# An Efficient GPU Implementation of a Multirate Resampler for Multi-carrier Systems

December 14 – 16, 2015

Scott C. Kim[1] and Shuvra S. Bhattacharyya[1,2]

1. University of Maryland – College Park, MD, USA
2. Tampere University of Technology, Finland

{sckim, ssb}@umd.edu

IEEE GlobalSIP 3rd IEEE Global Conference on Signal & Information Processing
Orlando, Florida, USA December 14-16 2015

# Introduction

- Sample rate conversion (SRC) plays a vital role in modern communication systems
  - Tight coupling between sampling rates and data rates
  - Supports different data rates, clock speeds, etc.
  - Fixed clock vs. tunable clock
- Simple SRC
  - Integer up or down sampling (interpolation or decimation)
  - Rational resampling – combined up and down sampling
- Arbitrary or asynchronous SRC (ASRC)
  - Dynamic re-computation of fractional resampling points in a single stage
  - Tradeoff: cost and complexity

# Motivation

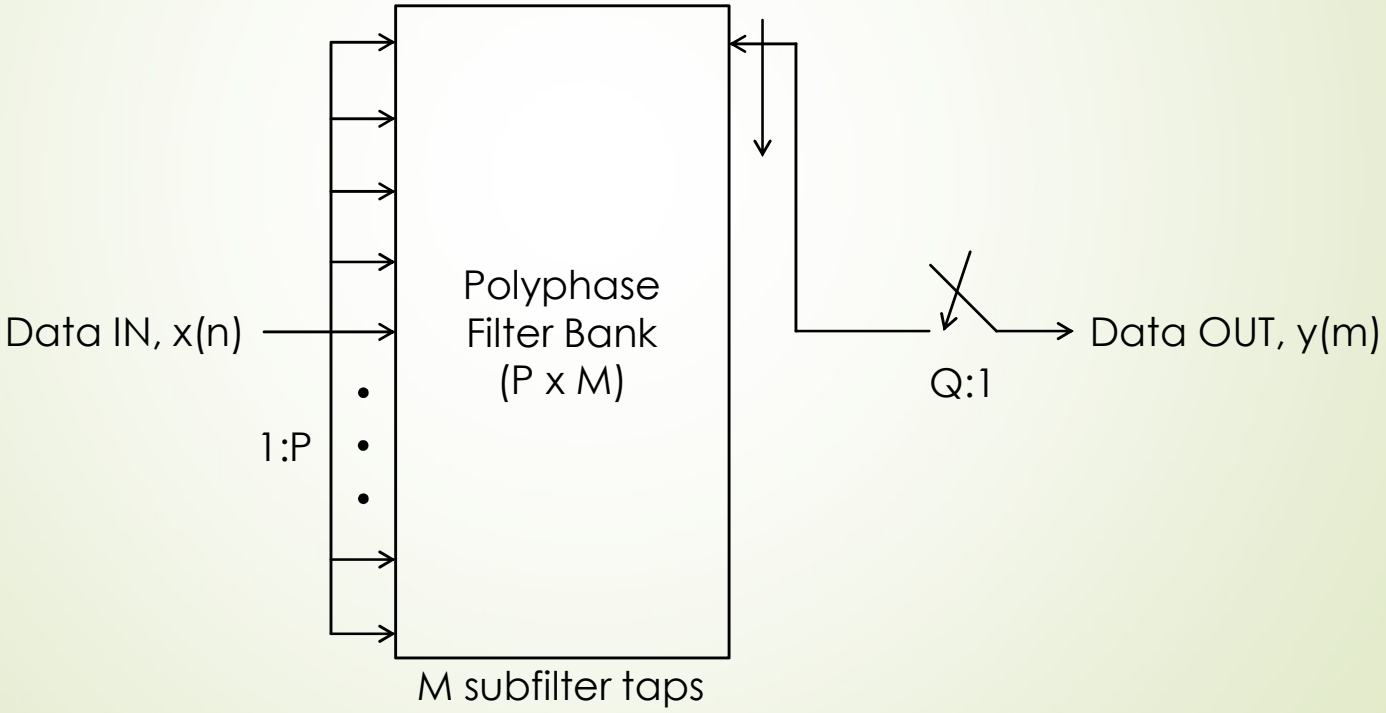
- A flexible, high performance, and resource efficient resampler for various data rates is desired
- Utilize graphics processing units (GPUs)
  - Compared to hardware-based resamplers (ASIC/FPGAs)
  - Streaming data in h/w vs. array of data (available) in s/w
- Wide bandwidth (BW) processing for high data rate systems desired
  - 100's of MHz of BW and over Gbps throughput desired
  - Challenge: spectrum fragmentations (900 MHz, 1800 MHz, etc.)
  - Carrier/band aggregation available
    - Intra-band vs. inter-band aggregation
- Use GPU as a real-time, software radio
  - High throughput and low latency

# Background – Resampling

- Rational resampling concept
  - Interpolation (reconstruction) by P
    - Upsample (i.e., zero stuffing) then filter (reject images)
  - Decimation (resampling) by Q
    - Filter (reject adjacent signals), then downsample (i.e., discard)
  - Combine two integer resamplers = fractional resampling
    - Overall ratio, R = P / Q
- Efficient multi-rate resampling available
  - Using polyphase filter bank (PFB) structure [1, 3, 4, 5]
- Avoid ratios involving co-prime integers that are in the hundreds (or more) — e.g., R = 766/500
- Multi-stage resampling, e.g., 3/5 then 2/3, etc.
  - Simpler but adds delays

# Background – Resampling

- Polyphase rational resampler
  - Combined interpolation and decimation using a single filter

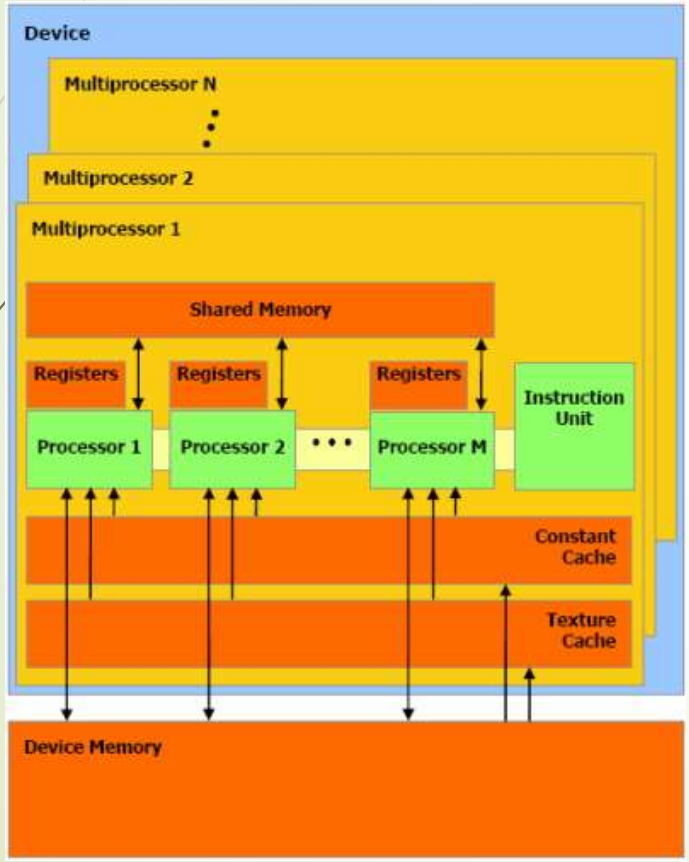


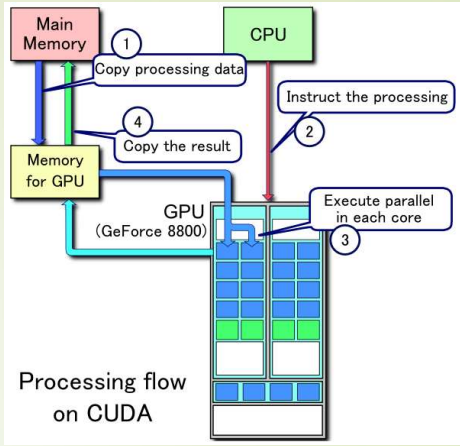
# Background — CUDA


- NVIDIA's parallel programming language
  - Based on SIMT architecture
- Memory hierarchy
  - Registers, shared memory, local memory, constant memory, global memory
- Scalable kernel dimensions (1D/2D/3D):
  - thread, block, grid
- Use coalescing or grouping for faster/linear access
- Minimize data transfers between CPU and GPU
- Use multiples of a *warp* (32 threads) to spawn threads
- Floating point supported:
  - single (32-bit) and double (64-bit) precision
- Performance tuning via occupancy calculator and visual profiler

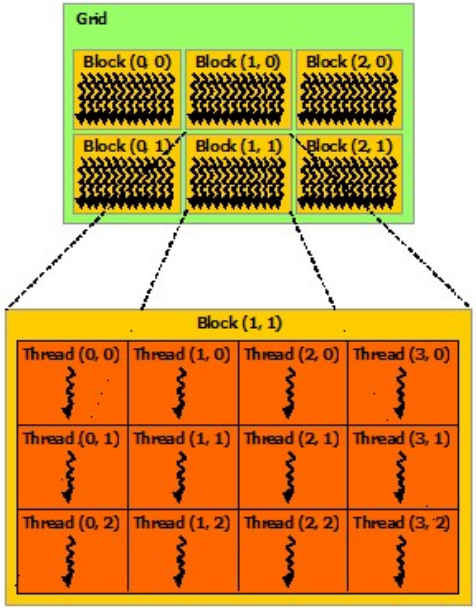
# CUDA Overview

## Hardware layout




## Kernel layout

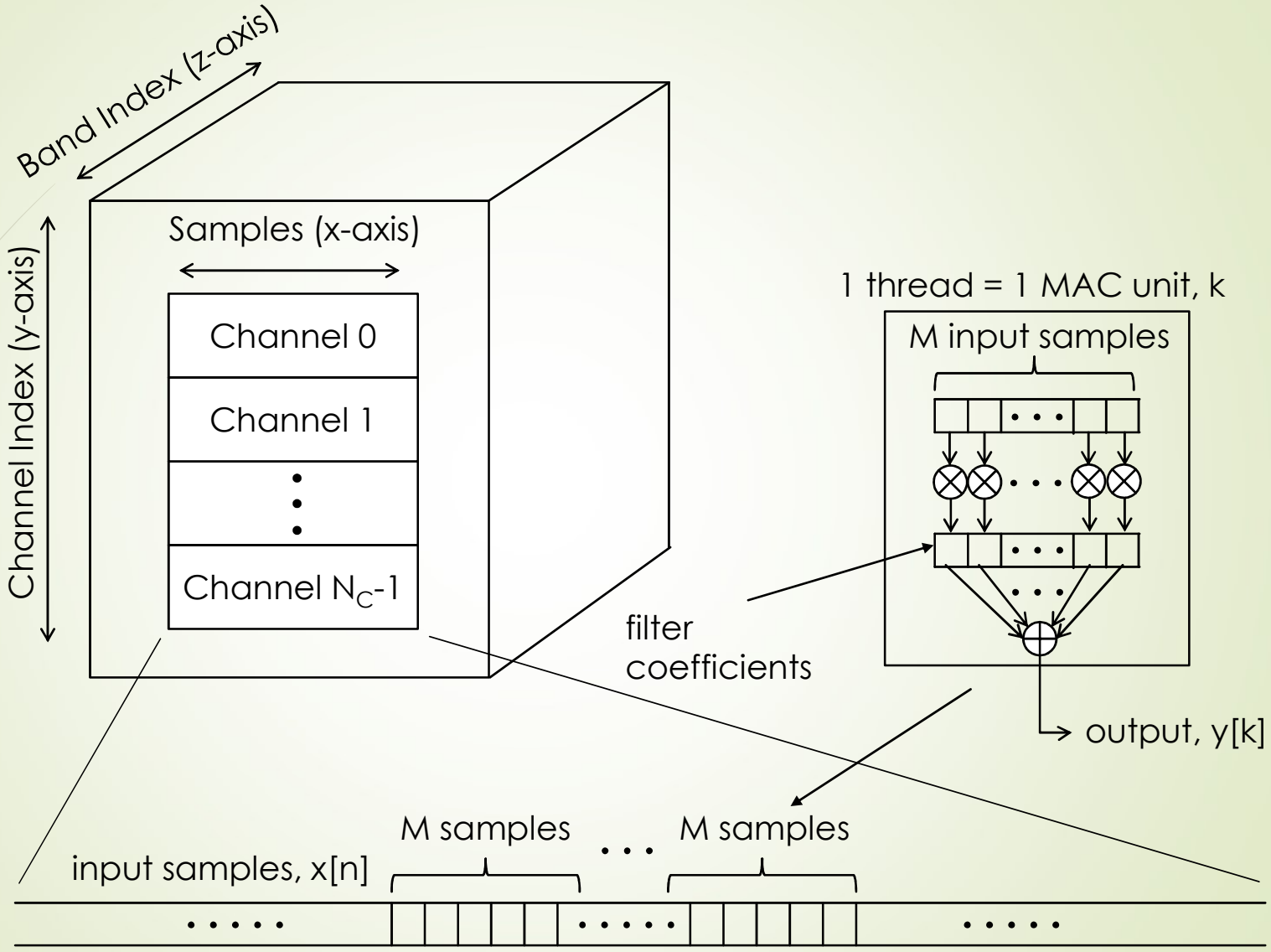



# Related Work

- Multicore acceleration of resampler [8, 9, 10]
- GPU-based ASRC [11, 12]
  - Real-time, high throughput
  - Texture memory based ASRC [12]
- FPGA/VLSI-based implementation [13, 14, 15]
  - Performance criteria met via system clock speed, resource usage (block RAM, multipliers, etc.)
- GPU-based SRC
  - Simple, efficient, effective all software-based solution
  - Non-custom, commercially-available device
  - Processes multiple channels and bands simultaneously
    - Realizes CA using a single GPU for baseband processing

# GPU-based Multi-carrier Resampler

- A single stage resampler
  - Interpolate, filter, then decimate in an integrated kernel
  - Entire operation contained within a block of threads
- Store filter coefficients in constant memory (CM)
  - Fast, read-only broadcasting (cached)
- Eliminate commutator approach
  - Vector accessing and vector-matrix multiplication
- 1 thread for 1 multiply-accumulate (MAC)
  - Instantiates fused multiply-add (FMA) for speed and accuracy
- Address irregular data access used by polyphase filter in GPU
  - Data is loaded column-wise but operated on row-wise


Band Index (z-axis)
Samples (x-axis)
Channel Index (y-axis)
Channel 0
Channel 1
Channel $N_C$-1
1 thread = 1 MAC unit, k
M input samples
filter
coefficients
output, y[k]
input samples, x[n]
M samples
M samples

# Implementation and Experimental Setup

- Target 3GPP radio frame duration of 10 ms
- Resample 10 MHz LTE signal from 25.6 MHz to 15.36 MHz
  - Resampling ratio, R = 3/5
- Use equiripple FIR filter
  - 96 taps, decomposed to P = 3 by M = 32
  - 70 dB attenuation, 0.3 dB passband ripple
- Use desktop NVIDIA GPU, GTX 680, 780 Ti, and 970 (reference)
- CUDA toolkit version 6.5
- 32-bit, complex, floating-point precision throughout
- A block dim = 256 threads in x-direction, 3 threads in y-direction, total 768 threads per block (TPB)
  - 1,000 blocks in x-direction, $N_C$ channels in y-direction

# Results 1 – Single Channel Interpolation

| GPU | with data transfer | without data transfer |
|---|---|---|
| GTX 680 | 1.967 ms / 260.32 MSps | 0.294 ms / 1,742.16 MSps |
| GTX 780 Ti | 2.331 ms / 219.64 MSps | 0.177 ms / 2,889.13 MSps |
| GTX 970 | 2.356 ms / 217.29 MSps | 0.207 ms / 2,474.86 MSps |
| Tegra K1 | 16.82 ms / 30.437 MSps | 6.532 ms / 78.38 MSps |

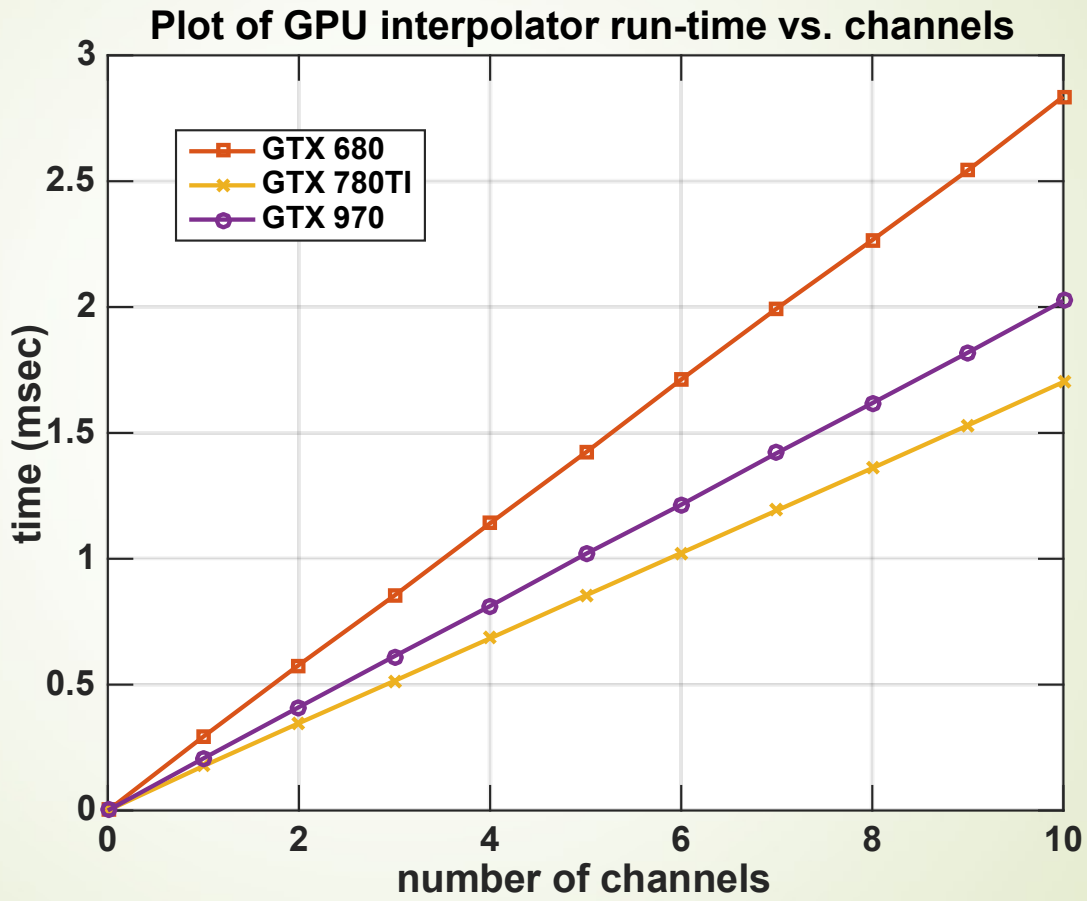
- NVIDIA's embedded GPU/SoC, Tegra K1 shown for comparison only, not suitable for real-time operation
  - However, significant decrease power consumption (<10 W)
- Discrete GPUs ran under target latency of 10 ms
  - w/o data transfer ran under LTE slot time of 0.5 ms

# Results 2 – Single Channel Resampling

| GPU | with data transfer | without data transfer |
|---|---|---|
| GTX 680 | 0.881 ms / 580.93 MSps | 0.306 ms / 1,671.72 MSps |
| GTX 780 Ti | 0.944 ms / 541.89 MSps | 0.189 ms / 2,712.78 MSps |
| GTX 970 | 0.959 ms / 533.51 MSps | 0.224 ms / 2,284.73 MSps |
| Tegra K1 | 8.889 ms / 57.59 MSps | 6.319 ms / 81.02 MSps |

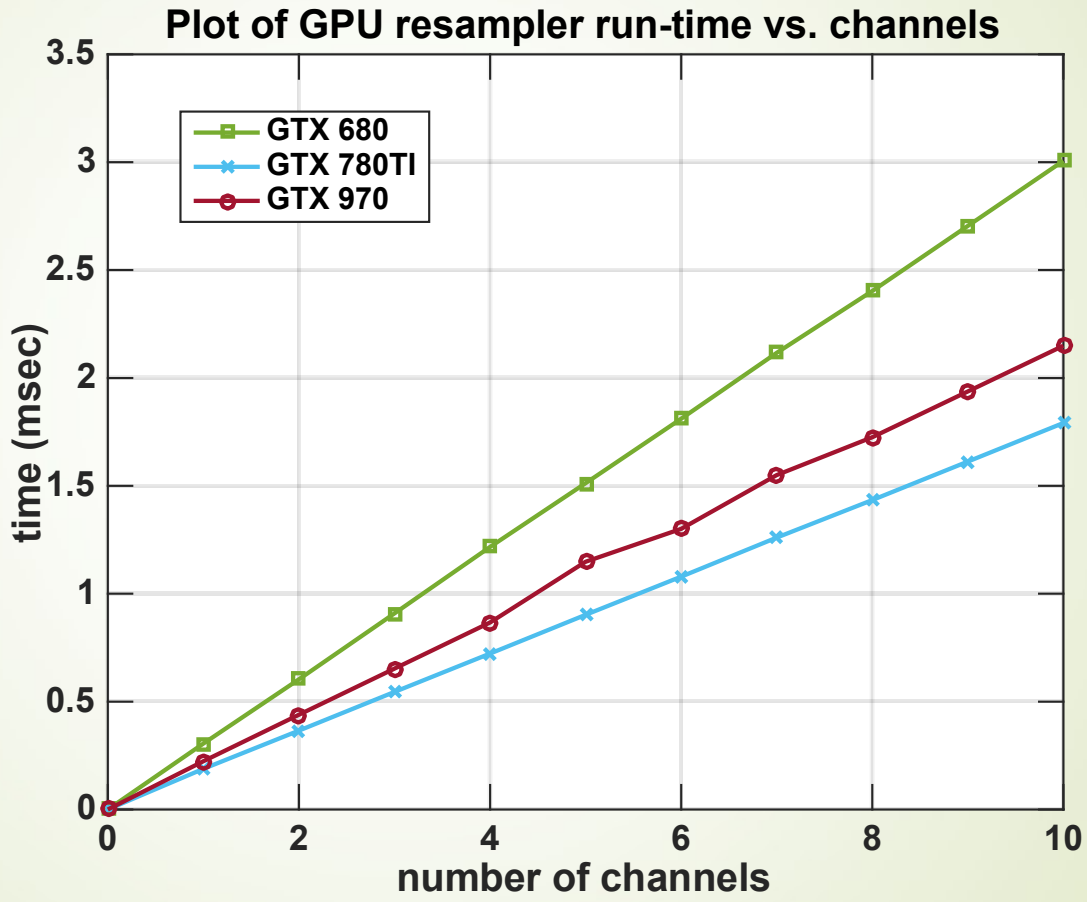
- Resource usage of GTX 970 (reference GPU)
  - GM load efficiency 100% (from 0%)
  - GM store efficiency 73% (from 0%)
    - Occasional bank conflict in SM and index transposing
- Occupancy calculator – block size, registers, SM usage, etc.
  - Theoretical 75%, achieved 72%
  - No register spills
  - Both kernels utilized FMA operations

# Results 3 – Interpolator (All Channels)




- Aggregation of channels up to 100 MHz (ten 10 MHz channels)

# Results 4 – Resampler (All Channels)




- Note: data transfer time not included (all GPU baseband processing)

# Conclusion

- Introduced a novel, efficient GPU-based rational resampler
- For multiple channels and bands
  - Utilize GPU's 3D architecture for CA
- A single integrated filter design
  - Elimination of commutator approach via array indexing
- Addressed irregular memory access patterns in polyphase filtering
  - Drastic improvement of global memory efficiency from 0%
- Achieved objectives of high throughput and low latency
  - High throughput achieved, over 1.5 GSps
  - All the kernels ran well below 10 ms target latency

# Questions?