Motivation

GPUs democratized HPC
Motivation

MOOCs have democratized education
Motivation

Learning to program involves programming and debugging

```plaintext
for d = 0 to \log_2 n - 1 do
  forall k = 0 to n - 1 by 2^{d+1} in parallel do
    x[k + 2^{d+1} - 1] \leftarrow x[k + 2^d - 1] + x[k + 2^{d+1} - 1]
  end
end
```
Motivation

GPU Availability and Hardware Characteristics

- Not everyone has a GPU
- Not everyone can install a development environment
- We want to teach GPU programming, not system configuration and compilation
- Even if a development environment can be installed, there is no way to ensure environment is consistent
- GPUs cannot be time sliced and preempted like CPUs
Motivation

Statically-provisioning remote GPU resources is expensive

<table>
<thead>
<tr>
<th>Year</th>
<th>Registered Users</th>
<th>Completions</th>
<th>Completion Rate</th>
<th>Certificates Issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>36896</td>
<td>2729</td>
<td>7.40%</td>
<td>-</td>
</tr>
<tr>
<td>2014</td>
<td>33818</td>
<td>1061</td>
<td>3.14%</td>
<td>286</td>
</tr>
<tr>
<td>2015</td>
<td>35940</td>
<td>1141</td>
<td>3.15%</td>
<td>442</td>
</tr>
</tbody>
</table>
Student Programming Behavior

- Host code is prone to memory errors
- Buffer overflows / double frees
- Parallel programming is hard (non-deterministic errors)
- Performance optimizations introduce non-trivial bugs
- GPUs are not robust enough -- with the right set of bugs you can crash the PCI bus (machine)
Student Programming Behavior
Active users per hour from Feb 8th 2015 to april 15th 2015

Deadlines are chosen to fall mid week to distribute the workload across people who can only work on weekends and ones who can only work before deadlines.
Current WebGPU Architecture
Decouple the GPUs from the UI

- Users interact with web interface
  - Users only need an internet connection and web browser
Current WebGPU Architecture
Decouple the GPUs from the UI

- The web interface interacts with a database server ② and one or more GPU worker ③
  - GPU workers scale based on active users
- User code is compiled and run with constraints within a sandboxed environment
  - Damage from poorly-behaved programs is contained
PUMPS Summer School Prerequisites

This site serves for the development of the prerequisites for PUMPS 2015 Summer School. The site is the place where all the programming assignments will be developed and submitted. To get started, you first need sign up for an account.

Using WebGPU
Vector Addition

The coding deadline is 18 days from now.

Objective

The purpose of this lab is to get you familiar with using the CUDA API by implementing a simple vector addition kernel and its associated setup code.

Prerequisites

Before starting this lab, make sure that:

- You have completed all week 1 lecture videos
- You have completed "Lab Tour with Device Query" MP
- You have looked over the tutorial document.
- Chapter 3 of the text book would also be helpful

Instruction

Edit the code in the code tab to perform the following:

- Allocate device memory
- Copy host memory to device
- Initialize thread block and kernel grid dimensions
- Invoke CUDA kernel
- Copy results from device to host
- Free device memory
- Write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

Suggestions

- The system's autosave feature is not an excuse to not backup your code and answers to your questions regularly.
```c
#include <wb.h>

__global__ void vecAdd(float * in1, float * in2, float * out, int len) {
    // Insert code to implement vector addition here
    int index = threadIdx.x + blockDim.x * blockIdx.x;
    if (index < len) {
        out[index] = in1[index] + in2[index];
    } else {
        printf("\dn", index);
    }
}

int main(int argc, char ** argv) {
    wbArg * argc;
    int inputLength;
    float * hostInput1;
    float * hostInput2;
    float * hostOutput;
    float * deviceInput1;
    float * deviceInput2;
    float * deviceOutput;
    argc = wbArg_read(argc, argv);
    wbTime_start(Generic, "Importing data and creating memory on host");
    hostInput1 = (float *) wbImport(wbArg_getInputFile(argc, 0), &inputLength);
    hostInput2 = (float *) wbImport(wbArg_getInputFile(argc, 1), &inputLength);
    hostOutput = (float *) malloc(inputLength * sizeof(float));
    wbTime_stop(Generic, "Importing data and creating memory on host");
    wbLog(TRACE, "The input length is \", inputLength);
    wbTime_start(GPU, "Allocating GPU memory.");
    // Allocate GPU memory here
    cudaMalloc((void **) &deviceInput1, inputLength*sizeof(float));
    cudaMalloc((void **) &deviceInput2, inputLength*sizeof(float));
    cudaMalloc((void **) &deviceOutput, inputLength*sizeof(float));
    wbTime_stop(GPU, "Allocating GPU memory.");
    wbTime_start(GPU, "Copying input memory to the GPU.");
    // Copy memory to the GPU here
    cudaMemcpy(deviceInput1, hostInput1, inputLength*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(deviceInput2, hostInput2, inputLength*sizeof(float), cudaMemcpyHostToDevice);
    wbTime_stop(GPU, "Copying input memory to the GPU.");
    // Initialize the grid and block dimensions here
    dim3 blockDim(32);
    dim3 gridDim(ceil(((float) inputLength)/(float) blockDim.x)));
    wbLog(TRACE, "Block dimension is \", blockDim.x);
    wbLog(TRACE, "Grid dimension is \", gridDim.x);
    wbTime_start(Compute, "Performing CUDA computation");
    // Launch the GPU kernel here
    vecAdd<<<gridDim, blockDim>>>(deviceInput1, deviceInput2, deviceOutput, inputLength);
    wbTime_stop(Compute, "Performing CUDA computation");
}````
Vector Addition Attempt

Attempt Summary

- Dataset Id: 1
- Created: less than a minute ago
- Status: Correct solution for this dataset.

Timer Output

<table>
<thead>
<tr>
<th>Kind</th>
<th>Location</th>
<th>Time (ms)</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>main.cu:26</td>
<td>1.34021</td>
<td>Importing data and creating memory on host</td>
</tr>
<tr>
<td>GPU</td>
<td>main.cu:34</td>
<td>0.153063</td>
<td>Allocating GPU memory</td>
</tr>
<tr>
<td>GPU</td>
<td>main.cu:41</td>
<td>0.028161</td>
<td>Copying input memory to the GPU</td>
</tr>
<tr>
<td>Compute</td>
<td>main.cu:54</td>
<td>0.711848</td>
<td>Performing CUDA computation</td>
</tr>
<tr>
<td>Copy</td>
<td>main.cu:60</td>
<td>0.024158</td>
<td>Copying output memory to the GPU</td>
</tr>
<tr>
<td>GPU</td>
<td>main.cu:85</td>
<td>0.14592</td>
<td>Freeing GPU Memory</td>
</tr>
</tbody>
</table>

Logger Output

<table>
<thead>
<tr>
<th>Level</th>
<th>Location</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace</td>
<td>main:32</td>
<td>The input length is 128</td>
</tr>
<tr>
<td>Trace</td>
<td>main:51</td>
<td>Block dimension is 32</td>
</tr>
</tbody>
</table>
Vector Addition Grade

Grade Summary (History)

<table>
<thead>
<tr>
<th>Created:</th>
<th>less than a minute ago</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Score:</td>
<td>100 out of 100 points</td>
</tr>
<tr>
<td>Coding Score:</td>
<td>100 out of 80 points</td>
</tr>
<tr>
<td>Coding Score on Coursera</td>
<td>100</td>
</tr>
<tr>
<td>Questions Score:</td>
<td>0</td>
</tr>
<tr>
<td>Peer Review Score:</td>
<td>20 out of 20 points</td>
</tr>
<tr>
<td>Peer Review Score on Coursera:</td>
<td>20 out of 20 points</td>
</tr>
</tbody>
</table>

Program Code

```c
#include <wb.h>

__global__ void vecAdd(float *in1, float *in2, float *out, int len) {
    // Insert code to implement vector addition here
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if (index < len)
    {
        out[index] = in1[index] + in2[index];
    } else {
        printf("\d\n", index);
    }
}

int main(int argc, char **argv) {
    wbrArg_t argv;
    int inputLength;
    float *hostInput1;
    float *hostInput2;
    float *hostOutput;
    float *deviceInput1;
    float *deviceInput2;
    float *deviceOutput;
    argv = wbrArg_read(argc, argv);
    wbrTime_start(Generic, "Importing data and creating memory on host");
    hostInput1 = (float *) wbrImport(wbrArg_getInputFile(argv, 0), &inputLength);
    hostInput2 = (float *) wbrImport(wbrArg_getInputFile(argv, 1), &inputLength);
    wbrTime_stop(Generic, "Importing data and creating memory on host");
    /* Call the kernel function to perform the vector addition */
    vecAdd(hostInput1, hostInput2, hostOutput, len);
    wbrTime_start(Generic, "Exporting results to file");
    wbrExport(wbrArg_getOutputFile(argv, 0), &outLength, hostOutput);
    wbrTime_stop(Generic, "Exporting results to file");
    return 0;
}
```
Limitations with Current WebGPU

- Ad hoc way of creating lab
- Expects that grading is automated - limited support for TA activities
- UI is not easily customizable
- Software used by labs must be installed on all worker nodes
Next Generation WebGPU Architecture
Decouple the scheduler from the web server

- Users interact with an LTI web interface \(^1\) such OpenEdx - this provides instructors with UI for lab authoring
- The web interface interacts with a queue server \(^2\) which has one or more GPU workers \(^3\) subscribed to it
- Workers have links to the DB server \(^4\)
- Static content (such as videos) is served from an Amazon S3 Bucket \(^5\)
Next Generation WebGPU Worker Architecture

More secure sandbox environment using Docker

- Each worker is subscribed to the queue server ①
- Logs are sent to a centralized logging server ② and a configuration server ③
- An agent ④ utilizes a pool of containers ⑤ and orchestrates the dispatching on user jobs to GPU devices ⑥ on the system
Insights and Experiences

- WebGPU is robust - used for all GPU courses at UIUC
  - Only 2 worker nodes are needed for a modest size class (1000 users)
  - Time between submission and result is less than 1 minute
  - 99.9% uptime in the past Coursera offering
- Be able to adapt the platform and UI based on student input
- WebGPU performs online grading since users expect instant grade feedback and information to be reflected on Coursera
- Sharing content and experiences in running parallel programming courses will only refine the offering and delivery
Thank You

Please Come to the NVIDIA Teaching Kit Hands-on Lab Tomorrow at 5PM

Teach GPU Accelerating Computing: Hands-on with NVIDIA Teaching Kit

Dr. Wen-Mei Hwu (University of Illinois), Joe Bungo (NVIDIA), Abdul Dakkak (University of Illinois)

IPDPS ‘16
Chicago Hyatt Regency - Acapulco Room, West Tower Gold Level
Tues, May 24th, 6 – 8:00 PM
CUDA Curriculum Origin

NVIDIA scouts schools for first GPU teaching center. November 2006

NVIDIA Announces G80. Internal UIUC training. February 2007

First offering of a CUDA programming course. March 2007

UIUC projects adopt CUDA and publish compute patterns. June 2007

NVIDIA releases CUDA 1.0

Summer schools offered such as VSCSE, PUMPS, China, Chile

CUDA course becomes an official UIUC course August 2011

First Coursera offering of course November 2012

Second Coursera offering of course January 2014

Second Coursera offering of course January 2015

GPU Teaching kit announced at SC. November 2015

Version 1 of GPU Teaching kit released at GTC.

Fourth Coursera offering planned. March 2016

Fourth Coursera offering planned. January 2017
# Course Labs

<table>
<thead>
<tr>
<th>Lab</th>
<th>Description</th>
<th>HPP</th>
<th>408</th>
<th>598</th>
<th>PUMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Query</td>
<td>Demo Lab to introduce WebGPU to students.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Vector Addition</td>
<td>CUDA kernels.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic Matrix Multiplication</td>
<td>Boundary checking and indexing.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tiled Matrix Multiplication</td>
<td>Introduce shared memory tiling.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D Convolution</td>
<td>Constant memory and shared memory.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduction and Scan</td>
<td>Floating-point, work-efficiency, tree-like structures.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Equalization</td>
<td>Atomic operations.</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>OpenCL Vector Addition</td>
<td>OpenCL</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Scatter to Gather</td>
<td>Transformation between scatter and gather.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stencil</td>
<td>Register tiling and thread-coarsening.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>SGEMM</td>
<td>Register tiling and thread-coarsening.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>SPMV</td>
<td>Sparse matrix formats and performance effects.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Binning</td>
<td>Input Binning and performance effects.</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>BFS Queuing</td>
<td>Hierarchical queuing performance effects.</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-GPU Stencil with MPI</td>
<td>Multi-GPU programming and MPI.</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>