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

By

Ahmed Mostafa Hosny

Ahmed Nour El-Deen Shahatah

Khaled Ahmed Ali

Mohamed Mostafa Nasr

Mohamed Osama Mohamed

Under supervision of

Dr. Ahmed Hesham

A Graduation Project Report Submitted to
the Faculty of Engineering at Cairo University
in Partial Fulfillment of the Requirements for the
Degree of
Bachelor of Science
in
Electronics and Communications Engineering

Faculty of Engineering, Cairo University

Giza, Egypt

July 2017

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

3GPP	3rd Generation Partnership Projects
3GPP2	3rd Generation Partnership Project 2
AMPS	Advance Mobile Phone System
AWGN	Additive white Gaussian noise
CB	Code Blocks
CDMA	Code Division Multiple Access
CM	Cubic Metric
CP	Cyclic Prefix
DFE	Decision feedback equalizer
DL	Downlink
DM-RS	Demodulation Reference Signals
EDGE	Enhanced Data Rates for GSM Evolution
EM	Electromagnetic
E-UTRA	Evolved UMTS Terrestrial Radio Access
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
EV-DO	Evolution-Data Optimized
FDD	Frequency Division Duplexing
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GPRS	General Packet Radio Service
GSM	Global system for Mobile communications
HSDPA	High Speed Downlink Packet Access
IFFT	Inverse Fast Fourier Transform
ISI	Intersymbol interference
ITU	International Telecommunication Union
LFSR	Linear-feedback shift register
LMMSE	Linear Mean Minimum Square Error
LOS	Line of sight
LS	Least square
LTE	Long Term Evolution

MIMO	Multiple Input Multiple Output
ML	Maximum Likelihood
MSE	Mean Square Error
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PHY	Physical Layer
PSK	Phase-Shift Keying
PUSCH	Physical Uplink Shared Channel
QOS	Quality of Service
RB	Resource Block
RE	Resource Element
SC-FDMA	Single Carrier Frequency Division Multiple Access
SIMD	Single Instruction Multiple Data
SIMO	Single Input Multiple Output
SNR	Signal-to-Noise Ratio
TACS	Total Access Communication System
TB	Transport Block
TDD	Time Division Duplexing
TDMA	Time Division Multiple Access
UE	User Equipment
UL	Uplink
UL-SCH	Uplink shared channel
UMB	Ultra-Mobile Broadband
UMTS	Universal Mobile Telecommunication System
W-CDMA	Wideband Code Division Multiple Access
N_{RB}^{DL}	Downlink bandwidth configuration, expressed in number of resource blocks
N_{RB}^{UL}	Uplink bandwidth configuration, expressed in number of resource blocks
N_{symb}^{PUSCH}	Number of SC-FDMA symbols carrying PUSCH in a subframe
$N_{symb}^{PUSCH-initial}$	Number of SC-FDMA symbols carrying PUSCH in the initial PUSCH transmission subframe

N_{symb}^{UL}	Number of SC-FDMA symbols in an uplink slot
N_{SRS}	Number of SC-FDMA symbols used for SRS transmission in a subframe (0 or 1).

Acknowledgment

Firstly, we would like to express our sincere gratitude to our supervisor Dr. Ahmed Hesham for the continuous support of our Graduation Project, for his patience, motivation, and immense knowledge. His guidance helped us a lot.

Besides our supervisor, we would like to thank Eng. Waleed Tag El-Din for his insightful suggestions, comments and encouragement.

Finally, we would like to thank Axxcelera Egypt, for the project idea, training sessions, and the opportunity to join such a great experience. Especially Eng. Mahmoud Abubakr.

Abstract

Nowadays wireless communication has influenced the world in many important ways. It had a dramatic impact on how companies conduct business, making it easier to keep in touch with customers around the world. Our project thesis is about the fourth generation of mobile communications also known as Long Term Evolution (LTE), especially on the physical uplink shared channel (PUSCH). The target of our project is to move the processing of the PUSCH from DSP to a general purpose CPU/GPU. The PUSCH transmitter and receiver are implemented using two different platforms: CUDA and Intel AVX with Intel math kernel library (MKL). First, moving the implementation from DSPs to general-purpose processor is discussed in the first chapter. Second, the used platforms are presented in chapters 2 and 3. Then, LTE introduction, PUSCH blocks and communication channel are presented in chapters 4 and 5. Finally, the BER plots and timing results are presented.

Chapter 1: Introduction

1.1 Problem Overview

Communications and DSP applications are known to be computational intensive. In communication systems we have to perform complex operations on the transmitted/received data. These complex operations increased in LTE PHY layer due to the use of FFT (Fast Fourier Transform) so a fast and powerful hardware is needed.

Due to the high computational requirements of the LTE eNodeB/UE, many specific platforms that are normally combinations of DSPs, general purpose processors and ASICS are available. The problem of using these platforms is that the platform may be totally unusable for subsequent LTE releases or for upgrading to a newer technology. This upgradability problem will be translated into cost as the platform needs to be changed.

1.2 Problem Solution

Implementing the LTE Physical layer on a general purpose processor will eliminate the need for a specific platform. The general purpose CPU/GPU will be reusable and expandable. Also the support for general purpose CPU development can be obtained easier than that of specific platforms.

But it would be slower than the specific platform (i.e. DSP) in doing the required operations, so we can use parallelism techniques to reduce the execution time like:

- AVX instruction (Supports Intel Core-i Processors).
- Intel Cilk Plus operations.
- Intel MKL (Math Kernel Library).
- Parallel Programing for GPU (CUDA – a parallel computing platform and programming model invented by NVIDIA).

Both techniques are discussed later in chapter 2 and chapter 3 respectively. Communication principles are introduced in chapters 4 and 5. Problem solution and results are discussed in chapter 6 in addition to the resources used like the HW platform and SW framework.

Chapter 2: Processors

2.1 General Purpose Processor

General purpose means that it can be used for many different tasks and not designed for a special purpose. A microprocessor can be programmed by the user. It is necessary for the user to know about the internal resources and features of the microprocessor. The programmers must also understand the instructions that a microprocessor can support. Every microprocessor will have its own associated set of instructions that it supports and this list is given by all the microprocessor manufacturers.

A microprocessor can either have a CISC, or complex instruction set computer, architecture or a RISC, or reduced instruction set computer, architecture. The CISC architecture is more complex and can perform complex commands. The RISC architecture is simpler, smaller and faster.

2.2 GPP Architecture

All approaches for designing microprocessor have in common the goal of exploiting parallelism hidden within programs. A program consists of a long sequence of instructions. The microprocessor maintains the illusion of executing one instruction at a time, but under the hood it attempts to overlap the execution of hundreds of instructions at a time. Overlapping instructions is challenging due to interactions among them (data and control dependencies). Another issue is the need for a speculative processor for overcoming lower throughput from branching instructions.

General purpose processors include branch prediction and speculation hardware for expanding the parallelism scope of the microprocessor to hundreds or thousands of instructions. They include also dynamic scheduling for extracting instructions that may execute in parallel and overlapping their execution with long-latency memory accesses. Furthermore, they include caching and prefetching to collapse the latency of memory accesses, and value prediction and speculation for parallelizing the execution of data-dependent instructions, to mention a few. [1]

Microprocessor architecture has evolved from single core processor to multi-core processors. A multi-core processor is a single computing component with two or more independent processing units (called "cores"). Processors with two or more cores are faster because they can process multiple pieces of information simultaneously.

The basic components of microprocessor architecture include:

- ALU: The Arithmetic Logic Unit performs all arithmetic and logic operations.
- Accumulator: Holds the results of operations performed by the ALU as a new operand to the ALU.
- PC (program counter): Holds the memory address of the next instruction to be executed.
- Status, data and address registers: The status register stores information about the result of a previous ALU operation, the data register stores data going to or coming from an I/O port or memory, and the address register stores the address of the memory location to be accessed.
- Control unit: Holds the circuitry that controls the process of executing, decoding and fetching program instructions.

2.3 Intel Processors

Intel® is the world’s oldest and most established microprocessor company, producing the world’s most popular microprocessor chips. Although perhaps best known for its PC processors, Intel devices are used in many field of electronics including automotive, robotics, consumer electronics, image processing, networking, encryption, military, and other industries.

Since the first tiny Intel 4004 microprocessor chip was made in 1971, Intel has produced an unbroken series of upgrades and improvements to the world’s best known microprocessor family. From its early 8-bit beginnings, the Intel architecture now encompasses a range of 32-bit and 64-bit microprocessors that address a range of applications, performance requirements, power levels, and price points.

The cornerstone of Intel architecture’s popularity is its compatibility. Each new generation of Intel architecture microprocessor is a superset of its predecessors, providing backward compatibility with older chips and older software, while also adding new or enhanced features. This compatibility allows engineers, programmers, and development teams to reuse the software and software-development tools from earlier projects, protecting their investment in time and talent.

Intel architecture chips have obviously undergone many changes over the past 40+ years. Early chips were given technical part numbers, such as 8086, 80386, or 80486. This led to the commonly used shorthand of “x86 architecture,” in reference to the last two digits of each chip’s part number. Beginning in 1993, the “x86” naming convention gave way to more memorable (and pronounceable) product names such as Intel® Pentium® processor, Intel® Celeron® processor, Intel® Core™ processor, and Intel® Atom™ processor.

Although every branch of the broad Intel architecture (or x86) family tree retains the same basic features and functionality as the earlier chips, each new generation also adds its own unique features. For example, Intel Pentium processor added multimedia extensions (called MMX™ technology) that accelerated audio and video processing. Extended temperature Intel Pentium processor with MMX technology is with more streaming-media capabilities known as Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Streaming SIMD Extensions 2 (Intel® SSE2). Floating-point units (FPUs) went from optional upgrade to standard feature of Intel architecture processors, and today encryption/decryption extensions, power-management features, and multilevel caches are now found on most Intel architecture processors. Data paths have widened from 8 bits to 32 bits, 64 bits, and even 128 bits and more. Operating frequencies have jumped from a few megahertz to 3 GHz (three billion cycles per second) and beyond. [2]

2.4 Single Instruction Multiple Data Instructions

SIMD represents a class of computation where multiple processing engines perform the same operation on multiple data elements simultaneously. SIMD Extensions Support is another example of the commodity general-purpose CPU adding specialized hardware support for domain-specific applications. Like floating-point operations, SIMD operations used to be the domain of highly specialized vector supercomputers (e.g., Control Data STAR 100, Cray 1 Computer System, and the Connection Machine® Model CM-1 from Thinking Machines Corporation).

Over time, the CPU increased in performance, thanks to Moore's law and Dennard scaling. In 1997, SIMD processing was introduced to commodity CPUs with the inclusion of MMX™ technology extensions to a model of the Pentium® processor. The MMX extensions enabled the calculation of 8 bytes or 4 words in parallel using 64-bit registers that supported vector integer operations. These enabled improvements in multimedia and communications workloads, in addition to improving graphics realism and full-screen, full-motion video.

As CPUs became more powerful, software running on them increased in complexity. The Streaming SIMD Extensions (SSE) in 1999 expanded the SIMD capabilities in the architecture to include 128-bit wide architectural registers and packed single precision calculations. To take advantage of this new architectural capability, implementations introduced dedicated hardware to support SSE such that performance gains could be capitalized on. These extensions also addressed some of the MMX limitations by supporting the floating point data type. This enabled mixed integer and floating-point SIMD. In 2001, the SSE2 extensions added support for 8-bit, 16-bit, and 32-bit integer vectors in addition to double-precision data types. These extensions also provided programmers greater control to access and cache data. Since streaming data types do not always have cache locality, the extensions also provided software direct control over cacheability to minimize cache pollution, as well as support for software directed prefetching of data. With the increasing sophistication of the SSE extensions, they supplanted x87 for most floating-point operations (except 80-bit extended precision) and MMX for media applications.

The instruction set architectures remain fluid to match the needs of applications, so it is not a typical for every CPU generation to add new instruction capabilities. The SSE3 extensions in 2004 provided additional instructions for format conversion, and supported unaligned address loads while the SSSE3 extensions in 2006 accelerated operations on packed integers. The SSE4.1 and SSE4.2 extensions in 2007 and 2008 provided further enhancements to improve compiler vectorization, support for packed double word computation and for string and text processing. [3]

In 2010, the Intel® Advanced Vector Extensions (AVX) widened the vector registers to 256 bits and introduced the first set of instructions to utilize these registers, focused on floating point. In 2012, the Intel AVX2 extensions widened the integer data type to 256 bits. More details about Intel AVX will be provided in the next sub-sections.

2.4.1 Intel® Advanced Vector Extensions

Intel Advanced Vector Extensions (Intel AVX) is a set of instructions for doing Single Instruction Multiple Data (SIMD) operations on Intel architecture CPUs. These instructions extend previous SIMD offerings (MMX and Intel SSE) by adding the following new features:

- The 128-bit SIMD registers have been expanded to 256 bits. Intel AVX is designed to support 512 as in AVX 512 that was already launched in 2016
- Three-operand, non-destructive operations have been added. Previous two-SIMD extensions performed operations such as A = A + B, which overwrites a source operand; AVX can perform operations like A = B + C, leaving the original source operands unchanged.
- A few instructions take four-register operands, allowing smaller and faster code by removing unnecessary instructions.
- Gather support, enabling vector elements to be loaded from non-contiguous memory locations (Supported by AVX2).
- Three-operand Fused Multiply Add operations (FMA3) support as performing this operation (A = A * B + C) in a single instruction (Supported by AVX2).
- Memory alignment requirements for operands are relaxed.

The AVX instruction set has performance higher than the legacy SSE, and conventional non-vectored (serial) codes, the following result is for running Mandelbrot set example codes (a Mandelbrot set is a computationally intensive operation on complex numbers) which were implemented using AVX, SSE, and conventional float and complex data types.

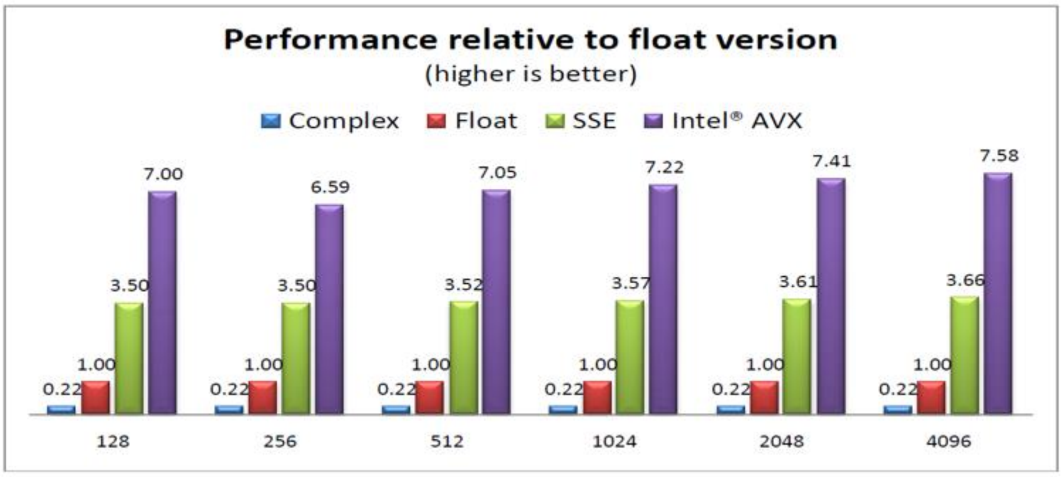


Figure 2.1: Performance comparison between Serial, AVX and SSE.

2.4.2 Programming with Intel® AVX

There are two ways of programming using AVX:

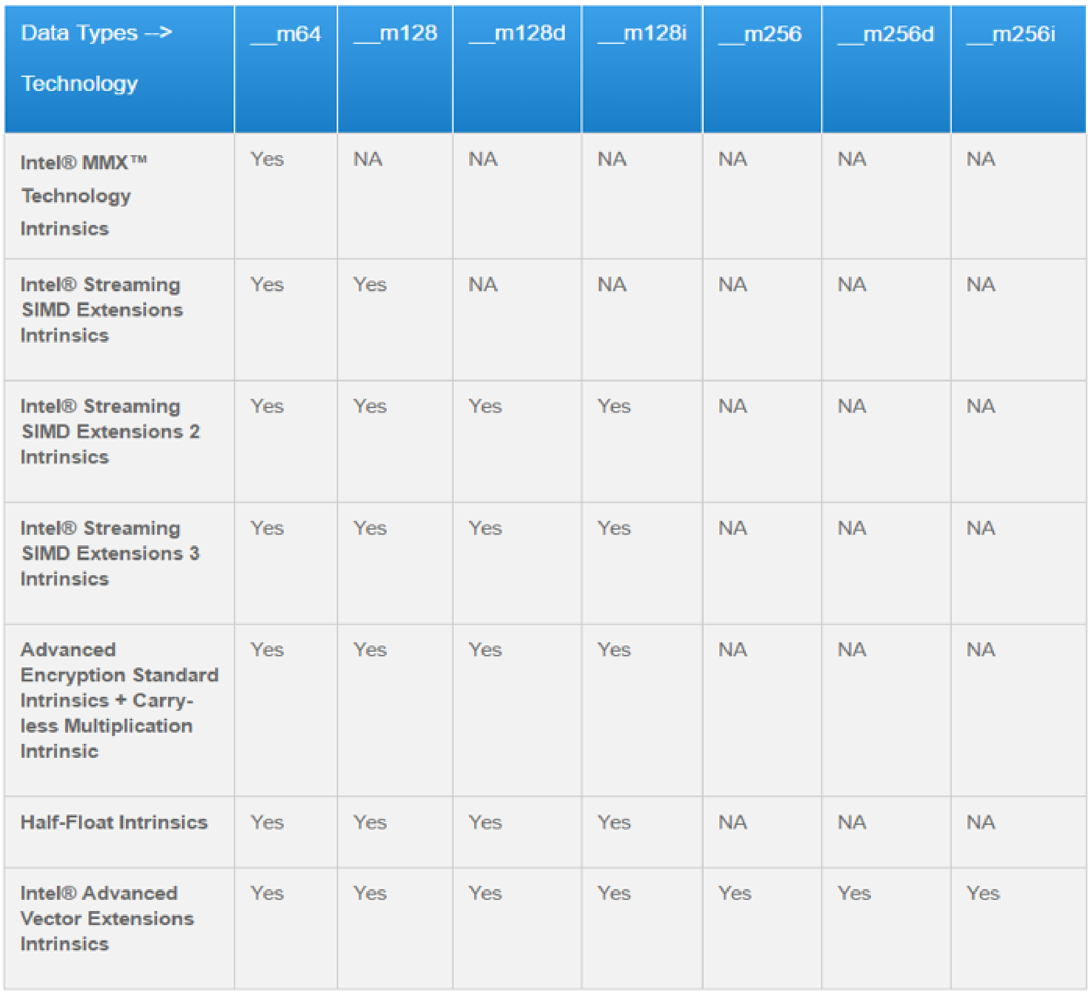
- Using the Assembly instructions.
- Using intrinsic Functions provided by Intel compiler.

The easier way is to use the 2nd method or the intrinsic functions. Intrinsic functions are assembly-coded. They allow the programmer to use C function calls and variables in place of assembly instructions. [4]

2.4.3 Intrinsic Data Types

Intrinsic functions use new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

The following figure shows the new data types and the instructions supporting each data type. A 'Yes' indicates that the data type is available for that group of intrinsic functions; an 'NA' indicates that the data type is not available for that group of intrinsic functions.



Data Types –> Technology	__m64	__m128	__m128d	__m128i	__m256	__m256d	__m256i
Intel® MMX™ Technology Intrinsics	Yes	NA	NA	NA	NA	NA	NA
Intel® Streaming SIMD Extensions Intrinsics	Yes	Yes	NA	NA	NA	NA	NA
Intel® Streaming SIMD Extensions 2 Intrinsics	Yes	Yes	Yes	Yes	NA	NA	NA
Intel® Streaming SIMD Extensions 3 Intrinsics	Yes	Yes	Yes	Yes	NA	NA	NA
Advanced Encryption Standard Intrinsics + Carry-less Multiplication Intrinsic	Yes	Yes	Yes	Yes	NA	NA	NA
Half-Float Intrinsics	Yes	Yes	Yes	Yes	NA	NA	NA
Intel® Advanced Vector Extensions Intrinsics	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figure 2.2: Intrinsic Data types supported in each SIMD extension.

2.5 Intel® Math Kernel Library:

The Intel® Math Kernel Library (Intel® MKL) provides a comprehensive set of math functions that are optimized and threaded to exploit all the features of the latest Intel® processors. Intel MKL improves performance with math routines for software applications that solve large computational problems as Intel MKL linear algebra routines, fast Fourier transforms, vectored math functions, random number generation functions, Matrix-Matrix operations and other functionality.

Intel MKL is built using the Intel® C++ and Fortran Compilers and threaded using OpenMP. Its algorithms are constructed to balance data and tasks for efficient use of multiple cores and processors.

2.5.1 Performance Enhancements:

The Intel® Math Kernel Library has been optimized by exploiting both processor and system features and capabilities as those routines that most profit from cache-management techniques.

The major optimization techniques used throughout the library include:

- Loop unrolling to minimize loop management costs
- Copying to reduce chances of data eviction from cache
- Data prefetching to help hide memory latency
- Multiple simultaneous operations to eliminate stalls due to arithmetic unit pipelines
- Use of hardware features such as the SIMD arithmetic units, where appropriate

To achieve all the above, the first time a function from the library is called, a runtime check is performed to identify the hardware on which the program is running. Based on this check, a code path is chosen to maximize use of instruction- and-register level SIMD parallelism and to choose the best cache-blocking strategy. Intel MKL is also designed to be thread safe, which means that its functions operate correctly when simultaneously called from multiple application threads.

2.5.2 Parallelism:

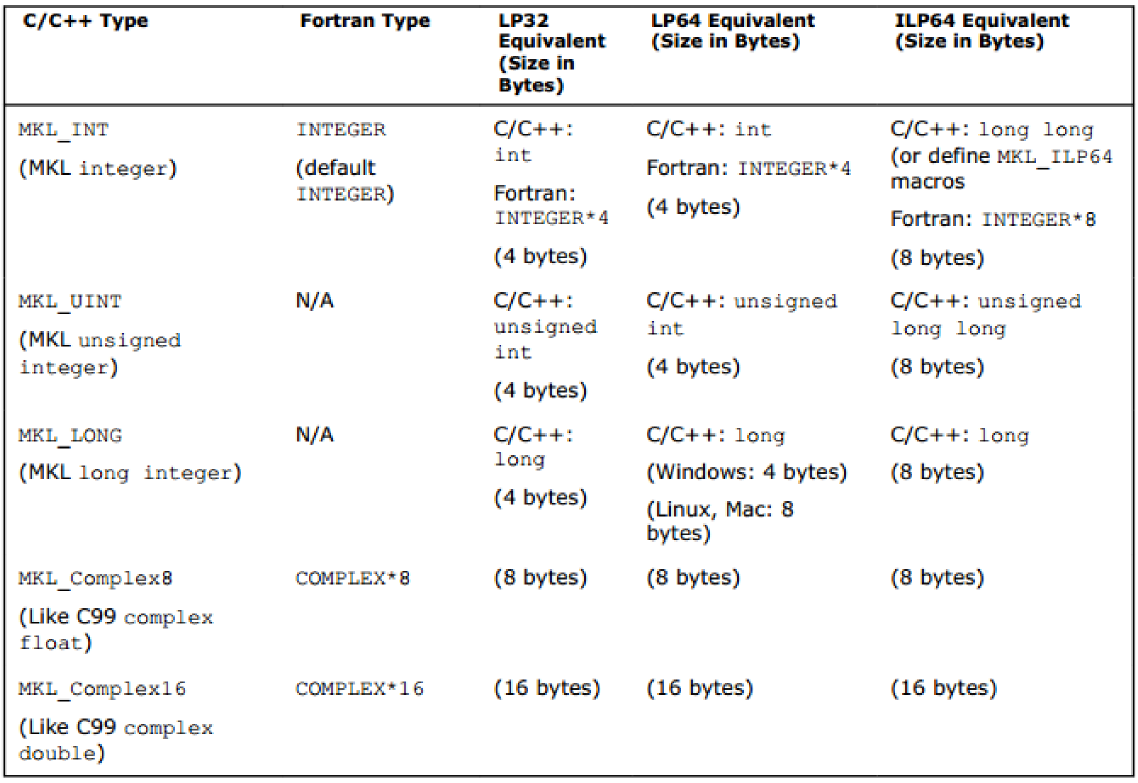
Intel® MKL offers performance gains through parallelism provided by the symmetric multiprocessing performance (SMP) feature. You can obtain improvements from SMP in the following ways:

- One way is based on user-managed threads in the program and further distribution of the operations over the threads based on data decomposition, domain decomposition, control decomposition, or some other parallelizing technique. Each thread can use any of the Intel MKL functions because the library has been designed to be thread-safe.
- Another method is to use the FFT and BLAS level 3 routines. They have been parallelized and require no alterations of your application to gain the performance enhancements of multiprocessing. Performance using multiple

processors on the level 3 BLAS shows excellent scaling. Since the threads are called and managed within the library, the application does not need to be recompiled thread-safe.

2.5.3 C Datatypes Specific to Intel MKL:

Intel® MKL defines new data types shown in the next figure:



C/C++ Type	Fortran Type	LP32 Equivalent (Size in Bytes)	LP64 Equivalent (Size in Bytes)	ILP64 Equivalent (Size in Bytes)
MKL_INT (MKL integer)	INTEGER (default INTEGER)	C/C++: int Fortran: INTEGER*4 (4 bytes)	C/C++: int Fortran: INTEGER*4 (4 bytes)	C/C++: long long (or define MKL_ILP64 macros Fortran: INTEGER*8 (8 bytes)
MKL_UINT (MKL unsigned integer)	N/A	C/C++: unsigned int (4 bytes)	C/C++: unsigned int (4 bytes)	C/C++: unsigned long long (8 bytes)
MKL_LONG (MKL long integer)	N/A	C/C++: long (4 bytes)	C/C++: long (Windows: 4 bytes) (Linux, Mac: 8 bytes)	C/C++: long (8 bytes)
MKL_Complex8 (Like C99 complex float)	COMPLEX*8	(8 bytes)	(8 bytes)	(8 bytes)
MKL_Complex16 (Like C99 complex double)	COMPLEX*16	(16 bytes)	(16 bytes)	(16 bytes)

Figure 2.3: Intel® MKL data types

2.6 Intel Cilk Plus

Intel Cilk Plus adds fine-grained task support to C and C++, making it easy to add parallelism to both new and existing software to efficiently exploit multiple processors and the vector instructions available on modern CPUs. It provides simple language extensions to express data and task parallelism to the C and C++ language implemented by the Intel C++ Compiler.

Intel Cilk Plus is made up of the following features:

- Task Parallelism

- Data Parallelism

2.6.1 Task Parallelism

1. Intel Cilk Plus adds three keywords to C and C++ to allow developers to express opportunities for parallelism:
    - Cilk_spawn: specifies that a function call can execute asynchronously, without requiring the caller to wait for it to return. This is an expression of an opportunity for parallelism, not a command that mandates parallelism. The Intel Cilk Plus runtime will choose whether to run the function in parallel with its caller.
    - Cilk_sync: specifies that all spawned calls in a function must complete before execution continues. There is an implied cilk_sync at the end of every function that contains a cilk_spawn.
    - Cilk_for: allows iterations of the loop body to be executed in parallel.

As stated above, the cilk_spawn and cilk_for keywords express opportunities for parallelism. Which portion of the application that actually runs in parallel is determined by Intel Cilk Plus runtime that implement task parallelism with an efficient work stealing scheduler.

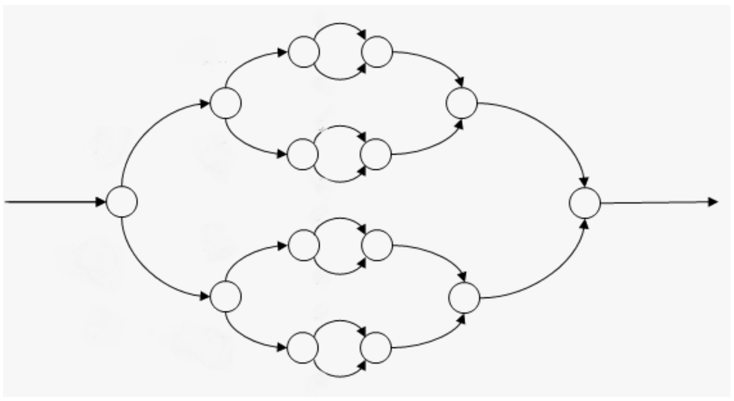


Figure 2.4: Cilk for example for multi-threads.

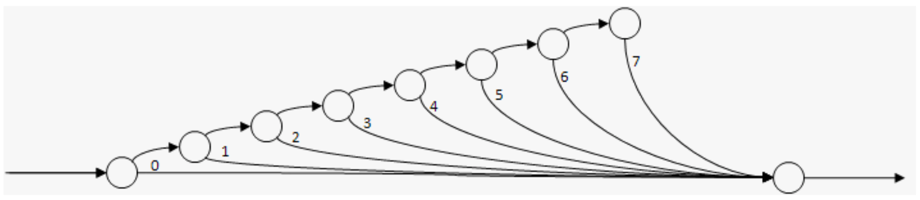


Figure 2.5: Example of Cilk spawn in a loop.

2. Reducers

   Intel Cilk Plus includes reducers to help make parallel programming easier. Traditional parallel programs use locks to protect shared variables, which can be problematic. Incorrect lock use can result in deadlocks. Contention for locked regions of code can slow a program down. And while locks can prevent races, there is no way to enforce ordering, resulting in non-deterministic results. Reducers provide a lock-free mechanism that allows parallel code to use private "views" of a variable which are merged at the next sync. The merge is done in an ordered manner to maintain the serial semantics of the Intel Cilk Plus application.

2.6.2 Data parallelism

1. Array notation: which provide data parallelism for sections/whole of array.

2. SIMD pragma: it is used to guide the compiler to vectorize more loops. Vectorization using the SIMD pragma complements (but does not replace) the fully automatic approach. [5] [6]

Chapter 3: Parallel Computing

3.1 Parallel computing vs Serial computing

Traditionally, software has been written for **serial computation** (see Figure 3.1):

- A problem is broken into a discrete series of instructions.
- Instructions are executed sequentially one after another.
- Executed on a single processor.
- Only one instruction may execute at any moment in time.

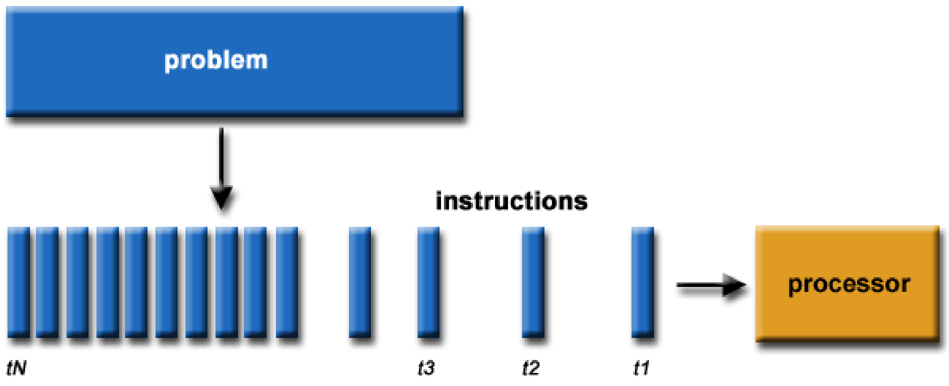


Figure 3.1: Serial Computing.

In the simplest sense, **parallel computing** (see Figure 3.2) is the simultaneous use of multiple compute resources to solve a computational problem [7]:

- A problem is broken into discrete parts that can be solved concurrently.
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different processors.
- An overall control/coordination mechanism is employed.

The computational problem should be able to:

- Be broken apart into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;

- Be solved in less time with multiple compute resources than with a single compute resource.

The compute resources are typically:

- A single computer with multiple processors/cores.
- An arbitrary number of such computers connected by a network.

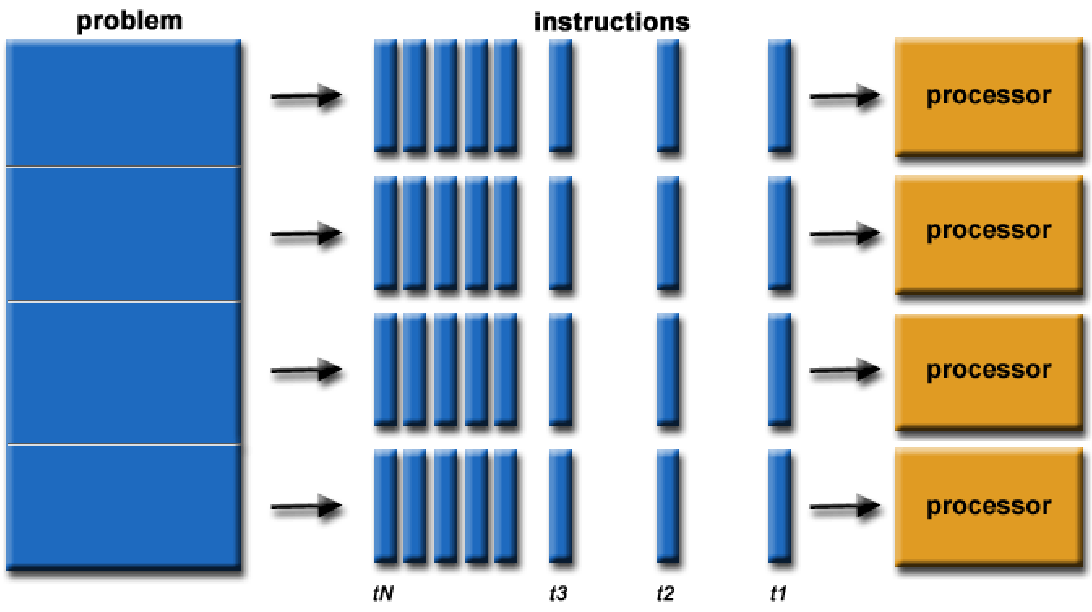


Figure 3.2: Parallel Computing.

3.2 Limits to Serial computing

Both physical and practical reasons pose significant constraints to simply building ever faster serial computers. Today, billions of transistors can fit on a chip the size of a five pence coin. This has meant that every couple of years, devices become more powerful, smaller and cheaper. But designing a processor to deliver high performance is far more than just increasing the clock rate.

3.2.1 Clock rates can't increase indefinitely

At first glance, it may seem that a processor simply executes a stream of instructions one after another, with performance increases attained through higher clock rates. However, increasing clock rate alone isn't enough. Power consumption and heat output increase as clock rates go up. With very high clock rates, significant increase in CPU core voltage become necessary and we eventually reach a point where excessive power consumption, heat output, and cooling requirements prevent further increases in clock rate. This limit was reached in 2004 (see Figure 3.3), in the days of the Pentium 4 Prescott. While recent improvements in power efficiency have helped, significant increases in clock rate are no longer feasible. [8]

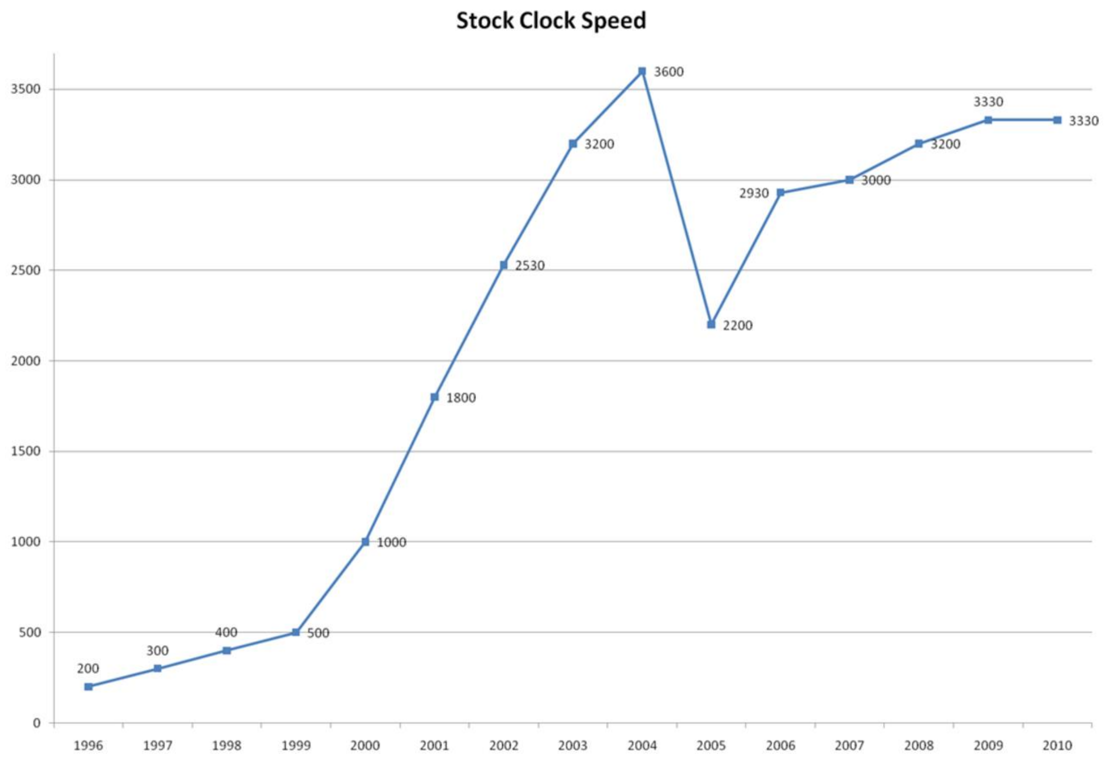


Figure 3.3: Stock clock speeds in cutting-edge enthusiast PCs over the years.

Now, the chip industry is no longer going to treat Gordon Moore's law, the beginning of the end started about a decade ago when Intel's CPU clock speed reached 4 GHz which led to many problems.

3.2.2 Overheating

In order to increase the rate, the same amount of electricity flows through the circuits more times per second. Thus the CPU uses more energy in the same time, its power consumption increases. Unfortunately, we don't have perfect conductors, so some of that power is lost as heat.

3.2.3 Transmission delays

Occur in the wires that connect things together on a chip. The "wires" on a chip are incredibly small aluminum or copper strips etched onto the silicon. A chip is nothing more than a collection of transistors and wires that hook them together, and a transistor is nothing but an on/off switch. When a switch changes its state from on to off or off to on, it has to either charge up or drain the wire that connects the transistor to the next transistor down the line. Imagine that a transistor is currently "on." The wire it is driving is filled with electrons. When the switch changes to "off," it has to drain off those electrons, and that takes time. The bigger the wire, the longer it takes. [9]

3.2.4 Parasitic capacitance

Each wire inside the CPU acts as a small capacitor. Also, the base of the transistor or the gate of the MOSFET act as small capacitors. In order to change the voltage on a connection, you must either charge the wire or remove the charge. As transistors shrink, it becomes more difficult to do that.

3.2.5 Slowness of the other system components

Even if we do manage to get a 3265810 THz processor working, another practical limit is how fast the rest of the system can support it. We either must have RAM, storage, glue logic, and other interconnects that perform just as fast, or you need an immense cache. [10]

In order to overcome the problems caused by increasing CPU clock rate without affecting the performance, today’s microprocessors focus on making the most effective use of each processing cycle instead of increasing processors clock speed which generate too much heat. For example, properly utilized CPU instruction sets can make some operations process more efficiently and reducing the amount of cycles needed to perform an operation, increasing the effectiveness of parallel processing, as well as increasing the efficiency of code execution.

3.3 Types of Parallel computing

There are several Types of Parallel Computing which are used worldwide.

3.3.1 Bit-Level Parallelism

From the advent of very-large-scale integration (VLSI) computer-chip fabrication technology in the 1970s until about 1986, speed-up in computer architecture was driven by doubling computer word size (the amount of information the processor can execute per cycle). Increasing the word size reduces the number of instructions the processor must execute to perform an operation on variables whose sizes are greater than the length of the word. For example, where an 8-bit processor must add two 16-bit integers, the processor must first add the 8 lower-order bits from each integer using the standard addition instruction, then add the 8 higher-order bits using an add-with-carry instruction and the carry bit from the lower order addition; thus, an 8-bit processor requires two instructions to complete a single operation, where a 16-bit processor would be able to complete the operation with a single instruction.

Historically, 4-bit microprocessors were replaced with 8-bit, then 16-bit, then 32-bit microprocessors. This trend generally came to an end with the introduction of 32-bit processors, which has been a standard in general-purpose computing for two decades. Not until recently (c. 2003–2004), with the advent of x86-64 architectures, have 64-bit processors become commonplace.

3.3.2 Instruction-Level Parallelism

A computer program is, in essence, a stream of instructions executed by a processor. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mid-1980s until the mid-1990s. Modern processors have multi-stage instruction pipelines. Each stage in the pipeline corresponds to a different action the processor performs on that instruction in that stage; a processor with an N-stage pipeline can have up to N different instructions at different stages of completion.

In addition to instruction-level parallelism from pipelining, some processors can issue more than one instruction at a time. These are known as superscalar processors. Instructions can be grouped together only if there is no data dependency between them.

Instruction-level parallelism (ILP) is a measure of how many of the operations in a computer program can be performed simultaneously.

3.3.3 Data Parallelism

Data parallelism is parallelism inherent in program loops, which focuses on distributing the data across different computing nodes to be processed in parallel. Parallelizing loops often leads to similar (not necessarily identical) operation sequences or functions being performed on elements of a large data structure. Many scientific and engineering applications exhibit data parallelism.

A loop-carried dependency is the dependence of a loop iteration on the output of one or more previous iterations. Loop-carried dependencies prevent the parallelization of loops.

3.3.4 Task parallelism

Task parallelism (also known as function parallelism and control parallelism) is a form of parallelization of computer code across multiple processors in parallel computing environments. Task parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. It contrasts to data parallelism as another form of parallelism.

In a multiprocessor system, task parallelism is achieved when each processor executes a different thread (or process) on the same or different data. The threads may execute the same or different code. In the general case, different execution threads communicate with one another as they work. Communication takes place usually to pass data from one thread to the next as part of a workflow.

3.4 Classes of parallel computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. [11]

3.4.1 Multi-Core computing

A multi-core CPU (or chip-level multiprocessor, CMP) combines two or more independent cores into a single package composed of single integrated circuits(IC), called a die, or more dies packaged together. A dual-core processor contains two cores, and a quad-core processor contains four cores. A multi-core microprocessor implements multiprocessing in a single physical package. A processor with all cores on a single die is called a monolithic processor.

3.4.2 Symmetric multiprocessing

In computing, **symmetric multiprocessing** or **SMP** involves a multiprocessor computer-architecture where two or more identical processors can connect to a single shared main memory. Most common multiprocessor systems today use an SMP architecture.

In case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors. SMP systems allow any processor to work on any task no matter where the data for that task are located in memory; with proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.

3.4.3 Distributed computing

Distributed computing deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime.

In distributed computing a program is split up into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often must deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers.

3.4.4 Grid computing

Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the Internet to work on a given problem. Because of the low bandwidth and extremely high latency available on the Internet, grid computing typically deals only with embarrassingly parallel problems.

3.4.5 Vector processors

A vector processor is a CPU or computer system that can execute the same instruction on large sets of data. "Vector processors have high-level operations that work on linear arrays of numbers or vectors. An example vector operation is $A = B \times C$, where $A, B,$ and C are each 64-element vectors of 64-bit floating-point numbers.

3.4.6 Accelerators

Another approach of parallel computing is to use a different hardware more suited for performing certain tasks as a co-processor. The non-CPU hardware in this configuration is known as an Accelerator. One of the most popular accelerators is the GPU which composed of hundreds of cores that can handle thousands of threads simultaneously.

In recent years, these accelerators are attracting a lot of attention. This is mainly due to the fact that the generic CPU's floating point arithmetic capability has leveled off at around 10 GFLOPS, while GPUs can perform between 100 GFLOPS and 1 TFLOPS for a relatively inexpensive price. It is also more Green, which makes it a better option than cluster server systems, since many factories and research labs are trying to cut back on the power usage.

In summary, an accelerator allows for a low-cost, low-powered, high-performance system. However, the transfer speed between the host CPU and the accelerator can become a bottle neck, making it unfit for applications requiring frequent I/O operations. Thus, a decision to use a hybrid system, as well as what type of a hybrid system, needs to be made wisely.

3.5 The limits of parallel computing

3.5.1 Amdahl's Law

Amdahl's law is often used in parallel computing to predict the theoretical speedup when using multiple processors. For example, if a program needs 20 hours using a single processor core, and a particular part of the program which takes one hour to execute cannot be parallelized, while the remaining 19 hours $(P = 0.95)$ of execution time can be parallelized, then regardless of how many processors are devoted to a parallelized execution of this program, the minimum execution time cannot be less than that critical one hour. Hence, the theoretical speedup is limited to at most 20 times ($\frac{1}{1-P} = 20$). For this reason, parallel computing with many processors is useful only for very parallelizable programs (see Figure 3.4). [12]

This relationship is given by the equation:

$$S(N) = \frac{1}{(1-P) - \frac{P}{N}} \tag{3.1}$$

where:

- $S(N)$ is the theoretical speedup of the execution of the whole task;
- P is the proportion of execution time that the part benefiting from improved resources originally occupied;
- N is the speedup of the part of the task that benefits from improved system resources.

3.5.2 Gustafson's Law

Gustafson's law is another law in computer engineering, closely related to Amdahl's law (see Figure 3.5). It can be formulated as [13]:

$$S(P) = P - \alpha(P-1) \tag{3.2}$$

where:

- $S(P)$ is the theoretical speedup of the execution of the whole task;
- P is the number of processors;
- α is the non-parallelizable part of the process.

The reason for the discrepancy between the speed up predictions by the two laws is that Gustafson's Law assumes that the amount of work to be done increases as the number of processor increases! Amdahl's Law, on the other hand, assumes the amount of work to be done is static no matter how much parallelization is available. Gustafson's Law essentially calculates how much computation can be done in a given amount of time with a given amount of parallelization, and divides that by the amount of time that the same amount of computation would have taken without parallelization.

So if we're parallelizing an algorithm which can expand the amount of computation to be done to fit the amount of parallelization available (for example a particle filter), then Gustafson's Law will be more appropriate. If the amount of computation to be done is fixed and parallelization can't change that, then Amdahl's Law is more appropriate.

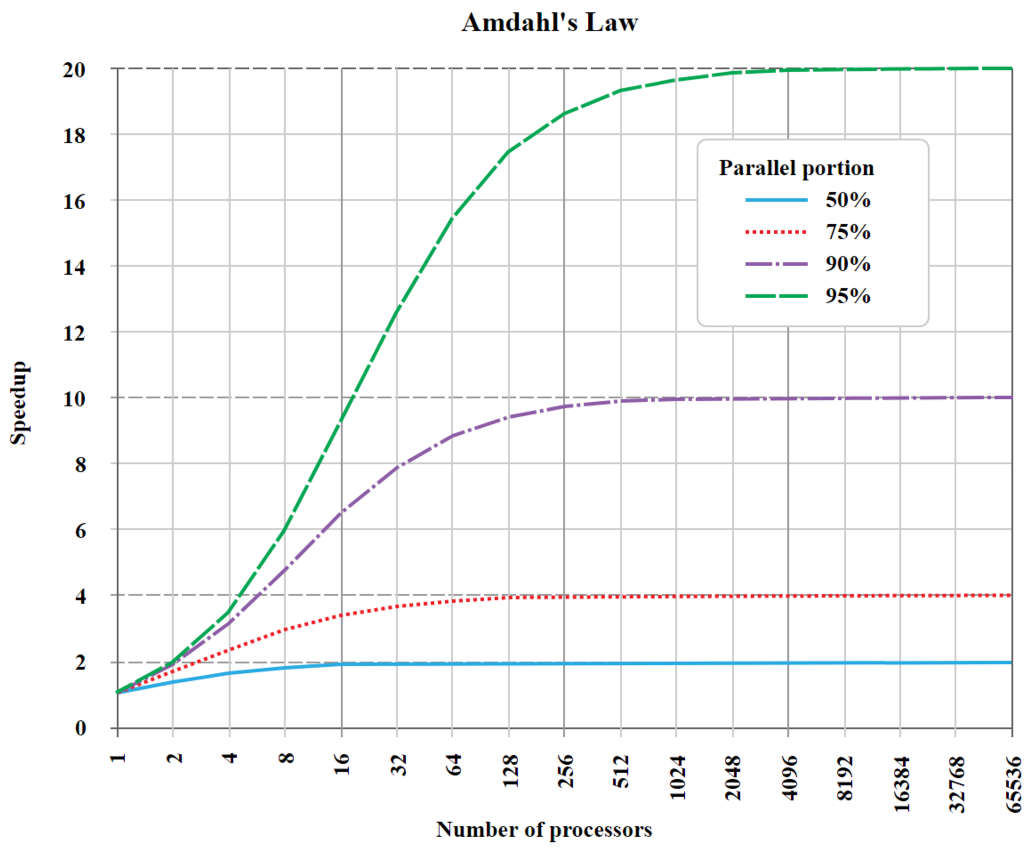


Figure 3.4: Evolution according to Amdahl's law of the theoretical speedup in latency of the execution of a program in function of the number of processors executing it.

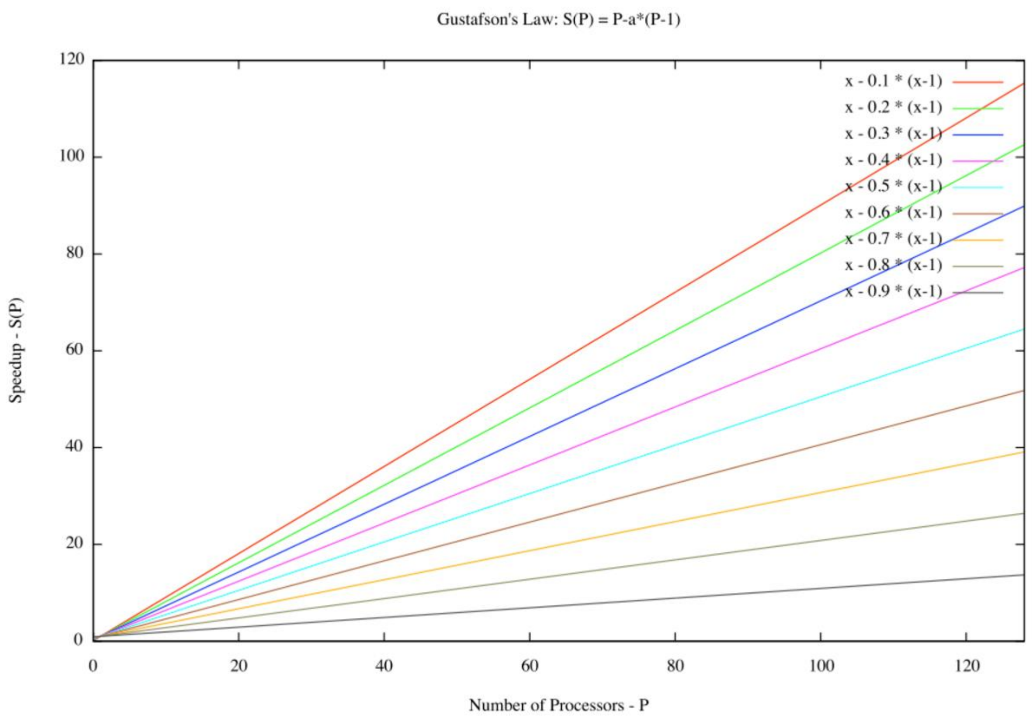


Figure 3.5: Evolution according to Gustafson's Law of the theoretical speedup in latency of the execution of a program in function of the number of processors executing it, for different values of p.

3.6 Graphics Processing Unit

August 31, 1999 marks the introduction of the Graphics Processing Unit (GPU) for the PC industry. The technical definition of a GPU is “a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second” [14].

The GPU’s advanced capabilities were originally used primarily for 3D game rendering. But now those capabilities are being harnessed more broadly to accelerate computational workloads in areas such as financial modeling, cutting-edge scientific research and oil and gas exploration.

GPUs are optimized for taking huge batches of data and performing the same operation over and over very quickly, unlike PC microprocessors, which tend to skip all over the place.

Architecturally, the CPU is composed of just few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously (see Figure

3.6). The ability of a GPU with 100+ cores to process thousands of threads can accelerate some software by 100x over a CPU alone. What’s more, the GPU achieves this acceleration while being more power- and cost-efficient than a CPU [15].

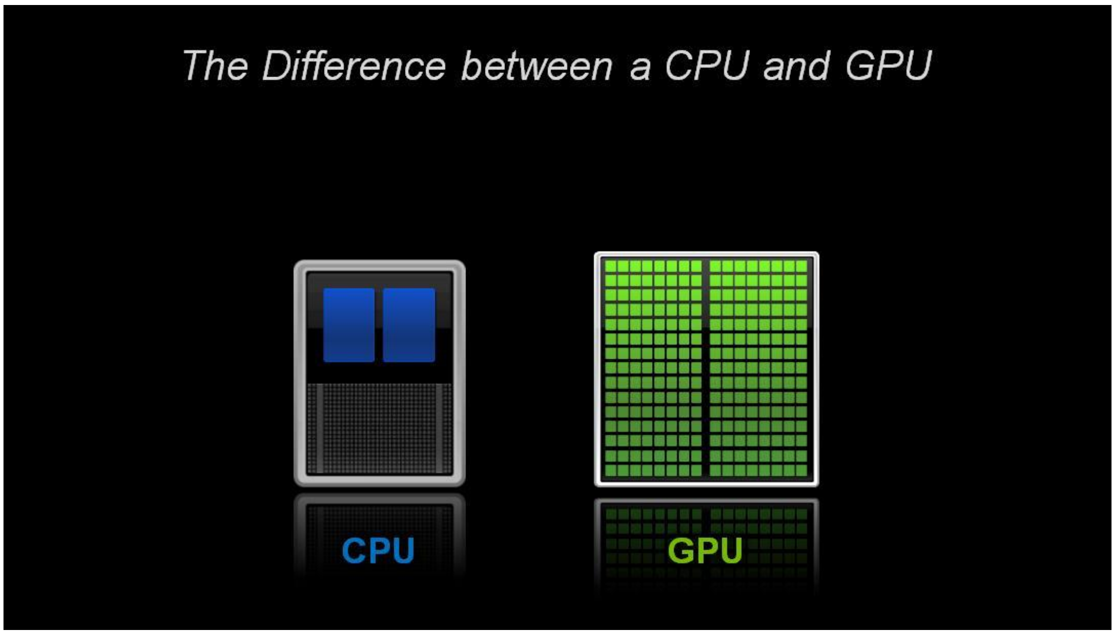


Figure 3.6: The difference between a CPU and GPU.

3.6.1 GPU-accelerated computing

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate deep learning, analytics, and engineering applications. Pioneered in 2007 by NVIDIA, GPU accelerators now power energy-efficient data centers in government labs, universities, enterprises, and small-and-medium businesses around the world.

GPU-accelerated computing offloads compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU as shown in Figure 3.7. From a user's perspective, applications simply run much faster. [16]

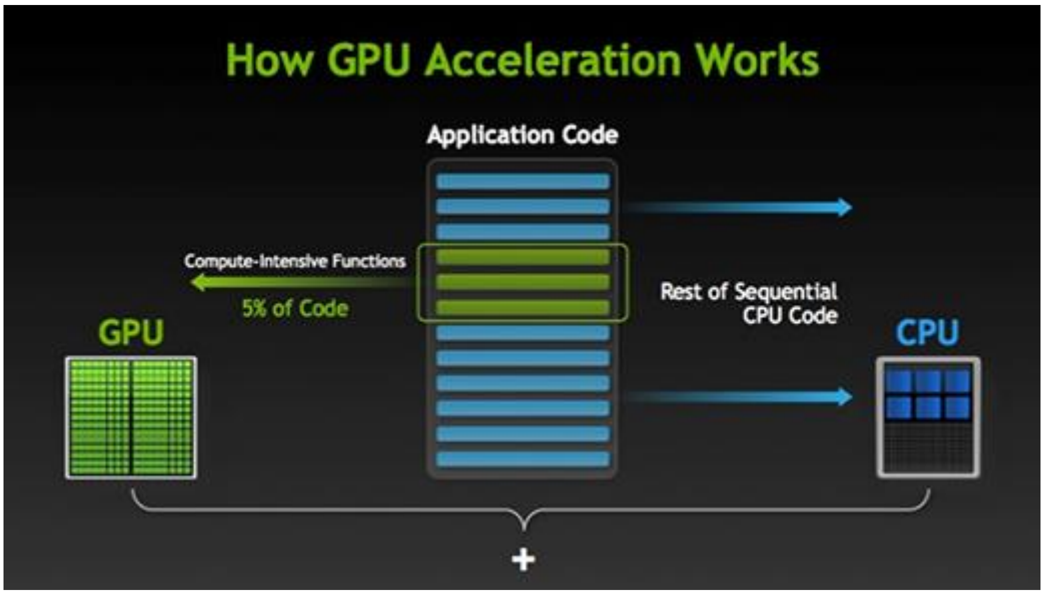

How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential CPU Code
GPU
CPU
+

Figure 3.7: How GPU accelerates software applications.

3.6.2 General-purpose computing on graphics processing units

General-purpose computing on graphics processing units (GPGPU) is the use of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU). The use of multiple video cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip [17].

CUDA and OpenCL (Open Computing Language) are two interfaces for GPU computing, both presenting similar features but through different programming interfaces. OpenCL is an open standard that can be used to program CPUs, GPUs, and other devices from different vendors, while CUDA is specific to NVIDIA GPUs.

3.7 CUDA

CUDA is a parallel computing platform and programming model invented by NVIDIA that makes using a GPU for general purpose computing simple and elegant. The developer still programs in the familiar C, C++, Fortran, or an ever expanding list of supported languages, and incorporates extensions of these languages in the form of a few basic keywords.

Using CUDA allows the programmer to take advantage of the massive parallel computing power of an NVIDIA graphics card in order to do general purpose computation. CPUs like Intel Core 2 Duo and AMD Opteron are good at doing one or two tasks at a time, and doing those tasks very quickly. Graphics cards, on the other hand, are good at doing a massive number tasks at the same time, and doing those tasks relatively quickly.

To put this into perspective, suppose you have a 20-inch monitor with a standard resolution of 1920 x 1200. NVIDIA graphics card has the computational ability to calculate the color of 2,304,000 different pixels, many times a second. In order to accomplish this feat, graphics cards use dozens, even hundreds of ALUs.

Fortunately, NVIDIA's ALUs are fully programmable, which enables us to harness an unprecedented amount of computational power into the programs that we write.

As stated previously, CUDA lets the programmer take advantage of the hundreds of ALUs inside a graphics processor, which is much more powerful than the handful of ALUs available in any CPU. However, this does put a limit on the types of applications that are well suited to CUDA.

3.7.1 When to Use CUDA

CUDA is only well suited for highly parallel algorithms: In order to run efficiently on a GPU, you need to have many hundreds of threads. Generally, the more threads you have, the better. If you have an algorithm that is mostly serial, then it does not make sense to use CUDA. Many serial algorithms do have parallel equivalents, but many do not. If you can't break your problem down into at least a thousand threads, then CUDA probably is not the best solution for you.

CUDA is extremely well suited for number crunching: If there is one thing that CUDA excels at, it's number crunching. The GPU is fully capable of doing 32-bit integer and floating point operations. In fact, it GPUs are more suited for floating point computations, which makes CUDA an excellent for number crunching. Some of the higher end graphics cards do have double floating point units, however there is only one 64-bit floating point unit for every 16 32-bit floating point units. So, using double floating point numbers with CUDA should be avoided if they aren't absolutely necessary for your application.

CUDA is well suited for large datasets: Most modern CPUs have a couple megabytes of L2 cache because most programs have high data coherency. However, when working quickly across a large dataset, say 500 Megabytes, the L2 cache may not be as helpful. The memory interface for GPUs is very different from the memory interface of CPUs. GPUs use massive parallel interfaces in order to connect with its memory. This type of interface is approximately 10 times faster than a typical CPU to memory interface, which is great.

3.7.2 CUDA Programming Model Basics

The CUDA programming model is a heterogeneous model in which both the CPU and GPU are used. In CUDA, the host refers to the CPU and its memory, while the device refers to the GPU and its memory. Code run on the host can manage memory on both the host and device, and also launches kernels which are functions executed on the device. These kernels are executed by many GPU threads in parallel. Given the heterogeneous nature of the CUDA programming model, a typical sequence of operations for a CUDA C program is:

1. Declare and Initialize host data.
2. Declare and allocate device memory.
3. Transfer data from the host to the device.
4. Execute one or more kernels.
5. Transfer results from the device to the host.

3.7.3 CUDA Limitations

- Unlike OpenCL, CUDA-enabled GPUs are only available from Nvidia.
- Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency.
- Exception handling is not supported in CUDA code.
- Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands.

Chapter 4: Communications

4.1 Introduction

Mobile communication has become an everyday commodity. In the last decades, it has evolved from being an expensive technology for a few selected individuals to today's ubiquitous systems used by a majority of the world's population. From the first experiments with radio communication by Guglielmo Marconi in the 1890s, the road to truly mobile radio communication has been quite long. To understand the complex mobile-communication systems of today, it is important to understand where they came from and how cellular systems have evolved. The task of developing mobile technologies has also changed, from being a national or regional concern to becoming an increasingly complex task undertaken by global standards-developing organizations such as the Third Generation Partnership Project (3GPP) and involving thousands of people.

Mobile communication technologies are often divided into generations (see Figure 4.1), with 1G being the analog mobile radio systems of the 1980s, 2G the first digital mobile systems, and 3G the first mobile systems handling broadband data. The next generation, 4G or Long-Term Evolution (LTE), provides even better support for mobile broadband. Further evolution steps of 4G LTE will be taken within the next few years. In a longer term perspective, around 2020 one may enter into what some would call "5G" radio access. This continuing race of increasing sequence numbers for mobile system generations is in fact just a matter of labels. What is important is the actual system capabilities and how they have evolved. [18]

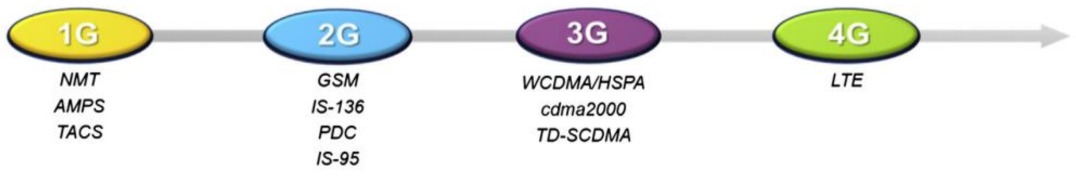


Figure 4.1: Generations of mobile communication systems.

4.2 Quick Overview of Wireless Standards

In the past two decades we have seen the introduction of various mobile standards, from 2G to 3G to the present 4G, and we expect the trend to continue (see Figure 4.2). The primary mandate of the 2G standards was the support of mobile telephony and voice applications. The 3G standards marked the beginning of the packet-based data revolution and the support of Internet applications such as email, Web browsing, text messaging, and other client-server services. The 4G standards will feature all-IP packet-based networks and will support the explosive demand for bandwidth-hungry applications such as mobile video-on-demand services.

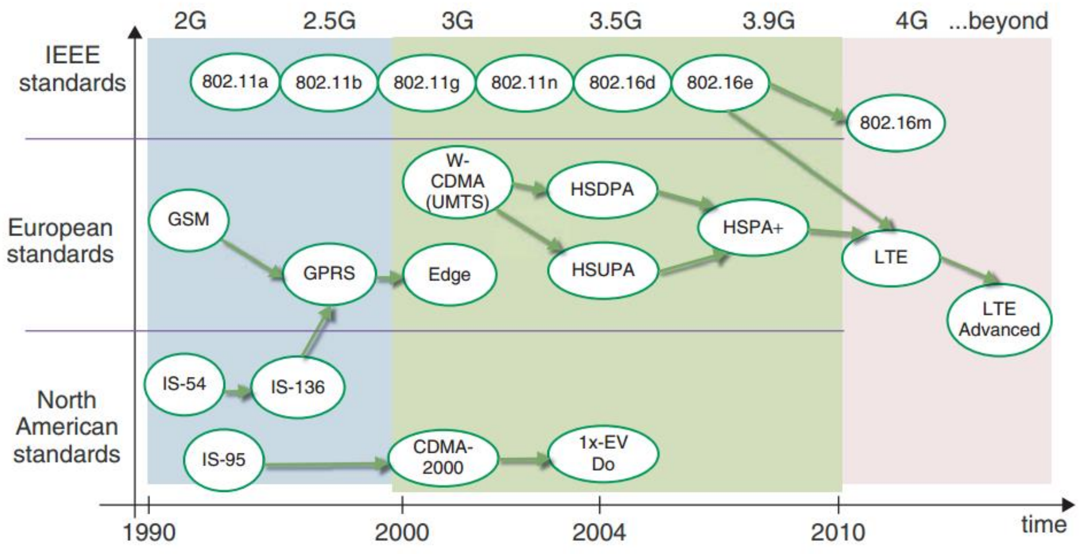


Figure 4.2: Evolution of wireless standards in the last two decades.

Historically, standards for mobile communication have been developed by consortia of network providers and operators, separately in North America, Europe, and other regions of the world. The second-generation (2G) digital mobile communications systems were introduced in the early 1990s. The technology supporting these 2G systems were circuit-switched data communications. The GSM (Global System for Mobile Communications) in Europe and the IS-54 (Interim Standard 54) in North America were among the first 2G standards. Both were based on the Time Division Multiple Access (TDMA) technology. In TDMA, a narrowband communication channel is subdivided into a number of time slots and multiple users share the spectrum at allocated slots. In terms of data rates, for example, GSM systems support voice services up to 13 kbps and data services up to 9.6 kbps.

The GSM standard later evolved into the Generalized Packet Radio Service (GPRS), supporting a peak data rate of 171.2 kbps. The GPRS standard marked the introduction of the split-core wireless networks, in which packet-based switching technology supports data transmission and circuit-switched technology supports voice transmission. The GPRS technology further evolved into Enhanced Data Rates for Global Evolution (EDGE), which introduced a higher-rate modulation scheme (8-PSK, Phase Shift Keying) and further enhanced the peak data rate to 384 kbps.

In North America, the introduction of IS-95 marked the first commercial deployment of a Code Division Multiple Access (CDMA) technology. CDMA in IS-95 is based on a direct spread spectrum technology, where multiple users share a wider bandwidth by using orthogonal spreading codes. IS-95 employs a 1.2284 MHz bandwidth and allows for a maximum of 64 voice channels per cell, with a peak data rate of 14.4 kbps per fundamental channel. The IS-95-B revision of the standard was developed to support high-speed packet-based data transmission. With the introduction of the new supplemental code channel supporting high-speed packet data, IS-95-B supported a peak data rate of 115.2 kbps. In North America 3GPP2 (Third Generation Partnership Project 2) was the standardization body that established technical specifications and standards for 3G mobile systems based on the evolution of CDMA technology. From 1997 to 2003, 3GPP2 developed a family of standards based on the original IS-95 that included 1xRTT, 1x-EV-DO (Evolved Voice Data Only), and EV-DV (Evolved Data and Voice). 1xRTT doubled the IS-95 capacity by adding 64 more traffic channels to achieve a peak data rate of 307 kbps. The 1x-EV-DO and 1x-EV-DV standards achieved peak data rates in the range of 2.4–3.1 Mbps by introducing a set of features including adaptive modulation and coding, hybrid automatic repeat request (HARQ), turbo coding, and faster scheduling based on smaller frame sizes.

The 3GPP (Third-Generation Partnership Project) is the standardization body that originally managed European mobile standard and later on evolved into a global standardization organization. It is responsible for establishing technical specifications for the 3G mobile systems and beyond. In 1997, 3GPP started working on a standardization effort to meet goals specified by the ITU IMT-2000 (International Telecommunications Union International Mobile Telecommunication) project. The goal of this project was the transition from a 2G TDMA-based GSM technology to a

3G wide-band CDMA-based technology called the Universal Mobile Telecommunications System (UMTS). The UMTS represented a significant change in mobile communications at the time. It was standardized in 2001 and was dubbed Release 4 of the 3GPP standards. The UMTS system can achieve a downlink peak data rate of 1.92 Mbps. As an upgrade to the UMTS system, the High-Speed Downlink Packet Access (HSDPA) was standardized in 2002 as Release 5 of the 3GPP. The peak data rates of 14.4 Mbps offered by this standard were made possible by introducing faster scheduling with shorter subframes and the use of a 16QAM (Quadrature Amplitude Modulation) modulation scheme. High-Speed Uplink Packet Access (HSUPA) was standardized in 2004 as Release 6, with a maximum rate of 5.76 Mbps. Both of these standards, together known as HSPA (High-Speed Packet Access), were then upgraded to Release 7 of the 3GPP standard known as HSPA+ or MIMO (Multiple Input Multiple Output) HSDPA. The HSPA+ standard can reach rates of up to 84 Mbps and was the first mobile standard to introduce a 2 × 2 MIMO technique and the use of an even higher modulation scheme (64QAM). Advanced features that were originally introduced as part of the North American 3G standards were also incorporated in HSPA and HSPA+. These features include adaptive modulation and coding, HARQ, turbo coding, and faster scheduling.

Another important wireless application that has been a driving force for higher data rates and spectral efficiency is the wireless local area network (WLAN). The main purpose of WLAN standards is to provide stationary users in buildings (homes, offices) with reliable and high-speed network connections. As the global mobile communications networks were undergoing their evolution, IEEE (Institute of Electrical and Electronics Engineers) was developing international standards for WLANs and wireless metropolitan area networks (WMANs). With the introduction of a family of WiFi standards (802.11a/b/g/n) and WiMAX standards (802.16d/e/m), IEEE established Orthogonal Frequency Division Multiplexing (OFDM) as a promising and innovative air-interface technology. For example, the IEEE 802.11a WLAN standard uses the 5 GHz frequency band to transmit OFDM signals with data rates of up to 54 Mb/s. In 2006, IEEE standardized a new WiMAX standard (IEEE 802.16m) that introduced a packet-based wireless broadband system. Among the features of WiMAX are scalable bandwidths up to 20 MHz, higher peak data rates, and better special efficiency profiles than were being offered by the UMTS and HSPA systems at the time. This advance essentially kicked off the effort by 3GPP to introduce a new wireless mobile standard that could compete with the WiMAX technology. This effort ultimately led to the standardization of the LTE standard.

Table 4.1: Peak data rates of various wireless standards introduced over the past two decades.

Technology	Theoretical peak data rate (at low mobility)
GSM	9.6 kbps
IS-95	14.4 kbps
GPRS	171.2 kbps
EDGE	473 kbps
CDMA-2000 (1xRTT)	307 kbps
WCDMA (UMTS)	1.92 Mbps
HSDPA (Rel 5)	14 Mbps
CDMA-2000 (1x-EV-DO)	3.1 Mbps
HSPA+ (Rel 6)	84 Mbps
WiMAX (802.16e)	26 Mbps
LTE (Rel 8)	300 Mbps
WiMAX (802.16m)	303 Mbps
LTE-Advanced (Rel 10)	1 Gbps

Table 4.1 summarizes the peak data rates of various wireless technologies. Looking at the maximum data rates offered by these standards, the LTE standard (3GPP release 8) is specified to provide a maximum data rate of 300 Mbps. The LTE-Advanced (3GPP version 10) features a peak data rate of 1 Gbps. These figures represent a boosts in peak data rates of about 2000 times above what was offered by GSM/EDGE technology and 50–500 times above what was offered by the W-CDMA/UMTS systems. This remarkable boost was achieved through the development of new technologies introduced within a time span of about 10 years. One can argue that this extraordinary advancement is firmly rooted in the elegant mathematical formulation of the enabling technologies featured in the LTE standards. It is our aim in this book to clarify and explain these enabling technologies and to put into context how they combine to achieve such a performance. We also aim to gain insight into how to simulate, verify, implement, and further enhance the PHY (Physical Layer) technology of the LTE standards. [19]

4.3 Long Term Evolution (4G)

LTE or Long Term Evolution is the brand name given to the efforts of 3GPP 4th Generation technology development efforts mostly in Europe and UMB (Ultra-Mobile Broadband) is the brand name for similar efforts by 3GPP2 in North America.

LTE have been first time introduced in 3GPP Release 8. This essential evolution will enable networks to offer the higher data throughput to mobile terminals needed in order to deliver new and advanced mobile broadband services. The primary objectives of this network evolution are to provide these services with a quality at least equivalent to what an end-user can enjoy today using their fixed broadband access at home, and to reduce operational expenses. LTE is also referred to as E-UTRA (Evolved UMTS Terrestrial Radio Access) or E-UTRAN (Evolved UMTS Terrestrial Radio Access Network).

Although 3G/3.5G technologies such as HSPA/EV-DO deliver significantly higher bit rates than 2G technologies, they do not fully satisfy the wireless broadband requirements of instant-on, always-on and multi-megabit throughput. With LTE delivering even higher peak throughput and much lower latency, mobile operators (either 3GPP or 3GPP2 based) have a unique opportunity to evolve their existing infrastructure to next generation wireless networks.

4.4 LTE Requirements

LTE is focusing on an optimum support of Packet Switched (PS) services. Main requirements for the design of an LTE system were identified in the beginning of the standardization work on LTE in 2004. They can be summarized as follows [20]:

Data Rate: Peak data rates target 100 Mbps (downlink) and 50 Mbps (uplink) for 20 MHz spectrum allocation, assuming 2 receive antennas and 1 transmit antenna at the terminal.

Throughput: Target for downlink average user throughput per MHz is 3-4 times better than 3GPP Release 6. Target for uplink average user throughput per MHz is 2-3 times better than 3GPP Release 6.

Spectrum Efficiency: Downlink target is 3-4 times better than 3GPP Release 6. Uplink target is 2-3 times better than 3GPP Release 6. Table 4.2 summarizes the data rate and spectrum efficiency requirements set for LTE.

Table 4.2: Data rate and spectrum efficiency requirements defined for LTE.

Downlink (20 MHz)			Uplink (20 MHz)		
Unit	**Mbps**	**bps/Hz**	**Unit**	**Mbps**	**bps/Hz**
Requirement	100.0	5.0	**Requirement**	50.0	2.5
2×2 MIMO	172.8	8.6	**16QAM**	57.6	2.9
4×4 MIMO	326.4	16.3	**64QAM**	86.4	4.3

Latency: User plane latency. The one-way transit time between a packet being available at the IP layer in either the device or radio access network and the availability of this packet at IP layer in the radio access network/device shall be less than 30 ms.

Control plane latency. Also C-plane, that means the time it takes to transfer the device from a passive connection with the network (IDLE state) to an active connection (CONNECTED state) shall be further reduced, e.g. less than 100 ms to allow fast transition times.

Bandwidth: LTE supports a subset of bandwidths of 1.4, 3, 5, 10, 15 and 20 MHz.

Mobility: The system should be optimized for low mobile speed (0-15 km/h), but higher mobile speeds shall be supported as well including high speed train environment as special case.

Spectrum allocation: Operation in paired (Frequency Division Duplex / FDD mode) and unpaired spectrum (Time Division Duplex / TDD mode) is possible.

Quality of Service: End-to-end Quality of Service (QoS) shall be supported. Voice over Internet Protocol (VoIP) should be supported with at least as good radio and backhaul efficiency and latency as voice traffic over the UMTS circuit switched networks.

4.5 LTE Advantages

4.5.1 IP network based protocol

The high-level network architecture of LTE is comprised of following three main components:

- The User Equipment (UE)
- The Evolved UMTS Terrestrial Radio Access Network (E-UTRAN)
- The Evolved Packet Core (EPC)

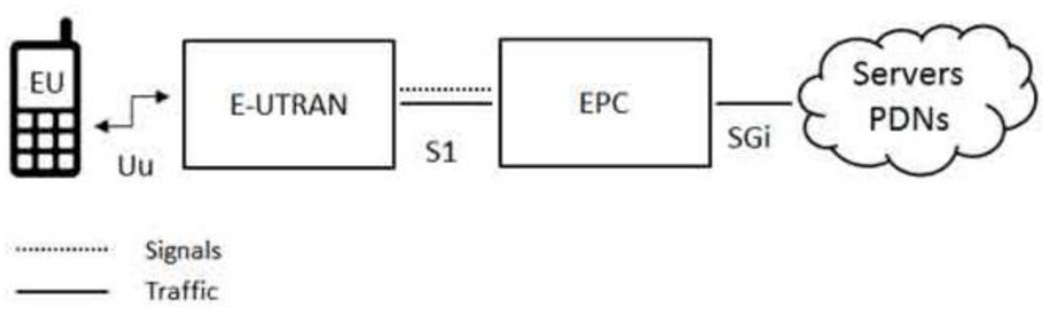


Figure 4.3: High level LTE network architecture.

For wireless service providers without a wire-line legacy, there are huge cost reductions in the wireless environment enabled by all-IP, namely the transition to LTE networks. However, wireless service providers, who have transitioned to LTE, still operate two different network architectures, circuit-switched for voice and packet-based for data. Once this redundancy can be converged, wireless networks will become much more consolidated, cheaper to operate, more fully featured, and interoperable

4.5.2 Higher data rate

LTE provides a big jump in Data rates by using many techniques like using MIMO (Multi-input Multi-output) techniques especially in spatial multiplexing mode, where each antenna at the transmitter side will provide different data symbols to reach antennas at the receiver side. Also using higher modulation schemes like 64QAM provides higher data rates but totally it depends on the channel.

4.5.3 UE battery life enhancement

Because of using the concept of SC-FDMA modulator, the issue of the PAPR (Peak to average power ratio) is completely resolved. The PAPR is generated due to the IFFT summation of multiple parallel subcarriers. That result higher power consumption in signal generation which isn’t accessible by UE.

4.5.4 Total user capacity increase

Because of using OFDMA technique in the downlink at the base station side the total capacity of the overall system has been increased. As in OFDM is based on the concept of the multi-carrier modulation at which the channel is divided into small narrow channels, each considered to be a flat fading channel that increases the efficiency of the data transmitted then the capacity of the channel would be increased as a result.

4.5.5 Quality improvement

The improved speed and low latency provided by LTE will offer a much improved end- user experience for all corporate services:

- For applications where data throughput is important - faster email and file uploads, enhanced VPN connection, high-speed internet.
- For interactive applications where latency is crucial - IMS based VoIP, mail and file synchronization with an on-line server, peer-to-peer applications, SIP multimedia services including video and voice conference over IP, application sharing.
- LTE will enable the introduction of new services, such as High Definition Video (or HD TV) and multi-user interactive gaming

4.6 Interference

One of the most challenging issues facing wireless communication systems is the interference, Interference is anything which modifies, or disrupts a signal as it travels along a channel between a source and a receiver. The term typically refers to the addition of unwanted signals to a useful signal.

4.6.1 Inter-symbol Interference

Inter-symbol interference is a signal distortion in telecommunication. One or more symbols can interfere with other symbols causing noise or a less reliable signal. The main causes of inter-symbol interference are multipath propagation or non-linear frequency in channels. This has the effect of a blur or mixture of symbols, which can reduce signal clarity. If inter-symbol interference occurs within a system, the receiver output becomes erroneous at the decision device. This is an unfavorable result that should be reduced to the most minimal amount possible. Error rates from inter-symbol interference are minimized through the use of adaptive equalization techniques and error correcting codes.

4.6.2 Inter Channel Interference

When signal bandwidth of adjacent carrier frequencies overlap with other, giving rise to inter channel interference. That can lead to symbols distortion.

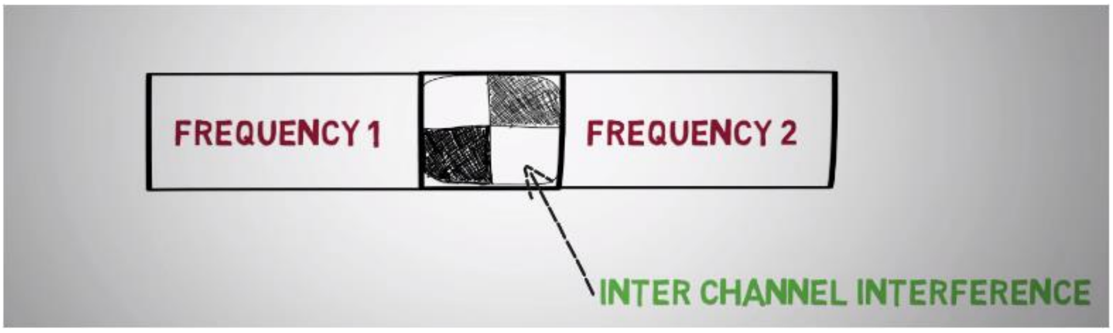


Figure 4.4: Inter Channel Interference between two adjacent carriers.

To avoid the occurrence of ICI (Inter Channel Interference) guard bands were introduced but that may waste the bandwidth. So in OFDM the carriers are all orthogonal to each other’s to solve this problem.

But still CP (Cyclic prefix is needed) to overcome the ISI problem.

4.7 Multiplexing & Multiple Access Techniques

Multiplexing is a way of sending multiple signals or streams of information over a communications link at the same time in the form of a single complex signal, the receiver recovers the separate signals (Demultiplexing). Two basic forms of multiplexing are Time Division Multiplexing (TDM), and Frequency Division Multiplexing (FDM).

Multiple access is a technique that lets multiple mobile users share the allotted spectrum in the most effective manner. Since the spectrum is limited, the sharing is necessary to improve the overall capacity over a geographical area. This is carried out by permitting the available bandwidth to be used simultaneously by different users.

One of the key elements of LTE is the use of OFDM (Orthogonal Frequency Division Multiplexing) as the signal bearer, as well as OFDM’s associated access schemes, OFDMA (Orthogonal Frequency Division Multiple Access) and SC-FDMA (Single Carrier Frequency Division Multiple Access).

4.7.1 OFDM

OFDM (Orthogonal Frequency Division Multiplexing) is a form of signal modulation is being used for many of the latest wireless and telecommunications standards which divides a high data rate modulating stream placing them onto many slowly modulated narrowband close-spaced subcarriers, and in this way is less sensitive to frequency selective fading as they have a bandwidth smaller than the mobile channel coherence bandwidth. This obviates the need for complex frequency equalizers which are featured in 3G technologies. The sub-carriers are mutually orthogonal in the frequency domain which mitigates Inter-Symbol Interference (ISI) as shown in Figure 4.5.

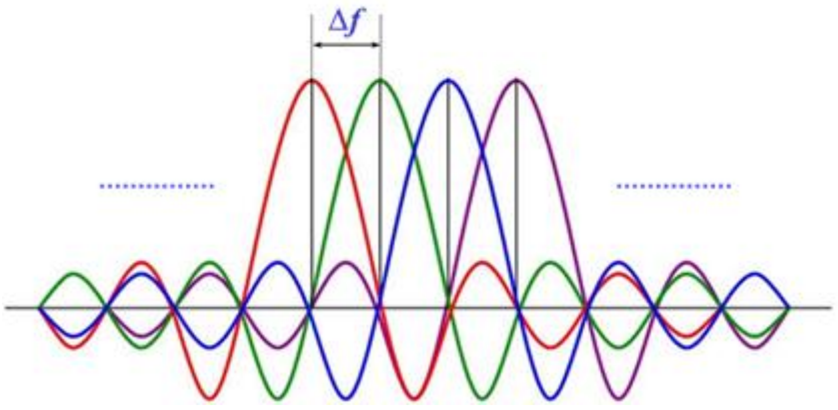


Figure 4.5: OFDM Spectrum.

The data to be transmitted on an OFDM signal is spread across the subcarriers of the signal, each subcarrier taking part of the payload. This reduces the data rate taken by each subcarrier. The lower data rate has the advantage that interference from reflections is much less critical. This is achieved by adding a guard band time or guard interval into the system.

Input bits are first grouped and assigned for transmission over different frequencies (sub-carriers) and then summed up this can be done using IFFT (Inverse Fast Fourier Transformation) and the resulting signal could then be sent over the air as shown in Figure 4.6.

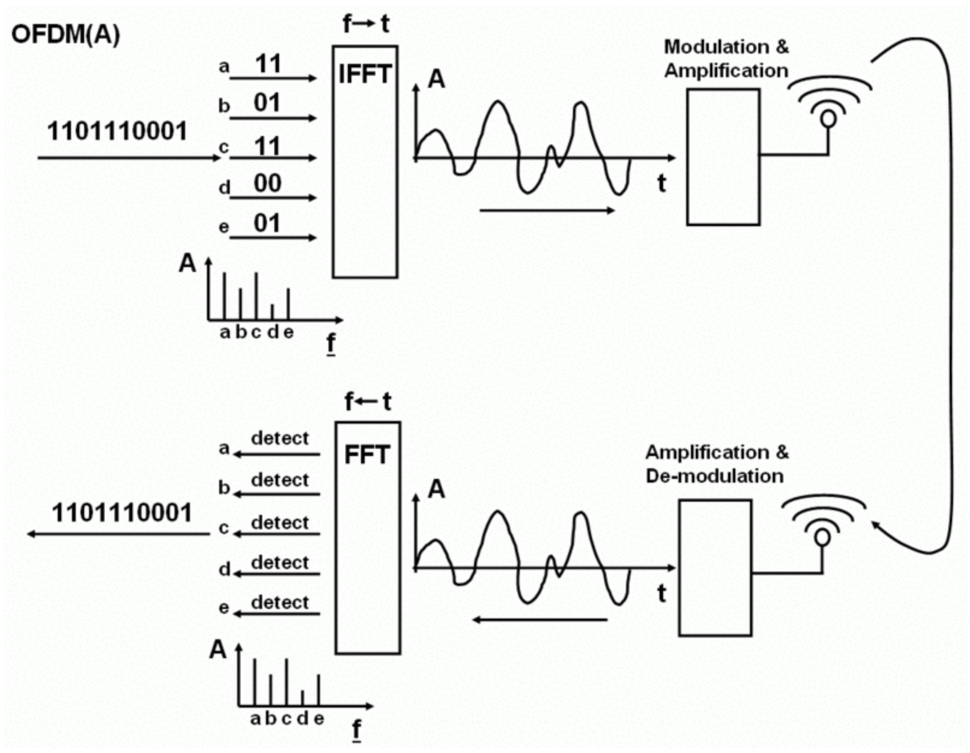


Figure 4.6: OFDM(A) Architecture.

OFDM has been adopted in the Wi-Fi arena where the standards like 802.11a, 802.11n, 802.11ac and more. It has also been chosen for the cellular telecommunications standard LTE / LTE-A, and in addition to this it has been adopted by other standards such as WiMAX. [21]

4.7.1.1 Cyclic Prefix

In telecommunications, the term cyclic prefix refers to the prefixing of a symbol with a repetition of the end. Although the receiver is typically configured to discard the cyclic prefix samples, the cyclic prefix serves two purposes.

- As a guard interval, it eliminates the intersymbol interference from the previous symbol.
- As a repetition of the end of the symbol, it allows the linear convolution of a frequency-selective multipath channel to be modelled as circular convolution, which in turn may be transformed to the frequency domain using a discrete Fourier transform. This approach allows for simple frequency-domain processing, such as channel estimation and equalization.

In order for the cyclic prefix to be effective (i.e. to serve its aforementioned objectives), the length of the cyclic prefix must be at least equal to the length of the multipath channel. Different OFDM cyclic prefix lengths are available in various systems. For example, within LTE a normal length and an extended length are available and after Release 8 a third extended length is also included, although not normally used.

Although the concept of cyclic prefix has been traditionally associated with OFDM systems, the cyclic prefix is now also used in single carrier systems to improve the robustness to multipath propagation.

Cyclic prefix disadvantage is that it takes up system capacity and reduces the overall data rate as the cyclic prefix re-transmits data that is already being transmitted. [22]

4.7.1.2 OFDM Advantages & Disadvantages

OFDM has been used in many high data rate wireless systems because of the many advantages it provides [21]:

- ***Immunity to selective fading:*** OFDM is more resistant to frequency selective fading than single carrier systems because it divides the overall channel into multiple narrowband signals that are affected individually as flat fading sub-channels.

- ***Resilience to Bandlimited Interference:*** Interference appearing on a channel may be bandwidth limited and in this way will not affect all the sub-channels. This means that not all the data is lost.
- ***Spectrum efficiency:*** Using close-spaced overlapping sub-carriers, a significant OFDM advantage is that it makes efficient use of the available spectrum.
- ***Resilient to ISI:*** This results from the low data rate on each of the sub-channels.
- ***Simpler channel equalization:*** One of the issues with CDMA systems was the complexity of the channel equalization which had to be applied across the whole channel. An advantage of OFDM is that using multiple sub-channels, the channel equalization becomes much simpler.
- ***Data rate Optimization:*** Allows optimization of data rates for all users in a cell by transmitting on the best (i.e. non-faded) subcarriers for each user.

This last feature is the fundamental aspect of OFDMA, the use of OFDM technology to multiplex traffic by allocating specific patterns of sub-carriers in the time-frequency space to different users. In addition to data traffic, control channels and reference symbols can be interspersed.

Whilst OFDM has been widely used, there are still a few disadvantages to its use which need to be addressed when considering its use:

- ***High peak to average power ratio:*** An OFDM signal has a noise like amplitude variation and has a relatively high large dynamic range, or peak to average power ratio. This impacts the RF amplifier efficiency as the amplifiers need to be linear and accommodate the large amplitude variations and these factors mean the amplifier cannot operate with a high efficiency level.

- ***Sensitive to carrier offset and drift:*** Another disadvantage of OFDM is that is sensitive to carrier frequency offset and drift. Single carrier systems are less sensitive.

4.7.2 SC-FDMA

For the LTE uplink, a different concept is used for the access technique. Although still using a form of OFDMA technology, the implementation is called Single Carrier Frequency Division Multiple Access (SC-FDMA).

One of the key parameters that affects all mobiles is that of battery life. Even though battery performance is improving all the time, it is still necessary to ensure that the mobiles use as little battery power as possible. With the RF power amplifier that transmits the radio frequency signal via the antenna to the base station being the highest power item within the mobile, it is necessary that it operates in as efficient mode as possible. This can be significantly affected by the form of radio frequency modulation and signal format. Signals that have a high peak to average ratio and require linear amplification do not lend themselves to the use of efficient RF power amplifiers. As a result, it is necessary to employ a mode of transmission that has as near a constant power level when operating. Unfortunately, OFDM has a high peak to average ratio. While this is not a problem for the base station where power is not a particular problem, it is unacceptable for the mobile. As a result, LTE uses a modulation scheme known as SC-FDMA - Single Carrier Frequency Division Multiplex which is a hybrid format. This combines the low peak to average ratio offered by single-carrier systems with the multipath interference resilience and flexible subcarrier frequency allocation that OFDM provides [23].

Despite its name, Single Carrier Frequency Division Multiple Access (SC-FDMA) also transmits data over the air interface in many sub-carriers but adds an additional processing step as shown in Figure 4.7. Instead of putting M bits together (e.g. 4 bits representing a 16-QAM modulation) as in the OFDM to form the signal for one sub-carrier, the additional processing block in SC-FDMA spreads the information of each bit over all the sub-carriers. This is done as follows: Again, a number of bits (e.g. 4 bits) are grouped together. In OFDM, these groups of bits would have been the input of the IFFT. In SC-FDMA, however, these bits are now piped into a Fast Fourier Transformation (FFT) function first. The output of the process is the basis for the creation of the sub-carriers for the following IFFT. As not all sub-carriers are used by the mobile station, many of them are set to zero in the diagram. These may or may not be used by other mobile stations.

On the receiver side the signal is demodulated, amplified and treated by the Fast Fourier Transformation function in the same way as in OFDMA. The resulting amplitude diagram, however, is now not analyzed straight away to get the original data stream but fed to the Inverse Fast Fourier Transformation function to remove the effect of the additional signal processing originally done at the transmitter side. The result of the IFFT is again a time domain signal. The time domain signal is now fed to a single detector block which recreates the original bits. Thus, instead of detecting the bits on many different sub-carriers, only a single detector is used on a single carrier [24].

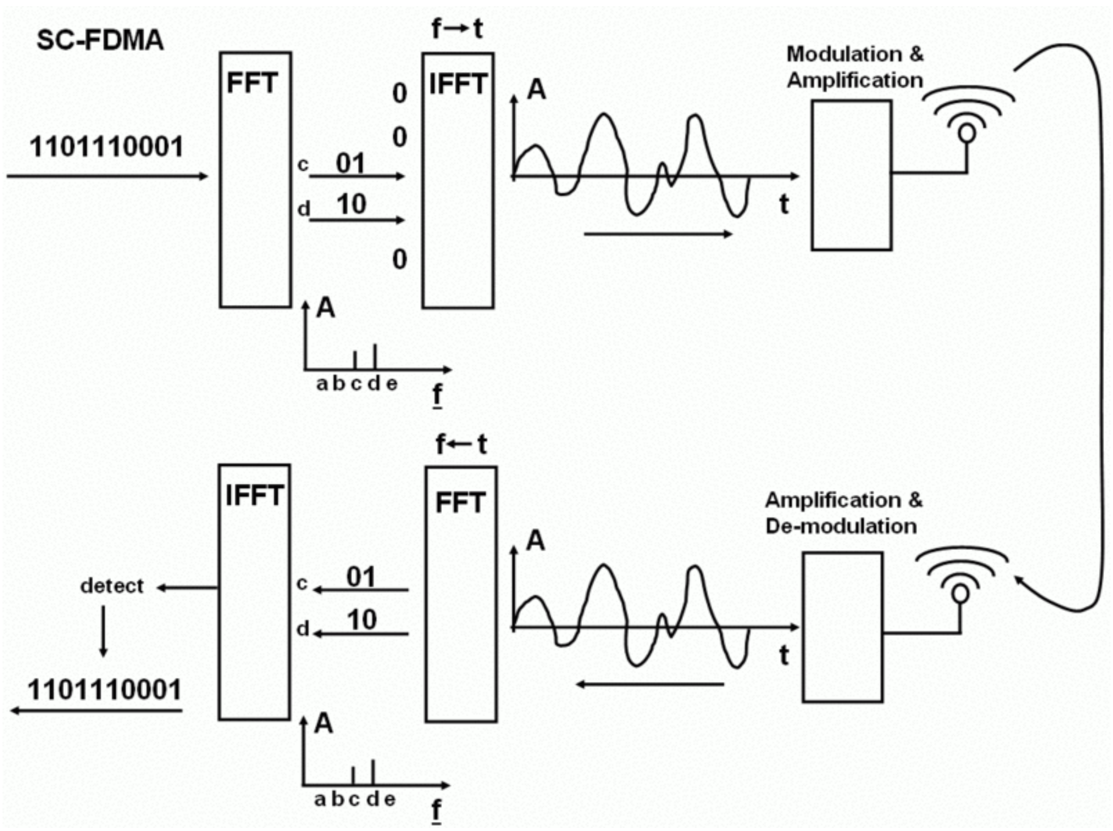


Figure 4.7: SC-FDMA Architecture.

4.7.3 Summary of the difference between OFDM and SC-FDMA:

OFDM takes groups of input bits (0's and 1's) to assemble the sub-carriers which are then processed by the IFFT to get a time signal. SC-FDMA in contrast first runs an FFT over the groups of input bits to spread them over all sub-carriers and then uses the result for the IFFT which creates the time signal. This is why SC-FDMA is sometimes also referred to as FFT spread OFDM.

Looking to Figure 4.8 it is possible to see the difference between SC-FDMA and OFDMA. Also, it is possible to notice that the intersymbol interference will be reduced since all subcarriers on a period of time represent the same symbol.

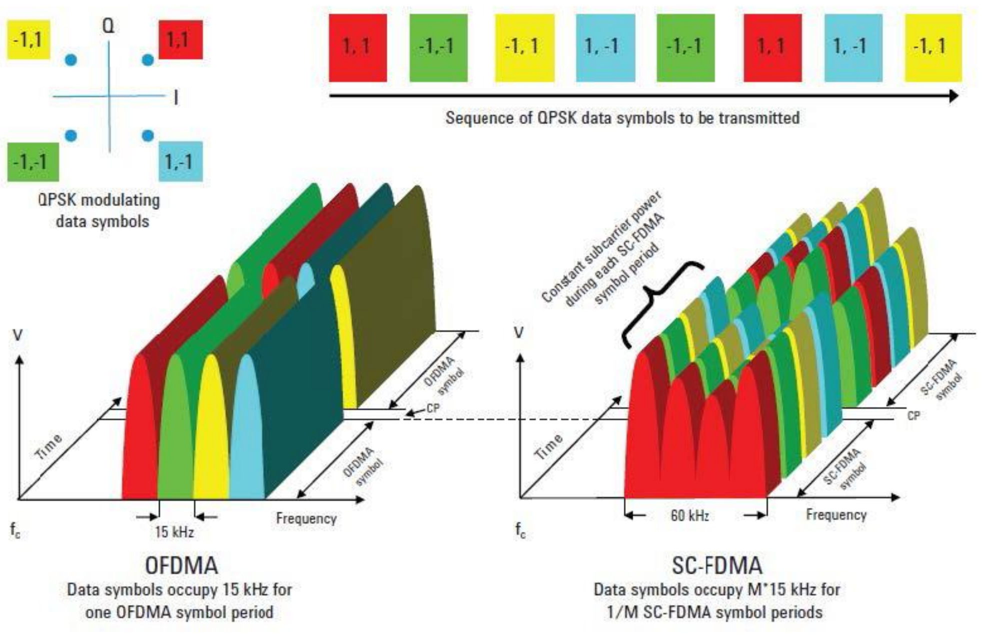

-1,1
Q
1,1
I
-1,-1
1,-1
QPSK modulating data symbols
1, 1
-1,-1
-1, 1
1, -1
-1,-1
1, 1
1, -1
-1, 1
Sequence of QPSK data symbols to be transmitted
V
Time
OFDMA symbol
CP
OFDMA symbol
15 kHz
Frequency
f_c
OFDMA
Data symbols occupy 15 kHz for one OFDMA symbol period
Constant subcarrier power during each SC-FDMA symbol period
V
Time
SC-FDMA symbol
CP
SC-FDMA symbol
60 kHz
Frequency
f_c
SC-FDMA
Data symbols occupy M*15 kHz for 1/M SC-FDMA symbol periods

Figure 4.8: OFDMA vs. SC-FDMA.

4.8 LTE communication channels

The information flow between the different protocols are known as channels and signals. LTE uses several different types of channels, which are distinguished by the type of information they carry and by the way in which the information is processed. [25]

1. Logical Channels

   Define “what type of data” is transferred, e.g. traffic channels (DTCH, MTCH) and control channels (CCCH, …). Data and signaling messages are carried on logical channels between the RLC and MAC protocols.
2. Transport Channels

   Define “how is” data transferred over the air, e.g. what are encoding, interleaving options used to transmit data. Data and signaling messages are carried on transport channels between the MAC and the physical layer.
3. Physical Channels

Define "where is" data transferred over the air, i.e. the frequency where the data will be sent. Data and signaling messages are carried on physical channels according to the mapping specified in Figure 4.9.

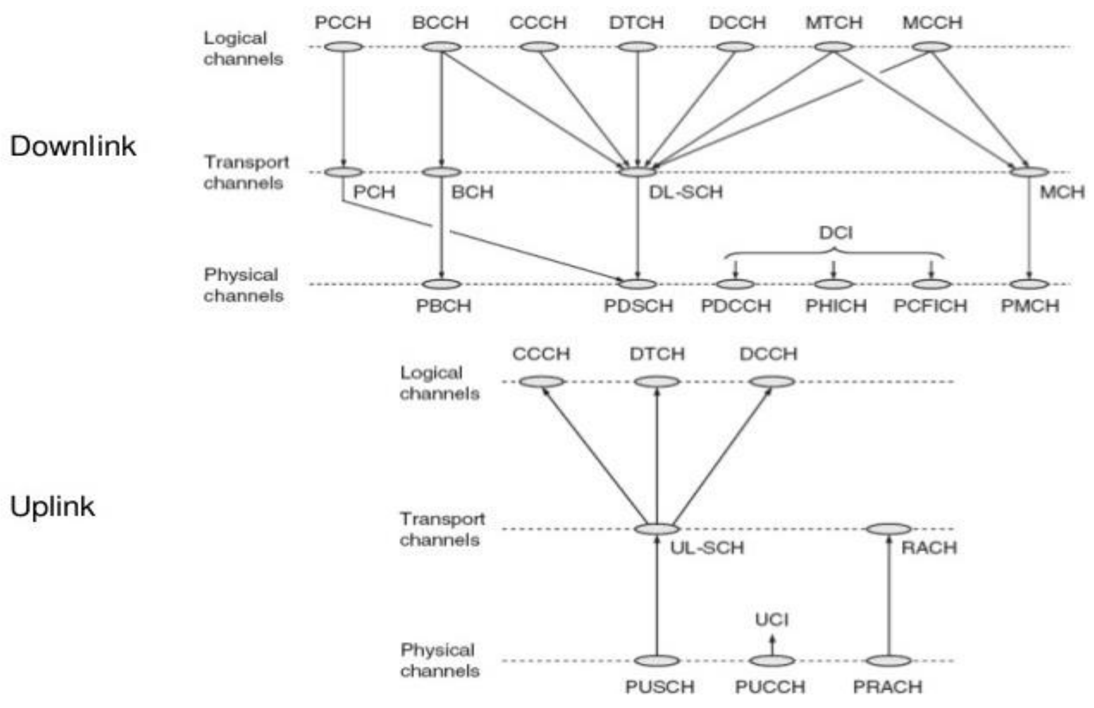


Figure 4.9: Mapping between LTE downlink and uplink communication channels.

4.8.1 Logical and physical channels classification

Both Logical and physical channels can be classified into two types:

1. Control Channels
2. Traffic Channels

For the transport channels, no dedicated channels are used anymore in LTE to reduce complexity of the LTE protocol architecture and the number of transport channels. [20]

4.9 Physical uplink channels

The physical uplink channels are: physical uplink shared channel (PUSCH), physical uplink control channel (PUCCH) and physical random access channel (PRACH). The research made in this project focuses on the physical uplink shared channel (PUSCH). The physical uplink shared channel (PUSCH) is used by uplink users (UEs) to transmit data to the base station (eNodeB). The modulation schemes supported by PUSCH are QPSK, 16QAM and 64QAM. The PUSCH uses the single carrier frequency division multiple access (SC-FDMA) for transmitting the data of the UEs. The Information bits are first interleaved then scrambled before modulation mapping, DFT-spreading, sub-carrier mapping and OFDM modulation. [20]

4.10 Physical layer parameters

There are two types of frame structure in the LTE standard, Type 1 and Type 2. Type 1 uses Frequency Division Duplexing (FDD) in which uplink and downlink are separated by frequency. Type 2 uses Time Division Duplexing (TDD) in which uplink and downlink are separated in time.

For full-duplex FDD, uplink and downlink frames are separated by frequency and are transmitted continuously and synchronously as shown in the figure below.

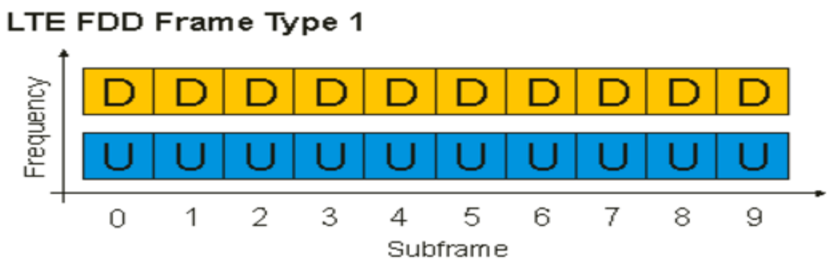


Figure 4.10: LTE FDD frame (Type 1).

In half-duplex FDD operation, the UE cannot transmit and receive at the same time.

For TDD mode, the uplink and downlink subframes are transmitted on the same frequency and are multiplexed in the time domain. The locations of the uplink, downlink and special subframes are determined by the uplink-downlink configuration.

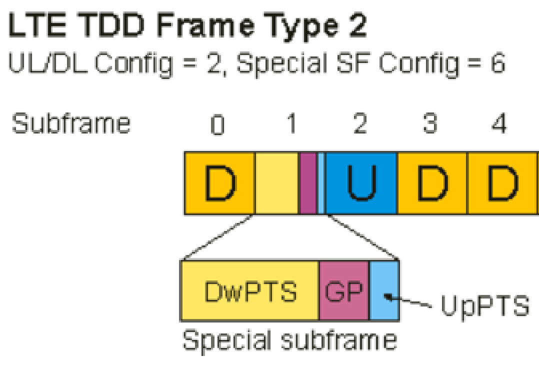


Figure 4.11: LTE TDD Frame (Type 2).

The figure above shows a radio frame with a special subframe in the 2nd and 6th subframes. Special subframes are used for switching from downlink to uplink and contain three sections: DwPTS (downlink pilot time slot), GP (guard period), and UpPTS (uplink pilot time slot). [26]

The radio frame has a length of 10 ms ($T_{frame} = 307200 * T_s$). Each frame is divided into 10 equally sized subframes of 1 ms. Each subframe consists of 2 equally sized slots of 0.5 ms. Each slot in turn consists of a number of OFDM symbols which can be either 7 (for normal cyclic prefix) or 6 (for extended cyclic prefix). T_s (sampling time) expresses the basic time unit for LTE, corresponding to a sampling frequency of 30.72 MHz. This sampling frequency is given due to the defined subcarrier spacing for LTE with $\Delta f = 15\, kHz$ and the maximum size for FFT to generate the OFDM symbols is 2048, $F_s = 2048 * 15\, kHz = 30.72\, MHz$. [20]

The figure below shows the structure of the uplink resource grid for both FDD and TDD. The horizontal axis shows the time domain of the subframe, i.e. the SC-FDMA symbols in the subframe. While the vertical axis shows the frequency domain of one resource block.

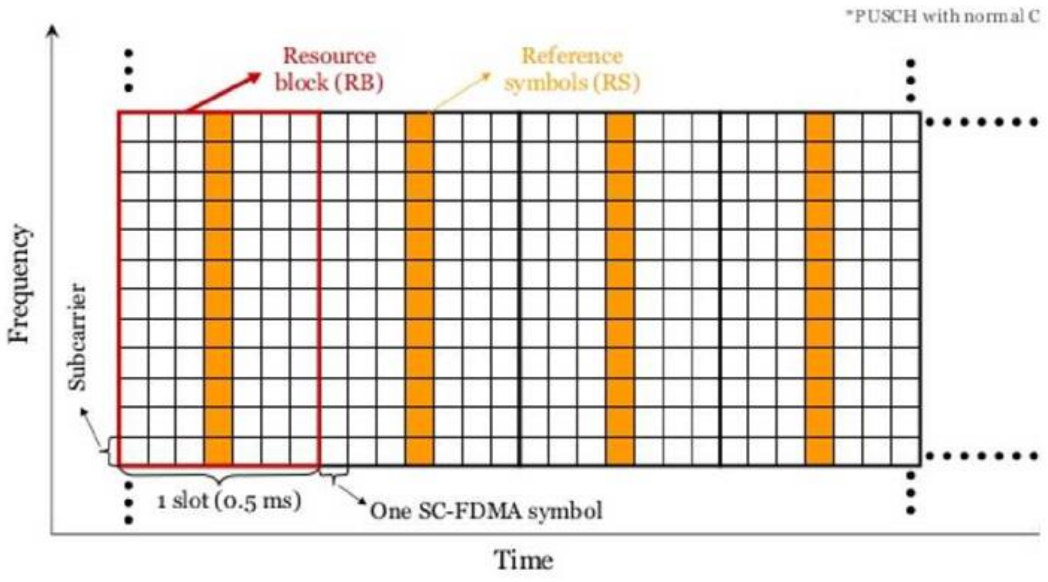


Figure 4.12: LTE uplink grid.

The smallest unit of resource is the Resource Element (RE) which consists of one SC-FDMA data block length on one sub-carrier. Resource element contains a single complex value representing data from a physical channel or signal. A resource block consists of 12 REs for the duration of a slot ($0.5\ ms$). The resource block (RB) is the smallest unit that can be allocated to a user. In frequency, resource blocks are either 12 subcarriers x 15 kHz subcarriers spacing or 24 subcarriers x 7.5 kHz subcarriers spacing. The resource block is 180 kHz wide in frequency and 1 slot long in time. [27]

Data is allocated to a device (User Equipment, UE) in terms of resource blocks. one UE can be allocated integer multiples of one resource block in the frequency domain. Each UE can use resource blocks assigned to it for a transmission time interval (TTI) of $1ms$. All scheduling decisions for downlink and uplink are done in the base station (eNodeB). [20]

4.10.1 Demodulation reference signal (DMRS)

Uplink demodulation reference signals are used for channel estimation in the eNodeB receiver for coherent demodulation of the Physical Uplink Shared Channel (PUSCH) to which the UL-SCH transport channel is mapped, as well as for the Physical Uplink Control Channel (PUCCH), which carries different types of L1/L2 control signaling. It is located on the 4th symbol in each slot (for normal cyclic prefix) and spans the same bandwidth as the allocated uplink data. The basic structure for demodulation reference signals is the same for PUSCH and PUCCH transmission, although there are some differences - for example, in terms of the exact set of SC-FDMA symbols in which the reference signals are transmitted. [20]

4.11 Bandwidth

The bandwidths defined by the standard are $1.4, 3, 5, 10, 15, and\ 20\ MHz$. The table below shows how many subcarriers and resource blocks in each bandwidth for uplink. [28]

Table 4.3: Number of resource blocks for different LTE bandwidths.

Channel Bandwidth	**1.4**	**3**	**5**	**10**	**15**	**20**
Number of resource blocks	6	15	25	50	75	100

4.12 Channel coding, and interleaving

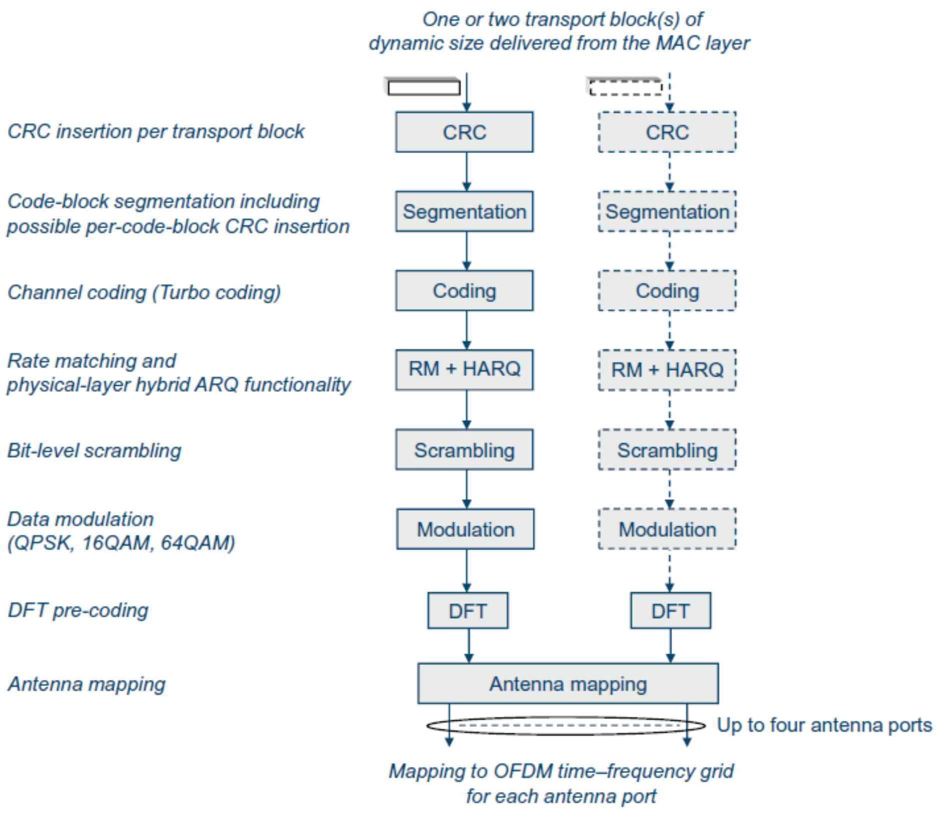

One or two transport block(s) of dynamic size delivered from the MAC layer
CRC insertion per transport block
Code-block segmentation including possible per-code-block CRC insertion
Channel coding (Turbo coding)
Rate matching and physical-layer hybrid ARQ functionality
Bit-level scrambling
Data modulation (QPSK, 16QAM, 64QAM)
DFT pre-coding
Antenna mapping
CRC
Segmentation
Coding
RM + HARQ
Scrambling
Modulation
DFT
Antenna mapping
Up to four antenna ports
Mapping to OFDM time–frequency grid for each antenna port

Figure 4.4.13: Physical-layer processing for UL-SCH.

Data and control streams from/to MAC layer are encoded /decoded to offer transport and control services over the radio transmission link. Channel coding scheme is a combination of error detection, error correcting, rate matching, interleaving and transport channel or control information mapping onto/splitting from physical channels.

4.12.1 Channel coding

The bit sequence input for a given code block to channel coding is denoted by $c_0, c_1, c_2, c_3, ..., c_{K-1}$, where *K* is the number of bits to encode. After encoding, the bits are denoted by $d_0^{(i)}, d_1^{(i)}, d_2^{(i)}, d_3^{(i)}, ..., d_{D-1}^{(i)}$, where *D* is the number of encoded bits per output stream and *i* indexes the encoder output stream. The relation between c_k and $d_k^{(i)}$ and between *K* and *D* is dependent on the channel coding scheme.

The following channel coding schemes can be applied to TrCHs:

- tail biting convolutional coding;
- turbo coding.

Usage of coding scheme and coding rate for the different types of TrCH is shown in Table 4.4.4. Usage of coding scheme and coding rate for the different control information types is shown in Table 4.4.5.

The values of *D* in connection with each coding scheme:

- tail biting convolutional coding with rate 1/3: $D = K$;
- turbo coding with rate 1/3: $D = K + 4$.

The range for the output stream index *i* is 0, 1 and 2 for both coding schemes.

Table 4.4.4: Usage of channel coding scheme and coding rate for TrCHs

TrCH	Coding scheme	Coding rate
UL-SCH	Turbo coding	1/3
DL-SCH		
PCH		
MCH		
BCH	Tail biting convolutional coding	1/3

Table 4.4.5 Usage of channel coding scheme and coding rate for control information

Control Information	Coding scheme	Coding rate
DCI	Tail biting convolutional coding	1/3
CFI	Block code	1/16
HI	Repetition code	1/3
UCI	Block code	variable
	Tail biting convolutional coding	1/3

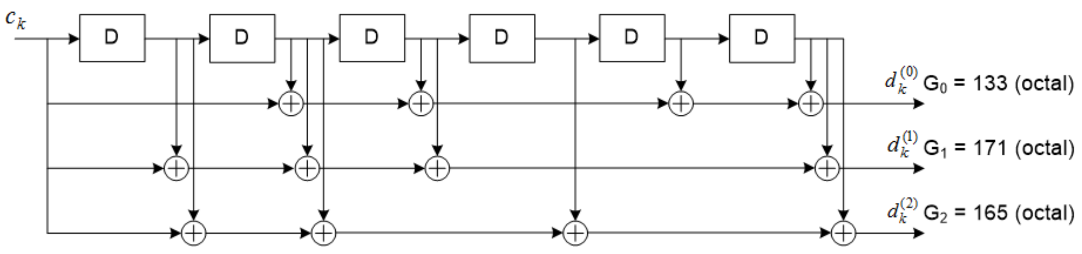
4.12.1.1 Tail biting convolutional coding

A tail biting convolutional code with constraint length 7 and coding rate 1/3 is defined.

The configuration of the convolutional encoder is presented in Figure 4.4.14.

The initial value of the shift register of the encoder shall be set to the values corresponding to the last 6 information bits in the input stream so that the initial and final states of the shift register are the same. Therefore, denoting the shift register of the encoder by $s_0, s_1, s_2, ..., s_5$, then the initial value of the shift register shall be set to

$$s_i = c_{(K-1-i)}$$



c_k
D
D
D
D
D
D
$d_k^{(0)}$ G_0 = 133 (octal)
$d_k^{(1)}$ G_1 = 171 (octal)
$d_k^{(2)}$ G_2 = 165 (octal)

Figure 4.4.14: Rate 1/3 tail biting convolutional encoder

The encoder output streams $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$ correspond to the first, second and third parity streams, respectively as shown in Figure 4.4.14.

4.12.2 Interleaver

Interleaving is the reordering of data that is to be transmitted so that consecutive bytes of data are distributed over a larger sequence of data to reduce the effect of burst errors. The use of interleaving greatly increases the ability of error protection codes to correct for burst errors. Many of the error protection coding processes can correct for small numbers of errors, but cannot correct for errors that occur in groups.

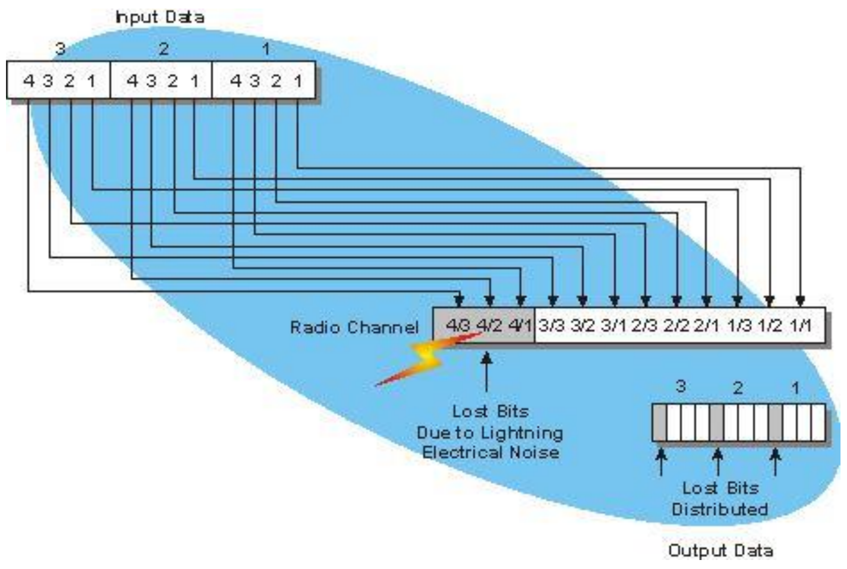


Figure 4.4.15: Interleaving Operation.

This diagram shows that a block of data information may be distributed over multiple time slots or frames in a carrier line to distribute the effect of burst errors on the information signal. In this example, a block of digital audio is being transmitted through a radio channel. The digital audio is divided into blocks of 4 bits and the bits for each block are distributed (interleaved) over a communication channel. During the transmission, a lightning bolt creates a burst of electrical noise that disrupts 3 bits of data transmission. Because these bits are interleaved, the received data has burst errors that are distributed. This allows the audio to be continuously heard with a marginal amount of distortion instead of completely losing the audio during the burst errors. [29]

The inputs to the interleaver are:

1. Data Bits.
2. RI (Rank Information) Control Bits.
3. ACK (Acknowledgement) Control Bits.

The interleaving operation is done as follows:

Step 1: an $(R_{mux} \times C_{mux})$ matrix is constructed (shown in Figure 4.4.16) where:

$$C_{mux} = N_{symb}^{PUSCH} \text{ , and } R_{mux} = (H'' \cdot Q_m) \,/\, C_{mux}$$

($\boldsymbol{H''}$ is the total number of data and RI bits. $\boldsymbol{Q_m}$ is the number of bits per symbol)

Let's denote the inputs to the interleaver by $\underline{g_0}, \underline{g_1}, \underline{g_2}, \dots, \underline{g_{H'-1}}$, $\underline{q_0^{RI}}, \underline{q_1^{RI}}, \underline{q_2^{RI}}, \dots, \underline{q_{Q'_{RI}-1}^{RI}}$, and $\underline{q_0^{ACK}}, \underline{q_1^{ACK}}, \underline{q_2^{ACK}}, \dots, \underline{q_{Q'_{ACK}-1}^{ACK}}$ respectively.

We also define $R'_{mux} = R_{mux}/Q_m$ (it represents the number of rows for the symbols)

Step 2: if rank information is transmitted in this subframe, the vector sequence $\underline{q_0^{RI}}, \underline{q_1^{RI}}, \underline{q_2^{RI}}, \dots, \underline{q_{Q'_{RI}-1}^{RI}}$ is written onto the columns indicated by Table 4.4.6, and by sets of Q_m rows starting from the last row and moving upwards according to the following pseudocode:

```
Set $i, j$ to 0.
Set $r$ to $R'_{mux} - 1$
while $i < Q'_{RI}$
    $c_{RI} = \text{Column Set}(j)$
    $\underline{y}_{r \times C_{mux} + c_{RI}} = \underline{q}_i^{RI}$
    $i = i + 1$
    $r = R'_{mux} - 1 - \lfloor i/4 \rfloor$
    $j = (j + 3) \bmod 4$
end while
```

Where Column Set is given in Table 4.4.6 and indexed left to right from 0 to 3.

Table 4.4.6: Column set for Insertion of rank information.

CP Configuration	Column Set
Normal	$\{1, 4, 7, 10\}$
Extended	$\{0, 3, 5, 8\}$

Table 4.4.7: Column set for Insertion of HARQ-ACK information.

CP Configuration	Column Set
Normal	$\{2, 3, 8, 9\}$
Extended	$\{1, 2, 6, 7\}$

After this step is done, RI bits will be placed in the cells (elements) as shown in Figure 4.4.16. These locations are chosen because **r** starts from the last row and **j** is varying as follows: 0,3,2,1,0,3, ... which results in column numbers 1,10,7,4,1,10, ...

Step 3: write the input vector sequence, for $k = 0,1, \dots, (H' - 1)$, into the matrix by sets of Q_m rows starting with the vector $\underline{y}_0$ in column 0 and row 0 to $(Q_m - 1)$ and skipping the matrix entries that are already occupied:

$$\begin{bmatrix} \underline{y}_0 & \underline{y}_1 & \underline{y}_2 & \cdots & \underline{y}_{C_{mux}-1} \\ \underline{y}_{C_{mux}} & \underline{y}_{C_{mux}+1} & \underline{y}_{C_{mux}+2} & \cdots & \underline{y}_{2C_{mux}-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \underline{y}_{(R'_{mux}-1)\times C_{mux}} & \underline{y}_{(R'_{mux}-1)\times C_{mux}+1} & \underline{y}_{(R'_{mux}-1)\times C_{mux}+2} & \cdots & \underline{y}_{(R'_{mux}\times C_{mux}-1)} \end{bmatrix}$$

The pseudocode is as follows:

```
Set i, k to 0.
    While k < $H'$ ,
      if $\underline{y}_i$ is not assigned to RI symbols
          $\underline{y}_i = \underline{g}_k$
          k = k + 1
      end if
      i = i+1
    end While
```

Step 4: if HARQ-ACK information is transmitted in this subframe, the vector sequence $\underline{q_0^{ACK}}, \underline{q_1^{ACK}}, \underline{q_2^{ACK}}, \ldots, \underline{q_{Q'_{ACK}-1}^{ACK}}$ is written onto the columns indicated by Table 4.4.7, and by sets of Q_m rows starting from the last row and moving upwards according to the following pseudocode. Note that this operation overwrites some of the channel interleaver entries obtained in **Step 3**.

```
Set $i$, $j$ to 0.
Set $r$ to $R'_{mux}-1$
while $i <$ $Q'_{ACK}$
    $c_{ACK} = \text{ColumnSet}(j)$
    $\underline{y}_{r\times C_{mux}+c_{ACK}} = \underline{q}_i^{ACK}$
    $i = i+1$
    $r = R'_{mux}-1-\lfloor i/4 \rfloor$
    $j = (j+3) \bmod 4$
end while
```

Where ColumnSet is given in Table 4.4.7 and indexed left to right from 0 to 3.

Step 5: the output of the block interleaver is the bit sequence read out column by column from the constructed matrix.

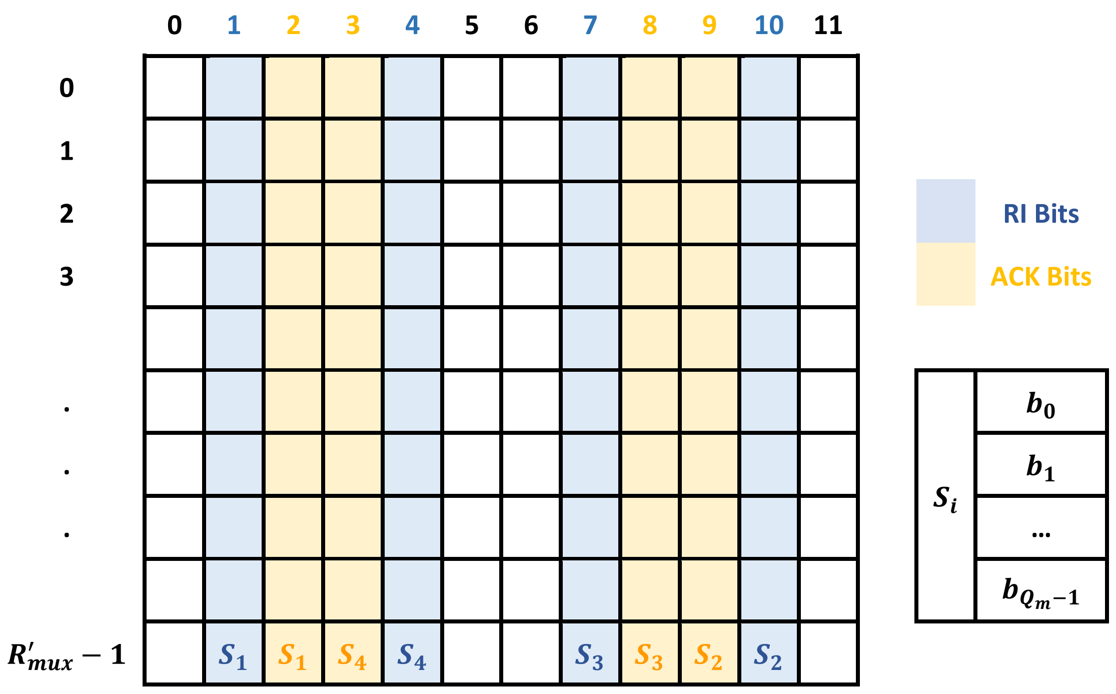


Figure 4.4.16: The $(R'_{mux} \times C_{mux})$ Matrix constructed in step 1.

The yellow columns show the allowed locations for the RI Bits, while the blue locations are for the ACK Bits. This is only valid for the case of Normal CP.

4.13 Physical uplink shared channel bit level processing

The baseband signal representing the physical uplink shared channel is defined in terms of the following steps:

- scrambling
- modulation of scrambled bits to generate complex-valued symbols
- transform precoding to generate complex-valued symbols
- mapping of complex-valued symbols to resource elements
- generation of complex-valued time-domain SC-FDMA signal for each antenna port

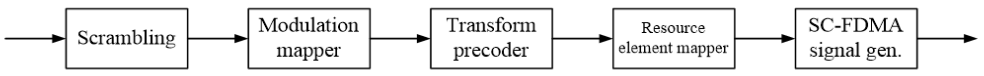


Figure 4.4.17: Overview of uplink physical channel processing.

4.13.1 Scrambler

Scramblers are circuits that pseudo-randomly change the values of bits in a data block or stream with the purpose of "whitening" its spectrum (spread it so that no strong spectral component will exist, thus reducing electromagnetic interference) or to introduce security (as part of an encryption procedure).

The pseudo-randomness is normally accomplished using an LFSR circuit. In this case, a scrambler is just an LFSR plus an additional modulo-2 adder (XOR gate), and it is specified using the LFSR's characteristic polynomial.

The algorithm of bit level is:

For each code word q, the block of bits $b^{(q)}(0)$, $b^{(q)}(1)$,…… $b^{(q)}(M_{bit}^{(q)} – 1)$,

where $M_{bit}^{(q)}$ is the number of bits in code word q transmitted on the physical channel

in one subframe , resulting in a block of scrambled according to

$b'^{(q)}(i) = (b^q(i) + C^q(i)) \bmod 2$

Where, $C^q(i)$ are the pseudo-random sequences defined by a length-31 Gold sequence as

$C(n) = (X_1(n + N_c) + X_2(n + N_c)) \bmod 2$

$X_1(n + 31) = (X_1(n + 3) + X_1(n)) \bmod 2$

$X_2(n + 31) = (X_2(n + 3) + X_2(n + 2) + X_2(n + 1) + X_2(n)) \bmod 2$

Where N_c =1600 and the first m-sequence shall be initialized with $X_1(0) = 1$,

$X_1(n) = 0$, n = 1,2,3,….,30.

$$C_{init} = \sum_{i=10}^{30} X2(i) * 2^i$$

The initialization of the second m-sequence is denoted by C_{init}.

$$C_{init} = n_{RNTI} * 2^{14} + q * 2^{13} + \lfloor ns/2 \rfloor * 2^9 + N_{ID}^{Cell}$$

Where n_{RNTI} corresponds to the identity of the UE(s) to which the PUSCH transmission is intended. Up to 2 code words can be transmitted in one subframe while in present release, only 1 code word (q=0) is supported.

$$C_{init} = n_{RNTI} * 2^{14} + \lfloor ns/2 \rfloor * 2^9 + N_{ID}^{Cell}$$

4.13.2 Modulation Mapper

The block of scrambled bits $\tilde{b}(0),...,\tilde{b}(M_{\text{bit}}-1)$ shall be modulated resulting in a block of complex-valued symbols $d(0),...,d(M_{\text{symb}}-1)$. the modulation mappings applicable for the physical uplink shared channel are.

- Binary PSK (BPSK), using 2 symbols
- Quadrature PSK (QPSK), using 4 symbols
- 16QAM, using 16 symbols
- 64QAM, using 64 symbols

The modulation mapper takes binary digits, 0 or 1, as input and produces complex-valued modulation symbols, $x=I+jQ$, as output.

4.13.2.1 BPSK

In case of BPSK modulation, a single bit, $b(i)$, is mapped to a complex-valued modulation symbol $x=I+\mathrm{j}Q$ according to Table 7.1.1-1 in [30].

4.13.2.2 QPSK

In case of QPSK modulation, pairs of bits, $b(i), b(i+1)$, are mapped to complex-valued modulation symbols $x=I+\mathrm{j}Q$ according to Table 7.1.2-1 in [30].

4.13.2.3 16QAM

In case of 16QAM modulation, quadruplets of bits, $b(i), b(i+1), b(i+2), b(i+3)$, are mapped to complex-valued modulation symbols $x=I+\mathrm{j}Q$ according to Table 7.1.3-1 in [30].

4.13.2.4 64QAM

In case of 64QAM modulation, hextuplets of bits, $b(i), b(i+1), b(i+2), b(i+3), b(i+4), b(i+5)$, are mapped to complex-valued modulation symbols $x=I+\mathrm{j}Q$ according to Table 7.1.4-1 in [30].

4.13.3 Transform precoding

The block of complex-valued symbols $d(0),...,d(M_{\text{symb}}-1)$ is divided into $M_{\text{symb}}/M_{\text{sc}}^{\text{PUSCH}}$ sets, each corresponding to one SC-FDMA symbol. Transform precoding shall be applied according to

$$z(l \cdot M_{\mathrm{sc}}^{\mathrm{PUSCH}} + k) = \frac{1}{\sqrt{M_{\mathrm{sc}}^{\mathrm{PUSCH}}}} \sum_{i=0}^{M_{\mathrm{sc}}^{\mathrm{PUSCH}}-1} d(l \cdot M_{\mathrm{sc}}^{\mathrm{PUSCH}} + i) e^{-j\frac{2\pi ik}{M_{\mathrm{sc}}^{\mathrm{PUSCH}}}}$$
$$k = 0, \ldots, M_{\mathrm{sc}}^{\mathrm{PUSCH}} - 1$$
$$l = 0, \ldots, M_{\mathrm{symb}} / M_{\mathrm{sc}}^{\mathrm{PUSCH}} - 1$$

resulting in a block of complex-valued symbols $z(0),\ldots,z(M_{\mathrm{symb}}-1)$. The variable $M_{\mathrm{sc}}^{\mathrm{PUSCH}} = M_{\mathrm{RB}}^{\mathrm{PUSCH}} \cdot N_{\mathrm{sc}}^{\mathrm{RB}}$, where $M_{\mathrm{RB}}^{\mathrm{PUSCH}}$ represents the bandwidth of the PUSCH in terms of resource blocks, and shall fulfil

$$M_{\mathrm{RB}}^{\mathrm{PUSCH}} = 2^{\alpha_2} \cdot 3^{\alpha_3} \cdot 5^{\alpha_5} \le N_{\mathrm{RB}}^{\mathrm{UL}}$$

where $\alpha_2, \alpha_3, \alpha_5$ is a set of non-negative integers.

4.13.4 Mapping to physical resources

The block of complex-valued symbols $z(0),\ldots,z(M_{\mathrm{symb}}-1)$ shall be multiplied with the amplitude scaling factor β_{PUSCH} in order to conform to the transmit power P_{PUSCH} and mapped in sequence starting with $z(0)$ to physical resource blocks assigned for transmission of PUSCH. The mapping to resource elements (k,l) corresponding to the physical resource blocks assigned for transmission and not used for transmission of reference signals and not reserved for possible SRS transmission shall be in increasing order of first the index k, then the index l, starting with the first slot in the subframe.

If uplink frequency-hopping is disabled, the set of physical resource blocks to be used for transmission are given by $n_{\mathrm{PRB}} = n_{\mathrm{VRB}}$ where n_{VRB} is obtained from the uplink scheduling grant as described in Section 8.1 in [31].

For simplicity we assumed that uplink frequency-hopping is disabled.

4.13.5 SC-FDMA baseband signal generation

This section applies to all uplink physical signals and physical channels except the physical random access channel.

The time-continuous signal $s_l(t)$ in SC-FDMA symbol l in an uplink slot is defined by

$$s_l(t) = \sum_{k=-\lfloor N_{\mathrm{RB}}^{\mathrm{UL}} N_{\mathrm{sc}}^{\mathrm{RB}}/2 \rfloor}^{\lceil N_{\mathrm{RB}}^{\mathrm{UL}} N_{\mathrm{sc}}^{\mathrm{RB}}/2 \rceil -1} a_{k^{(-)},l} \cdot e^{j2\pi(k+1/2)\Delta f(t-N_{\mathrm{CP},l}T_s)}$$

for $0 \le t < (N_{\mathrm{CP},l} + N) \times T_s$ where $k^{(-)} = k + \lfloor N_{RB}^{UL} N_{sc}^{RB}/2 \rfloor$, $N = 2048$, $\Delta f = 15\ \mathrm{kHz}$ and $a_{k,l}$ is the content of resource element (k,l).

The SC-FDMA symbols in a slot shall be transmitted in increasing order of l, starting with $l = 0$, where SC-FDMA symbol $l > 0$ starts at time $\sum_{l'=0}^{l-1}(N_{\mathrm{CP},l'} + N)T_s$ within the slot.

Table 4.4.8 lists the values of $N_{\mathrm{CP},l}$ that shall be used. Note that different SC-FDMA symbols within a slot may have different cyclic prefix lengths.

Table 4.4.8: SC-FDMA parameters.

Configuration	Cyclic prefix length $N_{CP,l}$
Normal cyclic prefix	160 for $l = 0$ 144 for $l = 1,2,...,6$
Extended cyclic prefix	512 for $l = 0,1,...,5$

4.14 Decoding of convolutional encoding using Viterbi algorithm

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the Viterbi algorithm is universally used as it provides maximum likelihood performance and is highly parallelizable. Viterbi decoders are thus easy to implement in software on CPUs with SIMD instruction sets.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the “Fano algorithm” is the best known.

The Viterbi algorithm is very suitable for the requirements of the project for its parallelization capability.

4.14.1 Viterbi overview:

The Viterbi algorithm is best illustrated using the trellis diagram. The trellis diagram is just a way to show the transition from one state to another state when the input to the convolutional encoder during time evolution. First, we will show the trellis diagram when encoding the bits.

The trellis diagram is shown in Figure 4.4.18:

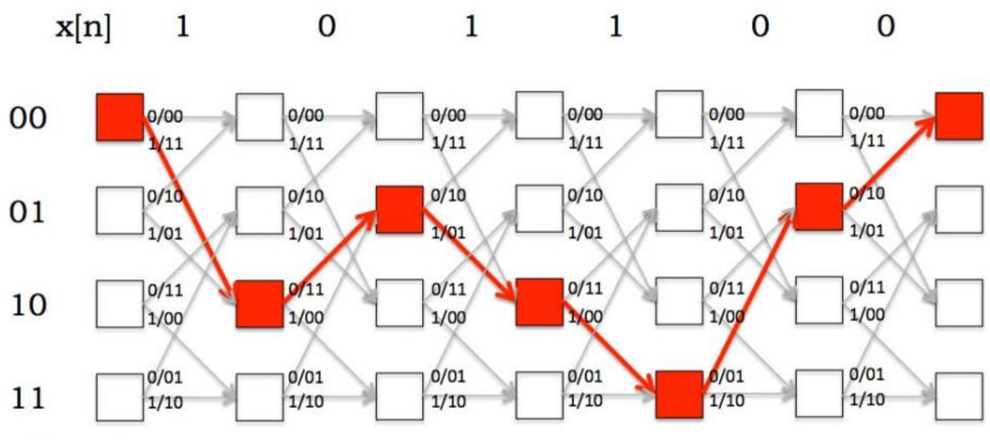


Figure 4.4.18: A trellis diagram showing the time evolution of the state machine.

In Figure 4.4.18 the x[n] represents the input to the convolutional encoder at each time clock.

The trellis diagram above corresponds to a convolutional encoder with 2 shift registers and coding rate = $\frac{1}{2}$

Each box represents one of the states for the shift registers. Therefore, we have 4 states which mean 4 vertical boxes. When the input to the convolutional encoder is “1’, the state in the 1st stage will change from “00” to “10”. The input then completes forming a path and emitting a codeword of 2 bits at each transition.

At the receiver, suppose we now receive a sequence of bits representing the output from the encoder. Then, there will be some path through the trellis that would exactly match the received sequence. But, when there are bit errors, we need to find the *most likely* transmitted message sequence. If we can come up with a way to capture the errors introduced by going from one state to the next, then we can accumulate the errors along a path and come up with an estimate of the total number of errors along the path. Then, the path with the smallest such accumulation of errors is the path we want and it represents the most likely transmitted message.

The Viterbi decoder solves these problems by computing the cost (errors) when going from one state to another. Then, the costs are accumulated and the path with the least cost represents the most likely transmitted message.

4.14.2 Viterbi algorithm Parameters:

The decoding algorithm has two metrics: the branch metric (BM) and the path metric (PM). The branch metric is a measure of the "distance" between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the *Hamming distance* between the expected parity bits and the received ones.

Path metric is defined for hard decision and it corresponds to the Hamming distance with respect to the received parity bit sequence over the most likely path from the initial state to the current state in the trellis. By “most likely”, we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states.

4.14.3 Steps of Viterbi algorithm:

1- Calculating the branch metric for all arcs in the trellis.

   As stated previously we will calculate the hamming distance between both the received sequence and all possible values coming out from the Viterbi decoder at all states.

2- ACS calculation (**A**dd, **C**ompare, and **S**hift).

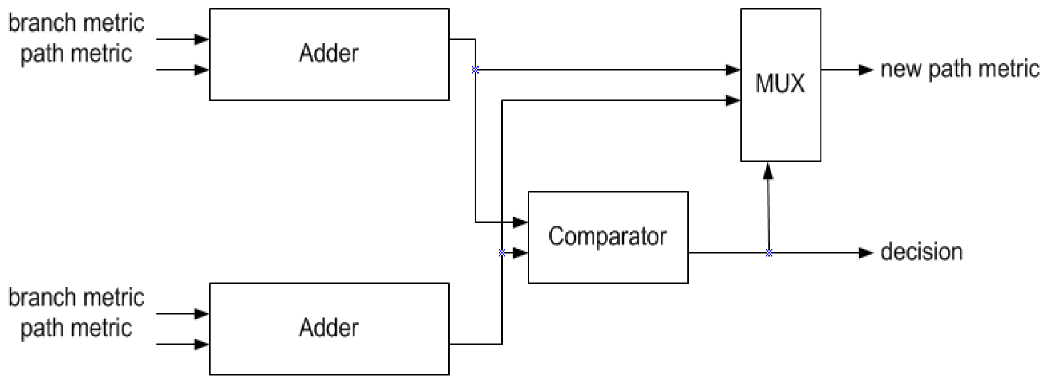


Figure 4.4.19: Block diagram of ACS.

As shown in Figure 4.4.19 we can sum up the ACS algorithm in each stage in the trellis as follows:

Add the predecessor state path metric to the current state branch metric for both predecessor states. **Compare** between the cost of the two addition results. By "cost", we mean the Hamming distance. **Select** the minimum cost to be the new path metric for the current state which will be predecessor in the next iteration.

Mathematically, at any state 'S' at time 'N':

$$PM[S,N] = \min(PM[\alpha,N-1] + BM[\alpha,N-1], PM[\beta,N-1] + BM[\beta,N-1])$$

PM[S, N]: path metric at state S at time N.

α and β are the two predecessor states.

3- Finding the most likely path.

After all data have been processed, we must find the most likely path through the trellis done by searching through the final “column” in PM and selecting the most likely (lowest cost) operation done infrequently compared to other calculations.

4- Traceback and Estimating the output.

In the 4th step we will find the total path from the final minimum cost state that we have already found it in the 3rd step to the first state.

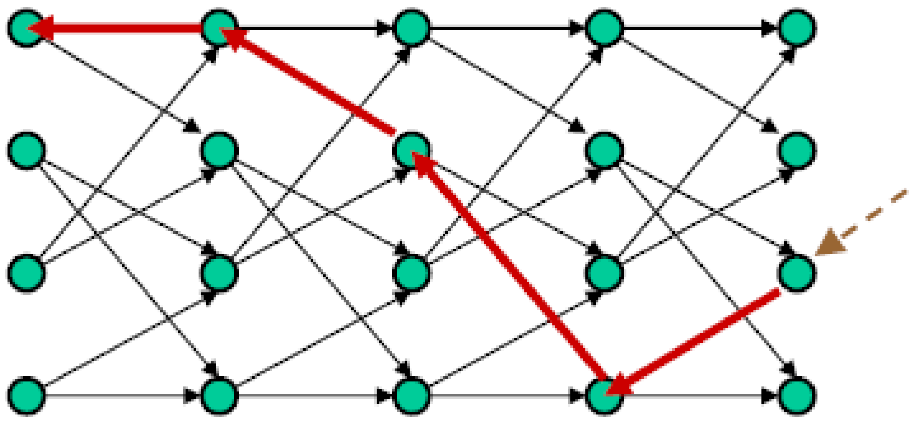


Figure 4.4.20: Traceback on trellis.

We can perform traceback by:

- Read the Path metric of the both predecessor states
- Select the minimum and add the new backward state into our traceback path.
- By comparing both the previous state and the current state, we can guess the input bit.

After that, we can reorder both the output bits and the traceback states.

Chapter 5: Channel

The communication channel provides the connection between the transmitter and the receiver. The physical channel may be a pair of wires that carry the electrical signal, or an optical fiber that carries the information on a modulated light beam, or an underwater ocean channel in which the information is transmitted acoustically, or free space over which the information-bearing signal is radiated by use of an antenna.

One common problem in signal transmission through any channel is additive noise and interference which may arise externally to the system, such as interference from other users of the channel. When such noise and interference occupy the same frequency band as the desired signal, its effect can be minimized by proper design of the transmitted signal and its demodulator at the receiver.

The effects of noise may be minimized by increasing the power in the transmitted signal. However, equipment and other practical constraints limit the power level in the transmitted signal. Another basic limitation is the available channel bandwidth. A bandwidth constraint is usually due to the physical limitations of the medium and the electronic components used to implement the transmitter and the receiver. These two limitations result in constraining the amount of data that can be transmitted reliably over any communications channel. Shannon's basic results relate the channel capacity to the available transmitted power and channel bandwidth as shown in the following equation [32]:

$$C = B.log_2(1 + SNR) \tag{5.1}$$

5.1 Additive White Gaussian Noise (AWGN)

To simulate the effect of the noise component, AWGN model is used. Its name didn’t come to existence by coincidence, however due to the following justifications:

- Additive: as the noise component is added into the received signal at the receiver side.
- White: as it has uniform power across the entire frequency band.
- Gaussian: as it has normal distribution in time domain with zero average value.

The noise effect presents due to various reasons (e.g. the thermal noise by the virtue of electrons movement in the electronic circuit being used for transmission and reception of the signal).

5.2 Channel fading

Basic Electromagnetic (EM) wave Common propagation phenomena that affect the channel between the transmitter and the receiver are reflection, diffraction, and scattering.

But the main problem in the wireless channel that needs to be mitigated is the channel fading.

In wireless communications, fading is variation of the attenuation of a signal with various variables. These variables include time, geographical position, and radio frequency. Fading is often modeled as a random process. A fading channel is a communication channel that experiences fading. In wireless systems, fading may either be due to multipath propagation, referred to as multipath induced fading, weather (particularly rain), or shadowing from obstacles affecting the wave propagation, sometimes referred to as shadow fading. [33]

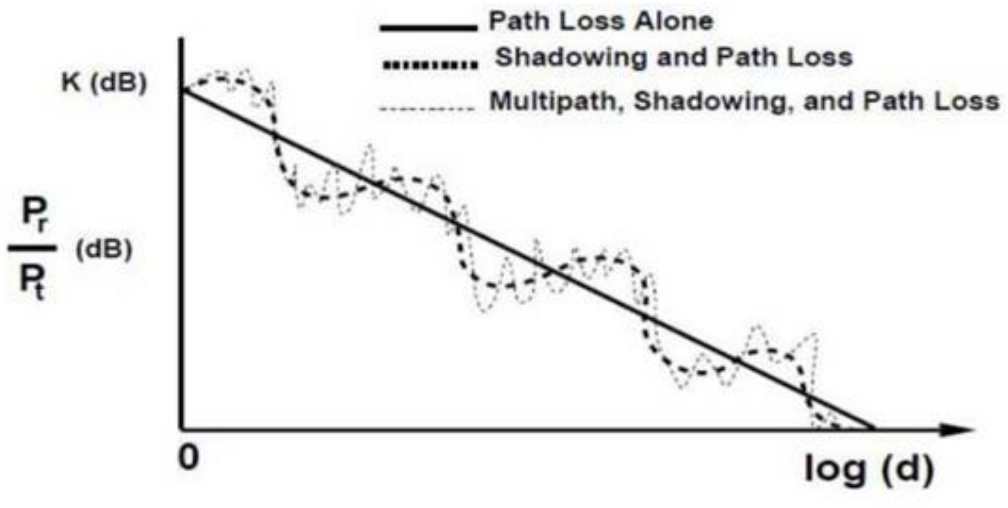


Figure 5.1: main sources of wireless channel fading

As shown in Figure 5.1, there are three main sources of fading:

- Large scale fading or path-loss.
- Medium scale fading or shadowing.
- Small scale fading or multipath effect.

5.2.1 Large scale fading (path-loss)

Path-loss is a reduction of the power density of the transmitted EM signal due to different effects, such as:

- FSPL (Free Space Path Loss) due to the distance between the transmitter and the receiver passing through the medium.

$$FSPL = \left(\frac{4\pi df}{c}\right)^2 \tag{5.2}$$

- Refraction if there are different media in the propagation path.
- Reflection.
- Diffraction losses when part of the radio-wave front is obstructed by an opaque obstacle.

Equation (5.2) mentioned above was for free-space medium. For general medium

$$PL(d)\alpha\left(\frac{d}{d_0}\right)^n \tag{5.3}$$

As shown in equation (5.3), the path-loss exponent n is a variable that depends on the medium:

Table 5.1: Path-loss exponents for different environments

Environment	Path-loss exponent, n
Free space	2
Urban area cellular radio	2.7 to 3.5
Shadowed urban cellular radio	3 to 5
In building line-of-sight	1.6 to 1.8
Obstructed in building	4 to 6
Obstructed in factories	2 to 3

5.2.2 Medium scale fading (shadowing)

The reasons of shadowing are not so far from that of the large scale fading. It may be considered as the average of the small scale fading as shown in Figure 5.1.

5.2.3 Small scale fading (multi-path fading)

Small scale fading is a characteristic of radio propagation resulting from the presence reflectors and scatters that cause multiple versions of the transmitted signal to arrive at the receiver, each distorted in amplitude, phase and angle of arrival. The superposition of all of these versions produces a distorted received signal.

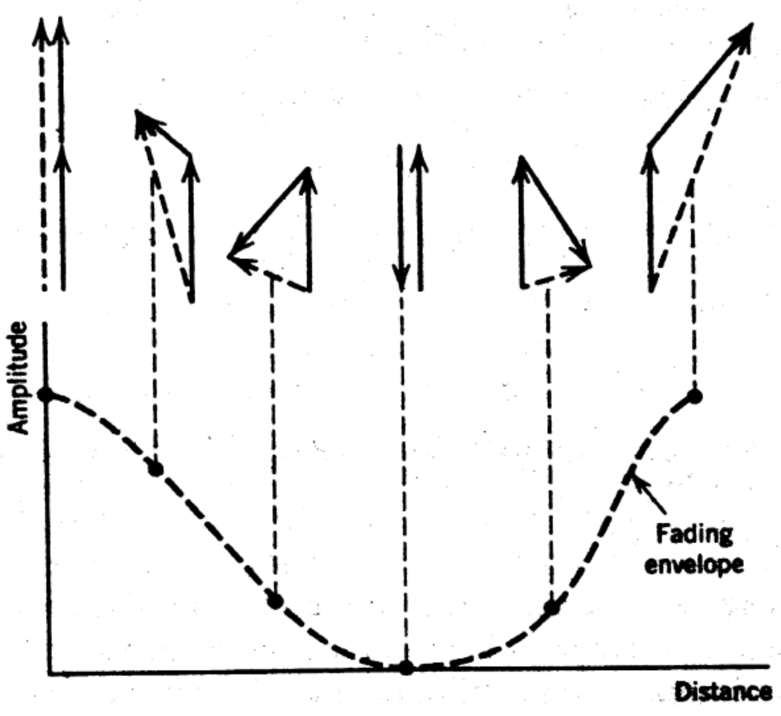


Figure 5.2: Illustrate how the envelop fades as two incoming signals combine with different phase

The small scale fading affects the communication link between transmitters and receivers, by the following:

- Rapid changes in signal strength over a small travel distance or time interval.
- Time dispersion (echoes) caused by multipath propagation.
- Random frequency modulation due to varying Doppler shifts.

The above effects can be reduced or mitigated by tuning the following factors:

- Speed of mobile and speed of surrounding objects (that affect Doppler shift).
- Signal Transmission BW.

5.2.4 Types of fading

Channels maybe classified using different concepts but with the concept introduced in the previous subsection of fading, channels can be:

- Flat fading channel
- Frequency selective
- Slow fading
- Fast fading

A channel is said to be a flat fading channel if the channel is flat over the transmitting BW or the symbol time is greater than the delay spread of the channel. Otherwise the channel is said to be a frequency selective.

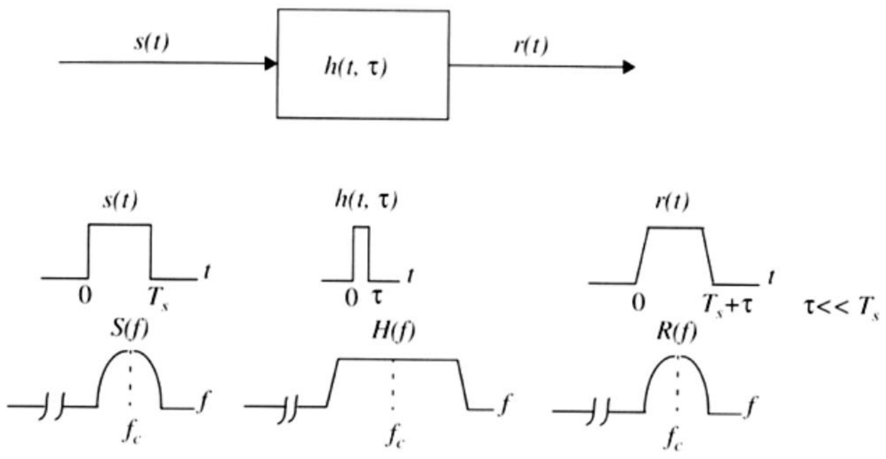


Figure 5.3: Flat fading channel

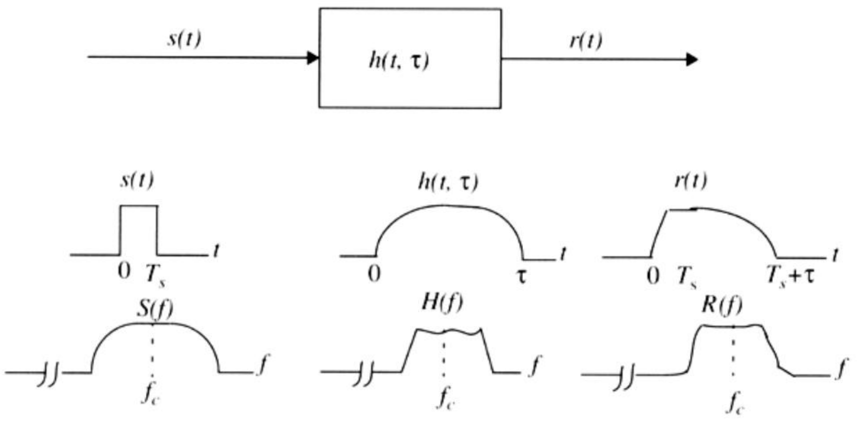


Figure 5.4: Frequency selective channel

A channel is said to be a slow fading channel if the channel impulse response changes at a rate much slower than the transmitter baseband signal. Otherwise the channel is said to be a fast fading channel.

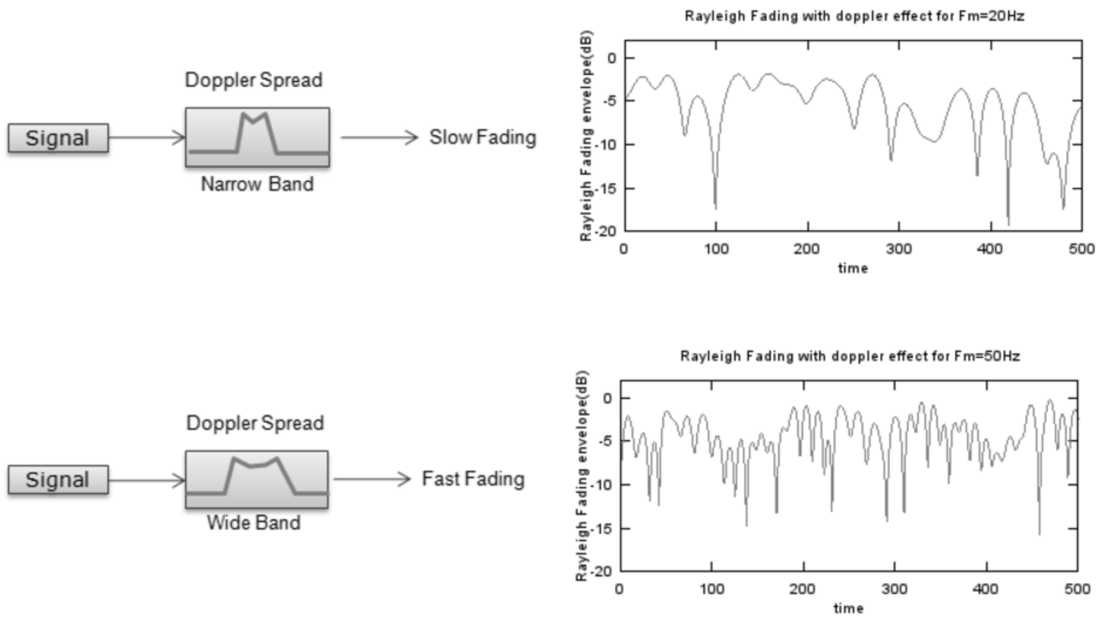


Figure 5.5: Slow and Fast fading channels

5.3 Channel estimation

The pilot symbols in LTE are assigned positions within a sub-frame depending on the eNodeB cell identification number and which transmit antenna is being used, as shown in the following figure:

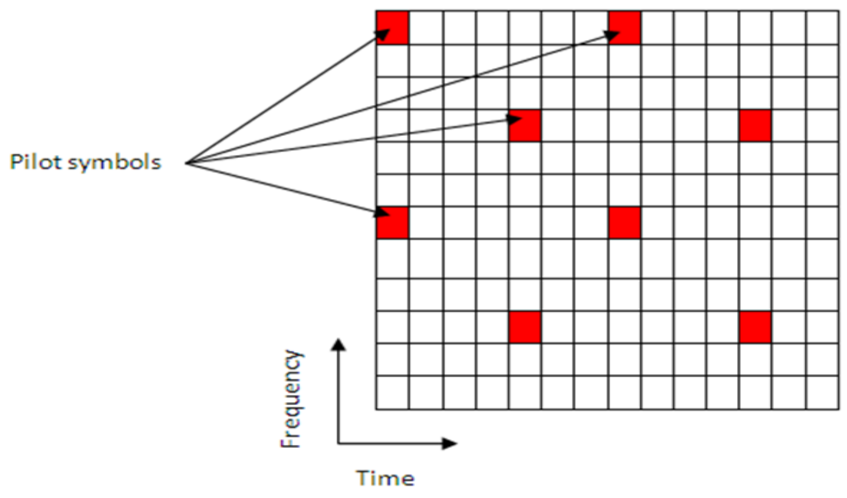


Figure 5.6: Pilot position

The unique positioning of the pilots ensures that they do not interfere with one another and can be used to provide a reliable estimate of the complex gains imparted onto each resource element within the transmitted grid by the propagation channel. [34]

Both transmit and receive chains and the propagation channel model are shown in the block diagram (Figure 5.7).

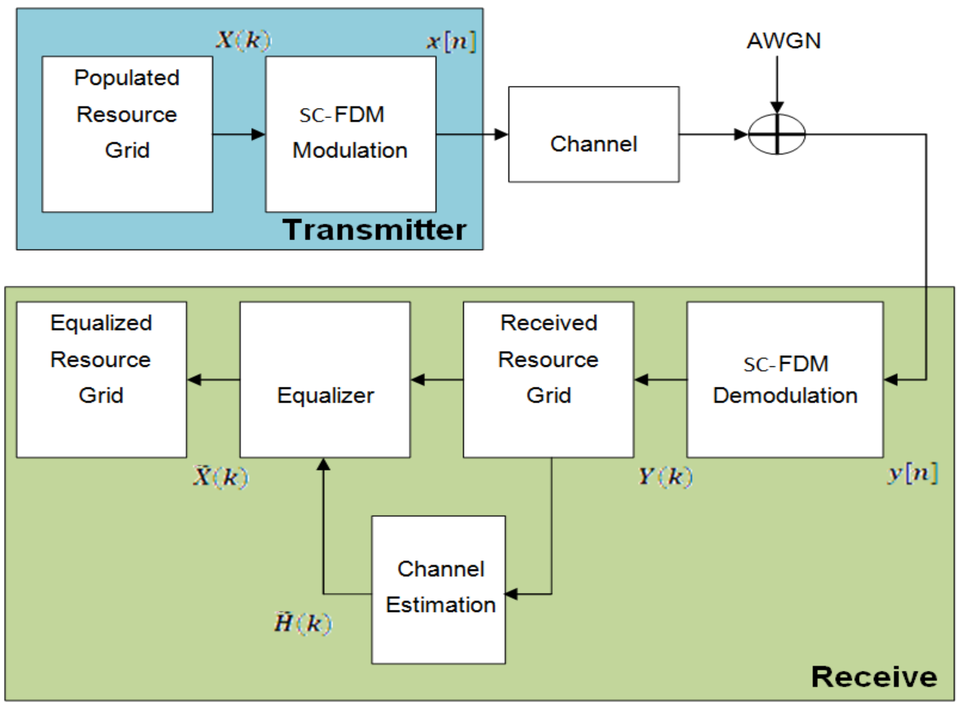


Figure 5.7: Both transmitter and receiver chain.

The populated resource grid represents several sub-frames containing data. This grid is then SC-FDM modulated and passed through the model of the propagation channel. Channel noise in the form of additive white Gaussian noise (AWGN) is added before the signal enters the receiver. Once inside the receiver the signal is SC-OFDM demodulated and a received resource grid can be constructed. The received resource grid contains the transmitted resource elements which have been affected by the complex channel gains and the channel noise. Using the known pilot symbols to estimate the channel, it is possible to equalize the effects of the channel and reduce the noise on the received resource grid.

LTE assigns each antenna port a unique set of locations within a subframe to which to map reference signals. Because no other antenna transmits data at these locations in time and frequency, channel estimation for multi-antenna configurations can be performed. The channel estimation algorithm extracts the reference signals for a transmit/receive antenna pair from the received grid. The least squares estimates of the channel frequency response at the pilot symbols are calculated as described in On Channel Estimation in SC-OFDM Systems.

The least squares estimates are then averaged to reduce any unwanted noise from the pilot symbols. Because it is possible that no pilots are located near the sub-frame edge, virtual pilot symbols are created to aid the interpolation process near the edge of the sub-frame. Using the averaged pilot symbol estimates and the calculated virtual pilot symbols, interpolation is then carried out to estimate the entire sub-frame. This process is demonstrated in the following block diagram.

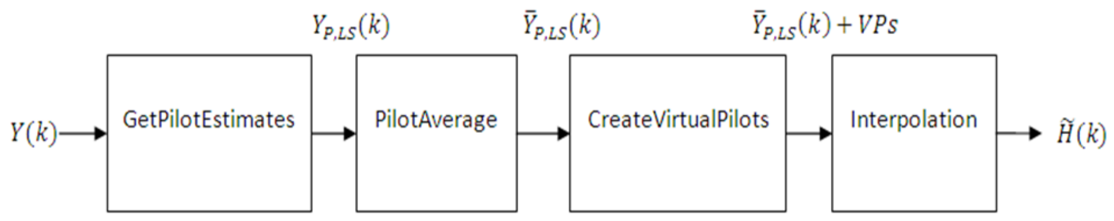


Figure 5.8: Channel estimation block diagram

5.3.1 Get Pilot Estimates subsystem

The first step in determining the least squares estimate is to extract the pilot symbols from their known location within the received sub-frame. Because the value of these pilot symbols is known, the channel response at these locations can be determined using the least squares estimate. The least squares estimate is obtained by dividing the received pilot symbols by their expected value.

$$Y(k) = H(k).X(k) + noise \tag{5.4}$$

where

- $Y(k)$ is a received complex symbol value;
- $X(k)$ is a transmitted complex symbol value;
- $H(k)$ is a complex channel gain experienced by a symbol.

Known pilot symbols can be sent to estimate the channel for a subset of REs within a subframe. In particular, if pilot symbol $X_P(k)$ is sent in an RE, an instantaneous channel estimate $H_{PI}(k)$ for that RE can be computed using:

$$H_{pI}(k) = \frac{Y_p(k)}{X_p(k)} = H_p(k) + noise \tag{5.5}$$

where

- $Y_P(k)$ represents the received pilot symbol values;
- $X_P(k)$ represents the known transmitted pilot symbol values;
- $H_P(k)$ is the true channel response for the RE occupied by the pilot symbol.

5.3.2 Pilot Average subsystem

To minimize the effects of noise on the channel estimates, the least square estimates are averaged using an averaging window. This simple method produces a substantial reduction in the level of noise found on the pilot REs.

The Averaging method uses the approach described in TS 36.141 [35], Annex F.3.4. Time averaging is performed across each subcarrier that contains a pilot symbol, resulting in a column vector containing an average amplitude and phase for each subcarrier that is carrying a reference signal.

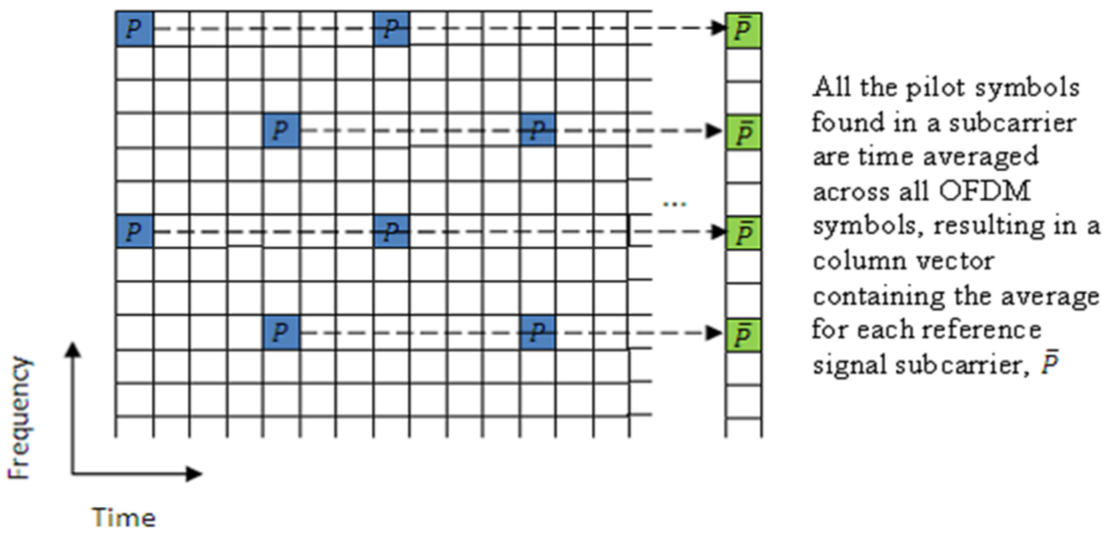

P
P
$\bar{P}$
Frequency
Time
...
All the pilot symbols found in a subcarrier are time averaged across all OFDM symbols, resulting in a column vector containing the average for each reference signal subcarrier, $\bar{P}$

Figure 5.9: Time average of the received pilots.

The averages of the pilot symbol subcarriers are then frequency averaged using a moving window of maximum size 19.

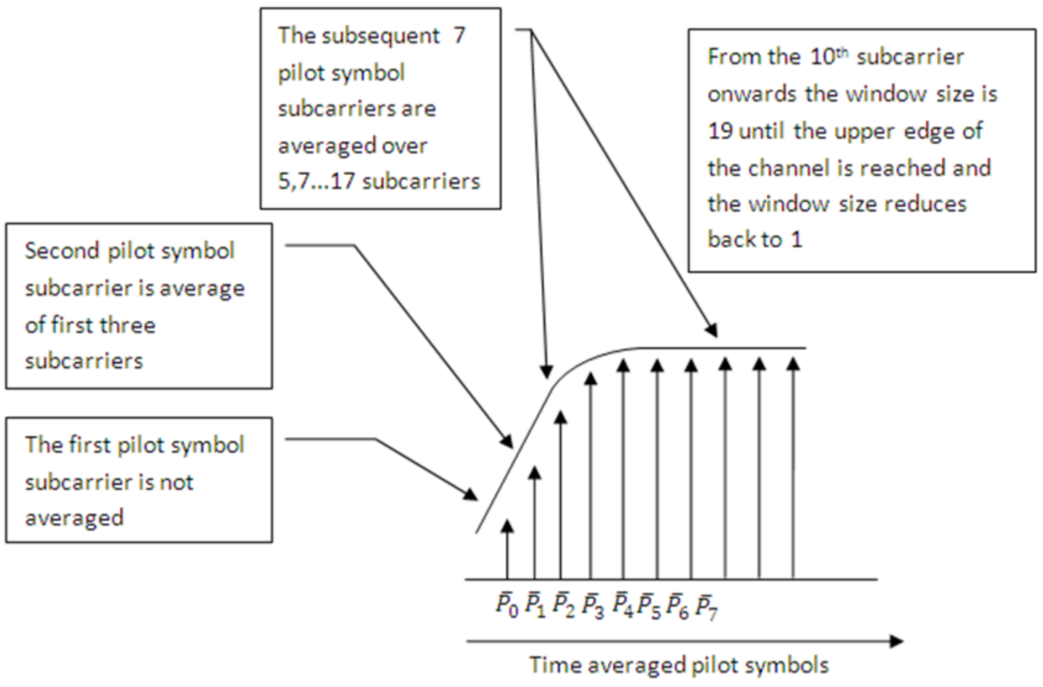

The subsequent 7 pilot symbol subcarriers are averaged over 5,7...17 subcarriers
From the 10th subcarrier onwards the window size is 19 until the upper edge of the channel is reached and the window size reduces back to 1
Second pilot symbol subcarrier is average of first three subcarriers
The first pilot symbol subcarrier is not averaged
$\bar{P}_0$ $\bar{P}_1$ $\bar{P}_2$ $\bar{P}_3$ $\bar{P}_4$ $\bar{P}_5$ $\bar{P}_6$ $\bar{P}_7$
Time averaged pilot symbols

Figure 5.10: Frequency averaging of time averaged received pilots.

5.3.3 Create Virtual Pilot System

In many instances, edges of the resource grid do not contain any pilot symbols. This effect is shown in the resource grid in the following figure.

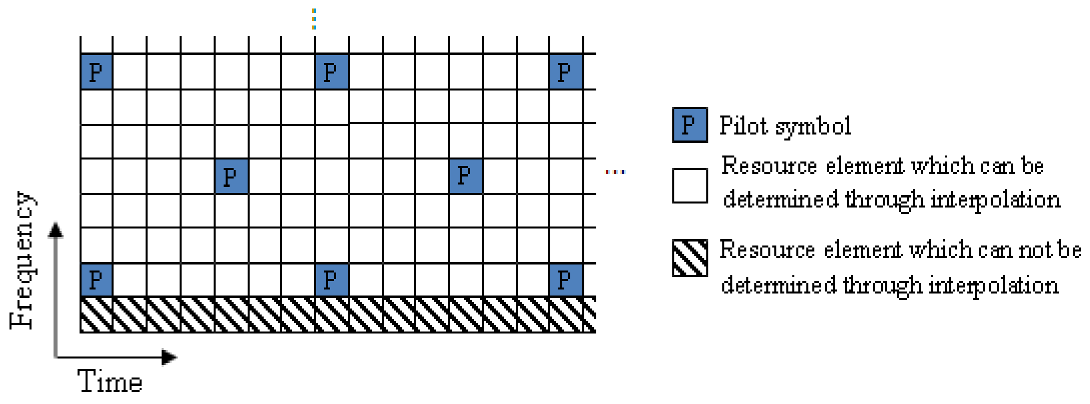
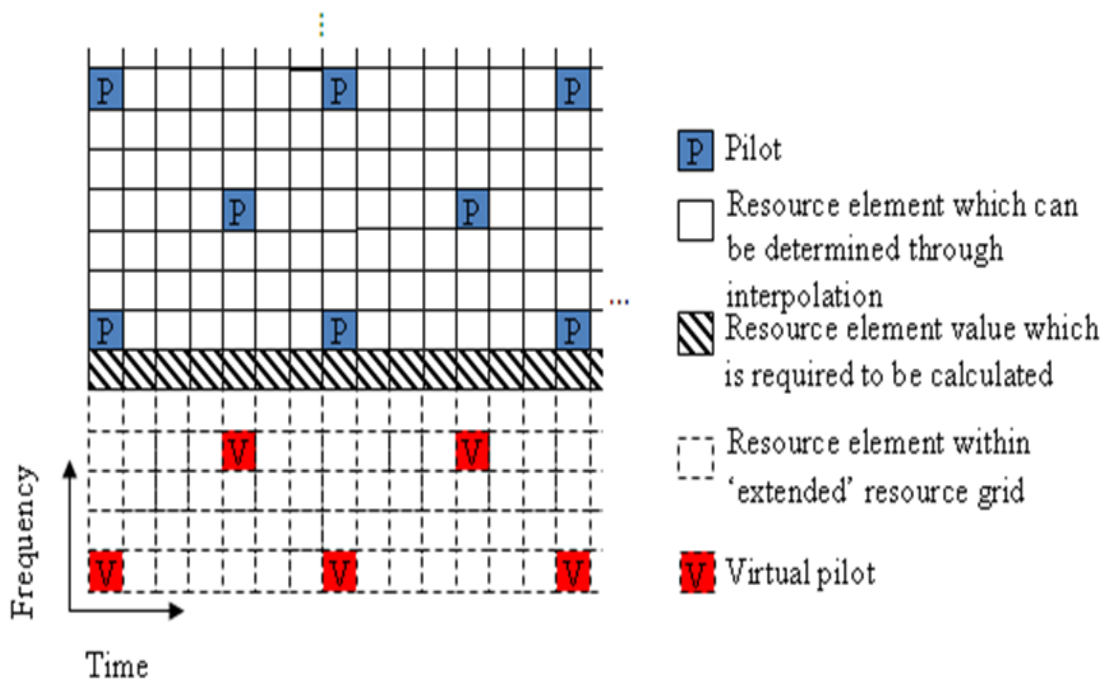

P
Pilot symbol
Resource element which can be determined through interpolation
Resource element which can not be determined through interpolation
Frequency
Time

Figure 5.11: Symbol at Edges.

In this case, channel estimates at the edges cannot be interpolated from the pilot symbols. To overcome this problem, virtual pilot symbols are created. Virtual pilot symbols are created as shown in the following figure:



P
V
Pilot
Resource element which can be determined through interpolation
Resource element value which is required to be calculated
Resource element within 'extended' resource grid
Virtual pilot
Frequency
Time

Figure 5.12: Virtual pilot placement

In this system, the resource grid is extended, with virtual pilot symbols created in locations which follow the original reference signal pattern. The presence of virtual pilot symbols allows the channel estimate at the resource elements, which previously could not be calculated by interpolation, to be calculated by interpolation using the original and virtual pilot symbols.

The virtual pilot symbols are calculated using the original pilot symbols. For each virtual pilot symbol, the value is calculated following these steps:

The closest 10 ordinary pilots in terms of Euclidian distance in time and frequency are selected. The search is optimized to consider 10 these pilots, rather than checking all possible pilots. Based on the possible configurations of the cell RS, using 10 pilots provides sufficient time and frequency diversity in the pilots for the virtual pilot calculation.

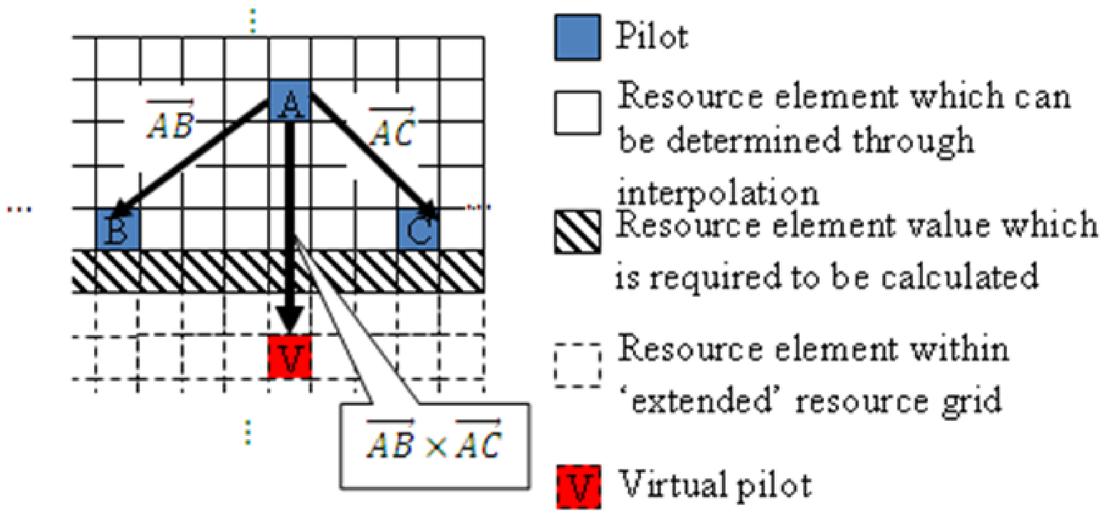
Using this set of the 10 pilots, the closest three pilot symbols are selected. These three symbols must occupy at least two unique subcarriers and two unique OFDM symbols.

Using this set of three pilots, two vectors are created. One vector between the closest and furthest pilot symbols, and one vector between the second closest and furthest pilot symbols.

The cross-product of these two vectors is calculated to create a plane on which the three points reside.

The plane is extended to the position of the virtual pilot to calculate the value based on one of the actual pilot values.

This diagram shows the virtual pilot calculation.



A
B
C
V
$\overrightarrow{AB}$
$\overrightarrow{AC}$
$\overrightarrow{AB} \times \overrightarrow{AC}$
Pilot
Resource element which can be determined through interpolation
Resource element value which is required to be calculated
Resource element within 'extended' resource grid
Virtual pilot

Figure 5.13: Virtual pilot calculation

Linear interpolation is used in our implemented estimator so no virtual pilots are used

5.3.4 Interpolation Subsystem

Once the noise has been reduced or removed from the least squares pilot symbol averages and sufficient virtual pilots have been determined, it is possible to use interpolation to estimate the missing values from the channel estimation grid.

The pilot averaging method described in TS 36.141 [34], Annex F.3.4, requires the use of simple linear interpolation on the time-averaged and frequency-averaged column vector. The interpolation is one-dimensional, since it only estimates the values between the averaged pilot symbol subcarriers in the column vector. The resulting vector is then replicated and used as the channel estimate for the entire resource grid.

5.4 Noise Estimation

The performance of some receivers can be improved through knowledge of the noise power present on the received signal. The noise power can be determined by analyzing the noisy least squares estimates and the noise averaged estimates.

The noisy least-squares estimates from the Get Pilot Estimates Subsystem and the noise averaged pilot symbol estimates from the Pilot Average Subsystem provide an indication of the channel noise. The least-squares estimates and the averaged estimates contain the same data, apart from additive noise. Simply taking the difference between the two estimates results in a noise level value for the least squares channel estimates at pilot symbol locations.

Averaging the instantaneous channel estimates over the smoothing window, we have

$$H_{pI}^{Avg}(k) = \frac{1}{|S|} \sum_{m \epsilon S} H_{PI}(m) \approx H_P(k) \tag{5.6}$$

Where *S* is the set of pilots in the smoothing window and $|S|$ is the number of pilots in *S*. Thus, an estimate of the noise at a particular pilot RE can be formed using:

$$noise_{estimated} = H_{PI}(m) - H_{PI}^{Avg}(k) \tag{5.7}$$

In practice, it is not possible to remove all the noise using averaging. Because it is only possible to reduce the noise, only an estimate of the noise power can be made.

Using the value of the noise power found in the channel response at pilot symbol locations, the noise power per resource element (RE) can be calculated by taking the variance of the resulting noise vector. The noise power per RE for each transmit and receive antenna pair is calculated and stored. The mean of this matrix is returned as the estimate of the noise power per RE.

5.5 Channel Equalization

5.5.1 Zero forcing

In this method, the ISI component at the output of equalizer is forced to zero by using appropriate linear time invariant filter having suitable transfer function.

If transmitted symbol is represented by x_1 and x_2, h_{11} represent the channel from first transmitter to first receiver, h_{12} represent the channel from second transmitter to first receiver, h_{21} represent the channel from first transmitter to second receiver and h_{22} represent the channel from second transmitter to second receiver and n_1, n_2 represent noise on first and second receiver then the received symbol on first receiver is given by:

$$y1 = h_{11}x_1 + h_{12}x_2 + n_1 = (h_{11}\ h_{12})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + n_1 \tag{5.8}$$

And the received symbol on second receiver is given by:

$$y2 = h_{22}x_2 + h_{21}x_1 + n_2 = (h_{22}\ h_{21})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + n_2 \tag{5.9}$$

These two above equation can also be written as:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \tag{5.10}$$

From here it is clear that in order to find x from above equation, we need to find out the matrix which is inverse of matrix H.

If W represents the inverse of H then it must satisfy the property

$$WH = I \tag{5.11}$$

Where I is the identity matrix.

The matrix W which satisfies the above mentioned property is known as the zero forcing linear detector, and is computed by following equation:

$$W = (H^H H)^{-1} H^H \tag{5.12}$$

Where $H^H H$ is calculated using the following:

$$H^H H = \begin{pmatrix} h_{11}^* & h_{12}^* \\ h_{21}^* & h_{22}^* \end{pmatrix}\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \tag{5.13}$$

From this matrix it is clear that the off diagonal term is non-zero and hence zero forcing equalizer cancel out the interference signal. It is reasonably simple and easy to implement but its main drawback is that it tends to amplify the noise and hence gives noisy output.

5.5.2 MMSE Equalizer (Minimum Mean Square Error)

This type of equalizer uses the squared error as performance measurement. The receiver filter is designed to fulfill the minimum mean square error criterion. Main objective of this method is to minimize the error between target signal and output obtained by filter. The computation for this method is as follows.

- Computes the coefficient of matrix W using MMSE algorithm which minimize the condition:

$$E\{[W_y - x][W_y - x]^H\} \tag{5.14}$$

- Solving the above equation gives

$$W = \ (H^H H + \ N_0 I)^{-1} H^H \tag{5.15}$$

- From that equation, it is clear that this equation is different from the equation of zero forcing equalizer by the term $N_o I$. If we put $N_o I$=0 in this equation, then MMSE equalizer becomes zero forcing equalizer.

5.6 Channel models

The multipath fading channel model specifies the following three delay profiles.

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

These three delay profiles represent a low, medium, and high delay spread environment, respectively. The multipath delay profiles for these channels are shown in the following tables.

Table 5.2: 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

Table 5.3: EVA Delay Profile

Excess Tap delay (ns)	Relative Power (dB)
0	0.0
30	-1.5
150	-1.4
310	-3.6
370	-0.6
710	-9.1
1090	-7.0

Table 5.4: ETU Delay Profile

Excess Tap delay (ns)	Relative Power (dB)
0	-1.0
50	-1.0
120	-1.0
200	-0.0
230	-0.0
500	-0.0
1600	-3.0
2300	-5.0
5000	-7.0

All the taps in the preceding tables have a classical *Doppler* spectrum. In addition to multipath delay profile, a maximum Doppler frequency is specified for each multipath fading propagation condition, as shown in the following table.

Table 5.5: Maximum Doppler frequency for different fading propagation.

Channel model	Maximum Doppler frequency
EPA 5Hz	5Hz
EVA 5Hz	5Hz
EVA 70Hz	70Hz
ETU 70Hz	70Hz
ETU 300Hz	300Hz

Chapter 6: Project Flow

6.1 Project milestones

1. Learning Steps:
   - LTE Fundamentals through Rohde and Schwarz documents.
   - 3GPP standards (releases 8, 9 and 10) for LTE (PUSCH).
   - Parallel Programming Course using CUDA Language (Udacity online course).
   - AVX instruction (Supported in Intel® Core™ i7 Processors).
2. Implementing PUSCH using MATLAB LTE System Toolbox.
3. Implementing PUSCH chain on MATLAB with the help of the openLTE [36] open source implementation and testing its results versus the MATLAB LTE System Toolbox results.
4. Implementing the PUSCH on Intel® Core™ i7 processors (ANSI-C code) and test it versus MATLAB results. Make use of Intel MKL and AVX instructions for parallelizing the LTE chain.
5. Implementing the PUSCH with CUDA (C code) and test it versus MATLAB results.
6. Optimizing the C code implementation for SISO chain using CUDA.
7. Implementing 2x2, 4x4 and 64x64 MIMO chains.
8. Obtaining time profiling results for both SISO and MIMO PUSCH chains.

6.2 Used Tools

To achieve better results in both testing and implementation, faster, more productive, and well-known tools were used. The used tools can be divided into:

1. System design/test tools
   - MATLAB R2016b
2. Integrated Development Environments
   - Microsoft Visual Studio Community 2015
   - Eclipse
3. Other Tools
   - Intel C++ Compiler

- CUDA 8.0 (C/C++)
- Nvidia Visual Profiler

6.3 Attempted optimization techniques

Not only the mentioned techniques in chapters (2) and (3) were experimented, but also other techniques were examined on both Intel and Nvidia architectures.

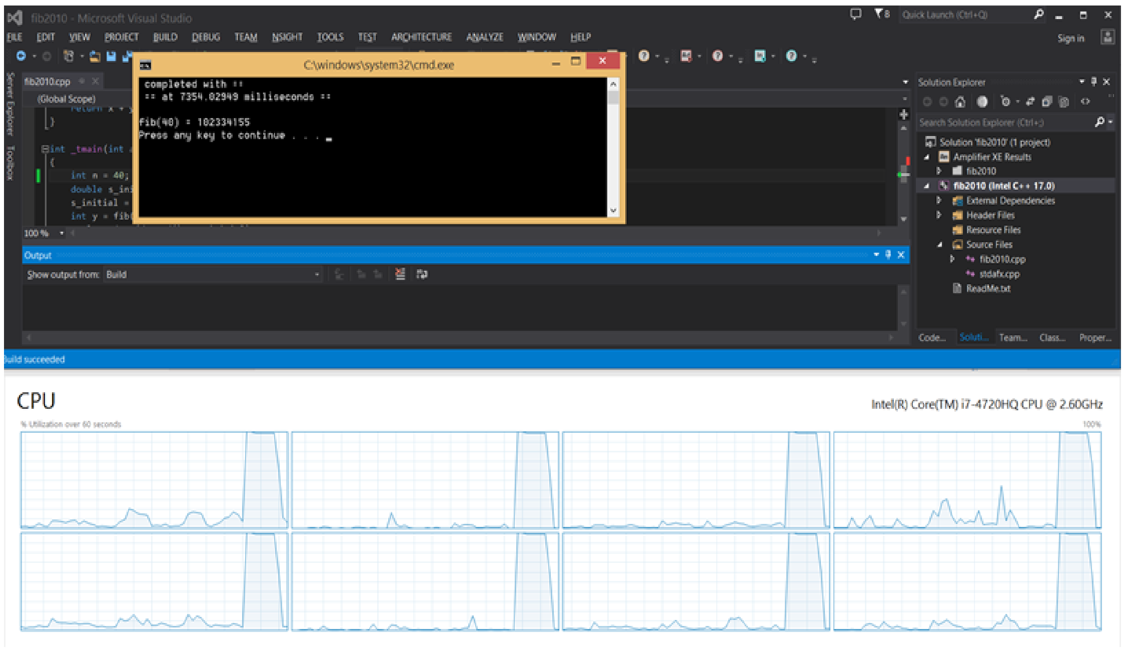
Intel techniques:

- Cilk spawn
- Intel internal graphics off loading
- Intel openMP

Nvidia techniques:

- Parallel Streams
- Unified Memory

All of the above techniques were tested on the chain. One example to see the effect of the cilk spawn operation on the performance of Intel CPU will be seen in Figure 6.1. It shows the experiment of the Fibonacci series algorithm over $n = 40$, where n is the argument of the recursive function. Each recursive call is spawned using cilk_spawn. Finally cilk_sync is used at the end of the function body to end threads together. All CPUs reach their max performance at the same time.



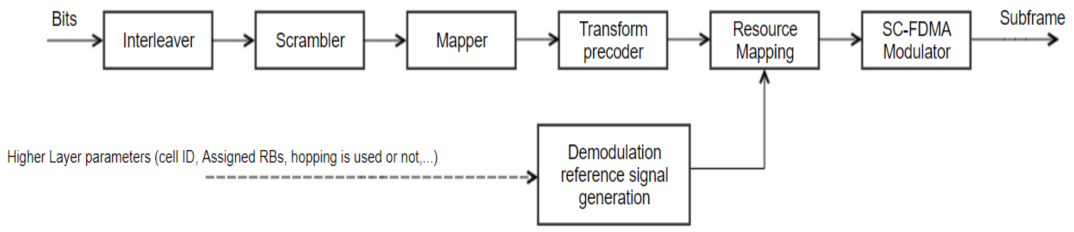
```
completed with ::
:: at 7354.02949 milliseconds ::

fib(40) = 102334155
Press any key to continue . . .
```

fib2010 - Microsoft Visual Studio
C:\windows\system32\cmd.exe
Solution Explorer
Output
Show output from: Build
Build succeeded
CPU
% Utilization over 60 seconds
Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
100%

Figure 6.1: Performance of Cilk for operation on Fibonacci series example.

6.4 Time Profiling

The time profiling for the chain is performed to make sure that the software meets the timing requirements of the LTE standard. The LTE standard specifies the maximum timing for the physical layer chain to be 1 ms. The following figures show the implemented uplink transmitter chains (SISO/MIMO) defined by the LTE standard. The time profiling results are presented in section 7.2.



Bits
Interleaver
Scrambler
Mapper
Transform precoder
Resource Mapping
SC-FDMA Modulator
Subframe
Higher Layer parameters (cell ID, Assigned RBs, hopping is used or not,...)
Demodulation reference signal generation

Figure 6.2: Uplink transmitter SISO chain.

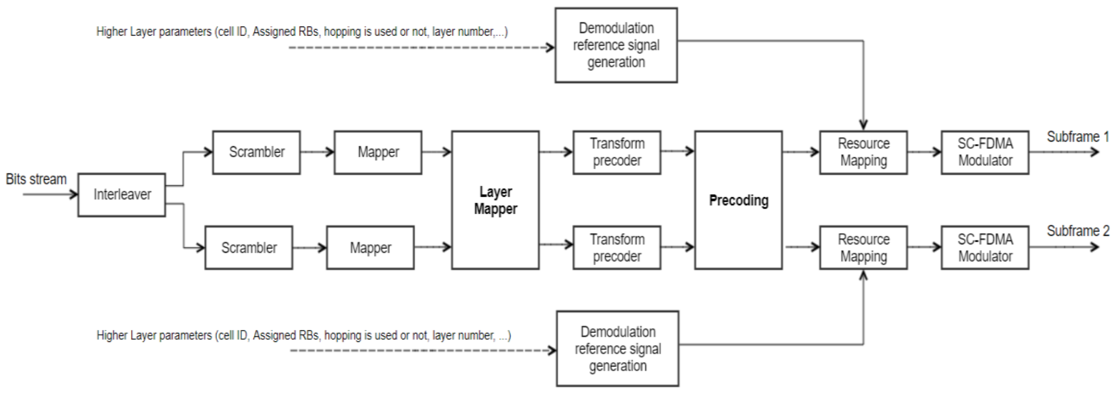

Higher Layer parameters (cell ID, Assigned RBs, hopping is used or not, layer number,...)
Demodulation reference signal generation
Bits stream
Interleaver
Scrambler
Mapper
Scrambler
Mapper
Layer Mapper
Transform precoder
Transform precoder
Precoding
Resource Mapping
Resource Mapping
SC-FDMA Modulator
SC-FDMA Modulator
Subframe 1
Subframe 2
Higher Layer parameters (cell ID, Assigned RBs, hopping is used or not, layer number, ...)
Demodulation reference signal generation

Figure 6.3: Uplink transmitter MIMO chain.

The timing is measured using two methods:

- Chrono time library supported from C++11.
- Nvidia visual profiler.

Both methods get the same timing for the chain. The figure below shows an example of profiling the transmitter chain using Nvidia visual profiler. The timing of three blocks of SISO chain: Interleaver, Scrambler, and Mapper (five kernels) is shown in the timeline in Figure 6.4.

The overhead of memory allocation and plan creation for FFT is neglected as it is done only one time at startup (see Figure 6.5).

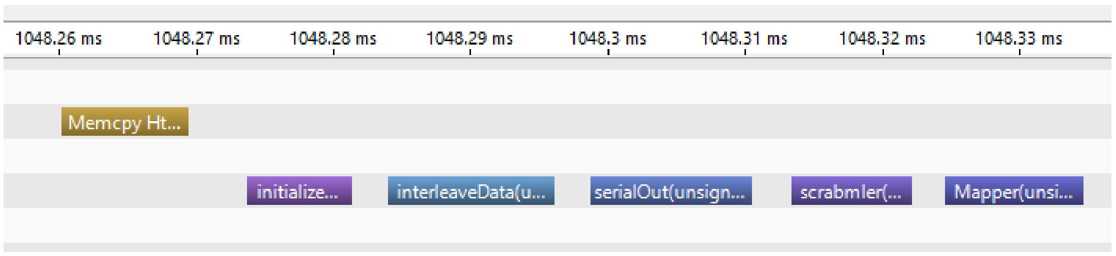


Figure 6.4: Nvidia visual profiler for interleaver, scrambler and Mapper.

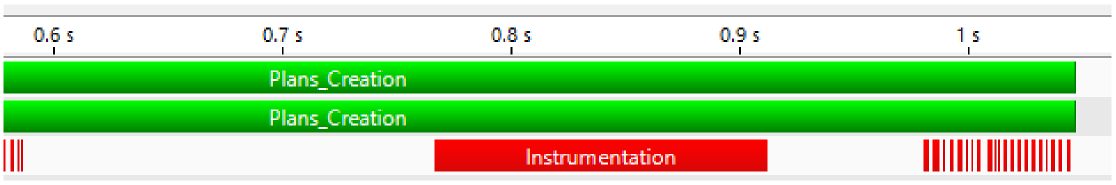


Figure 6.5: Overhead of Plans Creation.

Chapter 7: Results

In this chapter we are going to conclude our achievements and results through a year full of team work, enthusiasm, hard work and research.

7.1 BER plots on MATLAB

7.1.1 SISO chain plots for AWGN channel

BER of the SISO chain in Figure 6.2 is plotted for QPSK and 64QAM modulation schemes as shown in Figure 7.1 and Figure 7.2 respectively.

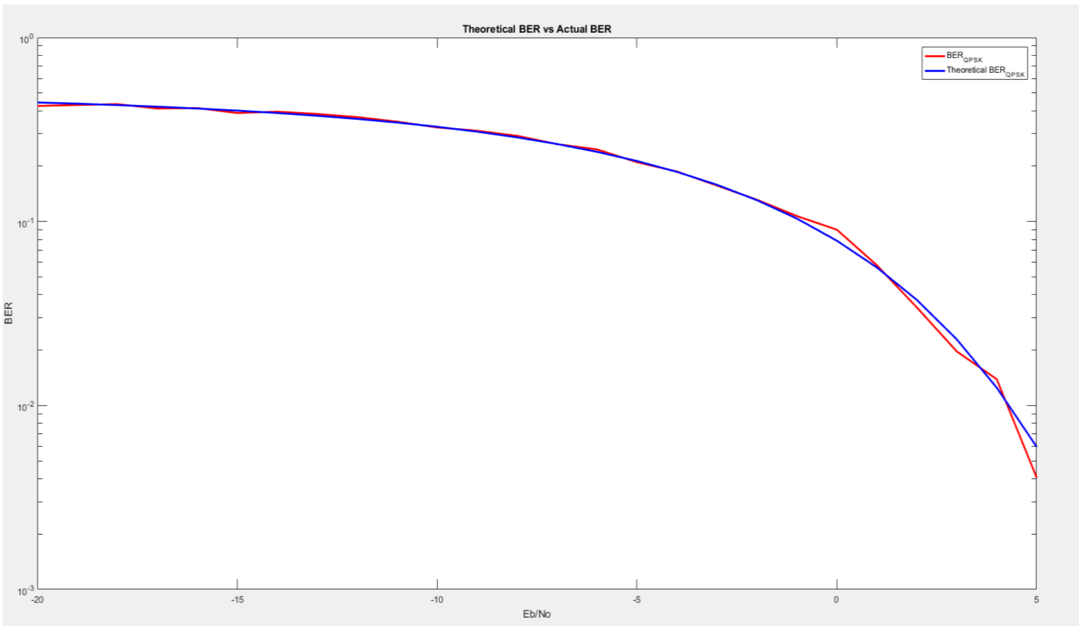


Figure 7.1: BER plot of SISO chain with QPSK modulation.

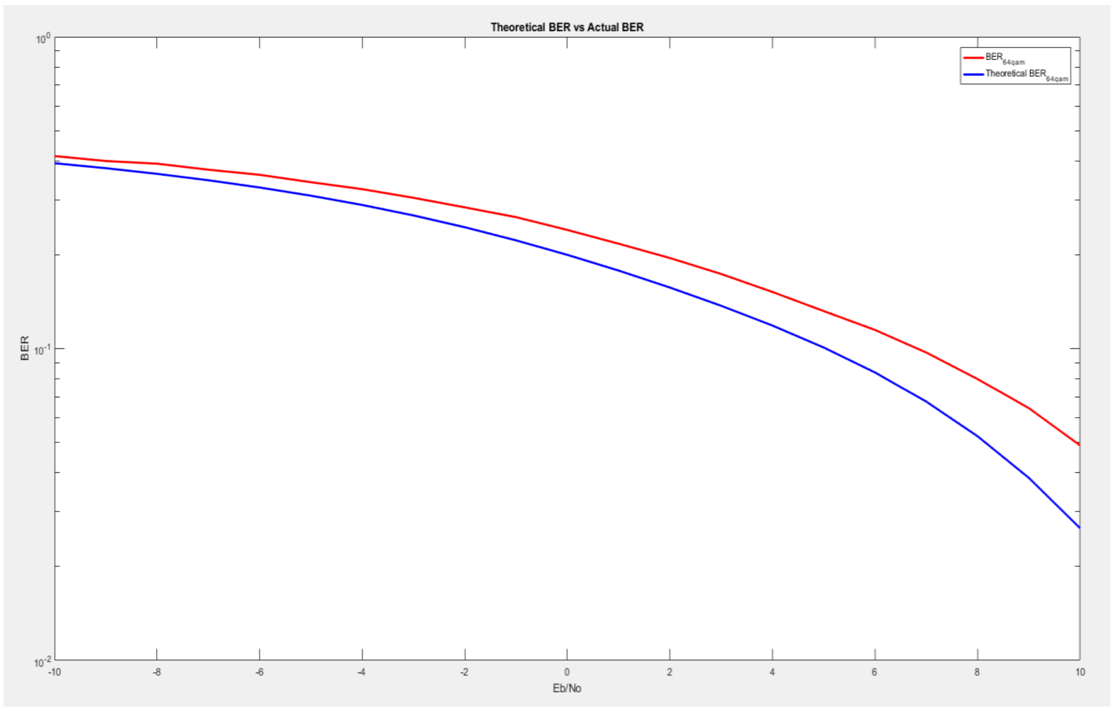


Figure 7.2: BER plot of SISO chain with 64-qam modulation.

7.1.2 SISO chain plots for noisy fading channel

BER of full SISO chain without channel encoding and using channel estimation and zero forcing equalization in Extended Pedestrian A channel is shown in Figure 7.3:

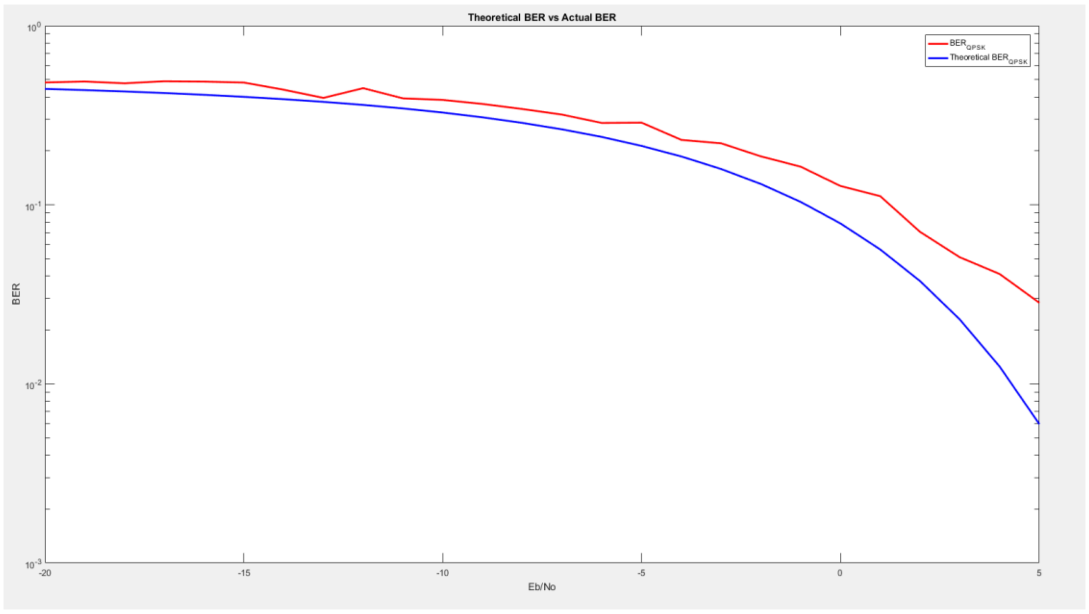


Figure 7.3: BER for fading channel with zero forcing equalization

The BER for the same channel but with MMSE equalization is shown in Figure 7.4:

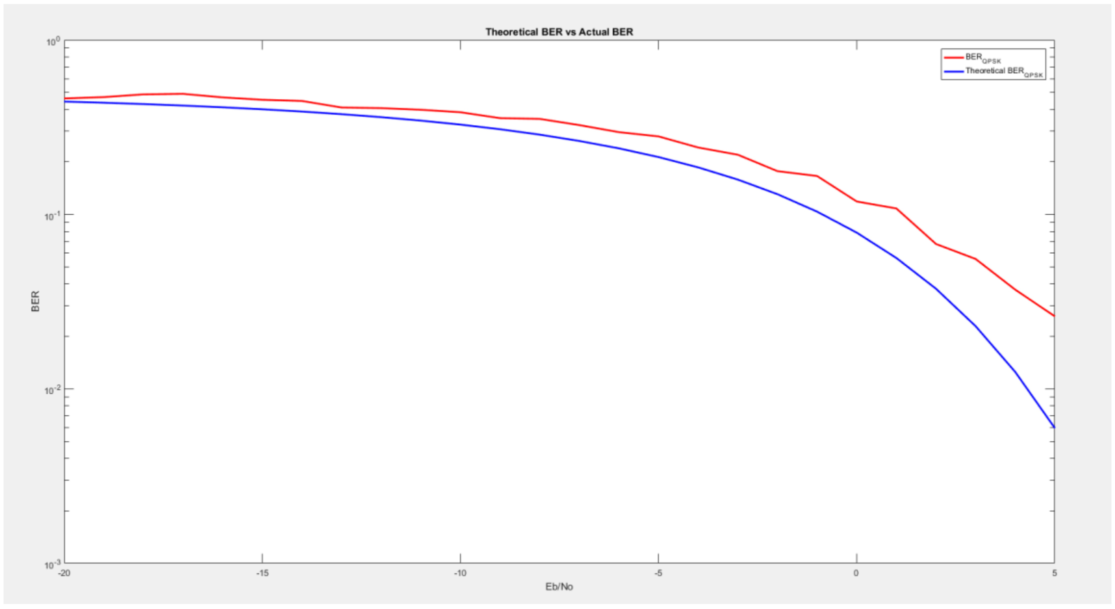


Figure 7.4: BER for fading channel with MMSE equalization

7.1.3 SISO chain with tail biting channel encoding plots for AWGN channel.

Tail biting encoding and Viterbi decoder were implemented, and the BER of the chain for AWGN channel is plotted in Figure 7.5.

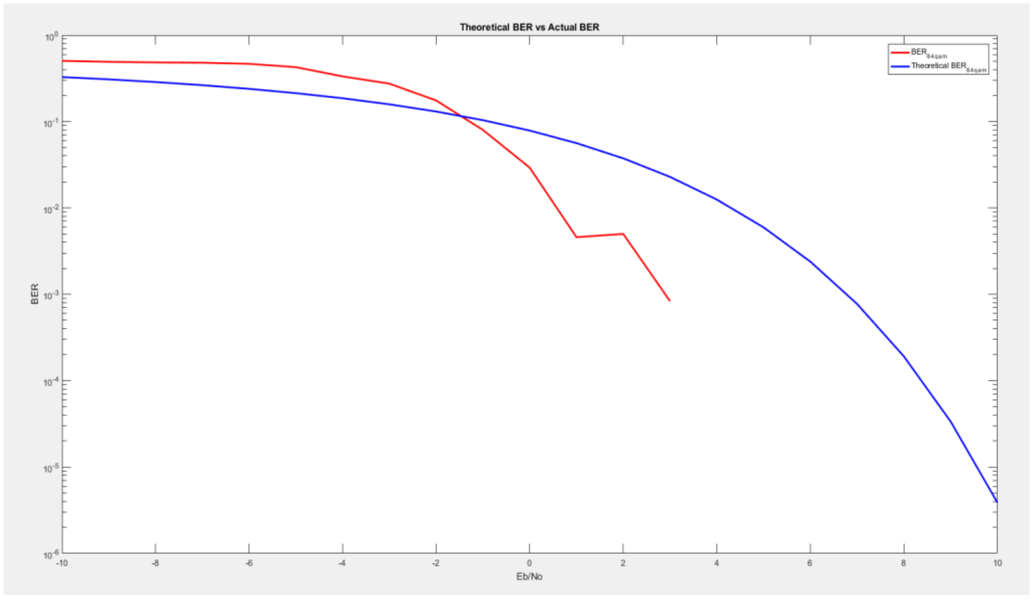


Figure 7.5: BER of SISO chain with channel encoding for AWGN channel.

7.1.4 MIMO chain plots for AWGN channel

BER of the MIMO chain in Figure 6.3 is plotted for QPSK as shown in

Figure 7.6: BER plot of MIMO chain with QPSK modulation.

.

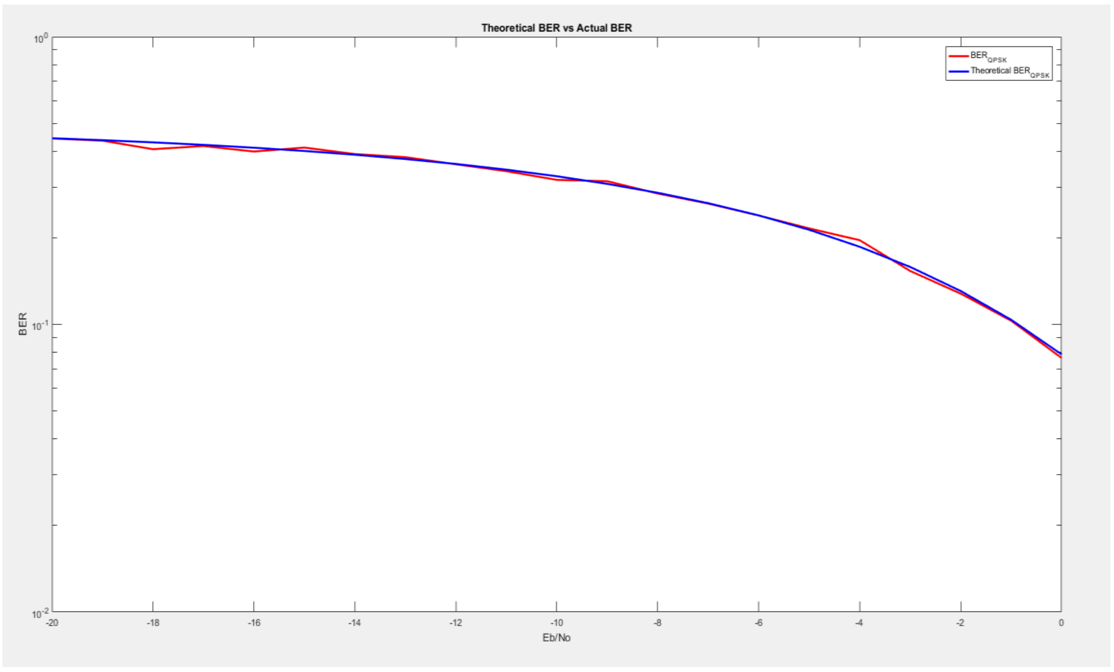


Figure 7.6: BER plot of MIMO chain with QPSK modulation.

7.1.5 MIMO chain with tail biting channel encoding plots for AWGN channel.

BER rate plot of MIMO chain with channel encoding and viterbi decoding is plotted in Figure 7.7.

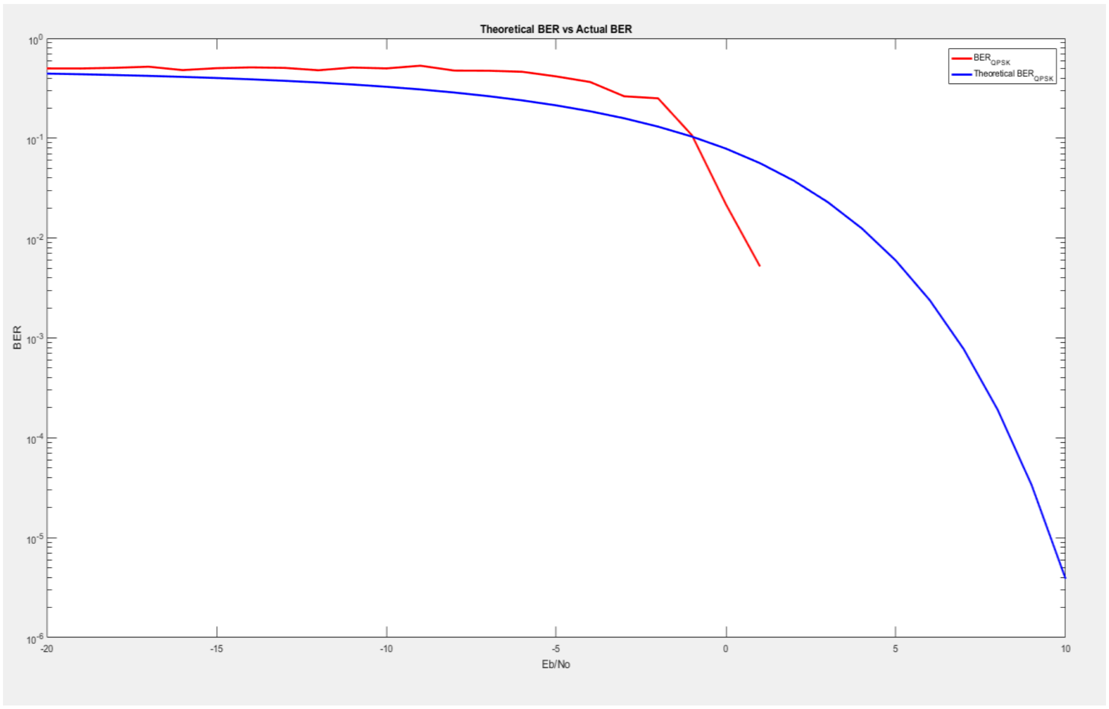


Figure 7.7: BER of MIMO chain with channel encoding for AWGN channel.

7.1.6 MIMO chain plots for noisy fading channel

BER of full MIMO chain without channel encoding and using channel estimation and equalization in Extended Pedestrian A model channel with no channel correlation is shown in Figure 7.8:

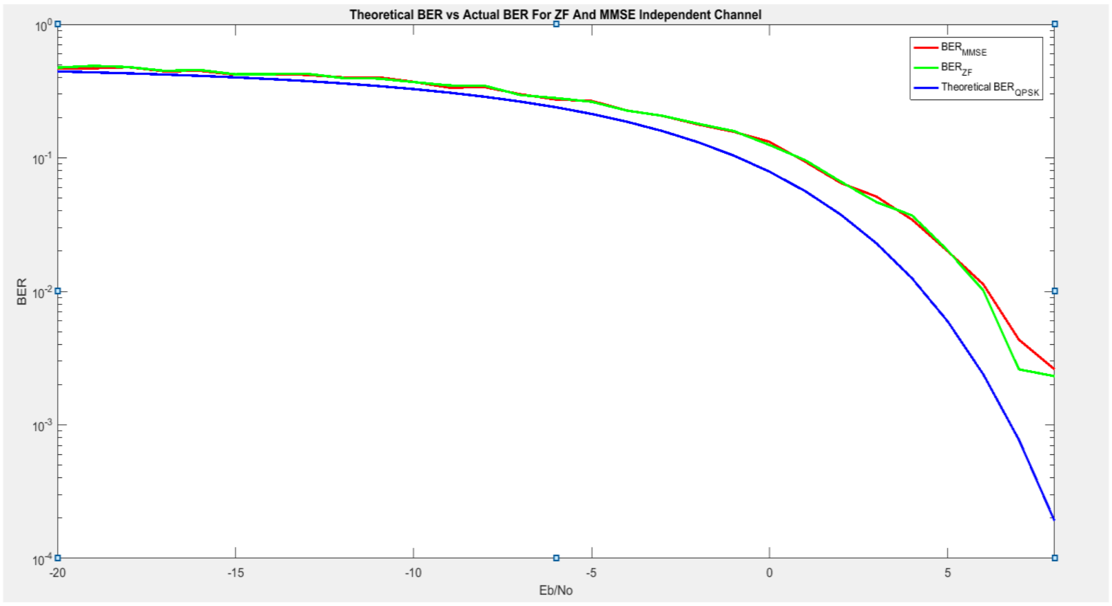


Figure 7.8: BER of MIMO chain with channel encoding for AWGN channel

BER of full MIMO chain without channel encoding and using channel estimation and equalization in Extended Pedestrian A model channel with channel correlation is shown in Figure 7.9:

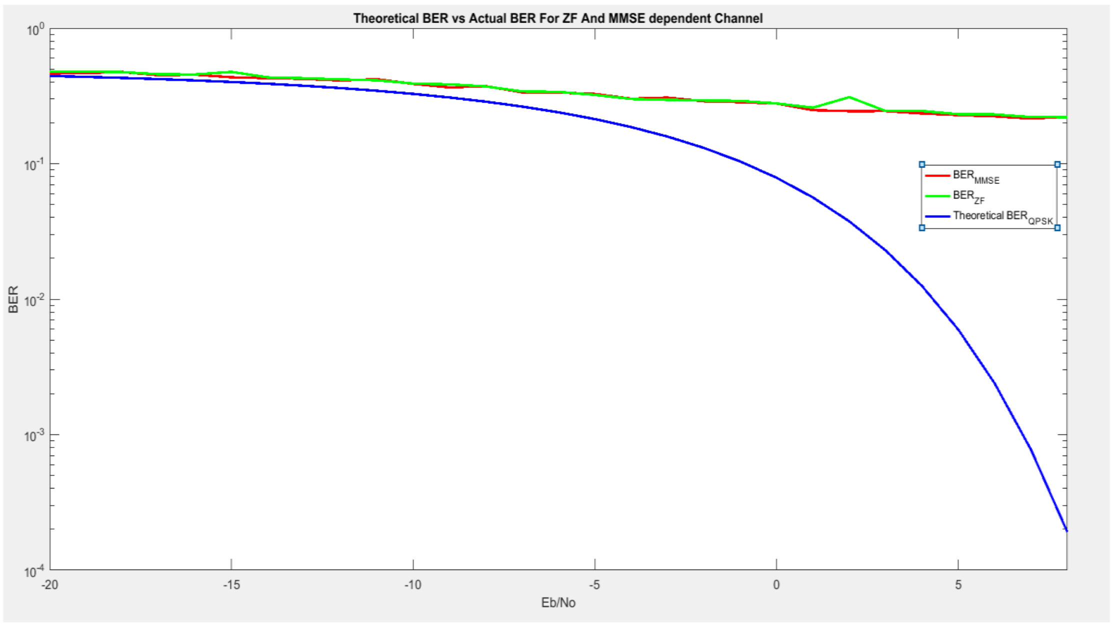


Figure 7.9: BER of MIMO chain with channel encoding for AWGN channel.

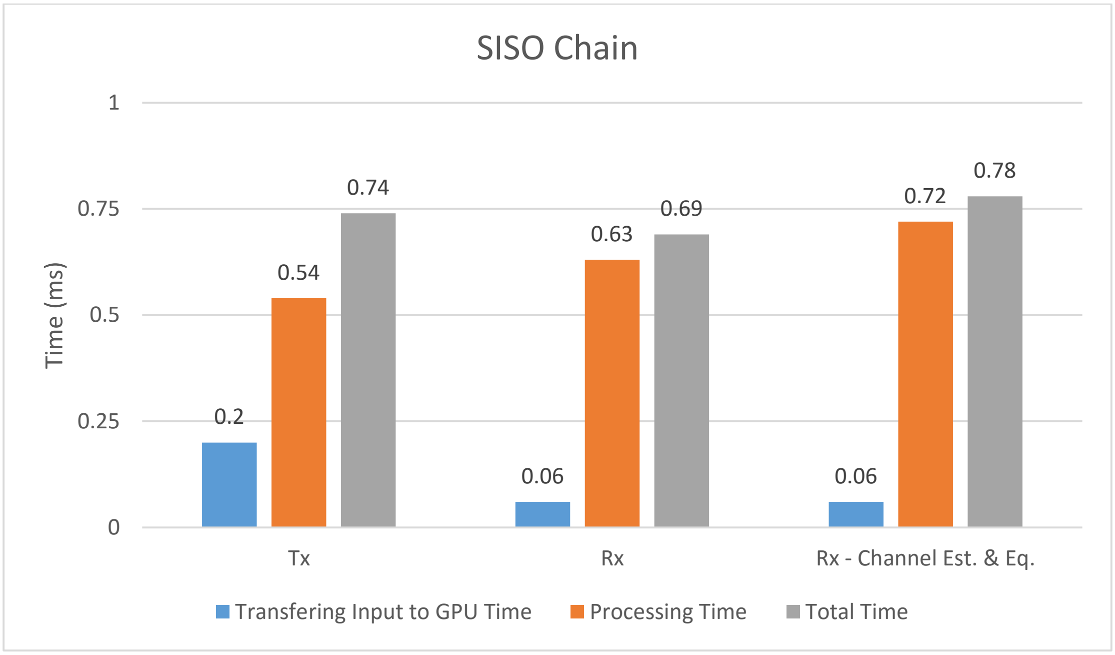
7.2 Timing Results

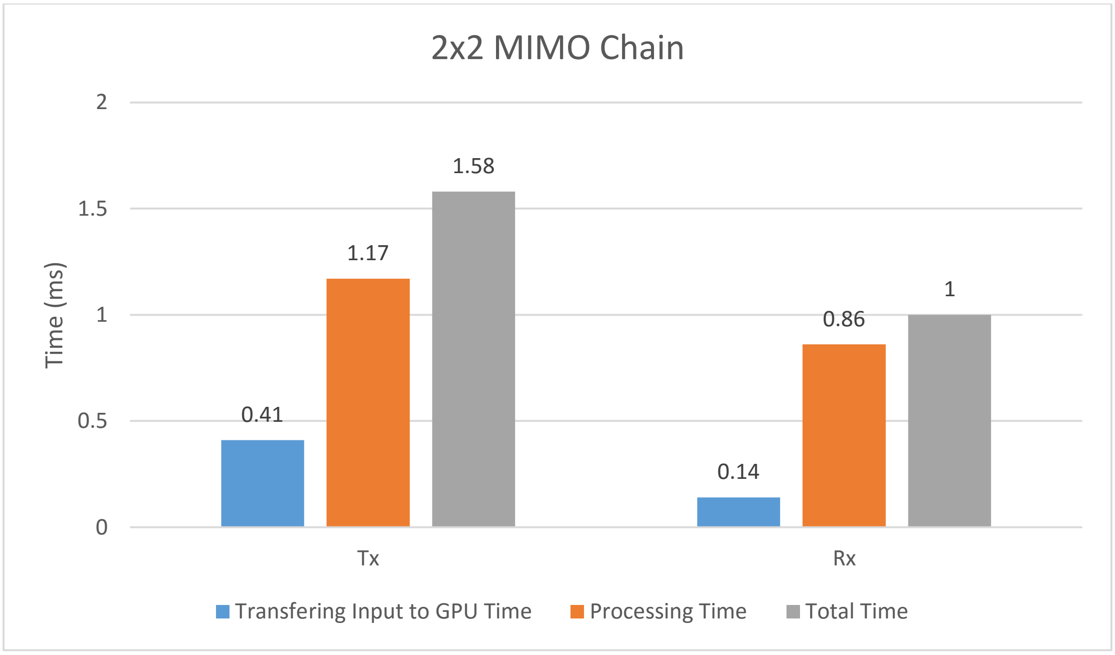
Used platform:

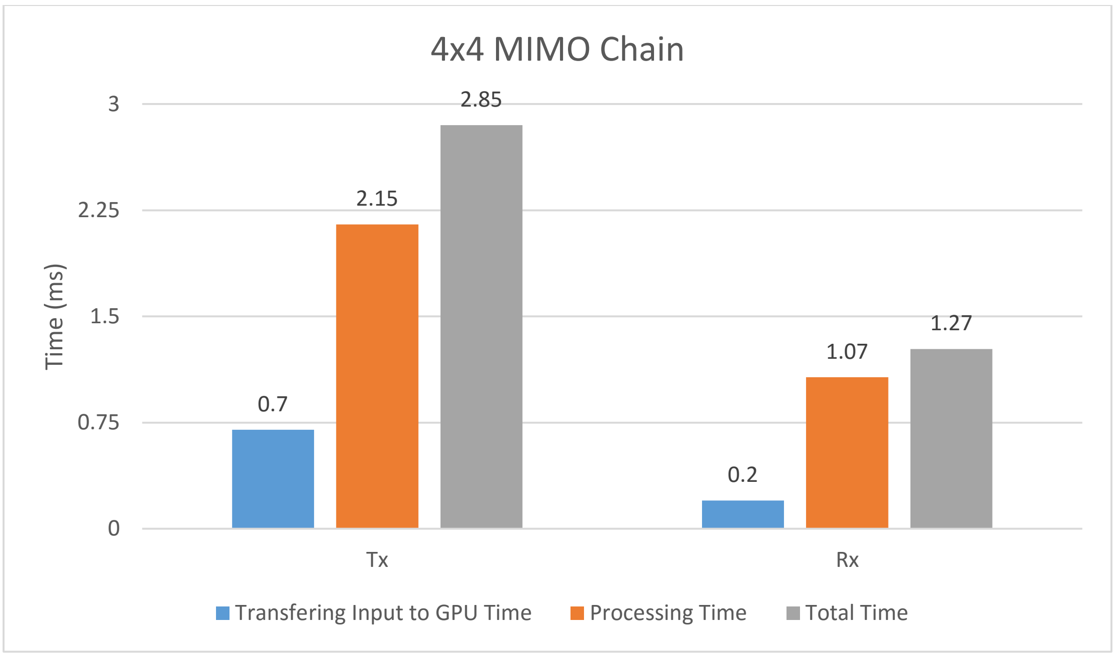
- **CPU:** Intel® Core™ i7-4720HQ (6M Cache, up to 3.60 GHz)
- **GPU:** NVIDIA GeForce GTX 960M (CUDA Cores: 640, Base Clock: 1096 MHz)
- **Memory:** 16.0 GB DDR3L (Dual-Channel)
- **Hard Drive:** Samsung SSD 850 EVO 1TB
- **Operating System:** Windows 10 Pro (Build 10240)

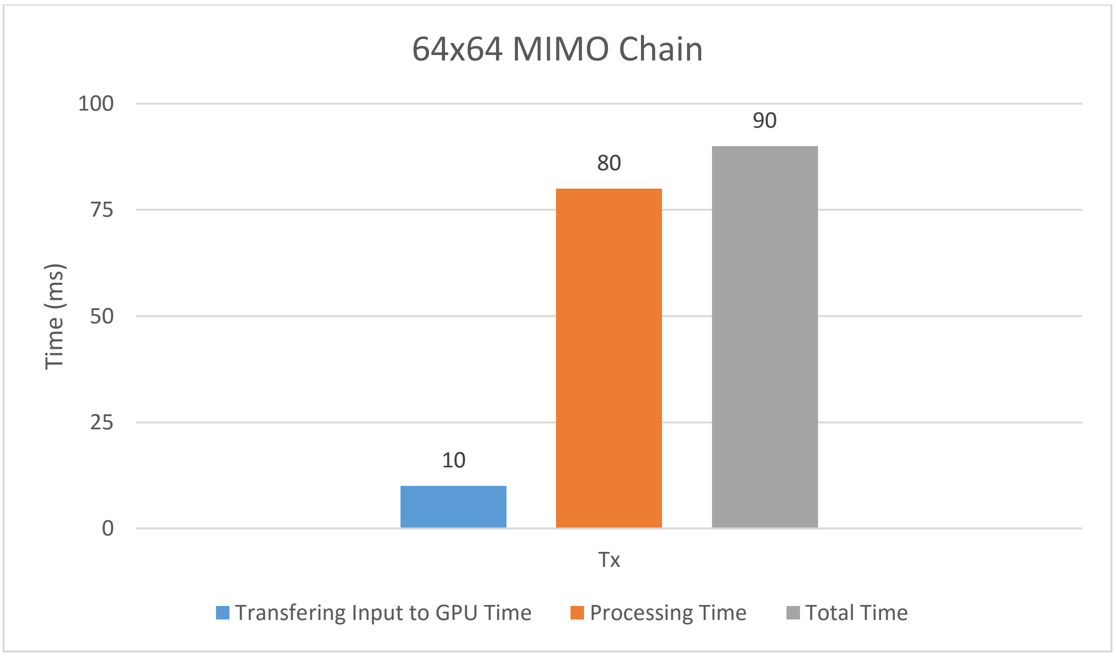
7.2.1 CUDA Implementation

The following timing results are the average of 1000 iterations.

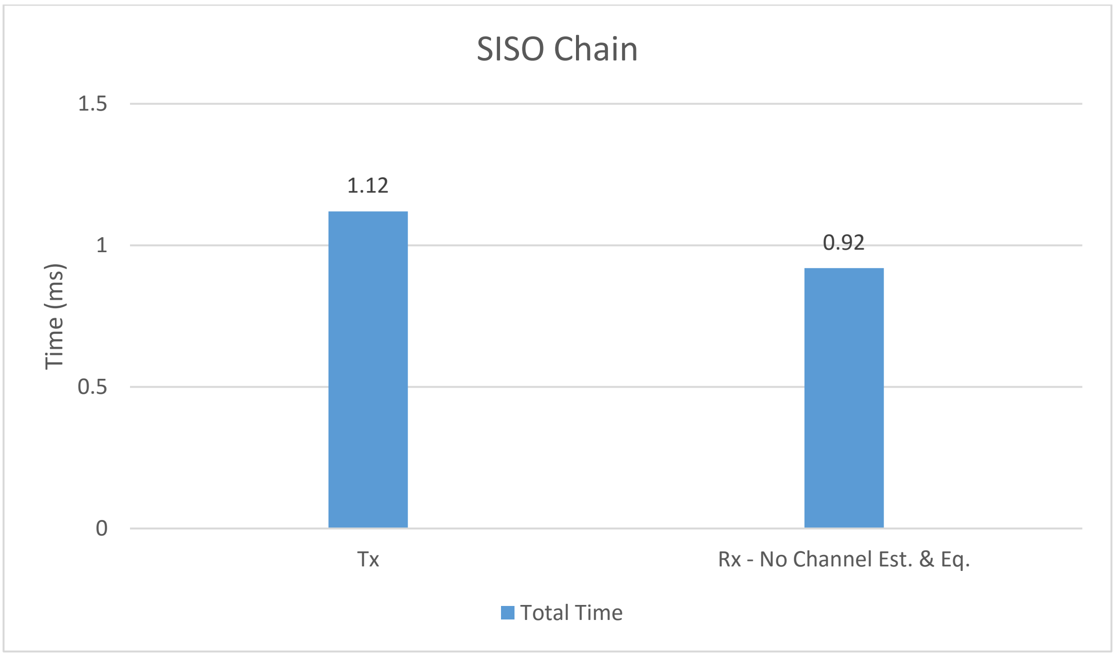





4x4 MIMO Chain
Time (ms)
3
2.25
1.5
0.75
0
0.7
2.15
2.85
0.2
1.07
1.27
Tx
Rx
Transfering Input to GPU Time
Processing Time
Total Time


64x64 MIMO Chain
Time (ms)
100
75
50
25
0
10
80
90
Tx
Transfering Input to GPU Time
Processing Time
Total Time

7.2.2 Intel AVX & MKL Implementation



Conclusion

For CUDA implementation, subframe time of the **SISO** bit processing is $\mathbf{0.75}\ \boldsymbol{ms}$ which satisfies the 3GPP standard stating that the processing time of the subframe should be less than $1\ ms$. Also, for **2x2 MIMO Rx** implementation, subframe time is $\mathbf{1}\ \boldsymbol{ms}$**,** while for the **2x2 MIMO Tx**, subframe time is $\mathbf{1.58}\ \boldsymbol{ms}$ which should be optimized to follow the standard. Also massive MIMO processing time is investigated to give indication of the required improvements of the entire solution in order to serve large number of antennas.

Future work

After the implementation of the PUSCH chain using MATLAB and C, investigating the processing time of the SISO and MIMO configurations, and trying multiple optimization techniques, there is still work to be done to finish the processing of the physical layer completely.

The work that needs to be done is:

- Trying other optimization techniques like kernel merging to reduce the time needed by the CPU to invoke multiple kernels.
- Support control channels.
- Trying more complex channel estimation and equalization techniques which can mitigate the effect of the correlation of the channels for the MIMO system.
- Trying Intel AVX2 and AVX512.
- Trying architecture optimization by merging between Intel AVX and CUDA.

References

[1] North Carolina ECE Reaserch, "Microprocessor Architecture," North Carolina ECE Reaserch, [Online]. Available: https://www.ece.ncsu.edu/ research/cas/ma. [Accessed 10 July 2017].

[2] Intel Embedded, Design Center, "Introduction to Intel® Architecture," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-introduction-basics-paper.html. [Accessed 10 July 2017].

[3] R. Rajwar, M. Dixon and R. Singhal, "Specialized Evolution of the General-Purpose CPU," cidrdb.org, Hillsboro, USA, 2015.

[4] Intel, "Intel Intrinsics Guide," Intel, [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2. [Accessed 10 July 2017].

[5] I. Corporation. [Online]. Available: https://software.intel.com/en-us/node/522582.

[6] C. developers. [Online]. Available: https://www.cilkplus.org/cilk-plus-tutorial.

[7] B. Barney and L. Livermore, "Introduction to Parallel Computing," [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/. [Accessed 10 July 2017].

[8] G. M. Ung and A. Castle, "The History of a Dream: How the Ultimate PC Has Evolved In 15 Years," [Online]. Available: https://web.archive.org/ web/20150418074002/http://www.maximumpc.com:80/article/home/ history_dream_how_ultimate_pc_has_evolved_15_years. [Accessed 11 July 2017].

[9] "Why are there limits on CPU speed?," [Online]. Available: http://computer.howstuffworks.com/question307.htm. [Accessed 11 July 2017].

[10] "What limits CPU speed?," [Online]. Available: https://electronics.stackexchange.com/questions/122050/what-limits-cpu-speed. [Accessed 11 July 2017].

[11] A. Waghmare, "Parallel Computing Seminar Report," 2008.

[12] "Amdahl's law," [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%27s_law. [Accessed 11 July 2017].

[13] "Gustafson's law," [Online]. Available: https://en.wikipedia.org/wiki/Gustafson%27s_law. [Accessed 11 July 2017].

[14] "Graphics Processing Unit (GPU)," [Online]. Available: http://www.nvidia.com/object/gpu.html. [Accessed 11 7 2017].

[15] "What’s the Difference Between a CPU and a GPU?," [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. [Accessed 11 7 2017].

[16] "ACCELERATED COMPUTING," [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html. [Accessed 11 7 2017].

[17] "General-purpose computing on graphics processing units," [Online]. Available: https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units. [Accessed 11 7 2017].

[18] E. Dahlman, S. Parkvall and J. Skold, 4G: LTE/LTE-Advanced for Mobile Broadband, Oxford: Academic Press, 2014.

[19] D. H. Zarrinkoub, Unerstanding LTE with MATLAB From Mathematical Modeling to Simulation and Prototyping, Massachusetts: Wiley, 2014.

[20] Rohde and Schwarz, Rohde & Schwarz: LTE Technology Introduction, Munich: http://www.academia.edu, 2012, pp. 15-16.

[21] "OFDM Orthogonal Frequency Division Multiplexing Tutorial," [Online].

Available: http://www.radio-electronics.com/info/rf-technology-design/ofdm/ofdm-basics-tutorial.php. [Accessed 7 July 2017].

[22] "Cyclic prefix," [Online]. Available: https://en.wikipedia.org/wiki/Cyclic_prefix. [Accessed 7 July 2017].

[23] "LTE OFDM, OFDMA SC-FDMA & Modulation," [Online]. Available: http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/lte-ofdm-ofdma-scfdma.php. [Accessed 7 July 2017].

[24] "An Introduction To SC-FDMA Used By LTE In Uplink Direction," [Online]. Available: http://mobilesociety.typepad.com/ mobile_life/2007/05/an_introduction.html. [Accessed 7 July 2017].

[25] tutorials point, "LTE Communication Channel," tutorials point, [Online]. Available: http://www.tutorialspoint.com/ lte/lte_communication_channels.htm. [Accessed 7 July 2017].

[26] Keysight Technologies, Inc., "LTE Physical Layer Overview," Keysight Technologies, Inc., [Online]. Available: http://rfmw.em.keysight.com/ wireless/helpfiles/89600B/webhelp/subsystems/lte/content/lte_overview.htm. [Accessed 7 July 2017].

[27] Telesystem Innovations Inc., LTE in a Nutshell: The Physical Layer, Telesystem Innovations Inc., 2010, pp. 4-7.

[28] 3GPP, 3GPP TS 36.101; User Equipment (UE) radio transmission and reception (Release 8), France: 3GPP, 2008.

[29] "wirelessdictionary," [Online]. Available: http://www.wirelessdictionary.com/Wireless-Dictionary-Interleaving-Definition.html. [Accessed 11 July 2017].

[30] 3GPP TS 36.211 version 8.7.0 Release 8, Nice: 3GPP, 2009.

[31] 3GPP TS 36.213 version 8.8.0 Release 8, Nice: 3GPP, 2009.

[32] N. Instruments, "National Instruments," [Online]. Available: http://www.ni.com/white-paper/14919/en/.

[33] Wikipedia , "Wikipedia," 30 June 2017. [Online]. Available: https://en.wikipedia.org/wiki/Fading.

[34] MathWorks, "MathWorks," [Online]. Available: https://www.mathworks.com/help/lte/ug/channel-estimation.html.

[35] 3GPP, 3GPP TS 36.141 version 10.1.0 Release 10, Nice: 3GPP, 2011.

[36] "openLTE, an open source 3GPP LTE implementation.," [Online]. Available: https://sourceforge.net/projects/openlte/. [Accessed 12 July 2017].