The PyCUDA module

Will Landau

Iowa State University

December 2, 2013

Outline

Outline

demo.py

- Import and initialize PyCUDA:

```
1 import pycuda.driver as cuda
2 import pycuda.autoinit
3 from pycuda.compiler import SourceModule
```

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

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

- Many NVIDIA cards only support single precision:

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

demo.py

- Allocate device memory:

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

- Send data to the device:

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

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

```
 9 mod = SourceModule("""
10   __global__ void doublify(float *a)
11   {
12     int idx = threadIdx.x + threadIdx.y*4;
13     a[idx] *= 2;
14   }
15   """)
```

demo.py

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

```
16 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

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

demo.py

- Make a NumPy array to store the results:

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

- Copy the results to the host:

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

- Print arrays:

```
20 print a_doubled
21 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

```
1  import pycuda.autoinit
2  import pycuda.driver as drv
3  import numpy
4
5  from pycuda.compiler import SourceModule
6  mod = SourceModule("""
7  __global__ void multiply_them(float *dest, float *a, float *b)
8  {
9    const int i = threadIdx.x;
10   dest[i] = a[i] * b[i];
11 }
12 """)
13
14 multiply_them = mod.get_function("multiply_them")
15
16 a = numpy.random.randn(400).astype(numpy.float32)
17 b = numpy.random.randn(400).astype(numpy.float32)
18
19 dest = numpy.zeros_like(a)
20 multiply_them(
21         drv.Out(dest), drv.In(a), drv.In(b),
22         block=(400,1,1), grid=(1,1))
23
24 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

```
1  import pycuda.driver as cuda
2  import pycuda.autoinit
3  from pycuda.compiler import SourceModule
4
5  import numpy
6
7  a = numpy.random.randn(4,4)
8  a = a.astype(numpy.float32)
9  print "Original array:"
10 print a
11
12 mod = SourceModule("""
13   __global__ void doublify(float *a)
14   {
15     int idx = threadIdx.x + threadIdx.y*4;
16     a[idx] *= 2;
17   }
18   """)
19
20 func = mod.get_function("doublify")
21 func(cuda.InOut(a), block=(4, 4, 1))
22
23 print "Doubled array:"
24 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.

```
1 import pycuda.gpuarray as gpuarray
2 import pycuda.driver as cuda
3 import pycuda.autoinit
4 import numpy
5
6 a_gpu = gpuarray.to_gpu(numpy.random.randn(4,4).astype(numpy.float32))
7 a_doubled = (2*a_gpu).get()
8 print a_doubled
9 print a_gpu
```

- The output is analogous.

Outline

functiontemplates.py

```
1  import pycuda.gpuarray as gpuarray
2  import pycuda.driver as drv
3  import pycuda.autoinit
4  import numpy as np
5
6  from pycuda.compiler import SourceModule
7  func_mod = SourceModule("""
8  template <class T>
9  __device__ T incr(T x) {
10     return (x + 1.0);
11 }
12
13 // Needed to avoid name mangling so that PyCUDA can
14 // find the kernel function:
15 extern "C" {
16     __global__ void func(float *a, int N)
17     {
18         int idx = threadIdx.x;
19         if (idx < N)
20             a[idx] = incr(a[idx]);
21     }
22 }
23 """, no_extern_c=1)
```

functiontemplates.py

```
24 func = func_mod.get_function('func')
25
26 N = 5
27 x = np.asarray(np.random.rand(N), np.float32)
28 x_orig = x.copy()
29 x_gpu = gpuarray.to_gpu(x)
30
31 func(x_gpu.gpudata, np.uint32(N), block=(N, 1, 1))
32 print 'x:       ', x
33 print 'incr(x): ', x_gpu.get()
```

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

MatmulSimple.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  """
5  Multiples two square matrices together using a *single* block of threads
       and
6  global memory only. Each thread computes one element of the resulting
       matrix.
7  """
8
9  import numpy as np
10 from pycuda import driver, compiler, gpuarray, tools
11
12 # -- initialize the device
13 import pycuda.autoinit
```

MatmulSimple.py

```
14 kernel_code_template = """
15 __global__ void MatrixMulKernel(float *a, float *b, float *c)
16 {
17     // 2D Thread ID (assuming that only *one* block will be executed)
18     int tx = threadIdx.x;
19     int ty = threadIdx.y;
20
21     // Pvalue is used to store the element of the matrix
22     // that is computed by the thread
23     float Pvalue = 0;
24
25     // Each thread loads one row of M and one column of N,
26     //   to produce one element of P.
27     for (int k = 0; k < %(MATRIX_SIZE)s; ++k) {
28         float Aelement = a[ty * %(MATRIX_SIZE)s + k];
29         float Belement = b[k * %(MATRIX_SIZE)s + tx];
30         Pvalue += Aelement * Belement;
31     }
32
33     // Write the matrix to device memory;
34     // each thread writes one element
35     c[ty * %(MATRIX_SIZE)s + tx] = Pvalue;
36 }
37 """
```

MatmulSimple.py

```
38 # define the (square) matrix size
39 #  note that we'll only use *one* block of threads here
40 #  as a consequence this number (squared) can't exceed max_threads,
41 #  see http://documen.tician.de/pycuda/util.html#pycuda.tools.DeviceData
42 #  for more information on how to get this number for your device
43 MATRIX_SIZE = 2
44
45 # create two random square matrices
46 a_cpu = np.random.randn(MATRIX_SIZE, MATRIX_SIZE).astype(np.float32)
47 b_cpu = np.random.randn(MATRIX_SIZE, MATRIX_SIZE).astype(np.float32)
48
49 # compute reference on the CPU to verify GPU computation
50 c_cpu = np.dot(a_cpu, b_cpu)
51
52 # transfer host (CPU) memory to device (GPU) memory
53 a_gpu = gpuarray.to_gpu(a_cpu)
54 b_gpu = gpuarray.to_gpu(b_cpu)
55
56 # create empty gpu array for the result (C = A * B)
57 c_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
```

MatmulSimple.py

```
59 # get the kernel code from the template
60 # by specifying the constant MATRIX_SIZE
61 kernel_code = kernel_code_template % {
62     'MATRIX_SIZE': MATRIX_SIZE
63     }
64
65 # compile the kernel code
66 mod = compiler.SourceModule(kernel_code)
67
68 # get the kernel function from the compiled module
69 matrixmul = mod.get_function("MatrixMulKernel")
70
71 # call the kernel on the card
72 matrixmul(
73     # inputs
74     a_gpu, b_gpu,
75     # output
76     c_gpu,
77     # (only one) block of MATRIX_SIZE x MATRIX_SIZE threads
78     block = (MATRIX_SIZE, MATRIX_SIZE, 1),
79     )
80
81 # print the
```

MatmulSimple.py

```
82 # print the results
83 print "-" * 80
84 print "Matrix A (GPU):"
85 print a_gpu.get()
86
87 print "-" * 80
88 print "Matrix B (GPU):"
89 print b_gpu.get()
90
91 print "-" * 80
92 print "Matrix C (GPU):"
93 print c_gpu.get()
94
95 print "-" * 80
96 print "CPU-GPU difference:"
97 print c_cpu - c_gpu.get()
98
99 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.

```
1  import pycuda.gpuarray as gpuarray
2  import pycuda.autoinit
3  import numpy
4  from pycuda.curandom import rand as curand
5
6  a_gpu = curand((50,))
7  b_gpu = curand((50,))
8
9  from pycuda.elementwise import ElementwiseKernel
10 lin_comb = ElementwiseKernel(
11         "float a, float *x, float b, float *y, float *z",
12         "z[i] = a*x[i] + b*y[i]",
13         "linear_combination")
14
15 c_gpu = gpuarray.empty_like(a_gpu)
16 lin_comb(5, a_gpu, 6, b_gpu, c_gpu)
17
18 import numpy.linalg as la
19 assert la.norm((c_gpu - (5*a_gpu+6*b_gpu)).get()) < 1e-5
```

reduction.py

```
1  import pycuda.gpuarray as gpuarray
2  import pycuda.driver as cuda
3  import pycuda.autoinit
4  import numpy
5  from pycuda.reduction import ReductionKernel
6
7  a = gpuarray.arange(400, dtype=numpy.float32)
8  b = gpuarray.arange(400, dtype=numpy.float32)
9
10 print a
11
12 krnl = ReductionKernel(numpy.float32, neutral="0",
13         reduce_expr="a+b", map_expr="x[i]*y[i]",
14         arguments="float *x, float *y")
15
16 my_dot_prod = krnl(a, b).get()
17 print my_dot_prod
```

scan.py

```
1  import pycuda.gpuarray as gpuarray
2  import pycuda.driver as cuda
3  import pycuda.autoinit
4  import numpy as np
5  from pycuda.scan import InclusiveScanKernel
6
7  knl = InclusiveScanKernel(np.int32, "a+b")
8
9  n = 2**20-2**18+5
10 host_data = np.random.randint(0, 10, n).astype(np.int32)
11 dev_data = gpuarray.to_gpu(host_data)
12
13 knl(dev_data)
14 assert (dev_data.get() == np.cumsum(host_data, axis=0)).all()
```

MeasureGpuarraySpeedRandom.py

```
1  #! /usr/bin/env python
2  import pycuda.autoinit
3  import pycuda.driver as drv
4  import pycuda.curandom as curandom
5  import numpy
6  import numpy.linalg as la
7  from pytools import Table
8
9  def main():
10     import pycuda.gpuarray as gpuarray
11
12     sizes = []
13     times = []
14     flops = []
15     flopsCPU = []
16     timesCPU = []
```

MeasureGpuarraySpeedRandom.py

```
17      for power in range(10, 25): # 24
18          size = 1<<power
19          print size
20          sizes.append(size)
21          a = gpuarray.zeros((size,), dtype=numpy.float32)
22
23          if power > 20:
24              count = 100
25          else:
26              count = 1000
27
28          #start timer
29          start = drv.Event()
30          end = drv.Event()
31          start.record()
32
33          #cuda operation which fills the array with random numbers
34          for i in range(count):
35              curandom.rand((size, ))
36
37          #stop timer
38          end.record()
39          end.synchronize()
```

MeasureGpuarraySpeedRandom.py

```
40          #calculate used time
41          secs = start.time_till(end)*1e-3
42
43          times.append(secs/count)
44          flops.append(size)
45
46          #cpu operations which fills teh array with random data
47          a = numpy.array((size,), dtype=numpy.float32)
48
49          #start timer
50          start = drv.Event()
51          end = drv.Event()
52          start.record()
53
54          #cpu operation which fills the array with random data
55          for i in range(count):
56              numpy.random.rand(size).astype(numpy.float32)
57
58          #stop timer
59          end.record()
60          end.synchronize()
61
62          #calculate used time
63          secs = start.time_till(end)*1e-3
64
65          #add results to variable
66          timesCPU.append(secs/count)
67          flopsCPU.append(size)
```

MeasureGpuarraySpeedRandom.py

```
68      #calculate pseudo flops
69      flops = [f/t for f, t in zip(flops,times)]
70      flopsCPU = [f/t for f, t in zip(flopsCPU,timesCPU)]
71
72      #print the data out
73      tbl = Table()
74      tbl.add_row(("Size", "Time GPU", "Size/Time GPU", "Time CPU","Size/
            Time CPU","GPU vs CPU speedup"))
75      for s, t, f,tCpu,fCpu in zip(sizes, times, flops,timesCPU,flopsCPU):
76          tbl.add_row((s,t,f,tCpu,fCpu,f/fCpu))
77      print tbl
78
79  if __name__ == "__main__":
80      main()
```

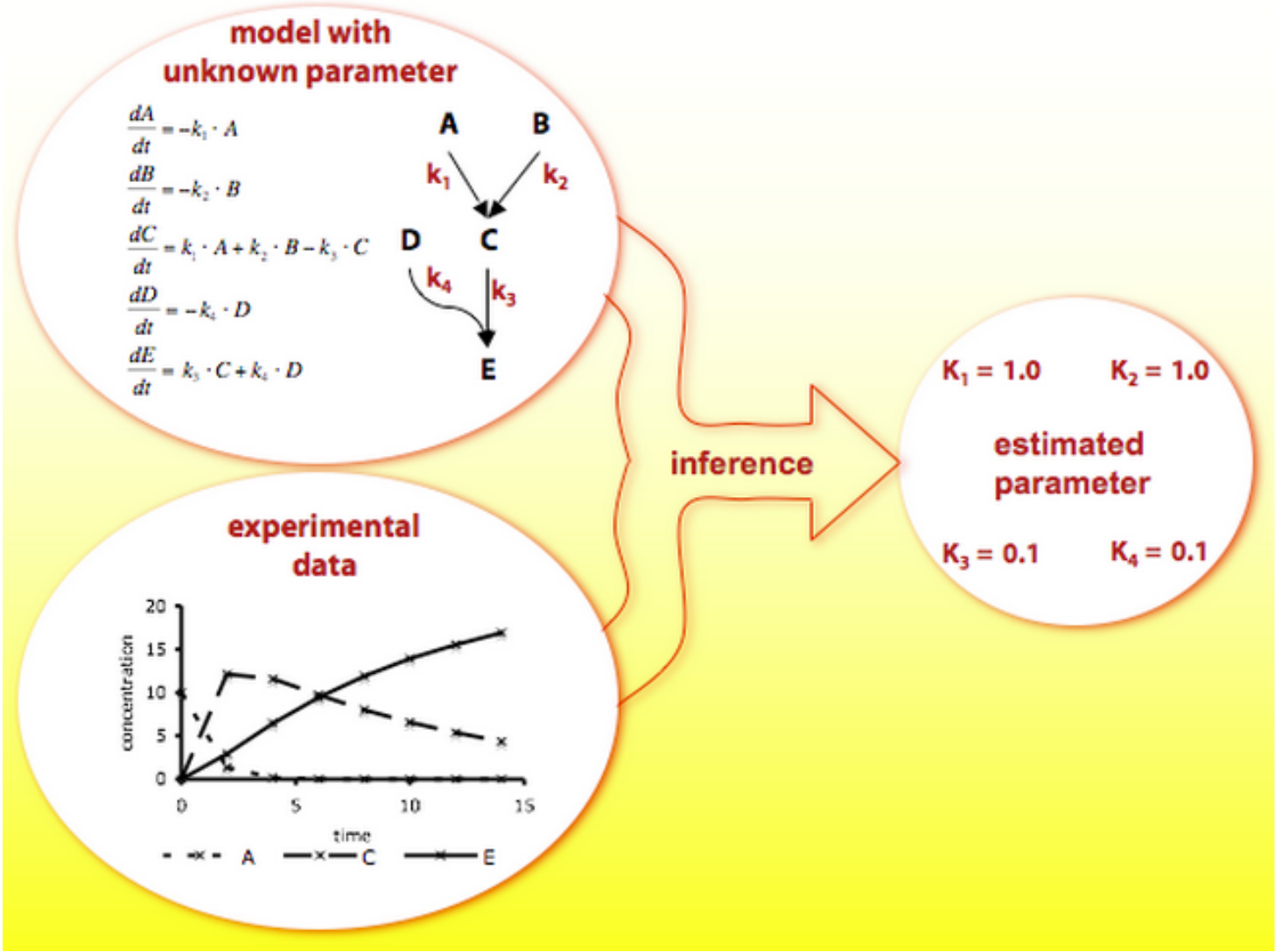
DumpProperties.py

```
1  import pycuda.driver as drv
2
3  drv.init()
4  print "%d device(s) found." % drv.Device.count()
5
6  for ordinal in range(drv.Device.count()):
7      dev = drv.Device(ordinal)
8      print "Device #%d: %s" % (ordinal, dev.name())
9      print "  Compute Capability: %d.%d" % dev.compute_capability()
10     print "  Total Memory: %s KB" % (dev.total_memory()//(1024))
11     atts = [(str(att), value)
12             for att, value in dev.get_attributes().iteritems()]
13     atts.sort()
14
15     for att, value in atts:
16         print "  %s: %s" % (att, value)
```

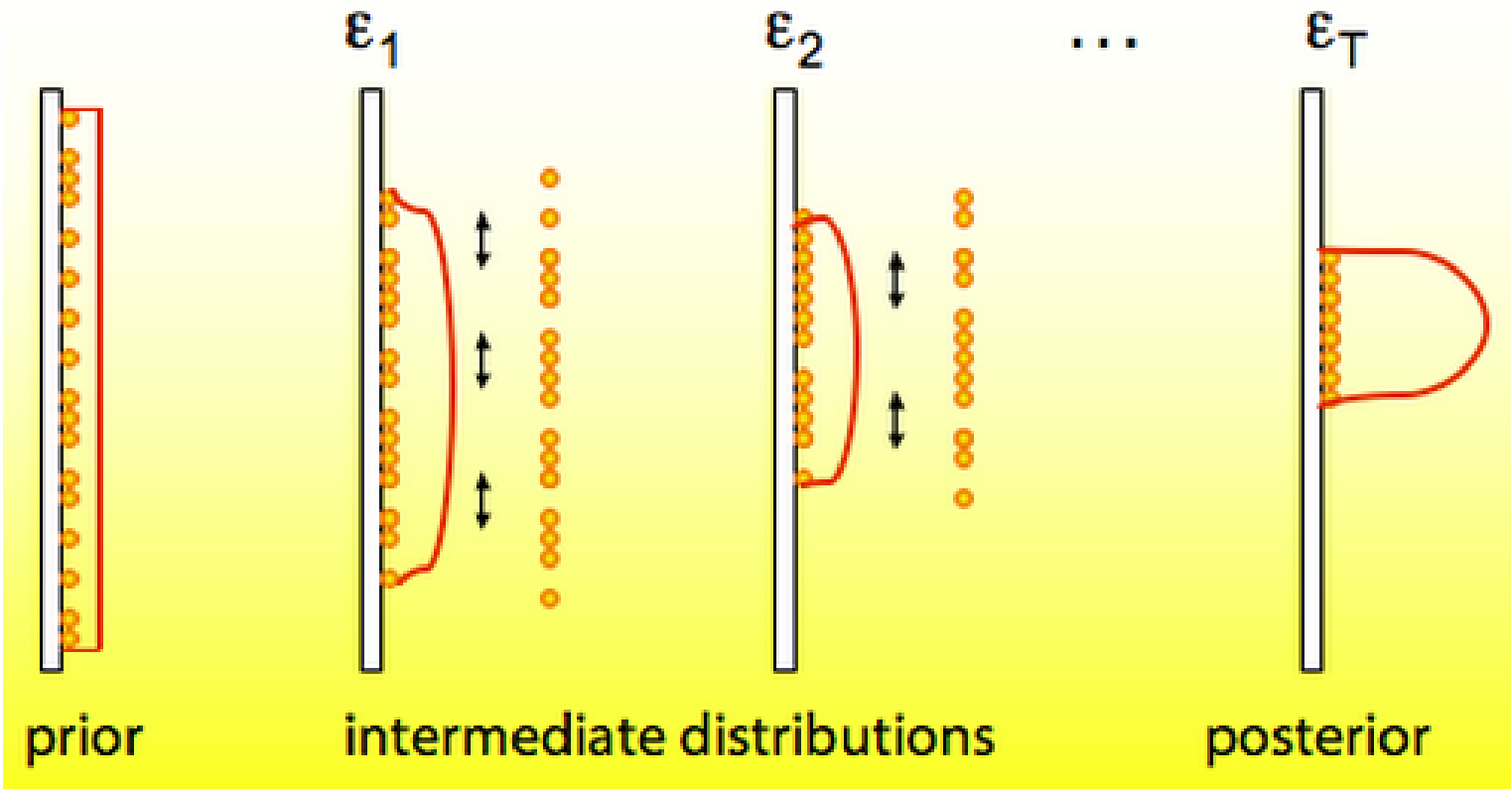
Outline

ABC-SysBio: a PyCUDA-implemented toolkit

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



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

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.