

GPU Computation for the Masses

Abdul Dakkak

Purpose

Cannot assume a user has a GPU/development environment

or a C compiler for that matter. These people should be able to participate

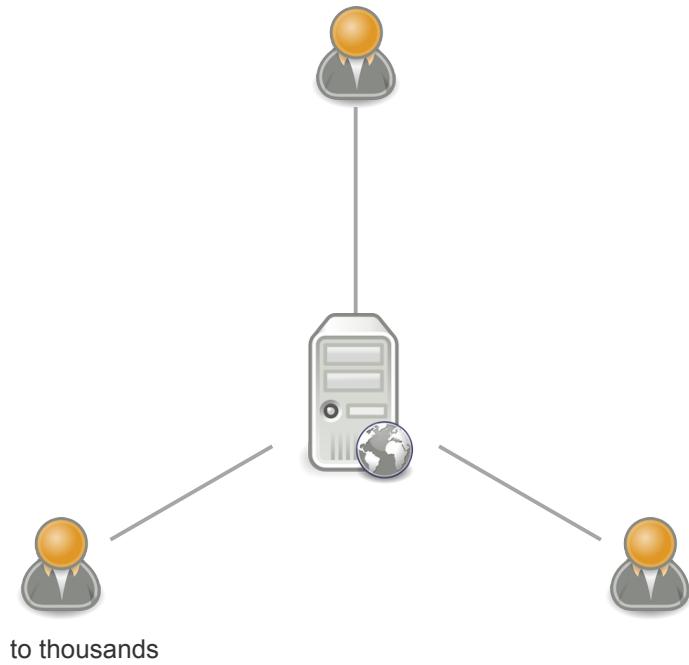


Most of us do not have a C compiler or CUDA on most of our machines, but we still might want to use them to program

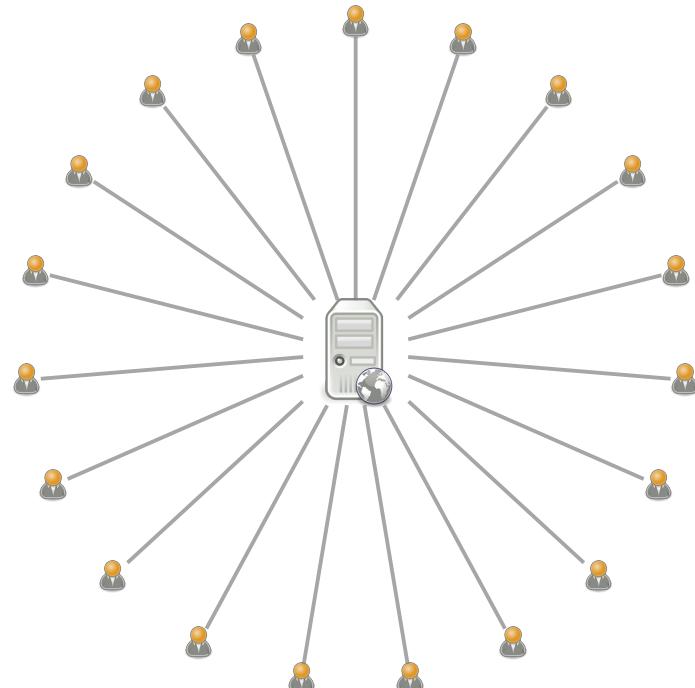


Constraint 1: Support a bazillion users

Should be able to scale from a few users



to thousands

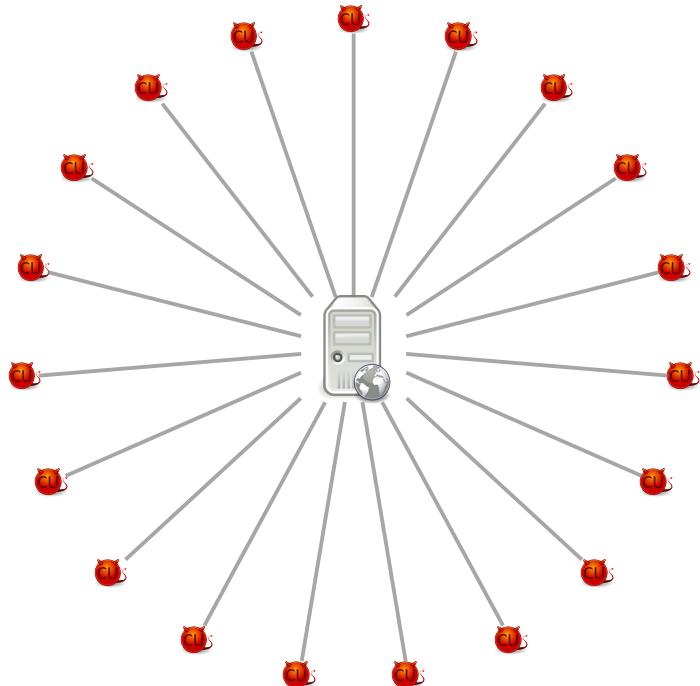


Constraint 2: Support to have a dynamic number of worker nodes

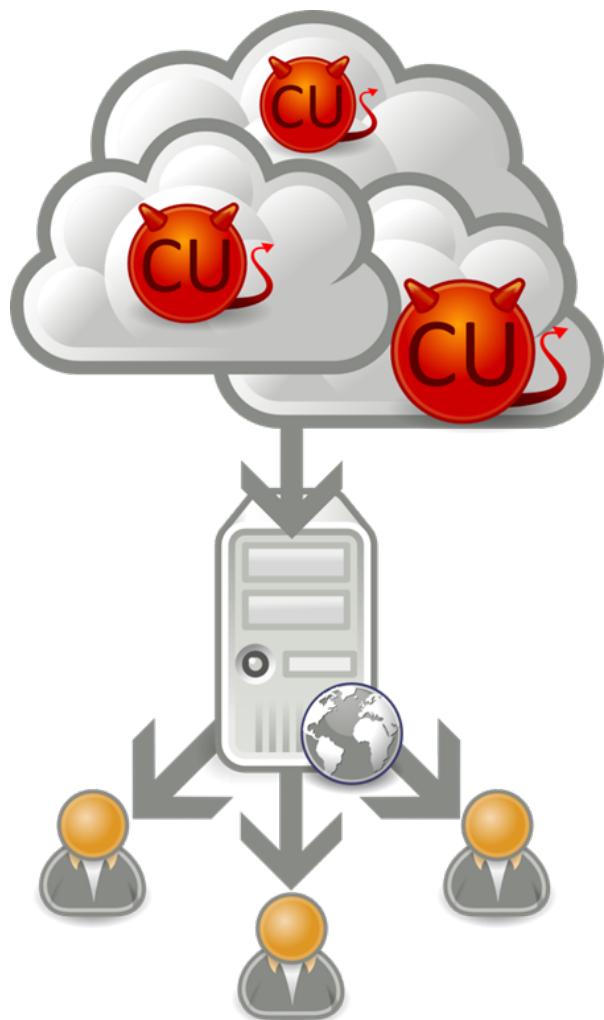
the server is supposed to support one daemon (slave worker)



or thousands

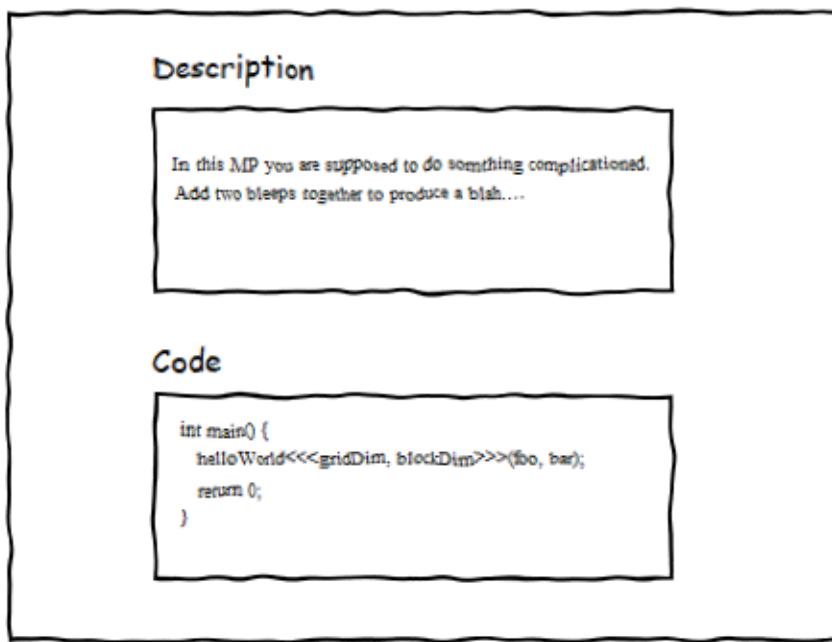


Architecture



Constraint 3: Supposed to be user friendly

The course is about learning how to program CUDA, not about how to login to a system and use ssh



Users worry about writing their code, nothing else.

Constraint 4: Supposed to be low risk for deployers

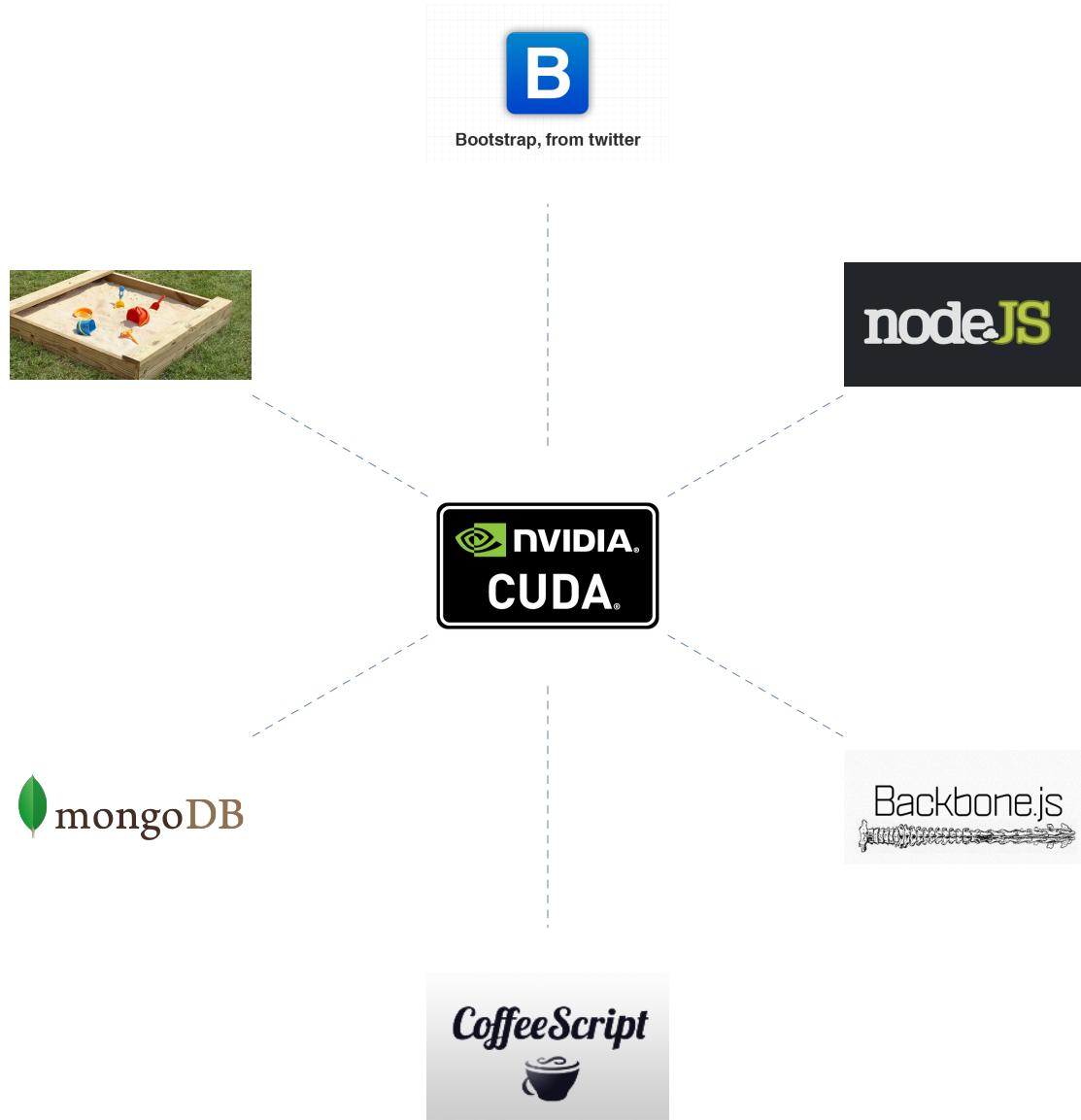
A person should not compromise the system or interfere with others' code.



Demo

Technical Details

A collection of buzz words



Bootstrap

Provides use with cross platform CSS and Javascript GUI elements for the website's frontend

Backbone.js

Provides use with a way to have MVC on the client side

CUDA

What we are teaching!!!!

NodeJS

Both the server and worker daemon are written in Node and CoffeeScript

CoffeeScript

Coffeescript is a language that compiles to Javascript

Mongodb (will be switched with mysql)

A NoSQL database (will be switched with a SQL database)

Sandbox

We use a combination of syscall captures and program analysis to make sure nothing malicious is being compiled/executed

Features

- Login system
- Grading system
- Decentralized
- Sandboxed
- Cross Platform (even on windows)
- Dynamic scheduler

Demo

GPU Computing Problems Home MP1 MP2 MP3

Problem Code Analysis

MP1: Vector Addition

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.

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.

About the Submission

write something up about how this thing works, but later.

GPU Computing Problems Home MP1 MP2 MP3

Problem Code Analysis

```

1 #include <wb.h>
2
3
4 __global void vecAdd(float * in1, float * in2, float * out, int len) {
5     //@@ Insert code to implement vector addition here
6     int index = blockIdx.x * blockDim.x;
7     if (index < len) {
8         out[index] = in1[index] + in2[index];
9     }
10 }
11
12 int main(int argc, char ** argv) {
13     wbArg_t args;
14     int inputLength;
15     float * hostInput1;
16     float * hostInput2;
17     float * deviceOutput;
18     float * deviceInput1;
19     float * deviceInput2;
20     float * deviceOutput2;
21     float * deviceOutput3;
22
23     args = wbArg_read(argc, argv);
24
25     wbTime_start(TRACE, "Importing data and creating memory on host");
26     hostInput1 = (float *) wbImport(wbArg_getInputFile(args, 0), inputLength);
27     hostInput2 = (float *) wbImport(wbArg_getInputFile(args, 1), inputLength);
28     hostOutput = (float *) malloc(inputLength * sizeof(float));
29     wbTime_stop(TRACE, "Importing data and creating memory on host");
30
31     wbLog(TRACE, "The input length is ", inputLength);
32
33     wbTime_start(GPU, "Allocating GPU memory");
34     //@@ Allocate GPU memory here
35     cudaMalloc((void **) &deviceInput1, inputLength * sizeof(float));
36     cudaMalloc((void **) &deviceInput2, inputLength * sizeof(float));
37     cudaMalloc((void **) &deviceOutput, inputLength * sizeof(float));
38     wbTime_stop(GPU, "Allocating GPU memory");
39
40     wbTime_start(GPU, "Copying input memory to the GPU");
41     //@@ Copy memory to the GPU here
42     cudaMemcpy(deviceInput1, hostInput1, inputLength * sizeof(float), cudaMemcpyHostToDevice);
43     cudaMemcpy(deviceInput2, hostInput2, inputLength * sizeof(float), cudaMemcpyHostToDevice);
44     wbTime_stop(GPU, "Copying input memory to the GPU");
45
46     //@@ Initialize the grid and block dimensions here
47     dim3 blockDim(32);
48     dim3 gridDim((ceil((float) inputLength / ((float) blockDim.x)));
49
50     wbLog(TRACE, "Block dimension is ", blockDim.x);
51     wbLog(TRACE, "Grid dimension is ", gridDim.x);
52
53     wbTime_start(Compute, "Performing CUDA computation");
54     //@@ Launch the GPU Kernel here
55     vecAdd<<<gridDim, blockDim>>>(deviceInput1, deviceInput2, deviceOutput, inputLength);
56     cudaThreadSynchronize();
57     wbTime_stop(Compute, "Performing CUDA computation");
58
59     wbTime_start(Copy, "Copying output memory to the CPU");
60     //@@ Copy the GPU memory back to the CPU here
61     cudaMemcpy(hostOutput, deviceOutput, inputLength * sizeof(float), cudaMemcpyDeviceToHost);
62     wbTime_stop(Copy, "Copying output memory to the CPU");
63
64     wbTime_start(GPU, "Freeing GPU Memory");
65     //@@ FREE THE GPU MEMORY HERE

```

Dataset 0 Submit

Demo Scenarios

Solution is correct

GPU Computing Problems																															
Home MP1 MP2 MP3																															
Problem Code Analysis																															
Timer																															
<table border="1"> <thead> <tr> <th>I Kind</th><th>I Elapsed Time (in seconds)</th><th>I Line</th><th>I Message</th></tr> </thead> <tbody> <tr> <td>Generic</td><td>0.000651813</td><td>25</td><td>Importing data and creating memory on host</td></tr> <tr> <td>GPU</td><td>0.11407421</td><td>33</td><td>Allocating GPU memory.</td></tr> <tr> <td>GPU</td><td>0.000045172</td><td>40</td><td>Copying input memory to the GPU.</td></tr> <tr> <td>Compute</td><td>0.00004458</td><td>53</td><td>Performing CUDA computation</td></tr> <tr> <td>Copy</td><td>0.000020888</td><td>59</td><td>Copying output memory to the CPU</td></tr> <tr> <td>GPU</td><td>0.000118923</td><td>64</td><td>Freeing GPU Memory</td></tr> </tbody> </table>				I Kind	I Elapsed Time (in seconds)	I Line	I Message	Generic	0.000651813	25	Importing data and creating memory on host	GPU	0.11407421	33	Allocating GPU memory.	GPU	0.000045172	40	Copying input memory to the GPU.	Compute	0.00004458	53	Performing CUDA computation	Copy	0.000020888	59	Copying output memory to the CPU	GPU	0.000118923	64	Freeing GPU Memory
I Kind	I Elapsed Time (in seconds)	I Line	I Message																												
Generic	0.000651813	25	Importing data and creating memory on host																												
GPU	0.11407421	33	Allocating GPU memory.																												
GPU	0.000045172	40	Copying input memory to the GPU.																												
Compute	0.00004458	53	Performing CUDA computation																												
Copy	0.000020888	59	Copying output memory to the CPU																												
GPU	0.000118923	64	Freeing GPU Memory																												
Logger																															
<table border="1"> <thead> <tr> <th>Level</th><th>Location</th><th>Message</th></tr> </thead> <tbody> <tr> <td>Trace</td><td>main :: 31</td><td>The input length is 100</td></tr> <tr> <td>Trace</td><td>main :: 50</td><td>Block dimension is 32</td></tr> <tr> <td>Trace</td><td>main :: 51</td><td>Grid dimension is 4</td></tr> </tbody> </table>				Level	Location	Message	Trace	main :: 31	The input length is 100	Trace	main :: 50	Block dimension is 32	Trace	main :: 51	Grid dimension is 4																
Level	Location	Message																													
Trace	main :: 31	The input length is 100																													
Trace	main :: 50	Block dimension is 32																													
Trace	main :: 51	Grid dimension is 4																													
Attempts																															
<table border="1"> <thead> <tr> <th>Dataset #</th><th>Result</th><th>Run Time</th><th>Time</th></tr> </thead> <tbody> <tr> <td>0</td><td>Solution is correct.</td><td>0.153807114 sec</td><td>less than a minute ago</td></tr> </tbody> </table>				Dataset #	Result	Run Time	Time	0	Solution is correct.	0.153807114 sec	less than a minute ago																				
Dataset #	Result	Run Time	Time																												
0	Solution is correct.	0.153807114 sec	less than a minute ago																												

Solution is incorrect

GPU Computing Problems																															
Home MP1 MP2 MP3																															
Problem Code Analysis																															
Timer																															
<table border="1"> <thead> <tr> <th>I Kind</th><th>I Elapsed Time (in seconds)</th><th>I Line</th><th>I Message</th></tr> </thead> <tbody> <tr> <td>Generic</td><td>0.000660047</td><td>25</td><td>Importing data and creating memory on host</td></tr> <tr> <td>GPU</td><td>0.09123867</td><td>33</td><td>Allocating GPU memory.</td></tr> <tr> <td>GPU</td><td>0.000022701</td><td>40</td><td>Copying input memory to the GPU.</td></tr> <tr> <td>Compute</td><td>0.000033956</td><td>53</td><td>Performing CUDA computation</td></tr> <tr> <td>Copy</td><td>0.000015796</td><td>59</td><td>Copying output memory to the CPU</td></tr> <tr> <td>GPU</td><td>0.000076288</td><td>64</td><td>Freeing GPU Memory</td></tr> </tbody> </table>				I Kind	I Elapsed Time (in seconds)	I Line	I Message	Generic	0.000660047	25	Importing data and creating memory on host	GPU	0.09123867	33	Allocating GPU memory.	GPU	0.000022701	40	Copying input memory to the GPU.	Compute	0.000033956	53	Performing CUDA computation	Copy	0.000015796	59	Copying output memory to the CPU	GPU	0.000076288	64	Freeing GPU Memory
I Kind	I Elapsed Time (in seconds)	I Line	I Message																												
Generic	0.000660047	25	Importing data and creating memory on host																												
GPU	0.09123867	33	Allocating GPU memory.																												
GPU	0.000022701	40	Copying input memory to the GPU.																												
Compute	0.000033956	53	Performing CUDA computation																												
Copy	0.000015796	59	Copying output memory to the CPU																												
GPU	0.000076288	64	Freeing GPU Memory																												
Logger																															
<table border="1"> <thead> <tr> <th>Level</th><th>Location</th><th>Message</th></tr> </thead> <tbody> <tr> <td>Trace</td><td>main :: 31</td><td>The input length is 100</td></tr> <tr> <td>Trace</td><td>main :: 50</td><td>Block dimension is 32</td></tr> <tr> <td>Trace</td><td>main :: 51</td><td>Grid dimension is 4</td></tr> </tbody> </table>				Level	Location	Message	Trace	main :: 31	The input length is 100	Trace	main :: 50	Block dimension is 32	Trace	main :: 51	Grid dimension is 4																
Level	Location	Message																													
Trace	main :: 31	The input length is 100																													
Trace	main :: 50	Block dimension is 32																													
Trace	main :: 51	Grid dimension is 4																													
Attempts																															
<table border="1"> <thead> <tr> <th>Dataset #</th><th>Result</th><th>Run Time</th><th>Time</th></tr> </thead> <tbody> <tr> <td>0</td><td>The solution did not match the expected results at column 0 and row 0. Expecting 488.543 but got 935.796.</td><td>0.141575066 sec</td><td>less than a minute ago</td></tr> <tr> <td>0</td><td>Solution is correct.</td><td>0.153807114 sec</td><td>less than a minute ago</td></tr> </tbody> </table>				Dataset #	Result	Run Time	Time	0	The solution did not match the expected results at column 0 and row 0. Expecting 488.543 but got 935.796.	0.141575066 sec	less than a minute ago	0	Solution is correct.	0.153807114 sec	less than a minute ago																
Dataset #	Result	Run Time	Time																												
0	The solution did not match the expected results at column 0 and row 0. Expecting 488.543 but got 935.796.	0.141575066 sec	less than a minute ago																												
0	Solution is correct.	0.153807114 sec	less than a minute ago																												

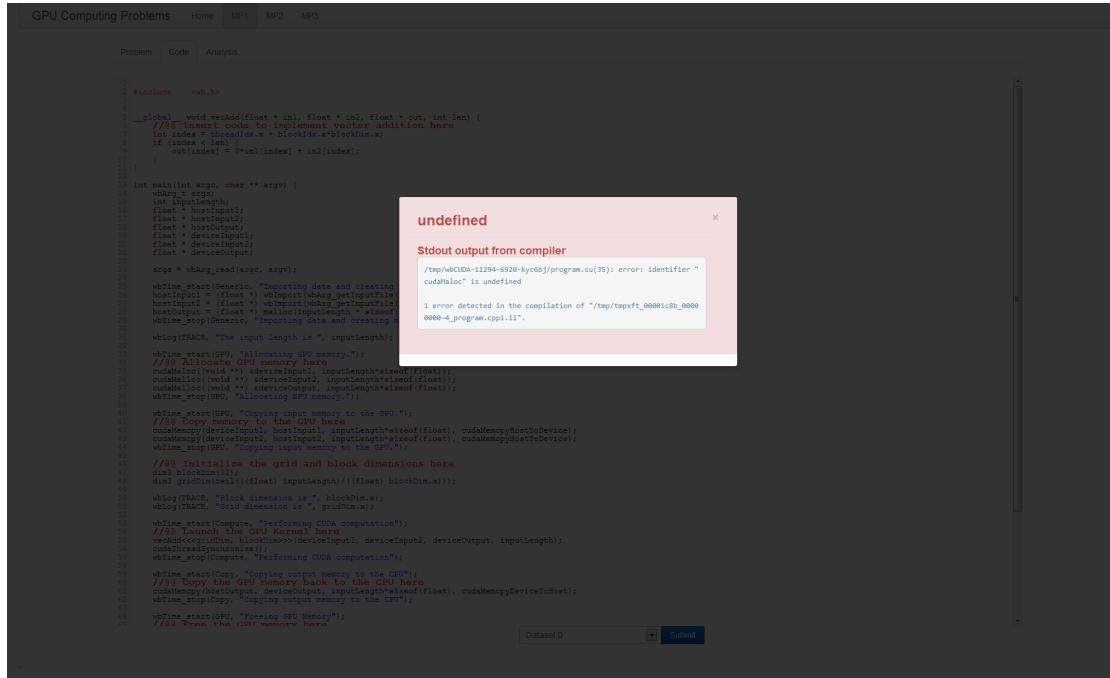
Logging and debugging

Logger		
Level	Location	Message
Trace	main:: 26	Read arguments 9
Trace	main:: 35	The input length is 100
Trace	main:: 54	Block dimension is 32
Trace	main:: 55	Grid dimension is 4

Timing

Timer			
Kind	Elapsed Time (in seconds)	Line	Message
Generic	0.000634308	29	Importing data and creating memory on host
GPU	0.099005559	37	Allocating GPU memory.
GPU	0.00002367	44	Copying input memory to the GPU.
Compute	0.000033975	57	Performing CUDA computation
Copy	0.000016072	63	Copying output memory to the CPU
GPU	0.000076214	68	Freeing GPU Memory

Invalid Syntax



Infinite loop in host code

```
21     wbSetOutput,
22
23     args = wbArg_read(argc, argv);
24
25     while(true) {} //@@ an infinite loop!!!!
26
27     wbTime_start(Generic, "Importing data and creating");
28     hostInput1 = (float *) wbImport(wbArg_getInputFile);
29     hostInput2 = (float *) wbImport(wbArg_getInputFile);
```

The screenshot shows a code editor window titled "GPU Computing Problems" with tabs for "Problem", "Code", and "Analysis". The "Code" tab is selected, displaying the following C code:

```

1 #include <cuda.h>
2
3 //@@ Insert code to implement vector addition here
4
5 __global__ void vecAdd(float * in1, float * in2, float * out, int len) {
6     int index = threadIdx.x + blockIdx.x*blockDim.x;
7     if (index < len)
8         out[index] = in1[index] + in2[index];
9 }
10
11
12 int main(int argc, char ** argv) {
13     wbArg_t args;
14     int inputLength;
15     float * deviceInput1;
16     float * hostInput1;
17     float * deviceInput2;
18     float * hostInput2;
19     float * deviceOutput;
20     float * hostOutput;
21     float * deviceTemp;
22
23     args = wbArg_read(args, argv);
24
25     while(true) {} //@@ an infinite loop!!!!
26
27     wbTime_start(&args, "Importing data and creating memory");
28     hostInput1 = (float *)wbImport(wbArg_getInputFile(args, 0), inputLength);
29     hostInput2 = (float *)wbImport(wbArg_getInputFile(args, 1), inputLength);
30     hostOutput = (float *)malloc(inputLength * sizeof(float));
31     wbTime_stop(&args, "Importing data and creating memory on host");
32
33     wbLog(TRACE, "The input length is ", inputLength);
34
35     wbTime_start(&args, "Allocating GPU memory");
36     cudaMalloc(&deviceInput1, inputLength*sizeof(float));
37     cudaMalloc(&deviceInput2, inputLength*sizeof(float));
38     cudaMalloc(&deviceOutput, inputLength*sizeof(float));
39     wbTime_stop(&args, "Allocating GPU memory");
40
41     wbTime_start(&args, "Copying input memory to the GPU");
42     cudaMemcpy(deviceInput1, hostInput1, inputLength*sizeof(float), cudaMemcpyHostToDevice);
43     cudaMemcpy(deviceInput2, hostInput2, inputLength*sizeof(float), cudaMemcpyHostToDevice);
44     wbTime_stop(&args, "Copying input memory to the GPU");
45
46     //@@ Initialize the grid and block dimensions here
47     dim3 blockDim(32);
48     dim3 gridDim((float) inputLength/(float) blockDim.x));
49
50     wbLog(TRACE, "Block dimension is ", blockDim.x);
51     wbLog(TRACE, "Grid dimension is ", gridDim.x);
52
53     wbTime_start(&args, "Performing CUDA computation");
54     //@@ Call the GPU Kernel here
55     vecAdd<<<(gridDim, blockDim)(deviceInput1, deviceInput2, deviceOutput, inputLength);
56     wbTime_stop(&args, "Performing CUDA computation");
57
58     wbTime_start(&args, "Copying output memory to the CPU");
59     cudaMemcpy(hostOutput, deviceOutput, inputLength*sizeof(float), cudaMemcpyDeviceToHost);
60     wbTime_stop(&args, "Copying output memory to the CPU");
61
62     //@@ Copy the GPU memory back to the CPU here
63     cudaMemcpy(hostOutput, deviceOutput, inputLength*sizeof(float), cudaMemcpyDeviceToHost);
64
65     wbTime_stop(&args, "Copying output memory to the CPU");
66
67 }

```

A red box highlights the line `while(true) {}` with the annotation "Program was terminated based on timeout policy." A status bar at the bottom shows "Dataset 0" and "submit".

Infinite loop in GPU code

```

5     __global__ void vecAdd(float * in1, float * in2, float *
6         //@@ Insert code to implement vector addit
7         int index = threadIdx.x + blockIdx.x*blockDim.x;
8
9         while(true) {} //@@ an infinite loop!!!!
10
11         if (index < len) {
12             out[index] = in1[index] + in2[index];
13         }
14     }

```

```

1 #include "wbd.h"
2
3 // Global void vecAdd(float * in1, float * in2, float * out, int len) {
4 //@@ Insert code to implement vector addition here
5 int index = threadIdx.x + blockDim.x*blockIdx.y;
6
7 while(true) {} //@@ an infinite loop!!!!
8
9 if (index < len) {
10     out[index] = in1[index] + in2[index];
11 }
12 }
13
14 int main(int argc, char ** argv) {
15     wbArg_t args;
16
17     wbImport(&args);
18
19     float * hostInput1;
20     float * hostInput2;
21     float * hostOutput;
22     float * deviceInput1;
23     float * deviceInput2;
24     float * deviceOutput;
25
26     args = wbdg_read(argc, argv);
27
28     wbTime_start(Generic, "Importing data and creating memory on host");
29     hostInput1 = (float *) wbImport(wbArg_getInputFile(&args, 0), inputLength);
30     hostInput2 = (float *) wbImport(wbArg_getInputFile(&args, 1), inputLength);
31     hostOutput = (float *) wbAlloc(sizeof(float)*len);
32     wbTime_stop(Generic, "Importing data and creating memory on host");
33
34     wbLog(TRACE, "The input length is ", inputLength);
35
36     wbTime_start(GPU, "Allocating GPU memory");
37     //@@ Allocate GPU memory here
38     cudaMemAlloc(&deviceInput1, inputLength*sizeof(float));
39     cudaMemAlloc(&deviceInput2, inputLength*sizeof(float));
40     cudaMemAlloc(&deviceOutput, inputLength*sizeof(float));
41     wbTime_stop(GPU, "Allocating GPU memory");
42
43     wbTime_start(GPU, "Copying input memory to the GPU");
44     //@@ Copy input memory to the GPU here
45     cudaMemcpy(deviceInput1, hostInput1, inputLength*sizeof(float), cudaMemcpyHostToDevice);
46     cudaMemcpy(deviceInput2, hostInput2, inputLength*sizeof(float), cudaMemcpyHostToDevice);
47     wbTime_stop(GPU, "Copying input memory to the GPU");
48
49     wbLog(TRACE, "Performing CUDA computation");
50
51     //@@ Launch the GPU Kernel here
52     wbdgLaunchKernel(wbdgCompute, deviceInput1, deviceInput2, deviceOutput, inputLength);
53     cudaThreadSynchronize();
54
55     wbTime_stop(Compute, "Performing CUDA computation");
56
57     wbTime_start(Copy, "Copying output memory to the CPU");
58     //@@ Copy the GPU memory back to the CPU here
59     cudaMemcpy(hostOutput, deviceOutput, inputLength*sizeof(float), cudaMemcpyDeviceToHost);
60     wbTime_stop(Copy, "Copying output memory to the CPU");
61
62     wbTime_start(CleanUp, "Freeing memory");
63     //@@ Free the GPU memory here
64     cudaFree(deviceInput1);
65     cudaFree(deviceInput2);
66
67 }

```

Program was terminated based on timeout policy.

Allocating too much memory in host code

```

25
26     void * mem = malloc(10000000000);
27
28
29     wbTime_start(Generic, "Importing dat
hostInput1 = (float *) wbImport(wbAr

```

```

1 //@@ INSERT CODE TO IMPLEMENT VECTOR ADDITION HERE
2 int index = threadIdx.x + blockDim.x*blockIdx.y;
3
4 if (index < len) {
5     out[index] = in1[index] + in2[index];
6 }
7
8
9 int main(int argc, char ** argv) {
10     wbArg_t args;
11
12     int inputLength;
13     float * hostInput1;
14     float * hostInput2;
15     float * hostOutput;
16     float * deviceInput1;
17     float * deviceInput2;
18     float * deviceOutput;
19
20     args = wbdg_read(argc, argv);
21
22     void * mem = malloc(10000000000);
23
24     wbTime_start(Generic, "Importing data and creating
hostInput1 = (float *) wbImport(wbArg_getInputFile(
25     hostInput2 = (float *) wbImport(wbArg_getInputFile(
26     hostOutput = (float *) wbAlloc(sizeof(float)*
27     wbTime_stop(Generic, "Importing data and creating
28
29     wbLog(TRACE, "The input length is ", inputLength);
30
31     wbTime_start(GPU, "Allocating GPU memory");
32     //@@ Allocate GPU memory here
33     cudaMemAlloc(&deviceInput1, inputLength*sizeof(float));
34     cudaMemAlloc(&deviceInput2, inputLength*sizeof(float));
35     cudaMemAlloc(&deviceOutput, inputLength*sizeof(float));
36     wbTime_stop(GPU, "Allocating GPU memory");
37
38     wbTime_start(GPU, "Copying input memory to the GPU");
39     //@@ Copy memory to the GPU here
40     cudaMemcpy(deviceInput1, hostInput1, inputLength*sizeof(float), cudaMemcpyHostToDevice);
41     cudaMemcpy(deviceInput2, hostInput2, inputLength*sizeof(float), cudaMemcpyHostToDevice);
42     wbTime_stop(GPU, "Copying input memory to the GPU");
43
44     //@@ Initialize the grid and block dimensions here
45     dim3 gridDim((float) inputLength/(float) blockDim.x);
46
47     wbLog(TRACE, "Block dimension is ", blockDim.x);
48     wbLog(TRACE, "Grid dimension is ", gridDim.x);
49
50     wbTime_start(Compute, "Performing CUDA computation");
51
52     //@@ Launch the GPU Kernel here
53     wbdgLaunchKernel(wbdgCompute, deviceInput1, deviceInput2, deviceOutput, inputLength);
54     cudaThreadSynchronize();
55
56     wbTime_stop(Compute, "Performing CUDA computation");
57
58     wbTime_start(Copy, "Copying output memory to the CPU");
59     //@@ Copy the GPU memory back to the CPU here
60     cudaMemcpy(hostOutput, deviceOutput, inputLength*sizeof(float), cudaMemcpyDeviceToHost);
61     wbTime_stop(Copy, "Copying output memory to the CPU");
62
63     wbTime_start(CleanUp, "Freeing GPU memory");
64     //@@ Free the GPU memory here
65     cudaFree(deviceInput1);
66     cudaFree(deviceInput2);
67
68 }

```

Program was terminated based on memory allocation policy.

Restrict user from adding includes

Restricted syscall in host code

uses a combination of textual filtering and seccomp (linux kernel syscall filtering)

```
22  
23  
24     fopen("/etc/passwd", "r");  
25  
26     args = wbArg_read(argc, argv);
```

The screenshot shows a software interface for GPU computing problems. At the top, there are tabs for 'Problems', 'Code', and 'Analysis'. The 'Code' tab is selected, displaying a C++ code snippet. A modal dialog box titled 'Invalid operation in sandbox.' is open, stating 'The following keyword specified is not allowed in sandbox mode: fopen'. The code in the editor includes several CUDA-related functions like `wbImport`, `wbLog`, and `wbTime`, along with standard C++ code for file I/O and memory management.

```

1 #include <stdio.h>
2
3 #include "wbs.h"
4
5 //@@ Insert code to implement vector addition here
6 int index = blockIdx.x*blockDim.x;
7 if (index < len)
8     outIndex = in1[index] + in2[index];
9
10
11
12
13
14 int main(int argc, char ** argv) {
15     wbArg_t args;
16     int inputLength;
17     float * hostInput1;
18     float * hostInput2;
19     float * deviceInput1;
20     float * deviceInput2;
21     float * deviceOutput;
22
23     args.wbArgInit();
24
25     fopen("/etc/passwd", "r");
26
27     args = wbArg_read(argc, argv);
28
29     wbTime_start(Generic, "Importing data and creating hostInput1");
30     hostInput1 = (float *) wbImport(wbArg_getInputFile);
31     wbTime_stop(Generic, "Importing data and creating hostInput1");
32     wbLog(TRACE, "The input length is ", inputLength);
33
34     wbTime_start(GPU, "Allocating GPU memory");
35     //@@ Allocate GPU memory here
36     cudaMemAlloc((void **) &deviceInput1, inputLength*sizeof(float));
37     cudaMemAlloc((void **) &deviceInput2, inputLength*sizeof(float));
38     cudaMemAlloc((void **) &deviceOutput, inputLength*sizeof(float));
39     wbTime_stop(GPU, "Allocating GPU memory");
40
41     wbTime_start(GPU, "Copying input memory to the GPU");
42     //@@ Copy memory to the GPU here
43     cudaMemcpy(deviceInput1, hostInput1, inputLength*sizeof(float), cudaMemcpyHostToDevice);
44     cudaMemcpy(deviceInput2, hostInput2, inputLength*sizeof(float), cudaMemcpyHostToDevice);
45     wbTime_stop(GPU, "Copying input memory to the GPU");
46
47     //@@ Initialize the grid and block dimensions here
48     dim3 blockDim(32);
49     dim3 grid((inputLength/(float) inputLength)/(float) blockDim.x));
50
51     wbLog(TRACE, "Block dimension is ", blockDim.x);
52     wbLog(TRACE, "Grid dimension is ", grid.dim.x);
53
54     wbTime_start(CUDA, "Performing CUDA computation");
55     //@@ Launch the GPU Kernel here
56     //@@ kernelName<<>>(deviceInput1, deviceInput2, deviceOutput, inputLength);
57     cudaThreadSynchronize();
58     wbTime_stop(CUDA, "Performing CUDA computation");
59
60     wbTime_start(Copy, "Copying output memory to the CPU");
61     //@@ Copy the GPU memory back to the CPU here
62     cudaMemcpy(hostOutput, deviceOutput, inputLength*sizeof(float), cudaMemcpyDeviceToHost);
63     wbTime_stop(Copy, "Copying output memory to the CPU");
64
65     wbArg_free(&args);
66 }

```

See previous attempts

Attempts			
Dataset #	Result	Run Time	Time
2	Solution is correct.	0.201906503 sec	less than a minute ago
1	Solution is correct.	0.142507239 sec	less than a minute ago
0	Solution is correct.	0.141190539 sec	less than a minute ago
	Failed to determine solution	0 sec	less than a minute ago
	Failed to determine solution	0 sec	about a minute ago
	Failed to determine solution	0 sec	2 minutes ago
	Compilation failed	0 sec	2 minutes ago
	Failed to determine solution	0 sec	3 minutes ago
	Failed to determine solution	0 sec	4 minutes ago
	Failed to determine solution	0 sec	6 minutes ago
	Failed to determine solution	0 sec	6 minutes ago
	Failed to determine solution	0 sec	6 minutes ago

Applications

Teaching GPU Programming

Remote GPU evaluation from non-accelerated devices

Batch processing of GPU jobs

Code Breakdown

- ~5kLOC for C support code (timer/logger/solution verifier/...)
- ~1.5kLOC of coffeescript for the daemon
- ~1.5kLOC of coffeescript for the server
- ~1kLOC of javascript for the interface

Extending

Model for running native code in restricted environment

Questions/Discussion