A Common GPU n-Dimensional Array for Python and C

Université de Montréal

Frédéric Bastien
Arnaud Bergeron
Andreas Klöckner
Pascal Vincent
Yoshua Bengio

Why do we need this?

- Efficient linear algebra is at the core of many scientific applications
- On the CPU, numpy ndarray provides a standard object (for python at least)

Why a new implementation?

There are already a number of existing GPU computing codebases:
Theano, PyCUDA/PyOpenCL, CUDAmat, Gnumpy, Thrust, ...
But:

1. All are incompatible
2. They do not support the full range of numpy ndarray features
3. None support both CUDA and OpenCL

Features desired

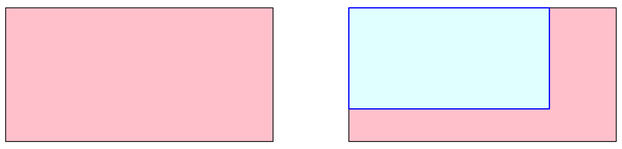
- Support for varying datatypes
- Support for an arbitrary number of dimensions
- Support for strides
- Support for broadcasting
- Compatibility with CUDA and OpenCL

Easy to develop

- Not always a good idea to make a gpu code work for all memory layout.
  - Harder to code
  - Harder to get efficient
- Just call **as_{contiguous,fortran}_memory()** on inputs!

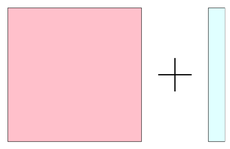
Strides

- Strides is a way to specify how much memory to skip between each element of a dimension.
  - This corresponds to the size of one element times the number of elements
- We can use strides to take submatrix B from A without copying any memory.
  - The strides stay the same but the number of elements in each dimension is reduced

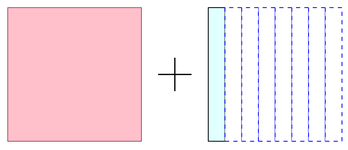


Broadcasting

- We have matrix A (size [8,8]) and we want to add a bias vector b (size [8,1]) to it.
  - This doesn't fit the rules for elementwise operations since both objects do not have the same number of elements.



- So we make virtual copies of b along the last dimension until it has the same size as A.
  - Then we can proceed as usual for elementwise.



Comparison of existing implementations

Package	strides	broadcast	dimensions	types	backends
Theano	yes[a]	yes	any	float32	CUDA
PyCUDA	no	no	any	all	CUDA
PyOpenCL	no	no	any	all	OpenCL
CUDAMat	no	yes[b]	2	float32	CUDA
Gnumpy	no	yes	any	float32[c]	CUDA
Thrust	no	no	1	all	CUDA
Desired	yes	yes	any	all	both

[a] as number of elements
[b] via a function
[c] and a hackish form of boolean

Why has this not been done before?

- Hard and time consuming to get right and efficient
- Certain algorithms cannot work on a general memory layout
- Indexing computations take up a significant portion of time on the GPU

Functionality

What we have

- data types
- dimensions
- strides, views
- broadcasting
- elementwise kernels
- partial reductions
- support for CUDA and OpenCL

Interfaces

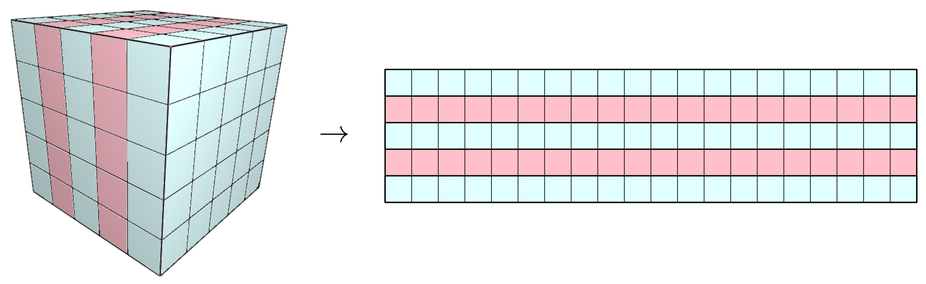
- Python
- C++ interface similar to Numpy C-API (depends on python)

Missing

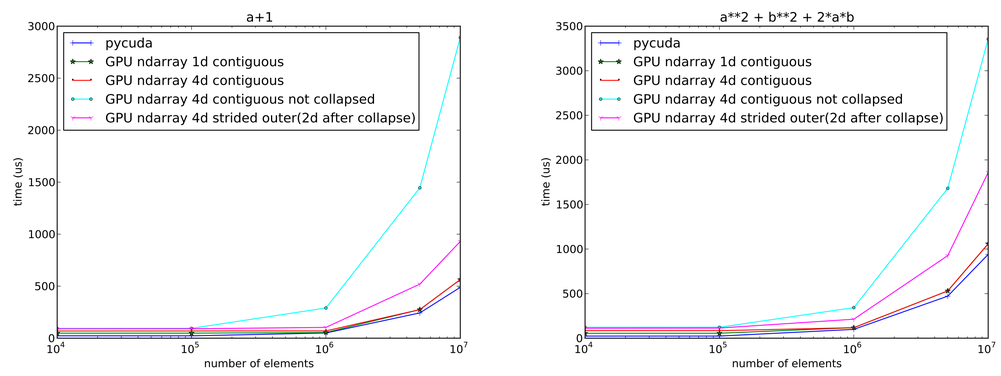
- assignation
- reshaping
- a clean C interface

Element-wise dimension collapsing

- Indexing computations are expensive
- The cost is paid per dimension (irrespective of their size)
- Suppose we have some elementwise work to do on a 3d tensor B that is a view of A, but strided in the innnermost dimension.
  - We can merge the two outer dimensions to obtain an equivalent array that accesses the same memory but with easier indexing.



Benchmarks



Speed for contiguous cases is similar to other implementations.

Future Plans

- Use in Theano/PyOpenCL/PyCUDA
- Design and implement a good C/C++ interface
- Find ways to lower the overhead
- Use the implicit looping provided by CUDA and OpenCL
- World domination!

References

Gpundarray: A common n-dimensional array on the gpu. https://github.com/inducer/compyte/wiki.

Travis E. Oliphant. Python for scientific computing. In *Computing in Science and Engineering*, 9:10-20, 2007.

James Bergstra, Olivier Breleux, Frédéric Bastien, Pascal Lamblin, Rasvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010. Oral Presentation.

More references in the paper.