Generic Polyphase Filterbanks with CUDA

Jan Krämer

DLR German Aerospace Center
Communication and Navigation
Satellite Networks
Weßling

04.02.2017

DLR

Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes

Outline

Once upon a time in a space project

- Multicarrier scheme with 15/30/45 carrier

Once upon a time in a space project

- Multicarrier scheme with 15/30/45 carrier
- So let's just use a PFB, right?

Early Trouble

- 45 carrier means 45x the bandwidth
- Only 12-15 % guardband available
- At least 3x oversampling needed
- Up to 1500 tap filters needed

Early Trouble

- 45 carrier means 45x the bandwidth
- Only 12-15 % guardband available
- At least 3x oversampling needed
- Up to 1500 tap filters needed

Early Trouble

- CPU reference implementation
- 1000 taps
- 35dB rejection
- Originally 9x oversampling
- 2 Msamples/second achieved $\Rightarrow$ 4 Msamples/second needed

Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes

What is CUDA

- NVidias framework for GPGPU
- Used mainly to accelerate scientific computing
- Uses the massive amount of available compute cores inside a GPU

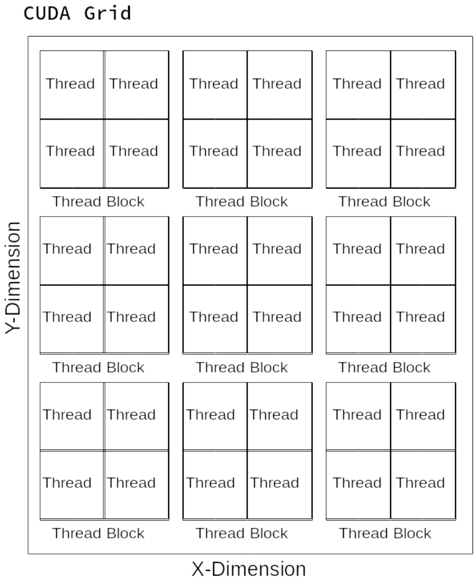
GPU Interior

- GPU consists of several Streaming Multiprocessors (SM)
- Each SM consists of numerous compute or CUDA cores
- Single-Instruction Multiple-Threads (SIMT) structure
- Several kinds of memory
  - Global Memory (GDDR5 RAM) (slow)
  - On-Chip (shared) Memory per SM (faster)
  - Registers (blazingly fast)

CUDA Interior

- Builds a (up to) 3 dimensional Grid
- The Gid contains the (up to) 3 dimensional Thread Blocks containing the threads
- Groups of 32 threads inside a Thread Block are grouped together $\Rightarrow$ Warp

CUDA Interior



Thread Execution

- Each Block has a unique ID inside the Grid ⇒ Each thread has a unique global ID
- Thread Scheduler assigns each Thread Block to one SM and executed concurrently
- All threads in a Warp are executed concurrently inside the SM

Performance Bottlenecks

- Uncoalesced loads from global memory
  $\Rightarrow$ Several cache-lines to be loaded

Performance Bottlenecks

- Uncoalesced loads from global memory
  $\Rightarrow$ Several cache-lines to be loaded
- Bank conflicts when accessing shared memory

Performance Bottlenecks

- Uncoalesced loads from global memory
  $\Rightarrow$ Several cache-lines to be loaded
- Bank conflicts when accessing shared memory
- Branching $\Rightarrow$ Which instruction should be executed?

Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes

Why PFBs and Channelizers/Synthesizers?

- Used to reduce computational complexity for resampling filters
- Used to separate small bandwidth channels
- Used to generate multicarrier ’broadband’ signals

Structure of a PFB Channelizer

- Extracting a channel with $\frac{1}{N}$ of the total bandwidth
  - Mix Signal to Baseband
  - Apply anti-alias filter
  - Downsample the signal

Structure of a PFB Channelizer

- Extracting a channel with $\frac{1}{N}$ of the total bandwidth
  - Mix Signal to Baseband
  - Apply anti-alias filter
  - Downsample the signal
- N-phase PFB splits one-dimensional filter in its N different phase shares

Structure of a PFB Channelizer

- Taps of the regular prototype filter

t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15

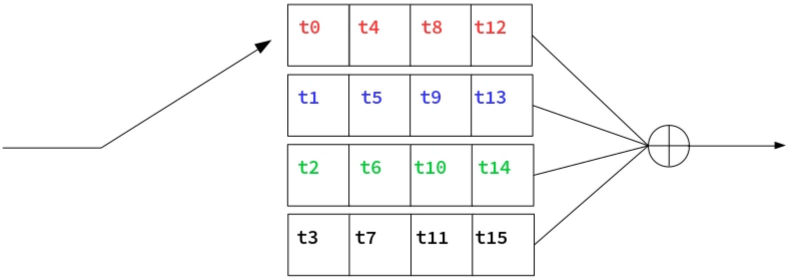
Structure of a PFB Channelizer

- Taps of the regular prototype filter
- Split into 4 polyphase partitions

t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15

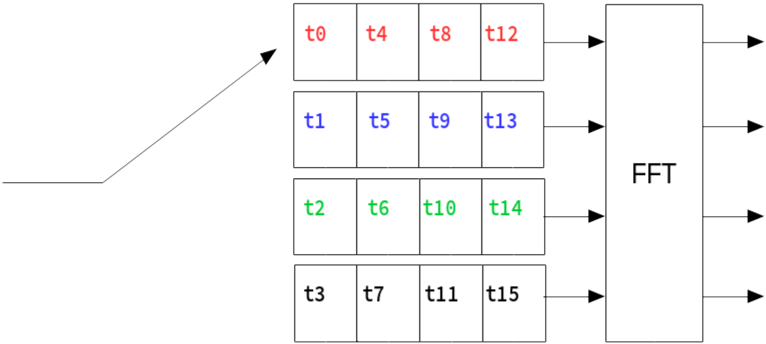
Structure of a PFB Channelizer

- Taps of the regular prototype filter
- Split into 4 polyphase partitions
- Newly structured dataflow



Structure of a PFB Channelizer

- Taps of the regular prototype filter
- Split into 4 polyphase partitions
- Newly structured dataflow
- FFT separates all the channels



Structure of a PFB Channelizer

- Oversampling can be achieved by manipulating the input commutator and FFT input
- To synthesize several incoming channels just the reorder the operations

Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes


EVERYDAY
IM WITH STOOPID
I'M SHUFFLIN'

Identifying necessary operations

- Channelizer consists of 4 operations
  - Shuffle the input stream
  - Polyphase filtering
  - FFT
  - Shuffle the output stream

Input Shuffling

- Input Commutator implemented as matrix traversal
- Number of threads needs to accomodate to the filter history
  $\Rightarrow$ Grid dimension takes care of this
- Input buffer reads are coalesced $\Rightarrow$ Block x-dimension same size as polyphase partition
- Intermediate buffer writes are therfore not coalesced

Filter Operations

- Block X dimension computes several input samples
- Block Y dimension computes oversampled output samples
- Grid X dimension represents polyphase partitions
- Grid Y dimension provide additional concurrency (due to block thread limits)

Filter Operations

- Each threadblock transfers memory from global memory to shared memory
- Each sample is accessed several times $\Rightarrow$ shared memory offers faster memory transfers
- Register and shared memory spills are avoided

FFT and Output Shuffling

- FFT is the CuFFT of CUDA
- Output shuffling implemented as double loop done on Host CPU (for now)

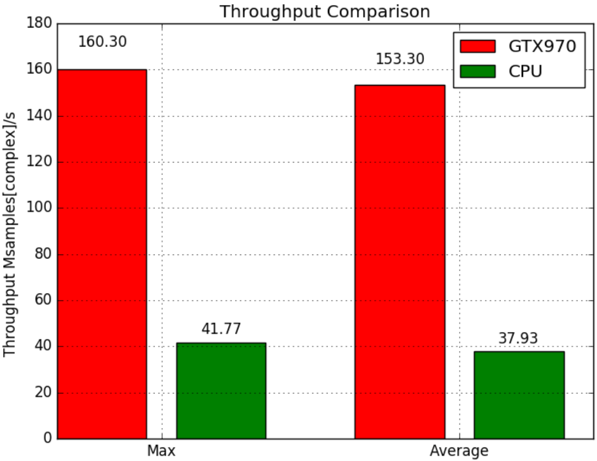
Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes

32 Channel PFB

- 32 Channels
- No Oversampling
- 437 taps prototype filter

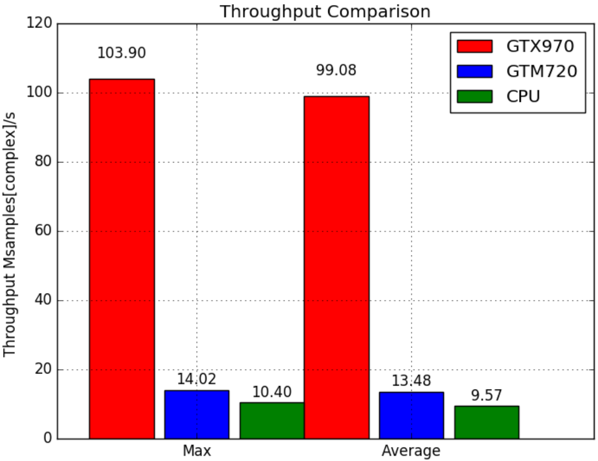
32 Channel PFB



45 Channel PFB

- 45 Channels
- 3x Oversampling
- 1501 taps prototype filter

45 Channel PFB



Outline

1. Motivation
2. Short introduction to CUDA
3. PFBs and the Channelizer
4. Translation to CUDA
5. Results
6. Release plans and future changes

Release Plan

- Release Date: TBD
  - Still some bureaucratic hurdles
  - Still dependent on project code
- License: LGPL3
- Platform Github (Group KN-SAN)
- Follow `https://github.com/spectrejan` for release news

Contact:
j.kraemer@dlr.de
@JanKrmer
`https://github.com/spectrejan`