GPU TECHNOLOGY CONFERENCE
April 4-7, 2016 | Silicon Valley

BIFROST

HIGH-THROUGHPUT CPU/GPU PIPELINES MADE EASY

Ben Barsdell, 4/7/2016

PRESENTED BY
NVIDIA.

DISAMBIGUATION

The ‘Bifrost’ presented here is…

NOT the stellar atmospheres code of the same name

NOT the fluid simulation framework of the same name

NOT a burning rainbow bridge that connects Midgard and Asgard
(although that’s where the name comes from)

https://www.youtube.com/watch?v=K7qM7l7GE5E

OUTLINE

Background

What Bifrost **is**

What's inside

Future work

ACKNOWLEDGEMENTS

Stems from many useful discussion with:

Lincoln Greenhill, Danny Price, Hugh Garsden @ Harvard CFA (the LEDA project)

Work related to the LWA project based at UNM

BACKGROUND

Application areas

Pipeline processing

Soft real-time constraints

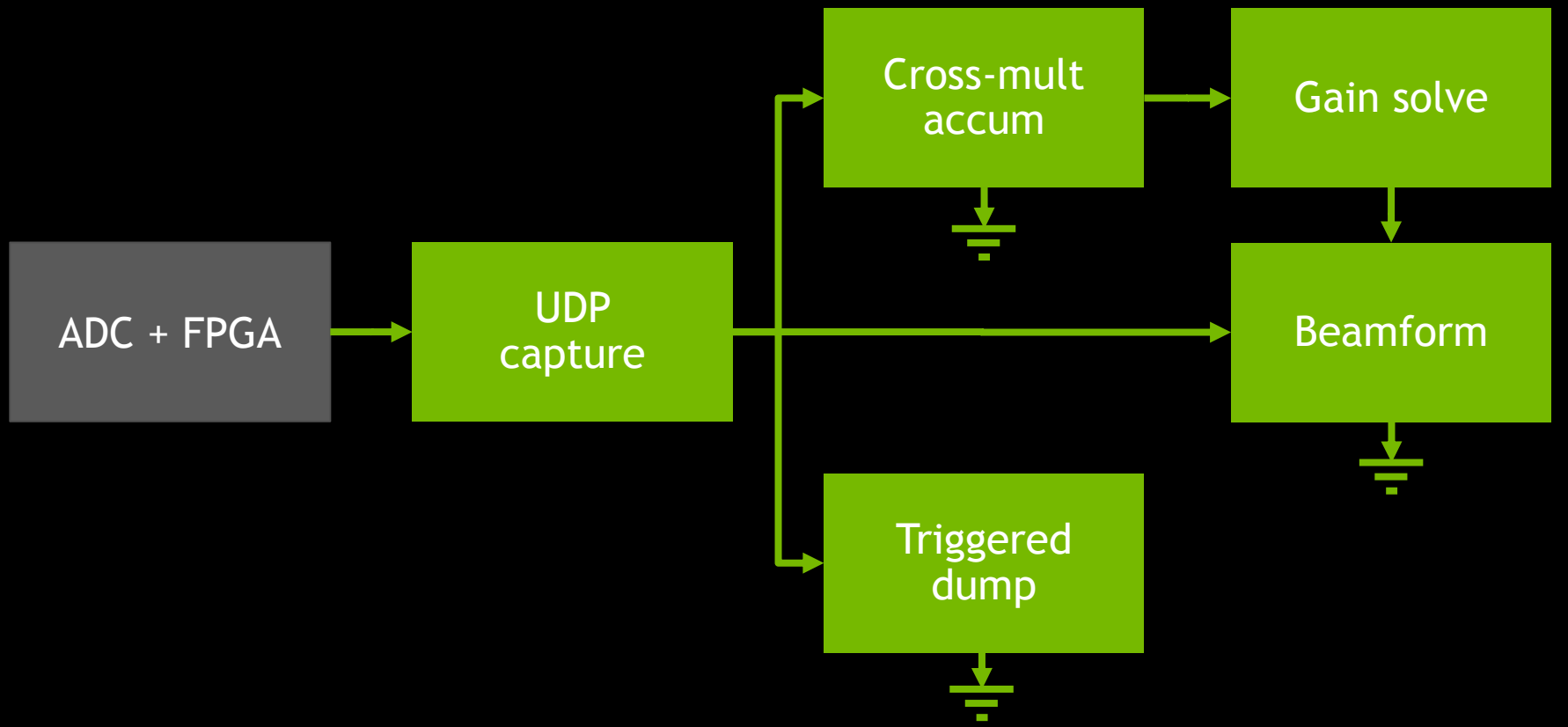
High throughput demands (latency not a big concern)

Experimental science, computer vision

Can’t afford to be inefficient

BACKGROUND

Example: Radio astronomy correlator pipeline



BACKGROUND

Current approaches

	PRODUCTIVITY	PERFORMANCE
Numpy, Matlab etc.	High	Low
Monolithic C/C++/CUDA	Low	Medium
Pipeline C/C++/CUDA	Very low	High

BACKGROUND

Motivation

We know **GPUs are great at signal processing**

Many efficient kernels have been written

BUT:

Sharing of code within the community could be improved

Stitching together a pipeline is still a **hard problem**

Debugging a pipeline can be very painful

BACKGROUND

Existing software

PSRDADA

HashPipe

Pelican

GNU Radio

CASPER toolflow

Plus many standalone processing pipelines for individual projects…

BIFROST

What it aims to be

A framework for flexible CPU/GPU pipelines

+ a library of common operations

Productivity: high-level API, rapid prototyping and debugging

Performance: competitive with best-in-class, suitable for instant deployment

BIFROST

What it aims to be

Describe pipelines in, e.g., JSON or simple Python

Iterate quickly on new ideas, watch results in real time

Share and reuse common operations within the community

Reduce total development time by 10x

BIFROST

What it actually is

Still very early in development! Lots more work to be done.

Currently consists of:

Flexible ring buffer implementation (the heart of the framework)

Small selection of useful functions

Prototype packet capture functionality

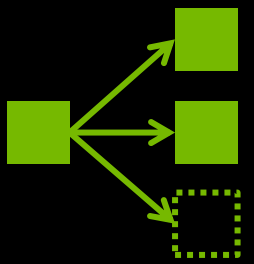
Portable C API with C++ and Python wrappers

BIFROST

Ring buffer

CPU or GPU memory space

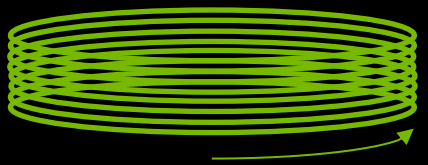
Independent access to contiguous spans of any size at any offset



Fully thread-safe, including resize at any time

Multiple readers, guaranteed or commensal

‘Ringlets’ (aka channels) allow time to be fastest-changing dimension



Sequence management with random access by name or time tag

BIFROST

Library functions

Memcpy/memset wrappers

General ND array transpose (1-16 byte elements)

Under development: CMAC, delay-and-sum, gain solve

Eventually: filtering, imaging, RFI mitigation, transient searching…

Existing implementations can be wrapped for integration into pipelines

BIFROST

Asynchronous execution model

Launch processing operations in different CPU threads

Communicate via ring buffers, copy-free

Pass metadata via sequence headers in the ring

Execute **synchronously** within each thread, but **don't block the GPU**
(use local stream + cudaStreamSynchronize)

IO + CPU + H2D + GPU + D2H in separate threads => **full pipelining**

BIFROST

Packet capture

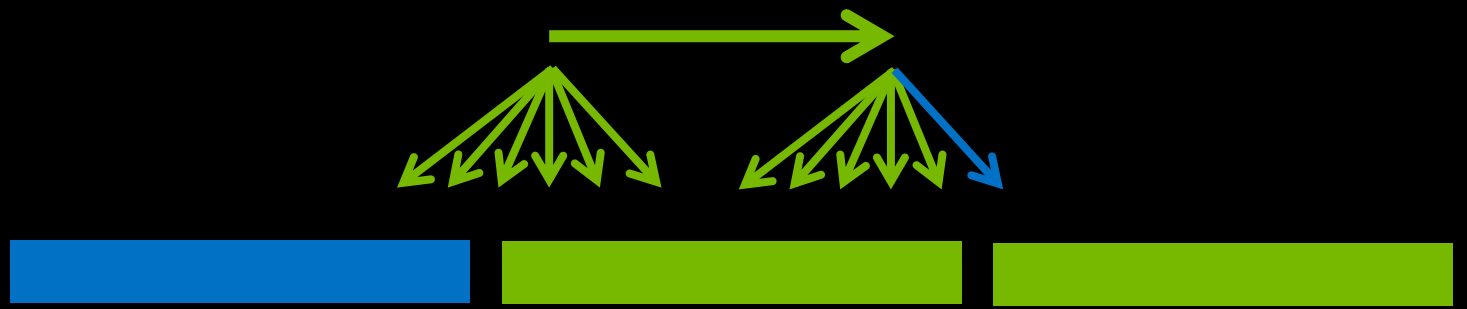
Fast UDP packet capture very important for radio telescope backends

Want to achieve line rate on 10 or 40 Gbps ethernet NICs

Catch packets and scatter into correct order in ring buffer

Keep up to 3 ‘spans’ open for writing, commit the earliest when the latest is touched

Auto-segment based on header changes or timeouts



BIFROST

Triggered dump operation

"Triggered baseband dumps" are a common feature of radio telescopes

Use large ring buffer to keep the past X seconds in memory

Ring sequences enable random access to buffered points in time

BIFROST

The importance of metadata

Sequence headers can be used to store metadata

Enables strong decoupling of processing operations

Allows ‘smart’ operations; avoids manual configuration/adjustment of parameters

Using a standard encoding (e.g., json) simplifies mixed-language pipelines

BIFROST

Python operation example

```
class TransposeOp(object):
    def main(self):
        with self.oring.begin_writing() as oring:
            for iseq in self.iring:
                ihdr = json.loads(iseq.header.tostring())
                dtype = np.dtype(ihdr['dtype'])
                ohdr = {}
                ohdr['frame_shape'] = ihdr['ringlet_shape']
                …
                ohdr = json.dumps(ohdr)
                self.oring.resize(ogulp_nbyte)
                with oring.begin_sequence(iseq.name, ohdr, onringlet) as oseq:
                    for ispan in iseq.read(ogulp_nbyte, self.guarantee):
                        with oseq.reserve(igulp_nbyte) as ospan:
                            src = ispan.data_view(dtype)
                            dst = ospan.data_view(dtype)
                            bfTranspose(dst, src, axes=[1,0])
```

Metadata handling

Ring handling

Processing

BIFROST

Ring buffer C API sample

```
BFstatus bfRingCreate
BFstatus bfRingDestroy
BFstatus bfRingResize

BFstatus bfRingSequenceBegin
BFstatus bfRingSequenceEnd

BFstatus bfRingSequenceOpen
BFstatus bfRingSequenceOpenAt
BFstatus bfRingSequenceOpenLatest
BFstatus bfRingSequenceOpenEarliest
BFstatus bfRingSequenceOpenNext
BFstatus bfRingSequenceClose

BFstatus bfRingSpanReserve
BFstatus bfRingSpanCommit

BFstatus bfRingSpanAcquire
BFstatus bfRingSpanRelease
```

FUTURE WORK

Current plans

Abstractions for quickly writing new ops

Automated pipeline construction (threads, ring allocation, metadata handling etc.)

Large library of operations that can be strung together

Fast and customizable UDP packet capture

Live streaming data visualization (‘scopes’)

FUTURE WORK

Contributions

Looking for feedback, suggestions, contributions

Planning to push new code to GitHub soon

http://beingevil.tumblr.com/post/10980294735/horrible-thor-pickup-lines-1

GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

THANK YOU

JOIN THE CONVERSATION

#GTC16

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join

PRESENTED BY

NVIDIA.