# GPU-CPU Benchmarking and Analysis: Case Study Using Deep Learning Libraries

Clayton Turner
CSIS 612 - Advanced Computer Architecture
Fall 2015
Code Available at: https://github.com/ClaytonTurner/GPU-Benchmarking*

*References will be made to files here throughout the presentation

# Outline

- Theano
  - Overview/Installation
  - Example
- Caffe
  - Overview/Installation
  - Example
- Torch
  - Overview/Installation
  - Example

Hardware/Specifications

Deep Learning Overview

Speedups by Comparison

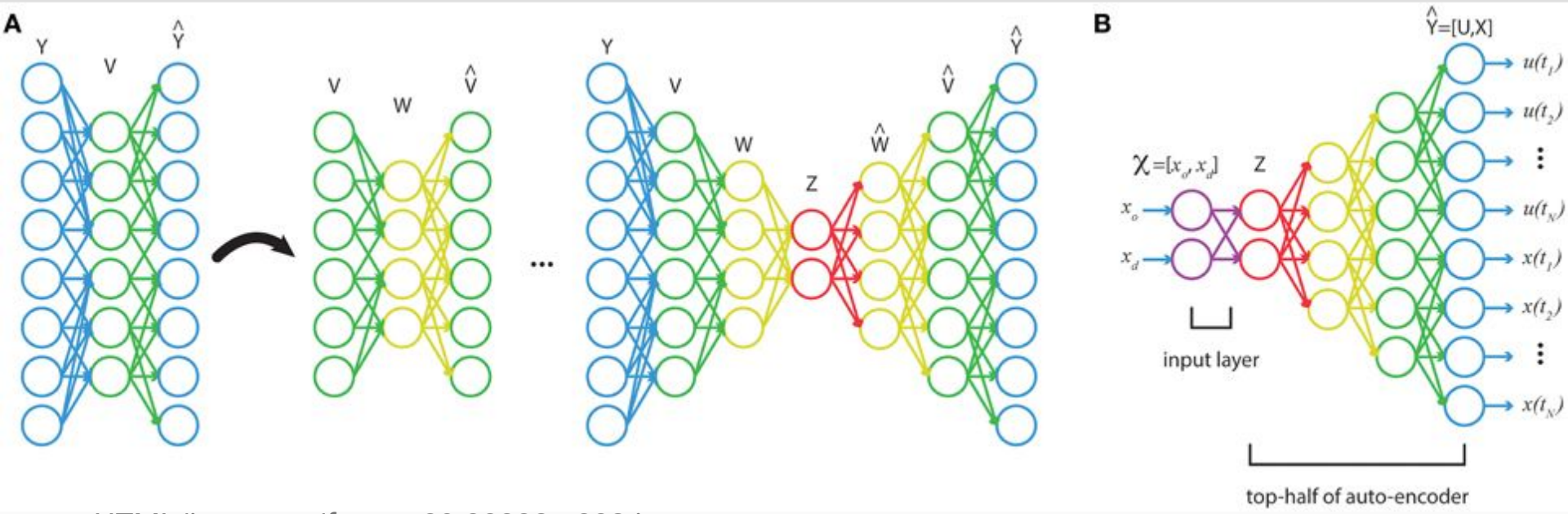
Demos

# Hardware

GPU: NVIDIA GeForce GTX 550 Ti
- 192 CUDA Cores
- Graphics Clock: 900 MHz
- Processor Cock: 1800 MHz

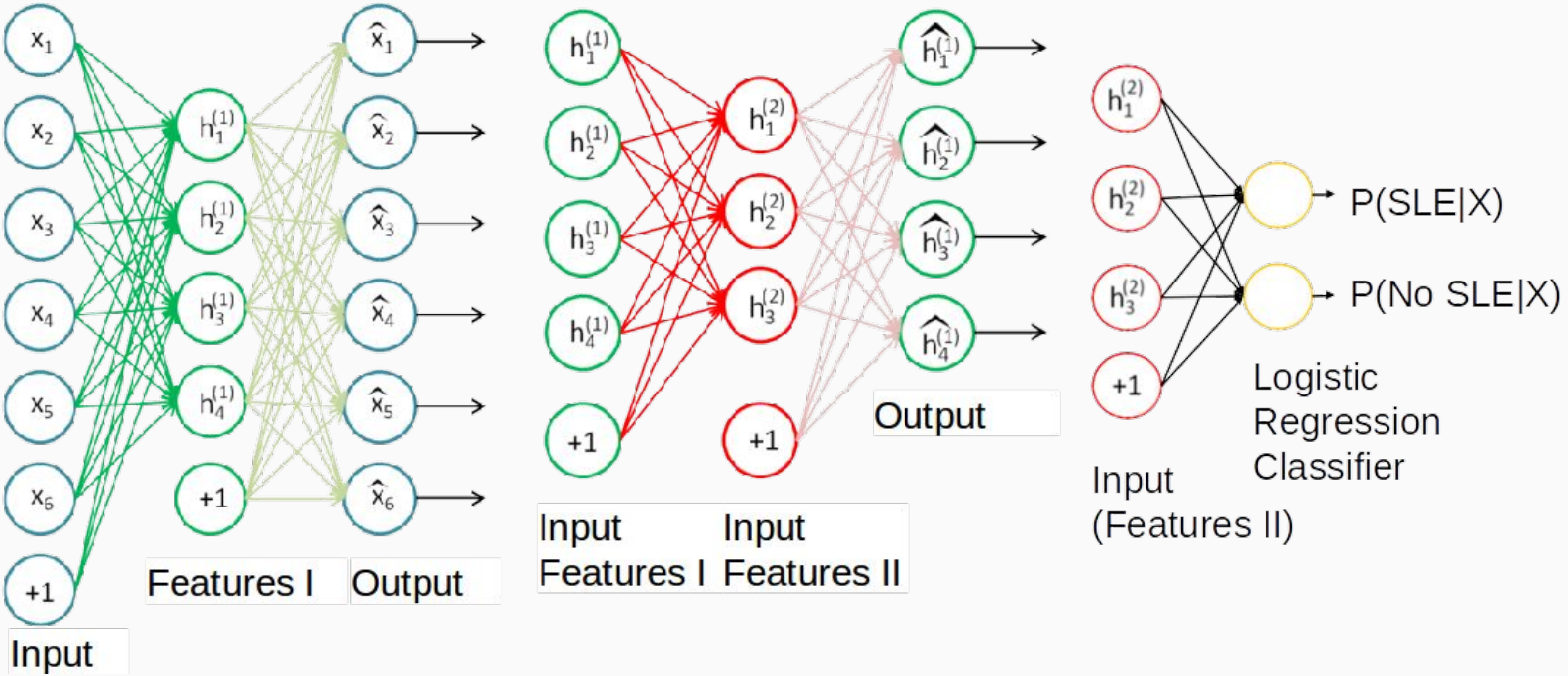
CPU: AMD FX™-8150 Eight-Core Processor (3.6 GHz)

# What Is Deep Learning? (SdA as use case)




HTML/image_m/fncom-09-00032-g002.jpg

# What Is Deep Learning? (SdA as use case)



# Theano: Overview

Python Deep Learning Library
Optimizes and evaluates user-defined multi-dimensional operations

Available through Git and PyPI (pip)

- https://github.com/Theano/Theano
- http://www.deeplearning.net/software/theano

# Theano: Features

**Transparent Use of GPU - up to 140x speedup over a CPU**
Tightly integrated with NumPy
Efficient symbolic differentiation
Speed and stability optimizations - get the right answer for log(1+x) even when x is really tiny
**Dynamic C code generation**
Built-in unit-testing and self-verification

http://www.deeplearning.net/software/theano

# Theano: Execution Modes

Runnable Modes

- FAST_COMPILE ~ Python implementations where available with cheap graph transformations
- FAST_RUN ~ C implementations where available with all transformations
- DebugMode ~ Both implementations available with all transformations
- MonitorMode ~ Debug mode with step-through (like pdb)

Example call using my file:

- THEANO_FLAGS=mode=FAST_RUN,device=gpu,floatX=float32 python theano_exponentiate.py

# Theano: Notes

Unused Optimizations

- cuDNN <- NVIDIA library used by deep networks to optimize a subset of operations (such as a convolution)
- CNMeM <- NVIDIA Deep Learning CUDA optimized memory management
  Utilizes CUDA toolkit, pthreads/native Windows, and C++

# Theano: Code

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP... | caturner3@caturner3-desktop: ~/GP...

```
1  # Adapted from deeplearning.net
2
3  from theano import function, config, shared, sandbox
4  import theano.tensor as T
5  import numpy
6  from timeit import default_timer as timer # Since we need to use Python 2.7, this is a good alt
   ernative to Perfcounter
7
8  vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
9  iters = 1000000
10
11 # Keep consistent results with random seed
12 rng = numpy.random.RandomState(42)
13
14 # Use Theano's shared variables in order to distribute computation to the GPU
15 # floatX == float32 must hold True for GPU to work - command line specification
16 # rng.rand(z) creates an array of size z
17 x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
18
19 # Function syntax:
20 #   arg[0] = parameters to be passed in
21 #   arg[1] = the function to be performed
22 # We pass our shared array as part of the function definition
23 f = function([], T.exp(x))
24
25 begin = timer()
26 for i in xrange(iters):
27         r = f()
28 end = timer()
29 print("Looping %d times took %f seconds" % (iters, end - begin))
30 print("Result is %s" % (r,))
31
32 # Let's make sure we used the correct processing unit by printing and comparing to what we expe
   ct
33 if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
34     print('Used the cpu')
35 else:
36     print('Used the gpu')
                                                                    1,1           All
```

# Theano: Digging Deeper ~ Optimization

https://github.com/Theano/Theano/blob/master/theano/tensor/opt.py

Optimization: Creating graphs for tensors before values are reached

```
def inplace_elemwise_optimizer_op(OP) # Parametrization for GPU | CPU
    def inplace_elemwise_optimizer(fgraph)
```

ex  // x + y + z -> x += y += z
ex2// (x + y) * (x * y) -> (x += y) *= (x * y) or (x + y) *= (x *= y)

# Caffe: Overview

Python Deep Learning framework
Focused on expression, speed, and modularity

Available on git and PyPI (requires several dependencies)

https://github.com/BVLC/caffe
http://caffe.berkeleyvision.org/installation.html

# Caffe: Features

**Expressive Architecture**

switch between CPU and GPU through flags*

Extensible Code

**Speed**

can process over 60M images per day with a single NVIDIA K40 GPU

*We only show GPU results in later slides due to the nature of extracting functions from Caffe

http://caffe.berkeleyvision.org

# Caffe:Notes

Works with Python and Matlab

Python version uses Boost.Python to integrate C++ and Python
http://www.boost.org/doc/libs/1_59_0/libs/python/doc/index.html

Hardware performance benchmarking: http://caffe.berkeleyvision.org/performance_hardware.html

# Caffe: Code

Utils file located on Git: https://github.com/ClaytonTurner/GPU-Benchmarking/blob/master/code/caffe_utils.py

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP... | caturner3@caturner3-desktop: ~/GP...

```
1  from __future__ import print_function
2  from caffe import layers as L, params as P, to_proto
3  from caffe.proto import caffe_pb2
4  from timeit import default_timer as timer
5  import numpy
6
7  # helper function for common structures
8  vlen = 10 * 30 * 768
9  rng = numpy.random.RandomState(42)
10 blob = caffe_pb2.BlobProto()
11 blob.data.extend(list(rng.rand(vlen)))
12 data = blob.data
13
14 def conv_relu(bottom, ks, nout, stride=1, pad=0, group=1):
15     conv = L.Convolution(bottom, kernel_size=ks, stride=stride,
16                                  num_output=nout, pad=pad, group=group)
17     return conv, L.ReLU(conv, in_place=True)
18
19 def fc_relu(bottom, nout):
20     fc = L.InnerProduct(bottom, num_output=nout)
21     return fc, L.ReLU(fc, in_place=True)
22
23 def max_pool(bottom, ks, stride=1):
24     return L.Pooling(bottom, pool=P.Pooling.MAX, kernel_size=ks, stride=stride)
25
26 def euc_loss(x):
27     return L.EuclideanLoss(x,x)
28
29 def caffenet(lmdb, batch_size=256, include_acc=False):
30     euc1 = euc_loss(data)
31     #loss = L.SoftmaxWithLoss(euc1, label)
32     #return to_proto(loss)
33     return None
34
35 def make_net():
36     iters = 1000000
37     begin = timer()
38     for i in xrange(iters):
39         with open('../data/train.prototxt', 'w') as f:
40             print(caffenet('/path/to/caffe-train-lmdb'), file=f)
41         with open('../data/test.prototxt', 'w') as f:
42             print(caffenet('/path/to/caffe-val-lmdb', batch_size=50, include_acc=True), file=f)
43     end = timer()
44     print("Looping %d times took %f seconds" % (iters, end - begin))
45
46 if __name__ == '__main__':
47     make_net()
```

1,1 All

# Caffe: Digging Deeper ~ Establishing GPU

https://github.com/BVLC/caffe/blob/master/src/caffe/parallel.cpp

```
}

template<typename Dtype>
Params<Dtype>::Params(shared_ptr<Solver<Dtype> > root_solver)
    : size_(total_size<Dtype>(root_solver->net()->learnable_params())),
      data_(),
      diff_() {
}

template<typename Dtype>
GPUParams<Dtype>::GPUParams(shared_ptr<Solver<Dtype> > root_solver, int device)
    : Params<Dtype>(root_solver) {
#ifndef CPU_ONLY
  int initial_device;
  CUDA_CHECK(cudaGetDevice(&initial_device));

  // Allocate device buffers
  CUDA_CHECK(cudaSetDevice(device));
  CUDA_CHECK(cudaMalloc(&data_, size_ * sizeof(Dtype)));

  // Copy blob values
  const vector<Blob<Dtype>*>& net =
      root_solver->net()->learnable_params();
  apply_buffers(net, data_, size_, copy);

  CUDA_CHECK(cudaMalloc(&diff_, size_ * sizeof(Dtype)));
  caffe_gpu_set(size_, Dtype(0), diff_);

  CUDA_CHECK(cudaSetDevice(initial_device));
#else
  NO_GPU;
#endif
}

template<typename Dtype>
GPUParams<Dtype>::~GPUParams() {
#ifndef CPU_ONLY
  CUDA_CHECK(cudaFree(data_));
  CUDA_CHECK(cudaFree(diff_));
#endif
}

template<typename Dtype>
void GPUParams<Dtype>::configure(Solver<Dtype>* solver) const {
  const vector<Blob<Dtype>*>& net =
      solver->net()->learnable_params();
  apply_buffers(net, data_, size_, replace_gpu);
  apply_buffers(net, diff_, size_, replace_gpu_diff);
}
```

# Caffe: Digging Deeper ~ Updating Layers with Mutex Locking

https://github.com/BVLC/caffe/blob/master/src/caffe/layer.cpp

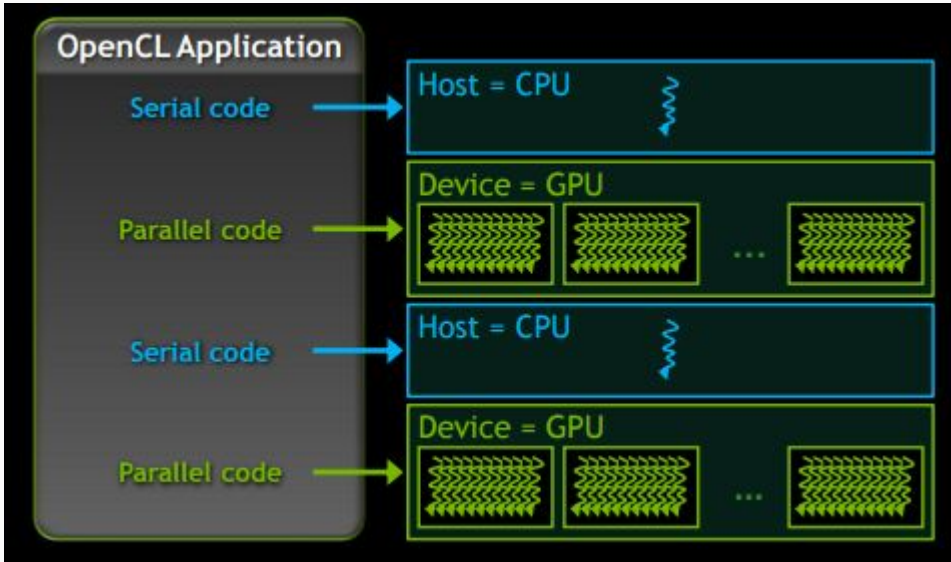
```cpp
#include <boost/thread.hpp>
#include "caffe/layer.hpp"

namespace caffe {

template <typename Dtype>
void Layer<Dtype>::InitMutex() {
  forward_mutex_.reset(new boost::mutex());
}

template <typename Dtype>
void Layer<Dtype>::Lock() {
  if (IsShared()) {
    forward_mutex_->lock();
  }
}

template <typename Dtype>
void Layer<Dtype>::Unlock() {
  if (IsShared()) {
    forward_mutex_->unlock();
  }
}

INSTANTIATE_CLASS(Layer);

}  // namespace caffe
```

# Torch: Overview

TorchCl Library for integrating with OpenCL

Flexible, fast framework operating under LuaJIT and C/CUDA

Mostly neural network and optimization libraries



# Torch: Features

**Powerful N-dimensional array**
Efficient data munging routines
**Interface to C, via LuaJIT**
Linear algebra routines
Neural network and energy-based models at heart
**Numeric optimization routines**
**Fast and efficient GPU support**
Embeddable with iOS, Android, and FPGA backends

http://torch.ch/

# Torch: Code

caturner3@caturner3-desktop: ~/GPU-Be

caturner3@caturner3-desktop: ~ | caturner

```
1 x = torch.rand(1,10*30*768)
2
3 for i = 1, 1000 do
4         y = torch.exp(x)
5 end
6
7 time = os.clock()
8 print("Time elapsed: ", time)
```

CPU

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP...

```
1  require 'cltorch'
2  require 'math'
3
4  x = torch.Tensor(1,10*30*768):uniform():cl()
5  y = torch.Tensor(1,10*30*768):zero():cl()
6
7  iters = 1000000
8
9  --begin = os.time()
10 for i=1,iters do
11         y:apply("x = exp(x)")
12         --x:map(y, "x = exp(x)")
13 end
14 --stop = os.time()
15
16 -- os.time() has second resolution - garbage for us
17 -- So we're going to use os.clock() even though it includes th
18 --print("Time elapsed: ",os.difftime(stop,begin))
19 print("Time elapsed: ", os.clock())
```

GPU

# Torch: Digging Deeper ~ C <-> Lua integration

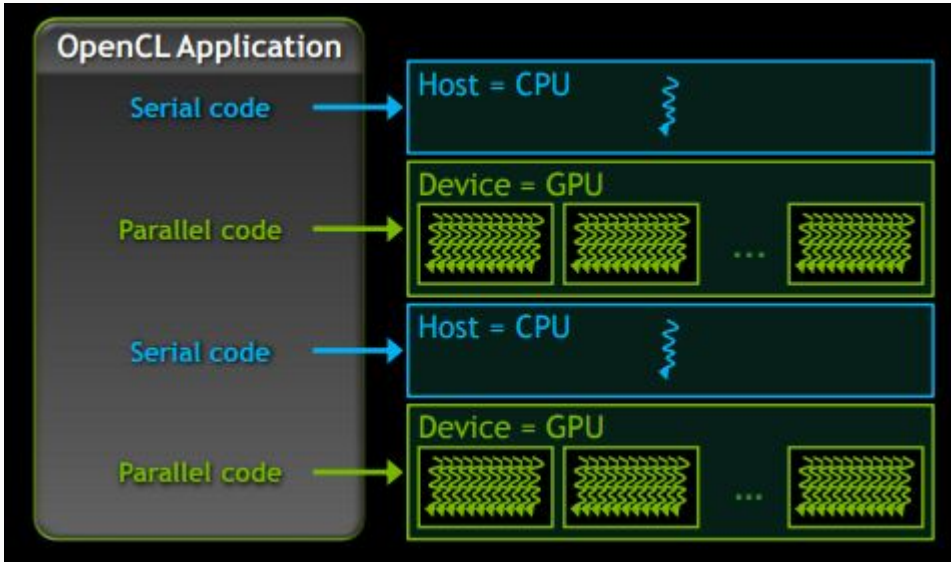
https://github.com/torch/torch7/blob/master/lib/luaT/luaT.c

```c
#include <stdlib.h>
#include <string.h>

#include "luaT.h"

void* luaT_alloc(lua_State *L, long size)
{
  void *ptr;

  if(size == 0)
    return NULL;

  if(size < 0)
    luaL_error(L, "$ Torch: invalid memory size -- maybe an overflow?");

  ptr = malloc(size);
  if(!ptr)
    luaL_error(L, "$ Torch: not enough memory: you tried to allocate %dGB. Buy new RAM!", size/1073741824);

  return ptr;
}

void* luaT_realloc(lua_State *L, void *ptr, long size)
{
  if(!ptr)
    return(luaT_alloc(L, size));

  if(size == 0)
  {
    luaT_free(L, ptr);
    return NULL;
  }

  if(size < 0)
    luaL_error(L, "$ Torch: invalid memory size -- maybe an overflow?");

  ptr = realloc(ptr, size);
  if(!ptr)
    luaL_error(L, "$ Torch: not enough memory: you tried to reallocate %dGB. Buy new RAM!", size/1073741824);
  return ptr;
}

void luaT_free(lua_State *L, void *ptr)
{
  free(ptr);
}

void luaT_setfuncs(lua_State *L, const luaL_Reg *l, int nup)
{
#if LUA_VERSION_NUM == 501
  luaL_checkstack(L, nup+1, "too many upvalues");
```

# PyOpenCL: OpenCL Refresher

Used for benchmarking standard OpenCL integration with Python for results

http://mathema.tician.de/software/pyopencl/
http://documen.tician.de/pyopencl/



# PyOpenCL: Code

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP... | caturner3@caturner3-desktop: ~/GP...

```
1  import pyopencl as cl
2  import numpy
3  import sys
4  from timeit import default_timer as timer
5
6  class CL(object):
7      def __init__(self, size=10*30*768):
8          self.size = size
9          self.ctx = cl.create_some_context()
10         self.queue = cl.CommandQueue(self.ctx)
11         self.iters = 1000000
12
13     def load_program(self):
14         # Using exponentiate instead of exp to avoid overloading/overriding
15         fstr="""
16         __kernel void exponentiate(__global float* a, __global float* c)
17         {
18             unsigned int i = get_global_id(0);
19
20            c[i] = a[i] * a[i];
21         }
22         """
23         self.program = cl.Program(self.ctx, fstr).build()
24
25     def popCorn(self):
26         mf = cl.mem_flags
27
28     # We use float32s here to have a fair comparison to the CPU
29         self.a = numpy.array(range(self.size), dtype=numpy.float32)
30
31         self.a_buf = cl.Buffer(self.ctx, mf.READ_ONLY | mf.COPY_HOST_PTR,
32                                hostbuf=self.a)
33         self.dest_buf = cl.Buffer(self.ctx, mf.WRITE_ONLY, self.a.nbytes)
34
35     def execute(self):
36         begin = timer()
37         for i in xrange(self.iters):
38             self.program.exponentiate(self.queue, self.a.shape, None, self.a_buf, self.dest_buf
   )
39         end = timer()
40         c = numpy.empty_like(self.a)
41         cl.enqueue_read_buffer(self.queue, self.dest_buf, c).wait()
42         print("Looping %d times took %f seconds" % (self.iters, end - begin))
43         print("Result is %s" % (c))
44
45 if __name__ == '__main__':
46     matrixmul = CL(10000000)
47     matrixmul.load_program()
48     matrixmul.popCorn()
49     matrixmul.execute()
                                                                    2,1           All
```

# Numpy Code

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP... | caturner3@caturner3-desktop: ~/GP...

```
1  import numpy
2  from timeit import default_timer as timer # Since we need to use Python 2.7, this is a good alt
   ernative to Perfcounter
3
4  vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
5  iters = 1000
6
7  # Keep consistent results with random seed
8  rng = numpy.random.RandomState(42)
9
10 # float32 so we have a fair comparison to the Theano times
11 # rng.rand(z) creates an array of size z
12 x = numpy.asarray(rng.rand(vlen), numpy.float32)
13
14 begin = timer()
15 for i in xrange(iters):
16         r = numpy.exp(x)
17 end = timer()
18 print("Looping %d times took %f seconds" % (iters, end - begin))
19 print("Result is %s" % (r,))
                                                                    1,1           All
```

# Baseline Code

caturner3@caturner3-desktop: ~/GPU-Benchmarking/code

caturner3@caturner3-desktop: ~ | caturner3@caturner3-desktop: ~/GP... | caturner3@cat

```
# We convert back to Python native lists before timing to avoid timing bia

import numpy
from timeit import default_timer as timer # Since we need to use Python 2.
ernative to Perfcounter

vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 1000

# Keep consistent results with random seed
rng = numpy.random.RandomState(42)

# float32 so we have a fair comparison to the Theano times
# rng.rand(z) creates an array of size z
x = numpy.asarray(rng.rand(vlen), numpy.float32)
x = x.tolist()

def exp(exp_list):
    for i in xrange(len(exp_list)):
        exp_list[i] = exp_list[i] ** 2
    return exp_list

begin = timer()
for i in xrange(iters):
        r = exp(x)
end = timer()
print("Looping %d times took %f seconds" % (iters, end - begin))
# We're not printing the result because python doesn't, by default, have t
en in Theano and Numpy by default
#print("Result is %s" % (r,))
```

# Exponentiating Numbers

1000 iterations of exponentiating

Torch*
- Timing included data migration to GPU (see code)

Caffe**
- No simple array interface
- Utilized sum squared layers to mimic exponentiating

| | |
|---|---|
| Torch - TorchCL (GPU)* | 0.156538 seconds |
| Caffe** | 0.167299 seconds |
| Theano - FAST_RUN (GPU) | 0.555048 seconds |
| PyOpenCL | 0.976866seconds |
| Numpy | 2.904454 seconds |
| Theano - FAST_COMPILE (CPU) | 5.129712 seconds |
| Torch - (CPU) | 6.009791 seconds |
| Normal Looping | 22.315899 seconds |
| Theano - DebugMode | 52.846434 seconds |

# Exponentiating Numbers

1000000 iterations of exponentiating

Torch*
- Timing included data migration to GPU (see code)

Caffe**
- No simple array interface
- Utilized sum squared layers to mimic exponentiating

| | |
|---|---|
| Torch - TorchCL (GPU)* | 83.952904 seconds |
| Caffe** | 305.870450 seconds |
| Theano - FAST_RUN (GPU) | 525.722036 seconds |
| PyOpenCL | 1359.186030 seconds |
| Numpy | 2886.777740 seconds |
| Theano - FAST_COMPILE (CPU) | 2921.630621 seconds |
| Torch - (CPU) | 5375.984789 seconds |
| Normal Looping | seconds |
| Theano - DebugMode | seconds |

# Demos/Else

https://github.com/lisa-lab/DeepLearningTutorials

http://demo.caffe.berkeleyvision.org/

https://github.com/torch/demos

Code displayed using VIM with a custom .vimrc
Available at: https://github.com/ClaytonTurner/UsefulScripts/tree/master/vim

# References


NVIDIA

http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-550ti

Theano

http://deeplearning.net/software/theano/index.html

Caffe

http://caffe.berkeleyvision.org/

Torch

http://torch.ch/docs/getting-started.html

https://github.com/hughperkins/cltorch

PyOpenCL

http://documen.tician.de/pyopencl/