PyCUDA and GPUArray

See `https://documen.tician.de/pycuda/` for more documentation.

Initialization

A few modules have to be loaded to initialize communication to the GPU:

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

The `pycuda.driver` module has methods that:

1. Allocate memory on the GPU (`cuda.mem_alloc()`)
2. Copy numpy arrays to the GPU (`cuda.memcpy_htod()`)
3. Execute kernels on the GPU described by CUDA code (see `compiler.SourceModule`)
4. Copy data from the GPU back to numpy arrays (`cuda.memcpy_dtoh()`).

GPUArrays

The `pycuda.gpuarray.GPUArray` is particularly useful since they can associate a numpy array with an array on the device, handle transfers, and express array operations on the GPU in with numpy array syntax. E.g.:

```
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
```

Elementwise functions (e.g. sqrt()) are available in the module `pycuda.cumath`. Reductions can also be performed, e.g. `pycuda.gpuarray.sum(a)`.

Also the `pycuda.elementwise.ElementwiseKernel` class allows one to put in the relevant snippets of C code. Similarly there one can create a custom *Map Reduce* function. See the documentation for details. Also note that when timing any kernel functions, be sure that the compilation has happened first.

Random numbers

For performance, one should use the GPU optimized random number generators. The XORWOW generator should be good enough:

```
import pycuda.curandom
rg = pycuda.curandom.XORWOWRandomNumberGenerator()
arrayrand = rg.gen_uniform(100, numpy.float32)
```

`arrayrand` is now a GPUArray of random numbers.