GTC 2019

CuPy

NumPy compatible GPU library for fast computation in Python

Preferred Networks
Crissman Loomis crissman@preferred.jp
Shunta Saito shunta@preferred.jp

What is CuPy?

CuPy is...

a library to provide NumPy-compatible features with GPU

NumPy

```python
import numpy as np
X_cpu = np.zeros((10,))
W_cpu = np.zeros((10, 5))
y_cpu = np.dot(x_cpu, W_cpu)
```

CuPy

```python
import cupy as cp
x_gpu = cp.zeros((10,))
W_gpu = cp.zeros((10, 5))
y_gpu = cp.dot(x_gpu, W_gpu)
```

```python
y_cpu = cp.asnumpy(y_gpu)
```

```python
y_gpu = cp.asarray(y_cpu)
```

NumPy

CuPy

```python
import numpy as np
X_cpu = np.zeros((10,))
W_cpu = np.zeros((10, 5))
y_cpu = np.dot(x_cpu, W_cpu)
```

```python
import cupy as cp
x_gpu = cp.zeros((10,))
W_gpu = cp.zeros((10, 5))
y_gpu = cp.dot(x_gpu, W_gpu)
```

```python
for xp in [np, cp]:
    x = xp.zeros((10,))
    W = xp.zeros((10, 5))
    y = xp.dot(x, W)
```

Support both CPU and GPU with the same code!

Why develop CuPy? (1)

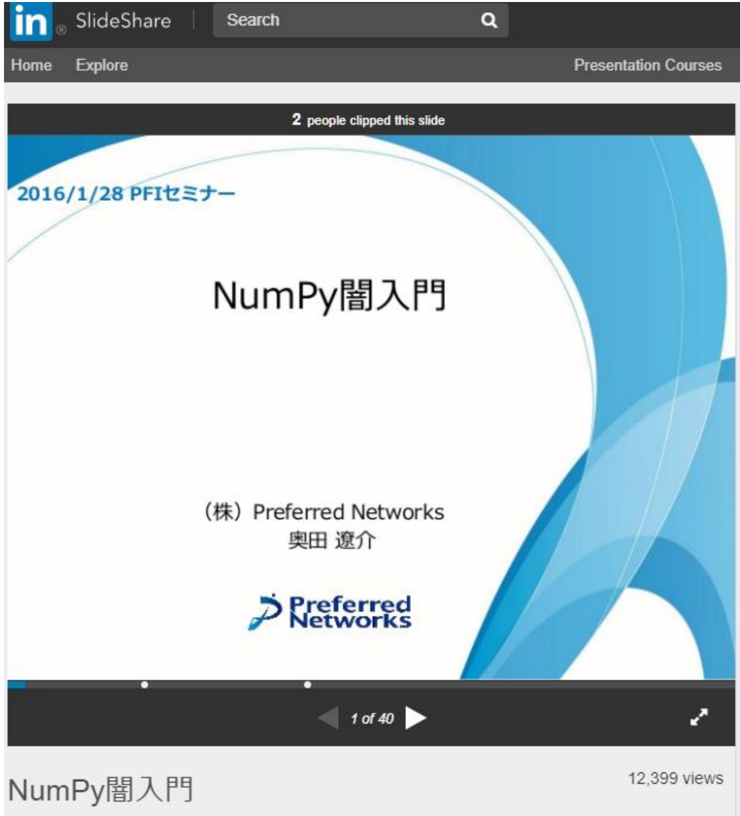
- Chainer functions had separate implementations in NumPy and PyCUDA to support both CPU and GPU

Even writing simple functions like “Add” or “Concat” took several lines...

```
_args = 'const float* x, float* y, int cdimx, int cdimy, int rdim, int coffset'
_preamble = '''
#define COPY(statement) \
    int l   = i / (rdim * cdimx);  \
    int c   = i / rdim % cdimx + coffset;  \
    int r   = i % rdim;  \
    int idx = r + rdim * (c + cdimy * l);  \
    statement;
'''


class Concat(function.Function):

    """Concatenate multiple tensors towards specified axis."""
```

Why develop CuPy? (2)

- Needed a NumPy-compatible GPU array library
  - NumPy is complicated
    - dtypes
    - Broadcast
    - Indexing



https://www.slideshare.net/ryokuta/numpy-57587130

Why develop CuPy? (3)

- There was no convenient library
  - gnumpy
    - Consists of a single file which has 1000 lines of code
    - Not currently maintained
  - CUDA-based NumPy
    - No pip package is provided

⇒**Needed to develop it ourselves**

CuPy was born as a GPU backend of Chainer

CuPy: Add and use a new GPU array backend with NumPy-compatible interface #266

Edit

Merged beam2d merged 305 commits into `master` from `cupy` on 20 Aug 2015

Conversation 6 | Commits 250+ | Checks 0 | Files changed 297 | +15,915 −4,904

beam2d commented on 27 Jul 2015 — Member

This is a large PR aiming at replacing the CUDA array backend from PyCUDA/scikit-cuda to a new one named *CuPy*. This PR includes the implementation of CuPy and updates on Chainer.

Background: PyCUDA is a great wrapper of CUDA that enables us to write our own kernels and call them from Python. However, its GPUArray has few functionalities and almost every time we have to write our own kernels to write down Function implementations. We want to make it easier to write user-defined Functions runnable on GPU. It requires us to use more powerful GPU-array implementations.

Reviewers

No reviews

Assignees

No one—assign yourself

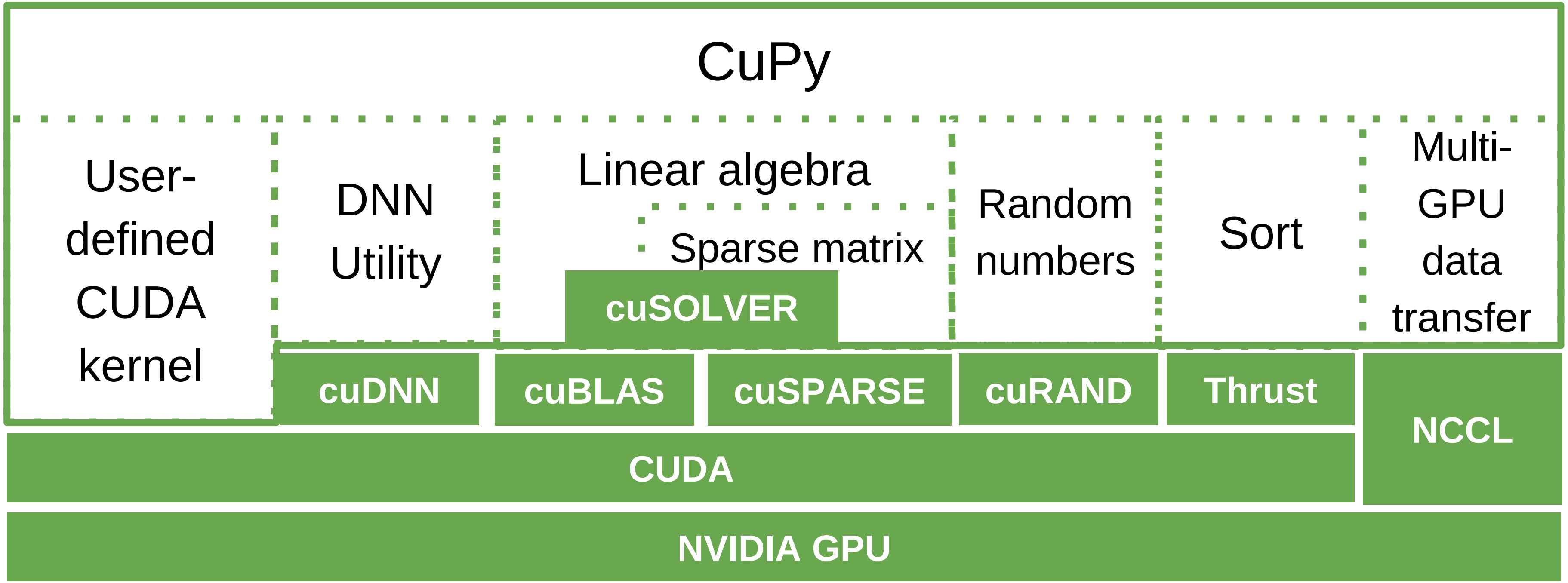
Labels

feature

History of CuPy

2015/6/5	Chainer v1.0	PyCUDA Age
2015/7/?		CuPy development started
2015/9/2	Chainer v1.3	From PyCUDA to CuPy
2017/2/21	CuPy v1.0 a1	CuPy independence day
2018/4/17	CuPy v4.0	Started quarterly releases

Inside CuPy


CuPy
User-defined CUDA kernel
DNN Utility
Linear algebra
Sparse matrix
cuSOLVER
Random numbers
Sort
Multi-GPU data transfer
cuDNN
cuBLAS
cuSPARSE
cuRAND
Thrust
NCCL
CUDA
NVIDIA GPU

NumPy compatible features

- Data types (dtypes)
  - bool_, int8, int16, int32, int64, uint8, uint16, uint32, uint64, float16, float32, float64, complex64, and complex128
- All basic indexing
  - indexing by ints, slices, newaxes, and Ellipsis
- Most of advanced indexing
  - except indexing patterns with boolean masks
- Most of the array creation routines
  - empty, ones_like, diag, etc...
- Most of the array manipulation routines
  - reshape, rollaxis, concatenate, etc...
- All operators with broadcasting
- All universal functions for element-wise operations
  - except those for complex numbers
- Linear algebra functions accelerated by cuBLAS
  - including product: dot, matmul, etc...
  - including decomposition: cholesky, svd, etc...
- Reduction along axes
  - sum, max, argmax, etc...
- Sort operations implemented by Thrust
  - sort, argsort, and lexsort
- Sparse matrix accelerated by cuSPARSE

New features after CuPy v2

- Narrowed the gap with NumPy
- Speedup: Cythonized, Improved MemoryPool
- CUDA Stream support
- Added supported functions
  - From NumPy
  - Sparse Matrix, FFT, scipy ndimage support

Comparison with other libraries

	CuPy	PyCUDA*	Theano	MinPy**
NVIDIA CUDA support	✔	✔	✔	✔
CPU/GPU agnostic coding	✔		✔	✔
Autograd support	***		✔	✔
NumPy compatible Interface	✔			✔
User-defined CUDA kernel	✔	✔		
			2017/11 Halted	2018/2 Halted

* https://github.com/inducer/pycuda
** https://github.com/dmlc/minpy
*** Autograd is supported by Chainer, a DL framework on top of CuPy

Projects exploiting CuPy

Chainer

Deep learning framework
https://chainer.org/

pomegranate

Probabilistic and graphical modeling
https://github.com/jmschrei/pomegranate

spaCy

Natural language processing
https://spacy.io/

OpenCL version of CuPy: ClPy

ClPy: OpenCL backend for CuPy

ClPy is an implementation of CuPy's OpenCL backend. In other words, ClPy enables softwares written in CuPy to work also on OpenCL devices, not only on CUDA (NVIDIA) devices.

Current status

Current ClPy is beta version, forked from CuPy v2.1.0. ClPy is still under development and works on only limited APIs.

- Most of ndarray are supported, but not perfectly
- Most of universal functions are supported, but not perfectly
- All custom kernels are supported.
- All BLAS APIs used by ClPy itself are supported. Other types are currently not.
- Sparse matrix, dnn, rand libraries are not supported
- half and complex are not supported
- Works on only a single device

Where CuPy is headed

- Support GPU in Python code with **minimal changes**
  - High compatibility with other libraries made for CPUs
  - Not only NumPy, but also SciPy etc.
- Enable GPU acceleration with **minimal effort**
  - Easy installation
  - No need for tuning

How to use CuPy

Installation

https://github.com/cupy/cupy#installation

1. Install CUDA SDK
   - If necessary, install cuDNN and NCCL too
2. (Use environment variable CUDA_PATH for custom installation)
   - setup.py of CuPy findS CUDA libraries automatically
3. $ pip install cupy

Pre-built binaries!

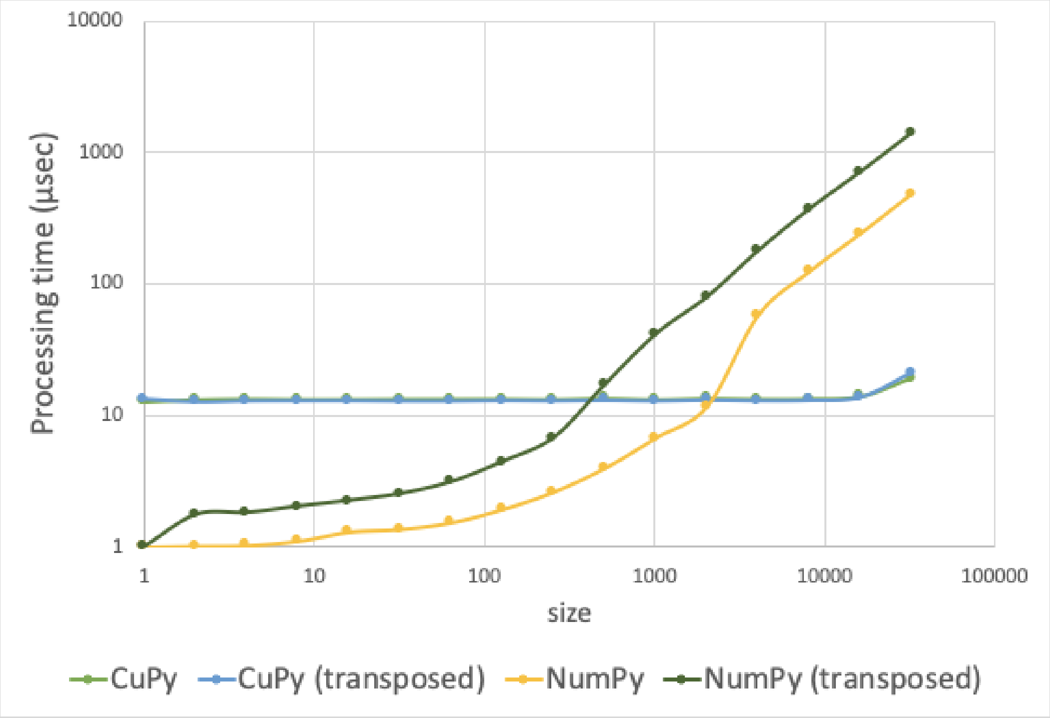
$ pip install cupy-cuda80	(Binary Package for CUDA 8.0)
$ pip install cupy-cuda90	(Binary Package for CUDA 9.0)
$ pip install cupy-cuda91	(Binary Package for CUDA 9.1)
$ pip install cupy-cuda92	(Binary Package for CUDA 9.2)
$ pip install cupy-cuda100	(Binary Package for CUDA 10.0)

cuDNN and NCCL included!

How much faster is CuPy than NumPy? Add funcs

```python
a = xp.ones((size, 32), 'f')
b = xp.ones((size, 32), 'f')

def f():
  a + b
```

```python
# Transpose
a = xp.ones((32, size), 'f').T
b = xp.ones((size, 32), 'f')

def f():
  a + b
```

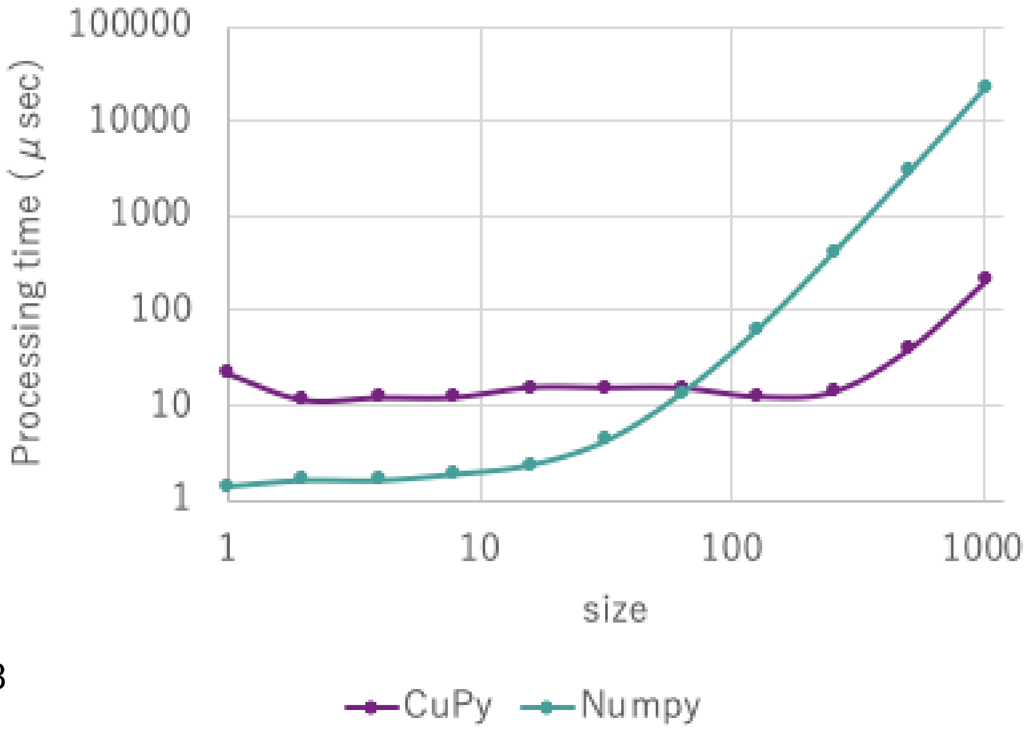


https://github.pfidev.jp/okuta/cupy-bench
Xeon Gold 6154 CPU @ 3.00GHz
Tesla V100-PCIE-16GB

How much faster is CuPy than NumPy? Dot products

```python
a = xp.ones((size, size), 'f')
b = xp.ones((size, size), 'f')

def f():
  xp.dot(a, b)
```

For a rough estimation, if the array size is larger than L1 cache of your CPU, CuPy gets faster than NumPy.



Try on Google Colab! http://bit.ly/cupywest2018

Advanced Features

Preferred Networks
Researcher, Shunta Saito

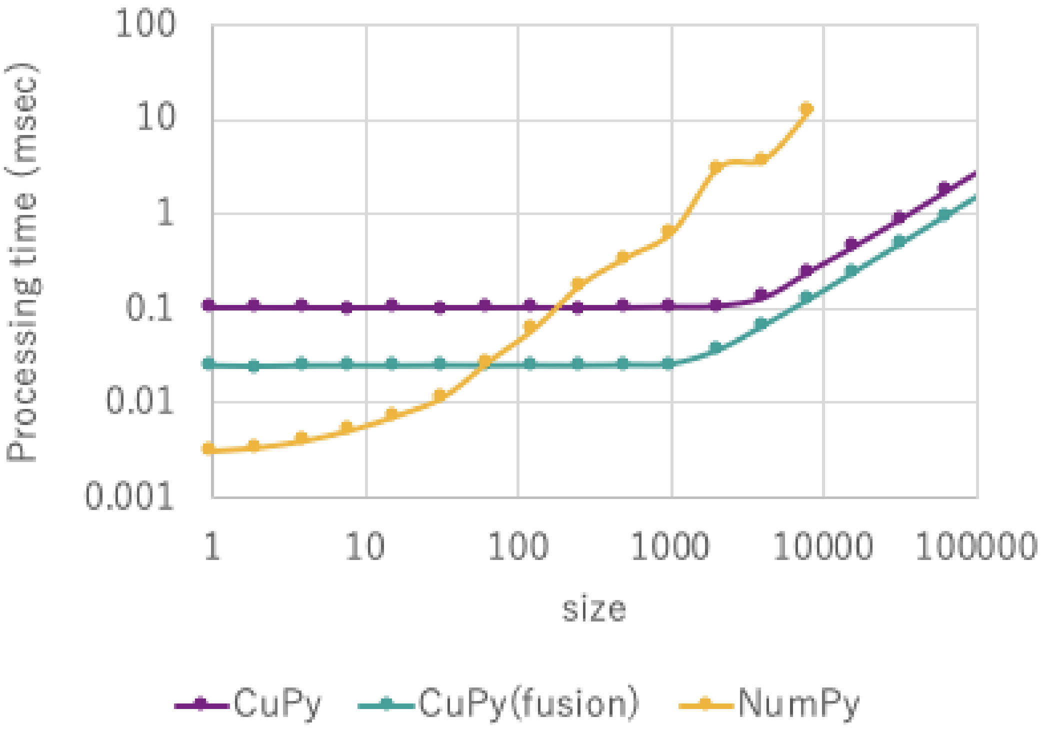
Agenda

- Kernel Fusion
- Unified Memory
- Custom Kernels
- Compatibility with other libraries
  - SciPy-compatible features
  - Direct use of NumPy functions via __array_interface__
  - Numba
  - PyTorch via DLPack
  - cuDF / cuML

Fusion: fuse kernels for further speedup!

```
a = numpy.float32(2.0)
x = xp.ones((1024, size), 'f')
y = xp.ones((1024, size), 'f')

def saxpy(a, x, y):
    return a * x + y

saxpy(a, x, y)  # target
```

```
@cupy.fuse()
def saxpy(a, x, y):
    return a * x + y

saxpy(a, x, y)  # target
```



Advantages of @cupy.fuse()

- Speedup function calls
- Reduce memory consumption
- Relax the bandwidth bottleneck

Limitations of @cupy.fuse()

- Only element-wise and reduction operations are supported
- Other operations like `cupy.matmul()` and `cupy.reshape()` are not yet supported

You want to save GPU memory?

```
import cupy as cp
size = 32768
a = cp.ones((size, size)) # 8GB
b = cp.ones((size, size)) # 8GB
cp.dot(a, b)              # 8GB
```

```
Traceback (most recent call last):
  ...
cupy.cuda.memory.OutOfMemoryError: out of memory to
allocate 8589934592 bytes (total 17179869184 bytes)
```

Try Unified Memory! (Supported only on V100)

- Just edit 2 lines to enable **unified memory**

```
import cupy as cp

pool = cp.cuda.MemoryPool(cp.cuda.malloc_managed)
cp.cuda.set_allocator(pool.malloc)

size = 32768
a = cp.ones((size, size)) # 8GB
b = cp.ones((size, size)) # 8GB
cp.dot(a, b)              # 8GB
```

Custom Kernels

- CuPy provides classes to compile **your own CUDA kernel**:
  - ElementwiseKernel
  - ReductionKernel
  - **RawKernel (from v5)**
    - For CUDA experts who love to write everything by themselves
    - Compiled with NVRTC

Basic usage of ElementwiseKernel

```
squared_diff = cp.ElementwiseKernel(
    'float32 x, float32 y',   # input params
    'float32 z',              # output params
    'z = (x - y) * (x - y)',  # element-wise operation
    'squared_diff'            # the name of this kernel
)

x = cp.arange(10, dtype=np.float32).reshape(2, 5)
y = cp.arange(5, dtype=np.float32)

squared_diff(x, y)
```

Type-generic kernels

```
squared_diff_generic = cp.ElementwiseKernel(
    'T x, T y',                  # input params
    'T z',                       # output params
    'z = (x - y) * (x - y)',     # element-wise operation
    'squared_diff'               # the name of this kernel
)

x = cp.arange(10, dtype=np.float32).reshape(2, 5)
y = cp.arange(5, dtype=np.float32)

squared_diff_generic(x, y)
```

Type-generic kernels

```
squared_diff_generic = cp.ElementwiseKernel(
    'T x, T y',
    'T z',
    '''
        T diff = x - y;
        z = diff * diff;
    ''',
    'squared_diff_generic')

x = cp.arange(10, dtype=np.float32).reshape(2, 5)
y = cp.arange(5, dtype=np.float32)

squared_diff_generic(x, y)
```

Manual indexing with raw specifier

```
add_reverse = cp.ElementwiseKernel(
    'T x, raw T y',                    # input params
    'T z',                             # output params
    'z = x + y[_ind.size() - i - 1]',  # element-wise operation
    'add_reverse'                      # the name of this kernel
)

x = cp.arange(5, dtype=np.float32)
y = cp.arange(5, dtype=np.float32)


add_reverse(x, y)
```

=> **This is same as : `x + y[::-1]`**

Reduction Kernel

```
l2norm_kernel = cp.ReductionKernel(
    'T x',           # input array
    'T y',           # output array
    'x * x',         # map
    'a + b',         # reduce
    'y = sqrt(a)',   # post-reduction map
    '0',             # identity value
    'l2norm'         # kernel name
)
x = cp.arange(1000, dtype=np.float32).reshape(20, 50)
l2norm_kernel(x, axis=1)
```

=> This is same as : `cp.sqrt((x * x).sum(axis=1))` **but much faster!**

How a RawKernel looks...

```
import cupy as cp

square_kernel = cp.RawKernel(r'''
extern "C" __global__ void my_square(long long* x) {
    int tid = threadIdx.x;
    x[tid] *= x[tid];
}
''', name='my_square')

x = cp.arange(5)
square_kernel(grid=(1,), block=(5,), args=(x,))
print(x)  # [ 0  1  4  9 16]
```

SciPy-compatible features: ndimage

Interpolation

`cupyx.scipy.ndimage.affine_transform`	Apply an affine transformation.
`cupyx.scipy.ndimage.map_coordinates`	Map the input array to new coordinates by interpol
`cupyx.scipy.ndimage.rotate`	Rotate an array.
`cupyx.scipy.ndimage.shift`	Shift an array.
`cupyx.scipy.ndimage.zoom`	Zoom an array.

OpenCV mode

`cupyx.scipy.ndimage` supports additional mode, `opencv`. If it is given, the function performs like cv2.warpAffine or cv2.resize.

SciPy-compatible features: scipy.sparse

Functions

Building sparse matrices

cupyx.scipy.sparse.eye	Creates a sparse matrix with ones on diagonal.
cupyx.scipy.sparse.identity	Creates an identity matrix in sparse format.
cupyx.scipy.sparse.spdiags	Creates a sparse matrix from diagonals.
cupyx.scipy.sparse.rand	Generates a random sparse matrix.
cupyx.scipy.sparse.random	Generates a random sparse matrix.

Identifying sparse matrices

cupyx.scipy.sparse.issparse	Checks if a given matrix is a sparse matrix.
cupyx.scipy.sparse.isspmatrix	Checks if a given matrix is a sparse matrix.
cupyx.scipy.sparse.isspmatrix_csc	Checks if a given matrix is of CSC format.
cupyx.scipy.sparse.isspmatrix_csr	Checks if a given matrix is of CSR format.
cupyx.scipy.sparse.isspmatrix_coo	Checks if a given matrix is of COO format.
cupyx.scipy.sparse.isspmatrix_dia	Checks if a given matrix is of DIA format.

Linear Algebra

cupyx.scipy.sparse.linalg.lsqr	Solves linear system with QR decomposition.

Sparse matrix classes

cupyx.scipy.sparse.coo_matrix	COOrdinate format sparse matrix.
cupyx.scipy.sparse.csc_matrix	Compressed Sparse Column matrix.
cupyx.scipy.sparse.csr_matrix	Compressed Sparse Row matrix.
cupyx.scipy.sparse.dia_matrix	Sparse matrix with DIAgonal storage.
cupyx.scipy.sparse.spmatrix	Base class of all sparse matrixes.

Use CuPy with Numba!

```
import cupy as cp
from numba import cuda

@cuda.jit
def square(x):
    start = cuda.grid(1)
    stride = cuda.gridsize(1)
    for i in range(start, len(x), stride):
        x[i] **= 2

a = cp.arange(5)
square[1, 32](a)

print(a)  # => [ 0  1  4  9 16]
```

NumPy’s `__array_interface__` support

- From CuPy v6.0.0 beta 2, **you can pass a CuPy ndarray directory to NumPy functions!**

```
import numpy
import cupy

x = cupy.random.rand(10)  # CuPy array!
numpy.sum(x)  # Pass to a NumPy function!

# => array(4.5969301)
```

DLpack support

You can convert PyTorch tensors to CuPy ndarrays **without any memory copy thanks to DLPack**, and vice versa.

PyTorch Tensor -> CuPy array

```
import torch
import cupy
from torch.utils.dlpack import to_dlpack

tx = torch.randn(3).cuda()  # Create a PyTorch tensor
t1 = to_dlpack(tx)  # Convert it into a dlpack tensor

# Convert it into a CuPy array
cx = cupy.fromDlpack(t1)
```

DLpack support

You can convert PyTorch tensors to CuPy ndarrays **without any memory copy thanks to DLPack**, and vice versa.

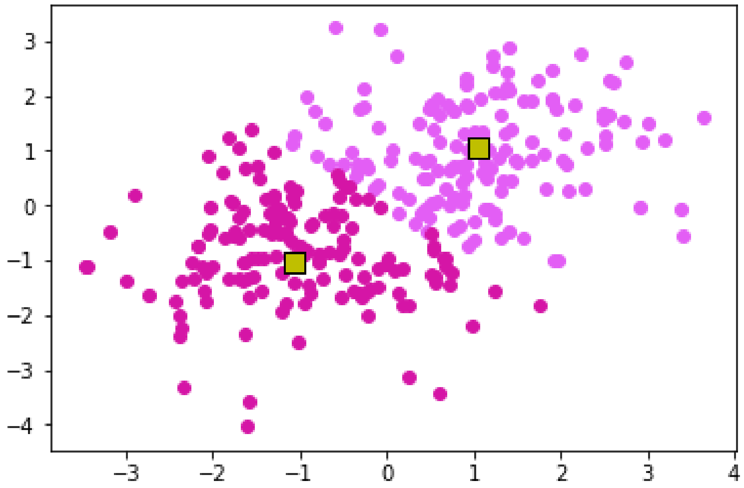
<u>**CuPy array -> PyTorch Tensor**</u>

```
import torch
import cupy
from torch.utils.dlpack import from_dlpack

# Create a CuPy array
ca = cupy.random.randn(3).astype(cupy.float32)
t2 = ca.toDlpack()  # Convert it into a dlpack tensor

cb = from_dlpack(t2)  # Convert it into a PyTorch tensor!
```

cuDF / cuML compatibility (From cuDF v0.6~)

```python
import cupy
import cudf
import cuml

# Input data preparation
samples = np.random.randn(5000000, 2)
X = np.r_[samples + 1, samples - 1]

# Create CuPy ndarray
X_cp = cupy.asarray(X, order='F')

# Convert to cuDF DataFrame
X_df = cudf.DataFrame(
  [(str(i), cudf.from_dlpack(xi.toDlpack()))
   for i, xi in enumerate(X_cp.T)])
```

```python
from cuml import KMeans

kmeans = KMeans(n_clusters=2, n_gpu=1)
kmeans.fit(X_df)
```



Future of CuPy

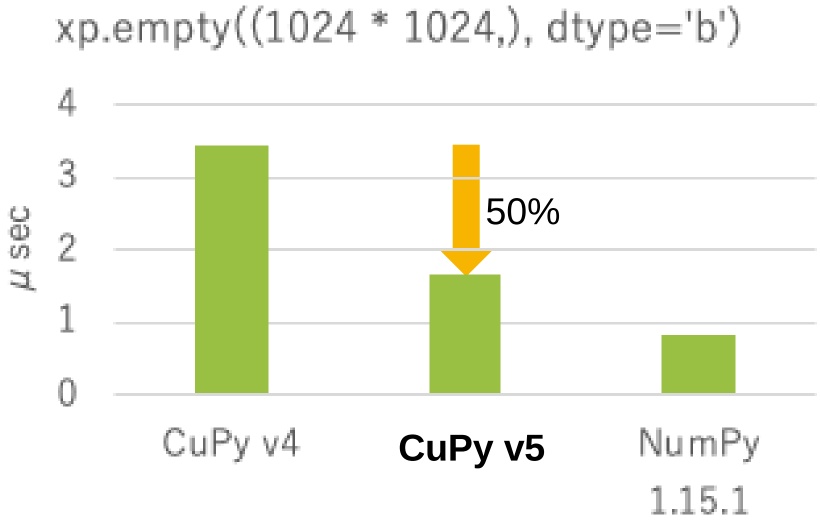
Future development plans

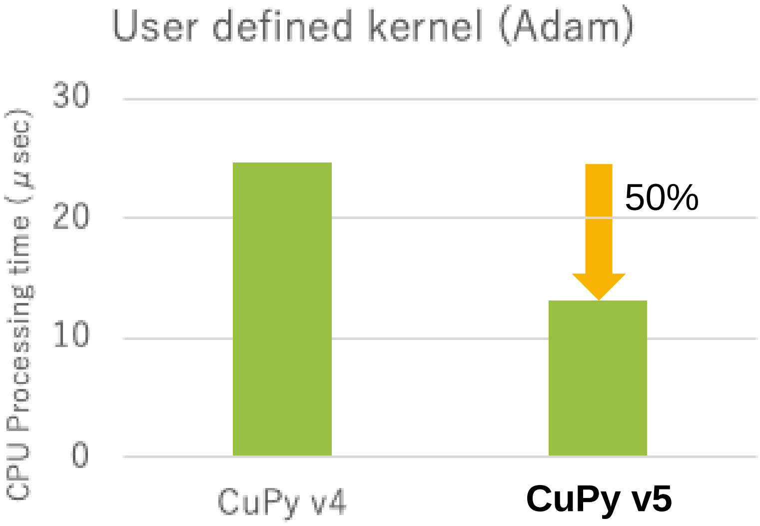
- ~~[v5] @cupy.fusion()~~
- ~~[v5] Raw CUDA Kernel (it replaces PyCUDA)~~
- ~~[v5] Adding more compatibility: Numba, DLPack~~
- ~~[v5] Windows support~~
- [v6] Adding more functions
- [v6] Improve memory allocation
- [v6] Speed-up kernel call
- [v6] Support more various GPUs
- ([?] CUDA Graphs support?)

DONE

Steady efforts increased speed

- How we can go closer to NumPy when allocating an array on GPU?





Any feedback is welcome!

- What do you use CuPy for?
- How do you use CuPy?
- What features of CuPy do you want?
- What part of CuPy do you want us to improve?

Github Issue: https://github.com/cupy/cupy/issues

#general-cupy channel in the official Slack team:

https://bit.ly/join-chainer-slack

Dear CuPy users...

- Please let the NVIDIA developers and GPU technologists know the fact that you are using CuPy.
  - NVIDIA will become to support CuPy development further
- If you developed a software using CuPy, please let us know!
  - We are making a list of softwares using CuPy:
    https://github.com/cupy/cupy/wiki/Projects-using-CuPy

CuPy : NumPy-like API accelerated with CUDA
(cuBLAS, cuDNN, cuRAND, cuSOLVER, cuSPARSE, cuFFT, Thrust, NCCL)

Install : $ pip install cupy-cuda**100**
(replace **100** with your CUDA ver. e.g., **92** for CUDA 9.2)

Web : https://cupy.chainer.org/

Github : https://github.com/cupy/cupy/

Example : https://github.com/cupy/cupy/tree/master/examples

Forum : https://groups.google.com/forum/#!forum/cupy

Slack : https://bit.ly/chainer-slack => Join #general-cupy channel

Please join us and accelerate CuPy development!