gpyfft Documentation

Release 0.1

Gregor Thalhammer

June 06, 2012

CONTENTS

CHAPTER

ONE

GPYFFT

A Python wrapper for the OpenCL FFT library APPML/clAmdFft from AMD

1.1 Introduction

AMD has created a nice FFT library for use with their OpenCL implementation called AMD Accelerated Parallel Processing Math Libraries This C library is available as precompiled binaries for Windows and Linux platforms. It is optimized for AMD GPUs. (Note: This library is not limited to work only with hardware from AMD, but according to this forum entry it currently yields wrong results on NVidia GPUs.)

This python wrapper is designed to tightly integrate with pyopencl. It consists of a low-level cython based wrapper with an interface similar to the underlying C library. On top of that it offers a high-level interface designed to work on data contained in instances of pyopencl.array.Array, a numpy work-alike array class. The high-level interface is similar to that of pyFFTW, a python wrapper for the FFTW library.

Compared to pyfft, a python implementation of Apple's FFT library, AMD's FFT library offers some additional features such as transform sizes that are powers of 2,3 and 5, and real-to-complex transforms. And on AMD hardware a better performance can be expected, e.g., gpyfft: 280 Gflops compared to pyfft: 63 GFlops (for single precision, accurate math, inplace, transform size 1024x1024, batch size 4, on AMD Cayman, HD6950).

1.2 Status

This wrapper is currently under development.

1.2.1 work done

- low level wrapper (mostly) completed
- high level wrapper: complex (single precision), interleaved data, in and out of place (some tests and benchmarking available)
- creation of pyopencl Events for synchronization

1.2.2 missing features

- debug mode to output generated kernels
- documentation for low level wrapper (instead refer to library doc)

- define API for high level interface
- high level interface: double precision data, planar data, real<->complex transforms
- high level interface: tests for non-contiguous data
- handling of batched transforms in the general case, e.g. shape (4,5,6), axes = (1,), i.e., more than one axes where no transform is performed. (not always possible with single call for arbitrary strides, need to figure out when possible)

1.3 Requirements

- python
- pyopencl (git version newer than 4 Jun 2012)
- cython
- APPML clAmdFft 1.8
- AMD APP SDK

1.4 Installation

1. Install the AMD library:
    - install clAmdFft
    - add clAmdFft/binXX to PATH, or copy clAmdFft.Runtime.dll to package directory
    - edit setup.py to point to clAmdFft and AMD APP directories

Then, either:

2. python setup.py install

Or:

3. inplace build: python setup.py build_ext –inplace

1.5 License:

LGPL

1.6 Tested Platforms

OS	Python	AMD APP	OpenCL	Device	Status
Win7 (64bit)	2.7, 64bit	2.7	OpenCL 1.2, Catalyst 12.4	AMD Cayman (6950)	works!
Win7 (64bit)	2.7, 32bit	2.7	OpenCL 1.1 AMD-APP-SDK-v2.4 (595.10)	Intel i7	works!
Win7 (64bit)	2.7, 32bit	2.7	OpenCL 1.1 (Intel)	Intel i7	works!
Win7 (64bit)	2.7, 32bit	2.7	OpenCL 1.0 CUDA 4.0.1 (NVIDIA)	Quadro 2000M	Fails
Win7 (64bit)	2.7, 32bit	2.7	OpenCL 1.2 AMD-APP (923.1)	Tahiti (7970)	works!
Win7 (64bit)	2.7, 32bit	2.7	OpenCL 1.2 AMD-APP (923.1)	AMD Phenom IIx4	works!

3. Gregor Thalhammer 2012

CHAPTER

TWO

BUILDING GPYFFT

Here will be detailed instructions for building gpyfft from source.

CHAPTER

THREE

GPYFFT CLASS STRUCTURE

3.1 GpyFFT

class `gpyfft.GpyFFT`

The GpyFFT object is the primary interface to the AMD FFT library

Methods

`get_version()`

returns the version of the underlying AMD FFT library

Returns: A tuple with the major, minor, and patch level of the AMD FFT library.

example: (1L, 8L, 214L)

Raises: GpyFFT_Error: An error occurred accessing the clAmdFftGetVersion function

`create_plan()`

creates an FFT plan based on the dimensionality of the input data

Args: context (object) : a PyOpenCL Context object shape (tuple) : the dimensionality of the input data

Kwargs: None

Returns: Plan (object) : a gpyfft.Plan object

Raises: None

`foo1()`

A one-line summary that does not use variable names or the function name.

(This is a numpy style doc string)

Several sentences providing an extended description. Refer to variables using back-ticks, e.g. *var*.

Parameters **var1** : array_like

Array_like means all those objects – lists, nested lists, etc. – that can be converted to an array. We can also refer to variables like *var1*.

var2 : int

The type above can either refer to an actual Python type (e.g. `int`), or describe the type of the variable in more detail, e.g. `(N,) ndarray` or `array_like`.

Long_variable_name : {'hi', 'ho'}, optional

Choices in brackets, default first when optional.

Returns **describe** : type

Explanation

output : type

Explanation

tuple : type

Explanation

items : type

even more explaining

Other Parameters **only_seldom_used_keywords** : type

Explanation

common_parameters_listed_above : type

Explanation

Raises **BadException** :

Because you shouldn't have done that.

See Also:

`otherfunc` relationship (optional)

`newfunc` Relationship (optional), which could be fairly long, in which case the line wraps here.

`thirdfunc`, `fourthfunc`, `fifthfunc`

Notes

Notes about the implementation algorithm (if needed).

This can have multiple paragraphs.

You may include some math:

References

Cite the relevant literature, e.g. [R1]. You may also cite these references in the notes section above.

[R1]

Examples

These are written in doctest format, and should illustrate how to use the function.

```
>>> a=[1,2,3]
>>> print [x + 3 for x in a]
[4, 5, 6]
>>> print "a\n\nb"
a
b
```

fn_with_sphinxy_docstring()
This function does something.

Parameters

- **name** (*str.*) – The name to use.
- **state** (*bool.*) – Current state to be in.

Returns int – the return code.

Raises AttributeError, KeyError

3.2 Plan

class gpyfft.**Plan**
The Plan object gathers information about the desired transforms and about the underlying OpenCL implementation and performs the bake operation and generates OpenCL kernels

Methods

__init__
Instantiates a Plan object.

Plan objects are created internally by gpyfft; normally a user does not create these objects

Args: context (object) : a PyOpenCL Context object shape (tuple) : the dimensionality of the input data lib (not sure) : not sure what this is

Kwargs: None

Raises: None

precision
the floating point precision of the FFT data

scale_forward
the scaling factor to be applied to the FFT data for forward transforms

scale_backward
the scaling factor to be applied to the FFT data for backward transforms

batch_size
the number of discrete arrays that this plan can handle concurrently

shape
the length of each dimension of the FFT

strides_in
the distance between consecutive elements for input buffers in a dimension

strides_out
the distance between consecutive elements for output buffers in a dimension

distances
the distance between array objects

layouts
the expected layout of the output buffers

inplace
determines if the input buffers are going to be overwritten with results (True == inplace, False == out of place)

temp_array_size
the buffer size (in bytes), which may be needed internally for an intermediate buffer

transpose_result
the final transpose setting of a multi-dimensional FFT

bake()
Prepare the plan for execution

After all plan parameters are set, the client has the option of "baking" the plan, which tells the runtime no more changes to the plan's parameters are expected, and the OpenCL kernels are to be compiled. This optional function allows the client application to perform this function when the application is being initialized instead of on the first execution. At this point, the clAmdFft runtime applies all implemented optimizations, possibly including running kernel experiments on the devices in the plan context.

Args: queues (not sure) : not sure

Kwargs: None

Returns: None

Raises: gpyfft.GpyFFT_Error : clAmdFftBakePlan returned an error

enqueue_transform()
Enqueue an FFT transform operation, and either return immediately, or block waiting for events.

This transform API is specific to the interleaved complex format, taking an input buffer with real and imaginary components paired together, and outputting the results into an output buffer in the same format.

Args: queues (not sure) : not sure in_buffers (?) : not sure

Kwargs: out_buffers (?) : not sure direction_forward (bool) : not sure wait_for_events (bool) : not sure temp_buffer (?) : not sure

Returns: tuple of event objects?

Raises: gpyfft.GpyFFT_Error : clAmdFftEnqueueTransform returned an error

3.3 GpyFFT_Error

class gpyfft.**GpyFFT_Error**
Exception wrapper for errors returned from underlying AMD library calls

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[R1] O. McNoleg, “The integration of GIS, remote sensing, expert systems and adaptive co-kriging for environmental habitat modelling of the Highland Haggis using object-oriented, fuzzy-logic and neural-network techniques,” Computers & Geosciences, vol. 22, pp. 585-588, 1996.

PYTHON MODULE INDEX

g

PYTHON MODULE INDEX

g