

LTE eNodeB/UE PHY Layer Implementation on General Purpose CPU and GPU

Supervisor

Dr. Ahmed Hesham



Presented By

Ahmed Mostafa

Khaled Ahmed

Mohamed Mostafa

Ahmed Nour

Mohamed Osama

Introduction

- Communication systems require complex operations like (FFT/IFFT).
- Fast and powerful hardware is needed.
- Special Purpose Processors are commonly used (like DSPs).



DSPs

Advantages

- Instruction set contains complex mathematical operations (Like FFT).
- Ability to execute SIMD instructions.
- These advantages made DSPs commonly used in communication systems.



DSPs

Disadvantages

- Subsequent LTE releases may require upgrading the hardware (cost).
- Upgrading the hardware may lead to software upgrading (waste of time).
- The reuse of the codes over different hardware may lead to several conflicts.



Solution: General Purpose Processors

Advantages

- It could solve the problem of upgradability.
- Compatibility solve software reusability problem.



Solution: General Purpose Processors

Disadvantages

- Complex mathematical operations (like FFT) are performed using basic mathematical Operations.
- GPP are slower than DSPs in doing the required operations.

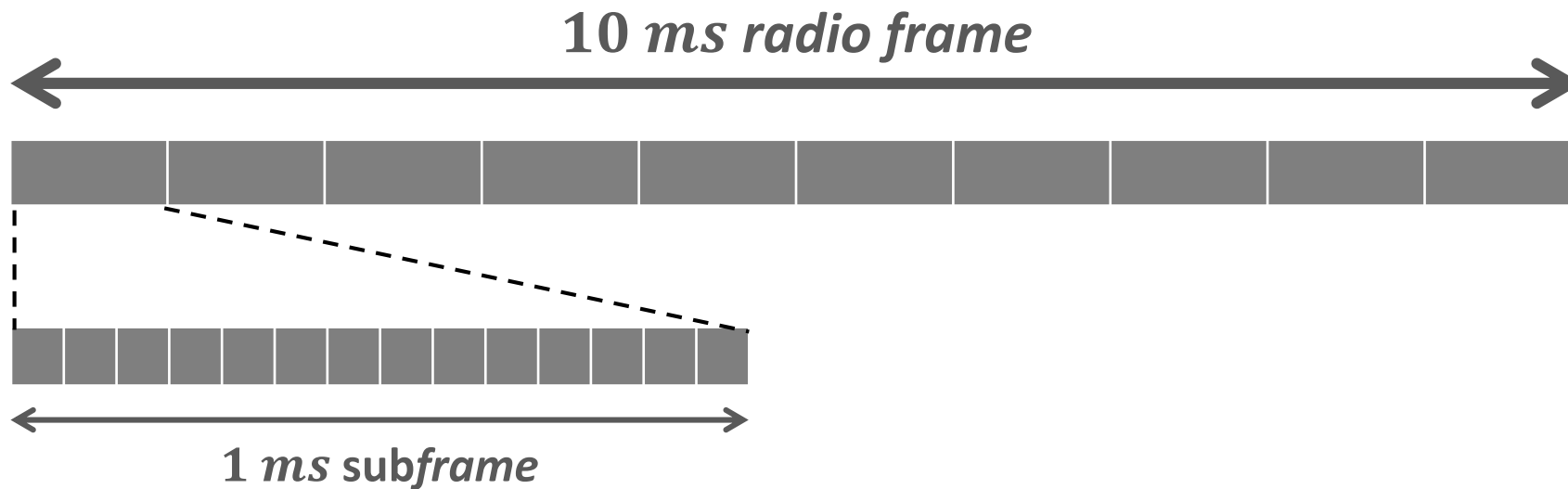
However, multiple parallelism techniques can be used:

- Intel AVX and Intel MKL
- Nvidia CUDA



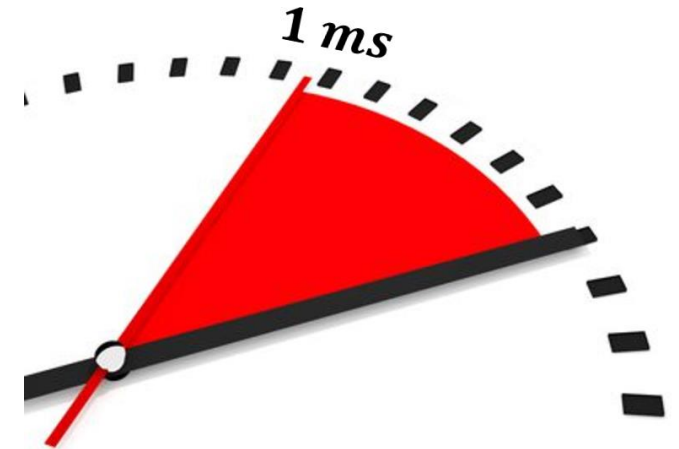
Project Objective

Implementing LTE's PUSCH chain for both Tx & Rx on GPP and achieve a subframe processing time $\leq 1\text{ ms}$.



Previous year

- Implementation of **Rx** chain only on GPU without DMRS generation.
- No **MIMO** support.
- Subframe processing time = **1.4 ms** neglecting API calling time.
- Subframe processing time = **4.7 ms** with API calling time.
- Subframe processing time on our platform = **2 ms**.



This year

- Implementation of both **Tx** & **Rx** chains on GPP and GPU.
- **MIMO** support on both MATLAB and GPU.
- Subframe processing time for Rx = **0.77 ms**.

Learning Steps

- LTE Fundamentals through Rohde and Schwarz documents.
- 3GPP standards (releases 8, 9 and 10) for LTE (PUSCH).
- Parallel Programming Course using CUDA (Udacity online course).
- AVX instruction (Supported in Intel® Core™ i7 Processors).



Project Milestones

- Implementing PUSCH using MATLAB LTE System Toolbox built-in functions.
- Implementing PUSCH chain on MATLAB with the help of the openLTE and testing its results versus the MATLAB LTE System Toolbox results.



Project Milestones

- Implementing the PUSCH on Intel® Core™ i7 processors (ANSI-C code) and test it versus MATLAB results.
- Implementing the PUSCH with CUDA and test it versus MATLAB results.
- Optimizing the C code implementation for SISO chain on Nvidia.
- Implementing 2x2, 4x4 and 64x64 MIMO chains.
- Obtaining time profiling results for both SISO and MIMO PUSCH chains.



Version Control Software


Private GitHub Repository


- Easy to get last updated version.
- Full Local History.
- Restoring Previous Versions.
- Easier teamwork.





Version Control Software

graduation project " LTE eNodb L1 Processing Core ix and GPU "

 274 commits

 1 branch

 0 releases

 5 contributors

Branch: master ▾


New pull request

Create new file

Upload files








Find file

Clone or download ▾

 AhmedMoustafaHosni

GP book version 2

Latest commit ac16629 2 days ago

 CUDA	siso receiver with channel equalization	4 days ago
 GP Book	GP book version 2	2 days ago
 Intel	Total SISO Chain / all modification needed	3 months ago
 Matlab	Update readme.txt	8 days ago
 docs	cuda documentation	4 months ago
 _config.yml	Set theme jekyll-theme-cayman	6 months ago
 index.md	Set theme jekyll-theme-cayman	6 months ago

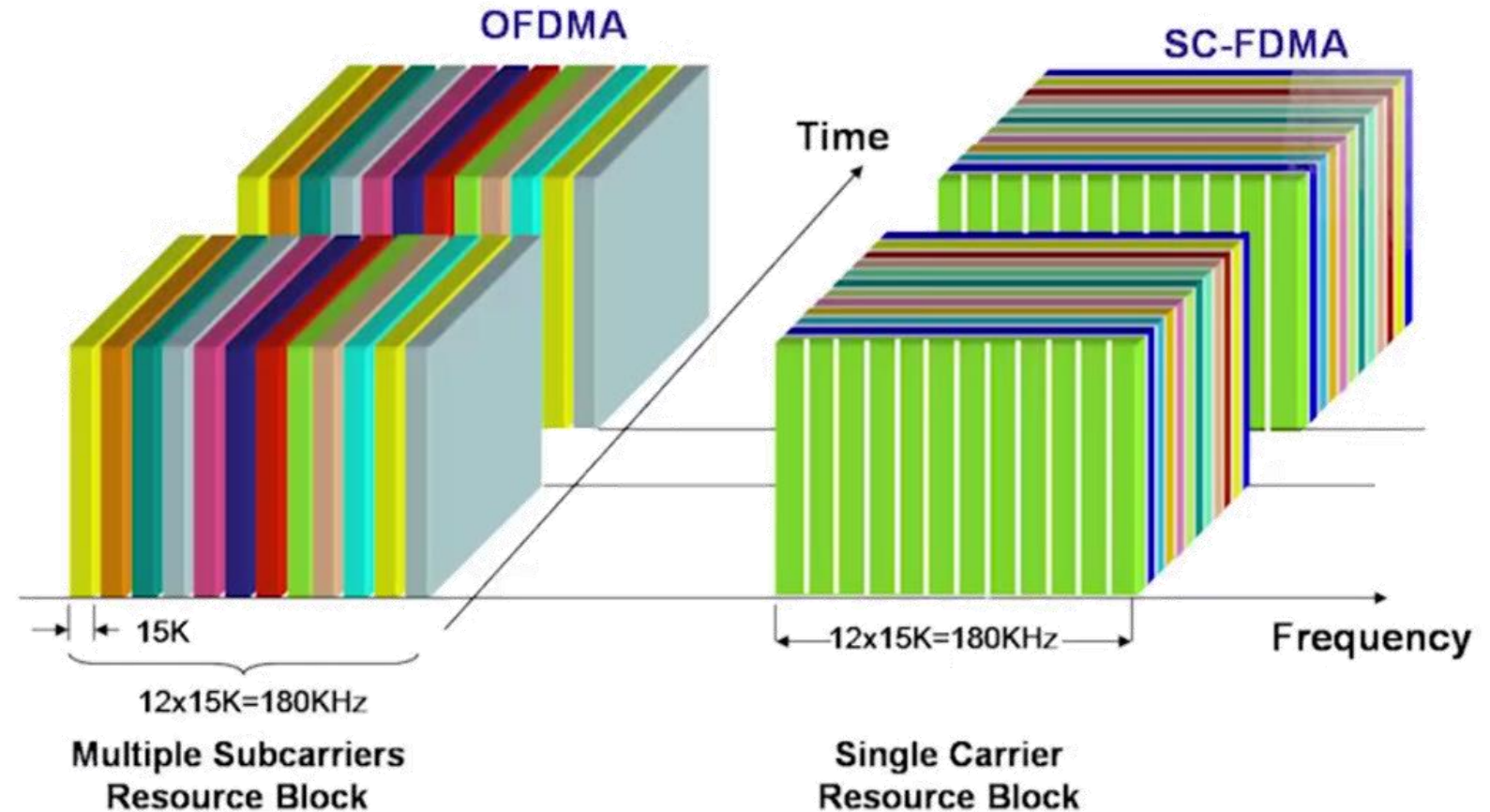
LTE

- LTE (Long Term Evolution) is the brand name proposed by 3GPP as the 4th Generation technology for mobile telecommunications.
- LTE has been first time introduced in 3GPP Release 8.
 - Downlink Speeds (for 20 MHz):
 - 100 Mb/s SISO (Single Input Single Output);
 - Uplink Speeds (for 20 MHz):
 - 58 Mb/s (16 QAM)
 - Bandwidth
 - 1.4 MHz up to 20 MHz.



Air Interface

- Downlink: **OFDMA**
- Uplink: **SC-FDMA**



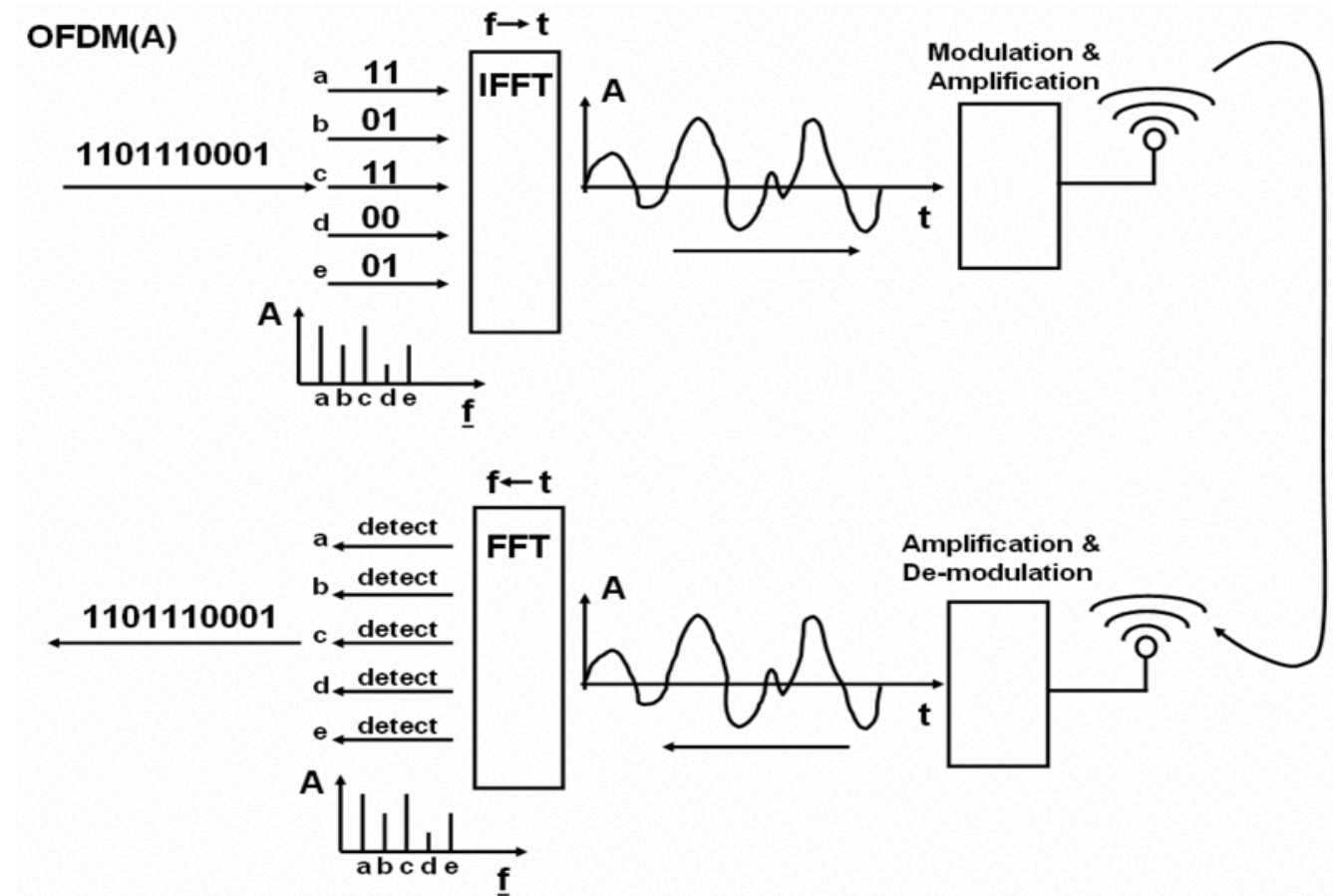
OFDMA

Advantages:

- Spectrum Efficiency (Orthogonal Carriers).
- Immunity to selective fading channel.
- Immunity to ISI.

Disadvantages:

- More sensitive to frequency offset.
- Large Peak-to-average-power-ratio (PAPR).



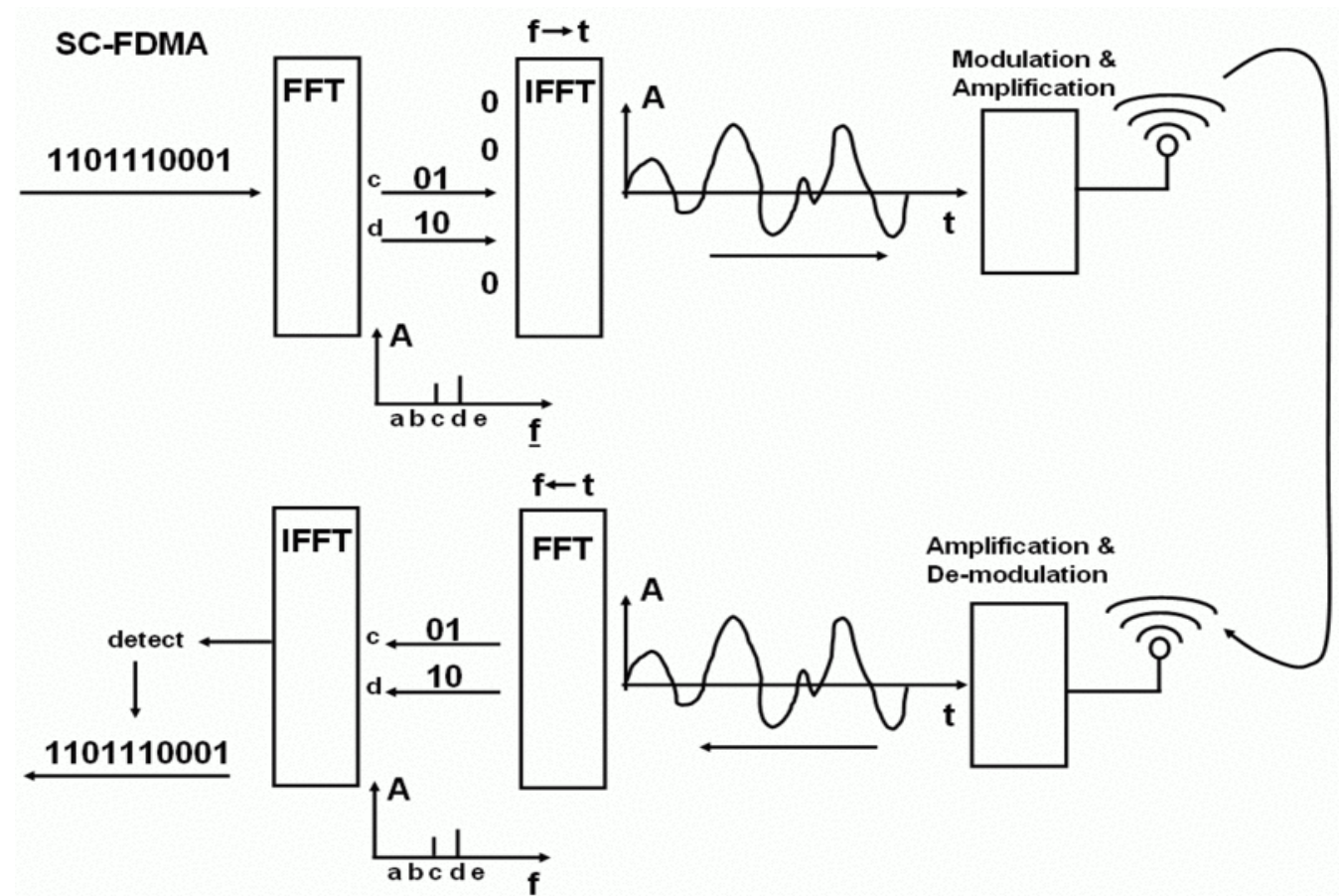
SC-FDMA

Advantage:

- Low Peak-to-average-power-ratio (PAPR).

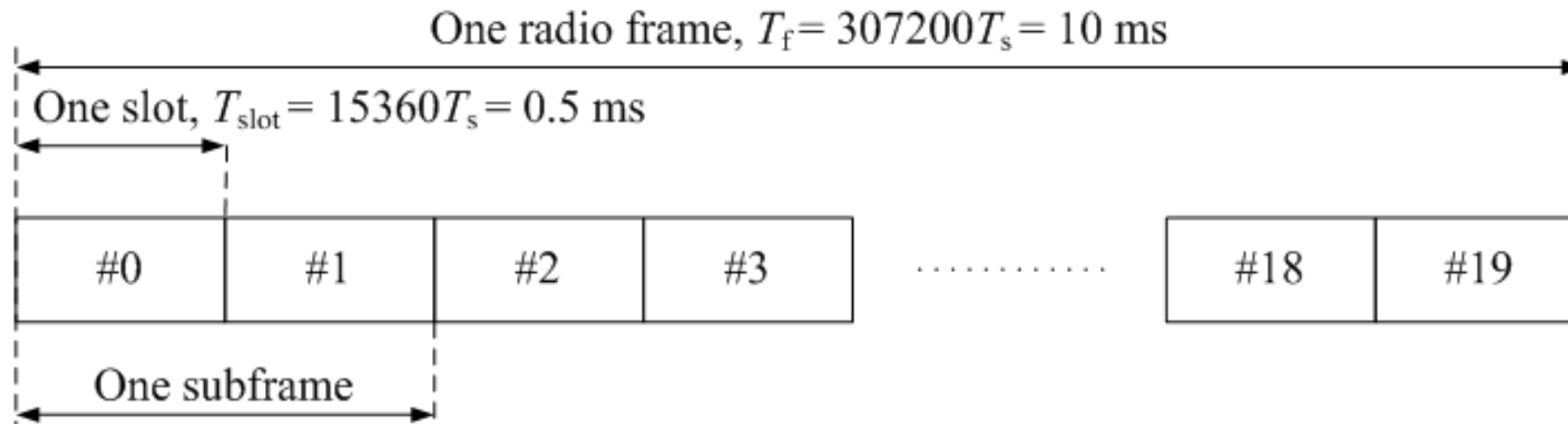
Disadvantage:

- Complexity of the Rx due to FFT block.



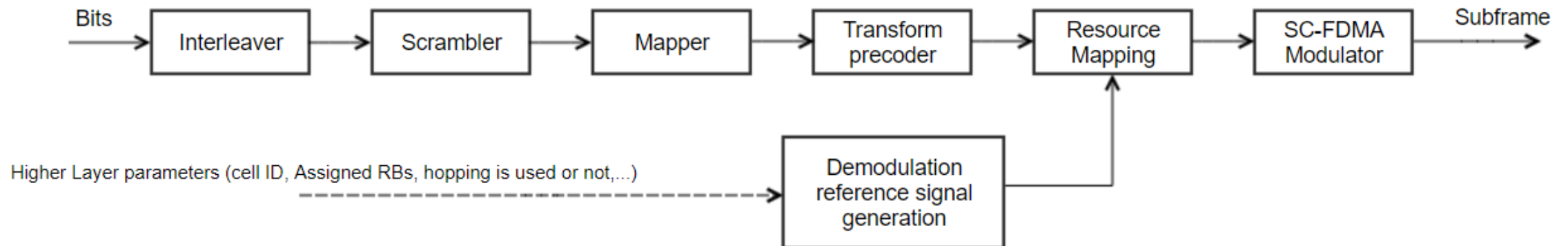
Physical Uplink Shared Channel

Frame structure



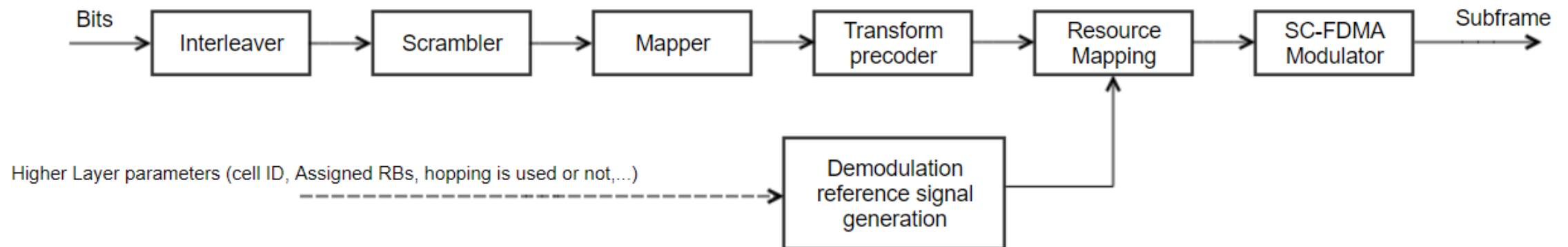
Physical Uplink Shared Channel

Block Diagram (SISO)



Interleaver

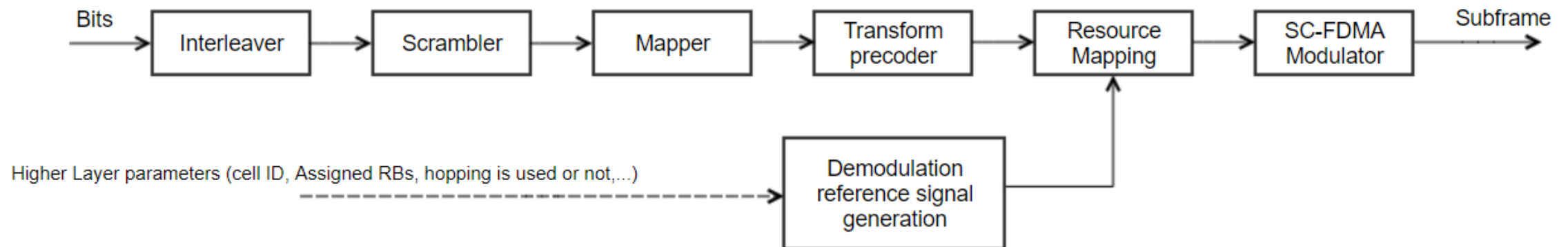
Bit shuffling to reduce the burst errors that might occur when the message bits are transmitted through a noisy channel.



Scrambler

Changing pseudo-randomly the values of bits in a data block to:

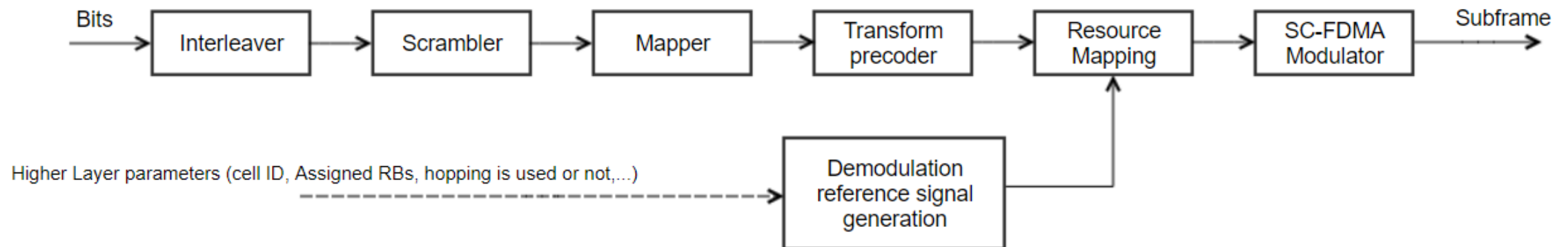
- Reduce electromagnetic interference;
- Introduce security.



Modulation Mapper

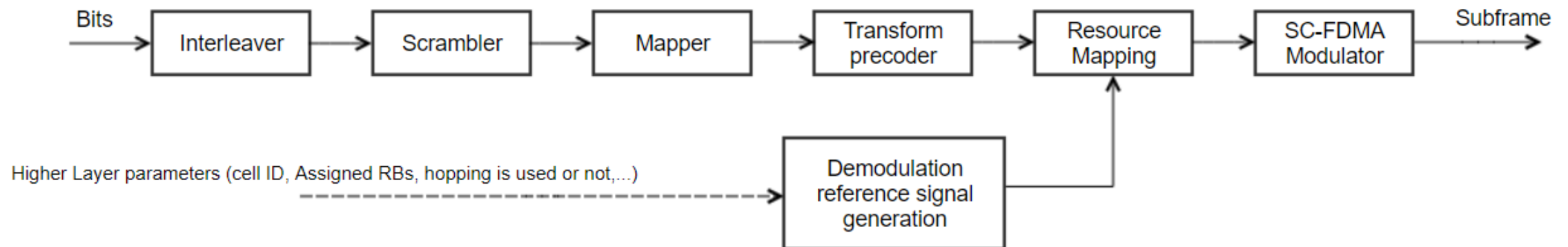
Bits are modulated resulting in a block of complex-valued symbols.

- BPSK (only in control channel)
- QPSK
- 16QAM
- 64QAM



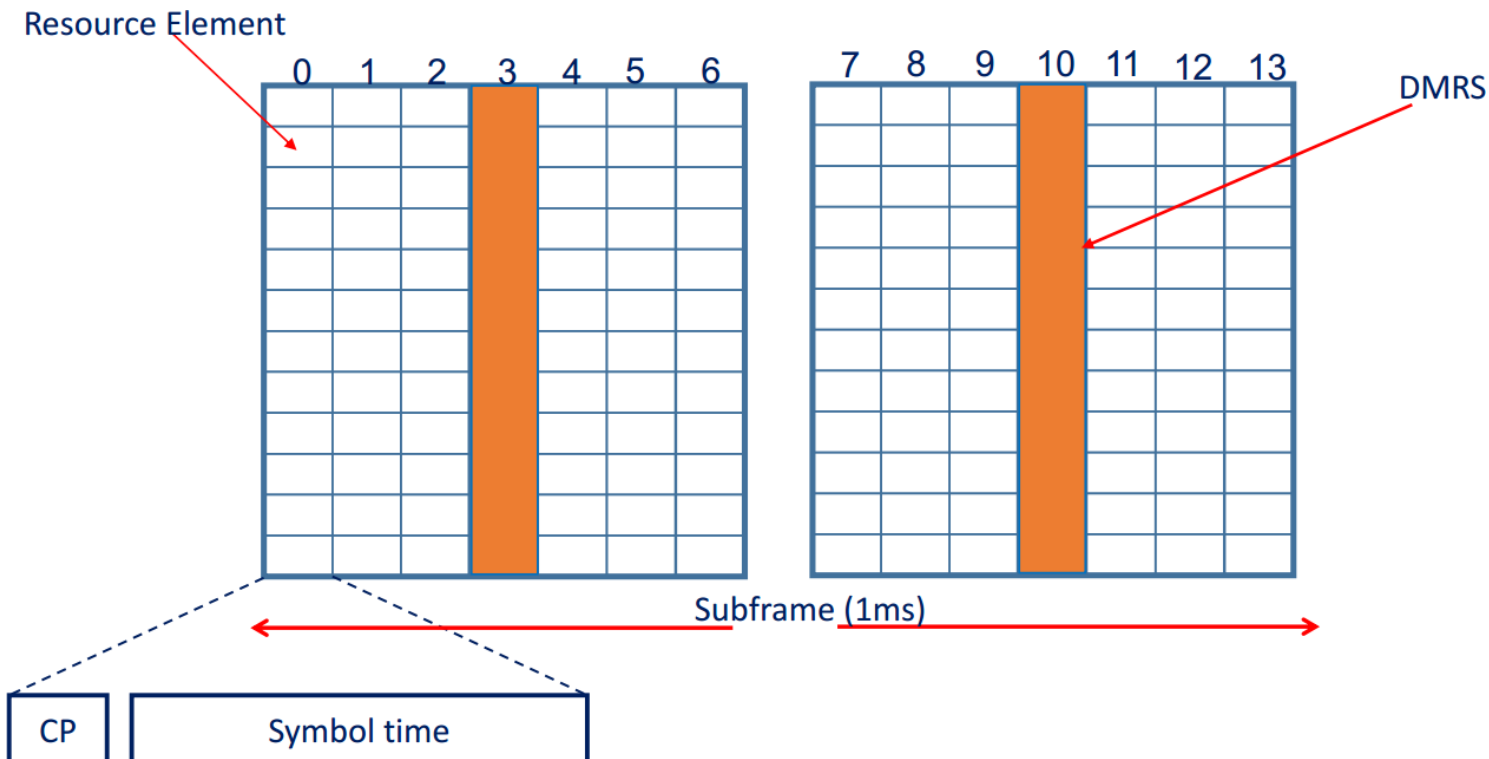
Transform Precoder

Complex-valued symbols are divided into sets, each corresponding to one SC-FDMA symbol.



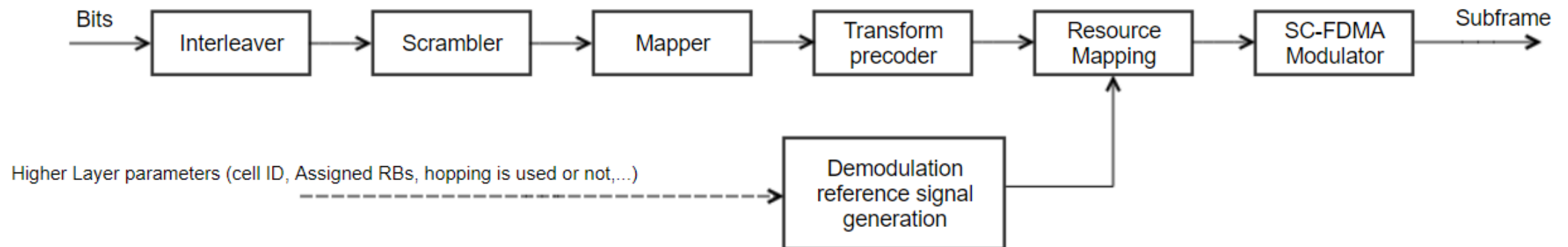
Mapping to Physical Resources

Composing the subframe by multiplexing the DMRS signal and data.



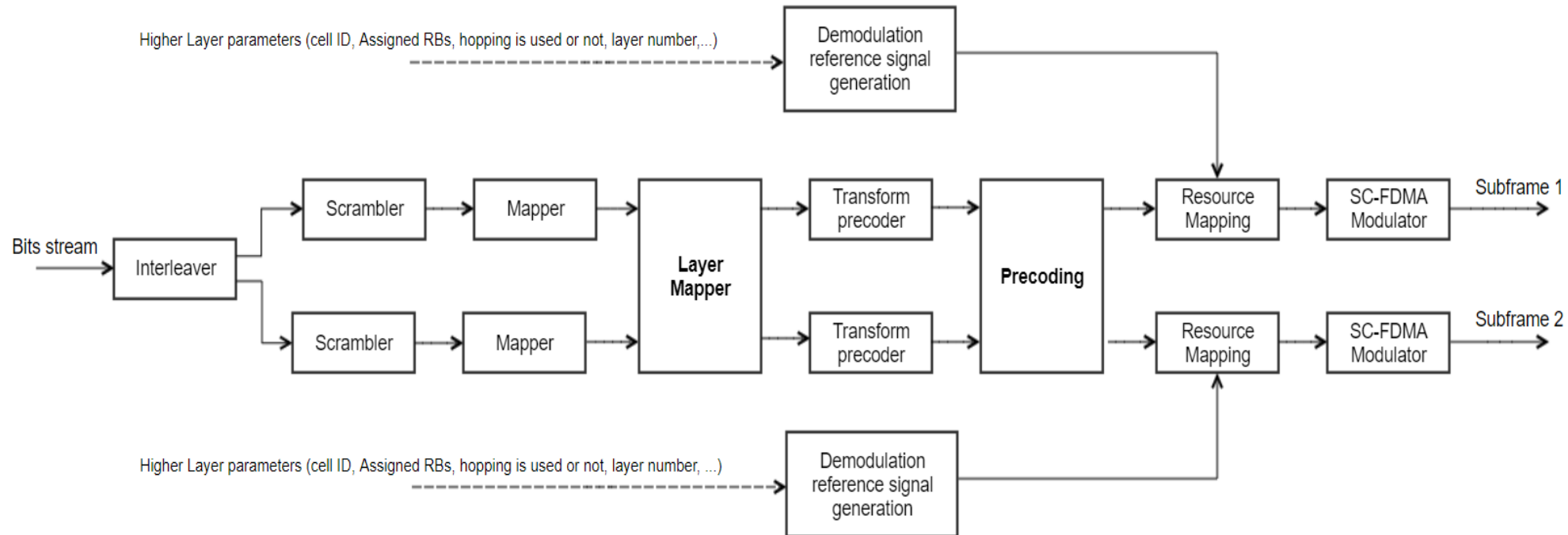
SC-FDMA Signal Generation

Perform IFFT for each SC-FDMA symbol and add cyclic prefix at the beginning of each symbol.



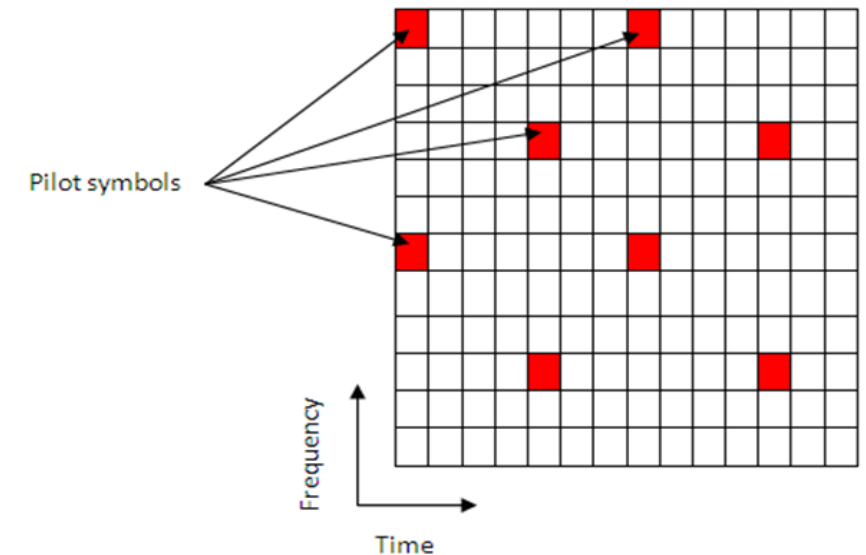
Physical Uplink Shared Channel

Block Diagram (MIMO)

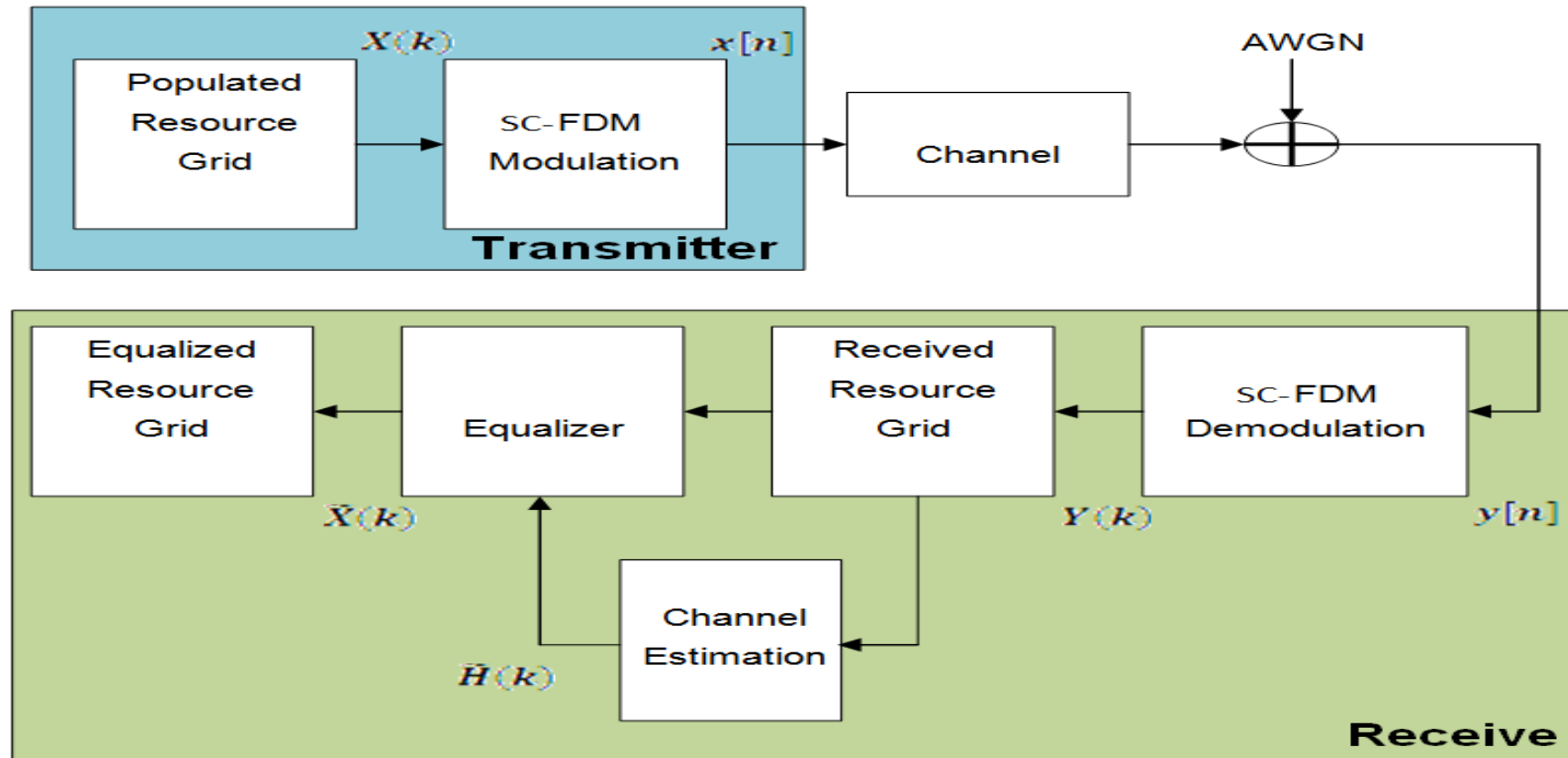


Channel estimation and equalization

- It is used to improve the system performance in terms of bit error rate.
- To facilitate the estimation of the channel characteristics DMRS inserted in both time and frequency.
- These pilot symbols provide an estimate of the channel at given locations within a subframe.



Channel estimation and equalization



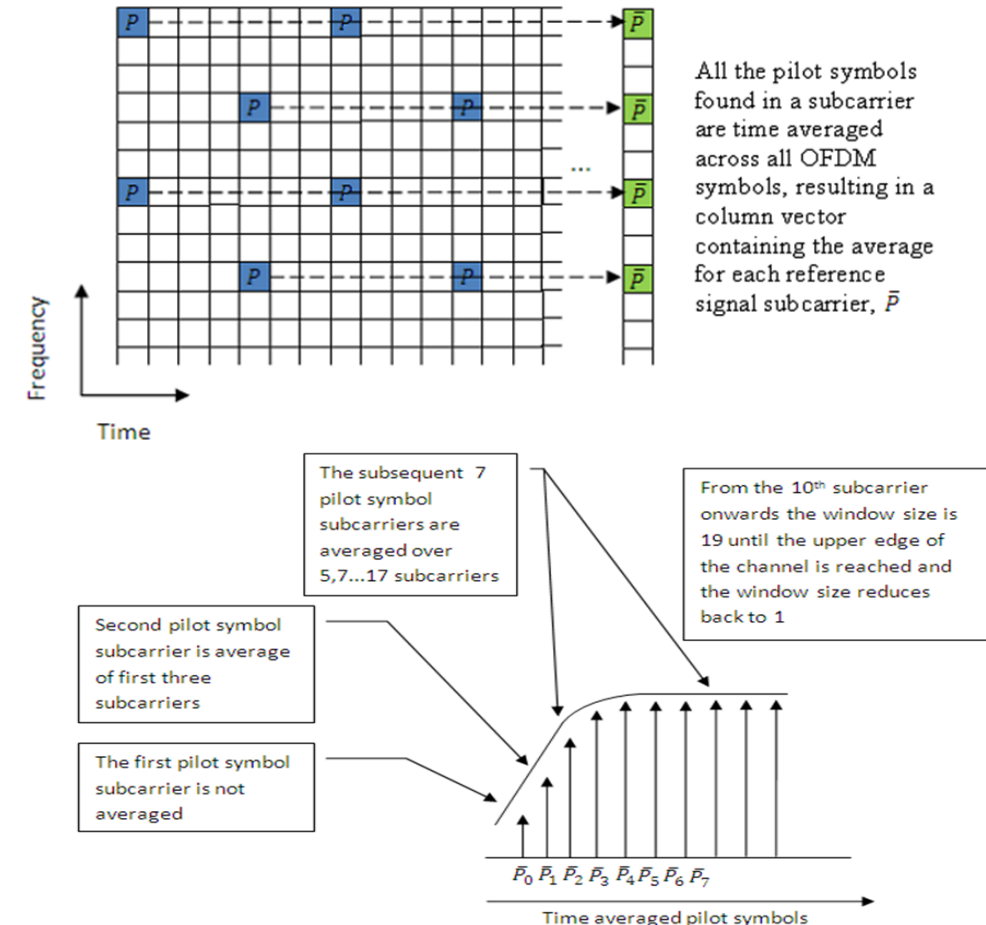
Channel estimation

1- Pilot estimates = $\frac{\text{received reference signal}}{\text{generated reference signal}}$

2- Get the time average of the pilot estimates.

3- Get the frequency average by using a moving window of maximum size 19.

4- The resulting vector is then replicated and used as the channel estimate for the entire resource grid.



Channel equalization

Received symbols

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}$$

We need to find W where $WH = I$

1- zero forcing: $W = (H^H H)^{-1} H^H$

2- MMSE: $W = (H^H H + N_0 I)^{-1} H^H$

Channel model

The LTE System Toolbox™ provides a set of channel models for the test and verification of UE:

- Extended Pedestrian A model (EPA)
- Extended Vehicular A model (EVA)
- Extended Typical Urban model (ETU)

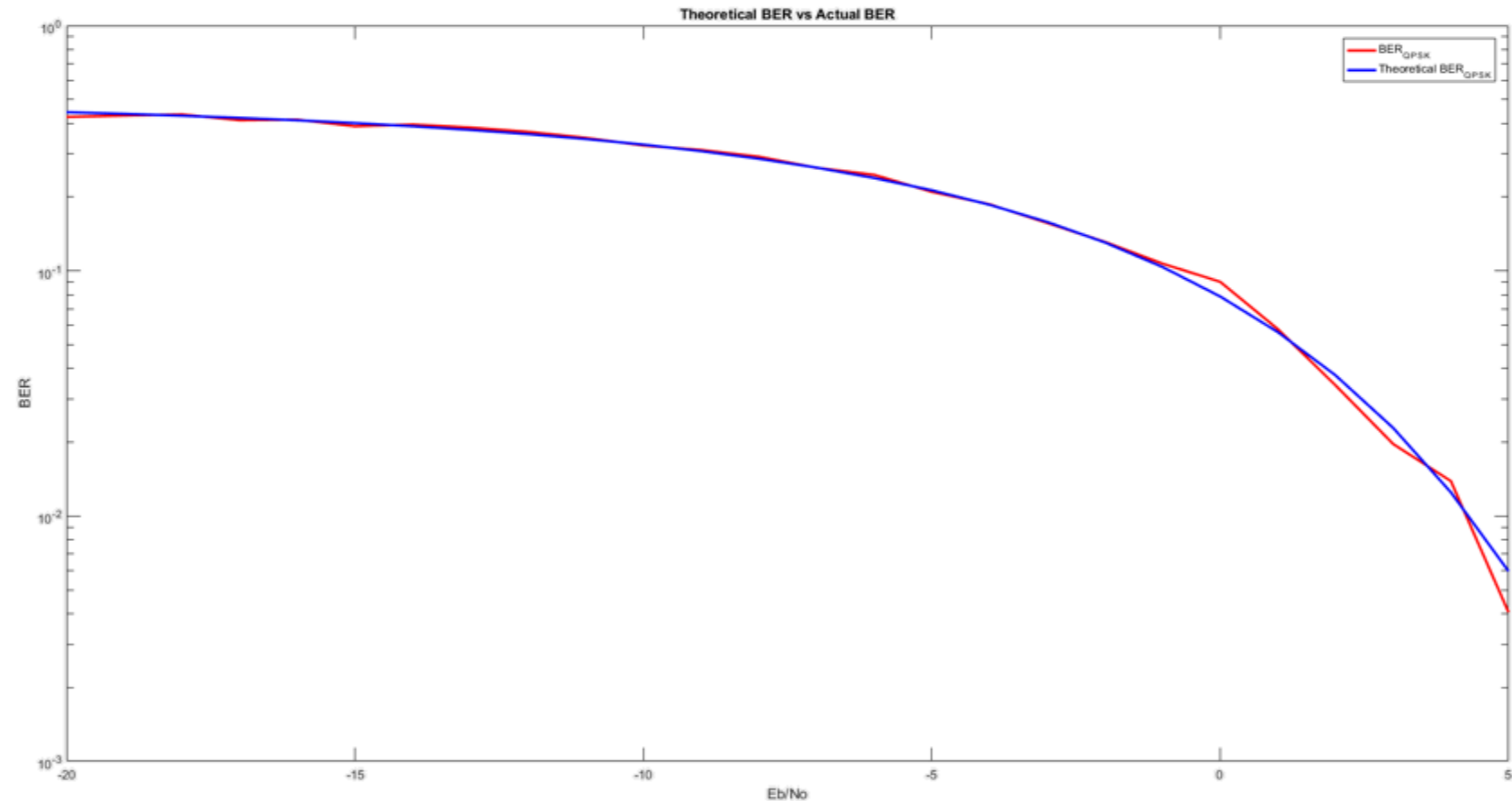
EPA Delay Profile

Excess Tap delay (ns)	Relative Power (dB)
0	0.0
30	-1.0
70	-2.0
90	-3.0
110	-8.0
190	-17.2
410	-20.8

MATLAB Simulation

1. SISO Tx & Rx full chain.

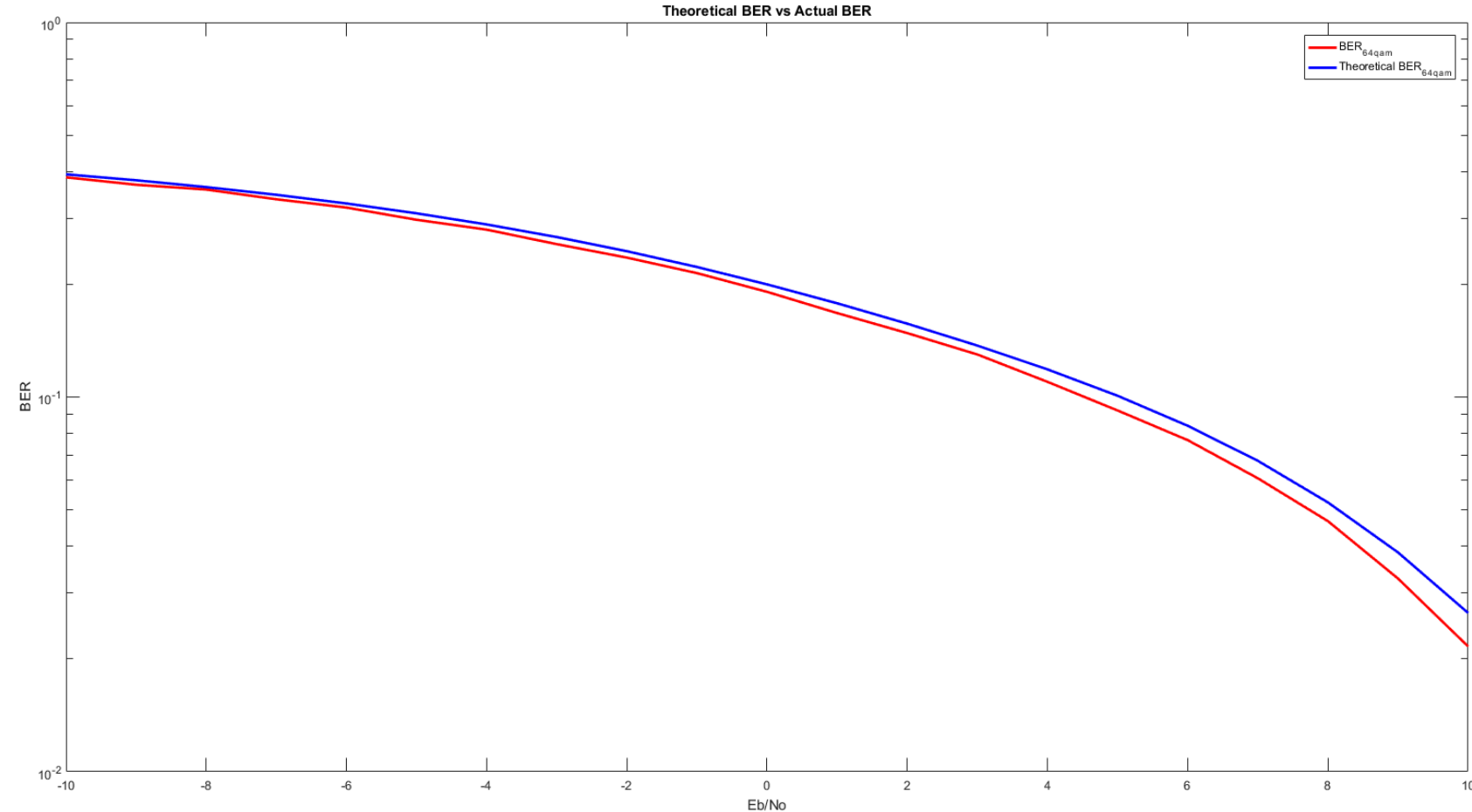
- RBs assigned to the UE: **100**
- Modulation: **QPSK**
- Number of bits: **28800 bits**
- Channel: **AWGN**
- SNR: $(-20 \rightarrow 5) \text{ dB}$
- **No** channel estimation.



MATLAB Simulation

2. SISO Tx & Rx full chain.

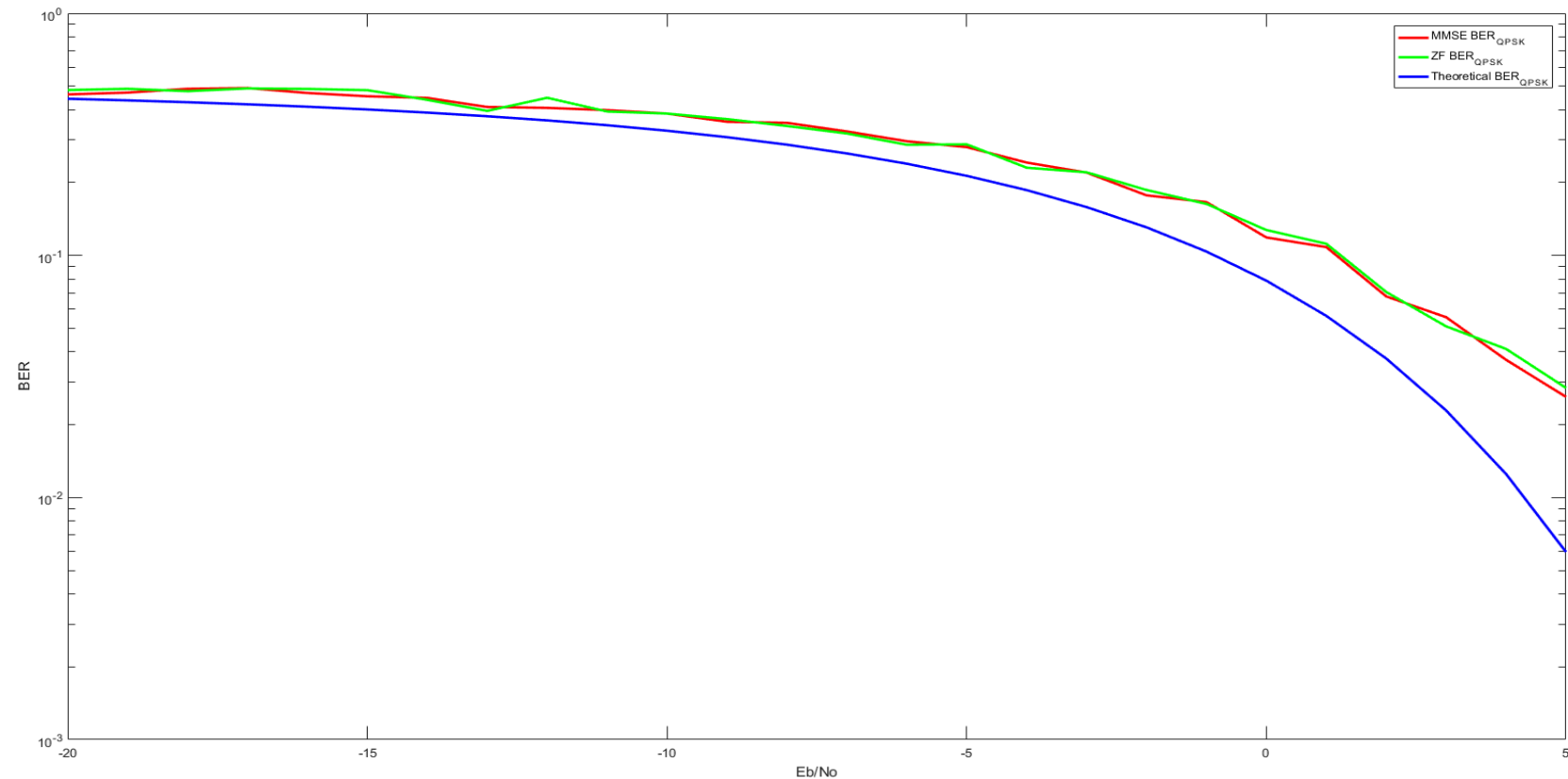
- RBs assigned to the UE: **100**
- Modulation: **64QAM**
- Number of bits: **86400 bits**
- Channel: **AWGN**
- SNR: **(-10 → 10) dB**
- **No** channel estimation.



MATLAB Simulation

3. SISO Tx & Rx full chain.

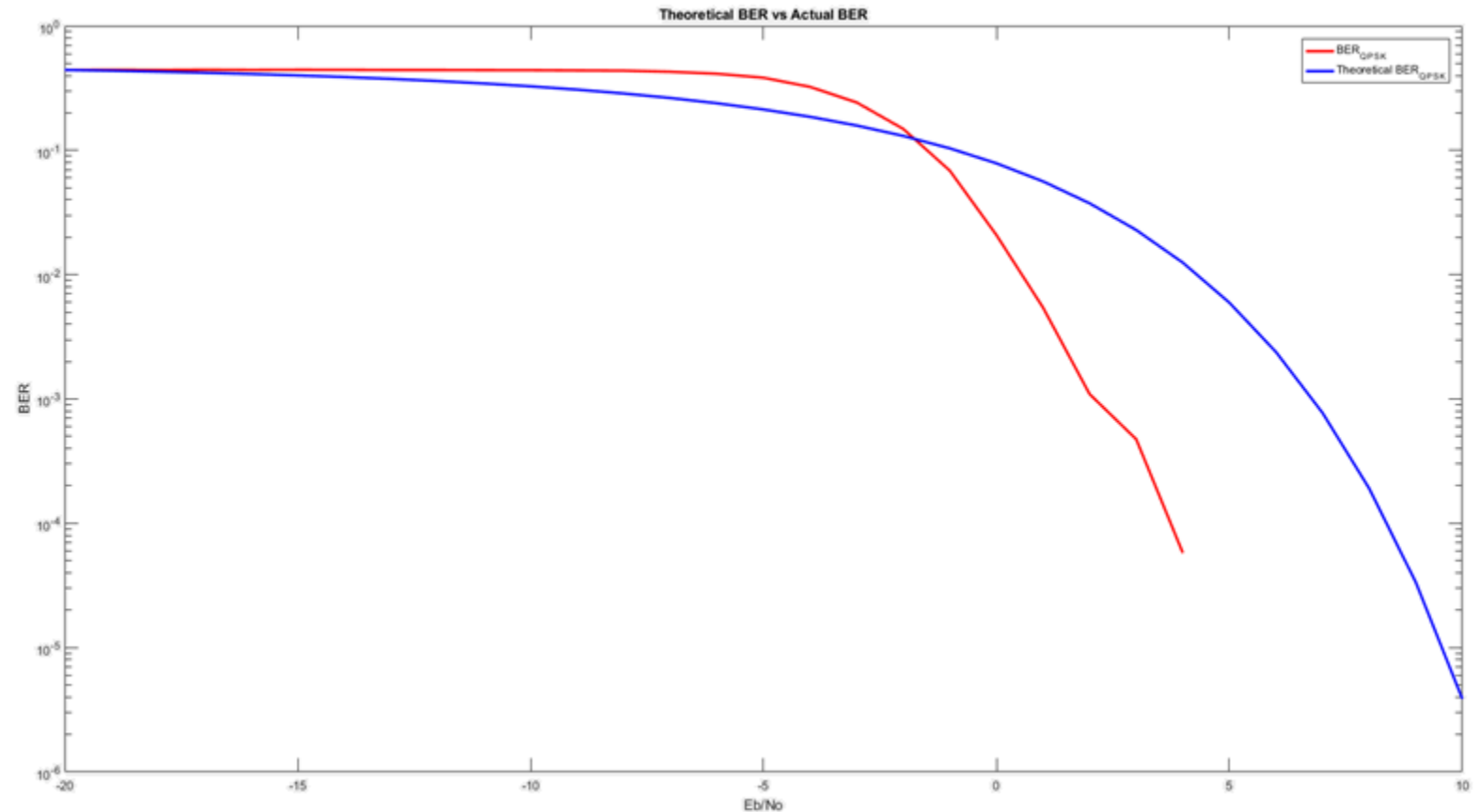
- RBs assigned to the UE: **6**
- Modulation: **QPSK**
- Number of bits: **1,728 bits**
- Channel: **Extended Pedestrian A channel**
- SNR: **($-20 \rightarrow 5$) dB**
- Channel estimation and equalization.



MATLAB Simulation

4. SISO Tx & Rx full chain.

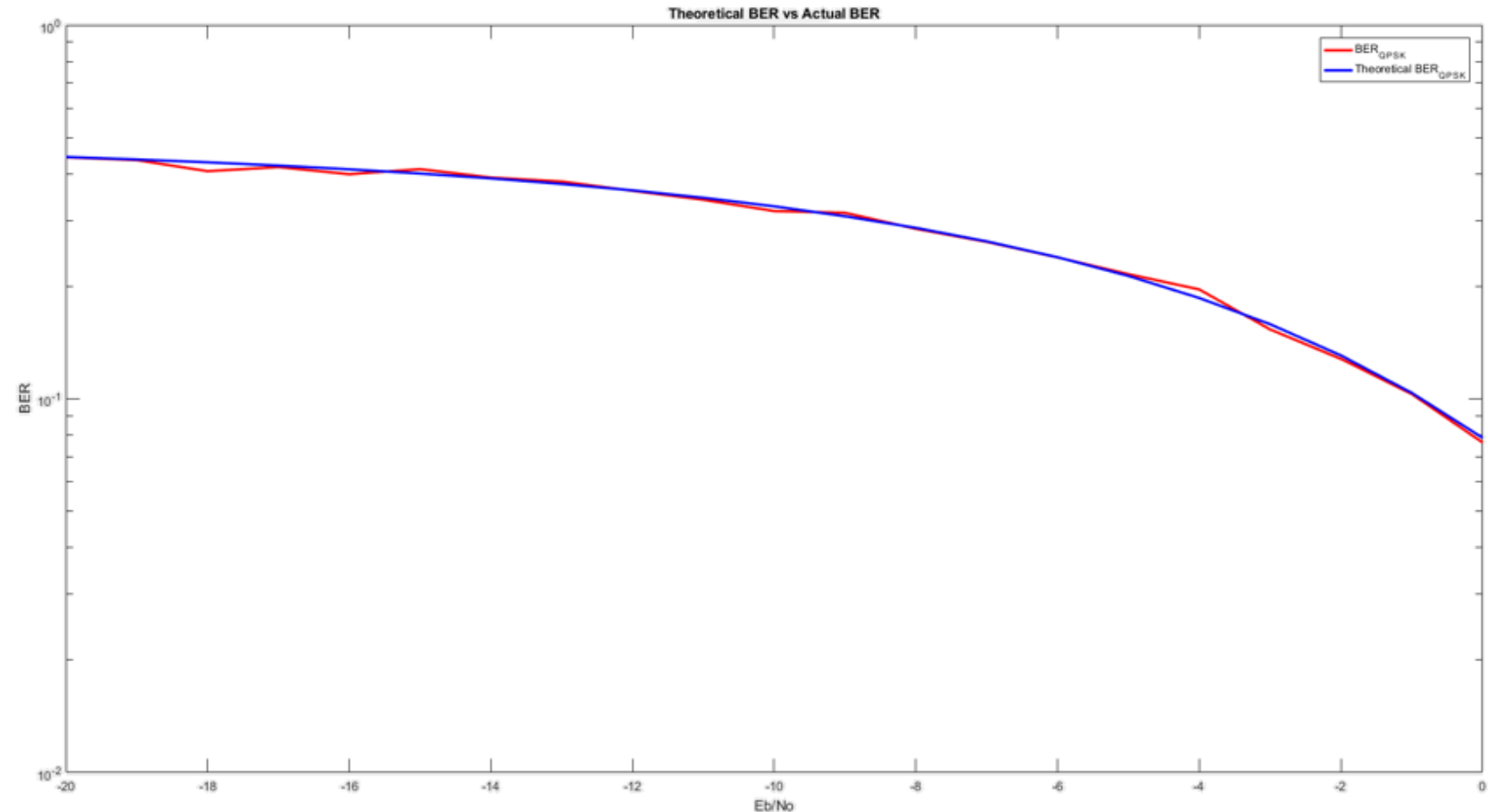
- Modulation: **QPSK**
- Number of bits: **500,000 bits**
- Channel: **AWGN**
- SNR: **$(-10 \rightarrow 10)$ dB**
- **Tail biting convolutional** channel encoding and **Viterbi** decoder.



MATLAB Simulation

5. MIMO Tx & Rx full chain.

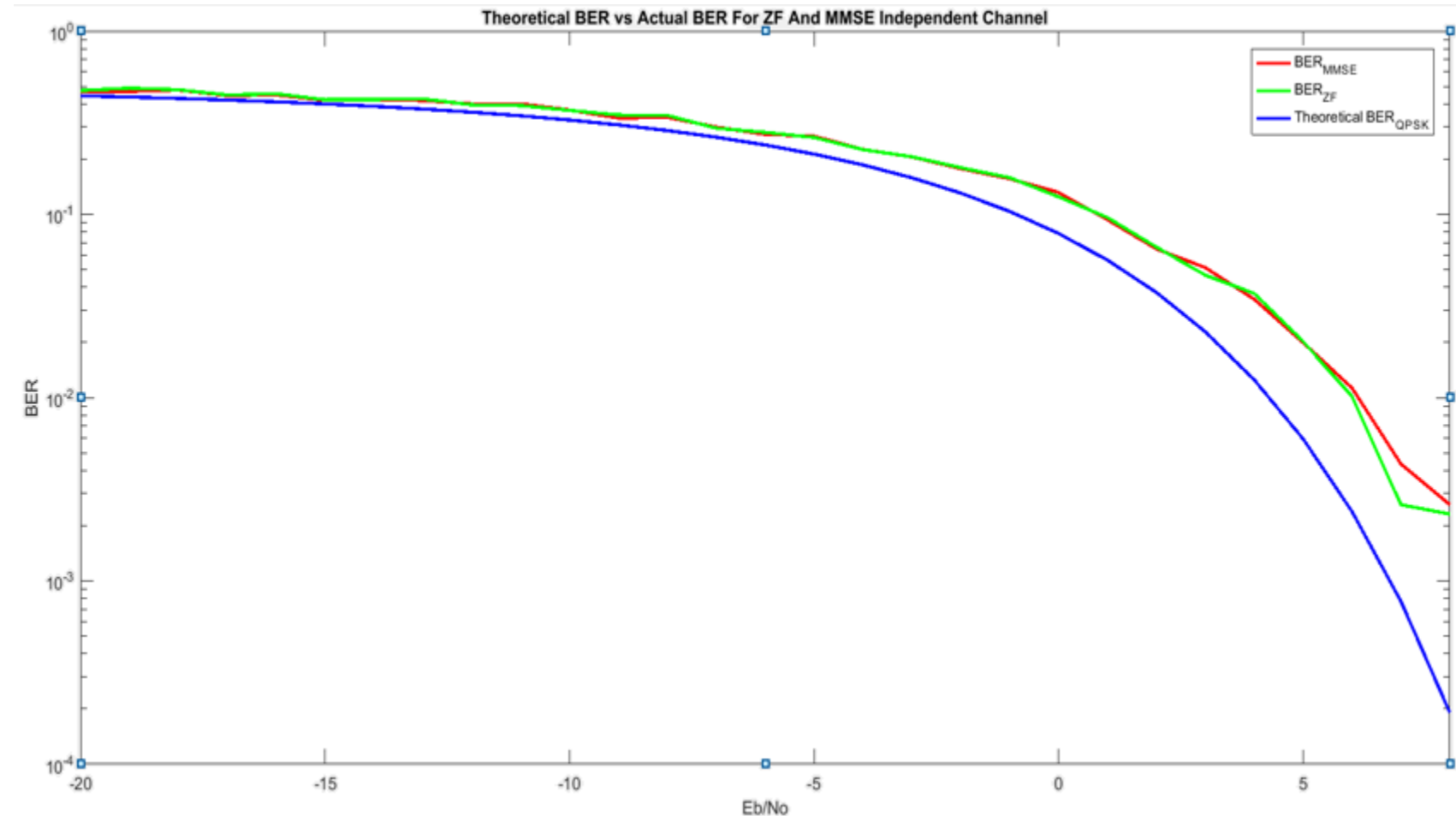
- RBs assigned to the UE: **6**
- Modulation: **QPSK**
- Number of bits: **1,728 bits**
- Channel: **AWGN**
- SNR: **$(-20 \rightarrow 0)$ dB**
- **No** channel estimation.



MATLAB Simulation

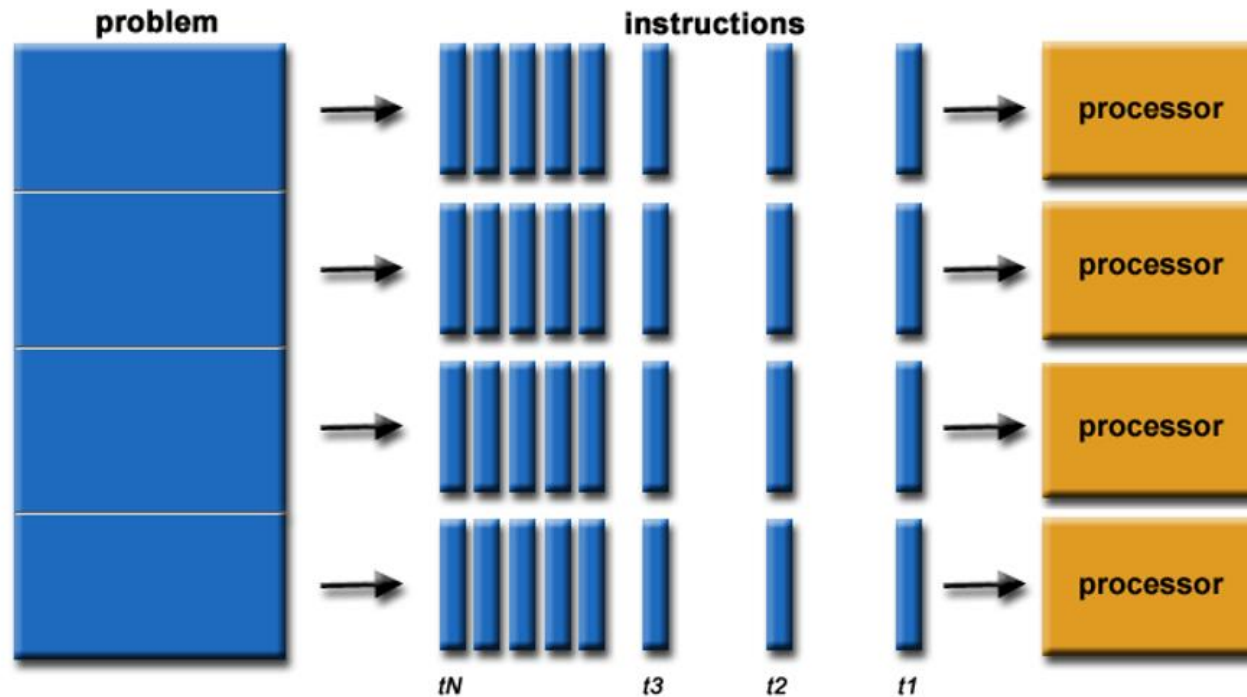
6. MIMO Tx & Rx full chain.

- RBs assigned to the UE: **6**
- Modulation: **QPSK**
- Number of bits: **1,728 bits**
- Channel: **Extended Pedestrian A model channel**
- SNR: $(-20 \rightarrow 8) \text{ dB}$
- Channel estimation and **ZF & MMSE** equalization.
- Without channel correlation.



Parallel Computing

A problem is broken into discrete parts that can be solved concurrently.



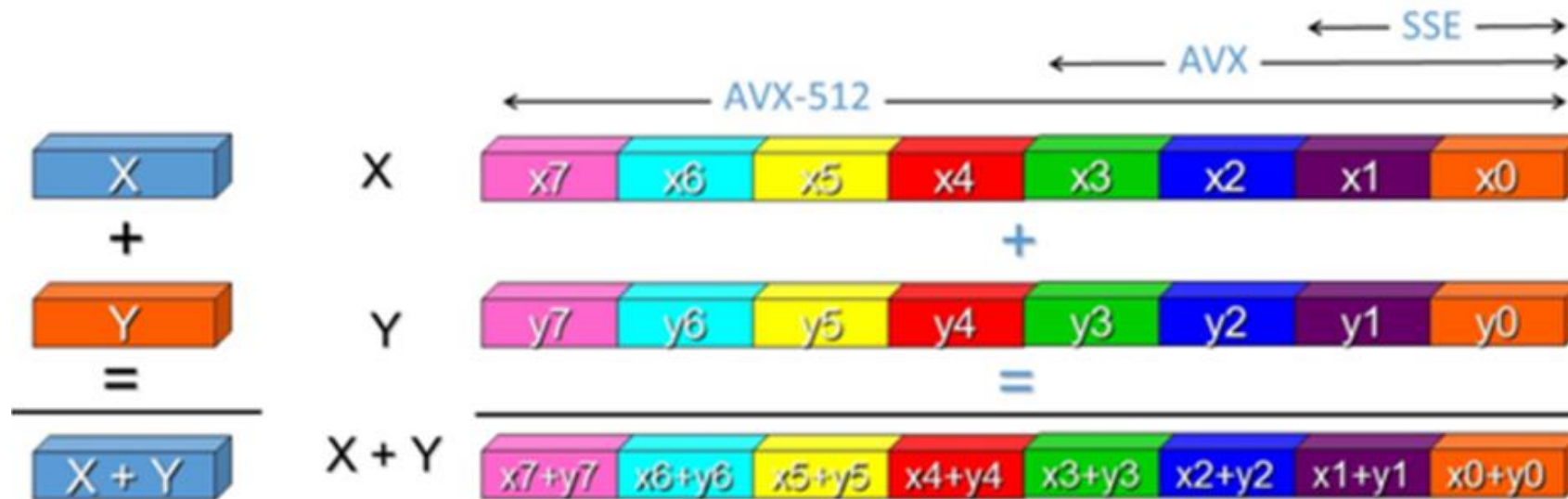
Parallel Computing

- Intel Parallel Techniques
- NVIDIA CUDA



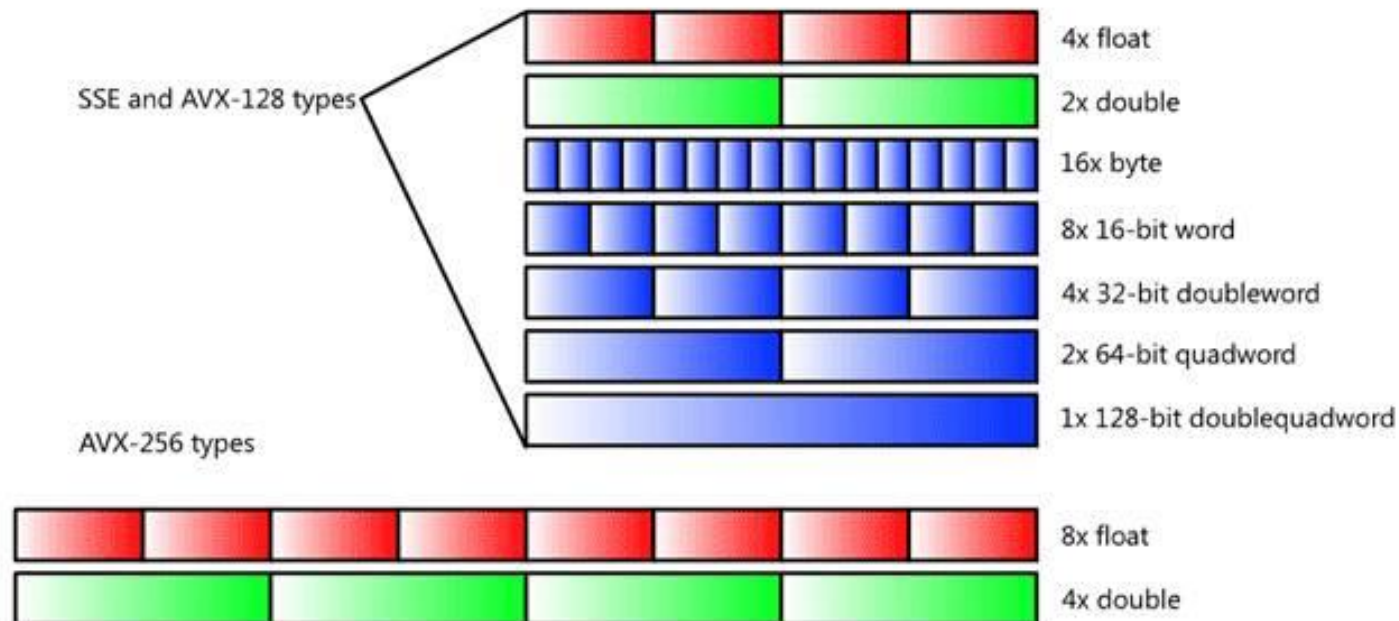
Intel AVX

- Set of instructions for doing SIMD operations on Intel CPUs.
- In SIMD mode, multiple data can be processed at the same time.



Intel AVX

- It uses 16 special registers in AVX and AVX2, 32 special registers in AVX512.
- Size of register is 128b, 256b in AVX and AVX2, 512b in AVX512.



Intel AVX



Intel Intrinsic functions:

- C style functions `__m256 _mm256_xor_ps (__m256 a, __m256 b)`
- Provide access to Intel SIMD instructions.
- No need to write assembly code.
- Supported only by Intel compiler.

Technologies

<input type="checkbox"/>	MMX
<input type="checkbox"/>	SSE
<input type="checkbox"/>	SSE2
<input type="checkbox"/>	SSE3
<input type="checkbox"/>	SSSE3
<input type="checkbox"/>	SSE4.1
<input type="checkbox"/>	SSE4.2
<input checked="" type="checkbox"/>	AVX
<input checked="" type="checkbox"/>	AVX2
<input type="checkbox"/>	FMA
<input type="checkbox"/>	AVX-512
<input type="checkbox"/>	KNC
<input type="checkbox"/>	SVML
<input type="checkbox"/>	Other

Intel AVX

AVX example for Scrambler:

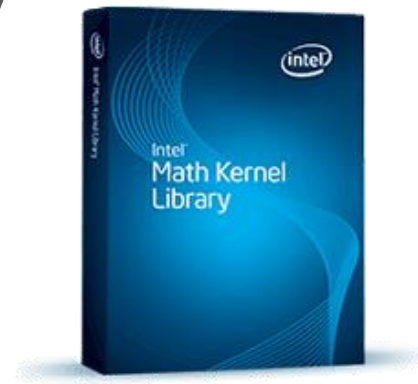
```
for (int i = 0; i < FRAME_LENGTH / 8; i++)
{
    // Loading bits into an AVX register
    __m256 vect_in1 = _mm256_setr_ps(bits[LEN * i], bits[LEN * i + 1], bits[LEN * i + 2],
                                     bits[LEN * i + 3], bits[LEN * i + 4], bits[LEN * i + 5],
                                     bits[LEN * i + 6], bits[LEN * i + 7]);

    // Loading pseudo random sequence generated into an AVX register
    __m256 vect_in2 = _mm256_setr_ps(c[LEN * i], c[LEN * i + 1], c[LEN * i + 2], c[LEN * i + 3],
                                     c[LEN * i + 4], c[LEN * i + 5], c[LEN * i + 6], c[LEN * i + 7]);

    // Perform AVX xor
    result[i] = _mm256_xor_ps(vect_in1, vect_in2);
}
```

Intel Math Kernel Library

- Intel MKL is a computing math library of highly optimized, extensively threaded routines for maximum performance.
- **optimization techniques used throughout the library include:**
 - Arrange operations to eliminate stalls due to arithmetic unit pipelines.
 - Use of hardware features as the SIMD, where appropriate.
- **The first time a function is called, a runtime check is performed to identify the hardware.**



Intel Math Kernel Library

Intel® MKL example for QPSK Mapper:

```
vsPackI(FRAME_LENGTH, &constant, 0, y);
```

```
cblas_saxpy(FRAME_LENGTH, SCALE, bits, 1, y, 1);
```

$$y = -\sqrt{2} * bits + \frac{1}{\sqrt{2}}$$

Other Intel Parallel Techniques

- Intel Cilk Plus is an extension to the C and C++ languages to support auto data and task parallelism.
- Data parallelism:
 - Array notations help the compiler to effectively vectorize the application.
- Task parallelism:
 - Cilk keywords (like `cilk_for`).
 - Divide loop body into threads.
 - Threads number depends on grain size.

ArrayNotation	Equivalent Scalar C/C+ Code
<code>A[:] = 5;</code>	<pre>for (i = 0; i < 10; i++) A[i] = 5;</pre>

NVIDIA CUDA

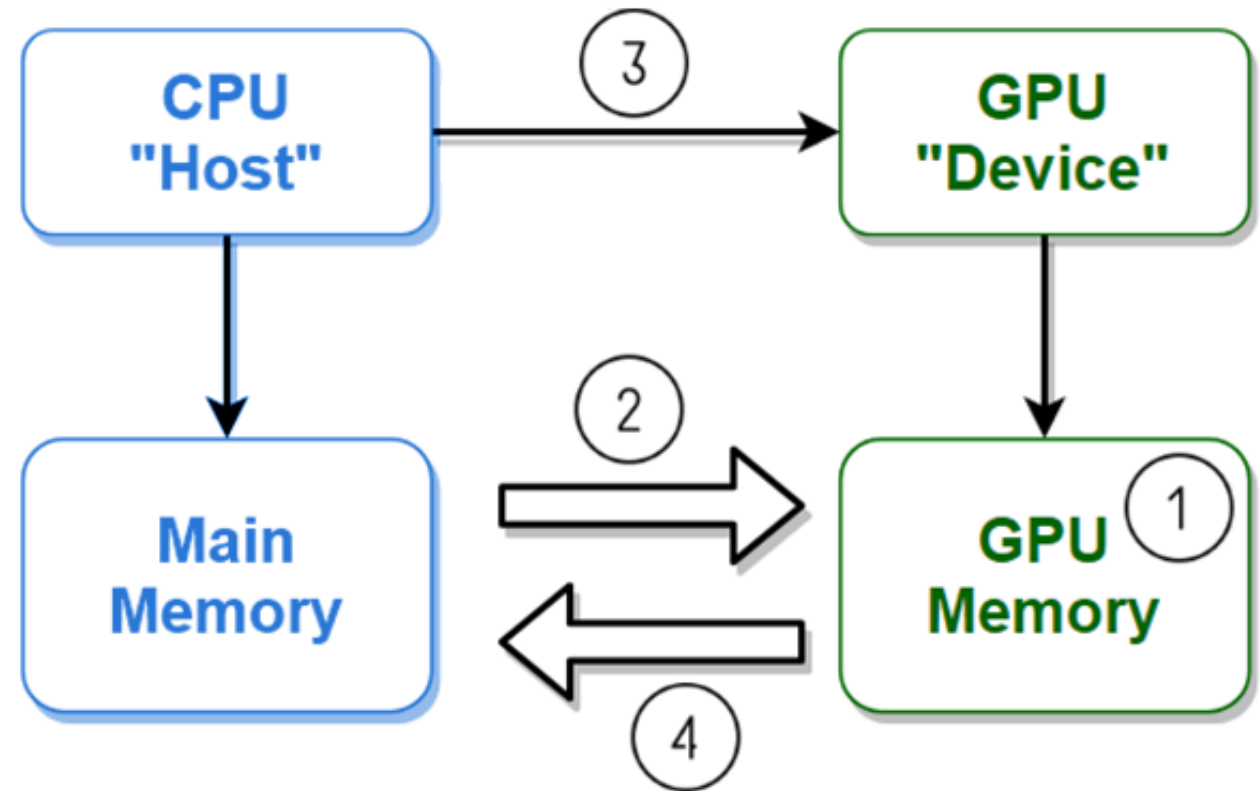
- It's a parallel computing platform invented by NVIDIA.
- It allows the programmer to take advantage of the massive parallel computing power of an NVIDIA GPU to do general purpose computation.

CUDA Programming Model

1. Allocate device memory.
2. Transfer data to the device.
3. Execute one or more kernels.
4. Transfer results back to the host.

- Target:

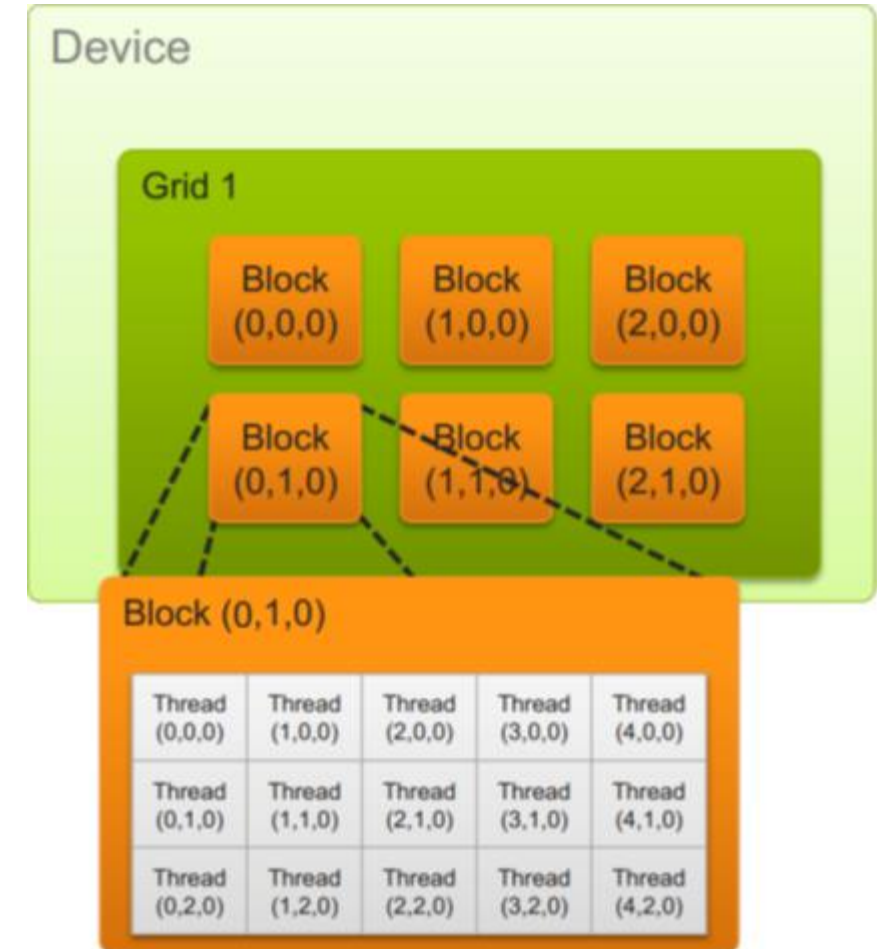
$$\frac{\text{Memcpy Time}}{\text{Kernels Time}} \downarrow \downarrow$$



CUDA Programming Model

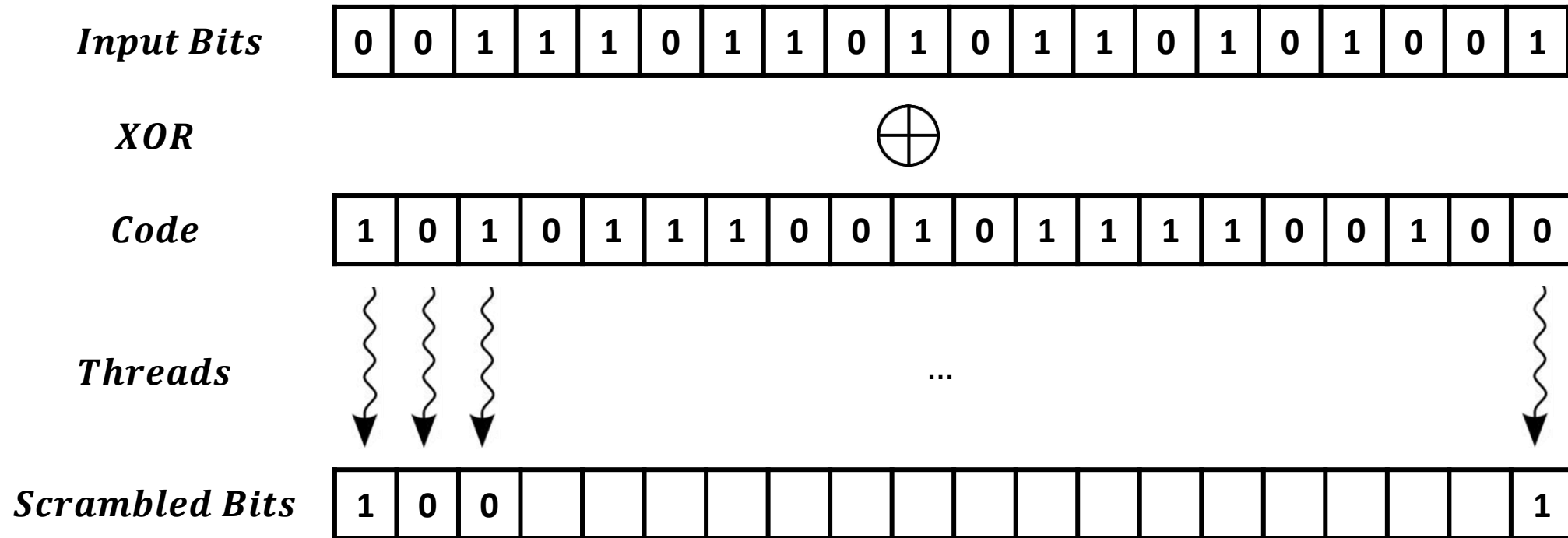
A kernel launches a grid of blocks of threads.

- Both Grids and Blocks can be 3D.
- Programmer should model the problem correctly.



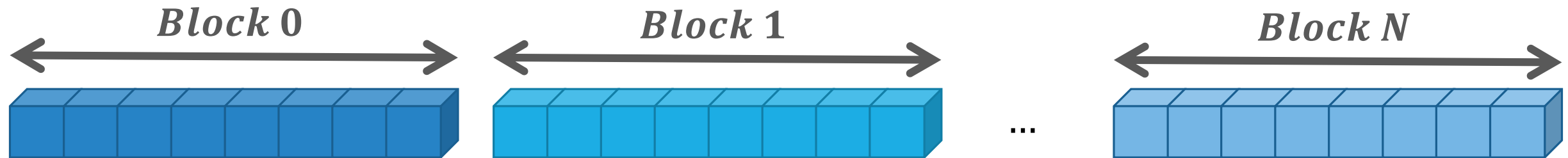
CUDA Example

Scrambler (one-to-one mapping)



CUDA Example

Scrambler can be modeled as 1D-Grid of 1D-Blocks.



CUDA Example

Host Code

```
void scrambler(Byte* bits_d, Byte** scrambledbits_d, Byte* c_d, const int N)
{
    //Calc. number of needed threads for calling kernel(s)
    int numThreads = N;
    int blockDim = (numThreads < 1024) ? numThreads : 1024; //block size in threads (max 1024 thread)
    int gridDim = numThreads / (blockDim)+(numThreads % blockDim == 0 ? 0 : 1);

    //Calling the kernel(s)
    scrabmler << <gridDim, blockDim >> > (bits_d, *scrambledbits_d, c_d, N);
}
```

CUDA Example

Device Code

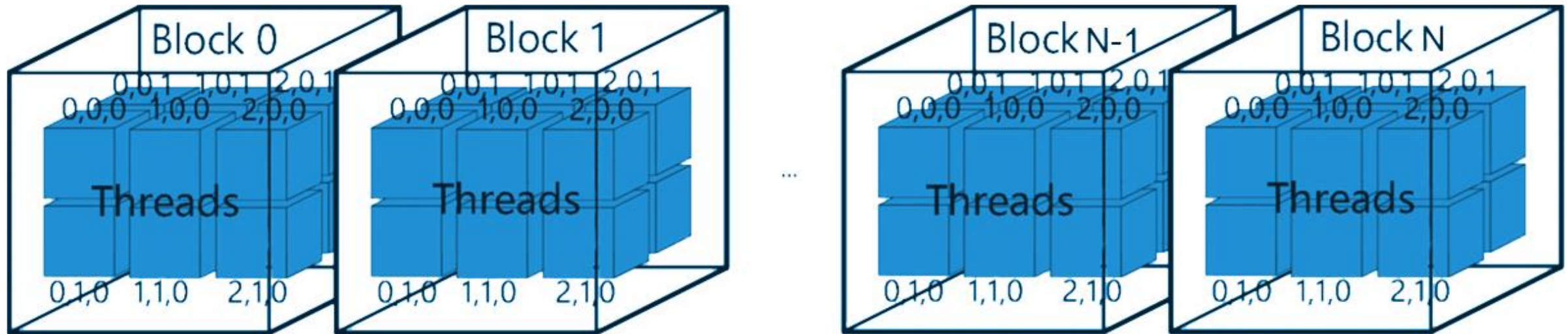
```
__global__ void scrabmler(Byte *bits_d, Byte *scrambledbits_d, Byte *c_d, int numThreads)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    //Not to run more threads than available data
    if (idx >= numThreads)
        return;

    scrambledbits_d[idx] = bits_d[idx] ^ c_d[idx];
}
```

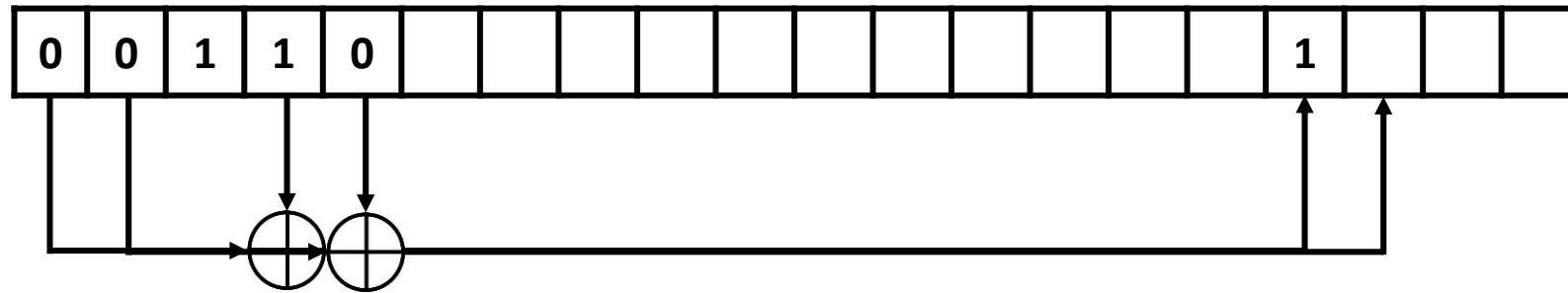
CUDA Programming Model

- Example: 1D Grid of 3D Blocks

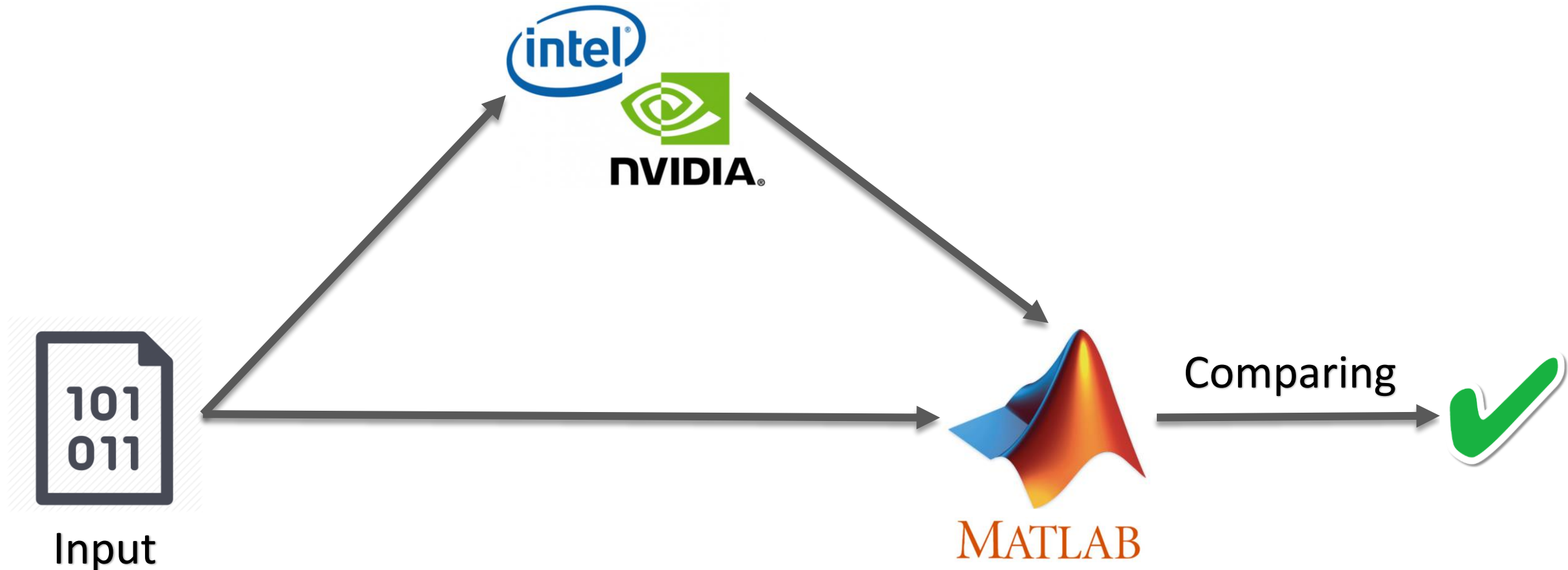


Pseudo-Random Sequence Generation

- By its nature, it's not **Parallelizable**.
- **Complex** parallel techniques takes **longer time** than serial implementation.



Output Verification



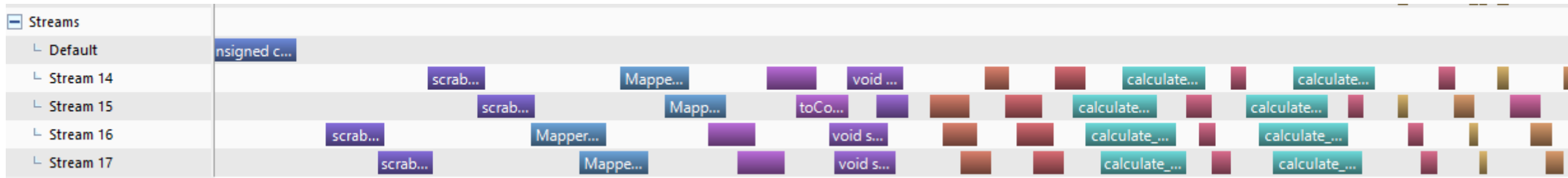
Error Calculations

$$\text{Relative Error} = \frac{\sum(\text{abs}(\text{MATLAB Symbols} - \text{CUDA Symbols}))}{\sum(\text{abs}(\text{MATLAB Symbols}))}$$

	Relative Error - Tx
SISO	$2.93 * 10^{-5}$
2x2 MIMO	$4.95 * 10^{-5}$
4x4 MIMO	$5.93 * 10^{-5}$
64x64 MIMO	$1.33 * 10^{-4}$

Parallel Streams

4x4 MIMO Tx Example:



Time Profiling

- “**Chrono.h**” Windows library timer is used.
- “**dsecnd**” function from “**MKL**” library is used for verification.
- “**GetTimeOfDay**” Linux function is also used.
- Timing results are the average of **1000** iterations.
- **Parallel Streams** are used in **Tx**.
- Used platform:
 - CPU: Intel® **Core™ i7-4720HQ** (6M Cache, up to 3.60 GHz)
 - GPU: NVIDIA GeForce **GTX 960M** (CUDA Cores: 640, Base Clock: 1096 MHz)

Time Profiling

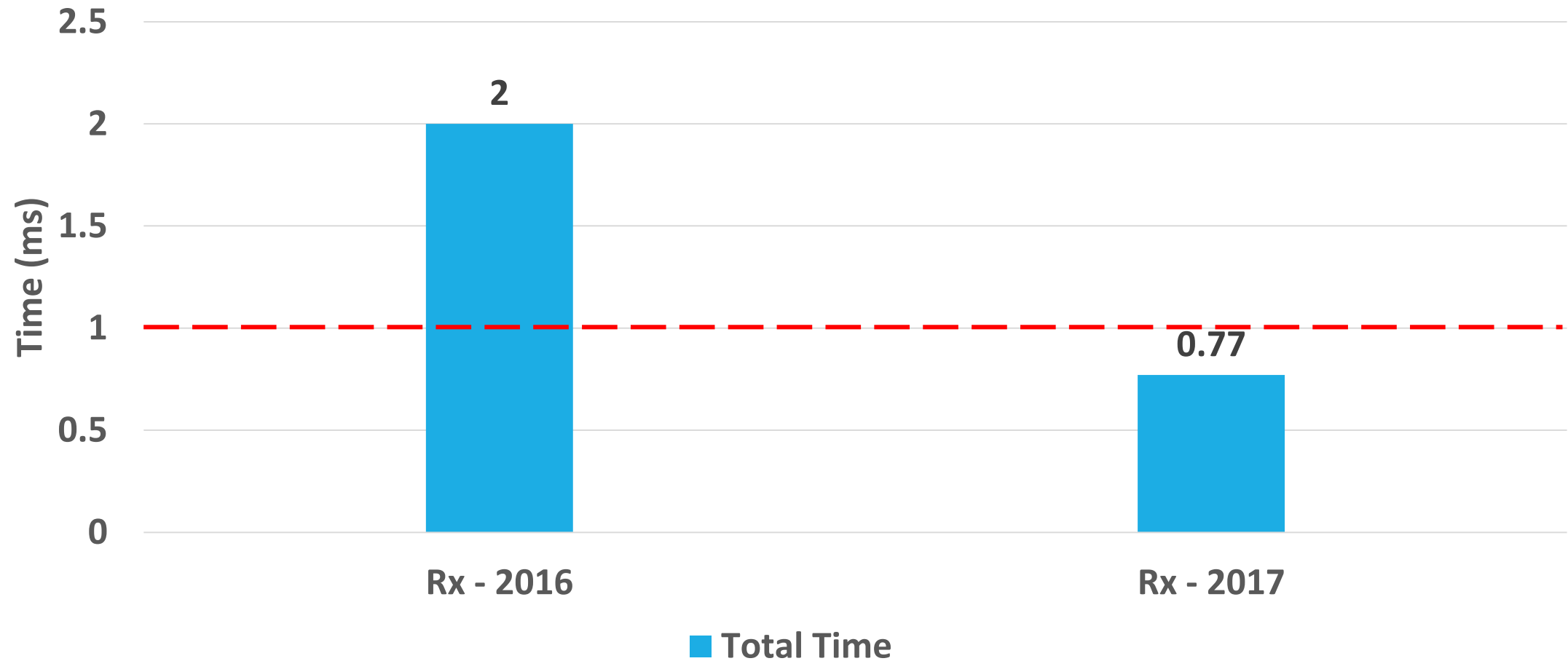
Configuration (Extreme Case)

- **Modulation: 64-QAM**
- **Resource Blocks: 100**

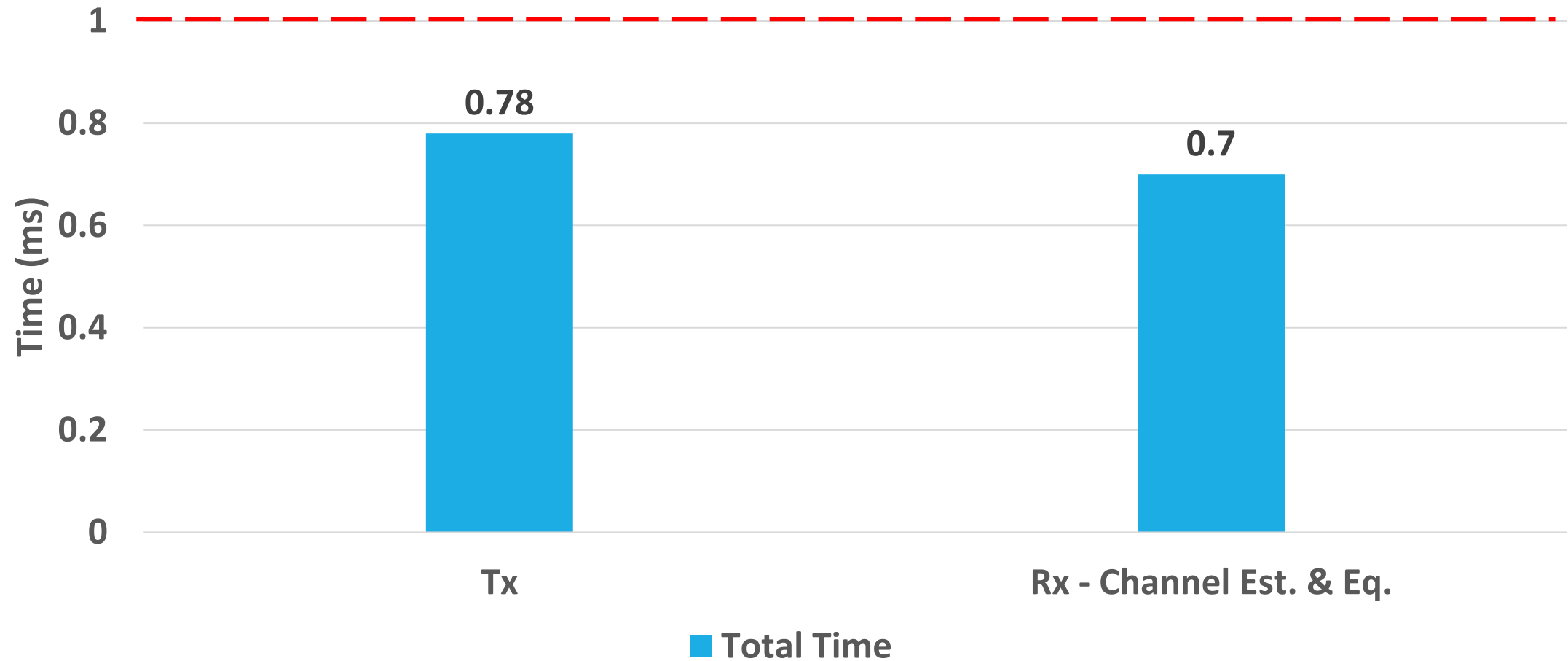
Previous Year's Configurations

- **Same machine** is used for time profiling.
- **DMRS** signal is read from a file (not generated).
- Modulation order is **QPSK** not **64-QAM**.

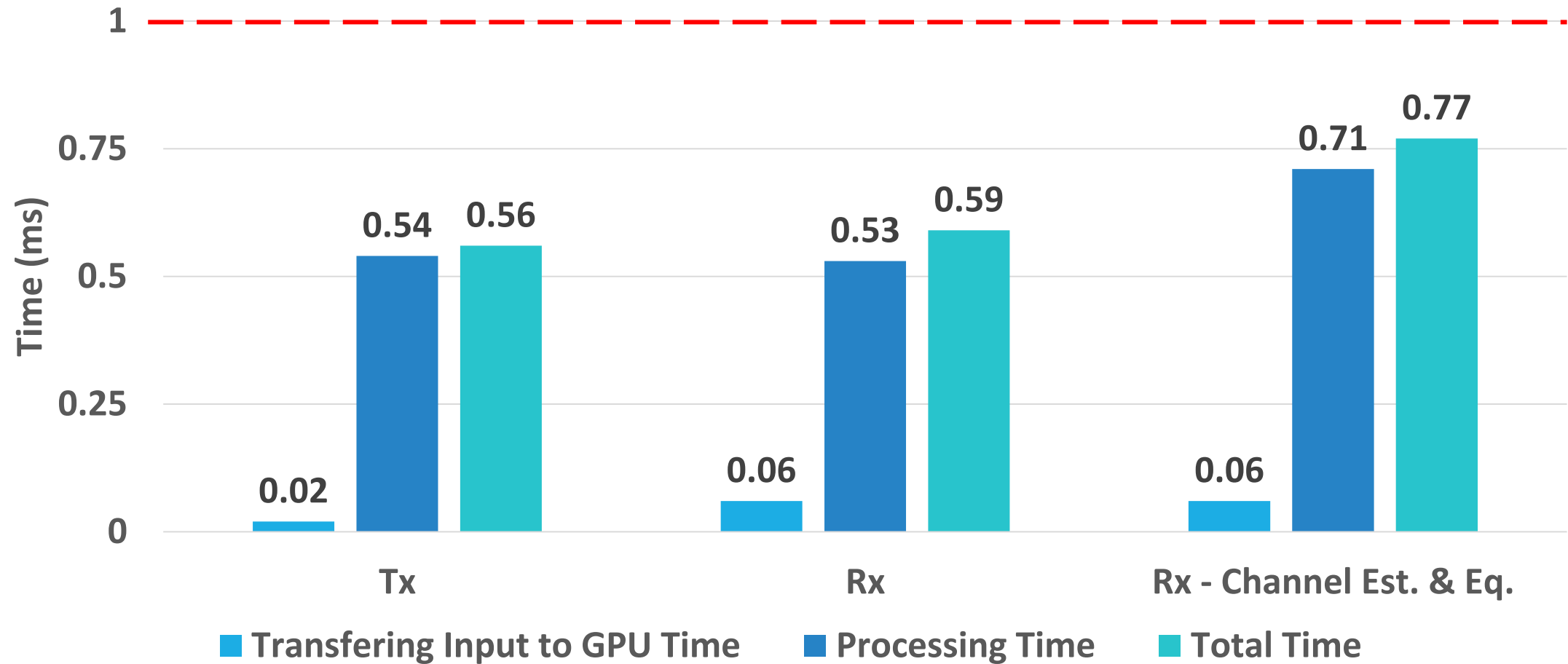
Comparison with Previous Year



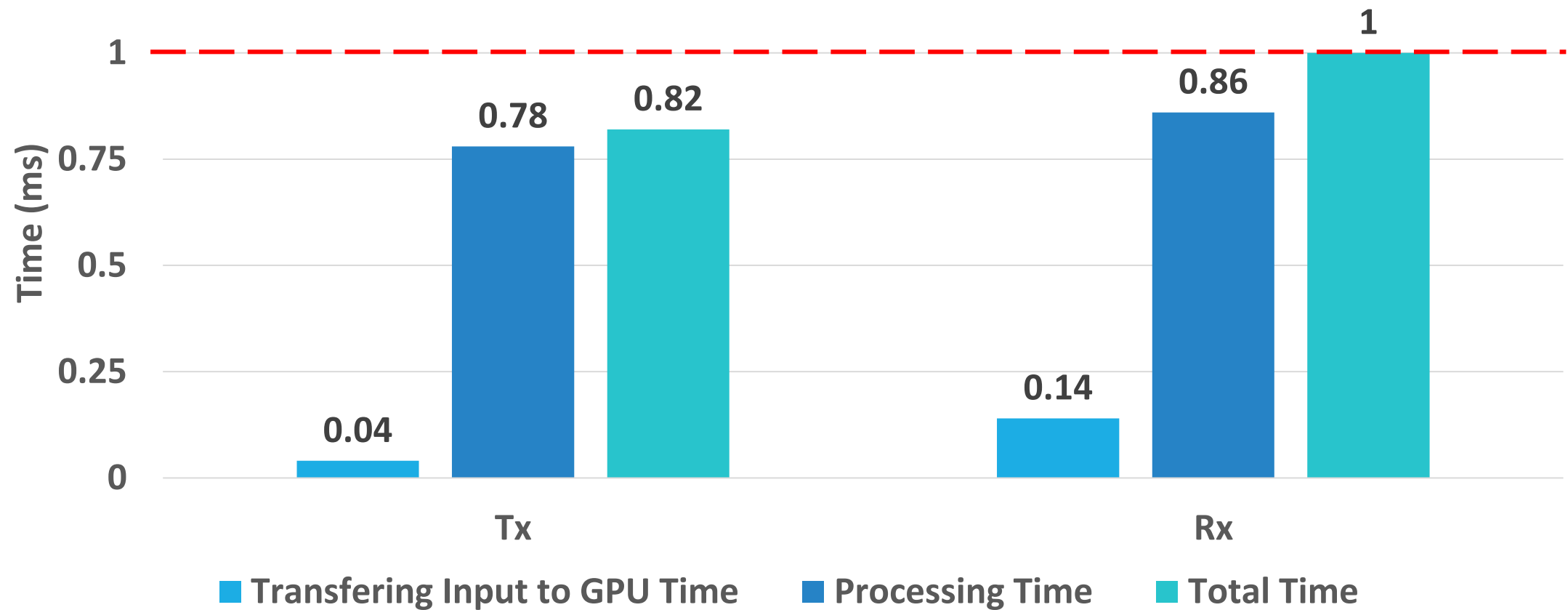
SISO Chain - Intel



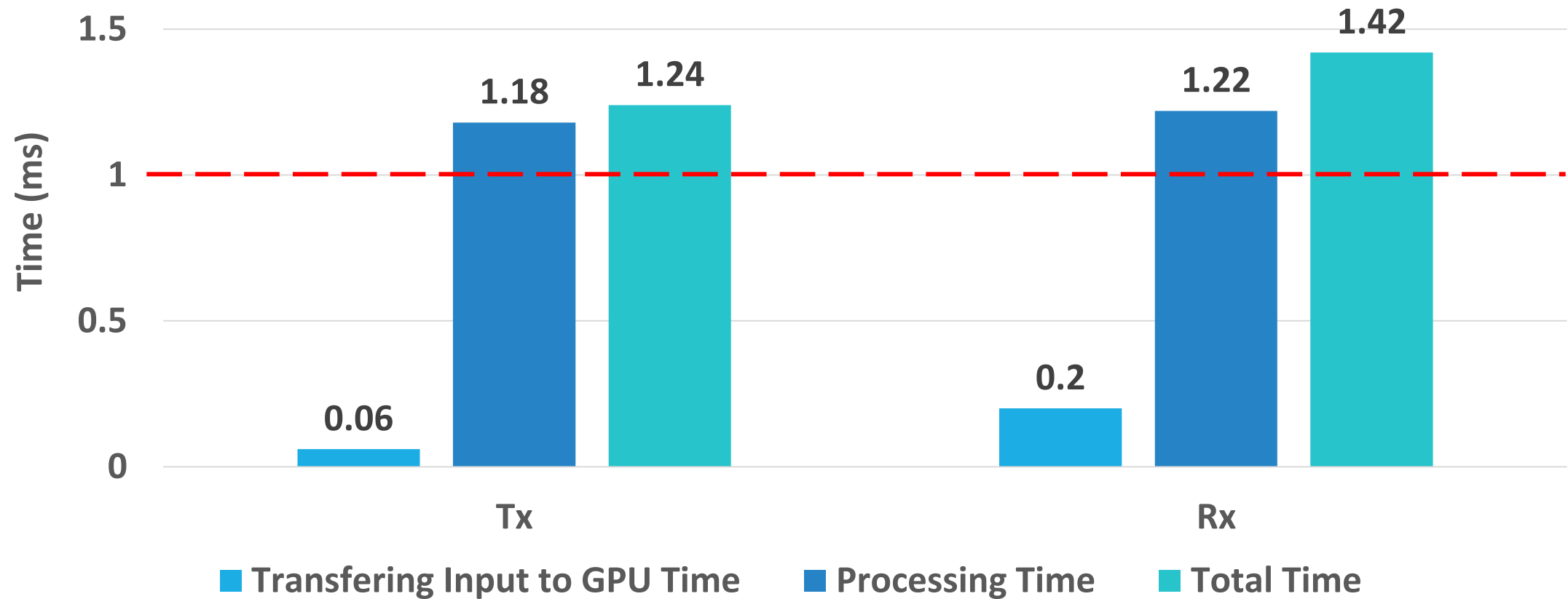
SISO Chain - CUDA



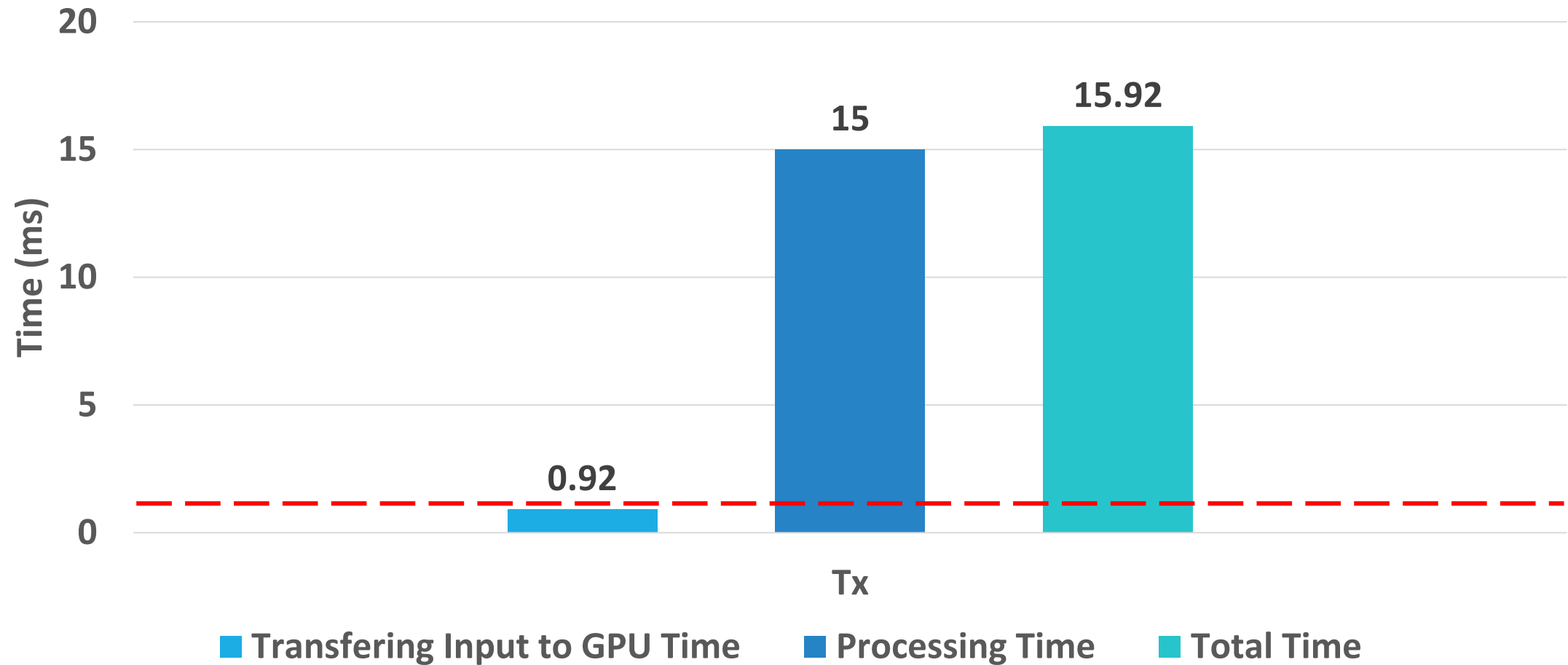
2x2 MIMO Chain - CUDA



4x4 MIMO Chain - CUDA



64x64 MIMO Chain - CUDA



Conclusion

	Tx (<i>in ms</i>)	Rx (<i>in ms</i>)
SISO – Intel	0.78	0.70
SISO – CUDA	0.56	0.77
2x2 MIMO	0.82	1.00
4x4 MIMO	1.24	1.42
64x64 MIMO	15.92	—



Streams

Future Work

- Trying **Kernel Merging**.
- Adding **parallel streams** to **MIMO Rx**.
- Implementing **parallel algorithm** for **Pseudo-Random Sequence Generation**.
- **Merging** between Intel **AVX** and **CUDA**.
- Using **High-End** Nvidia GPU.
- Trying Intel **AVX2** and **AVX512**.
- Trying **more complex channel estimation** and **equalization** techniques.

Thank
you!