The PyCUDA module

Will Landau

Iowa State University

December 2, 2013

Outline

Getting started

Short examples

A glimpse at ABC-SysBio

Outline

Getting started

Short examples

A glimpse at ABC-SysBio

demo.py

- Import and initialize PyCUDA:

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
```

- Initial data: a 4 × 4 array of numbers:

```
import numpy
a = numpy.random.randn(4,4)
```

- Many NVIDIA cards only support single precision:

```
a = a.astype(numpy.float32)
```

demo.py

- Allocate device memory:

```
a_gpu = cuda.mem_alloc(a.nbytes)
```

- Send data to the device:

```
cuda.memcpy_htod(a_gpu, a)
```

- Define a kernel to multiply each array entry by 2:

```
mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)
```

demo.py

- Turn our CUDA C kernel into a callable Python function:

```
func = mod.get_function("doublify")
```

- Call the kernel with:
  - 1 grid
  - 1 block
  - 4 threads in the x direction
  - 4 threads in the y direction
  - 1 thread in the z direction

```
func(a_gpu, block=(4,4,1))
```

demo.py

- Make a NumPy array to store the results:

```
a_doubled = numpy.empty_like(a)
```

- Copy the results to the host:

```
cuda.memcpy_dtoh(a_doubled, a_gpu)
```

- Print arrays:

```
print a_doubled
print a
```

Example output

```
[landau@impact1 PyCUDA_sandbox]$ python demo.py
[[-1.29063177  0.82264316  0.02254304  2.0740006 ]
 [ 1.40431428  1.95245779 -1.84627843 -1.5800966 ]
 [-2.77298713  0.99803442  1.85154581  0.63633269]
 [ 0.55860651 -0.50091052 -1.465307    4.12601614]]
[[-0.64531589  0.41132158  0.01127152  1.0370003 ]
 [ 0.70215714  0.97622889 -0.92313921 -0.7900483 ]
 [-1.38649356  0.49901721  0.92577291  0.31816635]
 [ 0.27930325 -0.25045526 -0.7326535   2.06300807]]
```

Simplifying memory transfer

- There are three function argument handlers that take care of memory transfer for the user:
  - `pycuda.driver.In`
  - `pycuda.driver.Out`
  - `pycuda.driver.InOut`

hello_gpu.py

```python
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(400,1,1), grid=(1,1))

print dest-a*b
```

Example output

```
> python hello_gpu.py
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.]
```

demohandler.py

```python
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

import numpy

a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
print "Original array:"
print a

mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)

func = mod.get_function("doublify")
func(cuda.InOut(a), block=(4, 4, 1))

print "Doubled array:"
print a
```

Example output

```
> python demohandler.py
Original array:
[[-0.35754886 -0.08118289  1.42489266  0.6799224 ]
 [ 0.54355925 -2.00721192 -0.6814152  -0.88118494]
 [ 1.29756403  1.37618589  0.78046876 -0.93179333]
 [-0.96092844  0.5301944  -0.36968505  1.54017532]]
Doubled array:
[[-0.71509773 -0.16236578  2.84978533  1.3598448 ]
 [ 1.08711851 -4.01442385 -1.3628304  -1.76236987]
 [ 2.59512806  2.75237179  1.56093752 -1.86358666]
 [-1.92185688  1.0603888  -0.73937011  3.08035064]]
```

demoshort.py

- Use a pycuda.gpuarray.GPUArray to shorten the code even more.

```
import pycuda.gpuarray as gpuarray
import pycuda.driver as cuda
import pycuda.autoinit
import numpy

a_gpu = gpuarray.to_gpu(numpy.random.randn(4,4).astype(numpy.float32))
a_doubled = (2*a_gpu).get()
print a_doubled
print a_gpu
```

- The output is analogous.

Outline

Getting started

Short examples

A glimpse at ABC-SysBio

functiontemplates.py

```
import pycuda.gpuarray as gpuarray
import pycuda.driver as drv
import pycuda.autoinit
import numpy as np

from pycuda.compiler import SourceModule
func_mod = SourceModule("""
template <class T>
__device__ T incr(T x) {
    return (x + 1.0);
}

// Needed to avoid name mangling so that PyCUDA can
// find the kernel function:
extern "C" {
    __global__ void func(float *a, int N)
    {
        int idx = threadIdx.x;
        if (idx < N)
            a[idx] = incr(a[idx]);
    }
}
""", no_extern_c=1)
```

functiontemplates.py

```python
func = func_mod.get_function('func')

N = 5
x = np.asarray(np.random.rand(N), np.float32)
x_orig = x.copy()
x_gpu = gpuarray.to_gpu(x)

func(x_gpu.gpudata, np.uint32(N), block=(N, 1, 1))
print 'x:       ', x
print 'incr(x): ', x_gpu.get()
```

```
> python functiontemplates.py
x:          [ 0.79577702  0.73002166  0.19413722  0.30437419  0.24752268]
incr(x):    [ 1.79577708  1.73002172  1.19413722  1.30437422  1.24752271]
```

MatmulSimple.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Multiples two square matrices together using a *single* block of threads and
global memory only. Each thread computes one element of the resulting matrix.
"""

import numpy as np
from pycuda import driver, compiler, gpuarray, tools

# -- initialize the device
import pycuda.autoinit
```

MatmulSimple.py

```
kernel_code_template = """
__global__ void MatrixMulKernel(float *a, float *b, float *c)
{
    // 2D Thread ID (assuming that only *one* block will be executed)
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Pvalue is used to store the element of the matrix
    // that is computed by the thread
    float Pvalue = 0;

    // Each thread loads one row of M and one column of N,
    //   to produce one element of P.
    for (int k = 0; k < %(MATRIX_SIZE)s; ++k) {
        float Aelement = a[ty * %(MATRIX_SIZE)s + k];
        float Belement = b[k * %(MATRIX_SIZE)s + tx];
        Pvalue += Aelement * Belement;
    }

    // Write the matrix to device memory;
    // each thread writes one element
    c[ty * %(MATRIX_SIZE)s + tx] = Pvalue;
}
"""
```

MatmulSimple.py

```
# define the (square) matrix size
#  note that we'll only use *one* block of threads here
#  as a consequence this number (squared) can't exceed max_threads,
#  see http://documen.tician.de/pycuda/util.html#pycuda.tools.DeviceData
#  for more information on how to get this number for your device
MATRIX_SIZE = 2

# create two random square matrices
a_cpu = np.random.randn(MATRIX_SIZE, MATRIX_SIZE).astype(np.float32)
b_cpu = np.random.randn(MATRIX_SIZE, MATRIX_SIZE).astype(np.float32)

# compute reference on the CPU to verify GPU computation
c_cpu = np.dot(a_cpu, b_cpu)

# transfer host (CPU) memory to device (GPU) memory
a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)

# create empty gpu array for the result (C = A * B)
c_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
```

MatmulSimple.py

```
# get the kernel code from the template
# by specifying the constant MATRIX_SIZE
kernel_code = kernel_code_template % {
    'MATRIX_SIZE': MATRIX_SIZE
    }

# compile the kernel code
mod = compiler.SourceModule(kernel_code)

# get the kernel function from the compiled module
matrixmul = mod.get_function("MatrixMulKernel")

# call the kernel on the card
matrixmul(
    # inputs
    a_gpu, b_gpu,
    # output
    c_gpu,
    # (only one) block of MATRIX_SIZE x MATRIX_SIZE threads
    block = (MATRIX_SIZE, MATRIX_SIZE, 1),
    )

# print the
```

MatmulSimple.py

```
# print the results
print "-" * 80
print "Matrix A (GPU):"
print a_gpu.get()

print "-" * 80
print "Matrix B (GPU):"
print b_gpu.get()

print "-" * 80
print "Matrix C (GPU):"
print c_gpu.get()

print "-" * 80
print "CPU-GPU difference:"
print c_cpu - c_gpu.get()

np.allclose(c_cpu, c_gpu.get())
```

Example output

```
python MatmulSimple.py
------------------------------------------------------------
Matrix A (GPU):
[[ 0.46055064 -0.85658211]
 [ 0.57233274  2.47072577]]
------------------------------------------------------------
Matrix B (GPU):
[[ 1.76631308  0.0654699 ]
 [-0.13310859  0.73874539]]
------------------------------------------------------------
Matrix C (GPU):
[[ 0.92749506 -0.60264391]
 [ 0.68204403  1.86270785]]
------------------------------------------------------------
CPU-GPU difference:
[[ 0.  0.]
 [ 0.  0.]]
```

pycurand.py

- We can use CURAND with PyCUDA and build kernels with the `pycuda.elementwise` module.

```python
import pycuda.gpuarray as gpuarray
import pycuda.autoinit
import numpy
from pycuda.curandom import rand as curand

a_gpu = curand((50,))
b_gpu = curand((50,))

from pycuda.elementwise import ElementwiseKernel
lin_comb = ElementwiseKernel(
        "float a, float *x, float b, float *y, float *z",
        "z[i] = a*x[i] + b*y[i]",
        "linear_combination")

c_gpu = gpuarray.empty_like(a_gpu)
lin_comb(5, a_gpu, 6, b_gpu, c_gpu)

import numpy.linalg as la
assert la.norm((c_gpu - (5*a_gpu+6*b_gpu)).get()) < 1e-5
```

reduction.py

```python
import pycuda.gpuarray as gpuarray
import pycuda.driver as cuda
import pycuda.autoinit
import numpy
from pycuda.reduction import ReductionKernel

a = gpuarray.arange(400, dtype=numpy.float32)
b = gpuarray.arange(400, dtype=numpy.float32)

print a

krnl = ReductionKernel(numpy.float32, neutral="0",
        reduce_expr="a+b", map_expr="x[i]*y[i]",
        arguments="float *x, float *y")

my_dot_prod = krnl(a, b).get()
print my_dot_prod
```

scan.py

```
import pycuda.gpuarray as gpuarray
import pycuda.driver as cuda
import pycuda.autoinit
import numpy as np
from pycuda.scan import InclusiveScanKernel

knl = InclusiveScanKernel(np.int32, "a+b")

n = 2**20-2**18+5
host_data = np.random.randint(0, 10, n).astype(np.int32)
dev_data = gpuarray.to_gpu(host_data)

knl(dev_data)
assert (dev_data.get() == np.cumsum(host_data, axis=0)).all()
```

MeasureGpuarraySpeedRandom.py

```python
#! /usr/bin/env python
import pycuda.autoinit
import pycuda.driver as drv
import pycuda.curandom as curandom
import numpy
import numpy.linalg as la
from pytools import Table

def main():
    import pycuda.gpuarray as gpuarray

    sizes = []
    times = []
    flops = []
    flopsCPU = []
    timesCPU = []
```

MeasureGpuarraySpeedRandom.py

```
    for power in range(10, 25): # 24
        size = 1<<power
        print size
        sizes.append(size)
        a = gpuarray.zeros((size,), dtype=numpy.float32)

        if power > 20:
            count = 100
        else:
            count = 1000

        #start timer
        start = drv.Event()
        end = drv.Event()
        start.record()

        #cuda operation which fills the array with random numbers
        for i in range(count):
            curandom.rand((size, ))

        #stop timer
        end.record()
        end.synchronize()
```

MeasureGpuarraySpeedRandom.py

```
        #calculate used time
        secs = start.time_till(end)*1e-3

        times.append(secs/count)
        flops.append(size)

        #cpu operations which fills teh array with random data
        a = numpy.array((size,), dtype=numpy.float32)

        #start timer
        start = drv.Event()
        end = drv.Event()
        start.record()

        #cpu operation which fills the array with random data
        for i in range(count):
            numpy.random.rand(size).astype(numpy.float32)

        #stop timer
        end.record()
        end.synchronize()

        #calculate used time
        secs = start.time_till(end)*1e-3

        #add results to variable
        timesCPU.append(secs/count)
        flopsCPU.append(size)
```

MeasureGpuarraySpeedRandom.py

```
    #calculate pseudo flops
    flops = [f/t for f, t in zip(flops,times)]
    flopsCPU = [f/t for f, t in zip(flopsCPU,timesCPU)]

    #print the data out
    tbl = Table()
    tbl.add_row(("Size", "Time GPU", "Size/Time GPU", "Time CPU","Size/
         Time CPU","GPU vs CPU speedup"))
    for s, t, f,tCpu,fCpu in zip(sizes, times, flops,timesCPU,flopsCPU):
        tbl.add_row((s,t,f,tCpu,fCpu,f/fCpu))
    print tbl

if __name__ == "__main__":
    main()
```

DumpProperties.py

```
import pycuda.driver as drv

drv.init()
print "%d device(s) found." % drv.Device.count()

for ordinal in range(drv.Device.count()):
    dev = drv.Device(ordinal)
    print "Device #%d: %s" % (ordinal, dev.name())
    print "  Compute Capability: %d.%d" % dev.compute_capability()
    print "  Total Memory: %s KB" % (dev.total_memory()//(1024))
    atts = [(str(att), value)
            for att, value in dev.get_attributes().iteritems()]
    atts.sort()

    for att, value in atts:
        print "  %s: %s" % (att, value)
```

Outline

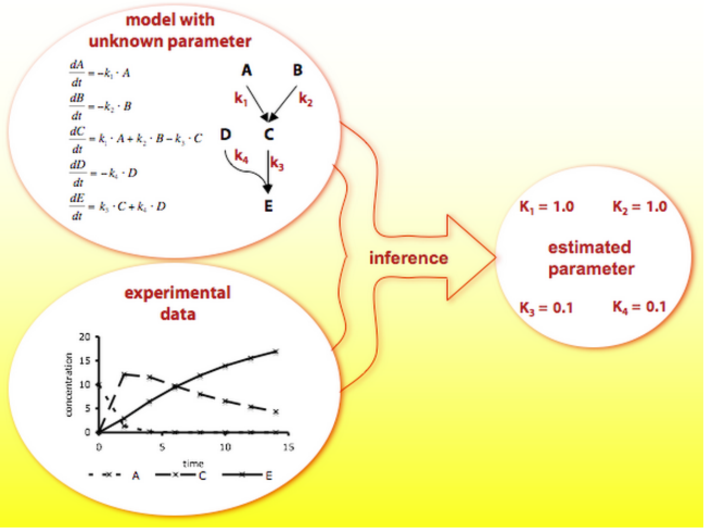
Getting started

Short examples

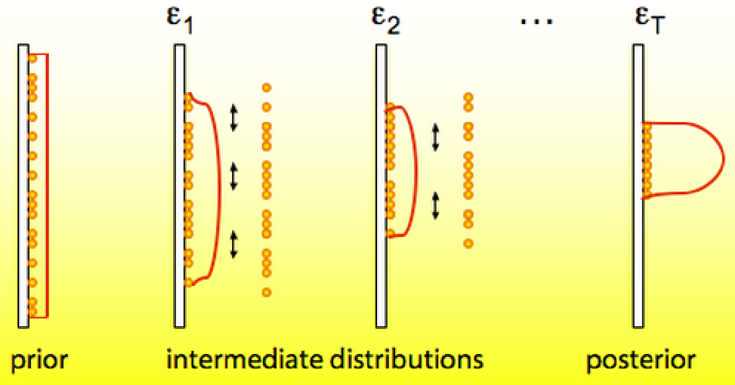
A glimpse at ABC-SysBio

ABC-SysBio: a PyCUDA-implemented toolkit

- GPU-accelerated approximate Bayesian computation for parameter estimation in biological dynamical systems


model with unknown parameter
$\frac{dA}{dt} = -k_1 \cdot A$
$\frac{dB}{dt} = -k_2 \cdot B$
$\frac{dC}{dt} = k_1 \cdot A + k_2 \cdot B - k_3 \cdot C$
$\frac{dD}{dt} = -k_4 \cdot D$
$\frac{dE}{dt} = k_3 \cdot C + k_4 \cdot D$
A
B
k_1
k_2
D
C
k_4
k_3
E
experimental data
concentration
20
15
10
5
0
0
5
10
15
time
A
C
E
inference
$K_1 = 1.0$
$K_2 = 1.0$
estimated parameter
$K_3 = 0.1$
$K_4 = 0.1$

ABC-SysBio: a PyCUDA-implemented toolkit



- Methods
    - ABC rejection sampler
    - ABC SMC for parameter inference
    - ABC SMC for model selection

ABC-SysBio: a PyCUDA-implemented toolkit

- ABC-SysBio is ready to use on impact1.
  - `import abcsysbio` (Python script)
  - `abc-sysbio-sbml-sum` (command line)
  - `run-abc-sysbio` (command line)
- For more information, visit:
  - http://www.theosysbio.bio.ic.ac.uk/resources/abc-sysbio
  - http://bioinformatics.oxfordjournals.org/content/26/14/1797.full?keytype=ref&ijkey=AVSfAhR7XFxjrMj
- For the input files in the online examples, visit:
  - http://will-landau.com/gpu/pycuda.html

Outline

Getting started

Short examples

A glimpse at ABC-SysBio

Other resources

- Guides and papers
  - Klockner A. Examples of PyCUDA usage. http://wiki.tiker.net/PyCuda/Examples. May 2012.
  - Klockner A. PyCUDA 2012.1 documentation. http://documen.tician.de/pycuda/index.html. June 2012.
  - C. Barnes, J. Liepe, E. Cule, S. Filippi, D. Rolando, S. McMahon, B. Lisowska, P. Kirk, K. Erguler, T. Toni, and M. Stumpf. *Abc-sysbio: A tool for parameter inference and model selection.* http://www.theosysbio.bio.ic.ac.uk/resources/abc-sysbio. 2011.
  - J. Liepe, C. Barnes, E. Cule, K. Erguler, P. Kirk, T. Toni, and M. Stumpf. *Abc-sysbio-approximate bayesian computation in Python with gpu support.* Bioinformatics, 26(14):17971799, May 2010.
- Example PyCUDA and ABC-SysBio code are available at http://will-landau.com/gpu/pycuda.html.

That's all for the semester.

- Series materials are available at http://will-landau.com/gpu.