader: Cheap SSL Acceleration with Commodity Processors** (Jang, Han, Han, Moon & Park)
- **Mars: A MapReduce Framework on Graphics Processors** (He, Fang, Govindaraju, Luo & Tuyong)
- **Augmenting Operating Systems With the GPU** (Sun & Ricci)

GPGPU & You

- Sorting
- Searching
- Reductions: average, median, etc
- Transforms: uniquifying, permutations

GPGPU & You

- Extrema: min/max
- Prefix scans & sums
- Set operations
- High quality random numbers

What is CUDA?

- GPGPU framework for NVIDIA GPUs
- Extends programmable shaders
- Extensions to C
- Runtime framework

Why Use CUDA?

- Wide array of cards
- Shallow learning curve
- Lots of resources

GeForce 210

CPU: 16 Cores
Memory:1 GB
Cost: $44.99

GeForce GT 240

CPU: 96 Cores
Memory: 1 GB
Cost: $79.99

GeForce GTS 450

CPU: 192 Cores
Memory:1 GB
Cost: $149.99

GeForce GTX 460

CPU: 336 Cores
Memory: 1 GB
Cost: $230 (approx.)

GeForce GTX 470

CPU: 448 Cores
Memory: 1.2 GB
Cost: $280 (approx.)

GeForce GTX 480

CPU: 480 Cores
Memory: 1.5 GB
Cost: $425 (approx.)

GeForce GTX 580

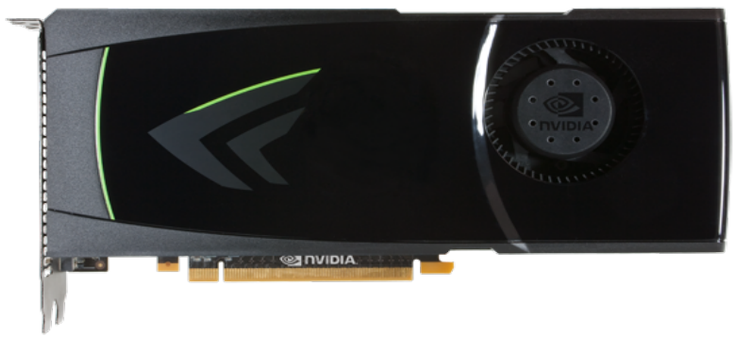
CPU: 512 Cores
Memory: 1.5 GB
Cost: $499+ (approx.)

GeForce GTX 590

CPU: 1024 Cores
Memory:3.5 GB
Cost: $699+ (approx.)

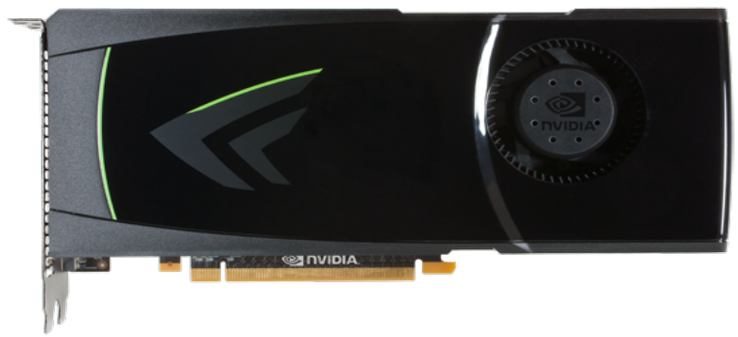
How CUDA Works

Write Data



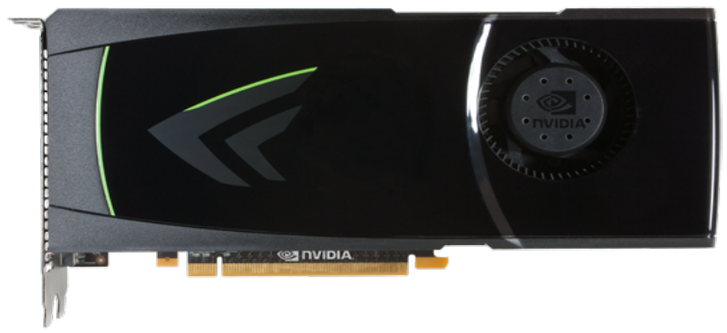
Runs at PCIe speeds
~ 2.5Gbps

Execute Code



Runs at PCIe speeds
~ 2.5Gbps

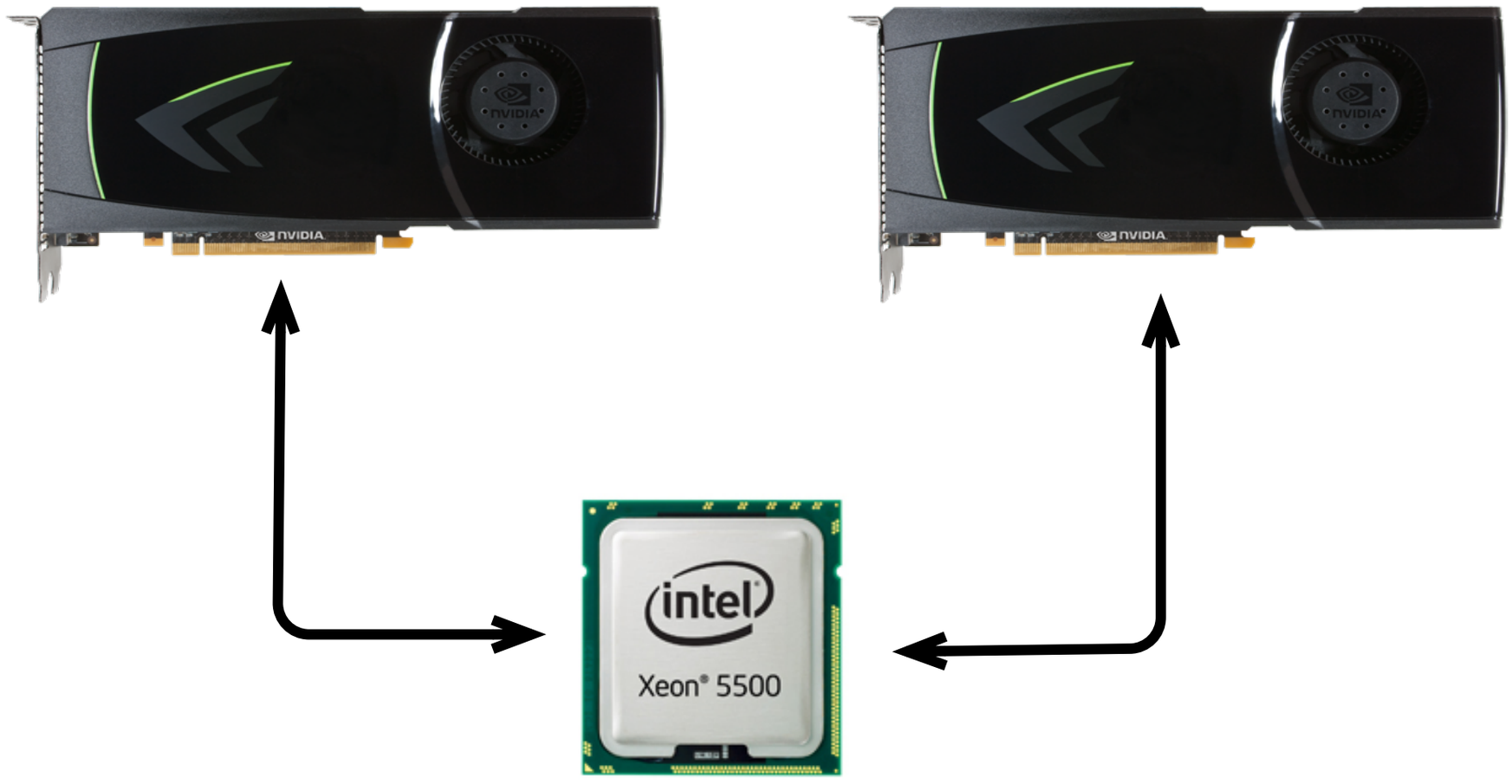
Read Result



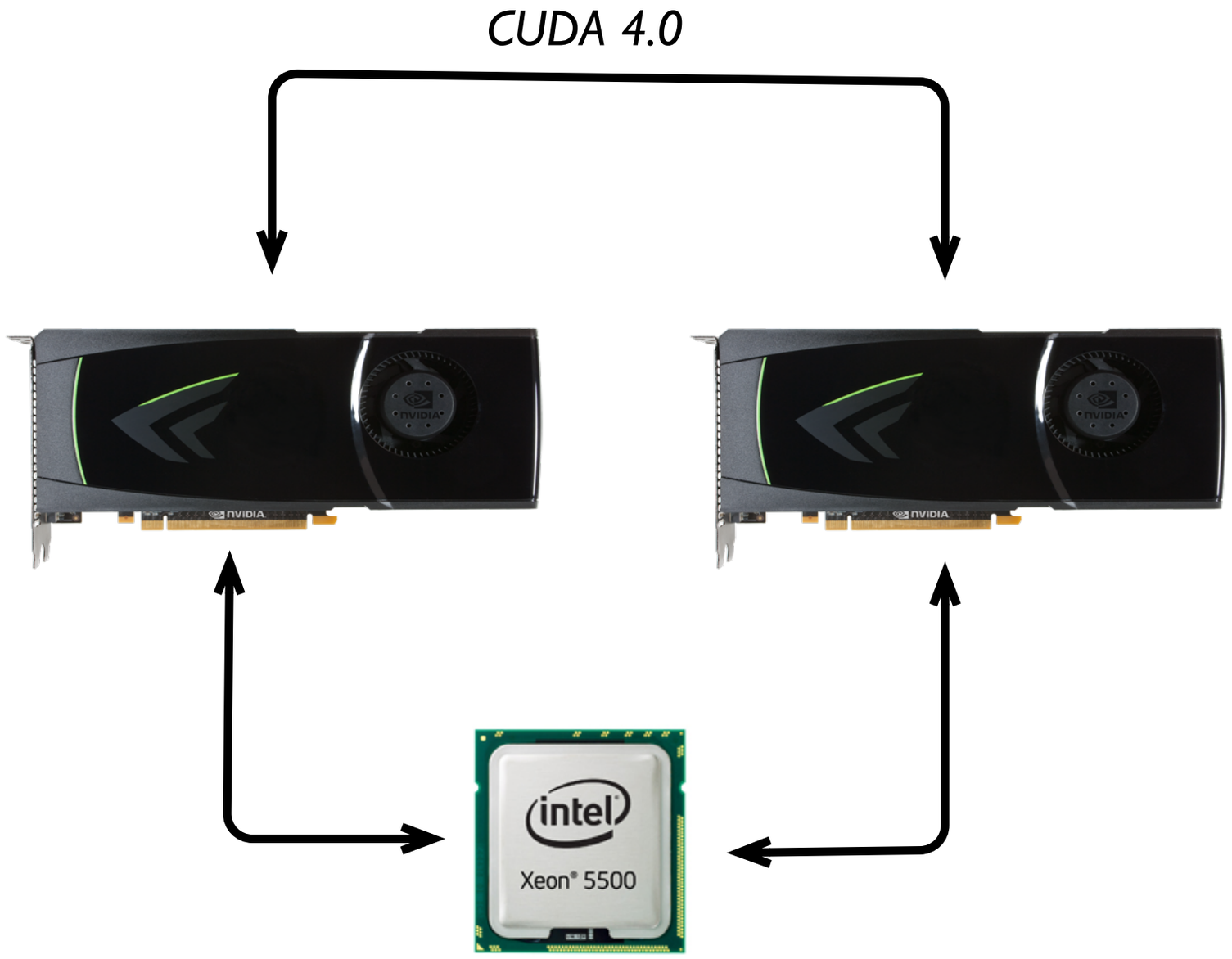
BOTTLENECK

~ 2.5Gbps

Multiple GPUs



Multiple GPUs



(Brief) CUDA Overview

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

```
#include <iostream>

#include "cuda.h"

__global__ void add(int a, int b, int *c) {
    *c = a + b;
}

int main() {
    int c;
    int *dev_c;
    add<<1, 1>>(100, 100, dev_c);
    cudaMemcpy(&c, dev_c, sizeof(int), cudaMemcpyDeviceToHost);
    printf("100 + 100 = %d\n", c);
    cudaFree(dev_c);

    return 0;
}
```

Summing Vectors

37	+	99	=	???
86	+	45	=	???
77	+	67	=	???
92	+	68	=	???

Summing Vectors

```
__global__ void add(int *first, int *second, int *result) {
    int thread_id = blockIdx.x;
    result[thread_id] = first[thread_id] + second[thread_id];
}
```

Summing Vectors

```
int main() {
    /* Declare data and result vectors */
    int first[100], second[100], result[100];
    int *dev_first, *dev_second, *dev_result;

    /* Alloc work space on device */
    cudaMalloc( (void **) &dev_first, 100 * sizeof(int));
    cudaMalloc( (void **) &dev_second, 100 * sizeof(int));
    cudaMalloc( (void **) &dev_result, 100 * sizeof(int));

    /* Populate vectors on host */
    for (int i = 0; i < 100; i++) {
        first[i] = i;
        second[i] = i * 2;
    }

    /* Move data to device */
    cudaMemcpy(dev_first, first, 100 * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_second, second, 100 * sizeof(int), cudaMemcpyHostToDevice);

    add<<25, 1>>(dev_first, dev_second, dev_result);

    /* Copy results back to host */
    cudaMemcpy(result, dev_result, 100 * sizeof(int), cudaMemcpyDeviceToHost);
   cudaFree(dev_first);
   cudaFree(dev_second);
   cudaFree(dev_thirst);
}
```

CUDA
Pros & Cons

- Uses a well-known language (C)
- Extends common idioms to the GPU
- Verbose, Error prone
- Complete control over computing environment
- Low-level

Introducing Thrust

- C++ framework for GPU computing
- STL-like algorithms for common CUDA operations
- Reduces verbosity & errors
- Included in CUDA 4.0

Summing Vectors

```
int main() {

    thrust::device_vector<int> dev_first(100);
    thrust::device_vector<int> dev_second(100);
    thrust::device_vector<int> dev_result(100);
    std::vector<int> result(100);

    // Fill device arrays with data
    thrust::sequence(dev_first.begin(), dev_first.end(), 1);
    thrust::sequence(dev_second.begin(), dev_second.end(), 1);

    thrust::add<int> op;

    // Add arrays
    thrust::transform(dev_first.begin(), dev_first.end(), dev_second.begin(),
                      dev_result.begin(), op);
    // Copy results to host array
    thrust::copy(dev_result.begin(), dev_result.end(), result.begin());

}
```

pteracuda

What is pteracuda?

- Experiment to integrate CUDA & Erlang
- Accelerate slow Erlang operations
- Built on CUDA 4.0 RC1

Main Concepts

- Context - Environment for CUDA operations

```
{ok, Ctx} = pteracuda_context:new().
pteracuda_context:destroy(Ctx).
```

Main Concepts

- Buffer - Typed list of data
  - Integer
  - Float
  - String ***

```
{ok, Buf} = pteracuda_buffer:new(integer).
pteracuda_buffer:destroy(Buf).
```

Working With Buffers

- Write
- Read
- Insert
- Delete
- Clear

CUDA Accelerated Operations

- Sort
- Contains (binary search)
- Intersection
- Min/Max

DEMO

https://github.com/kevsmith/pteracuda