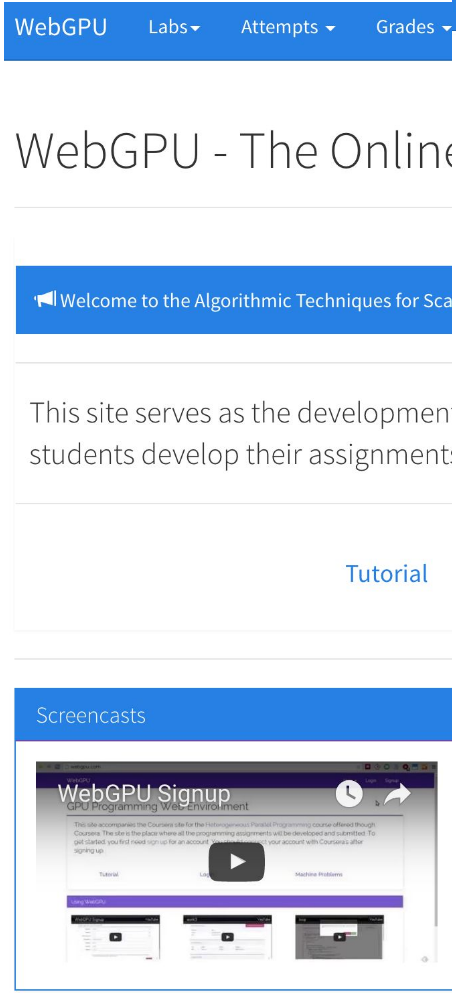
rai-project.com

Open-source Tools For GPU Programming in Large Classrooms

Abdul Dakkak, Carl Pearson, Cheng Li

{dakkak,pearson,cli99}@illinois.edu

WebGPU



WebGPU - The Online

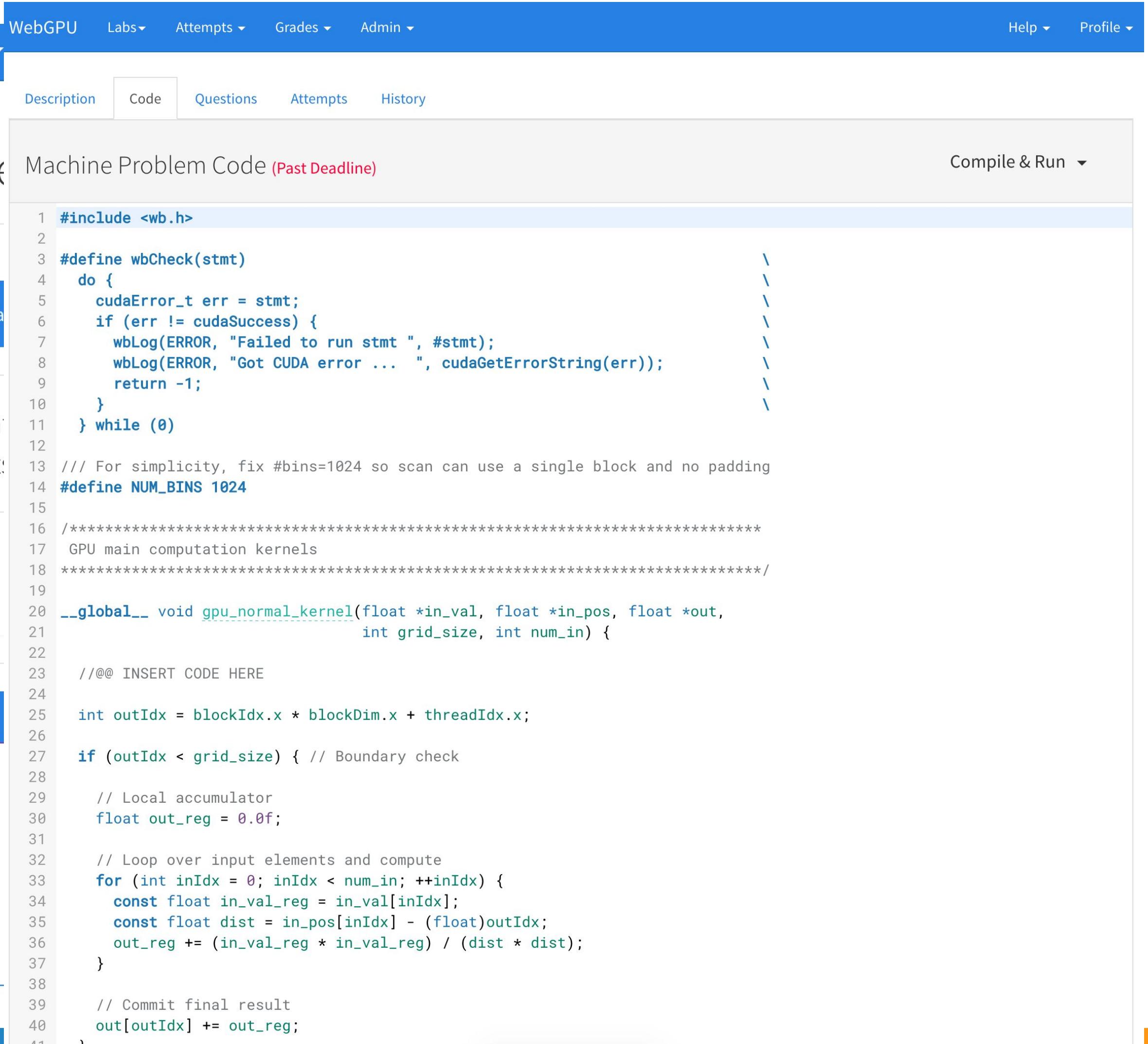
Welcome to the Algorithmic Techniques for Sca

This site serves as the developmen
students develop their assignments

Tutorial

Screencasts

WebGPU Signup
GPU Programming Web Environment



Machine Problem Code (Past Deadline)

```
#include <wb.h>

#define wbCheck(stmt)                                                           \
  do {                                                                          \
    cudaError_t err = stmt;                                                     \
    if (err != cudaSuccess) {                                                   \
      wbLog(ERROR, "Failed to run stmt ", #stmt);                               \
      wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));            \
      return -1;                                                                \
    }                                                                           \
  } while (0)

/// For simplicity, fix #bins=1024 so scan can use a single block and no padding
#define NUM_BINS 1024

/******************************************************************************
 GPU main computation kernels
*******************************************************************************/

__global__ void gpu_normal_kernel(float *in_val, float *in_pos, float *out,
                                  int grid_size, int num_in) {

  //@@ INSERT CODE HERE

  int outIdx = blockIdx.x * blockDim.x + threadIdx.x;

  if (outIdx < grid_size) { // Boundary check

    // Local accumulator
    float out_reg = 0.0f;

    // Loop over input elements and compute
    for (int inIdx = 0; inIdx < num_in; ++inIdx) {
      const float in_val_reg = in_val[inIdx];
      const float dist = in_pos[inIdx] - (float)outIdx;
      out_reg += (in_val_reg * in_val_reg) / (dist * dist);
    }

    // Commit final result
    out[outIdx] += out_reg;
```

Originally Designed for MOOC

- Around 100k students registered for Coursera's Heterogeneous Parallel Programming course
- Targeted weekly labs
- Labs auto-graded based with dataset

Students Per Offering

Intro to CUDA

Around 200 students from UIUC

Advanced CUDA

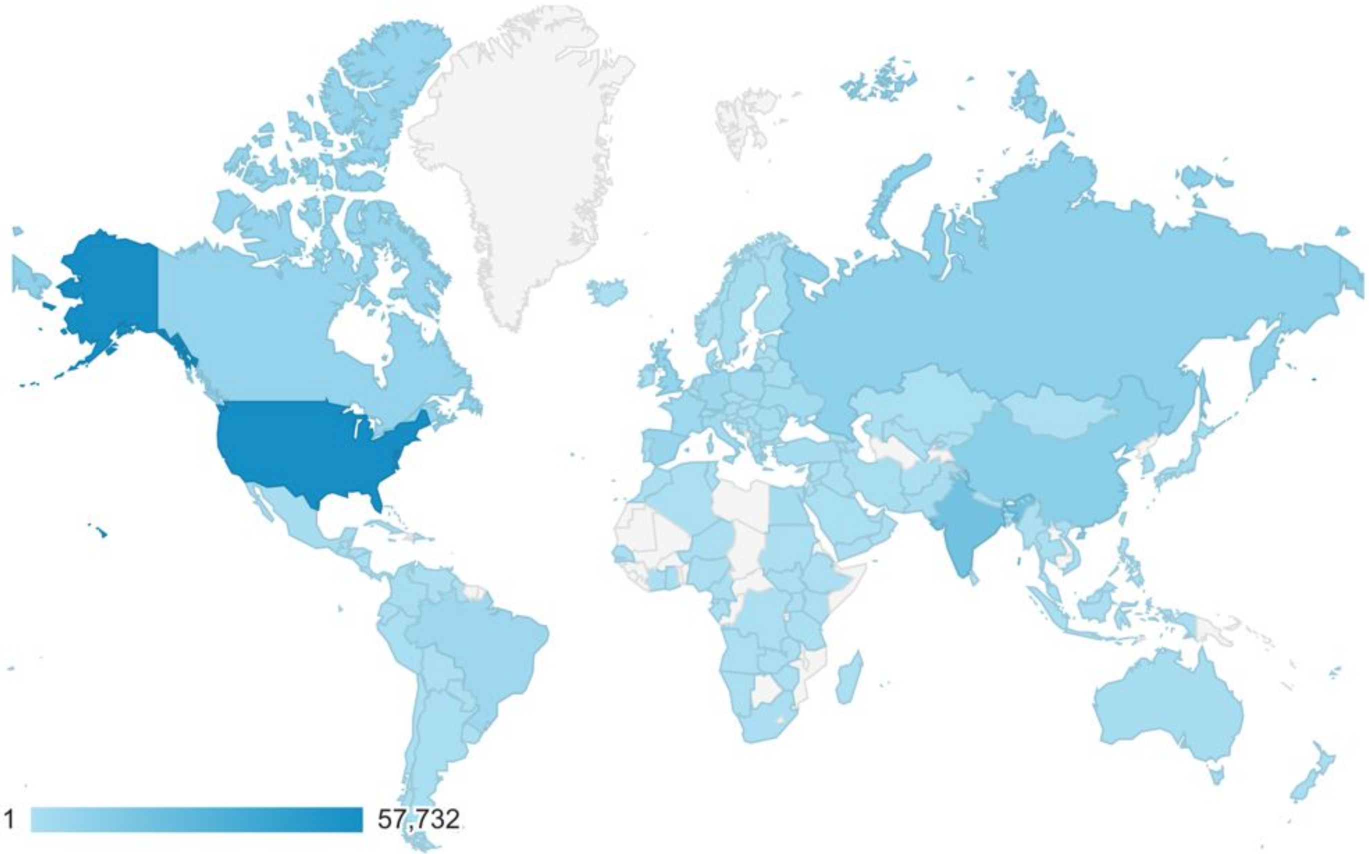
Around 100 students for UIUC and collaborating institutions

Summer School

Around 100 students from all over the world

Coursera HPP

Around 20,000 students worldwide


1
57,732

Problem

Restrictions with WebGPU

- ➔ Cannot modify programming environment
    - ◆ Build scripts / libraries / dataset / …
    - ◆ Cannot use profilers and debuggers
- ➔ User restricted within a sandboxed environment

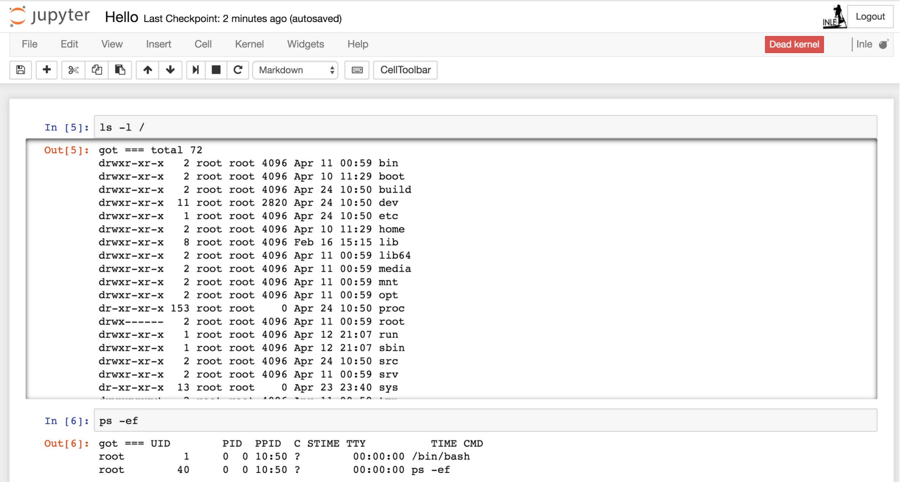
Intro and Advanced CUDA Project

- ➔ Develop a CUDA version of a CNN
- ➔ Given unoptimized sequential code
- ➔ Significant part of the total grade
- ➔ Around 4-6 weeks to complete
- ➔ Users should be "*root*"
- ➔ github.com/webgpu/ece408project
- ➔ github.com/webgpu/ece508-convlayer

Pipeline

Jupyter Notebook Interface to RAI

- ➔ Make it easy to develop interactive labs
- ➔ Built on top of Jupyter
- ➔ Implements a client/server that speaks the IPython protocol



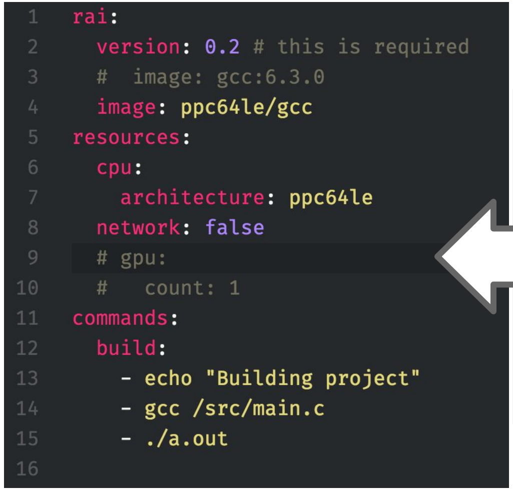
```
In [5]: ls -l /
```

```
Out[5]: got === total 72
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 bin
        drwxr-xr-x   2 root root 4096 Apr 10 11:29 boot
        drwxr-xr-x   2 root root 4096 Apr 24 10:50 build
        drwxr-xr-x  11 root root 2820 Apr 24 10:50 dev
        drwxr-xr-x   1 root root 4096 Apr 24 10:50 etc
        drwxr-xr-x   2 root root 4096 Apr 10 11:29 home
        drwxr-xr-x   8 root root 4096 Feb 16 15:15 lib
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 lib64
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 media
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 mnt
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 opt
        dr-xr-xr-x 153 root root    0 Apr 24 10:50 proc
        drwx------   2 root root 4096 Apr 11 00:59 root
        drwxr-xr-x   1 root root 4096 Apr 12 21:07 run
        drwxr-xr-x   1 root root 4096 Apr 12 21:07 sbin
        drwxr-xr-x   2 root root 4096 Apr 24 10:50 src
        drwxr-xr-x   2 root root 4096 Apr 11 00:59 srv
        dr-xr-xr-x  13 root root    0 Apr 23 23:40 sys
```

```
In [6]: ps -ef
```

```
Out[6]: got === UID        PID  PPID  C STIME TTY          TIME CMD
        root         1     0  0 10:50 ?        00:00:00 /bin/bash
        root        40     0  0 10:50 ?        00:00:00 ps -ef
```

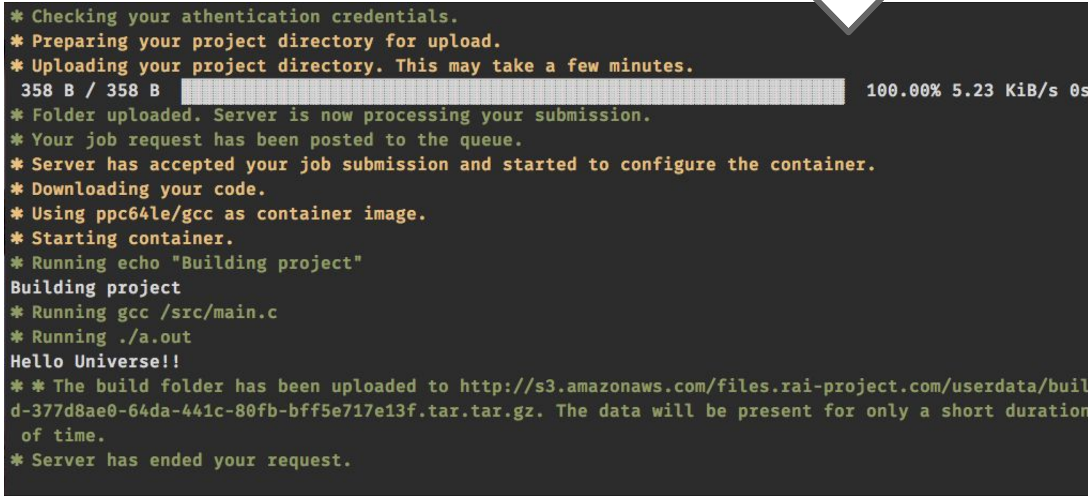
Command line Interface



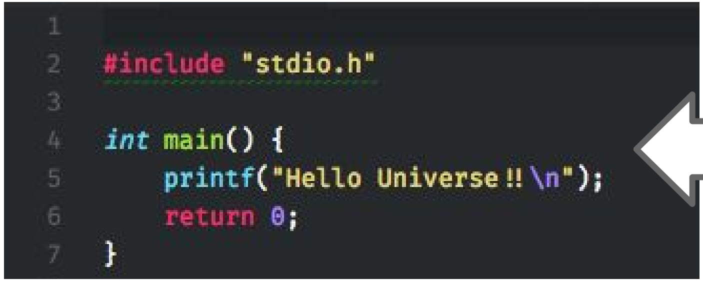
```yaml
rai:
  version: 0.2 # this is required
  #  image: gcc:6.3.0
  image: ppc64le/gcc
resources:
  cpu:
    architecture: ppc64le
  network: false
  # gpu:
  #   count: 1
commands:
  build:
    - echo "Building project"
    - gcc /src/main.c
    - ./a.out
```

Submission Spec

Output



```
✱ Checking your athentication credentials.
✱ Preparing your project directory for upload.
✱ Uploading your project directory. This may take a few minutes.
 358 B / 358 B                                                        100.00% 5.23 KiB/s 0s
✱ Folder uploaded. Server is now processing your submission.
✱ Your job request has been posted to the queue.
✱ Server has accepted your job submission and started to configure the container.
✱ Downloading your code.
✱ Using ppc64le/gcc as container image.
✱ Starting container.
✱ Running echo "Building project"
Building project
✱ Running gcc /src/main.c
✱ Running ./a.out
Hello Universe!!
✱ ✱ The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/buil
d-377d8ae0-64da-441c-80fb-bff5e717e13f.tar.tar.gz. The data will be present for only a short duration
 of time.
✱ Server has ended your request.
```



```c

#include "stdio.h"

int main() {
    printf("Hello Universe!!\n");
    return 0;
}
```

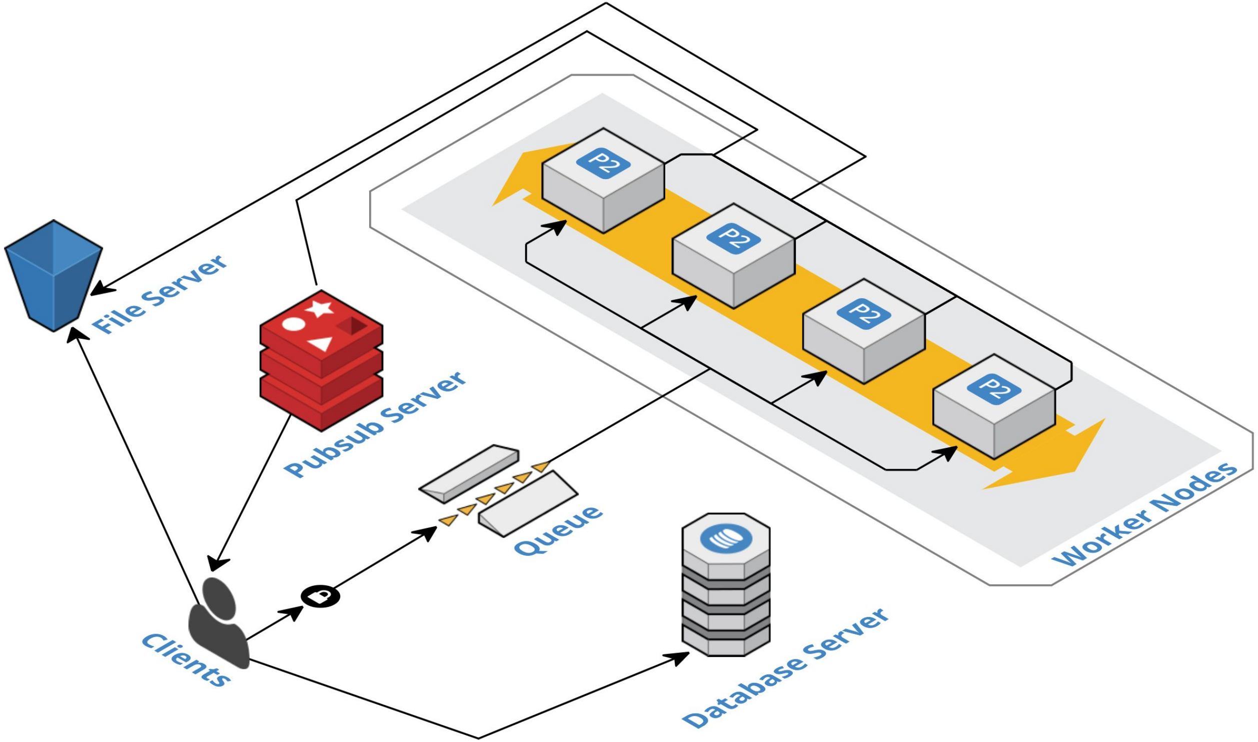
User Program

https://asciinema.org/a/6k5e96itnqu6ekbji60c3kgy4

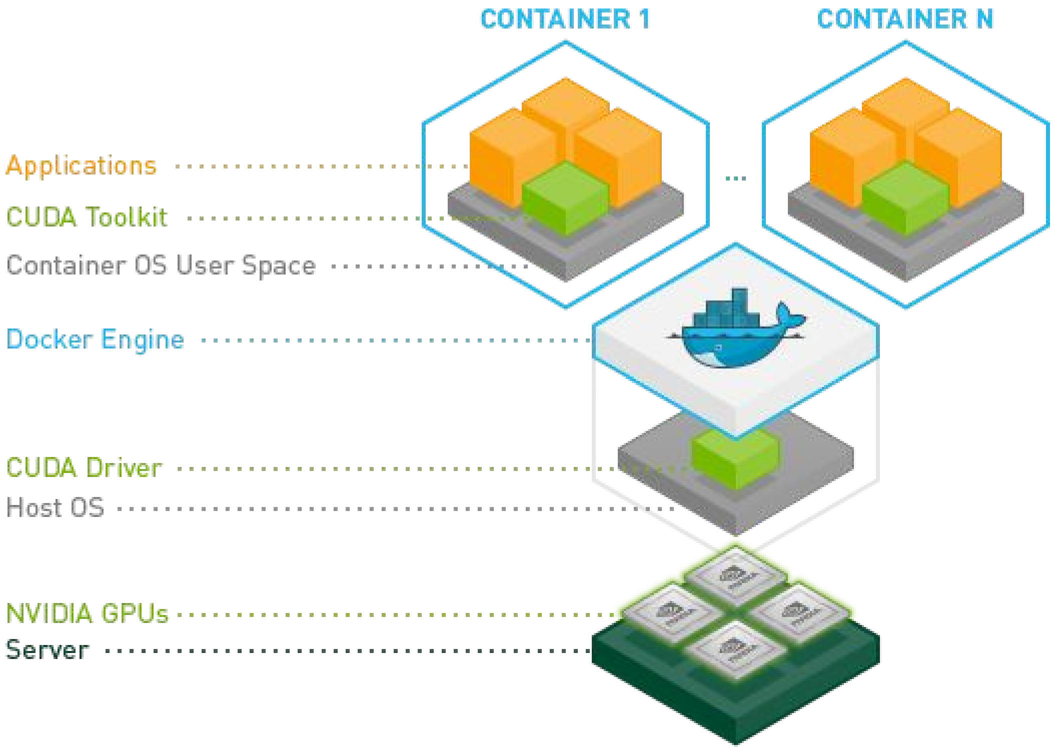
Demo

Architecture

Current Deployment Setup



Docker Layer



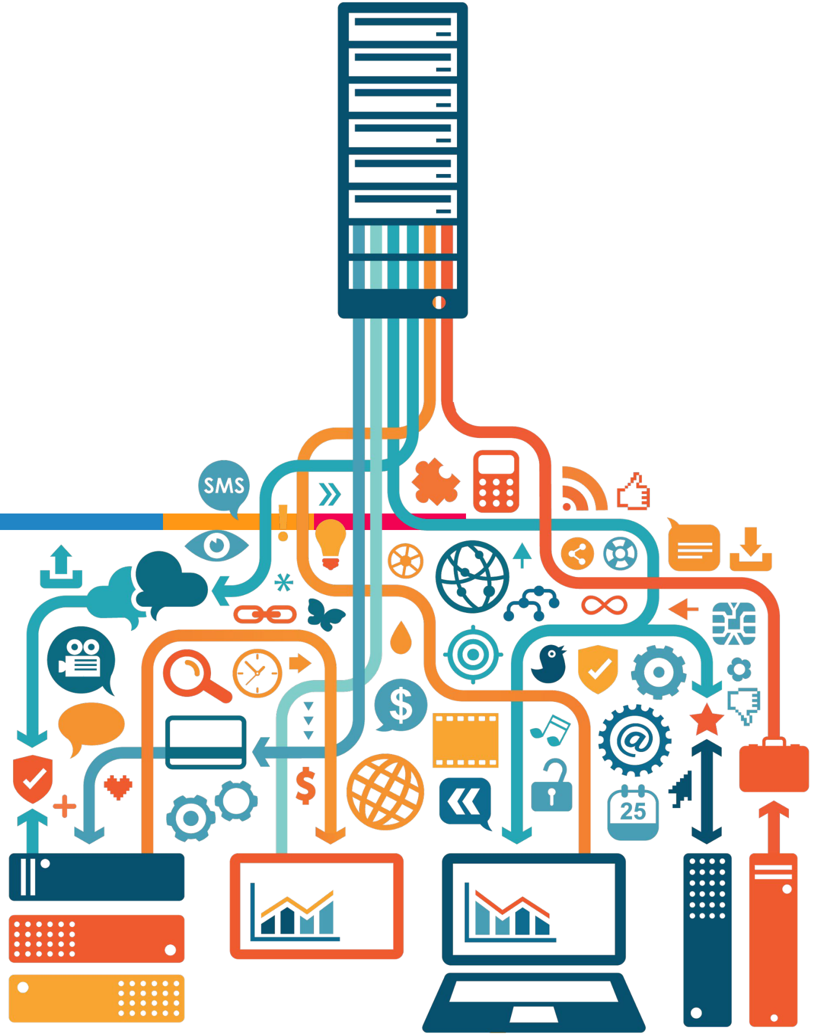
Wrote our own docker volume plugin

Not Just Project Submission

- A set of reusable components serving as a runtime
- Submission specific code is contained and small (<2KLoc)
  - Client logic is ~400 lines of code
  - Server logic is ~800 lines of code

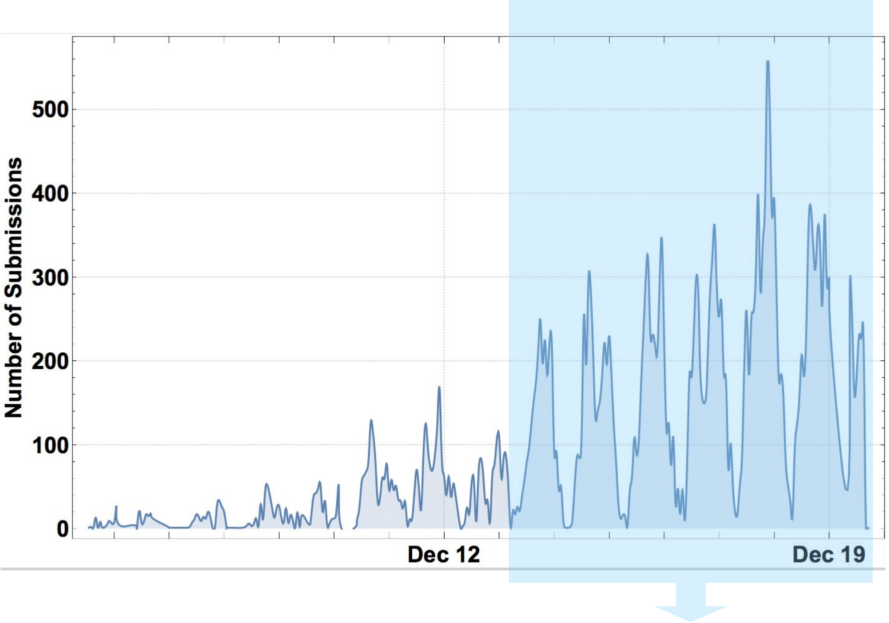
Service	Available Backends
Authentication	Secret, **Auth0**
Queue	NSQ, **SQS**, Redis, Kafka, NATS
Database	RethinkDB, MongoDB, MySQL, Postgres, SQLite, …
Registry	Etcd, Consul, BoltDB, Zookeeper
Config	**Yaml**, Toml, JSON, Environment
PubSub	EC, **Redis**, GCP, NATS, SNS
Tracing	XRay, Zipkin, StackDriver
Logger	**StackDriver**, **JournalD**, Syslog, Kinesis
Store	**S3**, Minio
Container	**Docker**
Serializer	BSON, **JSON**

IMPACT



Usage / Pedigree from Last Semester

- ➔ Around 170 students had to use the system for submission
- ➔ Students were using Linux, OSX, Windows, and WLS
- ➔ Students uploaded and generated around 100GB of data



Used 25 Workers

Currently

- Running on the 2 IBM Minsky machines
- Used by around 100 people in the 508 class (UIUC and Minnesota)
  - For the last lab
  - For open-ended projects
- Students developed their own containers solving anything from Matrix factorization (for recommender systems) to Molecular simulations

CarML

CarML - Deploy ML Artifacts w/RAI

- Make it easy to deploy ML artifacts
- Makes it possible for people to test tools / ML models without investing time in installing software dependencies and getting HW resources

Resources

GPU TEACHING KIT FOR ACCELERATED COMPUTING

Breaking the Barriers to GPU Education in Academia

Co-developed by UIUC and NVIDIA for educators

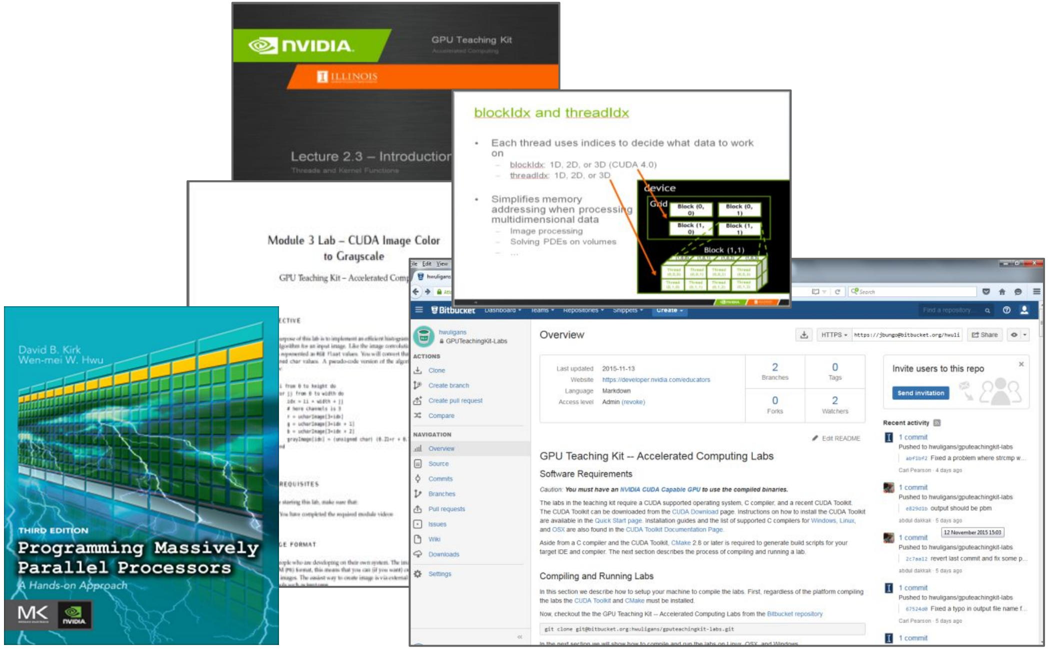
Comprehensive teaching materials

- 3^{rd} Ed. PMPP E-book by Hwu/Kirk
- Lecture slides and notes
- Lecture videos
- Hands-on labs/solutions
- Larger coding projects/solutions
- Quiz/exam questions/solution

GPU compute resources

- NVIDIA online free Qwiklab credits
- AWS credits

developer.nvidia.com/teaching-kits



blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D (CUDA 4.0)
  - threadIdx: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …

Module 3 Lab – CUDA Image Color to Grayscale

GPU Teaching Kit – Accelerated Comp

ECTIVE

REQUISITES

E FORMAT

hwuligans
GPUTeachingKit-Labs

ACTIONS
- Clone
- Create branch
- Create pull request
- Compare

NAVIGATION
- Overview
- Source
- Commits
- Branches
- Pull requests
- Issues
- Wiki
- Downloads
- Settings

Overview

HTTPS https://jbungo@bitbucket.org/hwuli

Last updated 2015-11-13
Website https://developer.nvidia.com/educators
Language Markdown
Access level Admin (revoke)

2 Branches
0 Tags
0 Forks
2 Watchers

Invite users to this repo

Send invitation

Recent activity

1 commit
Pushed to hwuligans/gputeachingkit-labs
abf1bf2 Fixed a problem where strcmp w...
Carl Pearson · 4 days ago

1 commit
Pushed to hwuligans/gputeachingkit-labs
e829d1b output should be pbm
abdul dakkak · 5 days ago

1 commit
12 November 2015 15:03
Pushed to hwuligans/gputeachingkit-labs
2c7aa12 revert last commit and fix some p...
abdul dakkak · 5 days ago

1 commit
Pushed to hwuligans/gputeachingkit-labs
67524d0 Fixed a typo in output file name f...
Carl Pearson · 5 days ago

1 commit

Edit README

GPU Teaching Kit -- Accelerated Computing Labs

Software Requirements

Caution: ***You must have an NVIDIA CUDA Capable GPU to use the compiled binaries.***

The labs in the teaching kit require a CUDA supported operating system, C compiler, and a recent CUDA Toolkit. The CUDA Toolkit can be downloaded from the CUDA Download page. Instructions on how to install the CUDA Toolkit are available in the Quick Start page. Installation guides and the list of supported C compilers for Windows, Linux, and OSX are also found in the CUDA Toolkit Documentation Page.

Aside from a C compiler and the CUDA Toolkit, CMake 2.8 or later is required to generate build scripts for your target IDE and compiler. The next section describes the process of compiling and running a lab.

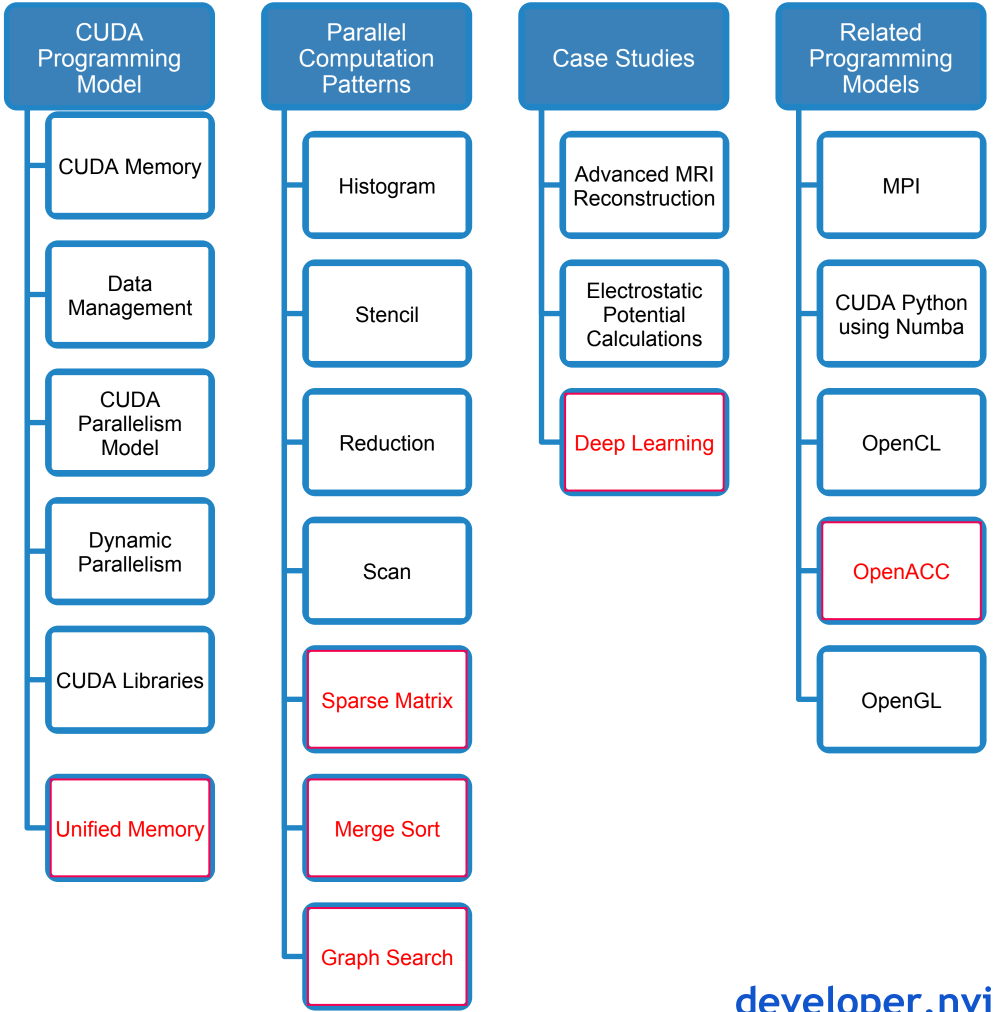
Compiling and Running Labs

In this section we describe how to setup your machine to compile the labs. First, regardless of the platform compiling the labs the CUDA Toolkit and CMake must be installed.

Now, checkout the the GPU Teaching Kit -- Accelerated Computing Labs from the Bitbucket repository

```
git clone git@bitbucket.org:hwuligans/gputeachingkit-labs.git
```

In the next section we will show how to compile and run the labs on Linux, OSX, and Windows.



- CUDA Programming Model
  - CUDA Memory
  - Data Management
  - CUDA Parallelism Model
  - Dynamic Parallelism
  - CUDA Libraries
  - Unified Memory
- Parallel Computation Patterns
  - Histogram
  - Stencil
  - Reduction
  - Scan
  - Sparse Matrix
  - Merge Sort
  - Graph Search
- Case Studies
  - Advanced MRI Reconstruction
  - Electrostatic Potential Calculations
  - Deep Learning
- Related Programming Models
  - MPI
  - CUDA Python using Numba
  - OpenCL
  - OpenACC
  - OpenGL

developer.nvidia.com/teaching-kits

Questions, Criticisms, and Concerns?

Thank you

Abdul Dakkak, Carl Pearson, Cheng Li

`{dakkak,pearson,cli99}@illinois.edu`

C3SR center for cognitive computing systems research

ILLINOIS UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

NVIDIA

The IMPACT Research Group