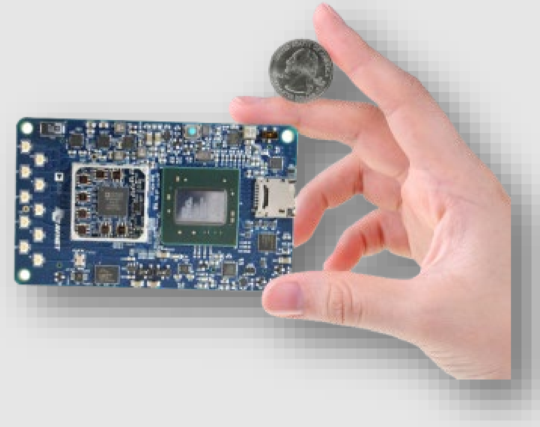
Module 44
CUDA Streams

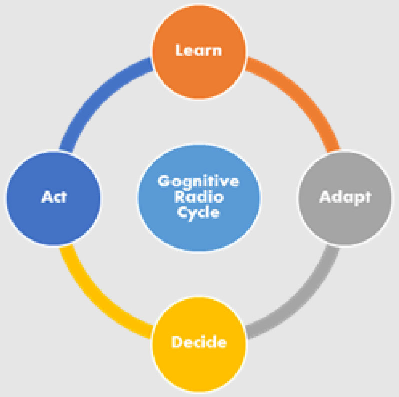
GPU based Streaming Enabled Spectral Correlation Density Function (SCD)

Signal Classification

THE UNIVERSITY OF ARIZONA®

- Software Defined Radio (SDR)
  - Radio components in software
  - Rapid reconfiguration
    - higher spectral efficiency based on environmental conditions.
      - signal parameters such as transmission frequency or modulation scheme





Task of reliable and real-time signal classification involving non-cooperative communication is difficult

Challenge

THE UNIVERSITY OF ARIZONA®

- Consider 5G Systems
  - Massive growth in number of edge devices, scaling to billions, in the wireless spectrum.
- Challenges:
  - Recognizing signals of a specific modulation type in real-time
  - Share the spectrum effectively without hampering the other transmissions at various priority levels

Accurate and fast modulation classification is needed!

Modulation Classification

- Spectral Correlation Density (SCD)
  - Signals that can be classified: BPSK, QPSK, 8PSK, 16QAM, GMSK, CPFSK, AM, FM, OFDM.
  - Relatively immune to noise
- Challenge:
  - Computational complexity

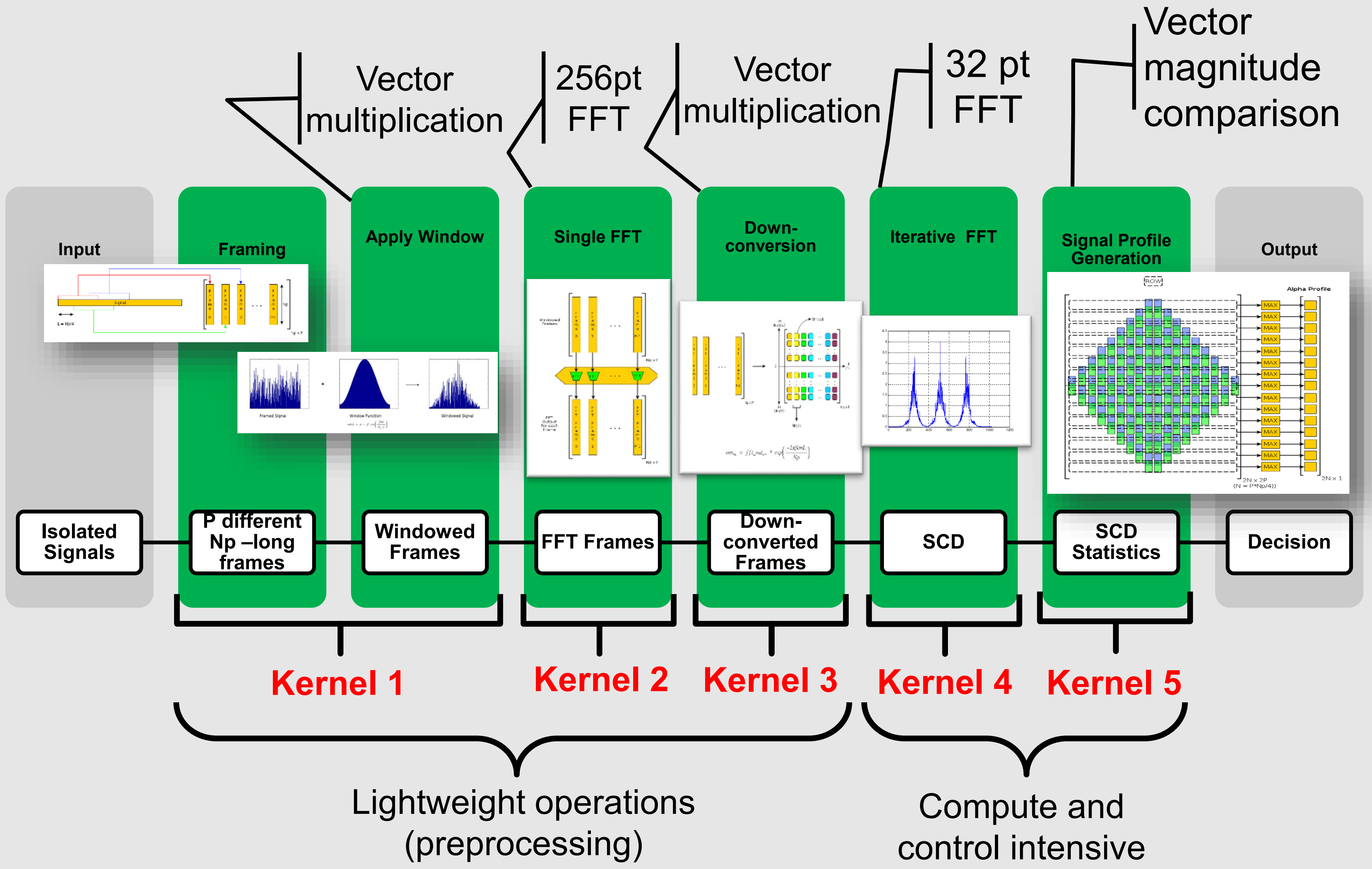
Goals and Outcomes

- High throughput SCD analysis based on GPU
  - Explore parallelization opportunities
    - <u>Algorithmic</u>
      - Full SCD
      - Quarter SCD (QSCD)
    - <u>Architecture Specific</u>
      - Kernel launch overhead
        - » Process multiple signals concurrently
        - » Data streaming
- Throughput Improvement
  - State of the art GPU based Full SCD 120 signals/second [1]
  - Our implementation [2] achieves 3300 signals/second

[1] [Nilangshu Bidyanta, Garrett Vanhoy, Mohammed Hirzallah, Ali Akoglu and Bo Ryu, “GPU and FPGA Based Architecture Design for Real-time Signal Classification,” Proc. 2015 Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio (WInnComm’15), March 24-26, 2015, San Diego, CA, pp. 70-79.

[2] Scott Marshall, Garrett Vanhoy, Ali Akoglu, Tamal Bose and Bo Ryu, “GPGPU based Parallel Implementation of Spectral Correlation Density Function,” *Journal of Signal Processing Systems,* vol. 92, pp.71-93, April 2019. DOI doi.org/10.1007/s11265-019-01448-7

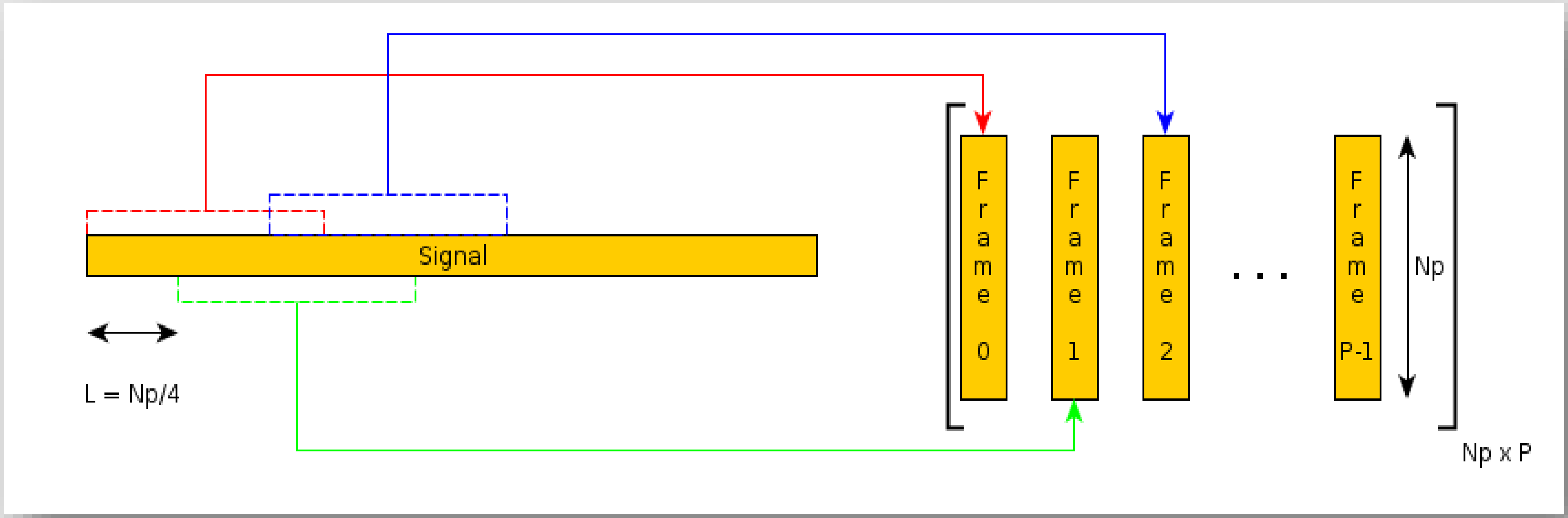
Process Chain on GPU



Spectral Correlation Density

THE UNIVERSITY OF ARIZONA

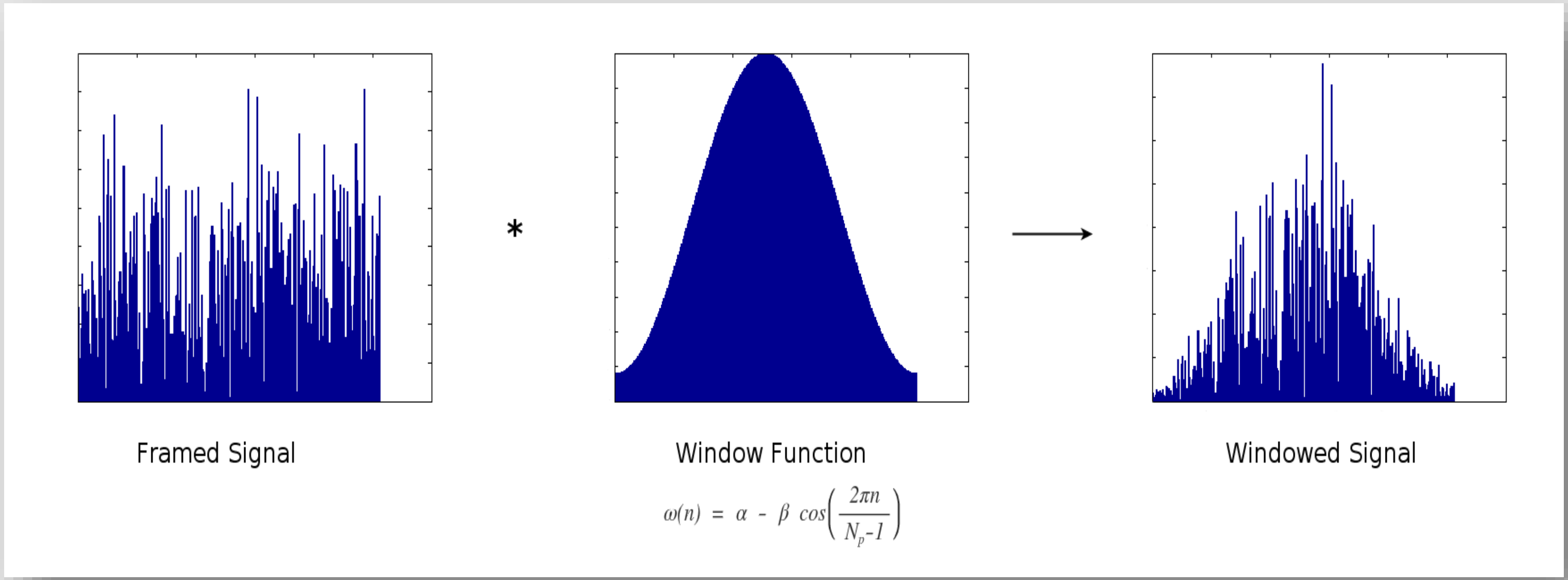
- Step-1: Framing the signal
  - An arbitrary length signal split into P parts (32) of length Np (256) each.
  - Each part overlaps with its predecessor.
    - The offset between the beginning of two consecutive parts is set to L where L = Np/4.
  - Frames arranged column-wise - Np x P matrix



Spectral Correlation Density

THE UNIVERSITY OF ARIZONA®

- Step-2: Hamming Window
  - Steep cut-offs at both ends of the frame (step-1) introduce high frequency components
  - "raised cosine" function of length Np
  - Window function remains the same for a given set of input parameters.



$$\omega(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N_p - 1}\right)$$

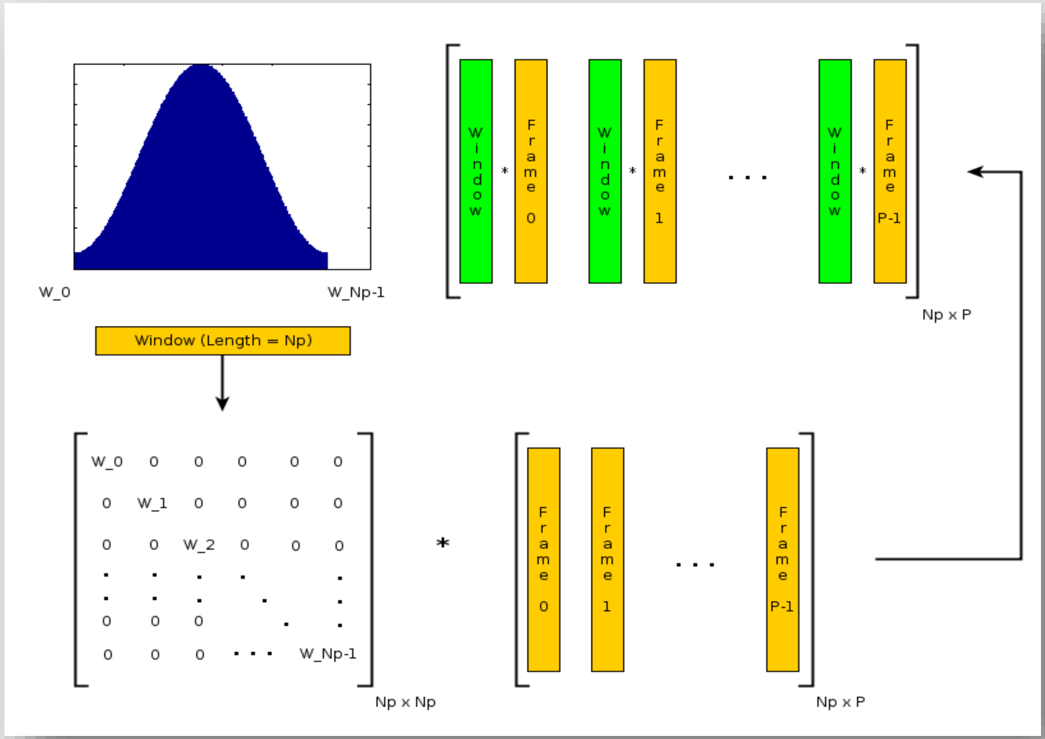
Spectral Correlation Density

THE UNIVERSITY OF ARIZONA®

- Kernel1: Framing and Windowing
  - Change the sparse matrix multiplication to P number of vector multiplications
    - The number of scalar multiplications is Np * P
    - Factoring in the number of signals: Np * P * signal count

✓ Typical Np, P and signal_count:
  - 256, 32 and 30

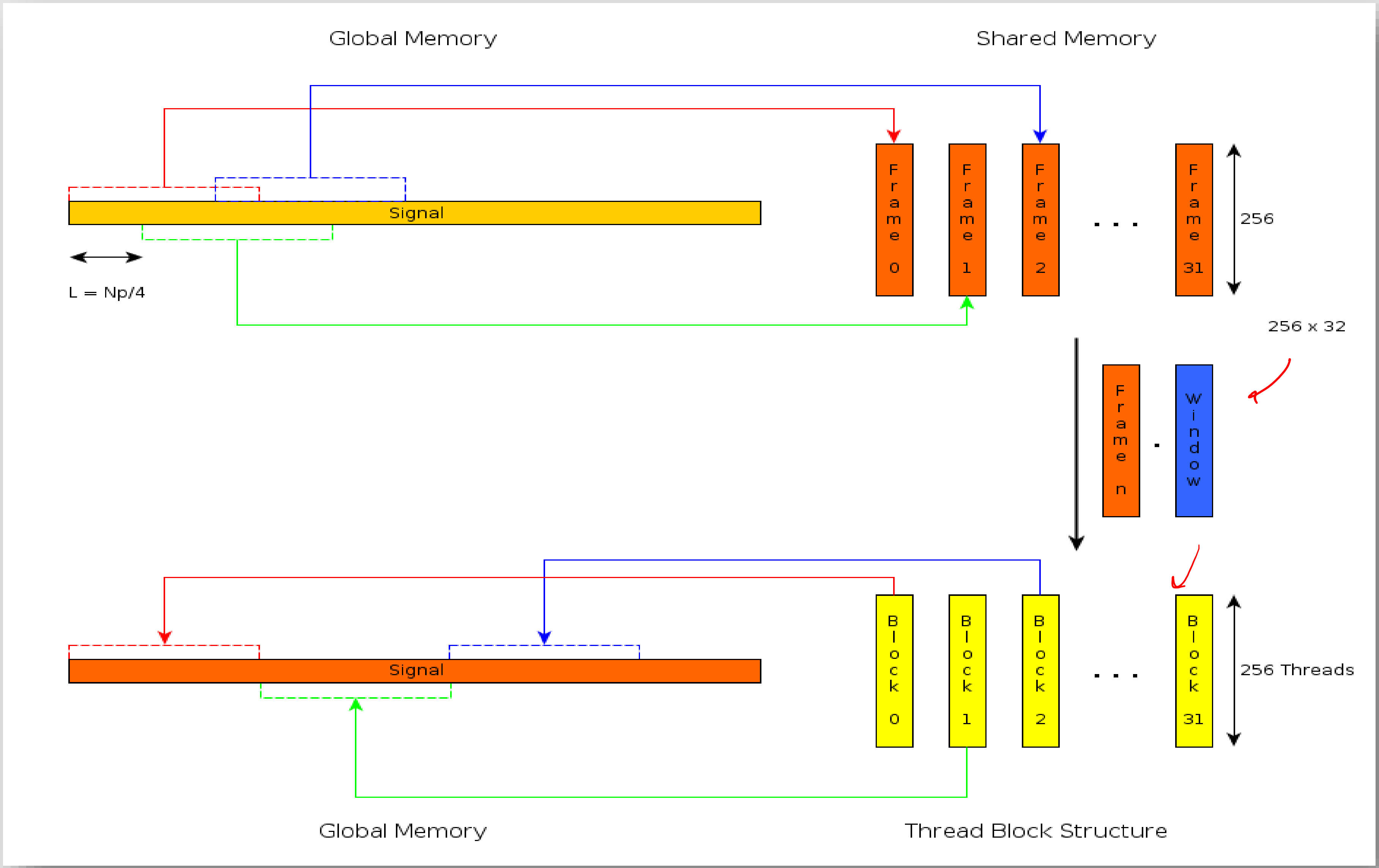
✓ # of scalar multiplications:
  - 245,760



GPU Mapping

THE UNIVERSITY OF ARIZONA®

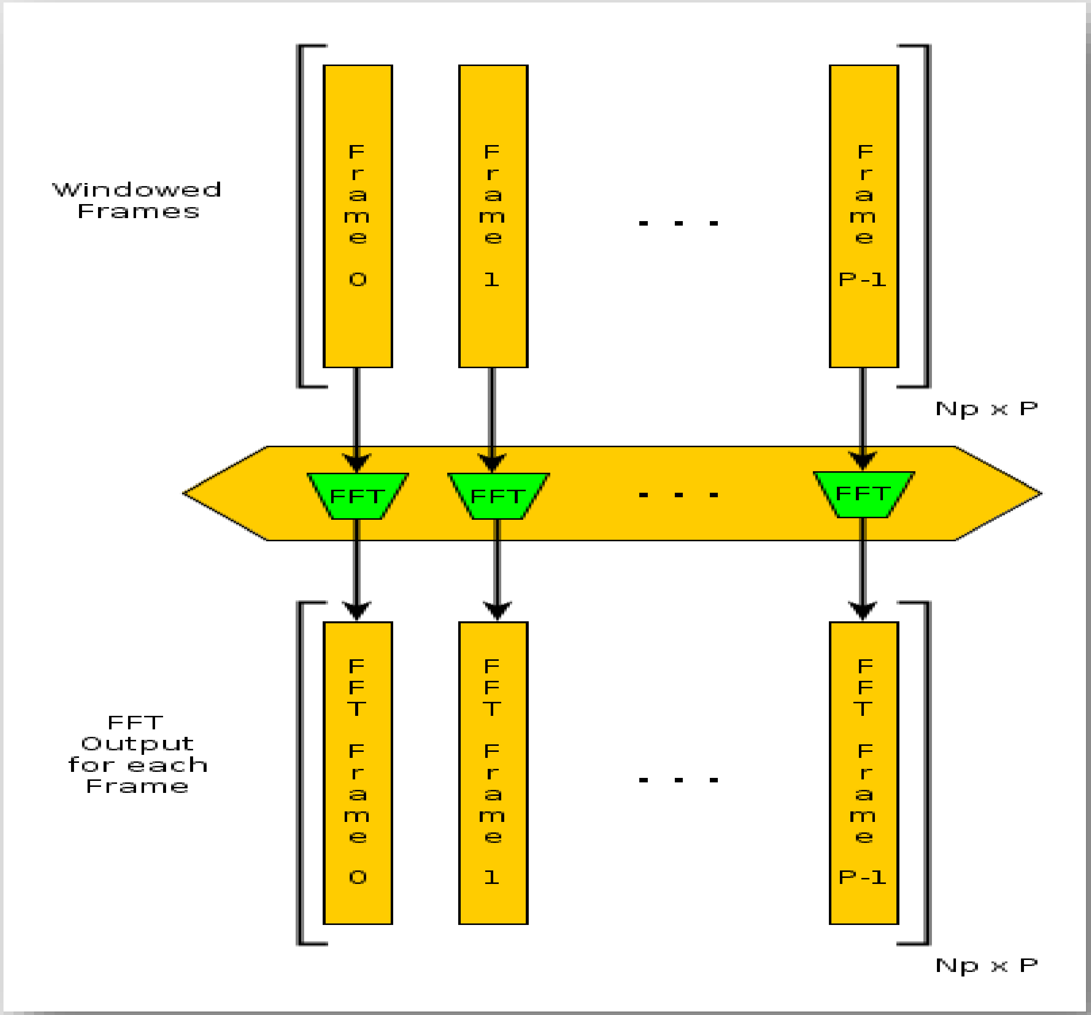
- Framing and Windowing (Kernel 1)



Spectral Correlation Density

THE UNIVERSITY OF ARIZONA®

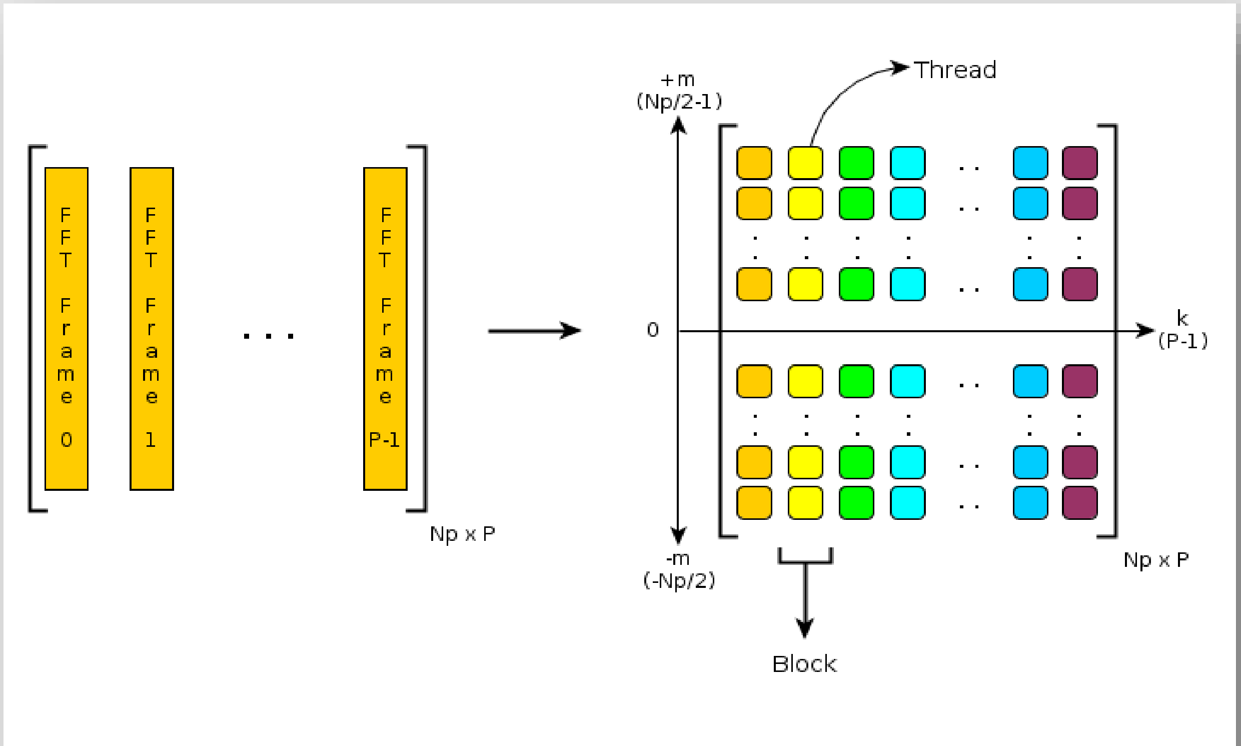
- Step-3: 256pt FFT (Kernel 2)
  - FFT applied in parallel to windowed frames



Spectral Correlation Density

THE UNIVERSITY OF ARIZONA®

- Step-4a: Down Conversion (Kernel 3)
  - Apply the window to multiple frames
  - Each element is handled by a thread.

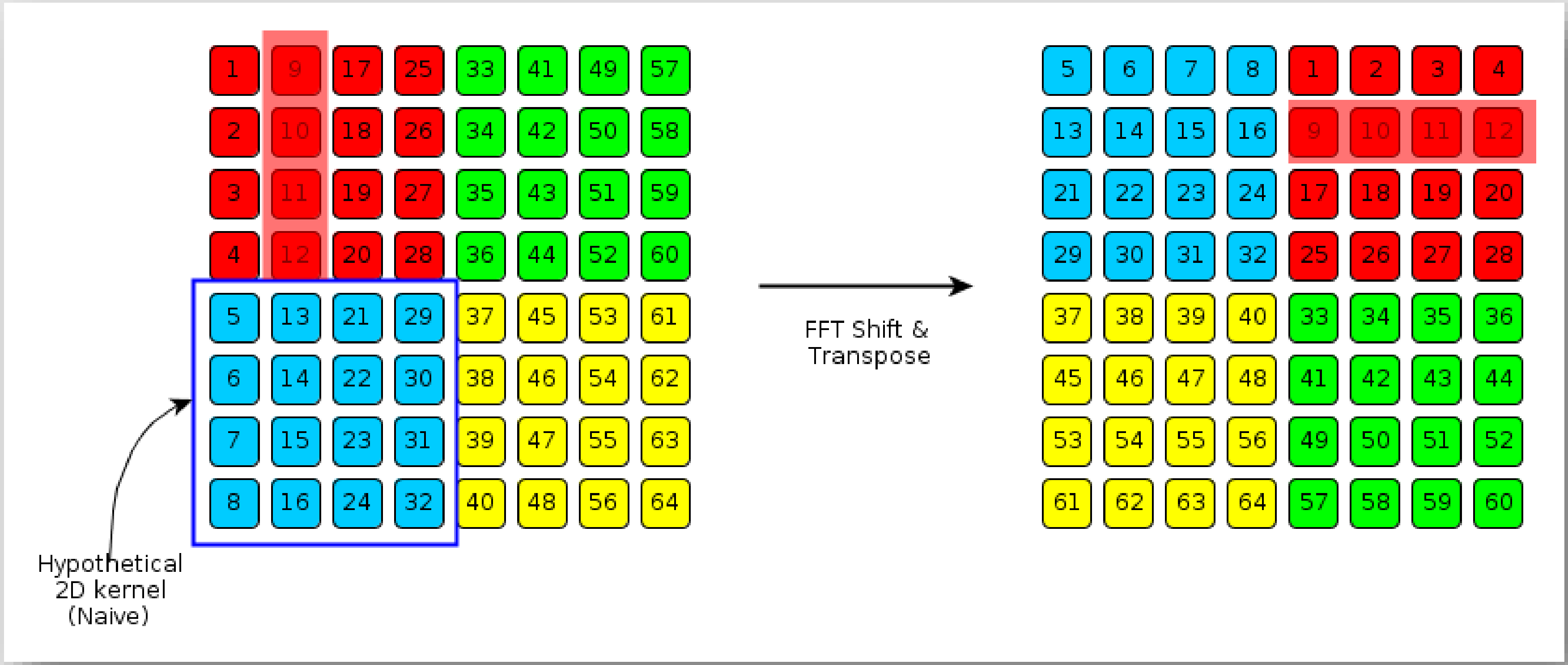


$$out_{km} = fft\_out_{km} * \exp\left(\frac{-2\pi jkmL}{Np}\right)$$

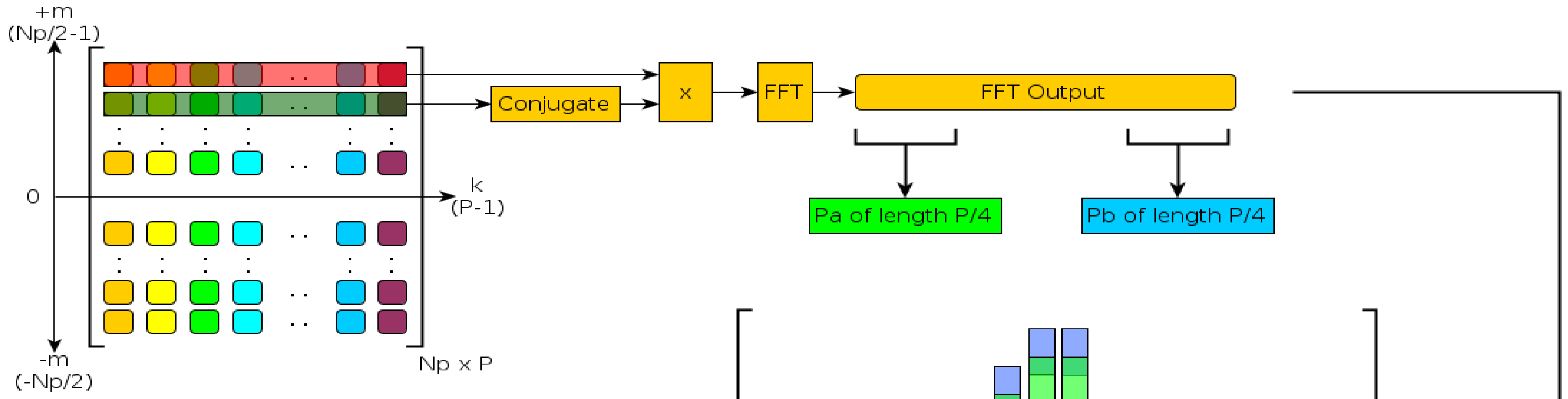
Spectral Correlation Density

THE UNIVERSITY OF ARIZONA®

- Step-4b: FFT Shift and Transpose (Kernel 3)
  - Matrix is arranged in column major order
  - Setting up for coalesced memory access for FFT
  - Color coding shows how the data move before and after the transformation takes place.



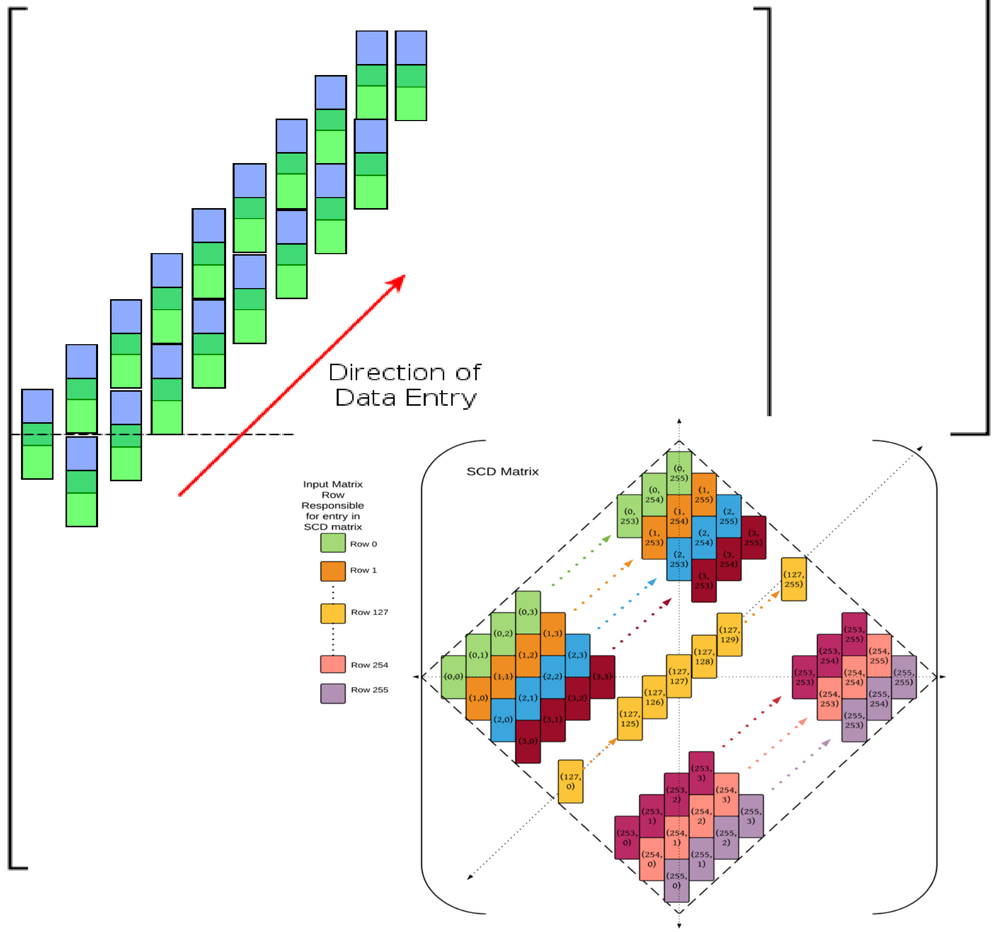
Iterative FFT



- Step-5: Iterative FFT (Kernel 4)
  - Process pair-wise combinations
  - 32*256*256 multiplications
  - 256*256 32pt FFT
    - Generate 65,536 of 16 element (PaPb pairs)

🟦 Pb

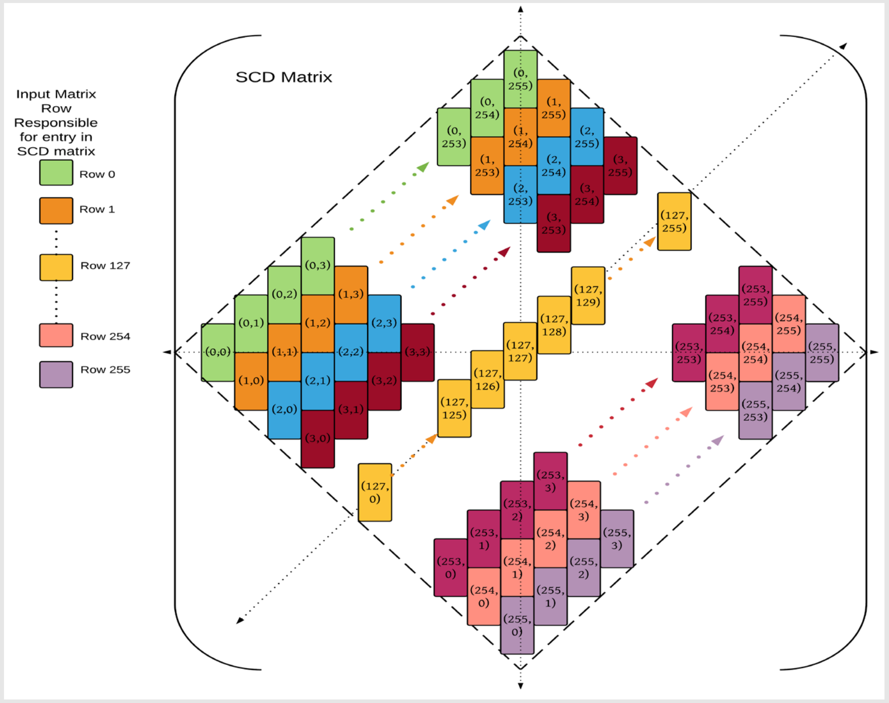
🟩 Pa

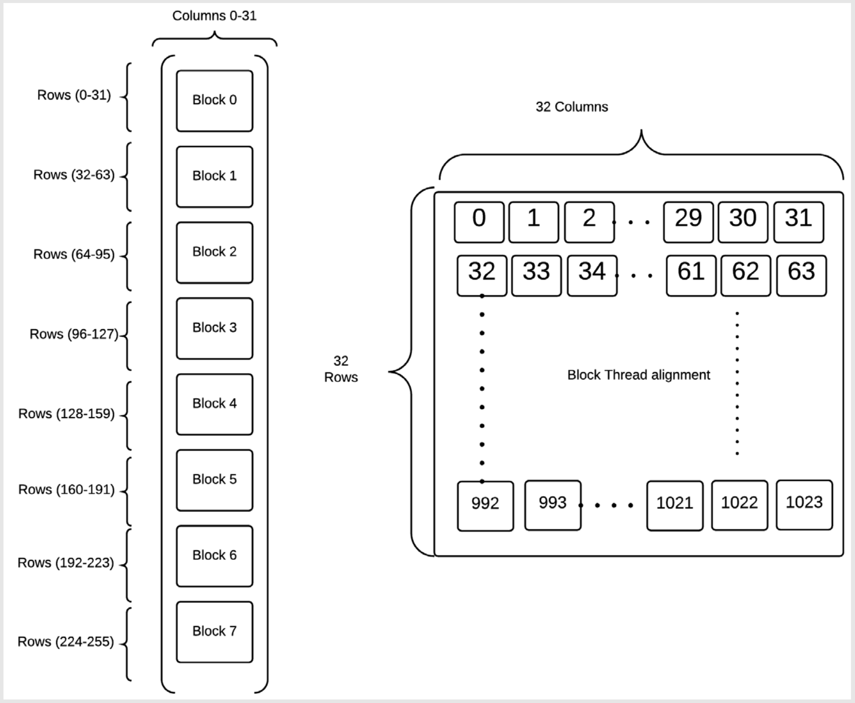


Thread Blocks

THE UNIVERSITY OF ARIZONA

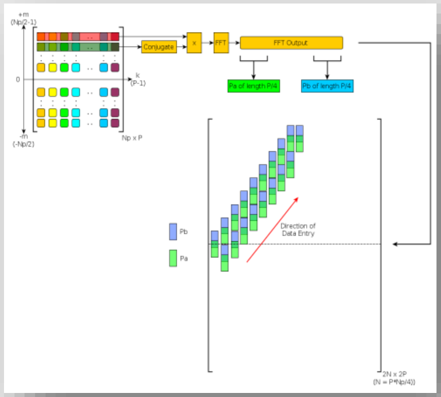
- threads in blocks of 1024 threads to manage
- 32 element multiplications in groups of 32 threads (warps





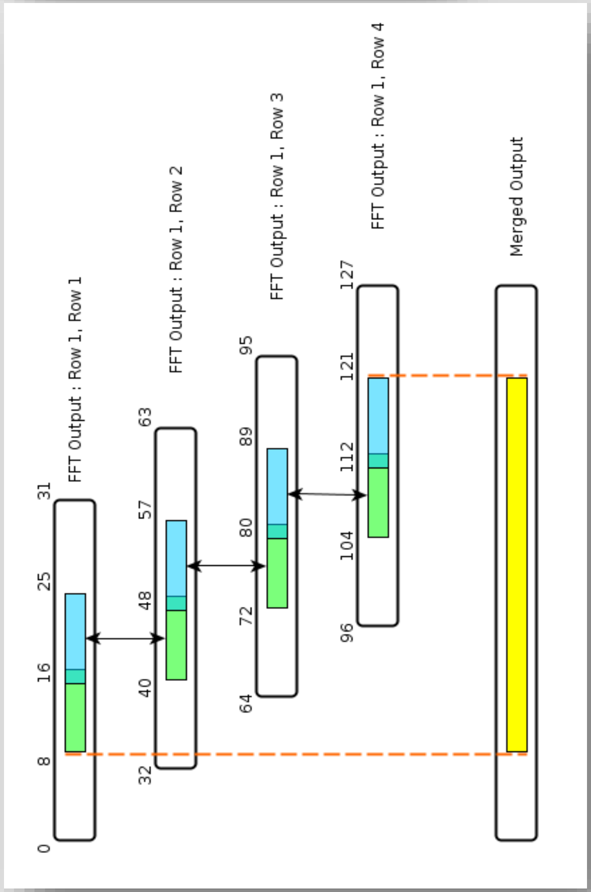
Spectral Correlation Density

THE UNIVERSITY OF ARIZONA

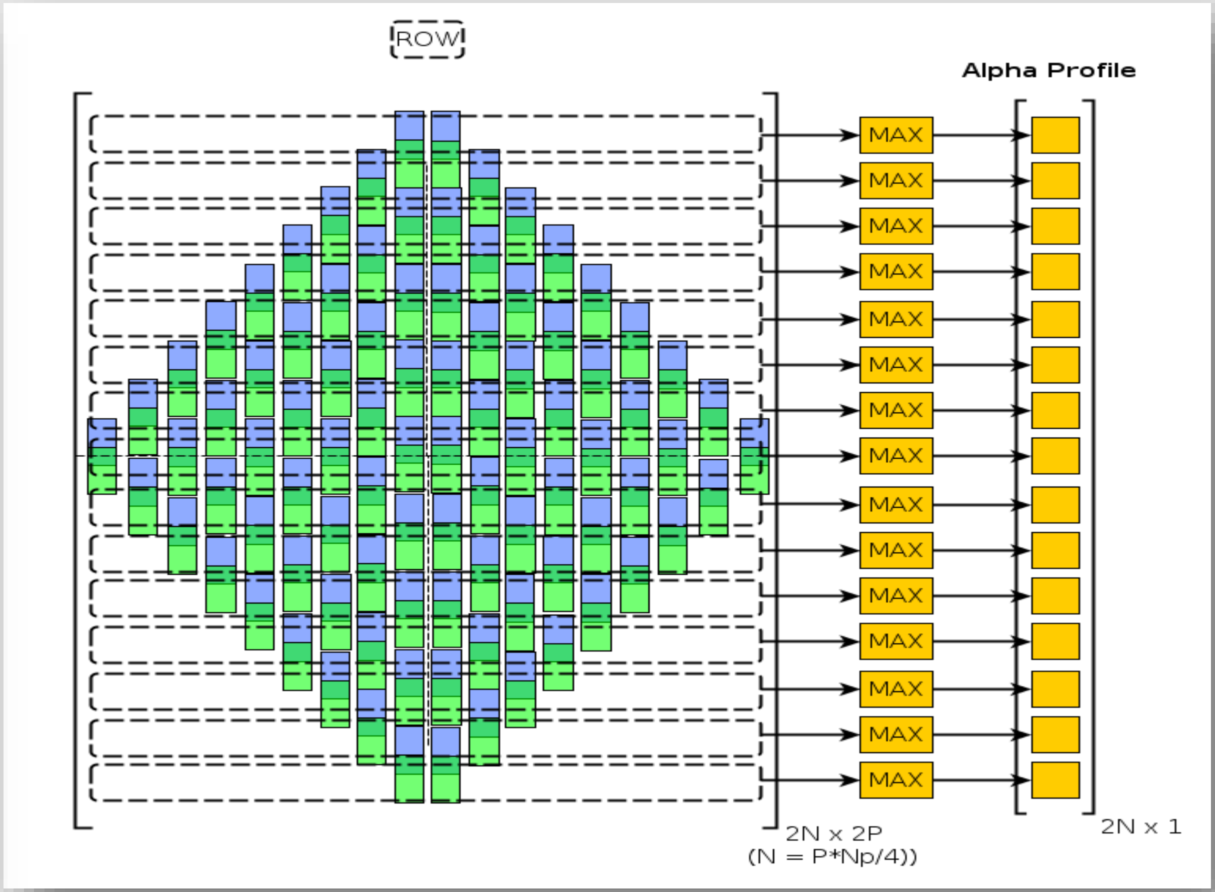


- Step-6: Signal Profile Generation

(Kernel 5)



FFT Output : Row 1, Row 1
FFT Output : Row 1, Row 2
FFT Output : Row 1, Row 3
FFT Output : Row 1, Row 4
Merged Output
0
8
16
25
31
32
40
48
57
63
64
72
80
89
95
96
104
112
121
127



GPU Results

THE UNIVERSITY OF ARIZONA®

- Earlier work
  - Mapped the entire Full SCD process onto NVIDIA GPU

GPU	Matlab Intel I7 3.3GHz 32GB RAM (Reference)	CUDA K20 GPU 706MHz 2496 cores (Earlier work)*	CUDA K40 GPU 745MHz 2880 cores (This study)
Execution Time/Signal (ms) (includes data transfer)	3502.29	8.96	**8.33**
Speedup		390X	420X
Throughput (Signals/sec)		111	120

- Results validated against the Matlab implementation
  - minimum error of 0.0041% and a maximum error of 0.0051%.
- Execution time is based on 4096 points digital signal.

*N. Bidyanta, G. Vanhoy, M. Hirzallah, A. Akoglu, and B. Ryu, "GPU and FPGA Based Architecture Design for Real-time Signal Classification," In Proceedings of the 2015 Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio (WInnComm'15), March 24-26, 2015, San Diego, CA, pp. 70-79.

Profiling and Opportunity

Algorithm 1 SCD Matrix and Alpha Profile Calculation

```
1:  procedure SCD($data[256][32], Alpha\_Profile[4096]$)
2:      for ($i = 0; i < 256; i++$){ do
3:          for ($j = 0; j < 256; j++$){ do
4:              Step 4.1: $x \leftarrow data[i] * Conjugate(data[j])$
5:              Step 4.2: $y \leftarrow FFT\_32(x)$
6:              $SCD\_Matrix \leftarrow \{y[31:24]\}\{y[7:0]\}$
7:              Step 4.3.1: $Calculate\ partial\ Alpha\_Profile$
8:              Step 4.3.2: $Merge\ partial\ Alpha\_Profiles$
9:          }# $end\ of\ inner\ loop$
10:         Step 4.3.3: $Update\ Alpha\_Profile$
11:     }# $end\ of\ outer\ loop$
12:     return $Alpha\_Profile$
```

Inner loop unrolled

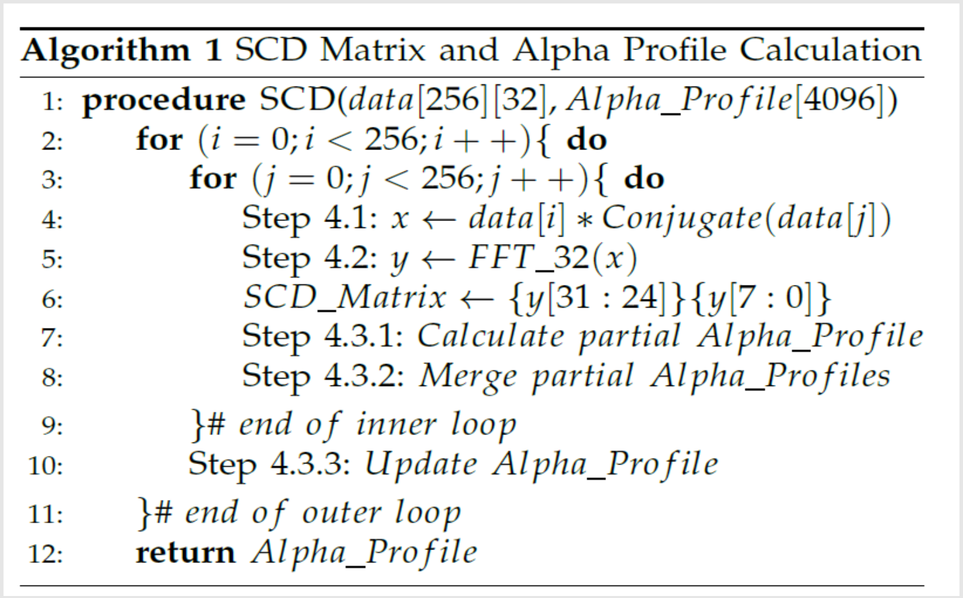
Key Features	K20x	K40
Compute Capability	3.5	3.5
Memory BW (GB/sec)	250	288
Clock (MHz)	732	745
Number of MPs	14	15
Number of Cores	2688	2880
Execution Time (ms)	8.9	8.3
Throughput	111	120

Kernel Name	% Total
1 - Framing	0.13
2 - 256 point FFT	0.26
3 - FFTShift	0.11
4.1 - Conjugate	18.79
4.2 - 32 point FFT	32.93
4.3.1 - Partial	24.69
4.3.2 - Merge	9.67
4.3.3 - Update	13.44

Kernel Launch Overhead

- 4.56 micro seconds on average for a kernel launch (256*5 kernel launches)
- This corresponds to 5.84 ms of time spent on launching kernels out of the total 8.3 ms execution time,
  - which comprises 70% of the total computation time.

What if we merge kernels?



Algorithm 1 SCD Matrix and Alpha Profile Calculation

```
procedure SCD($data[256][32], Alpha\_Profile[4096]$)
    for ($i = 0; i < 256; i++$){ do
        for ($j = 0; j < 256; j++$){ do
            Step 4.1: $x \leftarrow data[i] * Conjugate(data[j])$
            Step 4.2: $y \leftarrow FFT\_32(x)$
            $SCD\_Matrix \leftarrow \{y[31:24]\}\{y[7:0]\}$
            Step 4.3.1: $Calculate\ partial\ Alpha\_Profile$
            Step 4.3.2: $Merge\ partial\ Alpha\_Profiles$
        }# $end\ of\ inner\ loop$
        Step 4.3.3: $Update\ Alpha\_Profile$
    }# $end\ of\ outer\ loop$
    return $Alpha\_Profile$
```

Architecture Specific Optimization

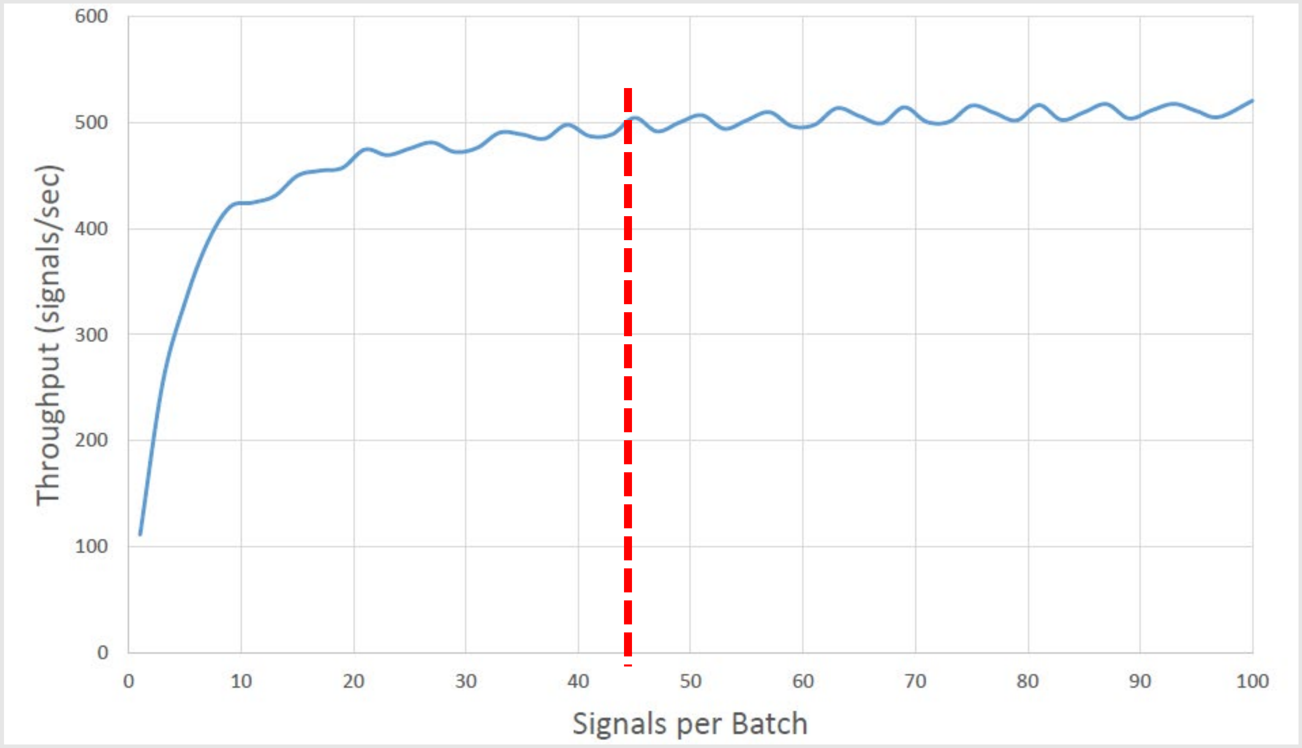
- **Batching**
  - Group multiple signals together in so-called "signal-batches,"
  - Increase number of calculations performed per kernel launch
  - Decrease total number of kernel launches
    - *Original: 5 kernels per signal*
    - *Revised: Batch of "n" signals still uses 5 kernel launches*

Question to answer: What is the optimal batch size?

Architecture Specific Optimization

THE UNIVERSITY OF ARIZONA®

- **Batching – Finding Optimal Batch Size**
  - Size 45 improves throughput from 120 to 503
    - For batch size 100: 1039 signals/second peak throughput



Saturation trend beyond batch size 40 since all threads are occupied. Execution time increases linearly as we add more signals to a batch

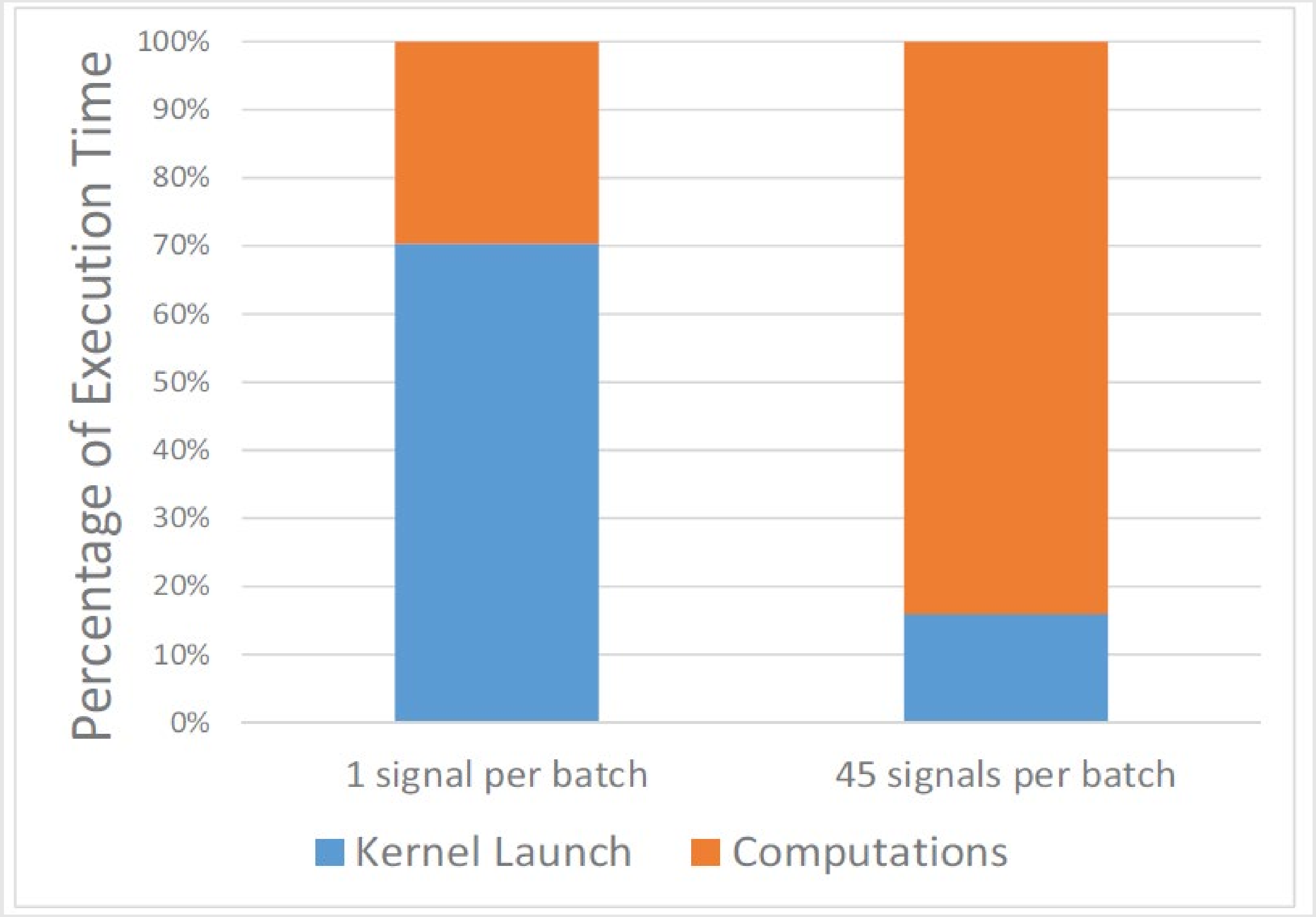
Batching Challenge

- Complicates the utilization of the low-latency shared memory available on the GPU primarily due to memory bank conflicts.
- Therefore during each kernel launch, we ensure that individual signals of a given batch are assigned to distinct thread blocks to avoid bank conflicts.

Kernel Launch Overhead

THE UNIVERSITY OF ARIZONA®

- 70% vs 16%



Batching

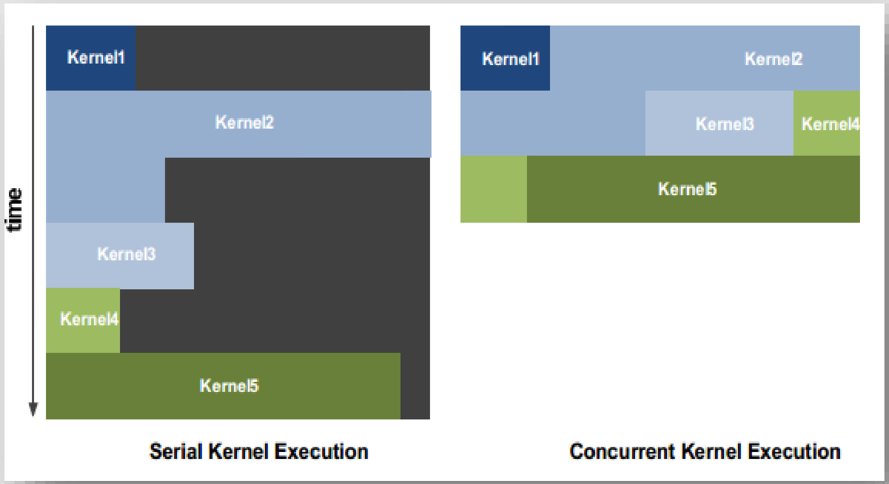
THE UNIVERSITY OF ARIZONA®

- While batching reduces the kernel launch overhead, it doesn't help improve the hardware utilization
  - True or False

Architecture Specific Optimization

THE UNIVERSITY OF ARIZONA®

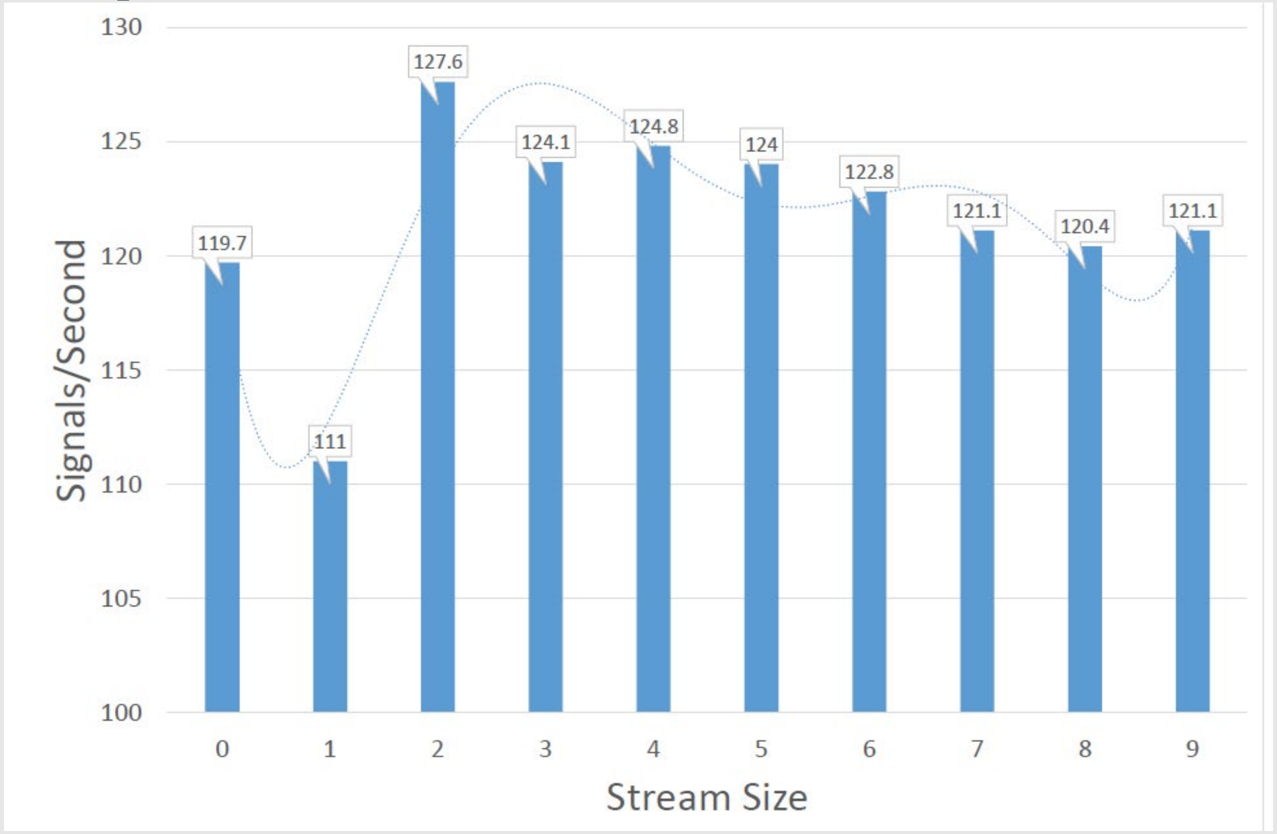
- **Batching with multiple streams**
  - Different signal kernels synchronously enqueued on to ***asynchronous streams***.
  - symmetric multi processors (SMPs)
    - K40 has 15 SMPs
    - Allows streaming multiple signals to be executed concurrently



Question to answer: What is the optimal stream size?

Architecture Specific Optimization

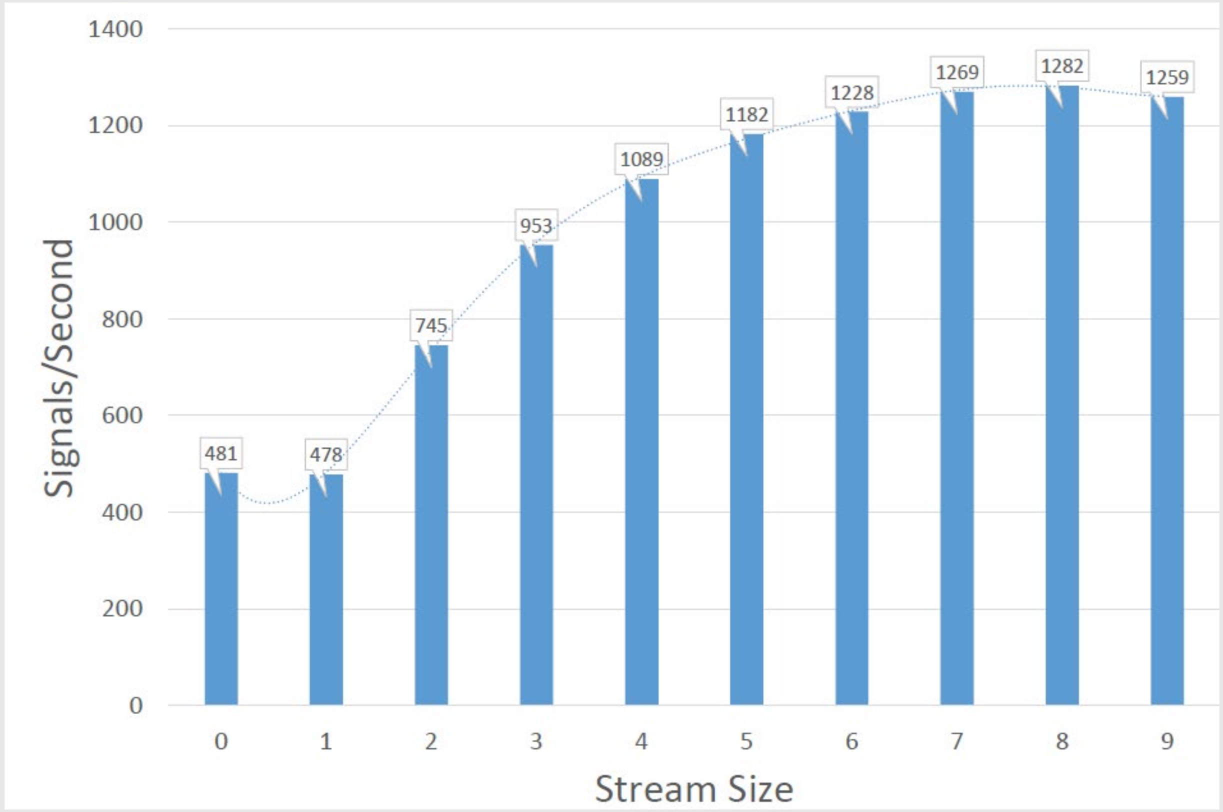
- **Batching with multiple streams– Finding Optimal Stream Size when batch size is 1**



Stream size 0: No streaming, batch of 1
Stream size 1: Stream setup overhead reduces throughput

Why is streaming not helping?

- **Batching with multiple streams– Finding Optimal Stream Size when batch size is 45**



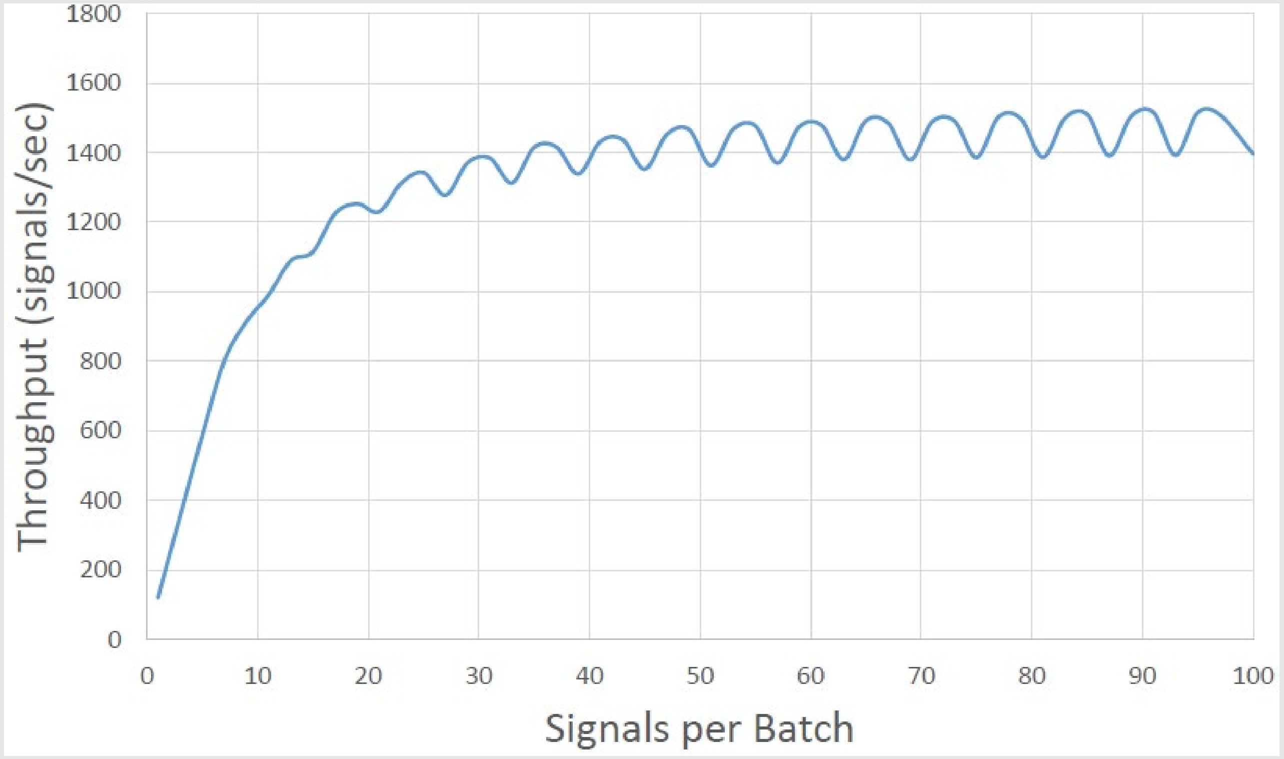
8 streams increases throughput 1282

Stream size 0: No streaming, batch of 45
Stream size 1: Stream setup overhead reduces throughput of batch size 45

Architecture Specific Optimization

THE UNIVERSITY OF ARIZONA®

- **Streaming with multiple batches– Finding Optimal Batch Size when stream size is 8**

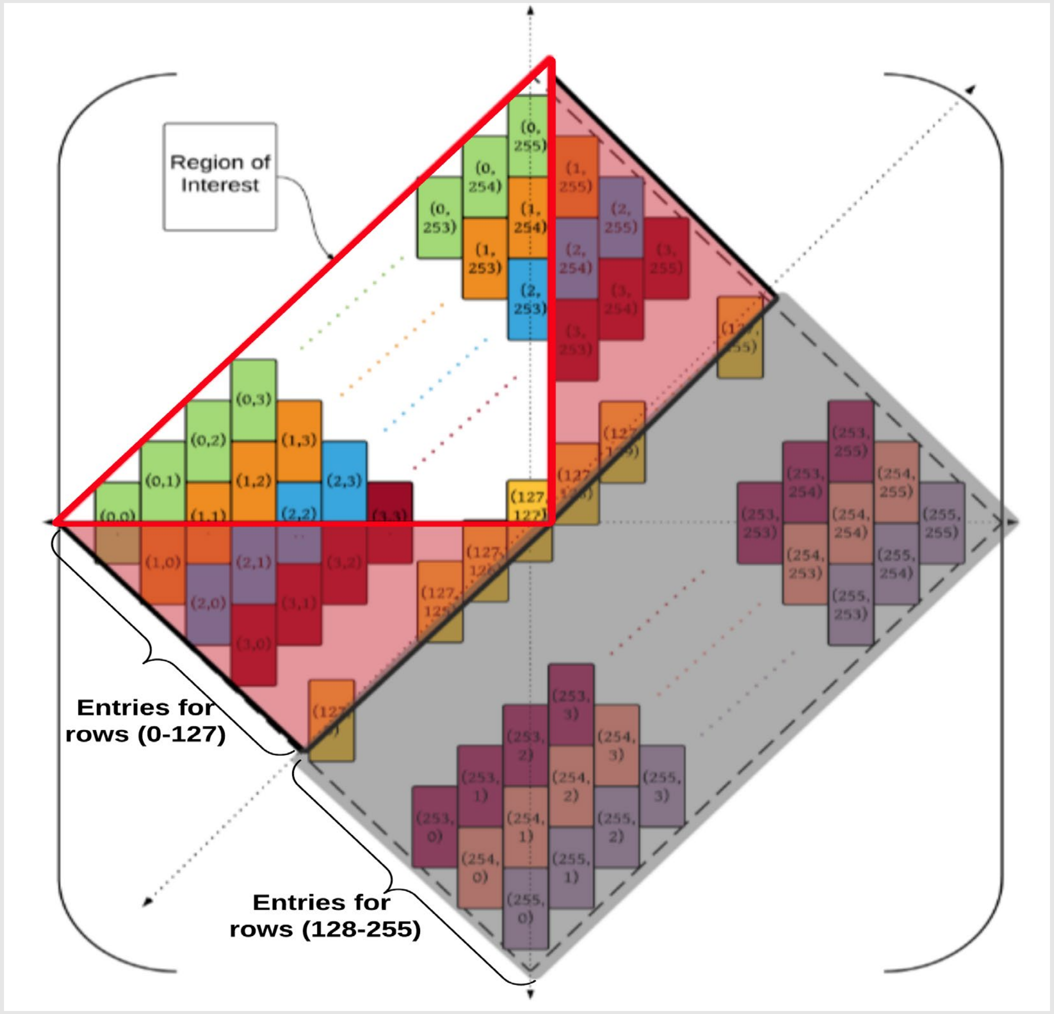


Peak throughput 1569 with batch size 91

Stream size 0: No streaming, batch of 45
Stream size 1: Stream setup overhead reduces throughput of batch size 45

Quarter SCD - Data trimming

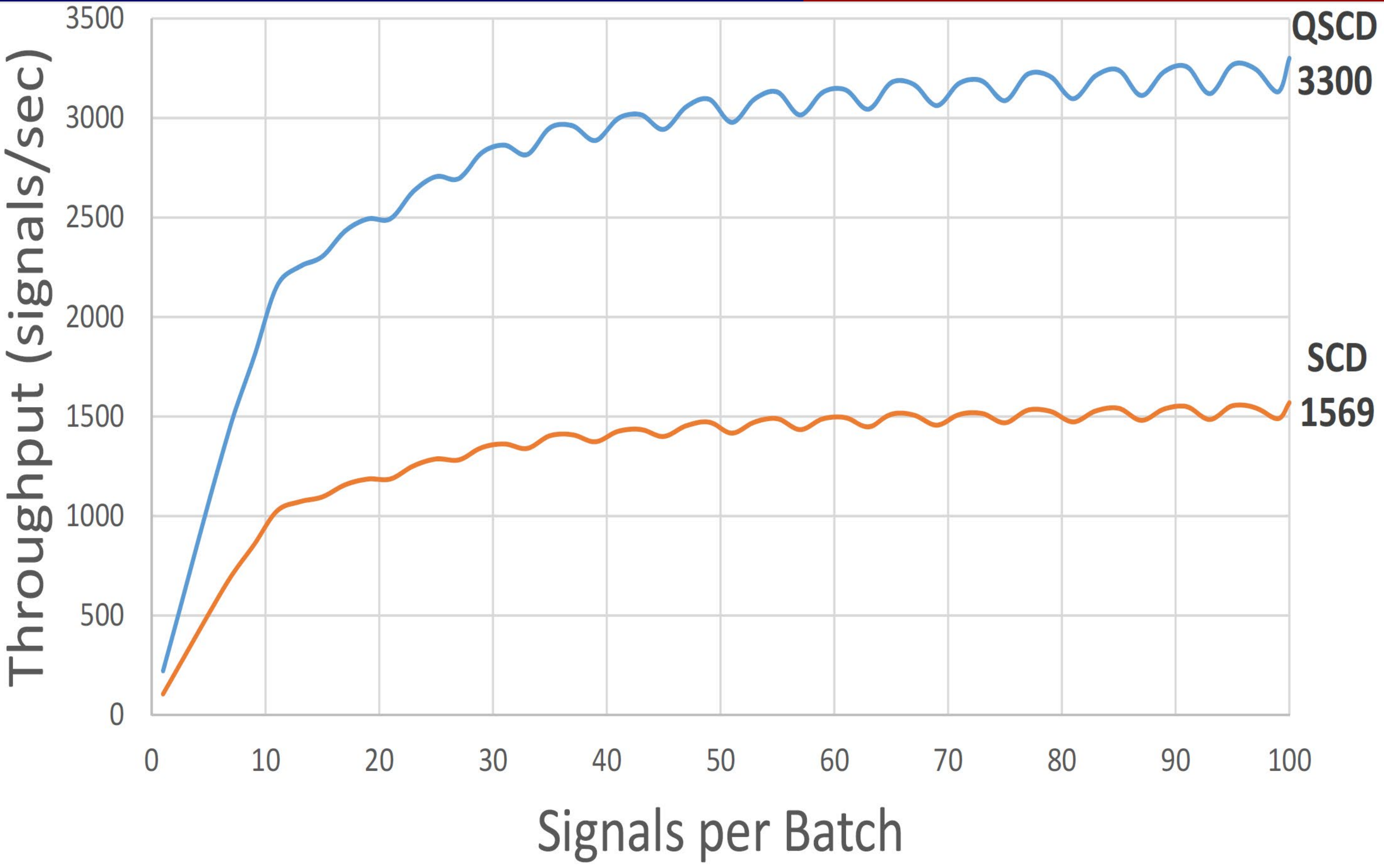
- Take advantage of the symmetry in the SCD estimate



- Divide SCD matrix into 4 Cartesian quadrants, and cells receding in top-left quadrant (quadrant 2) become the relevant cells for calculating alpha profile of the signal
  - Amount of computations and data transfers shrinks significantly

Stream Size Fixed

THE UNIVERSITY OF ARIZONA®



Conclusions

- Algorithmic and architecture specific optimizations
  - Quarter SCD
  - Batching
  - Streaming
- Peak throughput performance improvement over reference full SCD on GPU
  - 13.7x with batch size 45, stream size 9
    - 2719/signals/second
  - 27.5x with batch size 100, stream size 9
    - 3300 signals/second