PyCBC: a device independent approach to CBC analysis

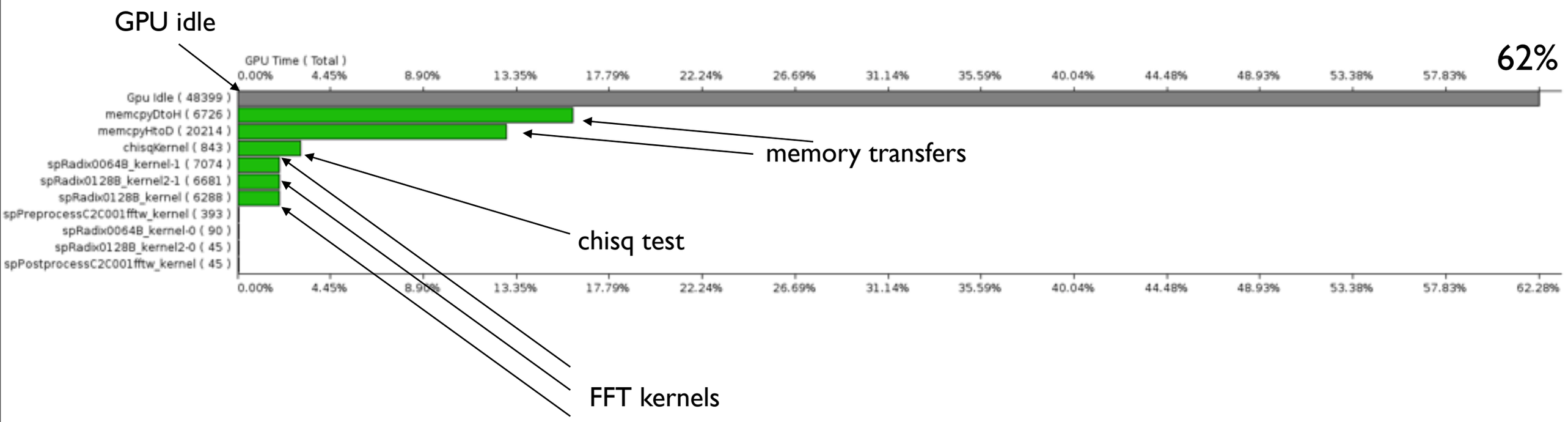
Duncan Brown[1], Gergely Debreczeni[2], Drew Keppel[3], Badri Krishnan,[3]
Adam Mercer[4], Alex Nitz,[1] Bence Somhegyi[2], Karsten Wiesner[3], Josh Willis[5]

[1]Syracuse University, [2]MTA KFKI RMKI Budapest, [3]Albert Einstein Institute Hannover,
[4]University of Wisconsin-Milwaukee, [5]Abilene Christian University

Aug - 9 - 2011

Previous work

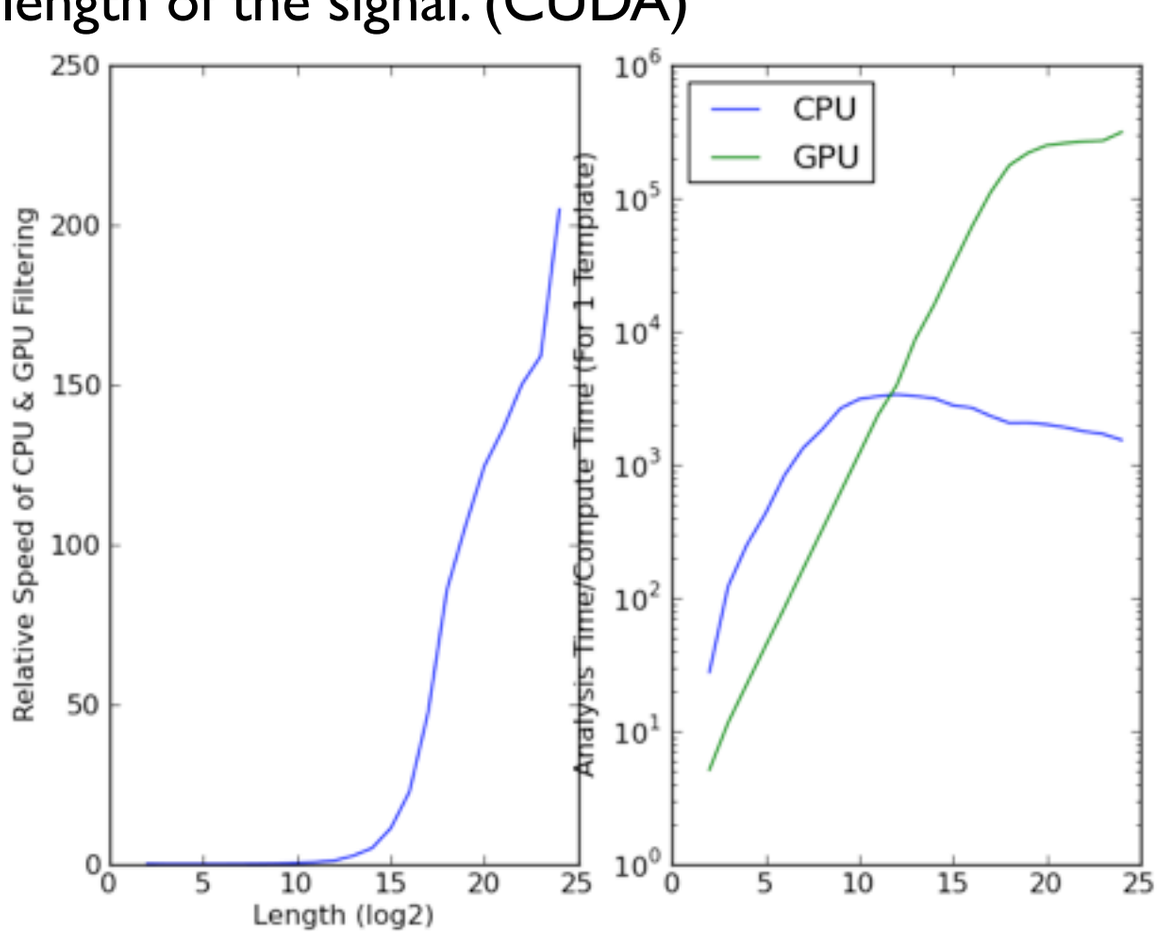
- Acceleration of lalapps_inspiral by GPUs obtained a **speedup factor of 15** by performing all FFTs and the chi-squared test on a GPU. (CUDA) https://dcc.ligo.org/cgi-bin/private/DocDB/ShowDocument?docid=40432
- Results of profiling: largest portions of time spent...
  - transferring memory to and from the GPU
  - running computations on the CPU (GPU is idle)
- **Conclusion:** a redesign focused on transparent GPU acceleration is needed



Aug - 9 - 2011

Preliminary Acceleration Investigations

- To optimize this, we need to implement more of the search engine (i.e., the whole hot loop) on the GPU.
  - The Budapest group has shown a possible **speedup factor of 92** by porting the search engine completely to the GPU. (OpenCl) http://www.grid.kfki.hu/twiki/bin/view/RmkiVirgo/GPU_inspiral
  - The Syracuse group has shown even large gains (**speedup factor of 200**) are possible depending on the length of the signal. (CUDA)



Coding for the future

- Speed not the only issue
- We want something sustainable and that can bring us into the Adv. Detector Era
- Targeting offline compute intensive searches

Aug - 9 - 2011

Requirements of a novel software framework for CBC analysis

- Design Requirements:
  - flexible (easily reconfigurable from the top layer)
    - decouple algorithm from implementation
  - transparent processing on different architectures
  - transparent memory management
  - better readability

- Code Requirements:
  - modularity and simple API
    - simple processing python objects (object oriented design)
    - new features are encapsulated in new objects or extensions of existing objects
    - robustness from unit tests
  - bindings to 3rd party libraries (e.g., lalsuite, fftw, numpy, etc.)
  - auto documentation (epydoc)

Aug - 9 - 2011

Inspiral processing graphs

- Planned how this code will look based on lalapps_inspiral
  - in particular where different processes will be run
- Created processing graphs
  - Shows how functions and kernels are called and where they live (layers)
  - Shows how data flows
  - Shows memory objects
    - Where they live
    - Who is the owner
    - Which kind of data do they hold
    - How long they live

Pycbc signal processing graph for single detector search pipeline using GPUs

Start

Python

StrainData

Injector

Resampler
16kHz -> 4096Hz

Highpass Filter

Straindata.
Convert()

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

clayer

real_vector_double_t s(t)

real_vector_single_t s(t)

GPU

libraries
(lal etc.)

frame_cpp

NRWaveInject

files

.gwf

.xml

Legend:

Data flow

Object destroyed

Function call

Composition (right side object
belongs to left side object)

Software layer

t

real_vector_single_t

Time domain data

real_vector_single_t

Frequency domain data

Pycbc signal processing graph for single detector search pipeline using GPUs

Python

[typ. 10^3] ParameterSpace

Overwhiten render()

Straindata. Segmenting()

Overwhitening Filter

Straindata *= Overw.-Filt

TemplateBank

TemplateBank. PreConditionData()

Straindata. render()

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

_vector_single_t s(t)

[typ. 15 segments freq data from typ 256s timedomain data -> (N=2^20)] complex_vector_single_t stilde(f) (Python list of datavector_cpu)

FFTs of overlapping input data

real_vector_single_t 1/Sn(f) alias inverse PSD alias overwhitening filter N=2^20

median - inverse

__mul()

render

render

IFFT - truncate - FFT (inplace)

clay

GPU

[ ] complex_vector_single_t stilde(f)

real_vector_single_t 1/Sn(f)

GLUE/ Lal

GLUE XMLread

files

.xml

Pycbc signal processing graph for single detector search pipeline using GPUs

Python

GPU

clayer

```
for template in bank:
```

```
for stilde in data:
```

[ ] mParameterSpace

TemplateBank

ChisqVeto

MatchedFilter

If(max>thresh)

ChisqVeto

SingleDetector Event

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

[] complex_vector_single_t stilde(f) (Python list of datavector_cuda)

real_vector_single_t 1/Sn(f) (N=2^20)

Generate_filter_waveform

complex_vector_single_t htilde

Compute_freq_bins

complex_vector_single_t qtilde bin (snr tilde)

Generate_snr

Find_maximum

Generate_chisq

Find_Events

complex_vector_single_t qtilde (snr tilde)

real_vector_single_t (chisq)

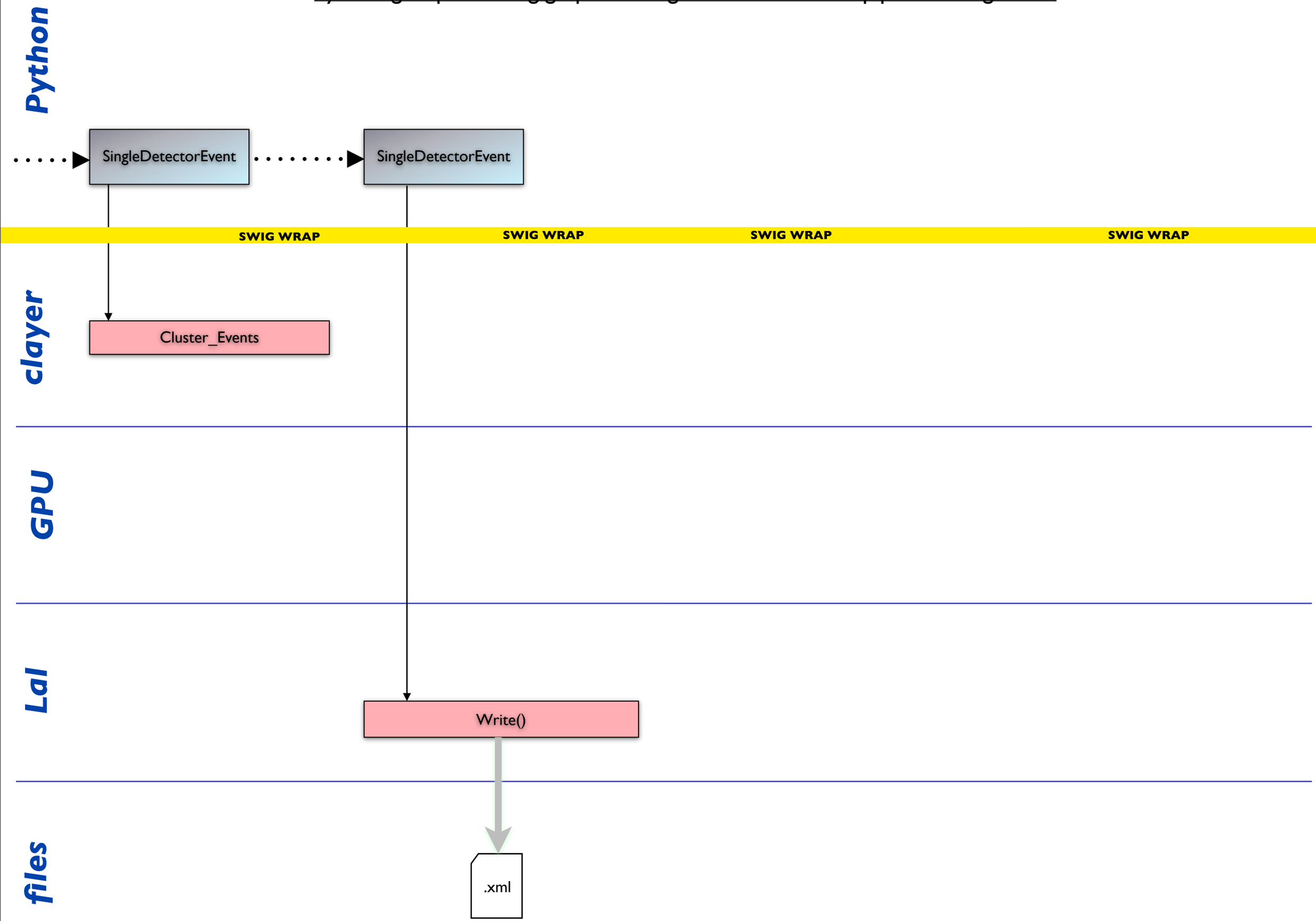
FFT

real_vector_single_t q (snr)

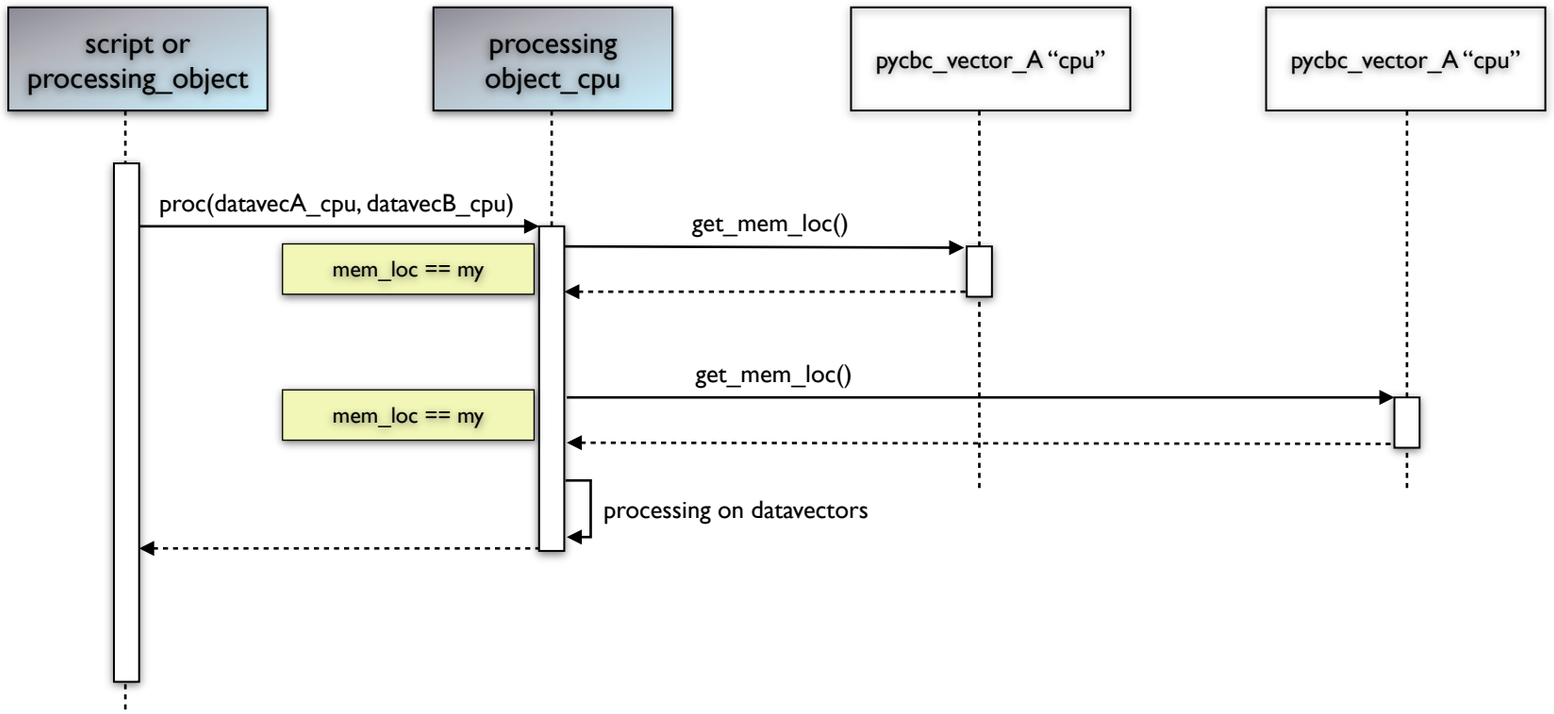
```
# filter the data against the template bank
for template in bank:
    bank.generate_filter_waveform(template)
    chisq.compute_frequency_bins(template.htilde, gpu_data.overwhitenf)
    for stilde in gpu_data.strain:
        matched_filter.perform_generate_snr(snr, stilde, htilde)
        if matched_filter.max() > 5.5:
            chisq = chisq_veto.generate_chisq(snr, qtilde)
            events.find_events(snr, chisq)

# cluster the events across the template bank
events.cluster_events(0.1)
events.write( "/path/to/output.xml" )
```

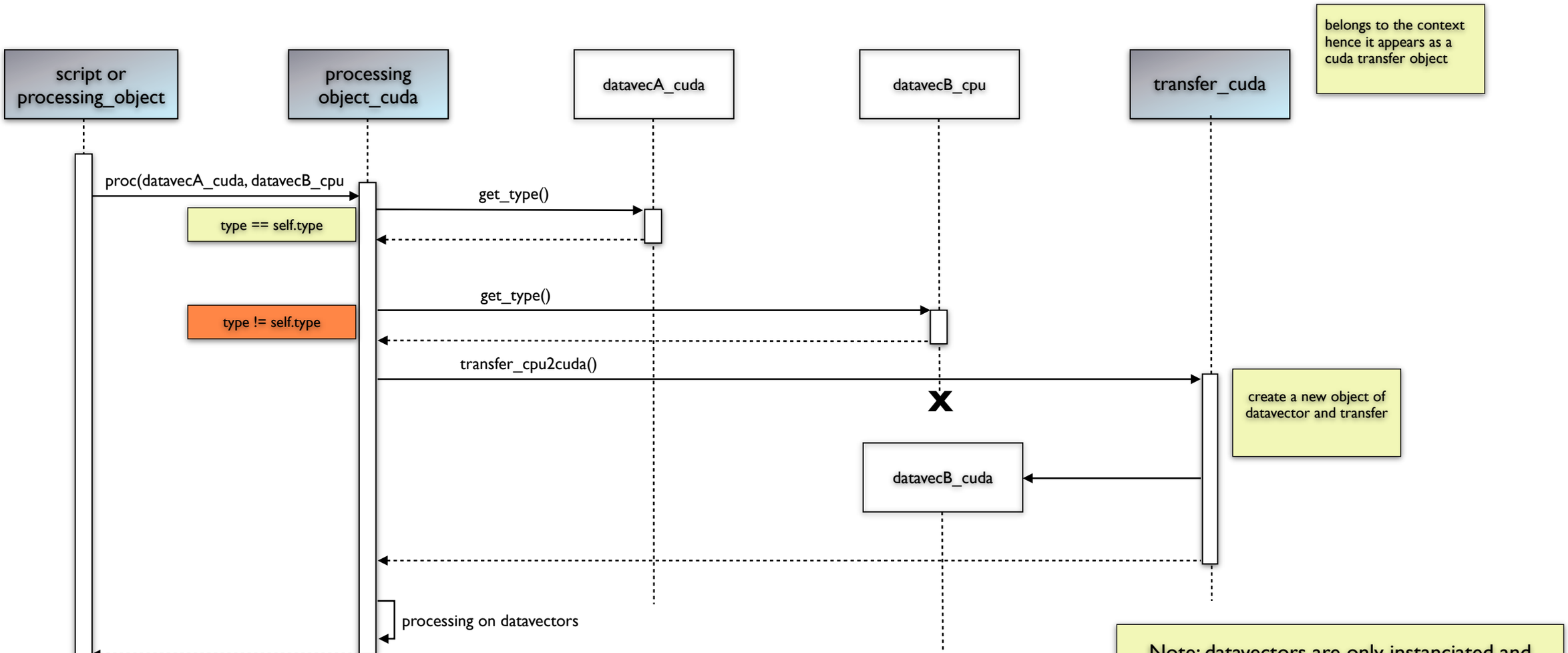
Pycbc signal processing graph for single detector search pipeline using GPUs



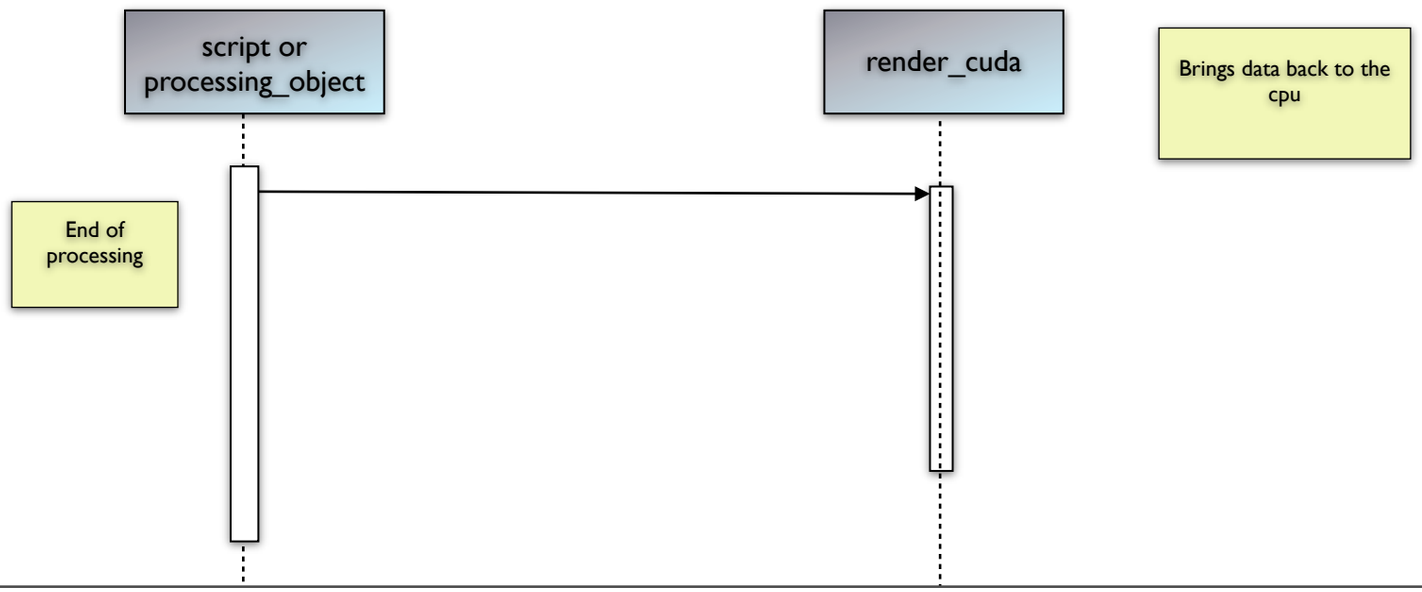
Device memory flow to and from cpu



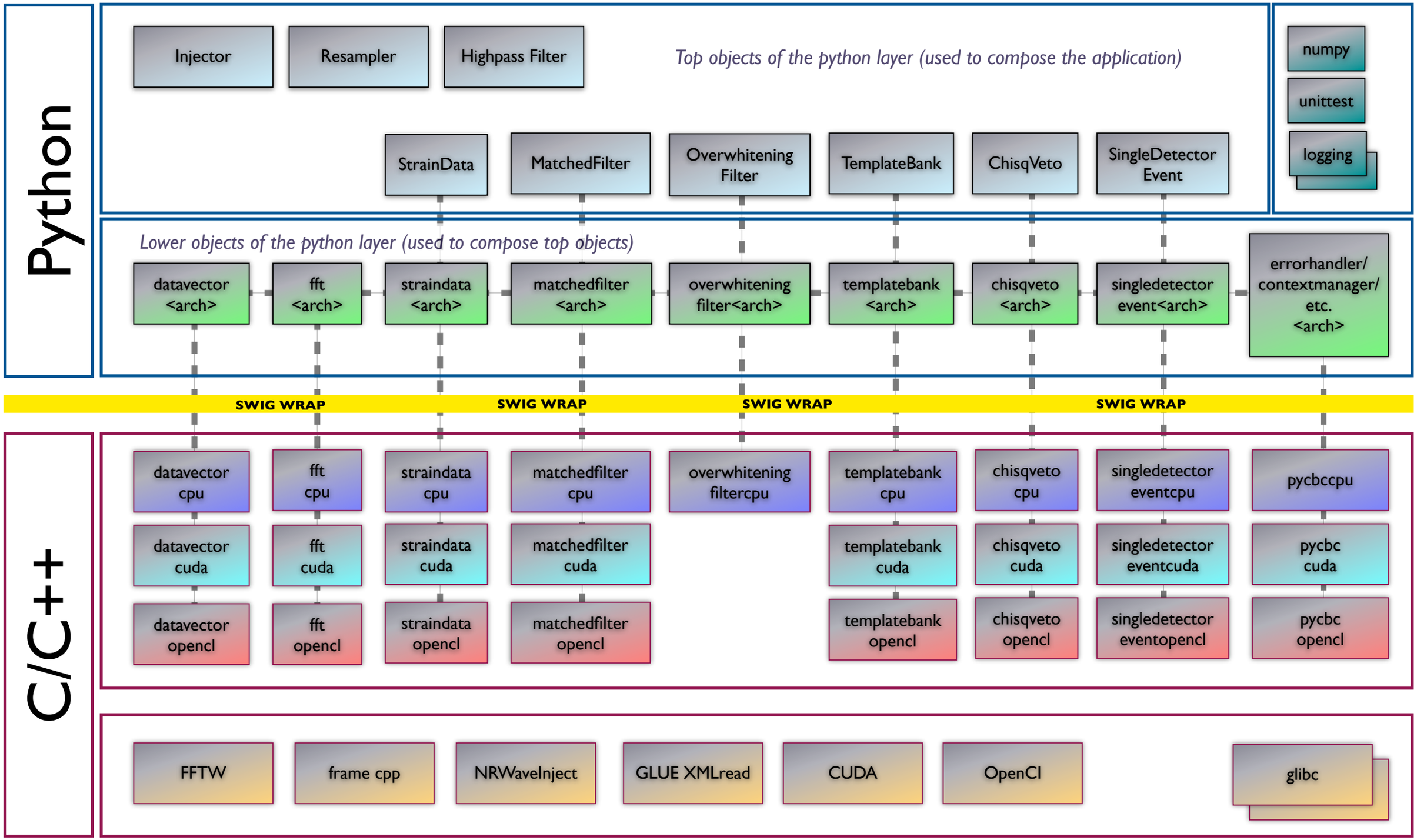
Device memory flow to and from cpu object orientated



End of processing



Pycbc Layers and packages



Aug - 9 - 2011

Inspiral analysis in a single slide...

```
import sys
import random

# preliminary hard coded path to packages
sys.path.append('/Users/kawies/dev/src/pycbc')

from pycbc.pycbc import CPUDeviceContext as ProcessingTargetContext

from pycbc.straindata.straindata_cpu import StrainDataCpu as StrainData
from pycbc.templatebank.templatebank_cpu import TemplateBankCpu as TemplateBank
from pycbc.matchedfilter.matchedfilter_cpu import MatchedFilterCpu as
MatchedFilter
from pycbc.chisqveto.chisqveto_cpu import ChisqVetoCpu as ChisqVeto
from pycbc.singledetectorevent.singledetectorevent_cpu import
SingleDetectorEventCpu as SingleDetectorEvent
from pycbc.datavector.datavectorcpu import real_vector_single_t as
SnrResultTimeSeries

import logging

logging.basicConfig(level=logging.DEBUG,
                    format='%(name)s %(asctime)s %(levelname)s %(message)s',
                    filename='pycbc_min_pipeline.log',
                    filemode='w')

logger= logging.getLogger('pycbc.main_script')


start_message = 'Starting pycbc single detector minimal pipeline ...'
logger.debug(start_message)
print start_message

# setup straindata
search_time = 128 # typ design spec: 2048
sample_freq = 256 # typ design spec: 4096
length =      search_time * sample_freq
segments = 15
gps_start_time= 871147532
gps_end_time= gps_start_time + search_time
interferometer = "H1"

# setup chisq veto
chisq_bins = 16

# setup event finder
snr_threshold = 5.5

with ProcessingTargetContext(1) as context:

    strain_data = StrainData(gps_start_time, gps_end_time,
                        segments, sample_freq,
                        interferometer, context)

    # initialize straindata w/ noise
    for i in range(length):
        tmp = random.uniform(-1,1)
        strain_data.time_series[i] = tmp

    # convert straindata to single precision
    strain_data.convert_to_single_preci()

    # segmenting straindata and transform into frequency domain
    strain_data.perform_fft_segments()

    # transfer straindata to appropriate memory space on target device
    strain_data.render()

    # create 5 templates (testing the iterator of TemplateBank)
    bank = TemplateBank(5, strain_data.segments_length,
                           strain_data.segments_delta_x, context)
    logger.debug("instantiated TemplateBank w/ waveform length: {0}"
                 .format(bank.waveform_length))

    # create matched filter (only generate_snr() has to be implemented
    # for the minimal pipeline)
    matched_filter = MatchedFilter(strain_data.segments_length, context)
    logger.debug("instantiated MatchedFilter w/ length: {0}".format
    (matched_filter.length))

    # instantiate result vectors
    snr = SnrResultTimeSeries(strain_data.segments_length,
                              strain_data.segments_delta_x, context)
    logger.debug("instantiated SnrResultTimeSeries as {0}".format(repr(snr)))

    # instantiate chisq_veto
    chisq_veto = ChisqVeto(strain_data.segments_length, chisq_bins, context)
    logger.debug("instantiated ChisqVeto as {0}".format(repr(chisq_veto)))

    # instantiate event finder
    events = SingleDetectorEvent(snr_threshold, context)
    logger.debug("instantiated SingleDetectorEvent as {0}".format(repr(events)))

    # filter the data against the template bank
    for template in bank:
        htilde = bank.perform_generate_waveform(template)
        for stilde in strain_data:
            matched_filter.perform_generate_snr(stilde, htilde, snr)
            if matched_filter.max() > snr_threshold:
                chisq = chisq_veto.generate_chisq(snr, qtilde)
                events.find_events(snr, chisq)

    # cluster the events across the template bank
    events.cluster_events(0.1)
    events.write("/path/to/output.xml")

    # leaving the ProccessingTargetContext NOW (destroy the device context)
    ###########################################################################

end_message = '... end of pycbc single detector minimal pipeline.'
logger.debug(end_message)
print end_message
```

Current Status

- Codes in progress
  - prototype OpenCL implementation (Budapest)
  - prototype CUDA implementation (Syracuse)
  - prototype framework (AEI)
  - working on integrating different implementations

- Started work as a GPU project
- Currently in a stage where we are rewriting lalapps_inspiral and want to open it up to the group
- Starting to have regular telecons (Tuesdays 2 hours before the CBC telecon)
- For discussion, please mail cbc+pycbc@gravity.phys.uwm.edu
- Can view repository at https://sugwg-git.phy.syr.edu/dokuwiki/doku.php?id=pycbc:home