DOI:10.15774/PPKE.ITK.2015.010

Design and Implementation of High-Performance Computing Algorithms for Wireless MIMO Communications

Thesis submitted for the degree of Doctor of Philosophy

Csaba Máté Józsa, M.Sc.

Supervisors

Géza Kolumbán, D.Sc.

Doctor of the Hungarian Academy of Sciences

and

Péter Szolgay, D.Sc.

Doctor of the Hungarian Academy of Sciences

Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Multidisciplinary Doctoral School of Sciences and Technology

Budapest, 2015

DOI:10.15774/PPKE.ITK.2015.010

DOI:10.15774/PPKE.ITK.2015.010

I would like to dedicate this work to my loving family.

DOI:10.15774/PPKE.ITK.2015.010

Acknowledgements

First of all, I am most grateful to my supervisors, *Profs. Géza Kolumbán* and *Péter Szolgay* for their motivation, guidance, patience and support.

I owe my deepest gratitude to *Prof. Tamás Roska* for the inspiring and thought-provoking discussions and for opening up new horizons. I deeply thank *Prof. Árpád Csurgay* and *Dr. András Oláh* for the motivation and encouragement, giving me the strength to go on.

I am immensely grateful to *Prof. Antonio M. Vidal* for inviting me to his research group at the Institute of Telecommunications and Multimedia Applications (iTEAM) of the Polytechnic University of Valencia and sharing with me his wide experience in computational mathematics every time I needed it. I am very grateful to *Prof. Alberto González*, *Dr. Gema Piñero* and *Dr. Francisco J. Martínez-Zaldívar* for their generous advice and help in the field of wireless communications and showing me the strength of the team work.

I would like to thank all my friends and colleagues with whom I spent these past few years *Dóra Bihary, Bence Borbély, Zsolt Gelencsér, Antal Hiba, Balázs Jákli, András Laki, Endre László, Norbert Sárkány, István Reguly, János Rudán, Zoltán Tuza, Tamás Fülöp, András Horváth, Miklós Koller, Mihály Radványi, Ádám Rák, Attila Stubendek, Gábor Tornai, Tamás Zsedrovits, Ádám Balogh, Ádám Fekete, László Füredi, Kálmán Tornai, Ákos Tar, Dávid Tisza, Gergely Treplán, József Veres, András Bojárszky, Balázs Karlócai, Tamás Krébesz.* Special thanks go to *István Reguly* for more than a decade of friendship and of continuous collaboration.

Thanks also to my colleagues at the iTEAM: *Emanuel Aguilera, Jose A. Belloch, Fernando Domene, Laura Fuster Pablo Gutiérrez, Jorge Lorente, Luis Maciá, Amparo Martí, Carla Ramiro* for all the good moments spent together while working at the lab.

I thank the Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, for accepting me as a doctoral student and supporting me throughout grants TÁMOP-4.2.1/B-11/2/KMR-2011-0002 and TÁMOP-4.2.2/B-10/1-2010-0014.

DOI:10.15774/PPKE.ITK.2015.010

Finally, I will be forever indebted to my parents *Enikő* and *Máté*, brother *István*, and grandparents *Böbe, Éva, Márton* for enduring my presence and my absence, and for being supportive all the while, helping me in every way imaginable.

Last, but not least, I would like to express my sincere gratitude to my wife *Ildikó*, who offered me unconditional love and support throughout the course of this thesis.

DOI:10.15774/PPKE.ITK.2015.010

Kivonat

A vezeték nélküli kommunikációban a többantennás (MIMO) rendszerek azzal kerültek a figyelem középpontjába, hogy jelentős adatátviteli sebesség növekedést és jobb lefedettséget tudnak elérni a sávszélesség és az adóteljesítmény növelése nélkül. A jobb teljesítmény ára a hardverelemek és a jelfeldolgozó algoritmusok megnövekedett komplexitása. A legújabb kutatási eredmények azt igazolják, hogy a masszívan párhuzamos architektúrák (MPA-k) számos számításigényes feladatot képesek hatékonyan megoldani. Kutatásom célja, hogy a többantennás vezeték nélküli kommunikáció területén megjelenő magas komplexitású jelfeldolgozási feladatokat a modern MPA-k segítségével, mint például az általános célú grafikus feldolgozó egységek (GP-GPU-k) vagy sokmagos központi egységek (CPU), hatékonyan megoldjak.

MIMO rendszerekben az optimális Maximum Likelihood (ML) becslésen alapuló detekció komplexitása exponenciálisan növekszik az antennák számával és a moduláció rendjével. A szférikus detektor (SD) jelentősen csökkentve a lehetséges megoldások keresési terét hatékonyan oldja meg az ML detekciós problémát. Az SD algoritmus fő hátránya a szekvenciális jellegében rejlik, ezért futtatása MPA-kon nem hatékony.

Kutatásom első részében bemutatom a párhuzamos szférikus detekciós (PSD) algoritmust, mely kiküszöböli az SD algoritmus hátrányait. A PSD algoritmus egy új fakeresési algoritmust valósít meg, ahol a párhuzamosságot egy hibrid, szélességi és mélységi fakeresés hatékony kombinációja biztosítja. Az algoritmus minden keresési szinten egy út metrikán alapuló párhuzamos rendezést hajt végre. A PSD algoritmus paraméterei segítségével képes meghatározni a memória igényét és párhuzamosságának mértékét ahhoz, hogy hatékonyan ki tudja használni számos párhuzamos architektúra erőforrásait. A PSD algoritmus MPA-ra való leképezését a rendelkezésre álló szálak számától függő, párhuzamos építőelemek segítségével valósítom meg. Továbbá, a párhuzamos építőelemek alapján a PSD algoritmus egy GP-GPU architektúrán kerül implementálásra. A csúcsteljesítmény eléréséhez több párhuzamossági szint meghatározására és különböző ütemezési stratégiákra van szükség.

DOI:10.15774/PPKE.ITK.2015.010

Többfelhasználós kommunikációs rendszerek esetén, ha a bázisállomás több antennával rendelkezik a térbeli diverzitás akkor is kihasználható, ha a mobil állomásoknak csak egy antennájuk van. Mivel a mobil állomások számára nem minden kommunikációs csatorna ismert, az összes jelfeldolgozási feladat a bázisállomásra hárul, ilyen például a szimbólumok előkódolása, mely a felhasználók közötti interferenciát szünteti meg. Ha a bázisállomás számára a mobil állomások visszacsatolják a csatorna paramétereit, akkor a lineáris és nemlineáris előkódolási technikák teljesítménye tovább javítható rácsredukciós algoritmusok alkalmazásával. Több kutatás is bizonyította, hogy a rácsredukcióval támogatott lineáris és nemlineáris előkódolás jelentősen csökkenti a bithibaarányt a rácsredukcióval nem támogatott előkódoláshoz képest. Továbbá számos kutatás megmutatta, hogy a rácsredukció a lineáris és nemlineáris detekció teljesítményét is tudja fokozni. A rácsredukció során létrejövő új bázis kondíciószáma és az ortogonalitási hiba mérséklése lehetővé teszi, hogy a kevésbé komplex lineáris detekciós algoritmusok is teljes rendű diverzitást érjenek el.

A rácsredukciós algoritmusok komplexitása a bázis méretétől függ. Mivel a rácsredukciót a MIMO rendszerek csatorna mátrixán kell végrehajtani, a rácsredukció komplexitása és ezzel együtt a feldolgozási idő kritikussá válhat nagy MIMO rendszerek esetén. Mivel a rácsredukció, pontosabban a polinomrendű Lenstra-Lenstra-Lovász (LLL) rácsredukciós algoritmus fontos szerepet játszik a vezeték nélküli kommunikáció területén, a kutatásom második részében az LLL algoritmus teljesítményének további fokozása a célom.

A párhuzamos All-Swap LLL (AS-LLL) algoritmus esetén a Gram-Schmidt együtthatók minden mátrix oszlopcsere és méret csökkentési procedúra után aktualizálásra kerülnek. Mivel a gyakori oszlopcsere és méret csökkentés több alkalommal módosítja a Gram-Schmidt együtthatók értékét, ezért egyes együtthatók aktualizálása feleslegessé válik. A költségcsökkentett AS-LLL (CR-AS-LLL) algoritmus megtervezésével megmutattam, hogy jelentős komplexitás csökkenés érhető el, ha csak azok a Gram-Schmidt együtthatók kerülnek aktualizálásra, melyek az LLL feltételek kiértékelésében vesznek részt. Ha már nincs végrehajtandó oszlopcsere és méret csökkentési procedúra a fennmaradó együtthatók aktualizálásra kerülnek. Bemutatom egy GP-GPU architektúrára való hatékony leképezését a CR-AS-LLL algoritmusnak, ahol egy kétdimenziós CUDA blokk szálai hajtják végre a redukciót.

Nagyobb méretű mátrixok esetén a párhuzamosság több szintje is kiaknázható. A módosított blokk LLL (MB-LLL) algoritmus egy kétszintű párhuzamosságot valósít meg. A magasabb párhuzamossági szinten a blokk redukciós koncepciót alkalmazom, míg az

DOI:10.15774/PPKE.ITK.2015.010

alacsonyabb szinten az egyes blokkok párhuzamos feldolgozását a CR-AS-LLL algoritmus valósítja meg.

A költség csökkentett MB-LLL algoritmusban a számítási komplexitás tovább csökkenthető, ha engedélyezett az első LLL feltétel lazítása az almátrixok rácsredukciója során. A CR-MB-LLL algoritmust leképeztem egy heterogén architektúrára és a teljesítőképességet összehasonlítottam egy dinamikus párhuzamosságot támogató GP-GPU és többmagos CPU leképezéssel. A heterogén architektúra alkalmazása lehetővé teszi a kernelek dinamikus ütemezését és lehetővé válik több CUDA folyam párhuzamos használata.

Az elért kutatási eredményeim azt mutatják, hogy a MPA-k fontos szerepet fognak játszani a jövőbeli jelfeldolgozó algoritmusok esetén és az aktuális jelfeldolgozó algoritmusok szekvenciális összetevőinek korlátozásai sok esetben megszüntethetők az algoritmusok teljes újratervezésével, ezáltal jelentős teljesítmény növekedés érhető el.

DOI:10.15774/PPKE.ITK.2015.010

Abstract

Multiple–input multiple-output (MIMO) systems have attracted considerable attention in wireless communications because they offer a significant increase in data throughput and link coverage without additional bandwidth requirement or increased transmit power. The price that has to be paid is the increased complexity of hardware components and algorithms. Recent results have proved that massively parallel architectures (MPAs) are able to solve computationally intensive tasks in a very efficient manner. The goal of my research was to solve computationally demanding signal processing problems in the field of wireless communications with modern MPAs, such as General-Purpose Graphics Processing Units (GP-GPUs), and multi-core Central Processing Units (CPUs).

The complexity of the optimal hard-output Maximum Likelihood (ML) detection in MIMO systems increases exponentially with the number of antennas and modulation order. The Sphere Detector (SD) algorithm solves the problem of ML detection by significantly reducing the search space of possible solutions. The main drawback of the SD algorithm is in its sequential nature, consequently, running it MPAs is very inefficient.

In the first part of my research I present the Parallel Sphere Detector (PSD) algorithm that overcomes the drawbacks of the SD algorithm. The PSD implements a novel tree search, where the algorithm parallelism is assured by a hybrid tree search based on the efficient combination of depth-first search and breadth-first search algorithms. A path metric based parallel sorting is employed at each intermediate stage. The PSD algorithm is able to adjust its memory requirements and extent of parallelism to fit a wide range of parallel architectures. Mapping details for MPAs are presented by giving the details of thread dependent, highly parallel building blocks of the algorithm. I show how it is possible to give a GP-GPU mapping of the PSD algorithm based on the parallel building blocks. In order to achieve high-throughput, several levels of parallelism are introduced, and different scheduling strategies are considered.

When a multiple-antenna base station (BS) is applied to multi-user communication, spatial diversity can be achieved even if the mobile stations (MSs) are not equipped

DOI:10.15774/PPKE.ITK.2015.010

with multiple antennas. However, since the MSs do not know other users' channels, the entire processing task must be done at the BS, especially symbol precoding to cancel multi-user interference. Assuming channel information is sent back by all the MSs to the BS, a promising technique that can be applied for both linear and non-linear precoding techniques is the lattice reduction (LR) of the channel matrix. It was shown that the bit error rate (BER) performance of LR-aided precoding is significantly decreased compared to non-LR-aided precoding. Recent research shows that the performance of linear and non-linear detectors can be improved when used in conjuction with LR techniques. The improved condition number and orthogonality defect of the reduced lattice basis achieves full diversity order even with less-complex linear detection methods.

However, the computational cost of LR algorithms depending on the matrix dimensions could be high, and can become critical for large MIMO arrays. Since LR, and more explicitly the polynomial-time Lenstra-Lenstra-Lovász (LLL) lattice reduction algorithm plays an important role in the field of wireless communications, in the second part of my research the focus is to further improve the performance of the LLL algorithm.

In the original parallel All-Swap LLL (AS-LLL) algorithm after every size reduction or column swap the Gram-Schmidt coefficients are updated. However, a lot of unnecessary computations are performed because the frequent size reductions and column swaps change the value of the Gram-Schmidt coefficients several times. With the Cost-Reduced AS-LLL (CR-AS-LLL) algorithm I showed that by updating only those Gram-Schmidt coefficients that are involved in the evaluation of the LLL conditions, a significant complexity reduction is achieved. The remaining coefficients are updated after no more swaps and size reductions have to be performed. I present a GP-GPU mapping of the CR-AS-LLL algorithm where the work is distributed among the threads of a two-dimensional thread block, resulting in the coalesced access of the high-latency global memory, and the elimination of shared memory bank conflicts.

For larger matrices it is possible to exploit several levels of parallelism. The Modified-Block LLL (MB-LLL) implements a two-level parallelism: a higher-level, coarse-grained parallelism by applying a block-reduction concept, and a lower-level, fine-grained parallelism is implemented with the CR-AS-LLL algorithm.

The computational complexity of the MB-LLL is further reduced in the Cost-Reduced MB-LLL (CR-MB-LLL) algorithm by allowing the relaxation of the first LLL condition while executing the LR of sub-matrices, resulting in the delay of the Gram-Schmidt coefficients update and by using less costly procedures during the boundary checks. A

DOI:10.15774/PPKE.ITK.2015.010

mapping of the CR-MB-LLL on a heterogeneous platform is given and it is compared with mappings on a dynamic parallelism enabled GP-GPU and a multi-core CPU. The mapping on the heterogeneous architecture allows a dynamic scheduling of kernels where the overhead introduced by host-device communication is hidden by the use of several Compute Unified Device Architecture (CUDA) streams.

The achieved results of my research show that MPAs will play an important role in future signal processing tasks and limitations imposed by the sequential components of existing signal processing algorithms can be eliminated with the complete redesign of these algorithms achieving significant performance improvement.

DOI:10.15774/PPKE.ITK.2015.010

Abbreviations

APP	A Posteriori Probability
ARBF	Adapting Reduced Breadth-First search
AS-LLL	All-Swap LLL algorithm
ASD	Automatic Sphere Detector
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BFS	Breadth-First Search
BLAST	Bell Laboratories Layered Space-Time architecture
BS	Base Station
CDF	Cumulative Distribution Function
CDI	Channel Distribution Information
CLPS	Closest Lattice Point Search
CPU	Central Processing Unit
CR-AS-LLL	Cost-Reduced AS-LLL algorithm
CR-MB-LLL	Cost-Reduced MB-LLL algorithm
CSI	Channel State Information
CSIR	CSI at the Receiver
CSIT	CSI at the Transmitter
CUDA	Compute Unified Device Architecture
DFS	Depth-First Search
DP	Dynamic Parallelism
DSP	Digital Signal Processor
EEP	Expand and Evaluate Pipeline
FBF	Full-Blown Breadth-First search
FLOP/s	Floating-Point Operations per Second
FP	Fincke-Phost

DOI:10.15774/PPKE.ITK.2015.010

FPGA	Field Programmable Gate Array
FSD	Fixed-Complexity Sphere Detector
GP-GPU	General-Purpose Graphics Processing Unit
HKZ	Hermite-Korkine-Zolotareff algorithm
ILS	Integer Least Squares
LDPC	Low-Density Parity-Check Codes
LLL	Lenstra-Lenstra-Lovász algorithm
LORD	Layered Orthogonal Lattice Detector
LR	Lattice Reduction
LRAP	Lattice-Reduction-Aided Precoding
MB-LLL	Modified-Block LLL algorithm
MIMD	Multiple Instruction, Multiple Data
MIMO	Multiple-Input Multiple-Output
MISD	Multiple Instruction, Single Data
MISO	Multiple-Input Single-Output
ML	Maximum Likelihood
MMSE	Minimum Mean Square Error
MPA	Massively Parallel Architecture
MPI	Message Passing Interface
MS	Mobile Station
NUMA	Non-Uniform Memory Access
OFDM	Orthogonal Frequency-Division Multiplexing
OpenMP	Open Multi-Processing
PE	Processing Element
PER	Packet Error Rate
PSD	Parallel Sphere Detector
QAM	Quadrature Amplitude Modulation
SD	Sphere Detector
SE	Schnorr-Euchner
SIC	Successive Interference Cancellation
SIMD	Single Instruction, Multiple Data
SIMT	Single Instruction, Multiple Threads
SINR	Signal-to-Interference plus Noise Ratio
SISD	Single Instruction, Single Data

DOI:10.15774/PPKE.ITK.2015.010

SISO	Single-Input Single-Output
SM(X)	Streaming Multiprocessor
SMP	Symmetric Multiprocessor
SNR	Signal-to-Noise Ratio
SSFE	Selective Spanning Fast Enumeration
STC	Space-Time Coding
SVD	Singular Value Decomposition
TB	Thread Block
THP	Tomlinson-Harashima Precoding
UMA	Uniform Memory Access
VLSI	Very Large Scale Integration
ZF	Zero-Forcing

DOI:10.15774/PPKE.ITK.2015.010

Contents

DOI:10.15774/PPKE.ITK.2015.010

DOI:10.15774/PPKE.ITK.2015.010

DOI:10.15774/PPKE.ITK.2015.010

List of Figures

DOI:10.15774/PPKE.ITK.2015.010

DOI:10.15774/PPKE.ITK.2015.010

DOI:10.15774/PPKE.ITK.2015.010

List of Tables

DOI:10.15774/PPKE.ITK.2015.010

List of Algorithms

DOI:10.15774/PPKE.ITK.2015.010

Chapter 1

Introduction

1.1 Motivation and scope

The most important driving forces in the development of wireless communications are the need for higher link throughput, higher network capacity and improved reliability. The limiting factors of such systems are equipment cost, radio propagation conditions and frequency spectrum availability. The ever increasing need for higher transmission rates motivated researchers to develop new methods and algorithms to reach the Shannon capacity limit of single transmit and receive antenna wireless systems. Research in information theory [8] has revealed that important improvements can be achieved in data rate and reliability when multiple antennas are applied at both the transmitter and receiver sides, referred to as MIMO systems [9]. The performance of wireless systems is improved by orders of magnitude at no cost of extra spectrum use.

The complexity of MIMO detectors used over different receiver structures depends on many factors, such as antenna configuration, modulation order, channel, coding, etc. In order to achieve optimal BER for Additive White Gaussian Noise (AWGN) channels ML detection has to be performed. The exhaustive search implementation of ML detection has a complexity that grows exponentially with both the number of elements in the signal set and the number of antennas, thus, this technique is not feasible in real systems. The SD seems to be a promising solution to significantly reduce the search space. The fundamental aim of the SD algorithm is to restrict the search to lattice points that lie within a certain sphere around a given received symbol vector. Reducing the search space will not affect the detection quality, because the closest lattice point inside the sphere will also be the closest lattice point for the whole lattice. The drawbacks of the SD algorithm are: (i) the complexity still suffers of an exponential growth, when increasing

DOI:10.15774/PPKE.ITK.2015.010

the number of antennas or the modulation order, (ii) the SD detection transforms the MIMO detection problem into a depth-first tree search that is highly sequential, and (iii) during every tree search several different paths have to be explored leading to a variable processing time.

When a multiple-antenna BS is applied to multi-user communication systems, spatial diversity can be achieved even if the MSs are not equipped with multiple antennas. However, since the MSs do not know other users' channels, the entire processing task must be done at the BS, especially symbol precoding to cancel multi-user interference. Assuming channel information is sent back by all the MSs to the BS, a promising technique that can be applied for both linear and non-linear precoding is the LR of the channel matrix [10], [11], [12], [13]. Furthermore, recent research [14], [15], [16], [17], [18] shows that the performance of linear and non-linear MIMO detectors can be improved when used in conjunction with LR techniques. The improved condition number and orthogonality defect of the reduced lattice basis achieves full diversity order even with less complex linear detection methods. However, the computational cost of LR algorithms depending on the lattice basis dimensions could be high, and can become critical for large MIMO arrays.

The computational complexity can grow extremely high when the optimal solution is required either for detection or precoding. However, during the research of different modulation schemes, channel models, precoding, detection and decoding techniques it might happen that the theoretical performance can be determined only by simulations. Another approach is to precondition or preprocess the problem, and afterwards, lower complexity signal processing algorithms (i.e. linear detection, precoding) can be performed. In this case, the complexity or the processing time is mostly influenced by the preprocessing algorithms. The conclusion of the above is that the price that has to be paid when using MIMO systems is the increased complexity of hardware components and signal processing algorithms and most of these algorithms can not be efficiently mapped to modern parallel architectures because of their sequential components.

Due to the major advances in the field of computing architectures and programming models the production of relatively low-cost, high-performance, MPAs such as GP-GPUs or Field Programmable Gate Arrays (FPGAs) was enabled. Research conducted in several scientific areas has shown that the GP-GPU approach is very powerful and offers a considerable improvement in system performance at a low cost [19]. Furthermore, market leading smartphones have sophisticated GP-GPUs, and high-performance GP-GPU

DOI:10.15774/PPKE.ITK.2015.010

clusters are already available. Consequently, complex signal processing tasks can be offloaded to these devices. With these powerful MPAs the relatively high and variable computational complexity algorithms could be solved for real-time applications or they could speed-up the time of critical simulations.

The trend of using MPAs in several heavy signal processing tasks is visible. Computationally heavy signal processing algorithms like detection [20], [21], [22], [23], [24], [25], [26], decoding [27], [28] and precoding [29] are efficiently mapped on to GP-GPUs. Moreover, several GP-GPU based communication systems have been proposed in [30, 31]. An FPGA implementation of a variant of the LLL algorithm, the Clarkson's algorithm, is presented in [32]. In [33], a hardware-efficient Very Large Scale Integration (VLSI) architecture of the LLL algorithm is implemented, which is used for channel equalization in MIMO wireless communications. In [34], Xilinx FPGA is used for implementing LR-aided detectors, whereas [35] uses an efficient VLSI design based on a pipelined architecture.

The underlying architecture is seriously influencing the processing time and the quality of the results. Since the existing algorithms are mostly sequential, it is necessary to fundamentally redesign the existing algorithms in order to achieve peak performance with the new MPAs. By using these powerful devices new limits are reached, so in this thesis my goal is twofold: (i) to design efficient and highly parallel algorithms that solve the high complexity ML detection problem and (ii) to design and implement highly parallel preconditioning algorithms, such as lattice reduction algorithms, that enable afterwards the use of low complexity signal processing algorithms, achieving a near-optimal system performance.

1.2 Thesis outline

The thesis is organized as follows. Chapter 2 discusses the Flynn's taxonomy of parallel architectures and gives a brief overview of the most important parallel programming models. Chapter 3 introduces the MIMO system model considered throughout the thesis and shows how MIMO can increase the spectral efficiency. Chapter 4 gives a comprehensive overview of MIMO detection methods and presents the newly introduced detection algorithm. Section 4.2 gives a classification of the MIMO detectors. Section 4.3 presents two fundamental low-complexity linear detectors and their performance is compared. Section 4.4 discusses the successive interference detection concept and gives a brief overview on the importance of detection ordering based on several metrics. Section 4.5 gives a theoretical investigation on the optimal ML detection. Section 4.5 describes

DOI:10.15774/PPKE.ITK.2015.010

the SD algorithm together with a complexity analysis based on statistical methods, and the automatic SD is presented that gives the lower bound of node expansions during detection. Section 4.7 presents the most important non-ML tree-search based detectors. The presented algorithms served as a good basis to start the design of the PSD algorithm.

Section 4.8 introduces the new PSD algorithm. The precise details of **Thesis group I. and II.** are found in this section. The key concepts of the PSD algorithm are given in Sec. 4.8.1, its general description, together with an algorithmic comparison with the SD algorithm, is provided in Sec. 4.8.2, forming **Thesis I.a**. Parallel building blocks are designed in Sec. 4.8.3 for every stage of the PSD algorithm which facilitates the mapping to different parallel architectures, forming **Thesis I.b**. In Sec. 4.8.4 several levels of parallelism are identified and it is showed how CUDA kernels detect several symbol vectors simultaneously. Two computing load distribution strategies are presented and their application to a multi-stream GP-GPU environment is discussed. The achieved results in this section define **Thesis I.c**. Section 4.8.5 evaluates the performance of the PSD algorithm proposed by giving simulation results on the average detection throughput achieved, showing the distribution of work over the threads available and gives a comprehensive comparison with the results published in the literature. In Sec. 4.8.5.2 the effects of symbol ordering on the PSD algorithm are analyzed and the achieved performance enhancement is presented, forming **Thesis II.a**. Finally, Sec. 4.9 concludes the main results of this chapter.

Chapter 5 introduces LR and its applicability to MIMO systems, and new LR algorithms and their efficient mapping to parallel architectures are discussed. After giving the most important definitions, structures and limits in Sec. 5.2, the three fundamental LR algorithms are presented in Sec. 5.3. Section 5.4 shows how LR is applied to MIMO detection and multiple-input single-output (MISO) precoding, and the BER performance of low-complexity methods used in conjunction with LR is presented. Section 5.5 presents two parallelization strategies. The redesigned LR algorithms heavily rely on these methods.

Section 5.6 introduces my research in the field of parallel LR algorithms and their mapping to multi-core and many-core architectures. The precise details of **Thesis group III.** are found in this section. Section 5.6.1 presents the CR-AS-LLL algorithm and its mapping on to a GP-GPU architecture is given, forming **Thesis III.a**. In Sec. 5.6.2 it is shown how the block concept is applied in the case of large matrices and the resulting MB-LLL algorithm mapping is also discussed, **Thesis III.b.** is defined based on these results.

DOI:10.15774/PPKE.ITK.2015.010

In Sec. 5.6.3 the complexity of the MB-LLL is further reduced, and the CR-MB-LLL is mapped to a heterogeneous platform where the efficient work scheduling of the CPU and GP-GPU is also discussed, **Thesis III.c.** is defined based on these results. Section 5.6.4 evaluates the performance of the new parallel algorithms on different architectures.

Finally, Chapter 6 presents the summary of the main results of my thesis.

DOI:10.15774/PPKE.ITK.2015.010

Chapter 2

High-performance computing architectures and programming models

2.1 Flynn's taxonomy of parallel architectures

Parallel architectures are playing a prominent role in nowadays computing challenges. Modern supercomputers are built on various parallel processing units, thus, the classification of these architectures helps to get a better insight into the similarities and differences of the parallel architectures. When mapping an algorithm to a parallel architecture, or designing an application for a cluster, several levels of parallelism have to be identified and exploited. Instruction level parallelism is available when multiple operations can be executed concurrently. Loop level parallelism can be achieved when there is no data dependency between consecutive loop iterations. Procedure level parallelism can be achieved with the domain decomposition or the functional decomposition of the computational work. High-level parallelism is available when running multiple independent programs concurrently.

Flynn in [36], [37], [38] developed a methodology to classify general forms of parallel architectures along the two independent dimensions of *instruction stream* and *data stream*. Stream is defined as a sequence of instructions or data operated on by the computing architecture. The flow of instructions from the main memory to the CPU is the instruction stream, while the bidirectional flow of operands from the main memory, I/O ports to the CPU registers is the data stream. Flynn's taxonomy is a categorization of forms of parallel computer architectures. The most familiar parallel architectures are

DOI:10.15774/PPKE.ITK.2015.010

categorized in four classes: (i) *single instruction, single data* (SISD) stream, (ii) *single instruction, multiple data* (SIMD) stream, (iii) *multiple instruction, single data* (MISD) stream and (iv) *multiple instruction, multiple data* (MIMD) stream.

The SISD class is implemented by traditional uniprocessors, i.e., CPUs containing only a single processing element (PE). During one clock cycle only one instruction stream is being processed and the input is provided by a single data stream. The concurrency achieved in these processors is achieved with the pipelining technique. The processing of an instruction is composed of different phases: (i) instruction fetch, (ii) decoding of the instruction, (iii) data or register access, (iv) execution of the operation, and (v) result storage. Since these phases are independent, the pipelining enables the overlapping of the different operations. However, the correct result is achieved if each instruction is completed in sequence. Scalar processors process a maximum of one instruction per cycle and execute a maximum of one operation per cycle. The next instruction is processed only after the previous instruction is completed and its results are stored. Superscalar processors decode multiple instructions in a cycle and use multiple functional units and a dynamic scheduler to process multiple instructions per cycle. This is done transparently to the user by analyzing multiple instructions from the instruction stream.

In the SIMD class the same instruction is executed by multiple PEs at every clock cycle, but each PE operates on a different data element. This is realized by broadcasting the same instruction from the control unit to every PE. The next instruction is only processed after every PE completed its work. This simultaneous execution is referred to as lock-stepping execution. The efficient data transfer from the main memory is achieved by dividing the main memory into several modules. Problems with a high degree of regularity, such as different matrix, vector operations, are best suited for these type of architectures. The common use of regular structures in many scientific problems makes SIMD architectures very effective in solving these problems. The SIMD class is implemented by array and vector processors.

An array processor consists of interconnected PEs with each having its own local memory space. There is possibility to access the global memory or the local memory of another PE, however, this has a high latency. By creating local interconnections of the PEs, a new level of shared memory can be defined that has a reduced latency, but its efficient use is implemented by simple and regular memory access patterns. Individual PEs could conditionally disable instructions in case of branching, entering in idle state.

A vector processor consists of a single processor that references a single global memory

DOI:10.15774/PPKE.ITK.2015.010

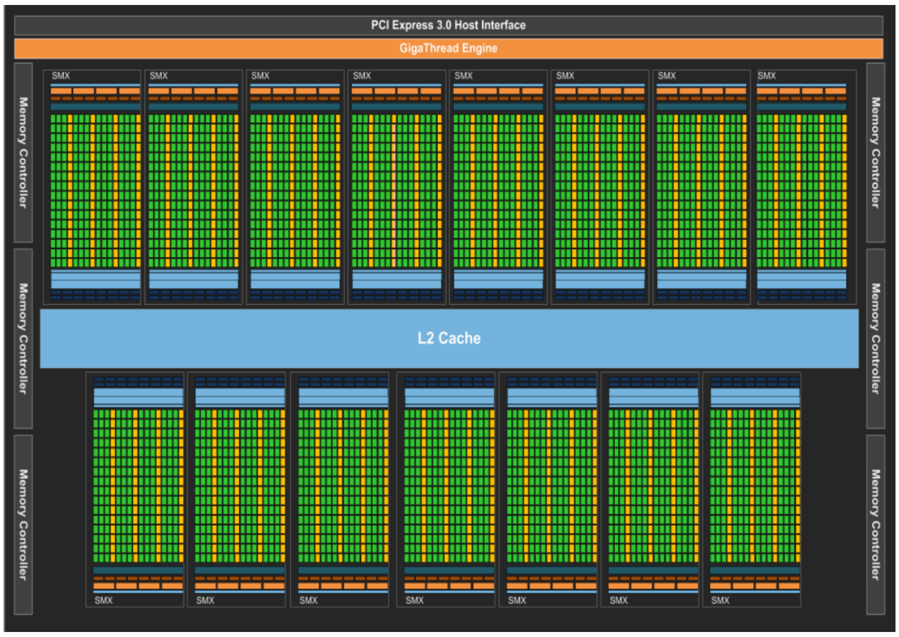


Figure 2.1: The block diagram of the GK110 Kepler architecture.

Reprinted from [39].

space and has special functional units that operate specifically on vectors. The vector processor shows high similarities with SISD processors except that vector processors can treat data sequences, referred to as vectors, through their function units as a single entity. The sophisticated pipelining techniques, the high clock rate, and the efficient data transition to the input vector of these functional units achieve a significant throughput increase compared to scalar function units. An efficient algorithm mapping can hide the latency of the data loads and stores between the vector registers and main memory with computations on values loaded in the registers.

GP-GPUs employ the *single instruction, multiple threads* (SIMT) multi-threaded model. An application launches a number of threads that all enter the same program together, and those threads get dynamically scheduled onto a SIMD datapath such that threads that are executing the same instruction get scheduled concurrently. Figure 2.2 shows the Kepler GK110 architecture block diagram. The Kepler GK110 and GK210 implementations include 15 streaming multiprocessor (SMX) units and six 64-bit memory controllers.

Figure 2.2 shows the block diagram of the GK110 SMX architecture. Based on [39] the following components and features are available in the GK110/GK210 SMX units:

- each of the Kepler GK110/210 SMX has 192 single-precision, 64 double-precision CUDA cores, 32 special function units, and 32 load/store units;

DOI:10.15774/PPKE.ITK.2015.010

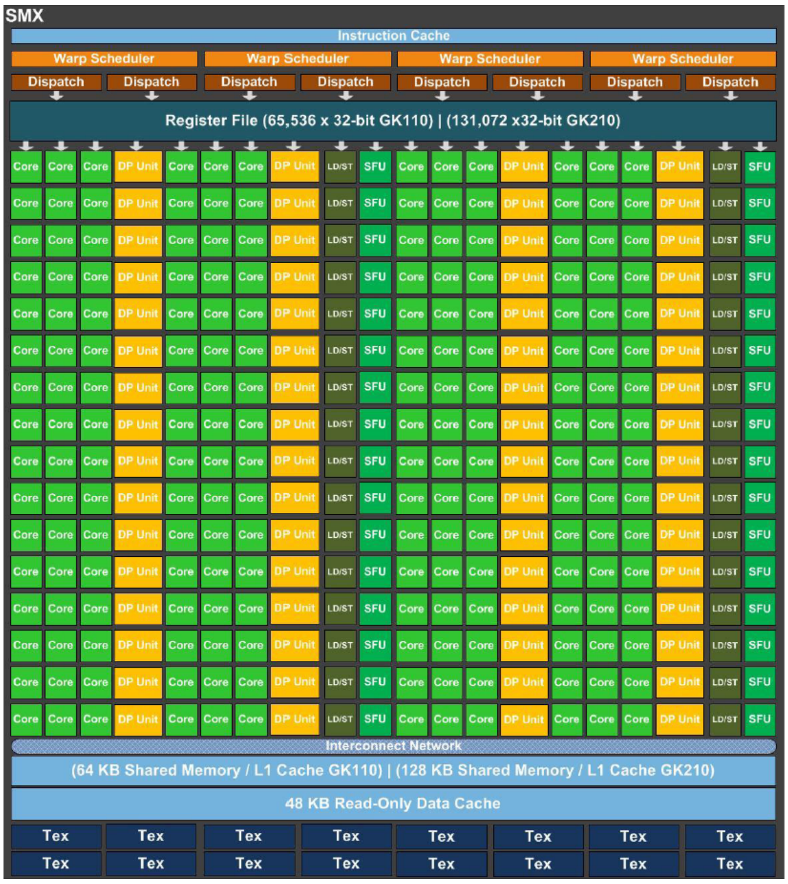

SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit GK110) | (131,072 x32-bit GK210)
Core
DP Unit
LD/ST
SFU
Interconnect Network
(64 KB Shared Memory / L1 Cache GK110) | (128 KB Shared Memory / L1 Cache GK210)
48 KB Read-Only Data Cache
Tex

Figure 2.2: The GK110/GK210 Kepler streaming multiprocessor (SMX) unit architecture.

Reprinted from [39].

- the SMX schedules threads in groups of 32 parallel threads called warps;
- each SMX has four warp schedulers and eight instruction dispatch units, allowing four warps to be issued and executed concurrently;
- Kepler's quad warp scheduler selects four warps, and two independent instructions per warp can be dispatched each cycle.

In the MISD class a single data stream is fed into multiple processing units and each PE has its own instruction stream. Thus, the number of control units is equal to the number of PEs. Only a few implementations of this class of parallel computer have ever existed. Problems that could be efficiently solved on these architectures are filter banks, i.e., multiple frequency filters operating on a single signal stream, or cryptography algorithms attempting to crack a single coded message.

Processors implementing the MIMD stream are said to be the parallel computers. Every PE has the possibility of executing a different instruction stream that is handled by a separate control unit and a data stream is available for every PE. The execution

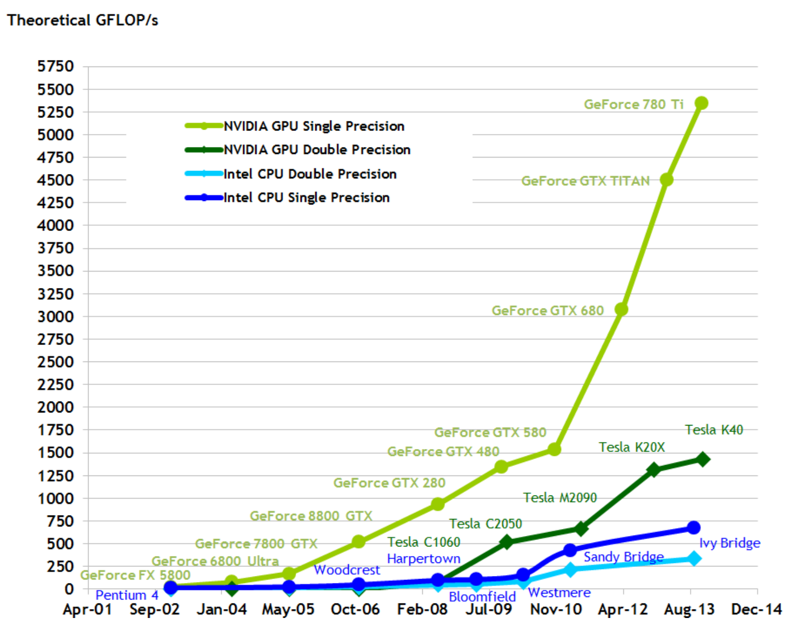


Figure 2.3: The evolution of theoretical floating-point operations per second (FLOP/s) for CPU and GP-GPU architectures.

Reprinted from [40].

of different tasks can be synchronous or asynchronous (they are not lock-stepped as in SIMD processors), they can start or finish at different times. Usually, the solution of a problem requires the cooperation of the independently running processors, and this co-operation is realized through the existing memory hierarchy. By using shared memory or distributed shared memory the problem of memory consistency and cache coherency has to be solved. With the use of thread synchronization, atomic operations, critical sections the programmer can overcome these problems, however, there are problems that can be solved exclusively through hardware techniques. Todays multi-threaded processors, multi-core and multiple multi-core processor systems implement the MIMD class.

The above discussion showed that modern CPUs fall in the MIMD class, while the GP-GPUs implement a more flexible SIMD stream. Figure 2.3 gives an overview of the theoretical floating-point operations per second (FLOP/s) for different CPU and GP-GPU architectures. It can be seen that there is a huge potential in the GP-GPU architecture because the achievable theoretical FLOP/s is many times higher compared to the CPU architecture. However, in order to exploit this huge potential, the algorithm design has to take in consideration the limitations imposed by the SIMT architecture that is often a challenging task.

DOI:10.15774/PPKE.ITK.2015.010

2.2 Overview of parallel programming models

A parallel programming model can be regarded as an abstraction of a system architecture. Thus, the program written based on a parallel programming model can be compiled and executed for the underlying parallel architecture. Because of the wide range of existing parallel systems and architectures different parallel programming models were created. A parallel programming model is general if the mapping of a wide range of problems for a variety of different architectures is efficient. The underlying memory architecture highly influences what parallel programming model is going to give a good abstraction. The three most fundamental parallel computer memory architectures are: *shared memory*, *distributed memory* and *hybrid distributed-shared memory*.

In *shared memory* parallel computers the processors share the same memory resources, thus, there is only a single memory address space. Cache coherency is available, meaning that every memory update is visible to every processor. Shared memory computers can be further classified as: uniform memory access (UMA) and non-uniform memory access (NUMA) computers. The UMA model is implemented by symmetric multiprocessor (SMP) computers, where identical processors are placed near the shared memory, thus, the memory access time is equal to every processor. The NUMA model is implemented by linking several SMPs and every SMP can directly access the memory of another SMP. However, in this case the memory access time is different for processors lying in different SMPs. The advantages of this model are the global address space and fast data-sharing. The main disadvantages are the lack of scalability and the need of synchronization in order to ensure data correctness.

The *distributed memory* architecture model is encountered in networked or distributed environments such as clusters or grids of computers. In this case every processor operates independently and has its own local memory, thus, there is no global memory address space. No cache coherency is present, so the memory change of one processor does not effect the memory of other processors. The programmer's task is to manage the data communication and the synchronization of different tasks. The advantage of this approach is that each processor can rapidly access its own memory because no overhead is required to assure global cache coherency, and the memory can easily scale with the number of processors. Among the disadvantages of this model is the non-uniform memory access time and the burden to manage the data communication between processors.

Nowadays, systems implementing the hybrid distributed-shared memory model are the largest and fastest computers. In this model multiple shared memory processors are

DOI:10.15774/PPKE.ITK.2015.010

connected through a network resulting in a distributed architecture. The hybrid model can be further improved by adding GP-GPUs to the shared memory architecture. This is a very powerful model, however, the programming complexity is significantly increased. Thus, in order to exploit the benefits of this hybrid model efficient algorithm mapping, computational load balancing and communication organization has to be designed.

The two predominant parallel programming models using pure shared or distributed memory approach are the *Open Multi-Processing* (OpenMP) [41] for shared memory and the *Message Passing Interface* (MPI) [42] for distributed memory systems. In addition, with the advent of the massively parallel many-core GP-GPUs new programming models like CUDA [40] and *Open Computing Language* (OpenCL) [43] were introduced.

OpenMP is a high-level abstraction of shared memory systems. OpenMP is implemented as a combination of a set of compiler directives, pragmas, and a runtime providing both management of the thread pool and a set of library routines. With the help of the directives the compiler is instructed to spawn new threads, to perform shared memory operations, to perform synchronization operations, etc. The fork-join threading model is used to switch between sequential and parallel operation modes. During a fork operation a thread splits into several threads, thus, the execution branches off in parallel, and after their task is completed they are joined resuming to sequential execution.

MPI is a parallel programming model for distributed memory systems and facilitates the communication between processes by interchanging messages. The communication is cooperative and occurs only when the first process executes a send operation and the second process executes a receive operation. The task of the programmer is to manage the workload by defining what tasks are to be performed by each process.

Since massively parallel many-core GP-GPUs are playing an important role in this thesis, in Sec. 2.2.1 the most important definitions and components of CUDA are presented.

2.2.1 The CUDA programming model

The programming of GP-GPU devices has became popular since Nvidia published the CUDA parallel programming model. Traditional CPUs are able to execute only a few threads, but with relatively high clock rate. In contrast, GP-GPUs have a parallel architecture that support the execution of thousands of threads with a lower clock-rate.

An extensive description of CUDA programming and optimization techniques can be found in [40]. The main entry points of GP-GPU programs are referred to as *kernels*.

DOI:10.15774/PPKE.ITK.2015.010

These kernels are executed N times in parallel by N different *CUDA threads*. CUDA threads are grouped in *thread blocks* (TBs). The number of threads in a TB is limited, however, multiple equally-shaped TBs can be launched simultaneously. A *grid* is a collection of TBs. The threads in the TB or the TBs in the grid can have a one-dimensional, two-dimensional or three-dimensional ordering.

The cooperation between the threads is realized with the help of multiple memory spaces that differ in size, latency and visibility. In CUDA the following hierarchy of memory levels are defined: (i) *private*, (ii) *shared*, (iii) *global*, (iv) *constant* and (v) *texture* memory.

In some cases specific threads have to wait for the results generated by other threads. Therefore, threads within a TB can be *synchronized*. In order to continue the execution, each thread has to reach the synchronization point. There is no similar mechanism to synchronize TBs in a grid. When a kernel finishes its execution it can be regarded as a global synchronization of the TBs.

The Nvidia GP-GPU architecture is built around a scalable array of multithreaded streaming multiprocessors (SMs). The TBs of the grid are distributed to the SMs with available execution capacity by the grid management unit. An important metric of the SMs usage is *occupancy*. The occupancy metric of each SM is defined as the number of active threads divided by the maximum number of threads. Groups of 32 threads, called *warps*, are executed together. The maximum number of TBs running simultaneously on a multiprocessor is limited by the maximum number of warps or registers, or by the amount of shared memory used by the kernel.

In order to concurrently execute hundreds of threads, the SMs employ a SIMT architecture. A warp executes one common instruction at a time. In the case of branching, the warp will serially execute each branch path. In order to achieve full efficiency, divergence should be avoided. Applications manage concurrency through *streams*. A stream is a sequence of commands that are executed in order. Different streams may execute their commands out of order with respect to one another or concurrently. Thus, launching multiple kernels on different streams is also possible. This can be very efficient when kernels can be launched independently from each other.

DOI:10.15774/PPKE.ITK.2015.010

Chapter 3

Overview of MIMO communications

3.1 Benefits of MIMO systems

The key feature of MIMO systems is the ability to turn multipath propagation, traditionally a pitfall of wireless transmissions, into a benefit for the user, thus, the performance of wireless systems is improved by orders of magnitude at no cost of extra spectrum use. The performance enhancements of MIMO systems are achieved through *array gain, spatial diversity gain*, and *spatial multiplexing gain.*

Array gain is defined as the efficient combination of the transmitted or received signals that result in the increase of the signal-to-noise ratio (SNR). As a result the noise resistance and the coverage of a wireless network is improved. Array gain is exploited by methods like: selection combining, maximal-ratio combining, equal-gain combining as discussed in [44].

In case of spatial diversity different representations of the same data stream (by means of coding) is transmitted on different parallel transmit branches, i.e., it introduces controlled redundancy in both space and time. At the receiver side independent copies of the same transmitted signal are encountered because of the rich scattering environment. If a signal path experiences a deep fade the original signal still can be restored, thus, the probability of error is minimized. The number of copies received is referred to as the diversity order. A MIMO channel with n transmit antennas and m receive antennas potentially offers $n \cdot m$ independent fading links, and hence a spatial diversity order of $n \cdot m$. Space-time coding (STC) was introduced to support transmit diversity in [45], [46], [47], [48].

DOI:10.15774/PPKE.ITK.2015.010

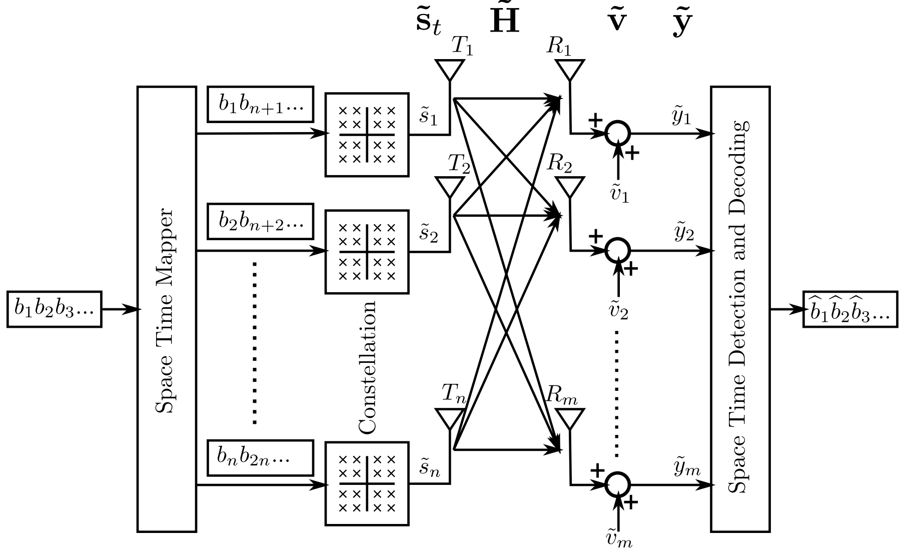


Figure 3.1: MIMO system architecture with n transmitter and m receiver antennas.

Spatial multiplexing focuses on maximizing the capacity of a radio link by transmitting independent data streams on different transmit branches simultaneously and within the same frequency band. In a rich scattering environment, the receiver can separate the data streams and since each data stream experiences the same channel quality as a single-input single-output (SISO) system, the capacity is increased by a multiplicative factor equal to the number of streams. Spatial multiplexing was exploited in the following works [129], [49], [8].

In hybrid MIMO configurations the benefits of spatial multiplexing and space-time coding schemes are combined. This is achieved by dividing the transmit antennas into sub-groups where each sub-group is space-time coded independently. At the receiver group spatial filters are used followed by space-time decoding. For a few good examples refer to [50], [51] and [52].

3.2 MIMO system model

A MIMO system consists of n transmit and m receive antennas as shown in Fig. 3.1. The transmit antennas are sending a complex signal vector $\tilde{\mathbf{s}}_t$ of size n during one symbol period. The received complex symbol vector $\tilde{\mathbf{y}} = (\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_m)^T$ is expressed as

$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\tilde{\mathbf{s}}_t + \tilde{\mathbf{v}} \tag{3.1}$$

where $\tilde{\mathbf{v}} = (\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_m)^T$ is the additive channel noise and the superposition of the transmitted symbols is modeled by the channel matrix $\tilde{\mathbf{H}} \in \mathbb{C}^{m \times n}$. The statistical prop-

DOI:10.15774/PPKE.ITK.2015.010

erties of the MIMO system model are summarized as follows:

- the noise vector is modeled as $\tilde{\mathbf{v}} \sim \mathcal{CN}(0, \mathbf{K})$ zero-mean, circularly-symmetric jointly-Gaussian complex random vector, with covariance matrix $\mathbf{K} = \sigma^2 \mathbf{I}_m$ and uncorrelated components are assumed,
- the elements $\tilde{h}_{ij} \sim \mathcal{CN}(0, 1)$ of the channel matrix $\tilde{\mathbf{H}}$ are assumed to be i.i.d. zero-mean, complex circularly-symmetric Gaussian variables with unit variance,
- the transmitted symbols are modeled as independent and identically distributed (i.i.d.) random variables which are uniformly distributed over a symbol set $\tilde{\Omega}$, moreover it is assumed that the symbol set is centered at zero $E\{\tilde{\mathbf{s}}_t\} = 0$ and the average transmit power of each antenna is normalized to one, i.e. $E\left\{\tilde{\mathbf{s}}_t \tilde{\mathbf{s}}_t^H\right\} = \mathbf{I}_n$.

The elements of the symbol set $\tilde{\Omega}$ are drawn from an M_c-ary Quadrature Amplitude Modulation (M-QAM) constellation usually employed in MIMO communications, where M_c stands for the number of constellation points and typically $\mathrm{M}_c = 4, 16, 64$. As a result the elements of the symbol set are defined as

$$\tilde{\Omega} = \{a + bi, a, b \in \mathbb{D}\}, \tag{3.2}$$

where

$$\mathbb{D} = \left\{\pm\frac{1}{2}a, \pm\frac{3}{2}a, \ldots, \pm\frac{\sqrt{\mathrm{M}_c} - 1}{2}a\right\} \text{ and } a = \sqrt{\frac{6}{\mathrm{M}_c - 1}}. \tag{3.3}$$

The parameter a is used for normalizing the power of the transmit signals to 1. The components of $\tilde{\mathbf{s}}_t = (\tilde{s}_1, \tilde{s}_2, ..., \tilde{s}_n)^T$ are drawn from $\tilde{\Omega}$, i.e. $\tilde{\mathbf{s}}_t \in \tilde{\Omega}^n$, and carry $\log_2 \mathrm{M}_c$ Gray-encoded bits each. Block-fading channel is assumed where the channel matrix remains quasi-static within a fading block, but it is independent between successive fading blocks. Furthermore, it is assumed that the channel matrix is known by the receiver, it has been estimated before without errors.

The original complex representation of the system model, presented in Eq. 3.1, can be transformed into an equivalent real-valued model at the cost of increasing its dimension, as follows:

$$\mathbf{y} = \mathbf{H}\mathbf{s}_t + \mathbf{v}, \tag{3.4}$$

DOI:10.15774/PPKE.ITK.2015.010

where

$$\mathbf{y} = \begin{pmatrix} \Re(\tilde{\mathbf{y}}) \\ \Im(\tilde{\mathbf{y}}) \end{pmatrix}_{M\times 1}, \mathbf{s}_t = \begin{pmatrix} \Re(\tilde{\mathbf{s}}_t) \\ \Im(\tilde{\mathbf{s}}_t) \end{pmatrix}_{N\times 1}, \mathbf{v} = \begin{pmatrix} \Re(\tilde{\mathbf{v}}) \\ \Im(\tilde{\mathbf{v}}) \end{pmatrix}_{M\times 1}, \mathbf{H} = \begin{pmatrix} \Re(\tilde{\mathbf{H}}) & -\Im(\tilde{\mathbf{H}}) \\ \Im(\tilde{\mathbf{H}}) & \Re(\tilde{\mathbf{H}}) \end{pmatrix}_{M\times N} \tag{3.5}$$

and where $M = 2m$ and $N = 2n$. In the transformed MIMO system model $\mathbf{y}, \mathbf{H}, \mathbf{s}_t$ are all real-valued quantities and $\Omega = \mathbb{D}$ is a real-valued signal set.

3.3 MIMO capacity

The notion of channel capacity was introduced by Shannon in [53]. The capacity of a channel, denoted by C, is the maximum rate at which reliable communication can be performed, and is equal to the maximum mutual information between the channel input and output vectors. Shannon proved two fundamental theorems: (i) for any rate $R < C$ and any desired non-zero probability of error P_e there exists a rate R code that achieves P_e and (ii) the error probability of rates $R > C$ higher then the channel capacity is bounded away from zero. As a result, the channel capacity is a fundamental limit of communication systems.

Several channel capacity definitions are available in the literature depending on: (i) what is known about the state of the channel, referred to as channel state information (CSI), or the distribution of the channel, referred to as channel distribution information (CDI), and the time scale of the fading process. For a time-varying channel where CSI is available at both the transmitter and receiver, namely the channel matrix $\tilde{\mathbf{H}}$ is known, the transmitter can adapt its rate or power based on the CSI. In this case the *ergodic capacity* is defined as the maximum mutual information averaged over all the channel states. Ergodic capacity is a relevant metric for quickly varying channels, since the channel experiences all possible channel states over the duration of a codeword. In case of perfect CSI at both the transmitter and receiver the *outage capacity* is defined as the maximum rate of reliable communication at a certain outage probability. Outage capacity requires a fixed data rate in all non-outage channel states and no data is transmitted when the channel is in outage since the transmitter knows this information. Outage capacity is the appropriate capacity metric in slowly varying channels, where the channel coherence time exceeds the duration of a codeword, thus each codeword is affected by only one channel realization.

In the following the capacity of single-user MIMO channel is considered for the case

DOI:10.15774/PPKE.ITK.2015.010

when: (i) CSI is available at the transmitter (CSIT) and receiver (CSIR) and the channel is constant, and (ii) CDIT and CSIR is available and fading channel is assumed.

When CSIT is available the estimated distribution or channel state is sent to the transmitter through a feedback channel. If the transmitter adapts to these time-varying short-term channel statistics then capacity is increased relative to the transmission strategy associated with just the long-term channel statistics. The average transmit power is constrained across all transmit antennas as $E[\tilde{\mathbf{s}}_t^H \tilde{\mathbf{s}}_t] \leq P$. When the channel is constant, and CSIT, CSIR is available, the capacity is defined as

$$C = \max_{\tilde{\mathbf{Q}}: tr(\tilde{\mathbf{Q}})=P} \log\det(\mathbf{I}_m + \tilde{\mathbf{H}}\tilde{\mathbf{Q}}\tilde{\mathbf{H}}^H) \tag{3.6}$$

where $\tilde{\mathbf{Q}}$ is the input covariance matrix, which is $n \times n$ positive semi-definite complex matrix.

With the help of the singular value decomposition (SVD) the channel matrix can be factorized as

$$\tilde{\mathbf{H}} = \tilde{\mathbf{U}}\mathbf{\Sigma}\tilde{\mathbf{V}}^H, \tag{3.7}$$

where $\tilde{\mathbf{U}}$ is $\mathbb{C}^{m\times m}$ unitary matrix, $\mathbf{\Sigma}$ is $\mathbb{R}^{m\times n}$ diagonal matrix with real non-negative entries, and $\tilde{\mathbf{V}}$ is $\mathbb{C}^{n\times n}$ unitary matrix. The diagonal elements of matrix $\mathbf{\Sigma}$, denoted by σ_i, are the singular values of $\tilde{\mathbf{H}}$ and a descending order is assumed $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(n,m)}$. The matrix $\tilde{\mathbf{H}}$ has exactly r positive singular values, where r is the rank of $\tilde{\mathbf{H}}$, and $r \leq \min(n,m)$. In [44], [8] it was shown that the SVD can convert channel $\tilde{\mathbf{H}}$ into $\min(n,m)$ parallel, noninterfering SISO channels after precoding the input, i.e. $\bar{\mathbf{s}}_t = \tilde{\mathbf{V}}\tilde{\mathbf{s}}_t$, and the received vector is multiplied by matrix $\tilde{\mathbf{U}}^H$, resulting in $\bar{\mathbf{y}} = \tilde{\mathbf{U}}^H\tilde{\mathbf{y}}$. The system model is transformed as

$$\begin{aligned} \bar{\mathbf{y}} &= \tilde{\mathbf{U}}^H(\tilde{\mathbf{H}}\bar{\mathbf{s}}_t + \tilde{\mathbf{n}}) \\ &= \tilde{\mathbf{U}}^H(\tilde{\mathbf{U}}\mathbf{\Sigma}\tilde{\mathbf{V}}^H(\tilde{\mathbf{V}}\tilde{\mathbf{s}}_t) + \tilde{\mathbf{n}}) \\ &= \mathbf{\Sigma}\tilde{\mathbf{s}}_t + \tilde{\mathbf{U}}^H\tilde{\mathbf{n}} \\ &= \mathbf{\Sigma}\tilde{\mathbf{s}}_t + \bar{\mathbf{n}} \end{aligned} \tag{3.8}$$

where $\bar{\mathbf{n}} = \tilde{\mathbf{U}}^H\tilde{\mathbf{n}}$. Since $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are unitary $\tilde{\mathbf{n}}$ and $\bar{\mathbf{n}}$ have the same distribution, and the transformations are power preserving $E[||\tilde{\mathbf{s}}_t||^2] = E[||\bar{\mathbf{s}}_t||^2]$. Since $\mathbf{\Sigma}$ is diagonal and the singular values σ_i are strictly positive, the non-interfering channels are described as

$$\bar{\mathbf{y}}_i = \sigma_i\tilde{\mathbf{s}}_{t_i} + \bar{\mathbf{n}}_i, \quad \text{for } i = 1, \ldots, \min(n,m). \tag{3.9}$$

DOI:10.15774/PPKE.ITK.2015.010

Note, that the above is possible only if the transmitter knows the channel, because the precoding requires the exact channel matrix state information. The water-filling algorithm [54] can be used to optimally allocate power over the different quality parallel channels, leading to the following allocation:

$$P_i = \max\left(\mu - \frac{1}{\sigma_i^2}, 0\right), \quad 1 \le i \le r, \tag{3.10}$$

where P_i is the power of $\tilde{\mathbf{s}}_{t_i}$, and the waterfill level μ is chosen such that $\sum_{i=1}^{r} P_i = P$. Thus, the covariance that achieves the maximum capacity defined in Eq. 3.6 is $\tilde{\mathbf{Q}} = \tilde{\mathbf{V}}\mathbf{P}\tilde{\mathbf{V}}^H$, where $\mathbf{P} \in \mathbb{R}^{n\times n}$ and $\mathbf{P} = diag(P_1, \ldots, P_r, 0, \ldots, 0)$. The resulting capacity is given by

$$C = \sum_{i=1}^{r} \max(\log(\mu\sigma_i^2), 0). \tag{3.11}$$

The constant channel model is easy to analyze from a mathematical point of view, however, wireless channels are time-varying. In this case a more common assumption is that the receiver is able to correctly estimate the channel state, but only the CDI is fed back to the transmitter. The perfect CSIR and CDIT model is motivated by the scenario where the channel state can be accurately tracked at the receiver and the statistical channel model at the transmitter is based on channel distribution information fed back from the receiver. The channel coefficients are typically assumed to be jointly Gaussian, so the channel distribution is specified by the channel mean and covariance matrices. When the transmitter has knowledge only on the channel distribution the precoding is not possible, thus an optimal strategy is to maintain a fixed-rate transmission that is optimized with the respect to the CDI. In [8] and [55] it was shown that the optimal transmit strategy is to allocate equal power in every spatial direction, thus the optimum input covariance matrix that maximizes the ergodic capacity is $\mathbf{Q} = \frac{P}{n}\mathbf{I}_n$. As a result, the ergodic capacity is defined as

$$C = E_{\mathbf{H}}\left[\log\det\left(\mathbf{I}_m + \frac{P}{n}\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H\right)\right]. \tag{3.12}$$

A useful approach to get insight of the multiplexing gain of MIMO systems is to analyze the asymptotic behavior of the ergodic capacity as either the SNR or the number of antennas are increasing. If n and m are fixed and SNR is taken to infinity, the capacity grows as $C \approx \min(n, m)\log_2 P + O(1)$. Consequently, a 3 dB increase in the SNR leads to an increase of $\min(n, m)$ bps/Hz in spectral efficiency, thus the multiplexing gain of a MIMO system is $\min(n, m)$ times better compared to the SISO case.

DOI:10.15774/PPKE.ITK.2015.010

Chapter 4

MIMO detection methods and algorithms

4.1 Introduction

In this chapter the focus is on detection methods applied on MIMO systems when spatial multiplexing is used. The complexity of detection algorithms depends on many factors, such as antenna configuration, modulation order, channel, coding, etc. Several detection techniques are investigated such as: (i) the linear detectors, (ii) non-linear detectors based on the successive interference cancellation, (iii) tree-search based detectors and (iv) the ML detector offering the best BER performance.

The ML detector offers the best BER performance, however, its exponential complexity is not suitable for real-time applications. The SD algorithm, discussed in Sec. 4.6.1, was proposed to significantly reduce the search space without degrading the BER performance of the ML detector. The main disadvantage of the ML detector is the exponentially growing complexity with both the number of elements in the signal set and the number of antennas.

In non-optimal detectors the complexity of the SD algorithm is reduced by introducing some *approximations* such as (i) early termination of the search, (ii) introducing constraints on the maximum number of nodes that the detector algorithm is allowed to visit or (iii) on the run-time of the detector. Each of these strategies introduces some approximations which prevents these detectors from achieving ML performance. For non-optimal detectors a trade-off has to be made between computational complexity and the quality of detection. Papers [56], [20], [57], [58], [59] and [60] focus on finding a near-ML solution with a significant decrease in the computational complexity. These

DOI:10.15774/PPKE.ITK.2015.010

detectors usually are used together with error correcting coding schemes or different preprocessing techniques in order to enhance the BER performance. Linear detectors based on Zero-Forcing, Minimum Mean Squared Error and Successive Interference Cancellation offer a low computational complexity. However, the BER performance achieved by these detectors is significantly worse than that of the ML and near-ML solutions.

The main results of this chapter are detailed in Sec. 4.8. The computationally intensive hard-output ML detection problem of spatial multiplexing MIMO systems is efficiently solved in the upcoming sections. The solution is a completely redesigned SD algorithm. The new PSD algorithm implements a novel hybrid tree search method where (i) the algorithm parallelism is assured by the efficient combination of depth-first search (DFS) and breadth-first search (BFS) algorithms and (ii) a path metric based parallel sorting is employed at each intermediate stage. The main advantages of the PSD algorithm are the ability to adjust both its memory requirements and the extent of its parallelism in order to fit a wide range of parallel architectures.

With the PSD algorithm the sequentiality of the SD is eliminated, thus, it can exploit the parallelism of the MPA platforms. In Sec. 4.8.3 mapping details for MPAs are provided by giving the schematics of the thread dependent, highly parallel building blocks of the algorithm. In order to achieve high-throughput, in Sec. 4.8.4 several levels of parallelism are introduced and different scheduling strategies are considered. Moreover, based on the building blocks a mapping of the PSD algorithm to a GP-GPU is performed. The performance of the proposed PSD algorithm is evaluated in Sec. 4.8.5 by giving simulation results on the average detection throughput achieved, showing the distribution of work over the available threads and a comprehensive comparison with the results published in the literature.

4.2 MIMO detectors classification

In Sec. 4.1 a brief overview was given on how MIMO can improve the overall performance of a radio link. In this section the most important detection methods are sorted based on (i) BER performance and (ii) algorithm family when *spatial multiplexing* is used as shown in Fig. 4.1.

The BER performance of the hard-output detector algorithms can be roughly categorized as ML and non-ML detectors. The exhaustive ML detector offers the best BER performance, however, its exponential complexity is not suitable for real-time applications. In case of the non-ML detectors a trade-off has to be made between the computational

DOI:10.15774/PPKE.ITK.2015.010

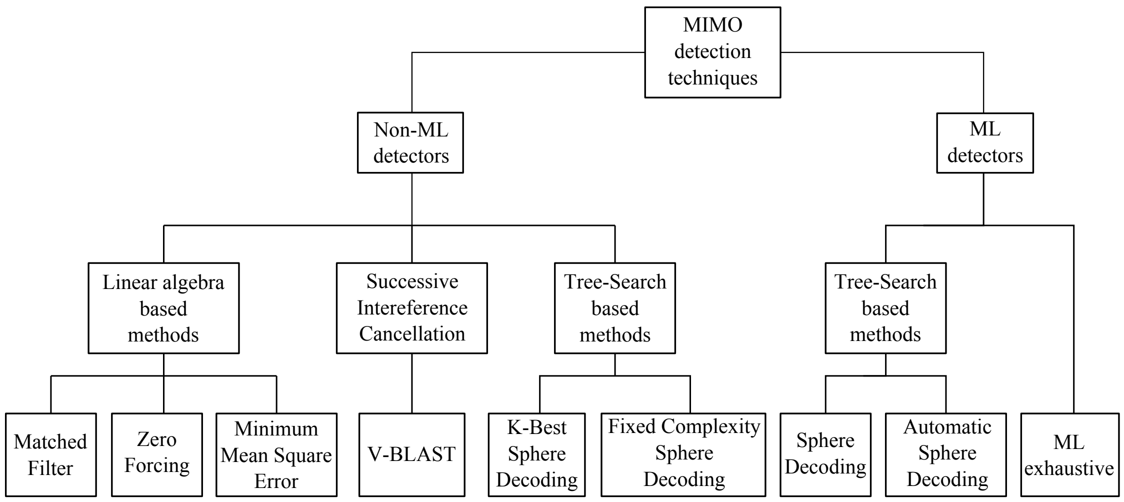

MIMO detection techniques
Non-ML detectors
ML detectors
Linear algebra based methods
Successive Intereference Cancellation
Tree-Search based methods
Tree-Search based methods
Matched Filter
Zero Forcing
Minimum Mean Square Error
V-BLAST
K-Best Sphere Decoding
Fixed Complexity Sphere Decoding
Sphere Decoding
Automatic Sphere Decoding
ML exhaustive

Figure 4.1: Classification of spatial multiplexing MIMO detection methods.

complexity and the quality of detection.

The methods used for detection are grouped as follows: (i) linear algebra, (ii) Successive Interference Cancellation (SIC) and (iii) tree-search based methods.

1. Methods based on linear algebra are usually computationally less complex, however, the BER performance is significantly reduced compared to optimal ML performance. In case of the linear methods the general approach is to find a weight matrix that is the solution of a minimization problem. After the weight matrix is found, a multiplication with the received symbol vector is performed followed by the clipping of the result. In Sec. 4.3 a brief overview is given on two fundametal linear detectors: the Zero-Forcing (ZF) and the Minimum Mean Square Error (MMSE) detector.

2. Methods based on SIC involve increased complexity computations resulting in the improvement of the BER performance. The increase in the computational complexity is motivated by finding the optimal ordering of the symbol detection, because in every iteration the Moore-Penrose pseudoinverse of the modified channel matrix has to be computed and a sorting has to be performed. Different ordering metrics are available throughout the literature. The rule of thumb is similar in this case as well, the higher the complexity a better performance is achieved. In Sec. 4.4 the concept of SIC is introduced and the ZF based ordering Vertical Bell Labs Layered Space-Time (V-BLAST) architecture is discussed.

3. As stated previously, the ML methods provide optimal ML performance, however, the complexity of the exhaustive ML search grows exponentially by increasing the

DOI:10.15774/PPKE.ITK.2015.010

number of antennas. To solve the problem the SD algorithm has been proposed that reduces significantly the search space of possible solutions while still providing the ML solution. For a few good examples refer to [61], [62] and [63]. The SD algorithm gives an efficient solution to the Integer Least Squares (ILS) problem and it turns out that there is an analogy between the search for the optimal solution and the bounded tree search methods. In Sec. 4.6 a detailed overview is given on the SD algorithm and it is discussed how the solution of the ILS problem can be transformed to a tree-search problem.

4. The complexity of SD algorithm can be further reduced by introducing some *approximations* such as (i) early termination of the tree-search, (ii) introducing constraints on the maximum number of nodes that the SD algorithm is allowed to visit or (iii) on the run-time of the detector. Each of these strategies introduces some approximations which prevents these detectors from achieving the ML performance. In case of the non-ML detectors a trade-off has to be made between the computational complexity and the quality of detection. Usually, when approximations are introduced some preprocessing methods are applied such as: matrix regularization, lattice reduction, ordering, improved decompositions. These methods increase the overall computational complexity, however, the benefit is significant and near-ML BER is achieved. Papers [20], [57], [56], [58], [59] and [60] focus on finding a near-ML solution with a significant decrease in computational complexity. In Sec. 4.7 the most effective non-ML tree-search based methods are discussed.

4.3 Linear detectors

4.3.1 Zero-forcing detection

The ZF detector aims to find the least-squares solution, i.e., it searches for the unconstrained vector $\tilde{\mathbf{s}} \in \mathbb{C}^n$ that minimizes the squared Euclidean distance to the received symbol vector $\tilde{\mathbf{y}}$ according to

$$\tilde{\mathbf{s}}_{ZF} = \arg\min_{\tilde{\mathbf{s}} \in \mathbb{C}^n} \|\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\tilde{\mathbf{s}}\|^2 \tag{4.1}$$

The problem looks very similar to the ML solution, however in this case the search space is not restricted to a finite alphabet. Consequently, in order to arrive to the final solution a clipping of the unconstrained solution $\hat{\mathbf{s}} = Q\{\tilde{\mathbf{s}}_{ZF}\}$ to a valid symbol vector is required.

DOI:10.15774/PPKE.ITK.2015.010

The optimization problem is solved by setting the derivative of the squared Euclidean distance to zero, as follows

$$\begin{aligned}\frac{\partial}{\partial\tilde{\mathbf{s}}}\|\tilde{\mathbf{y}}-\tilde{\mathbf{H}}\tilde{\mathbf{s}}\|^2 &= 0\\ \frac{\partial}{\partial\tilde{\mathbf{s}}}(\tilde{\mathbf{y}}-\tilde{\mathbf{H}}\tilde{\mathbf{s}})^H(\tilde{\mathbf{y}}-\tilde{\mathbf{H}}\tilde{\mathbf{s}}) &= 0\\ \frac{\partial}{\partial\tilde{\mathbf{s}}}(\tilde{\mathbf{y}}^H\tilde{\mathbf{y}}-\tilde{\mathbf{s}}^H\tilde{\mathbf{H}}^H\tilde{\mathbf{y}}-\tilde{\mathbf{y}}^H\tilde{\mathbf{H}}\tilde{\mathbf{s}}+\tilde{\mathbf{s}}^H\tilde{\mathbf{H}}^H\tilde{\mathbf{H}}\tilde{\mathbf{s}}) &= 0\\ -\tilde{\mathbf{H}}^H\tilde{\mathbf{y}}+\tilde{\mathbf{H}}^H\tilde{\mathbf{H}}\tilde{\mathbf{s}} &= 0\\ \tilde{\mathbf{s}} &= (\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^H\tilde{\mathbf{y}}\end{aligned} \tag{4.2}$$

Applying weight matrix $\tilde{\mathbf{W}}_{ZF}=(\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^H$ the channel interference is canceled as follows

$$\begin{aligned}\tilde{\mathbf{s}}_{ZF}=\tilde{\mathbf{W}}_{ZF}\cdot\tilde{\mathbf{y}}=(\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^H(\tilde{\mathbf{H}}\tilde{\mathbf{s}}_t+\tilde{\mathbf{v}})=\\ =\tilde{\mathbf{s}}_t+(\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-\mathbf{1}}\tilde{\mathbf{H}}^H\tilde{\mathbf{v}}=\tilde{\mathbf{s}}_t+\hat{\tilde{\mathbf{v}}}_{ZF}\end{aligned} \tag{4.3}$$

The BER performance is proportional to the power of $\hat{\tilde{\mathbf{v}}}_{ZF}$, i.e., $\|\hat{\tilde{\mathbf{v}}}_{ZF}\|_2^2$. By factorizing the channel matrix based on the SVD the expected value of post-detection noise power can be evaluated as

$$\begin{aligned}E\left[\|\hat{\tilde{\mathbf{v}}}_{ZF}\|_2^2\right] &= E\left[\|(\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^H\mathbf{v}\|^2\right]=E\left[\|(\tilde{\mathbf{V}}\mathbf{\Sigma}^2\tilde{\mathbf{V}}^H)^{-\mathbf{1}}\tilde{\mathbf{V}}\mathbf{\Sigma}\tilde{\mathbf{U}}^H\mathbf{v}\|^2\right]\\ &= E\left[\|\tilde{\mathbf{V}}\mathbf{\Sigma}^{-2}\tilde{\mathbf{V}}^H\tilde{\mathbf{V}}\mathbf{\Sigma}\tilde{\mathbf{U}}^H\mathbf{v}\|^2\right]=E\left[\|\tilde{\mathbf{V}}\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\mathbf{v}\|^2\right]\\ &= E\left[\mathbf{v}^H\tilde{\mathbf{U}}\mathbf{\Sigma}^{-\mathbf{1}}\tilde{\mathbf{V}}^H\tilde{\mathbf{V}}\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\mathbf{v}\right]=E\left[\|\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\mathbf{v}\|^2\right]\\ &= E\left[\mathrm{tr}(\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\mathbf{v}\mathbf{v}^H\tilde{\mathbf{U}}\mathbf{\Sigma}^{-1})\right]=\mathrm{tr}(\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\mathbf{E}\left[\mathbf{v}\mathbf{v}^H\right]\tilde{\mathbf{U}}\mathbf{\Sigma}^{-1})\\ &= \sigma^2\mathrm{tr}(\mathbf{\Sigma}^{-1}\tilde{\mathbf{U}}^H\tilde{\mathbf{U}}\mathbf{\Sigma}^{-1})=\sigma^2\mathrm{tr}(\mathbf{\Sigma}^{-2})\\ &= \sum_{i=1}^{n}\frac{\sigma^2}{\sigma_i^2}\end{aligned} \tag{4.4}$$

Eq. 4.4 shows that linear detection suffers from a significant noise enhancement when the condition number of the channel matrix is large, because this means that the minimum singular value is small. As a result, the post-detection noise power $E\left[\|\hat{\tilde{\mathbf{v}}}_{ZF}\|_2^2\right]$, i.e., the BER, is mainly determined by the minimum singular value of the channel matrix

$$E\left[\|\hat{\tilde{\mathbf{v}}}_{ZF}\|_2^2\right]=\sum_{i=1}^{n}\frac{\sigma^2}{\sigma_i^2}\approx\frac{\sigma^2}{\sigma_{min}^2} \tag{4.5}$$

As a conclusion the advantage of the ZF detection is that it completely eliminates the

DOI:10.15774/PPKE.ITK.2015.010

interference with low complexity computations, however, the BER degradation caused by the noise enhancement of the inverse channel matrix multiplication is the main drawback of this approach.

4.3.2 Minimum mean square error detection

It was shown in Sec. 4.3.1 that ZF detection completely eliminates interference, however, the noise power is enhanced significantly. The aim of MMSE detection is to maximize post-detection signal-to-interference plus noise (SINR) ratio, namely, MMSE tries to find a weight matrix $\tilde{\mathbf{W}}_{MMSE}$ that minimizes the following criteria

$$\tilde{\mathbf{W}}_{MMSE} = \underset{\tilde{\mathbf{W}} \in \mathbb{R}^{M \times N}}{\arg\min}\, E\left[\|\tilde{\mathbf{W}}^H \tilde{\mathbf{y}} - \tilde{\mathbf{s}}_t\|^2\right]. \tag{4.6}$$

The solution is found by setting the partial derivative to zero as follows

$$\begin{aligned}
\frac{\partial}{\partial \tilde{\mathbf{W}}} E\left[\|\tilde{\mathbf{W}}^H \tilde{\mathbf{y}} - \tilde{\mathbf{s}}_t\|^2\right] &= 0 \\
\frac{\partial}{\partial \tilde{\mathbf{W}}} E\left[\tilde{\mathbf{W}}^H \tilde{\mathbf{y}} \tilde{\mathbf{y}}^H \tilde{\mathbf{W}} - \tilde{\mathbf{s}}_t \tilde{\mathbf{y}}^H \tilde{\mathbf{W}} - \tilde{\mathbf{W}}^H \tilde{\mathbf{y}} \tilde{\mathbf{s}}_t^H + \tilde{\mathbf{s}}_t \tilde{\mathbf{s}}_t^H\right] &= 0 \\
\frac{\partial}{\partial \tilde{\mathbf{W}}} (\tilde{\mathbf{W}}^H \mathbf{R}_{yy} \tilde{\mathbf{W}} - \mathbf{R}_{sy} \tilde{\mathbf{W}} - \tilde{\mathbf{W}}^H \mathbf{R}_{ys} + \mathbf{R}_{ss}) &= 0 \\
\tilde{\mathbf{W}}^H \mathbf{R}_{yy} - \mathbf{R}_{sy} &= 0 \\
\tilde{\mathbf{W}}^H &= \mathbf{R}_{sy} \mathbf{R}_{yy}^{-1}
\end{aligned} \tag{4.7}$$

where $\mathbf{R}_{ys}$ is the cross-covariance matrix of the received symbol vector $\tilde{\mathbf{y}}$ and the symbol vector sent $\tilde{\mathbf{s}}_t$, and $\mathbf{R}_{yy}$ is the covariance matrix of the received symbol vectors. As a result, the MMSE weight matrix is equal to

$$\tilde{\mathbf{W}}_{MMSE}^H = \mathbf{R}_{sy} \mathbf{R}_{yy}^{-1}. \tag{4.8}$$

In order to determine the covariance matrices several assumptions are made that are common in the most of the communication systems:

- successive noise samples are uncorrelated and AWGN channel is assumed, thus, the noise covariance is $\mathbf{R}_{nn} = \sigma^2 \mathbf{I}_m$;
- the symbols of the transmitted symbol vector $\mathbf{s}_t$ are statistically independent, thus, the symbol vectors covariance is $\mathbf{R}_{ss} = \sigma_s^2 \mathbf{I}_n$;
- the noise samples are independent of the symbols in the symbol vector sent, thus, the covariance of the symbols and noise is $\mathbf{R}_{sn} = \mathbf{0}$.

DOI:10.15774/PPKE.ITK.2015.010

With the above assumption covariance matrices $\mathbf{R}_{sy}$ and $\mathbf{R}_{yy}$ are determined as follows:

$$\begin{aligned}\mathbf{R}_{yy} &= E\left[\tilde{\mathbf{y}}\tilde{\mathbf{y}}^H\right] \\ &= E\left[(\tilde{\mathbf{H}}\tilde{\mathbf{s}}_t + \tilde{\mathbf{n}})(\tilde{\mathbf{H}}\tilde{\mathbf{s}}_t + \tilde{\mathbf{n}})^H\right] \\ &= E\left[\tilde{\mathbf{H}}\tilde{\mathbf{s}}_t\tilde{\mathbf{s}}_t^H\tilde{\mathbf{H}}^H + \tilde{\mathbf{H}}\tilde{\mathbf{s}}_t\tilde{\mathbf{n}}^H + \tilde{\mathbf{n}}\tilde{\mathbf{s}}_t^H\tilde{\mathbf{H}}^H + \tilde{\mathbf{n}}\tilde{\mathbf{n}}^H\right] \\ &= \sigma_s^2\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H + \sigma^2\mathbf{I}_m\end{aligned} \tag{4.9}$$

$$\begin{aligned}\mathbf{R}_{sy} &= E\left[\tilde{\mathbf{s}}_t\tilde{\mathbf{y}}^H\right] \\ &= E\left[\tilde{\mathbf{s}}_t(\tilde{\mathbf{H}}\tilde{\mathbf{s}}_t + \tilde{\mathbf{n}})^H\right] \\ &= E\left[\tilde{\mathbf{s}}_t\tilde{\mathbf{s}}_t^H\tilde{\mathbf{H}}^H + \tilde{\mathbf{s}}_t\tilde{\mathbf{n}}^H\right] \\ &= \sigma_s^2\tilde{\mathbf{H}}^H\end{aligned} \tag{4.10}$$

Based on the above results the MMSE weight matrix is defined as follows:

$$\tilde{\mathbf{W}}_{MMSE}^H = \sigma_s^2\tilde{\mathbf{H}}^H(\sigma_s^2\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H + \sigma^2\mathbf{I}_m)^{-1} = \tilde{\mathbf{H}}^H(\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H + \frac{\sigma^2}{\sigma_s^2}\mathbf{I}_m)^{-1} \tag{4.11}$$

Complexity reduction is achieved with the following equivalent transformation

$$\tilde{\mathbf{W}}_{MMSE}^H = (\tilde{\mathbf{H}}^H\tilde{\mathbf{H}} + \frac{\sigma^2}{\sigma_s^2}\mathbf{I}_n)^{-1}\tilde{\mathbf{H}}^H. \tag{4.12}$$

$$\begin{aligned}\tilde{\mathbf{s}}_{MMSE} &= \tilde{\mathbf{W}}_{MMSE}^H \cdot \tilde{\mathbf{y}} \\ &= \hat{\tilde{\mathbf{s}}}_t + (\tilde{\mathbf{H}}^H\tilde{\mathbf{H}} + \frac{\sigma^2}{\sigma_s^2}\mathbf{I}_n)^{-1}\tilde{\mathbf{H}}^H\tilde{\mathbf{v}} \\ &= \hat{\tilde{\mathbf{s}}}_t + \hat{\tilde{\mathbf{v}}}_{MMSE}\end{aligned} \tag{4.13}$$

In Eq. 4.13 weight matrix $\tilde{\mathbf{W}}_{MMSE}$ is applied for detection. Unlike ZF detection it is shown that in $\hat{\tilde{\mathbf{s}}}_t$ the interference is not perfectly suppressed, however, the post-detection noise power can be much lower in certain situations. In order to compare the average post-detection noise power of MMSE detection $E\left[\|\hat{\tilde{\mathbf{v}}}_{MMSE}\|\right]$ with the ZF post-detection noise power, unit signal power is assumed $\sigma_s^2 = 1$, thus $E\left[\|\hat{\tilde{\mathbf{v}}}_{MMSE}\|\right]$ is evaluated as follows:

DOI:10.15774/PPKE.ITK.2015.010

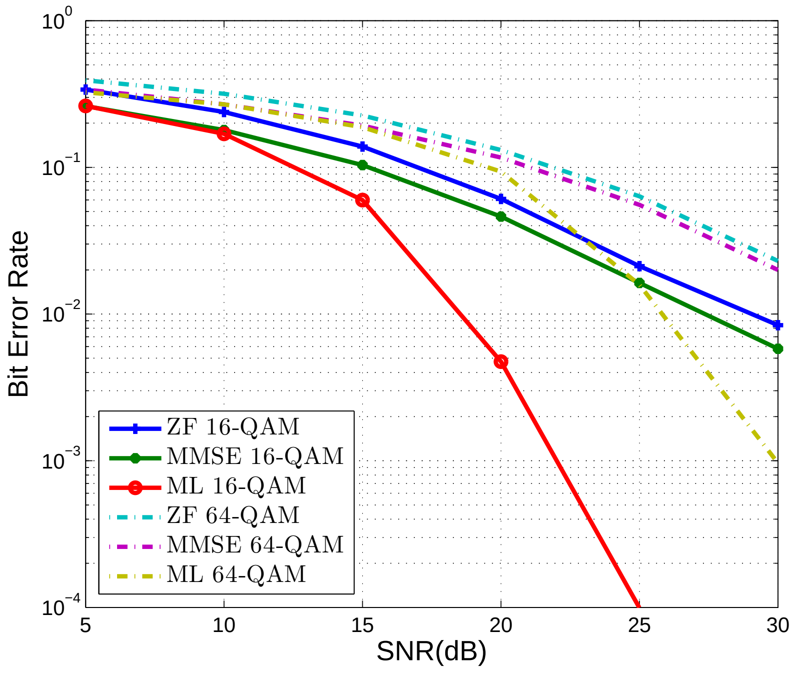


Figure 4.2: Bit error rate performance comparison of linear detectors for 4 × 4 MIMO systems with 16 and 64-QAM symbol constellations.

$$
\begin{aligned}
E\left[\|\hat{\tilde{\mathbf{v}}}_{MMSE}\|_2^2\right] &= E\left[\|(\tilde{\mathbf{H}}^H\tilde{\mathbf{H}}+\sigma^2\mathbf{I}_n)^{-1}\tilde{\mathbf{H}}^H\tilde{\mathbf{v}}\|^2\right]\\
&= E\left[\|(\tilde{\mathbf{V}}\mathbf{\Sigma}^2\tilde{\mathbf{V}}^H+\sigma^2\mathbf{I}_n)^{-1}(\mathbf{\Sigma}^{-1}\tilde{\mathbf{V}}^H)^{-1}\tilde{\mathbf{U}}^H\tilde{\mathbf{v}}\|^2\right]\\
&= E\left[\|(\mathbf{\Sigma}\tilde{\mathbf{V}}^H+\sigma^2\mathbf{\Sigma}^{-1}\tilde{\mathbf{V}}^H)^{-1}\tilde{\mathbf{U}}^H\tilde{\mathbf{v}}\|^2\right]\\
&= E\left[\|\tilde{\mathbf{V}}(\mathbf{\Sigma}+\sigma^2\mathbf{\Sigma}^{-1})^{-1}\tilde{\mathbf{U}}^H\tilde{\mathbf{v}}\|^2\right]\\
&= tr((\mathbf{\Sigma}+\sigma^2\mathbf{\Sigma}^{-1})^{-1}\tilde{\mathbf{U}}^H E\left[\tilde{\mathbf{v}}\tilde{\mathbf{v}}^H\right]\tilde{\mathbf{U}}(\mathbf{\Sigma}+\sigma^2\mathbf{\Sigma}^{-1})^{-1})\\
&= tr(\sigma^2(\mathbf{\Sigma}+\sigma^2\mathbf{\Sigma}^{-1})^{-2})\\
&= \sum_{i=1}^{N_t}\frac{\sigma_i^2\sigma^2}{(\sigma_i^2+\sigma^2)^2}
\end{aligned}
\tag{4.14}
$$

The noise enhancement caused by the minimum singular value in case of MMSE detection is given as follows

$$
E\left[\|\hat{\tilde{\mathbf{v}}}_{MMSE}\|_2^2\right] = \sum_{i=1}^{n}\frac{\sigma_i^2\sigma^2}{(\sigma_i^2+\sigma^2)^2} \approx \frac{\sigma_{min}^2\sigma^2}{(\sigma_{min}^2+\sigma^2)^2} \tag{4.15}
$$

Comparing the noise enhancement of the MMSE and ZF detection based on Eq. 4.15 and Eq. 4.5 when $\sigma_{min}^2 \ll \sigma^2$ it is visible that MMSE detection is less critical. In case when $\sigma^2 \ll \sigma_{min}^2$ the performance of the two detectors becomes the same.

DOI:10.15774/PPKE.ITK.2015.010

In Fig. 4.2 the performance of the linear detectors are compared against the ML detector. It is visible that the MMSE performs slightly better compared to the ZF detector, however, the optimal ML detector is orders of magnitude better in the higher SNR regions.

4.4 Successive interference cancellation detectors

4.4.1 Successive interference cancellation detection concept

SIC is a non-linear technique that performs *linear combinatorial nulling* and *symbol cancellation* for each transmitted substream. This is somewhat analogous to decision-feedback equalization. During SIC detection in every iteration a different substream is considered the desired signal and the remainder substreams are the interferers. In order to extract the original symbol sent a weight vector has to be defined such that

$$\tilde{\mathbf{w}}_i(\tilde{\mathbf{H}}_j) = \delta_{ij} \tag{4.16}$$

where $(\tilde{\mathbf{H}}_j)$ is the j-th column of the channel matrix $\tilde{\mathbf{H}}$, and δ is the Kronecker delta. As discussed in Sections 4.3.1 and 4.3.2, the ZF weight matrix completely eliminates the interference, however, the MMSE weight matrix achieves a better BER. As a result, the nulling weight vector $\tilde{\mathbf{w}}_i$ can be computed based on the above techniques. In the following the ZF method is used to compute the nulling weight vector. Thus, the detected symbol on the i-th substream is $\hat{\tilde{\mathbf{s}}}_i = Q\left(\tilde{\mathbf{w}}_i\tilde{\mathbf{y}}\right)$, where Q slices the unconstrained result of the linear nulling to a valid symbol.

After a symbol is detected, based on linear nulling, *symbol cancellation* is performed in order to extract the interference from the received symbol vector. As a result, the modified received symbol vector contains fewer interferers and it is formulated as follows

$$\tilde{\mathbf{y}}_{i+1} = \tilde{\mathbf{y}}_i - \hat{\tilde{\mathbf{s}}}_i \cdot (\tilde{\mathbf{H}})_i. \tag{4.17}$$

The last step of this process is the modification of the channel matrix. Since in the i-th iteration the interference of the detected symbol is canceled out from the received symbol vector the new channel matrix $\tilde{\mathbf{H}}_{\bar{i}}$ is obtained by zeroing column i. In order to continuously fulfill the condition imposed by Eq. 4.16, the Moore-Penrose pseudoinverse $\tilde{\mathbf{H}}_{\bar{i}}^{\dagger}$ of the modified channel matrix $\tilde{\mathbf{H}}_{\bar{i}}$ is computed, and the detection process continues by repeating these three steps: (i) linear nulling, (ii) symbol cancellation and (iii) the

DOI:10.15774/PPKE.ITK.2015.010

pseudoinverse computation of the modified channel matrix. This method ends when every symbol is detected.

4.4.2 The Vertical Bell Laboratories Layered Space-Time architecture

Foschini et al. in [49] introduced the Diagonal Bell Laboratories Layered Space-Time (D-BLAST) architecture. This architecture is able to exploit the capacity increase of the rich scattering multipath channels. During transmission space-time block coding is implemented, namely redundancy is introduced between the substreams with the help of inter-substream block coding. The main drawback of this approach is the increased complexity of the space-time coding process.

In order to overcome the implementation complexities of D-BLAST, Wolniansky et al. in [64] proposed a simplified Vertical BLAST (V-BLAST) architecture. The V-BLAST implements the spatial multiplexing approach where, instead of a sophisticated inter-stream coding and decoding, the vector encoding process is implemented by demultiplexing the bitstream followed by a bit-to-symbol mapping for every substream. The elimination of the complex coding scheme results in a reduced complexity V-BLAST architecture that enables high spectral efficiencies.

The V-BLAST builds on the SIC technique discussed in Sec. 4.4.1. When symbol cancellation is used the overall system performance, namely the achieved BER, is seriously influenced by the order of detected symbols. For example, in case if the detected symbol is different from the symbol sent $\hat{\tilde{\mathbf{s}}}_i \neq \tilde{\mathbf{s}}_i$ when subtracting its effect from the received symbol vector, noise is introduced instead of lowering the number of interferers. Several metrics can be defined to determine the order of detection. The main contribution of Wolniansky et al. in [64] was the proposed optimal detection ordering based on the post-detection SNR. The following enumeration presents the most important ordering metrics:

1. SINR based ordering. SINR is calculated when MMSE weight matrix is used. The ordering metric is calculated as follows:

$$SINR_i = \frac{\sigma_s^2 \mid \tilde{\mathbf{w}}_i \tilde{\mathbf{h}}_i \mid^2}{\sigma_s^2 \sum_{l \neq i} \mid \tilde{\mathbf{w}}_l \tilde{\mathbf{h}}_l \mid^2 + \sigma^2 \|\tilde{\mathbf{w}}_l\|^2} \tag{4.18}$$

   where in the denominator the effects of interference and noise are summed. The ordering based on the post-detection SINR metric achieves the best performance, however, the computational complexity is slightly increased.

DOI:10.15774/PPKE.ITK.2015.010

2. SNR based ordering. SNR is calculated with the ZF weight matrix. In this case, the interference is completely eliminated thus the ordering metric is computed as follows:
$$SNR_i = \frac{\sigma_s^2}{\sigma^2 \|\tilde{\mathbf{w}}_i\|^2}. \tag{4.19}$$
The computational complexity is reduced since there is no need to compute the effects of the interference, however, the achieved performance is inferior compared to the SINR ordering.

3. Column Norm based ordering. The metrics presented above involve complex computations, thus, a simpler metric based on the column norms of the channel matrix can be defined. Based on the system model shown in Eq. 3.1 the received symbol vector can be represented as follows:
$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\tilde{\mathbf{s}}_t + \tilde{\mathbf{v}} = \tilde{\mathbf{h}}_1\tilde{s}_1 + \tilde{\mathbf{h}}_2\tilde{s}_2 + \cdots + \tilde{\mathbf{h}}_n\tilde{s}_n + \tilde{\mathbf{v}} \tag{4.20}$$
where $\tilde{\mathbf{h}}_i$ represents the i-th column of the channel matrix $\tilde{\mathbf{H}}$. The ordering metric is based on the norms of the column vectors $\|\tilde{\mathbf{h}}_i\|$, as a result, the received signal strength is proportional with the ordering metric. The order of detection depends on the implemented algorithm. Algorithms based on SIC require to detect the strongest symbols first, however, the PSD presented in Sec. 4.8 starts the detection process with the lowest metric symbols.

In [64] Wolniansky et al. presented the full ZF V-BLAST algorithm where the ordering used was based on the post-detection SNR. Based on Eq. 4.19 the maximum post-detection SNR is achieved when $\|\tilde{\mathbf{w}}_i\|^2$ is minimal. As a result, in every iteration of the detection process the stream with the highest post-detection SNR is identified by recalculating the Moore-Penrose pseudoinverse of the modified channel matrix, and by identifying the minimum norm row of the resulting inverse. A brief overview of the ZF V-BLAST detection algorithm is given Alg. 1.

In Fig. 4.3 the V-BLAST detection BER performance is compared when (i) SINR, (ii) SNR and (iii) column norm based orderings are considered. As expected the SINR based ordering is performing better than the other two methods. Although, the above presented V-BLAST versions are interesting due to their low complexity, their BER performance is still far from the optimal ML performance.

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 1 Zero-Forcing Vertical BLAST detection algorithm

```
Require: $\tilde{\mathbf{y}}, \tilde{\mathbf{H}}$
1: for $i = 1$ to $n$ do
2:     $\tilde{\mathbf{W}}_\mathbf{i} = \tilde{\mathbf{H}}_\mathbf{i}^\dagger$
3:     $k_i = \underset{j \notin \{k_1, k_2, \ldots, k_{i-1}\}}{\operatorname{argmin}} \|(\tilde{\mathbf{W}}_\mathbf{i})_j\|^2$    ▷ $(\tilde{\mathbf{W}}_\mathbf{i})_j$ denotes the j-th row of matrix $\tilde{\mathbf{W}}_\mathbf{i}$
4:     $\tilde{\mathbf{w}}_{k_i} = (\tilde{\mathbf{W}}_\mathbf{i})_{k_i}$
5:     $\hat{\tilde{\mathbf{s}}}_{k_i} = Q\left(\tilde{\mathbf{w}}_{k_i} \cdot \tilde{\mathbf{y}}_i\right)$
6:     $\tilde{\mathbf{y}}_{i+1} = \tilde{\mathbf{y}}_i - \hat{\tilde{\mathbf{s}}}_{k_i} \cdot (\tilde{\mathbf{H}})_{k_i}$    ▷ $(\tilde{\mathbf{H}})_{k_i}$ denotes the $k_i$-th column of matrix $\tilde{\mathbf{H}}$
7:     $\tilde{\mathbf{H}}_{i+1} = \tilde{\mathbf{H}}_{\overline{k_1, k_2, \ldots, k_i}}$    ▷ $\tilde{\mathbf{H}}_{\overline{k_i}}$ denotes the modified channel matrix where
   column $k_i$ is zeroed.
8: end for
```

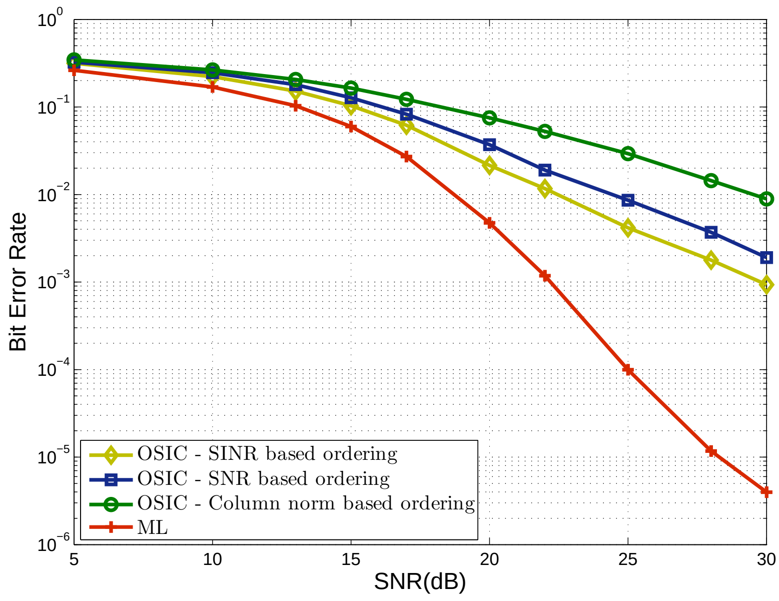


Figure 4.3: Vertical BLAST detection bit error rate performance comparison for 4×4 MIMO with 16-QAM symbol constellation considering (i) SINR, (ii) SNR and (iii) column norm based ordering.

DOI:10.15774/PPKE.ITK.2015.010

4.5 Maximum likelihood detection

The ML detector under the statistical assumptions given in Sec. 4.1 minimizes the probability of error

$$P_e = P(\tilde{\mathbf{s}}_t \neq \hat{\tilde{\mathbf{s}}})$$

that is equivalent to maximizing the probability of the correct estimation of the transmitted symbol vector $\tilde{s}_t$ from the given $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{H}}$ as

$$P(\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}}|\tilde{\mathbf{y}}, \tilde{\mathbf{H}}). \tag{4.21}$$

Applying Bayes's theorem the posterior probability can be given by multiplying the prior probability distribution with the likelihood and then dividing by the normalizing constant, as follows:

$$P(\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}}|\tilde{\mathbf{y}}) = \frac{f(\tilde{\mathbf{y}}|\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}})P(\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}})}{f(\tilde{\mathbf{y}})}, \tag{4.22}$$

where $P(\tilde{\mathbf{s}}_t)$ is the a priori probability of the transmitted symbol vector $\tilde{\mathbf{s}}_t$, $f(\tilde{\mathbf{y}}|\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}})$ is the conditional probability density function of the random observation symbol vector $\tilde{\mathbf{y}}$ given the transmission of symbol vector $\hat{\tilde{\mathbf{s}}}$, and $f(\tilde{\mathbf{y}})$ is the unconditional probability density function of $\tilde{\mathbf{y}}$. Since $f(\tilde{\mathbf{y}})$ does not depend on $\hat{\tilde{\mathbf{s}}}$ and assuming a priori equally likely symbols, namely $P(\tilde{\mathbf{s}})$ is constant, the maximization of the a posteriori probability reduces to the maximization of the likelihood. As a result the ML detector is defined as:

$$\hat{\tilde{\mathbf{s}}}_{ML} = \underset{\hat{\tilde{\mathbf{s}}} \in \tilde{\Omega}^n}{\arg\max} f(\tilde{\mathbf{y}}|\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}}). \tag{4.23}$$

By assuming the conditions presented in Sec. 3.2 and applying the model given in Eq. 3.1 the conditional probability density function $f(\tilde{\mathbf{y}}|\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}})$ equals

$$\begin{aligned} f(\tilde{\mathbf{y}}|\tilde{\mathbf{s}}_t = \hat{\tilde{\mathbf{s}}}) &= \frac{1}{det(\pi K)} e^{-(\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\hat{\tilde{\mathbf{s}}})^H \mathbf{K}^{-1} (\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\hat{\tilde{\mathbf{s}}})} \\ &= \frac{1}{det(\pi\sigma^2)^m} e^{-\frac{1}{\sigma^2}(\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\hat{\tilde{\mathbf{s}}})^H (\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\hat{\tilde{\mathbf{s}}})} \end{aligned} \tag{4.24}$$

The maximization of the ML metric implies the minimization of the exponent. The optimal ML solution ignoring the constant terms is:

$$\hat{\tilde{\mathbf{s}}}_{ML} = \underset{\hat{\tilde{\mathbf{s}}} \in \tilde{\Omega}^n}{\arg\min} \|\tilde{\mathbf{y}} - \tilde{\mathbf{H}}\hat{\tilde{\mathbf{s}}}\|^2. \tag{4.25}$$

A similar proof is possible for the real-valued model, given in Eq. 3.4. For the real-

DOI:10.15774/PPKE.ITK.2015.010

valued system the ML solution is

$$\hat{\mathbf{s}}_{ML} = \underset{\hat{\mathbf{s}} \in \Omega^N}{\arg\min} \|\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}\|^2 \tag{4.26}$$

where $\mathbf{y}, \mathbf{H}, \mathbf{s}_t, \hat{\mathbf{s}}_{ML}$ are all real-valued quantities and Ω is a real-valued signal set.

The exhaustive search implementation of ML detection has a complexity that grows exponentially with both the number of elements in the signal set Ω and the number of antennas. Consequently, the required computational performance becomes unattainable. For general lattices the problem has been shown to be NP-hard [65]. However, significant complexity reduction can be achieved by exploiting the structure of the lattice as shown in [66], [67]. In Sec. 4.6.1 the SD algorithm is discussed in details, showing how the significant complexity reduction can be achieved.

From a different perspective Eq. 4.26 shows that the ML estimate of the transmitted symbol vector is found by solving an ILS problem that is analogous to finding the closest lattice point of lattice $\mathbf{\Lambda} = \{\mathbf{Hs} | \mathbf{s} \in \Omega^N\}$ to a given point $\mathbf{y}$ [62], [68]. In lattice theory this problem is often referred to as the closest lattice point search (CLPS) [63], [61].

4.6 Maximum likelihood tree-search based detectors

4.6.1 The Sphere Detector algorithm

The fundamental aim of the SD algorithm is to restrict the search to lattice points that lie within a certain sphere of radius d around a given received symbol vector. Reducing the search space will not affect the detection quality because the closest lattice point inside the sphere will also be the closest lattice point for the whole lattice. The reduction of the search space is necessary in order to reduce the high computational complexity required by the ML detection.

4.6.1.1 General description of the Sphere Detector algorithm

In the following it is assumed that (i) $N \leq M$, i.e., the number of receive antennas is greater or equal to the number of transmit antennas, and (ii) the channel matrix has full rank. Furthermore, it is assumed that perfect CSI is available at the receiver. The *unconstrained least-squares* solution of the equivalent real-valued system is defined as

$$\hat{\mathbf{s}} = \mathbf{H}^\dagger \mathbf{y} \tag{4.27}$$

DOI:10.15774/PPKE.ITK.2015.010

where $\mathbf{H}^{\dagger} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ is the Moore–Penrose pseudoinverse of the channel matrix.

Applying QR factorization to the real channel matrix $\mathbf{H} = \mathbf{QR}$, the ML solution from Eq. 4.26 can be rearranged as

$$\begin{aligned} \mathbf{s}_{ML} &= \arg\min_{\mathbf{s}\in\Omega^N} \|\mathbf{y} - \mathbf{Hs}\|^2 \\ &= \arg\min_{\mathbf{s}\in\Omega^N} (\mathbf{s} - \hat{\mathbf{s}})^{\mathbf{T}}\mathbf{H}^{\mathbf{T}}\mathbf{H}(\mathbf{s} - \hat{\mathbf{s}}) \\ &= \arg\min_{\mathbf{s}\in\Omega^N} (\mathbf{s} - \hat{\mathbf{s}})^{\mathbf{T}}(\mathbf{QR})^{\mathbf{T}}(\mathbf{QR})(\mathbf{s} - \hat{\mathbf{s}}) \\ &= \arg\min_{\mathbf{s}\in\Omega^N} \|\mathbf{R}(\mathbf{s} - \hat{\mathbf{s}})\|^2 \end{aligned} \tag{4.28}$$

where matrix $\mathbf{Q}$ is orthogonal and matrix $\mathbf{R}$ upper triangular. In order to arrive to Eq. 4.28 the following expansion has to be considered

$$\begin{aligned} \|\mathbf{y} - \mathbf{Hs}\|^2 &= \|\mathbf{y} - \mathbf{Hs} - \mathbf{H}\hat{\mathbf{s}} + \mathbf{H}\hat{\mathbf{s}}\|^2 \\ &= (\mathbf{y} - \mathbf{Hs} - \mathbf{H}\hat{\mathbf{s}} + \mathbf{H}\hat{\mathbf{s}})^T(\mathbf{y} - \mathbf{Hs} - \mathbf{H}\hat{\mathbf{s}} + \mathbf{H}\hat{\mathbf{s}}) \\ &= \{(\mathbf{y} - \mathbf{H}\hat{\mathbf{s}})^{\mathbf{T}} + (\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs})^{\mathbf{T}}\}\{(\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}) + (\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs})\} \\ &= (\mathbf{y} - \mathbf{H}\hat{\mathbf{s}})^{\mathbf{T}}(\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}) + (\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs})^{\mathbf{T}}(\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs}) + \\ &\quad + (\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs})^{\mathbf{T}}(\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}) + (\mathbf{y} - \mathbf{H}\hat{\mathbf{s}})^{\mathbf{T}}(\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs}) \end{aligned} \tag{4.29}$$

Since $\hat{\mathbf{s}}$ is the unconstrained least-squares solution $(\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}) = \mathbf{0}$, consequently, Eq.4.29 reduces to

$$\begin{aligned} \|\mathbf{y} - \mathbf{Hs}\|^2 &= (\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs})^{\mathbf{T}}(\mathbf{H}\hat{\mathbf{s}} - \mathbf{Hs}) \\ &= (\mathbf{s} - \hat{\mathbf{s}})^{\mathbf{T}}\mathbf{H}^{\mathbf{T}}\mathbf{H}(\mathbf{s} - \hat{\mathbf{s}}) \end{aligned} \tag{4.30}$$

The lattice point $\mathbf{Hs}$ is included by the sphere $S(\mathbf{y}, d)$ with center point $\mathbf{y}$ and radius d if the following inequality is satisfied

$$\|\mathbf{R}(\mathbf{s} - \hat{\mathbf{s}})\|^2 \leqslant d^2 \tag{4.31}$$

$$\left\| \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ 0 & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{NN} \end{pmatrix} \begin{pmatrix} s_1 - \hat{s}_1 \\ s_2 - \hat{s}_2 \\ \vdots \\ s_N - \hat{s}_N \end{pmatrix} \right\|^2 \leqslant d^2.$$

Instead of evaluating all possible symbol combinations the upper triangular property of matrix $\mathbf{R}$ is exploited and a recursion is defined based on the dependency hierarchy of the terms.

In order to obtain a deeper insight, let $\mathbf{s}_i^N \triangleq (s_i, s_{i+1}, \cdots, s_N)^T$, referred to as partial symbol vector, denote the last $N-i+1$ components of the vector $\mathbf{s}$ and let

$$M(\mathbf{s}_i^N) = \sum_{j=i}^{N} \left| \sum_{k=j}^{N} r_{jk}(s_k - \hat{s}_k) \right|^2 \tag{4.32}$$

define the *path metric* of $\mathbf{s}_i^N$. The recursion starts at level N and a solution candidate is found when the first level is reached. In every iteration a partial symbol vector $\mathbf{s}_i^N$ is *expanded*. During the *expansion* one symbol $s_{i-1} \in \Omega$ is selected from the symbol set and it is added to the partial symbol vector as follows $\mathbf{s}_{i-1}^N = (s_{i-1}, s_i, \cdots, s_N) = (s_{i-1}, \mathbf{s}_i^N)$. The *evaluation* of the new partial symbol vector $\mathbf{s}_{i-1}^N$ is the computation of the path metric $M(\mathbf{s}_{i-1}^N)$. If the conditions are met, namely, when $M(\mathbf{s}_{i-1}^N) < d^2$, a new symbol $s_{i-2} \in \Omega$ has to be selected for the next dimension. If not, then the previously chosen symbol s_{i-1} is discarded and a new symbol for the same level is chosen from the signal set.

A possible solution is found if a complete symbol vector $\mathbf{s}_1^N$ satisfies the condition $M(\mathbf{s}_1^N) < d^2$. The solution with the smallest metric is the ML solution. If a too small initial radius is chosen so that a solution is not found, the process has to be restarted with a higher radius.

Based on the above description of the SD algorithm an analogy with bounded tree search can be found. The partial symbol vectors $\mathbf{s}_i^N$ can be regarded as tree nodes at level i. The symbol vectors $\mathbf{s}_1^N$ are the leaves of the tree and the weight of each node is defined by the symbol vector metric $M(\mathbf{s}_i^N)$. The continuous change of the partial symbol vector $\mathbf{s}_i^N$ is analogous to a DFS. The condition given in Eq. 4.31 can be regarded as the bounding criteria.

In Alg. 2 the pseudo-code of a SD algorithm is presented where a DFS is implemented with a simplified enumeration strategy which is similar to the Fincke-Phost enumeration presented in [66]. The main difference is that while the bounding interval is updated after every node expansion in [66], in Alg. 2 the bounding parameter is the radius of the sphere and it is updated only when a new leaf with lower path metric is found. The variables and the three main procedures implement the same functionality in both sequential and parallel algorithms.

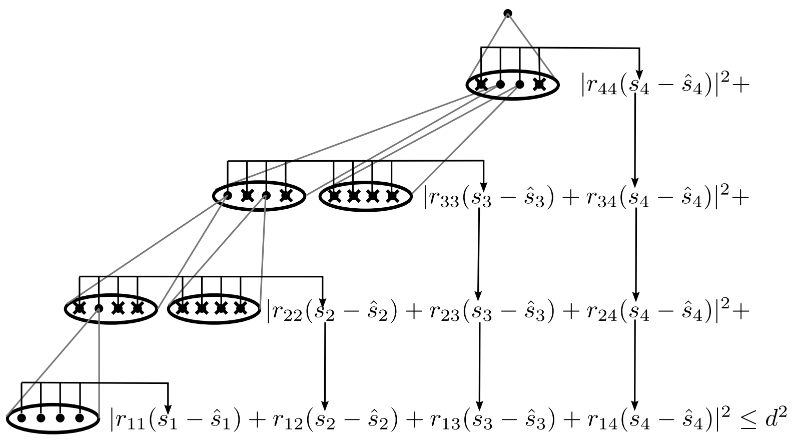
Fig. 4.4 shows a possible traversal of the tree, where the size of the symbol set $|\Omega| = 4$ and the depth of the tree is 4. This configuration belongs to a system where two receive

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 2 Sphere Detector algorithm for estimating $\mathbf{s}_{ML} = (s_1, s_2, \cdots, s_N)$

```
Require: $\hat{\mathbf{s}}, \mathbf{R}, |\Omega|$
 1: procedure Definition and Initialization of Variables
 2:     for $j = 1$ to $N$ do
 3:         $eval_j \leftarrow |\Omega|$  ▷ Number of partial symbol vector evaluations on level j after a node expansion
 4:         $buf_j[eval_j] = \{\}$  ▷ Denotes an empty buffer of size $eval_j$ for level $j$
 5:         $off_j \leftarrow 0$  ▷ Offset of processing on level j for buffer $buf_j$
 6:     end for
 7:     $buf_N \leftarrow$ expand and evaluate$(())$  ▷ Expand the root () of the tree and update $buf_N$
 8:     Traversal Process$(i \leftarrow N-1)$
 9: end procedure
10: procedure Traversal Process$(i)$
11:     while $i < N$ do
12:         if $off_{i+1} < eval_{i+1}$ then
13:             $\mathbf{s}_{i+1}^N \leftarrow buf_{i+1}(off_{i+1})$
14:             if $M(\mathbf{s}_{i+1}^N) < d^2$ then  ▷ Evaluate $M(\mathbf{s}_{i+1}^N)$ and check if $\mathbf{s}_{i+1}^N$ is inside the sphere $S(\mathbf{y}, d)$
15:                 $buf_i \leftarrow$ Expand and Evaluate$(\mathbf{s}_{i+1}^N)$  ▷ Expand partial symbol vector $\mathbf{s}_{i+1}^N$ of the tree and update $buf_i$
16:                 if $i = 1$ then
17:                     Find symbol vector $\mathbf{s}_{ML}^{'}$ in $buf_1$ with minimum path metric
18:                     $d_{temp}^2 \leftarrow \|\mathbf{R}(\mathbf{s}_{ML}^{'} - \hat{\mathbf{s}})\|^2$
19:                     if $d_{temp}^2 < d^2$ then $d^2 \leftarrow d_{temp}^2$ and $\mathbf{s}_{ML} \leftarrow \mathbf{s}_{ML}^{'}$ end if
20:                     $off_{i+1} \leftarrow off_{i+1} + 1$
21:                 else
22:                     $off_{i+1} \leftarrow off_{i+1} + 1$, $i \leftarrow i - 1$
23:                 end if
24:             else
25:                 $off_{i+1} \leftarrow off_{i+1} + 1$
26:             end if
27:         else
28:             $off_{i+1} \leftarrow 0$, $i \leftarrow i + 1$
29:         end if
30:     end while
31: end procedure
32: procedure Expand and Evaluate$(\mathbf{s}_i^N)$  ▷ The input is the partial symbol vector to be expanded
33:     for $j = 0$ to $|\Omega| - 1$ do
34:         if $\mathbf{s}_i^N = ()$ then  ▷ When expanding the root node the partial symbol vector is empty
35:             $\mathbf{s}_N^N \leftarrow \Omega[j]$
36:             $buf_N[j] \leftarrow \mathbf{s}_N^N$
37:         else
38:             $s_{i-1} \leftarrow \Omega[j]$
39:             $\mathbf{s}_{i-1}^N \leftarrow (s_{i-1}, \mathbf{s}_i^N)$
40:             $buf_{i-1}[j] \leftarrow \mathbf{s}_{i-1}^N$
41:         end if
42:     end for
43: end procedure
```

DOI:10.15774/PPKE.ITK.2015.010



$|r_{44}(s_4 - \hat{s}_4)|^2+$
$|r_{33}(s_3 - \hat{s}_3) + r_{34}(s_4 - \hat{s}_4)|^2+$
$|r_{22}(s_2 - \hat{s}_2) + r_{23}(s_3 - \hat{s}_3) + r_{24}(s_4 - \hat{s}_4)|^2+$
$|r_{11}(s_1 - \hat{s}_1) + r_{12}(s_2 - \hat{s}_2) + r_{13}(s_3 - \hat{s}_3) + r_{14}(s_4 - \hat{s}_4)|^2 \leq d^2$

Figure 4.4: Branch and bound search with the Sphere Detector algorithm.

antennas were used. The crosses denote invalid partial symbol vectors, because their metric is higher than the radius.

4.6.1.2 The Fincke-Phost and Schnorr-Euchner enumeration strategies

In order to exploit the advantage of the search space reduction, a good enumeration strategy is needed. In [61] the main enumeration strategies are presented and compared in a unified framework. In [69] Pohst proposed an efficient way of enumerating lattice points inside a sphere. Pohst's method was first implemented in digital communications by Viterbo and Biglieri [70]. Important speedups have been achieved by Schnorr and Euchner [67] by refining the Pohst method. Agrel et al. in [61] showed that the Schnorr-Euchner (SE) strategy can be efficiently used for CLPS. The main difference of these methods is the enumeration of the lattice points and the validity interval definition for every dimension. In the followings the two widely used techniques: (i) the *Fincke-Phost* (FP) and the SE enumerations are presented.

Fincke-Phost enumeration In case of the FP enumeration a validity interval I is calculated for every dimension based on the previously chosen symbols. The symbols $S = I \cap \Omega$ are expanded and evaluated starting from the lower bound of the interval. Based on Eq. 4.31 it is possible to write

$$d^2 \geq |r_{NN}(s_N - \hat{s}_N)|^2 + |r_{N-1,N-1}(s_{N-1} - \hat{s}_{N-1}) + r_{N-1,N}(s_N - \hat{s}_N)|^2 + \cdots \quad (4.33)$$

Thus, in the first interation the following inequality holds

$$\hat{s}_N - \frac{d^2}{r_{NN}} \leq s_N \leq \hat{s}_N + \frac{d^2}{r_{NN}}. \quad (4.34)$$

DOI:10.15774/PPKE.ITK.2015.010

The validity interval I_N of the first iteration is defined as

$$I_N = \left[\left\lceil \hat{s}_N - \frac{d^2}{r_{NN}} \right\rceil, \left\lfloor \hat{s}_N + \frac{d^2}{r_{NN}} \right\rfloor \right]. \tag{4.35}$$

The first symbol that is evaluated is the smallest element of the intesection $S_N = I_N \cap \Omega$. After selecting $s_N \in S_N$ the next interval I_{N-1} is further reduced because in the $N-1$ dimension there are two non-zero terms and the new symbol s_{N-1} has to satisfy the following inequality

$$\begin{aligned} \hat{s}_{N-1} - \frac{\sqrt[2]{d^2 - |r_{NN}(s_N - \hat{s}_N)|^2} - r_{N-1,N}(s_N - \hat{s}_N)}{r_{N-1,N-1}} \leq s_{N-1} \\ \leq \hat{s}_{N-1} + \frac{\sqrt[2]{d^2 - |r_{NN}(s_N - \hat{s}_N)|^2} - r_{N-1,N}(s_N - \hat{s}_N)}{r_{N-1,N-1}}. \end{aligned} \tag{4.36}$$

Once again based on the above inequality interval I_{N-1} is defined and $S_{N-1} = I_{N-1} \cap \Omega$ is computed. In case if S_{N-1} is not empty the expansion and evaluation continues with the enumeration of the elements from the smallest to the biggest, contrary the next $\tilde{s}_N$ element is evaluated. Note, the validity interval is constantly changing depending on the chosen symbols and dimension.

Schnorr-Euchner enumeration The SE enumeration builds on the Babai estimate. The Babai estimate $\hat{\mathbf{s}}_B$ is found by computing the unconstrained least-squares solution $\hat{\mathbf{s}} = \mathbf{H}^\dagger \mathbf{y}$ and applying the slicing operator to it $\hat{\mathbf{s}}_B = \lfloor \hat{\mathbf{s}} \rceil$. The slicing operator slices every element of $\hat{\mathbf{s}}$ to the nearest valid symbol. As a result, the search starts with the elements of the Babai estimate. Assuming that the Euclidean distance between the symbols is one, the enumeration strategy is defined as follows:

- if $\hat{s}_k \leq \lfloor \hat{s}_k \rceil$ the enumeration order is

$$s_k = \lfloor \hat{s}_k \rceil, \lfloor \hat{s}_k \rceil - 1, \lfloor \hat{s}_k \rceil + 1, \lfloor \hat{s}_k \rceil - 2, \cdots$$

- if $\hat{s}_k > \lfloor \hat{s}_k \rceil$ the enumeration order is

$$s_k = \lfloor \hat{s}_k \rceil, \lfloor \hat{s}_k \rceil + 1, \lfloor \hat{s}_k \rceil - 1, \lfloor \hat{s}_k \rceil + 2, \cdots$$

The conclusion of the above discussion is that the SE zigzag enumeration has several advantages over the FP enumeration.

- No initial radius is required to start the algorithm. The first leaf node found will be

DOI:10.15774/PPKE.ITK.2015.010

the Babai estimate and the radius can be updated to $d^2 = \|\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}_B\|^2$. Afterwards, if a better solution is found the radius is further reduced.

- There is no need to define validity intervals after every node expansion. If the selected symbol is outside the sphere the following symbols can be discarded because the difference of the selected symbol s_k and $\hat{s}_k$ will constantly increase with the SE enumeration.
- The zigzag enumeration assures that the chance of finding the correct symbol early is maximized.

4.6.1.3 Complexity analysis of the Sphere Detector algorithm

The complexity analysis of the SD algorithm has been thoroughly investigated by researchers. For a few good examples refer to [71], [72], [73], [74], [75]. Finding the solution of the ILS problem or that of the CLPS problem is known to be NP-hard. A general conclusion is that the complexity of the SD algorithm is directly proportional to the number of lattice points explored. Furthermore, the number of lattice points examined throughout the SD algorithm is highly influenced by the sphere radius. The optimal radius, i.e., the *covering* radius, requires a number of steps that grows exponentially [76] with the dimension of the lattice. Thus, finding the optimal radius is not feasible for real systems.

The goal of this section is to give a brief overview on the work presented by Hassibi et al. in [71] where the expected complexity of the ILS problem averaged over the noise and over the lattice is studied. Moreover, in [72] it is demonstrated that the expected complexity is polynomial for a wide range of SNRs and number of antennas. By quantifying their results it is shown that the expected number of lattice points for a wide range of SNRs can be handled in real-time with modern computing architectures.

The first problem that has to be solved is the choice of the radius. As stated above, finding the covering radius based on the lattice generator matrix $\mathbf{H}$ is an NP hard problem. One solution is to set the sphere's radius based on the noise statistics. A χ^2 random variable with M degrees of freedom is defined by scaling the noise vector as follows:

$$\frac{1}{\sigma^2} \cdot \|\mathbf{v}\|^2 = \frac{1}{\sigma^2}\|\mathbf{y} - \mathbf{H}\mathbf{s}_t\|. \tag{4.37}$$

In order to find a lattice point inside the sphere the integration of the probability density

DOI:10.15774/PPKE.ITK.2015.010

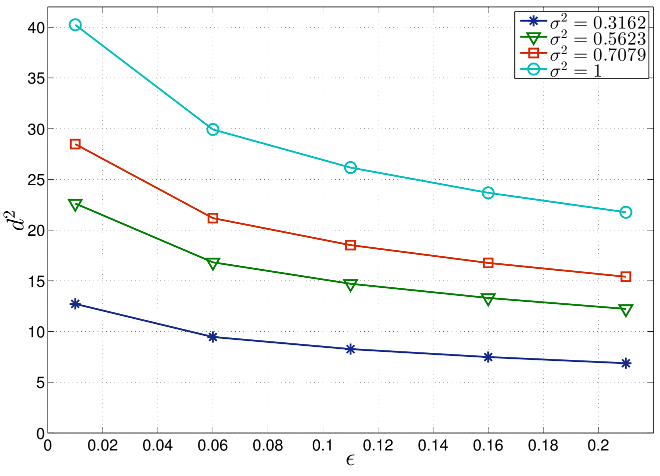


Figure 4.5: The radius size of the bounding sphere for different ε and σ^2 parameters.

function of χ^2 random variable with M degrees has to be close to one

$$\int_0^{\frac{\alpha \cdot M}{2}} \frac{\lambda^{\frac{M}{2}-1}}{2^{\frac{M}{2}}\Gamma(\frac{M}{2})} e^{\frac{-\lambda}{2}} d\lambda = 1 - \varepsilon \tag{4.38}$$

where $\Gamma(x)$ is the gamma function, $0 < \varepsilon \ll 1$ and α is a scaling parameter used to define the sphere radius as follows:

$$d^2 = \alpha M \sigma^2. \tag{4.39}$$

If no lattice point is found $1 - \varepsilon$ can be further increased by lowering the value of ε that results in the increase of the α parameter, as a result the sphere radius d^2 is also increased. Figure 4.5 shows the change of the radius for various ε and σ^2 parameter configurations.

After determining the radius, the next question is the number of visited nodes on the tree for dimensions $k = 1, \ldots, N$. Figure 4.4 shows a possible tree traversal, where on the $k = 1, 2$ dimension two nodes and on $k = 3$ only a single node fullfills the conditions of Eq. 4.31. As a result, the complexity of the SD algorithm equals the number of nodes visited on every dimension times the floating point operations required to expand and evaluate a node on the specific dimension. The following formula gives a more precise definition of the compelxity:

$$\begin{aligned} C(N,\sigma^2,d^2) &= \sum_{k=1}^{N} (\text{expected \# of nodes in } k\text{-dim sphere of radius } d) \cdot (\text{flops/node}) \\ &= \sum_{k=1}^{N} E_p(k, d^2 = \alpha M \sigma^2) \cdot f_p(k). \end{aligned} \tag{4.40}$$

The floating point operations performed for every visited node required by the SD algorithm based on the FP enumeration in [71] is defined as $f_p(k) = 2k + 11$. In the following the focus is on the derivation of $E_p(k, d^2)$.

The first step of determining the expected number of points inside the sphere is to determine how many lattice points lie within the sphere in every dimension. More formally, if $\mathbf{s}_t$ was transmitted and $\mathbf{y} = \mathbf{H}\mathbf{s}_t + \mathbf{v}$ was received what is the number of $\mathbf{s}_a$ arbitrary lattice points that satisfy

$$\begin{aligned} \|\mathbf{y} - \mathbf{H}\mathbf{s}_a\| &\le d^2 \\ \|\mathbf{v} + \mathbf{H}(\mathbf{s}_t - \mathbf{s}_a)\| &\le d^2. \end{aligned} \tag{4.41}$$

The resulting vector $\mathbf{w} = \mathbf{v} + \mathbf{H}(\mathbf{s}_t - \mathbf{s}_a)$ is an M-dimensional zero-mean Gaussian random vector where the (i,j) entry of the covariance matrix is $E\{w_i w_j\} = \delta_{ij}(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)$. This implies that $\|\mathbf{w}\|^2 / 2(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)$ is a χ^2 random variable with M degrees of freedom. As a result, the probability that $\mathbf{s}_a$ lies within a sphere of radius d around $\mathbf{y}$ is given by its cumulative distribution function (CDF)

$$F\left(M, \frac{d^2}{(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)}\right) = \frac{\gamma\left(\frac{d^2}{2(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)}, \frac{M}{2}\right)}{\Gamma\left(\frac{M}{2}\right)} \tag{4.42}$$

where $\gamma(x,k)$ is the lower incomplete Gamma function defined as

$$\gamma\left(\frac{d^2}{2(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)}, \frac{M}{2}\right) = \int_0^{\frac{d^2}{2(\sigma^2 + \|\mathbf{s}_t - \mathbf{s}_a\|^2)}} \lambda^{\frac{M}{2}-1} e^{-\lambda} d\lambda. \tag{4.43}$$

In Eq. 4.42 the CDF is determined for N dimensional lattice points. However, the CDF has to be determined for smaller dimensions as well. The derivation of the probability for partial symbol vectors is shown in [71] and the resulting formula is:

$$F\left(M - N + k, \frac{d^2}{(\sigma^2 + \|\mathbf{s}_t^k - \mathbf{s}_a^k\|^2)}\right) = \frac{\gamma\left(\frac{d^2}{2(\sigma^2 + \|\mathbf{s}_t^k - \mathbf{s}_a^k\|^2)}, \frac{M-N+k}{2}\right)}{\Gamma\left(\frac{M-N+k}{2}\right)} \tag{4.44}$$

DOI:10.15774/PPKE.ITK.2015.010

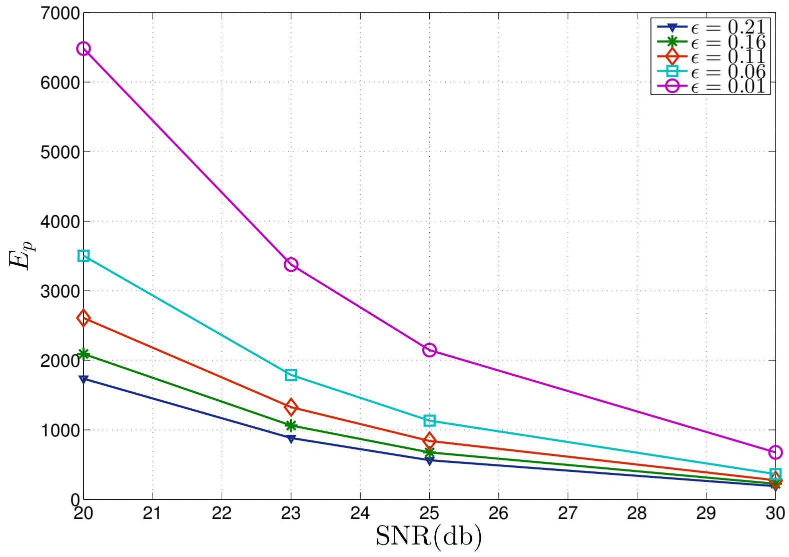


Figure 4.6: The expected number E_p of nodes visited for a 4×4 MIMO with $|\Omega| = 4$ and $\epsilon = 0.01, 0.06, 0.11, 0.16, 0.21$ parameter values.

In order to find the estimated number of points in every dimension the CDFs presented in Eqs. 4.42 and 4.44 have to be evaluated for each pair of points

$$\{(\mathbf{s}_t, \mathbf{s}_a) | \mathbf{s}_t, \mathbf{s}_a \in \Omega^k, \|\mathbf{s}_t^k - \mathbf{s}_a^k\|^2 = l\} \tag{4.45}$$

that is a computationally very intensive task. Instead of enumerating every pair of points, it is enough to count the number of points with the same argument of the gamma function. A modification of Euler's generating function technique was proposed, so with the appropiate combination of well defined generating polynomials the number of point pairs belonging to the same signal set Ω and having the same Euclidean distance l can be counted easily. The generating polynomials for a four element signal set $|\Omega| = 4$ are:

$$\theta_0 = 1 + x + x^4 + x^9 \text{ and } \theta_1 = 1 + 2x + x^4. \tag{4.46}$$

The closed form expression for the expected complexity is given as follows:

$$\begin{aligned} C(N, \sigma^2, d^2) &= \sum_{k=1}^{N} f_p(k) \cdot E_p(k, d^2 = \alpha M \sigma^2) \\ &= \sum_{k=1}^{N} f_p(k) \sum_q \frac{1}{2k} \sum_{l=0}^{k} \binom{k}{l} \cdot g_{kl}(q) \cdot F\left(M - N + k, \frac{d^2}{(\sigma^2 + l)}\right) \end{aligned} \tag{4.47}$$

where $g_{kl}(q)$ is the coefficient of x^q in the polynomial $(1 + x + x^4 + x^9)^l (1 + 2x + x^4)^{k-l}$.

In Fig. 4.6 the expected number of nodes for a 4×4 MIMO system with a four element

DOI:10.15774/PPKE.ITK.2015.010

signal set are shown. Note, that the solution might not be found, thus the search has to be restarted by increasing the size of the radius, however, the effects of possible further iterations are not accumulated. If $\epsilon = 0.01$ is chosen, which means that the probability of finding a lattice point inside the sphere is 99%, and at 20 dB SNR the expected number of visited nodes is ~ 6500 while at 30 dB SNR is less than 1000 nodes. This result is achieved with a SD using the FP enumeration. Algorithms with a lower complexity exist, i.e., algorithms based on the SE enumeration that achieve ML performance by visiting less nodes. Thus, the complexity formula shown in Eq. 4.47 can be regarded as an upper bound.

In many communication problems finding the ML solution reduces to solving an ILS problem. However, in the context of communication systems these problems are more operable because the given vector is not arbitrary but rather is an unknown lattice point that has been perturbed by an additive noise vector whose statistical properties are known. Therefore, the complexity of the algorithm has to be treated as a random variable as well. Based on these results it is possible to state that in case of high SNRs ML performance can be achieved for smaller sized MIMO configurations in real-time if the SD algorithm is suitable mapped on modern many-core architectures.

4.6.2 The Automatic Sphere Detector algorithm

Tha Automatic Sphere Detector (ASD) algorithm was introduced in [77] and the importance of this algorithm is that it expands the minimum number of nodes as the number of antennas and the modulation order is increasing. However, this does not necessarily imply that the overall computation time of the ASD is lower than that of all known decoders.

Usually, a SD algorithm during its execution performs node expansions and node evaluations, this is why the number of nodes visited in the tree is proportional with the computational complexity of the algorithm. However, existing algorithms improve their performance by implementing preprocessing techniques or by improving the search with ordering strategies, improved mathematical methods and sorting algorithms, consequently, the computational complexity together with the execution time are increased.

The ASD is a globally greedy algorithm, because in every iteration the best path metric partial symbol vector is expanded and evaluated. A detailed algorithm description is given in Alg. 3.

The conclusion is that the ASD algorithm has the following advantages: (i) it achieves

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 3 The Automatic Sphere Detector algorithm

```
Require: $\tilde{\mathbf{y}}, \tilde{\mathbf{H}}$
 1: Initialize variable size buffer $buf$
 2: EXPAND AND EVALUATE(())          ▷ Expand the root () of the tree and update $buf$
 3: $\mathbf{s} \leftarrow$ find and set the minimum metric partial symbol vector $\mathbf{s}_N^N$ and delete it from
    $buf$
 4: while $\mathbf{s}$ is not a leaf do
 5:     EXPAND AND EVALUATE($\mathbf{s}$)
 6:     $\mathbf{s} \leftarrow$ find and set the minimum metric partial symbol vector and delete it from
    $buf$
 7: end while
 8: procedure EXPAND AND EVALUATE($\mathbf{s}_i^N$)  ▷ The input is the partial symbol vector
    to be expanded
 9:     for $j = 0$ to $|\Omega| - 1$ do
10:         if $\mathbf{s}_i^N = ()$ then  ▷ When expanding the root node the partial symbol vector
    is empty
11:             $\mathbf{s}_N^N \leftarrow \Omega[j]$
12:             Insert $\mathbf{s}_N^N$ in buffer $buf$
13:         else
14:             $s_{i-1} \leftarrow \Omega[j]$
15:             $\mathbf{s}_{i-1}^N \leftarrow (s_{i-1}, \mathbf{s}_i^N)$
16:             Insert $\mathbf{s}_{i-1}^N$ in buffer $buf$
17:         end if
18:     end for
19: end procedure
```

ML performance, (ii) it always expands and evaluates the minimum number of nodes as the number of antennas and modulation order is increasing and (iii) there is no need to set an initial radius. However, the main drawback of this algorithm is that in every iteration of the algorithm the minimum metric partial symbol vector has to be found and the size of storage buffer might increase significantly. The underlying architecture determines what is the best way to store and search for the minimum element. By using tree-based data structures or frequency tables sublinear time can be achieved. However, in case of parallel architectures a faster solution would be to store the data in arrays and perform parallel search on them.

4.7 Non-maximum likelihood tree-search based detectors

Non-ML tree-search based detectors show a very similar tree-traversal to the SD algorithm, however, restrictions and constraints are introduced in order to reduce or to make constant the number nodes visited, i.e., computational complexity. With the introduced constraints the BER performance of the detection is clearly degraded, thus, the ML performance is not achieved. In order to enhance the BER performance different

DOI:10.15774/PPKE.ITK.2015.010

preprocessing methods are applied, such as: lattice reduction, detection ordering, matrix regulariztion, improved decompositions. With these techniques significant improvements can be achieved with a slight increase in the computational complexity.

In the following sections some fundamental techniques are presented that will serve as a good basis for the PSD algorithm.

4.7.1 K-Best Sphere Detector algorithm

The K-Best SD algorithm is a widely used detector algorithm, because of its fixed complexity, constant memory usage and highly parallel possibilities. However, this approach is not exactly an SD because the symbol vectors are not discarded based on their path metrics. Instead there is a limit K of how many survivor symbol vectors per level will be expanded in the following iterations.

Algorithm 4 gives a detailed overview of the K-Best SD algorithm. It can be seen that this algorithm is based on a breadth-first traversal approach, however, only the K best path metric symbol vectors are kept and expanded on every level. One K and a $K \times |\Omega|$ element buffer satisfies the memory needs of this algorithm. The constant memory needs and highly parallel behaviour of the K-Best algorithm makes possible its efficient implementation on parallel architectures. In the following papers [78], [79], [80], [81] efficient mappings and implementations are presented for VLSI architectures. Some versions of the K-Best SD algorithm further reduce the resource requirements and the computational complexity by introducing different K values for different levels or by introducing the radius constraint. Radius dependent K-Best algorithms were proposed in [82], [83].

In Figs. 4.7 and 4.8 the BER performance of the K-Best detector for different values of K in 4×4 MIMO system with 16 and 64-QAM modulation are shown. For a 4×4 MIMO system with 16-QAM modulation the K-Best algorithm with $K = 16$ performs close to the ML detector until 25 dB SNR, however, at 30 dB there is a slight difference between the two detectors. In case of 64-QAM modulation the K-Best with $K = 32$ performs similar to the ML detector, however, in case of higher SNRs the BER of the ML detector is slightly better.

4.7.2 Hybrid tree-search detectors

Hybrid tree search detectors try to further reduce the computational complexity by defining two search stages: a BFS stage and a DFS stage. Usually, the search starts

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 4 K-Best SD algorithm for estimating $\mathbf{s} = (s_1, s_2, \cdots, s_N)$

```
Require: $\hat{\mathbf{s}}, \mathbf{R}, \Omega, K$
 1: $buf_{exp}$    ▷ The size of $buf_{exp}$ buffer is $K \times |\Omega|$ and the result of the K-best node expansion is stored here for every level.
 2: $buf_{kbest}$  ▷ The size of $buf_{kbest}$ buffer is $K$ and the K-best results are stored here in every iteration.
 3: $buf_{exp} \leftarrow$ EXPAND AND EVALUATE((),0)    ▷ Expand the root () of the tree and update $buf_{exp}$
 4: Sort in ascending order the symbol vectors in $buf_{exp}$ based on their path metric.
 5: Copy the first K-best symbol vectors from $buf_{exp}$ to $buf_{kbest}$
 6: for $i = N - 1$ to 2 do
 7:     for $k = 0$ to $K - 1$ do
 8:         $\mathbf{s}_i^N \leftarrow buf_{kbest}[k]$
 9:         $buf_{exp} \leftarrow$ EXPAND AND EVALUATE($\mathbf{s}_i^N$,k)
10:     end for
11:     Sort in ascending order the symbol vectors in $buf_{exp}$ based on their $M(\mathbf{s}_i^N)$ path metric.
12:     for $k = 0$ to $K - 1$ do
13:         $buf_{kbest}[k] \leftarrow buf_{exp}[k]$
14:     end for
15: end for
16: $\mathbf{s} = buf_{kbest}[0]$    ▷ The lowest path metric result is at index 0 in $buf_{kbest}$.
17: procedure EXPAND AND EVALUATE($\mathbf{s}_i^N$,$k$)    ▷ The input is the partial symbol vector to be expanded
18:     for $j = 0$ to $|\Omega| - 1$ do
19:         if $\mathbf{s}_i^N = ()$ then    ▷ When expanding the root node the partial symbol vector is empty
20:             $\mathbf{s}_N^N \leftarrow \Omega[j]$
21:             $buf_{exp}[k \cdot |\Omega| + j] \leftarrow \mathbf{s}_N^N$
22:         else
23:             $s_{i-1} \leftarrow \Omega[j]$
24:             $\mathbf{s}_{i-1}^N \leftarrow (s_{i-1}, \mathbf{s}_i^N)$
25:             $buf_{exp}[k \cdot |\Omega| + j] \leftarrow \mathbf{s}_{i-1}^N$
26:         end if
27:     end for
28: end procedure
```

DOI:10.15774/PPKE.ITK.2015.010

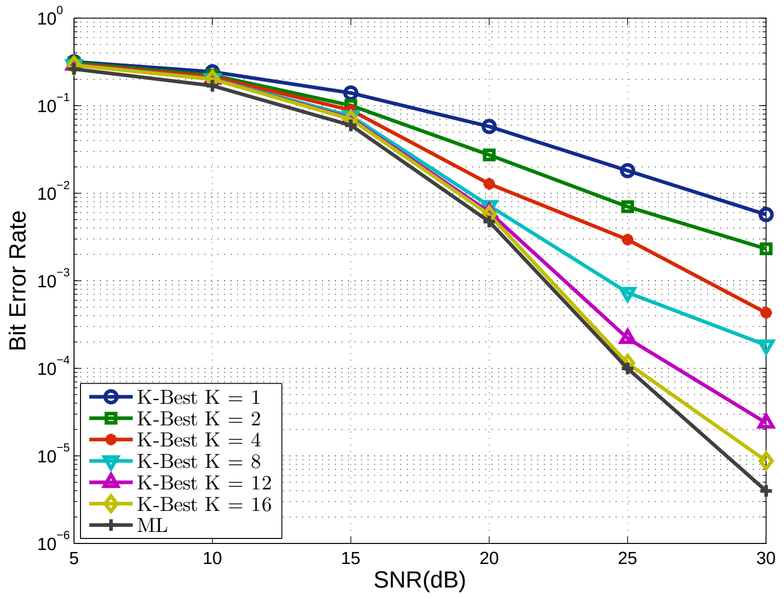


Figure 4.7: Bit error rate performance of the K-Best detector for $K = 1, 2, 4, 8, 12, 16$ in a 4×4 MIMO system with 16-QAM symbol constellation.

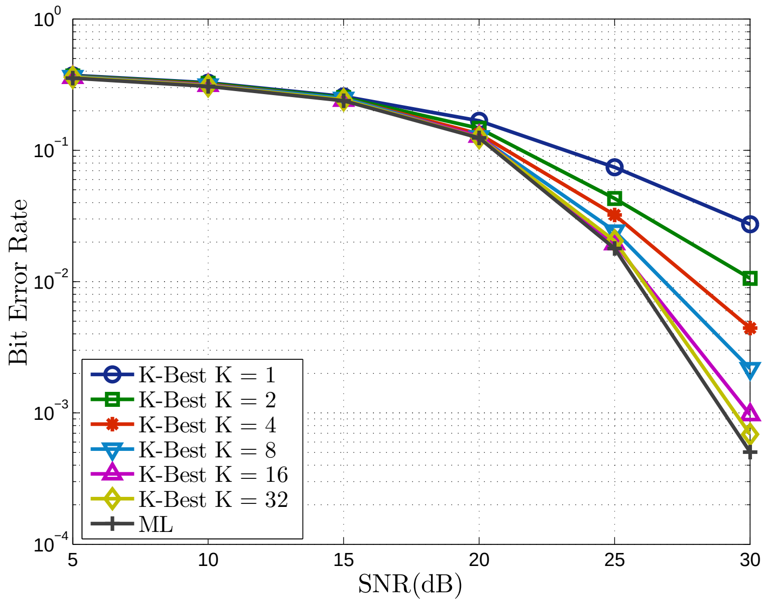


Figure 4.8: Bit error rate performance of the K-Best detector for $K = 1, 2, 4, 8, 16, 32$ in a 4×4 MIMO system with 64-QAM symbol constellation.

with a BFS followed by the evaluation of different metrics or probabilities based on the expanded nodes in order to determine the most likely paths that lead to the ML solution. Once the best nodes are found the search is continued with the DFS strategy on the resulting subtrees.

The main drawback of this hybrid search is that while the BFS can be implemented in a highly parallel manner, the sequential nature of the DFS is limiting the performance on a parallel architecture. In the following two different algorithms based on the hybrid tree search are discussed.

4.7.2.1 The Adaptive Reduced Breadth-First Search algorithm

Lai et al. in [84] have examined the possibility of the hybrid tree search. A two stage search was proposed. In the first stage a full-blown breadth-first (FBF) search is performed for N_{BF} levels followed by the sequential DFS tree traversal for every subtree. After the FBF search the symbol vectors $\mathbf{s}_{N_{BF}}^{N}$ representing the nodes on level N_{BF} are sorted based on the path metric $M(\mathbf{s}_{N_{BF}}^{N})$ of the symbol vectors. The sorted symbol vectors are denoted as $\mathbf{s}_{N_{BF}}^{N<0>}, \mathbf{s}_{N_{BF}}^{N<1>}, \cdots, \mathbf{s}_{N_{BF}}^{N<|\Omega|^{N_{BF}}>}$, where the lowest path metric is achieved by $\mathbf{s}_{N_{BF}}^{N<0>}$. For every expanded node $\mathbf{s}_{N_{BF}}^{N<j>}$ a subtree of depth $N - N_{BF}$ is associated, referred to as subtree j. Every subtree is traversed using the DFS similar as in the SD algorithm. The traversal starts on the subtree associated to the best path metric node. Whenever the traversal of subtree j is completed, subtree $j+1$ is selected. The DFS search is terminated if $M(\mathbf{s}_{N_{BF}}^{N<j+1>}) > d^2$ or all the subtrees have been searched. At this point the FBF-DFS algorithm finds the ML solution, the only difference compared to the SD algorithm is that there is a sorting on level N_{BF} and the DFS is performed in the ascending order of the path metrics.

In [84] the FBF-DFS algorithm was extended and further computational complexity reduction was achieved in the Adapting Reduced BF-DFS (ARBF-DFS) algorithm. The extensions of the ARBF-DFS algorithm are: (i) the channel dependent FBF level N_{BF} and (ii) the *reliability index* based tree pruning.

In the ARBF-DFS algorithm N_{BF} is allowed to vary indepently $1 \leq N_{BF} \leq N_{BF,max}$, where N_{max} denotes the maximum number of levels allowed for the BF stage. If the channel condition number is favorable N_{BF} is closer to one, thus, N_{BF} becomes a random variable. The worst case memory requirement is proportional to $\sim |\Omega|^{N_{BF,max}}$.

The second enhancement was the introduction of the *reliability index.* The aim of the reliability index is to help in the pruning of the tree, thus, the DFS stage is not

DOI:10.15774/PPKE.ITK.2015.010

performed on every subtree resulting in complexity reduction. In order to give a deeper insight, the partial ML solution is defined as $\mathbf{s}_{ML}^k = (s_k, s_{k+1}, \cdots, s_N)$. The reliability index is defined as $P(\mathbf{s}_k^{N<j+1>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R})$, namely the probability of $\mathbf{s}_k^{N<j+1>}$ being the correct path given the noise realization $\mathbf{n}_k$ and $\mathbf{R}$. A straightforward way to determine when to stop the FBF stage is to define a threshold T and evaluate if

$$P(\mathbf{s}_k^{N<1>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R}) > T. \tag{4.48}$$

Furthermore $L_k^{<j>}$ is define as follows:

$$L_k^{<j>} = \ln \left(\frac{P(\mathbf{s}_k^{N<1>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R})}{P(\mathbf{s}_k^{N<j>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R})} \right) = \frac{M(\mathbf{s}_k^{N<j>}) - M(\mathbf{s}_k^{N<1>})}{\sigma^2} \tag{4.49}$$

where the final result is achieved by using the path metric formula and the Gaussian probability density function. Suming the probabilities leads to a different form of the reliability index as follows:

$$\begin{aligned} \sum_{j=1}^{|\Omega|^k} P(\mathbf{s}_k^{N<j>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R}) = 1 \\ P(\mathbf{s}_k^{N<1>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R}) = \frac{1}{\sum_{j=1}^{|\Omega|^k} e^{-L_k^{<j>}}} \end{aligned} \tag{4.50}$$

The computation of Eq. 4.50 could be very expensive if k is large, thus, it can be simplified by computing only the first two terms of the sum. In this case the approximation of Eq. 4.50 becomes

$$P(\mathbf{s}_k^{N<1>} = \mathbf{s}_{ML}^k | \mathbf{n}_k, \mathbf{R}) = \frac{1}{\sum_{j=1}^{|\Omega|^k} e^{-L_k^{<j>}}} \approx \frac{1}{1 + e^{-L_k^{<2>}}} \tag{4.51}$$

By substituting Eq. 4.49 in 4.51 and rewriting Eq. 4.48 the resulting inequality is

$$M(\mathbf{s}_k^{N<2>}) > M(\mathbf{s}_k^{N<1>}) + \sigma^2 \ln \left(\frac{T}{1-T} \right). \tag{4.52}$$

This result can be further generalized, namely every partial symbol vector on levels $1 < k < N_{BF}$ whose path metric exceed the best path metric with threshold $\sigma^2 \ln \left(\frac{T}{1-T} \right)$ are pruned.

The introduced pruning reduces the average floating point operations by 2-5 times compared to the SD algorithm. The BER performance is highly influenced by the chosen threshold T. BER simulations with $T = 0.9$ achieved near-ML performance.

DOI:10.15774/PPKE.ITK.2015.010

4.7.2.2 The Fixed-Complexity Sphere Detector algorithm

The Fixed-Complexity Sphere Detector (FSD) was introduced by Barbero et al. in [56]. The FSD algorithm tree traversal is built on a hybrid scheme as well. However, several important differences are introduced compared to the ARBF-DFS algorithm. The main features of the FSD: (i) a channel independent tree traversal process consisting of full BFS on several levels followed by a single DFS path based on the SE enumeration and (ii) a novel channel matrix ordering method.

A conjecture was presented about the number of nodes that have to be visited in an uncoded MIMO system in order to achieve near-ML performance. The result is that two stages are defined: (i) in the FBF stage the maximum number of nodes are considered for N_{BF} levels and (ii) in the single path DFS stage only one child node is further expanded for N_{SE} levels and the selected child is based on the SE enumeration. The levels sum of the two stages $N_{BF}+N_{SE}=N$ equals the depth of the tree. The above configuration implies that the number of predefined paths is fixed and is equal $|P| = |\Omega|^{N_{BF}} \cdot 1^{N_{SE}}$. In [85] it was shown that FSD achieves the same diversity as the ML detection if $N_{BF} \geq \sqrt{M}-1$. The advantage of the predefined paths is the resulting rigid structure that enables the parallel implementation of the FSD algorithm and has the same computational complexity for every SNR value.

Another important part of this algorithm is the channel matrix ordering method. It determines the order of the symbols detection based on the stages of the algorithm. The aim is to detect symbols with the highest post-detection noise amplification in the FBF stage and symbols with the smallest post-detection noise amplification in the DFS stage. The motivation behind this strategy is to keep all the paths where the error probability is high and after that it is enough to follow only a single path from every expanded node because the probability of making a bad decision with the SE enumeration is low. The symbol $\mathbf{s_k}$ to be detected is selected according to

$$k = \begin{cases} \arg\max_{j} \|(\mathbf{H_i}^{\dagger})_j\|^2 & \text{, if } i \text{ is a FBF level} \\ \arg\min_{j} \|(\mathbf{H_i}^{\dagger})_j\|^2 & \text{, if } i \text{ is a DF level} \end{cases}$$

where $(\mathbf{H_i}^{\dagger})_j$ denotes the j-th row of $\mathbf{H_i}^{\dagger}$ and the calculation of matrix $\mathbf{H_i}^{\dagger}$ is detailed in Alg. 1 Sec. 4.4.2.

The FSD achieved near-ML BER performance for smaller systems, however, the difference of the BER is getting bigger as the number of antennas are increasing or channel

DOI:10.15774/PPKE.ITK.2015.010

correlation is introduced. The FSD was mapped for different parallel architectures. Mapping details and throughput measurements are presented in papers [22] ,[86] and [87].

4.8 The Parallel Sphere Detector algorithm

Section 4.6.1 concluded that the SD algorithm can be regarded as a branch and bound tree search problem. Thus, the ideas presented in Sections 4.6.2, 4.7.1 and 4.7.2 can serve as a good starting point in order to eliminate the drawbacks of the SD algorithm.

Khairy et al. showed in [57] that significant speed-up can be achieved by executing multiple sequential SDs simultaneously. However, to achieve even better performance it is mandatory to redesign the sequential algorithm using several effective parallel design patterns in order to exploit all advantages of parallel computing capabilities of multi-core and many-core architectures.

In Sec 4.7.2 the ARBF-DFS and the FSD algorithms were presented. The ARBF-DFS algorithm was introduced by Lai et al. in [84]. They have examined the possibility of the hybrid tree search. A two stage search was proposed, in the first stage a full BFS has been performed, followed by a DFS on every subtree. The main drawback of this hybrid search is that while the BFS can be implemented in a highly parallel manner, the sequential nature of the DFS is limiting the performance of a parallel architecture. However, it was shown, that starting the DFS subtree traversal from the best metric node the overall number of visited nodes is smaller compared to the SD algorithm.

The FSD algorithm also implements a two stage tree search method. In the first stage a full BFS is performed, followed by the SE enumeration on the previously expanded nodes. The benefit of this approach is the rigid structure that can be parallelized, however, the ML BER performance is not achieved.

The K-BEST algorithm discussed in Sec. 4.7.1 is based on the combination of BFS stages combined with sorting the nodes based on their path metric. The advantages of this approach are the fixed memory requirements that can be adapted to the architecture used and the parallel implementation possibilities. However, it might happen that the number of visited nodes is higher compared to the case of the SD algorithm.

The tasks during the design process of the new PSD algorithm are to apply the complexity reduction techniques discussed in the previous sections, to eliminate the limiting sequential parts of the SD algorithm and to make the algorithm customization possible so it can satisfy the requirements and to fully utilize the resources of different parallel architectures. The results of this Section form **Thesis group I** and **II**.

DOI:10.15774/PPKE.ITK.2015.010

4.8.1 Design objectives of the Parallel Sphere Detector algorithm

The key concepts of the PSD algorithm design process are as follows:

1. Consider an arbitrary lattice $\mathbf{\Lambda}$. The search of the optimal (covering) radius requires a number of steps that grows exponentially [76] with the dimension of the lattice, thus, its use is not practical. A possible solution to the initial radius problem is the ZF radius, since this radius guarantees the existence of at least one lattice point. However, it may happen that this choice of radius will yield too many lattice points lying inside the sphere. The ZF radius is defined as follows $d = \|\mathbf{y} - \mathbf{H}\hat{\mathbf{s}}_B\|$ where $\hat{\mathbf{s}}_B = \lfloor \hat{\mathbf{s}} \rceil$ is the Babai estimate and operator $\lfloor \cdot \rceil$ slices each element of the input vector to the closest symbol in the symbol set Ω.

   However, one of the design objectives is to make the detection completely independent of the initial radius size and to ensure that the detection process does not have to be restarted with an increased radius. By defining the initial radius as $d = \infty$ the above condition is fulfilled, but the SD algorithm becomes an exhaustive search. Consequently, the algorithm has to find a small path metric leaf node as fast as possible, in order to adjust the radius to the path metric of the leaf node.

2. The redesigned SD algorithm has to support parallel architectures. This can be achieved by introducing a new work generation and distribution mechanism that is able to keep all the processing elements busy continuously. By expanding and evaluating multiple symbol vectors simultaneously the above goal could be reached. However, the extent of parallelism should be controlled by well defined parameters so that the new algorithm can adjust itself to any multi-core or many-core architecture.

3. Parallel architectures may have different memory hierarchies. In order to make use of faster but usually smaller sized memories possible, the algorithm should have different parameter configuration in order to assure the efficient use of memory.

4.8.2 General description of the Parallel Sphere Detector algorithm

The parallelism of the SD algorithm is achieved by a hybrid tree search. The branching factor of the tree is equal to $|\mathbf{\Omega}|$. The depth of the tree depends on the number N of transmit antennas.

Algorithm 5 gives a high-level overview of the PSD algorithm. The definitions of the parameters used to describe the PSD algorithm are given in Table 4.1. The key

DOI:10.15774/PPKE.ITK.2015.010

Table 4.1: Definition of parameters used in the Parallel Sphere Detector algorithm.

Tree traversal parameters		
lvl_{nr}	the total number of tree levels where partial symbol vectors are evaluated	$0 < lvl_{nr} \leq N$
lvl_x	levels assigned for partial symbol vector evaluation	$lvl_0 = N+1$, $lvl_{lvl_{nr}} = 1$ $lvl_x > lvl_{x+1}$
exp_{lvl_x}	number of partial symbol vectors expanded simultaneously on level lvl_x	$exp_{lvl_0} = 1$ $exp_{lvl_x} \leq eval_{lvl_x}$
$eval_{lvl_x}$	number of partial symbol vectors needed to be evaluated on level lvl_x after the expansion of partial symbol vectors on level lvl_{x-1}	$eval_{lvl_x} = exp_{lvl_{x-1}} \lvert\Omega\rvert^{(lvl_{x-1} - lvl_x)}$
max_{lvl_x}	maximum number of partial symbol vectors on level lvl_x	$max_{lvl_x} = \lvert\Omega\rvert^{(lvl_0 - lvl_x)}$
Algorithm parameters		
tt	total number of threads assigned for detection	
t_{id}^k	thread with identifier k	
buf_{lvl_x}	buffer for the evaluated partial symbol vectors $\mathbf{s}_{lvl_x}^N$ on level lvl_x	$size(buf_{lvl_x}) = eval_{lvl_x}$
off_{lvl_x}	offset of processing on level lvl_x for buf_{lvl_x}	$0 \leq off_{lvl_x} \leq eval_{lvl_x}$
$\mathbf{s}_{lvl_x}^{N<j>}$	partial symbol vector on level lvl_x where j is the index of the partial symbol vector in buffer buf_{lvl_x}	$0 \leq j < eval_{lvl_x}$
vt_{lvl_x}	virtual thread identifier calculated from lvl_x, t_{id}^k and tt	Based on Eq. 4.54
vb_{lvl_x}	virtual block identifier calculated from lvl_x, t_{id}^k and tt	Based on Eq. 4.55

DOI:10.15774/PPKE.ITK.2015.010

parameters that determine the overall performance of the algorithm are: lvl_{nr}, lvl_x and exp_{lvl_x}. These parameters define the tree traversal process, determine the memory usage and, consequently, influence (i) the speed of reaching a leaf node, (ii) the metric of the first leaf node and (iii) the number of iterations required to find the optimal solution.

To get a better insight Table 4.2 shows a few valid parameter sets for different system configurations. Parameters lvl_x and exp_{lvl_x} are similar for configurations 1, 2 and 3. However, the size of the symbol set is different resulting in a significant change in the memory requirements. Note, that different parameters have to be used for the various system configurations and symbol sets.

Algorithm 5 High-level overview of the Parallel Sphere Detector algorithm

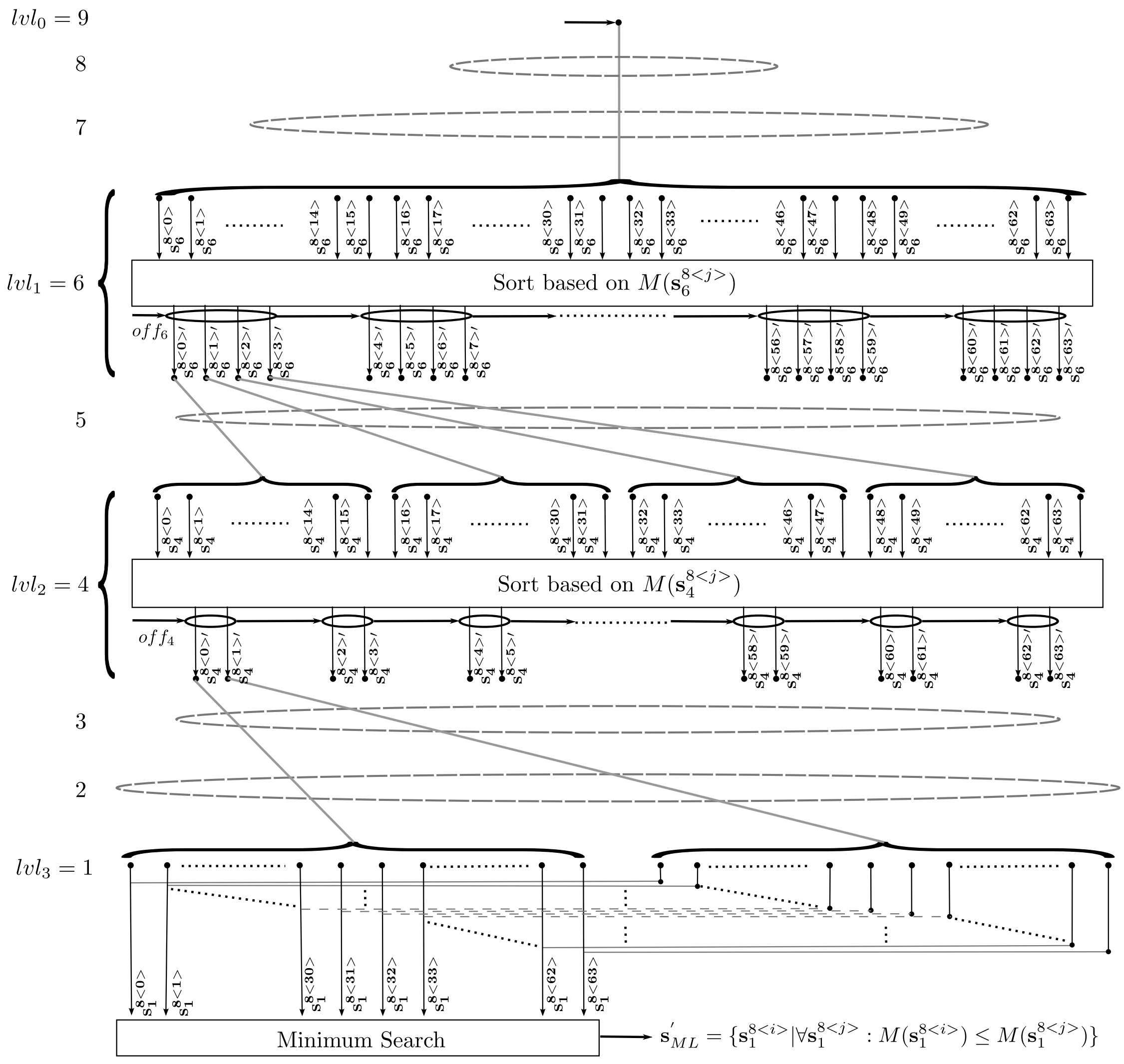
```
1: Expand and evaluate several nodes simultaneously for distinct levels.    ▷ This
   ensures enough computational load to keep the cores active.
2: Repeat steps 1-6 until a leaf level is reached:
   1. Sort the previously expanded nodes by their path metric.
   2. **if** the path metric of the first node in the sorted list is smaller than the sphere
      radius **then**
   3.    Expand nodes further from a subset of nodes sorted previously for the following
         distinct level.
   4. **else**
   5.    Step back to the previous level and continue the expansion of the next subset
         of previously sorted nodes.
   6. **end if**
3: When a leaf level is reached:
   1. Find the leaf with minimum metric and update the sphere radius.
   2. Proceed with the rest of the nodes evaluated at the previous level.
```

Figure 4.9 shows the PSD schematic for *configuration* 4 defined in Table 4.2. The levels referred to below are identified on the left side of the figure. The detection process starts from the root of the tree on level $lvl_0 = 9$. The partial symbol vector is empty on this level.

One of the key features of the PSD algorithm is the tree traversal process. That means that instead of evaluating the path metrics $M(\mathbf{s}_8^{8<j>})$ of partial symbol vectors $\mathbf{s}_8^{8<j>}$ on level 8, as done in the SD algorithm, the first node evaluation takes place at $lvl_1 = 6$. By expanding the root node of the tree, $eval_{lvl_1} = 64$ partial symbol vectors are generated and evaluated on level $lvl_1 = 6$. Note, levels 8 and 7 are skipped. Thus, there is no symbol vector expansion and evaluation on those levels.

After evaluating the obtained partial symbol vectors $\mathbf{s}_6^{8<j>}$, a sorting is applied based on their path metrics $M(\mathbf{s}_6^{8<j>})$. The sorted symbol vectors are denoted as $\mathbf{s}_6^{8<j>'}$ with $\mathbf{s}_6^{8<0>'}$ as the partial symbol vector with the lowest metric. When moving towards to the

DOI:10.15774/PPKE.ITK.2015.010



$lvl_0 = 9$
8
7
$lvl_1 = 6$
Sort based on $M(\mathbf{s}_6^{8<j>})$
off_6
5
$lvl_2 = 4$
Sort based on $M(\mathbf{s}_4^{8<j>})$
off_4
3
2
$lvl_3 = 1$
Minimum Search
$\mathbf{s}'_{ML} = \{\mathbf{s}_1^{8<i>} | \forall \mathbf{s}_1^{8<j>} : M(\mathbf{s}_1^{8<i>}) \leq M(\mathbf{s}_1^{8<j>})\}$

Figure 4.9: The hybrid tree traversal of the Parallel Sphere Detector algorithm for a 4×4 MIMO system with $|\Omega| = 4$.

DOI:10.15774/PPKE.ITK.2015.010

Table 4.2: Valid Parallel Sphere Detector algorithm parameter configurations.

Configuration	1	2	3	4	5	6
Antennas	2×2	2×2	2×2	4×4	4×4	4×4
Symbol set size	2	4	8	4	8	8
lvl_{nr}	2	2	2	3	4	4
lvl_0	5	5	5	9	9	9
lvl_1	2	2	2	6	7	7
lvl_2	1	1	1	4	6	6
lvl_3	0	0	0	1	3	2
lvl_4	0	0	0	0	1	1
exp_{lvl_0}	1	1	1	1	1	1
exp_{lvl_1}	4	4	4	4	2	2
exp_{lvl_2}	0	0	0	2	3	3
exp_{lvl_3}	0	0	0	0	4	4
$eval_{lvl_1}$	8	64	512	64	64	64
$eval_{lvl_2}$	8	16	32	64	16	16
$eval_{lvl_3}$	0	0	0	128	1536	12288
$eval_{lvl_4}$	0	0	0	0	256	32
$\sum_{x=1}^{lvl_{nr}} eval_{lvl_x}$	16	80	544	256	1872	12400

next level $lvl_2 = 4$, the $exp_{lvl_1} = 4$ best metric partial symbol vectors are selected and expanded from the previous level $lvl_1 = 6$. As a result, the partial symbol vectors $\mathbf{s}_4^{8<j>}$ are generated.

Note that a hybrid search is realized at this point. On level $lvl_1 = 6$ a full BFS is performed with a DFS continued with $exp_{lvl_1} = 4$ partial symbol vectors having the best metric. Until $lvl_2 = 4$ every possible symbol combination is evaluated and this process can be regarded as a BFS. This is how the two searching strategies are combined, resulting in no latency, delay or bottleneck.

If the inequality $M(\mathbf{s}_{lvl_x}^{N<off_{lvl_x}>}) < d^2$ does not hold, instead of increasing the corresponding offset off_{lvl_x}, the search is stopped on that level and the offset's value is updated to 0 with the search continued on lvl_{x-1}. The search can be stopped at a specific level because the partial symbol vectors are sorted by their path metric. Thus, if $M(\mathbf{s}_{lvl_x}^{N<j>}) > d^2$ then the remaining partial symbol vectors will have a higher path metric.

The selection, expansion, evaluation and sorting steps discussed above are repeated until the last level $lvl_3 = 1$ is reached. Upon reaching the last level, the symbol vector with the lowest metric has to be found. At level $lvl_3 = 1$, instead of sorting, a minimum search is performed. If a symbol vector $\mathbf{s}_1^8$ with the lowest metric satisfies the condition $M(\mathbf{s}_1^8) < d^2$, then a new ML candidate has been found. If an ML candidate already exists from a previous iteration then it is compared with the new candidate and the one

DOI:10.15774/PPKE.ITK.2015.010

with the smaller metric will become the new solution $\mathbf{s}_{ML} = \mathbf{s}_1^8$ and the sphere radius is adjusted. The further flow of the detection process is similar to the flow of the SD algorithm.

By sorting on each level the lowest path metric partial symbol vectors are found and the search is continued by expanding them. With this greedy strategy, where on each processed level locally optimal choices are made, a near-ML solution is found in a few iterations and the updated radius metric reduces the search space significantly. This is why the initial condition $d^2 = \infty$ is admissible.

The SE method first enumerates the symbols that are closer to the unconstrained least squares solution. Consequently, on every level the search is started with the corresponding symbol in the Babai estimate and it is continued in a zig-zag enumeration with the rest of the symbols in the symbol set. However, the locally optimal choice does not necessarily lead to the optimal ML solution. In case of the PSD algorithm the distance between consecutive levels can be greater than one and the search is always continued with the lowest path metric nodes. As a result, the effect of previously chosen symbols is propagated through several levels and the optimum is reached with a higher probability compared to the SE enumeration.

Algorithm 6 gives a detailed and precise description of the PSD algorithm. To make a comparison of the SD and PSD algorithms as easy as possible, the same notation is used in Algorithms 2 and 6. Both algorithms are divided into three main procedures: (i) *Definition and Initialization of Variables*, (ii) control of the tree *Traversal Process* and (iii) the *Expansion and Evaluation* of the tree nodes. The main differences between the SD and PSD algorithms are highlighted in Table 4.3.

In the *Definition and Initialization of Variables* procedure the main steps are as follows: (i) memory allocation for buffers on different levels, (ii) generating data for the first buffer and (iii) starting the tree traversal process. As shown in Table 4.3, the number of buffers is equal to the number of processed tree levels. In the SD algorithm, each buffer has a constant size that is equal to the number of symbols in the symbol set. In the PSD algorithm, the number of buffers is equal to lvl_{nr} where the size of buffers depends on both the lvl_x and exp_{lvl_x} parameters.

The *Traversal Process* procedure controls the tree traversal. In the case of finding a leaf node with a smaller path metric than found previously it updates the radius. The traversal process is implemented in a very different manner in the PSD and SD algorithms. While the breadth traversal of the tree, controlled by the offset variables off_{lvl_x}, is

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 6 Parallel Sphere Detector algorithm for estimating $\mathbf{s}_{ML} = (s_1, s_2, \cdots, s_N)$

```
Require: $\hat{\mathbf{s}}, \mathbf{R}, |\Omega|, lvl_{nr}, lvl_{0,1,2,\cdots,lvl_{nr}}, exp_{lvl_0,lvl_1,\cdots,lvl_{nr-1}}, tt$
 1: procedure Definition and Initialization of Variables
 2:     for $j = 1$ to $lvl_{nr}$ do
 3:         $eval_{lvl_j} \leftarrow exp_{lvl_{j-1}} \cdot |\Omega|^{lvl_{j-1}-lvl_j}$   ▷ Number of partial symbol vector evaluations on level $lvl_j$
 4:         Let $buf_{lvl_j}[eval_{lvl_j}] = \{\}$   ▷ Denote an empty buffer of size $eval_{lvl_j}$ for level $lvl_j$
 5:         $off_{lvl_j} \leftarrow 0$   ▷ Offset of processing on level $lvl_j$ for buffer $buf_{lvl_j}$
 6:     end for
 7:     $buf_{lvl_1} \leftarrow$ Expand and Evaluate($\{()\}$)   ▷ Expand the root node () of the tree and update $buf_{lvl_1}$
 8:     Traversal Process($i \leftarrow 2$)
 9: end procedure
10: procedure Traversal Process($i$)
11:     while $i > 1$ do
12:         if $off_{lvl_{i-1}} < eval_{lvl_{i-1}}$ then
13:             if $M(\mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}>}) < d^2$ then   ▷ Where $\mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}>}$ is the element of $buf_{lvl_{i-1}}$ at index $off_{lvl_{i-1}}$
14:                 if $i = lvl_{nr}$ then
15:                     $\mathbf{s}_{ML}^{'} \leftarrow$ Expand and Evaluate($\{\mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}>}, \cdots, \mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}+(exp_{lvl_{i-1}}-1)>}\}$)
16:                     $d_{temp}^2 \leftarrow \|\mathbf{R}(\mathbf{s}_{ML}^{'} - \hat{\mathbf{s}})\|^2$
17:                     If $d_{temp}^2 < d^2$ then $d^2 \leftarrow d_{temp}^2$ and $\mathbf{s}_{ML} \leftarrow \mathbf{s}_{ML}^{'}$ end if
18:                     $off_{lvl_{i-1}} \leftarrow off_{lvl_{i-1}} + exp_{lvl_{i-1}}$
19:                 else
20:                     $buf_{lvl_i} \leftarrow$ Expand and Evaluate($\{\mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}>}, \cdots, \mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}+(exp_{lvl_{i-1}}-1)>}\}$)
21:                     $off_{lvl_{i-1}} \leftarrow off_{lvl_{i-1}} + exp_{lvl_{i-1}}$, $i \leftarrow i + 1$
22:                 end if
23:             else
24:                 $off_{lvl_{i-1}} \leftarrow 0$, $i \leftarrow i - 1$
25:             end if
26:         else
27:             $off_{lvl_{i-1}} \leftarrow 0$, $i \leftarrow i - 1$
28:         end if
29:     end while
30: end procedure
31: procedure Expand and Evaluate($\{\mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}>}, \mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}+1>}, \cdots, \mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}+(exp_{lvl_{i-1}}-1)>}\}$)
    ▷ The input is the array of partial symbol vectors to be expanded
32:     for $n = 0$ to $\lceil eval_{lvl_i}/tt \rceil - 1$ do
33:         $ind \leftarrow t_{id}^k + n \cdot tt$
34:         $vt_{lvl_i} \leftarrow (t_{id}^k + n \cdot tt) \mod |\Omega|^{(\mathbf{lvl_{i-1}-lvl_i})}$   ▷ Virtual thread identifier based on Eq. 4.54
35:         $vb_{lvl_i} \leftarrow \lfloor (t_{id}^k + n \cdot tt)/|\Omega|^{(lvl_{i-1}-lvl_i)} \rfloor$   ▷ Virtual block identifiers based on Eq. 4.55
36:         $\mathbf{s}_{lvl_{i-1}}^N = \mathbf{s}_{lvl_{i-1}}^{N<off_{lvl_{i-1}}+vb_{lvl_i}>} \leftarrow vb_{lvl_i}$   ▷ Select partial symbol vector $\mathbf{s}_{lvl_{i-1}}^N$ from the input array based on $vb_{lvl_i}$
37:         $\mathbf{s}_{lvl_i}^{(lvl_{i-1}-1)} = (s_{lvl_i}, \cdots, s_{(lvl_{i-1}-2)}, s_{(lvl_{i-1}-1)}) \leftarrow vt_{lvl_i}$   ▷ Create partial symbol vector $\mathbf{s}_{lvl_i}^{(lvl_{i-1}-1)}$ based on $vt_{lvl_i}$
38:         $\mathbf{s}_{lvl_i}^N \leftarrow (s_{lvl_i}, \cdots, s_{(lvl_{i-1}-2)}, s_{(lvl_{i-1}-1)}, \mathbf{s}_{lvl_{i-1}}^N) = (\mathbf{s}_{lvl_i}^{(lvl_{i-1}-1)}, \mathbf{s}_{lvl_{i-1}}^N)$   ▷ Merge $\mathbf{s}_{lvl_{i-1}}^N$ and $\mathbf{s}_{lvl_i}^{(lvl_{i-1}-1)}$
39:         $buf_{lvl_i}[ind] = \mathbf{s}_{lvl_i}^N$
40:     end for
41:     if $lvl_i = 1$ then
42:         return $\mathbf{s}_{ML}^{'}$, which is the minimum path metric symbol vector in $buf_{lvl_i}$
43:     else
44:         return $buf_{lvl_i}$, where the partial symbol vectors are sorted based on the path metric $M(\mathbf{s}_{lvl_i}^N)$
45:     end if
46: end procedure
```

DOI:10.15774/PPKE.ITK.2015.010

Table 4.3: Algorithmic comparison of the Parallel Sphere Detector with the sequential Sphere Detector algorithm.

	Definition and Initialization of Variables	
	Number of buffers used	Accumulated buffer size
SD	N	$N \cdot \|\Omega\|$
PSD	$0 < lvl_{nr} \leq N$	$\sum_{x=1}^{lvl_{nr}} exp_{lvl_{x-1}} \cdot \|\Omega\|^{(lvl_{x-1}-lvl_x)}$
	Traversal process	
	Horizontal traversal	Vertical traversal
SD	$off_x \leftarrow off_x + 1$	$lvl_x - lvl_{x+1} = 1$
PSD	$off_{lvl_x} \leftarrow off_{lvl_x} + exp_{lvl_x}$	$1 \leq lvl_x - lvl_{x+1} \leq N$
	Expand and Evaluate	
	Newly evaluated partial symbol vectors in one iteration	
SD	$\|\Omega\|$	
PSD	$exp_{lvl_{x-1}} \cdot \|\Omega\|^{(lvl_{x-1}-lvl_x)}$	

always one in the SD algorithm, the PSD algorithm changes the offset variables based on the number of paths chosen on a specific level as follows from $off_{lvl_x} \leftarrow off_{lvl_x} + exp_{lvl_x}$. The depth traversal of the tree is controlled by the parameters lvl_x. While in the SD algorithm the difference between consecutive levels is always one, i.e., $lvl_x - lvl_{x+1} = 1$, the PSD can skip levels if $lvl_x - lvl_{x+1} > 1$. Using this technique the leaf nodes can be reached faster.

The *Expand and Evaluate* procedure is responsible for generating the new partial symbol vectors and to evaluate their metrics. During the expansion of a tree node its child nodes are defined, i.e., the partial symbol vector denoting the tree node is updated with new symbols that are representing the child nodes. The evaluation of a partial symbol vector is the calculation of its path metric. A detailed description of this process is given in Sec. 4.8.3. Depending on the parameters chosen, the amount of newly expanded and evaluated partial symbol vectors can be significantly higher in the PSD algorithm than that in the SD one. More details are given in Table 4.3. Since different nodes can be expanded and evaluated independently from each other, this can be done in parallel. As the generated work can be controlled with well defined parameters, the PSD algorithm can be adjusted to several computing platforms.

4.8.3 The main building blocks of the Expand and Evaluate pipeline

The operating principle and structure of the PSD algorithm was discussed in the previous section. All computations done on one level in the PSD algorithm shown in Fig. 4.9 are performed by the *Expand and Evaluate Pipeline* (EEP) depicted in Fig. 4.10. First a detailed description of EEP is given. Then, the iterative implementation of the

PSD algorithm with EEP blocks is discussed. For a detailed description of variables used by EEP refer to Table 4.1.

The stages of the EEP are as follows: (i) preparation of data sets for the partial symbol vectors, referred to as *Preparatory Block*, (ii) preparation of partial symbol vectors, referred to as *Selecting, Mapping and Merging Block*, (iii) metric calculation for each partial symbol vector, referred to as *Path Metric Evaluation Block*, (iv) sorting based on the calculated path metrics or finding the symbol vector with the smallest path metric, referred to as *Searching or Sorting Block*.

The operation principle of the EEP is given in the following subsections.

4.8.3.1 Preparatory block

In order to form a symbol vector $\mathbf{s}_{lvl_x}^{N<j>}$ on level lvl_x parameters such as vt_{lvl_x} and vb_{lvl_x} have to be defined. The work assigned for one thread depends on the number of symbol vectors needed to be evaluated on a given level and on the number of the threads launched. If the condition

$$eval_{lvl_x} \leq tt \tag{4.53}$$

is met then one symbol vector has to be evaluated by one thread. Otherwise one thread has to be assigned to process at most $\lceil eval_{lvl_x}/tt \rceil$ number of symbol vectors. A full BFS will take place on level lvl_x in the case when the condition $exp_{lvl_{x-1}} = max_{lvl_{x-1}}$ holds because all the symbol vectors on level lvl_x are expanded simultaneously. Assuming that $eval_{lvl_x}$ is divisible by tt, two sets are defined for each thread t_{id}^k: (i) set $VT_{lvl_x}^k$ containing the virtual thread identifiers and (ii) set $VB_{lvl_x}^k$ containing the virtual block identifiers. The virtual identifiers are computed in the following manner:

$$\begin{gathered} VT_{lvl_x}^k = \{vt_{lvl_x} | vt_{lvl_x} = (t_{id}^k + n \cdot tt) \mod |\Omega|^{(lvl_{x-1} - lvl_x)}, \\ n = 0 : \lceil eval_{lvl_x}/tt \rceil - 1\}, \end{gathered} \tag{4.54}$$

$$\begin{gathered} VB_{lvl_x}^k = \{vb_{lvl_x} | vb_{lvl_x} = \lfloor (t_{id}^k + n \cdot tt)/|\Omega|^{(lvl_{x-1} - lvl_x)} \rfloor, \\ n = 0 : \lceil eval_{lvl_x}/tt \rceil - 1\} \end{gathered} \tag{4.55}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x and $\lceil x \rceil$ is the smallest integer not less than x.

Each thread has to compute its own set of identifiers for every level. This first block,

DOI:10.15774/PPKE.ITK.2015.010

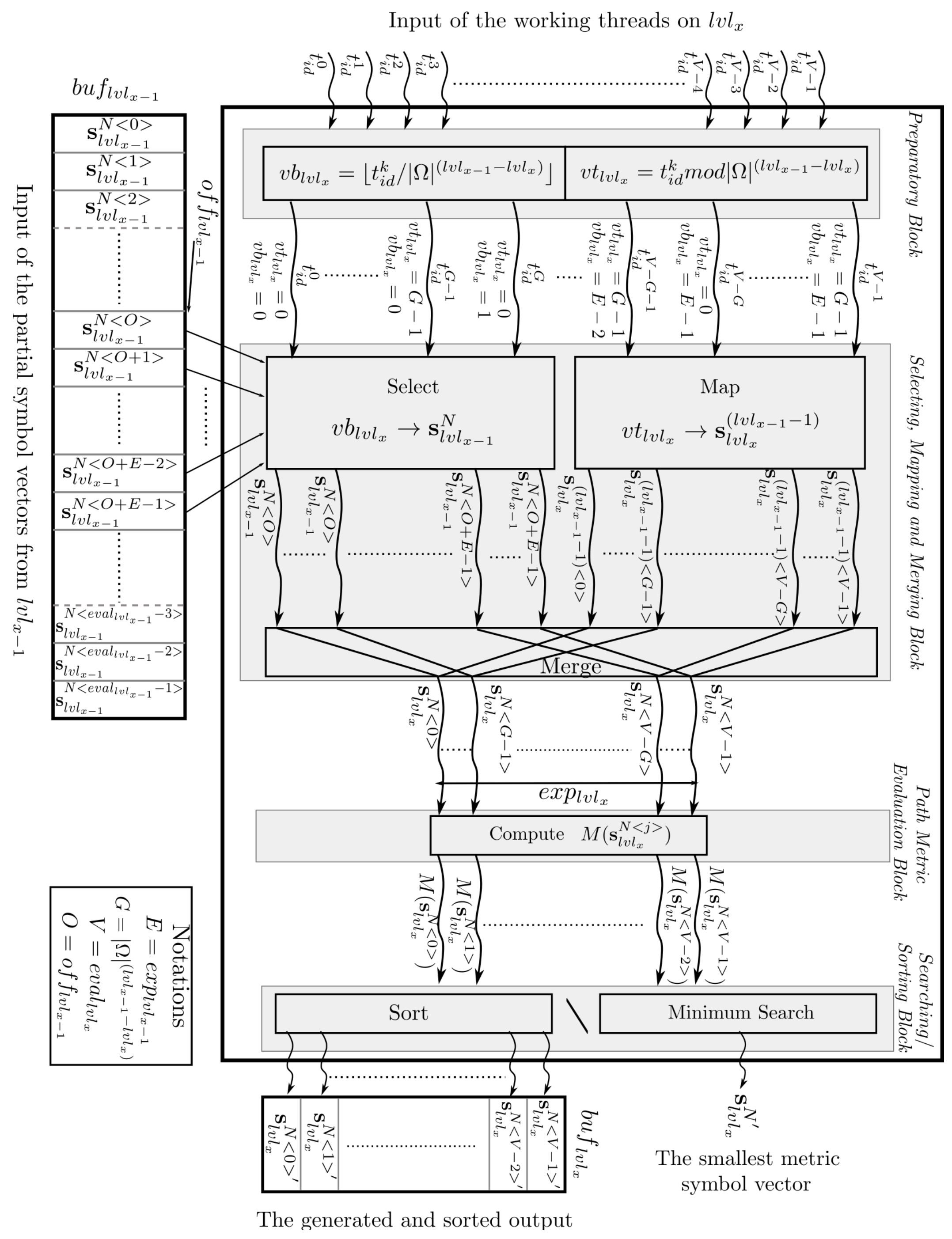

Input of the working threads on lvl_x
$buf_{lvl_{x-1}}$
Input of the partial symbol vectors from lvl_{x-1}
$off_{lvl_{x-1}}$
Preparatory Block
$vb_{lvl_x} = \lfloor t_{id}^k / |\Omega|^{(lvl_{x-1}-lvl_x)} \rfloor$
$vt_{lvl_x} = t_{id}^k mod |\Omega|^{(lvl_{x-1}-lvl_x)}$
Selecting, Mapping and Merging Block
Select
$vb_{lvl_x} \rightarrow \mathbf{s}_{lvl_{x-1}}^N$
Map
$vt_{lvl_x} \rightarrow \mathbf{s}_{lvl_x}^{(lvl_{x-1}-1)}$
Merge
exp_{lvl_x}
Path Metric Evaluation Block
Compute $M(\mathbf{s}_{lvl_x}^{N<j>})$
Searching/ Sorting Block
Sort
Minimum Search
buf_{lvl_x}
The smallest metric symbol vector
The generated and sorted output
Notations
$E = exp_{lvl_{x-1}}$
$G = |\Omega|^{(lvl_{x-1}-lvl_x)}$
$V = eval_{lvl_x}$
$O = off_{lvl_{x-1}}$

Figure 4.10: The block diagram of the Expand and Evaluate pipeline.

referred to as *Preparatory Block*, is completed when each thread has finished computing the virtual identifiers.

4.8.3.2 Selecting, mapping and merging block

In the *Selecting, Mapping and Merging* block the task is to generate symbol vectors $\mathbf{s}_{lvl_x}^{N<j>}$ for level lvl_x.

In the *Selecting* phase, $exp_{lvl_{x-1}}$ number of previously evaluated symbol vectors $\mathbf{s}_{lvl_{x-1}}^{N}$ are selected from $buf_{lvl_{x-1}}$ serving as inputs to this process. The *selection* is performed based on the virtual block identifiers and the corresponding offset $off_{lvl_{x-1}}$. The virtual block identifiers vb_{lvl_x} are computed based on (4.55). Each $vb_{lvl_x} \in VB_{lvl_x}^{k}$ serves as an index of the input partial symbol vector array. The selected partial symbol vector $\mathbf{s}_{lvl_{x-1}}^{N<j>}$ is the element at index j in the input buffer $buf_{lvl_{x-1}}$ and $j = off_{lvl_{x-1}} + vb_{lvl_x}$.

In the *Mapping* phase the goal is to create partial symbol vectors $\mathbf{s}_{lvl_x}^{lvl_{x-1}-1<j>}$ based on the $vt_{lvl_x} \in VT_{lvl_x}^{k}$ virtual thread identifiers. In order to achieve this, each vt_{lvl_x} is transformed to a binary vector of length $\log_2|\Omega| \cdot (lvl_{x-1} - lvl_x)$. Let B denote the transformation of a natural number to a binary vector of size l

$$B : (\mathbb{N}, l \in \mathbb{N}) \rightarrow \mathbb{B}^l = \{0, 1\}^l. \tag{4.56}$$

Let the binary vector $\mathbf{b}^l$ denote the result of transformation B with inputs vt_{lvl_x} and $\log_2|\Omega| \cdot (lvl_{x-1} - lvl_x)$:

$$\mathbf{b}^l = B(vt_{lvl_x}, \log_2|\Omega| \cdot (lvl_{x-1} - lvl_x)). \tag{4.57}$$

In vector $\mathbf{b}^l$, $(lvl_{x-1} - lvl_x)$ number of binary groups of size $\log_2|\Omega|$ are available. A one-to-one mapping between the binary groups and the symbol set elements is defined based on Gray mapping. Therefore, while iterating over the groups of binary elements, $\mathbf{b}_{(i\cdot\log_2|\Omega|):((i+1)\cdot\log_2|\Omega|-1)} \rightarrow s_i \in \Omega$ are selected and the partial symbol vector $\mathbf{s}_{lvl_x}^{lvl_{x-1}-1} = (s_{lvl_x}, s_{lvl_x+1}, \cdots, s_{lvl_{x-1}-1})$ is formed.

In the *Merging* phase the result of the selection and mapping is merged, namely, each selected vector $\mathbf{s}_{lvl_{x-1}}^{N<j>}$ and mapped symbol vector $\mathbf{s}_{lvl_x}^{lvl_{x-1}-1<j>}$ is merged as

$$\begin{aligned} \mathbf{s}_{lvl_x}^{N<j>} &= (\mathbf{s}_{lvl_x}^{lvl_{x-1}-1<j>}, \mathbf{s}_{lvl_{x-1}}^{N<j>}) \\ &= (s_{lvl_x}, \cdots s_{lvl_{x-1}-1}, s_{lvl_{x-1}}, \cdots, s_N). \end{aligned} \tag{4.58}$$

DOI:10.15774/PPKE.ITK.2015.010

4.8.3.3 Path metric evaluation block

In the *Path Metric Evaluation* block, the metric of created partial symbol vectors is computed. This is one of the most time-consuming steps, but the path metric is computed in parallel by several threads. Consequently, a significant speed-up can be achieved. Further speed-up can be achieved if the path metric of partial symbol vectors $M(\mathbf{s}^{N}_{lvl_{x-1}})$ computed previously are stored and only the contribution of the newly created partial symbol vectors $M(\mathbf{s}^{lvl_{x-1}-1}_{lvl_x})$ to the overall metric is computed.

4.8.3.4 Searching or sorting block

The last block of the EEP is one of the most important stages during the detection. Depending on the level of processing, either sorting or a minimum search is performed. The minimum search is applied only when the detection has reached the last processing level, while sorting is applied on all other levels. The use of the two algorithms is motivated by the lower complexity required by the minimum search algorithm. Recall, when the last level of the tree is reached, the task is to find the symbol vector with the smallest path metric.

As discussed in Sec. 4.8.2, the complexity of the algorithm can be reduced by adjusting the radius of the sphere after finding a leaf node of the tree. Sorting the buffers based on the path metric of symbol vectors and applying the hybrid searching strategy makes the finding of a leaf node possible after a few iterations. Several parallel algorithms exist in the literature that can exploit the parallel architectures in order to sort and search arrays [88], thus, the high computational power of these devices can be also exploited at this stage.

4.8.3.5 Application of the Expand and Evaluate pipeline

The EEP depicted in Fig. 4.10 implements one level of PSD algorithm. To implement the entire PSD algorithm, the EEP is used in an iterative manner as shown in Fig. 4.11. Depending on the processing level the EEP outputs are (i) the sorted partial symbol vectors placed in buf_{lvl_x} or (ii) the symbol vector with the smallest path metric. The inputs for this process are (i) the number of threads tt available for the processing and (ii) $exp_{lvl_{x-1}}$ number of previously computed partial symbol vectors retrieved from $buf_{lvl_{x-1}}$. In the last stage of the EEP a candidate ML solution might be returned.

DOI:10.15774/PPKE.ITK.2015.010

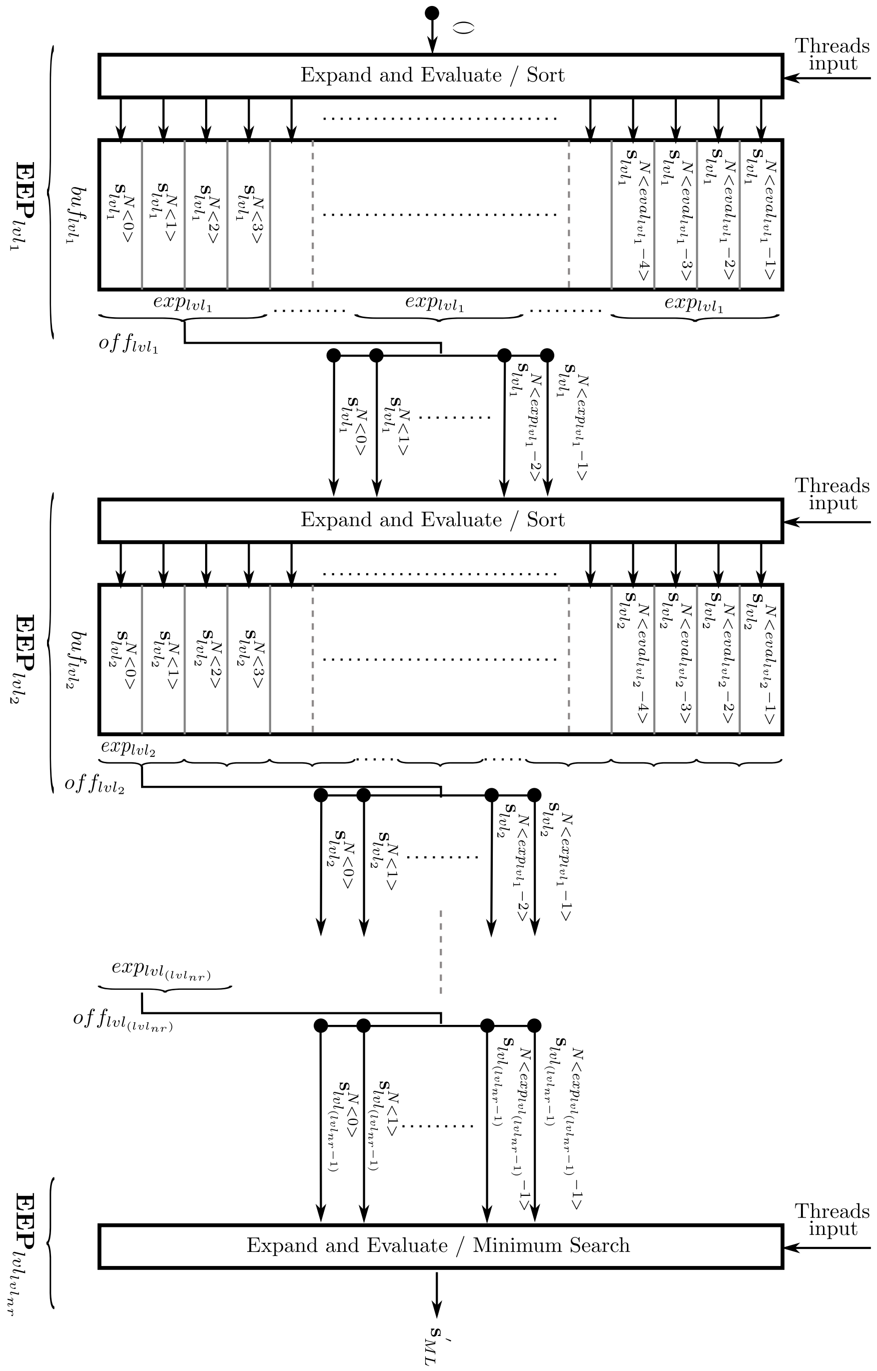


Figure 4.11: The iterative application of the Expand and Evaluate pipeline implementing the Parallel Sphere Detector algorithm.

DOI:10.15774/PPKE.ITK.2015.010

4.8.4 Levels of parallelism and CUDA mapping details

As described in Sec. 2.2.1, a grid is defined before launching a CUDA kernel. A grid may contain several TBs and each TB may contain several threads. Concurrent kernel executions are also possible for some devices using multiple streams. Hence, multiple levels of parallelism are available. The main challenge during the implementation is to well define the parallel possibilities of the system model, the parallel architecture and to make the correct bounding of these.

Algorithm level parallelism is the effective distribution of the work among the threads in a TB. The computationally intensive parts of the algorithm are the expansion and evaluation of the symbol vectors and the sorting. The *Expand and Evaluate* procedure is highly parallel. Every thread in the thread block is working at this point. The PSD algorithm through its parameters is able to adjust the generated work, thus, the algorithm can be easily adapted to different architectures.

For the sorting stage several parallel sorting algorithms can be used. In the PSD algorithm the sorting is done with the use of sorting networks [89], [88], [90]. Due to their data-independent structure, their operation sequence is completely rigid. This property makes this algorithm parallelizable for the GP-GPU architecture. The minimum search algorithm relies on the parallel scan algorithm [91].

Each TB launched is a one dimensional block with tt number of threads. In order to get fast detection, access time to global memory has to be minimized. A good solution is to store the heavily used buf_{lvl_x} arrays in the shared memory. If all buf_{lvl_x} buffers are stored in the shared memory, then a more severe limitation may be imposed on the parameters lvl_x and exp_{lvl_x}. This is because the size of the shared memory is significantly smaller than that of the global memory. The shared memory used by a TB is proportional to the sum $\sum_{x=1}^{lvl_{nr}} eval_{lvl_x}$ of the evaluated nodes at different levels. The excessive use of shared memory can lead to occupancy degradation, consequently, one SM can execute only a lower number of TBs at the same time. In case of GP-GPUs a good trade-off has to be found among the algorithm parameters and the resources of the SMs. Since different GP-GPUs have different memory configurations, the optimal algorithm parameters depend on the device used.

The model, presented in Sec. 3.2, assumes block-fading channel where the fading process is constant for a block of symbols and changing independently from one block to another. The block of symbol vectors for which the fading process is constant is called a *fading block*. A transmitted frame of length L symbols is affected by F independent

DOI:10.15774/PPKE.ITK.2015.010

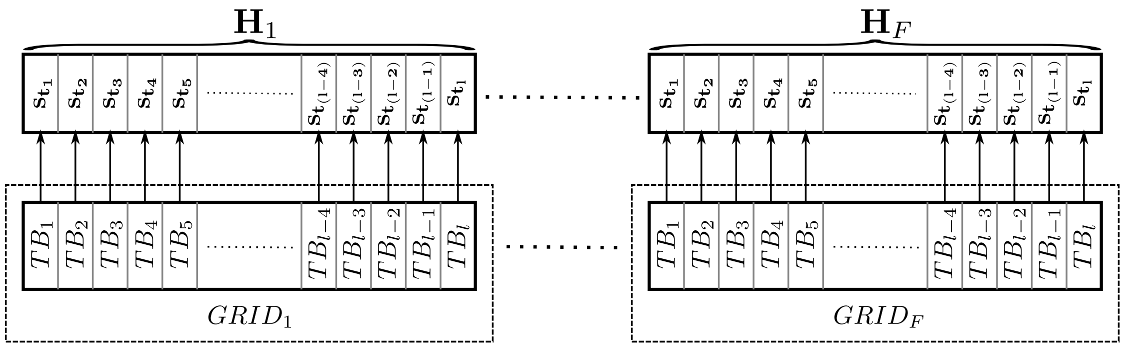

$\mathbf{H}_1$
$\mathbf{H}_F$
$\mathbf{St}_1$ $\mathbf{St}_2$ $\mathbf{St}_3$ $\mathbf{St}_4$ $\mathbf{St}_5$ $\mathbf{St}_{(l-4)}$ $\mathbf{St}_{(l-3)}$ $\mathbf{St}_{(l-2)}$ $\mathbf{St}_{(l-1)}$ $\mathbf{St}_l$
TB_1 TB_2 TB_3 TB_4 TB_5 TB_{l-4} TB_{l-3} TB_{l-2} TB_{l-1} TB_l
$GRID_1$
$GRID_F$

Figure 4.12: Equally distributed computing load with the direct biding of the thread blocks and symbol vectors.

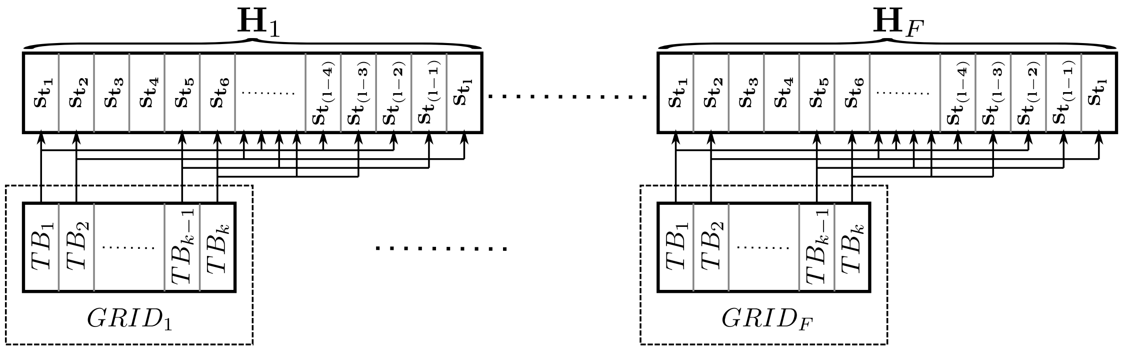

$\mathbf{H}_1$
$\mathbf{H}_F$
$\mathbf{St}_1$ $\mathbf{St}_2$ $\mathbf{St}_3$ $\mathbf{St}_4$ $\mathbf{St}_5$ $\mathbf{St}_6$ $\mathbf{St}_{(l-4)}$ $\mathbf{St}_{(l-3)}$ $\mathbf{St}_{(l-2)}$ $\mathbf{St}_{(l-1)}$ $\mathbf{St}_l$
TB_1 TB_2 TB_{k-1} TB_k
$GRID_1$
$GRID_F$

Figure 4.13: Dynamically distributed computing load with the dynamic biding of the thread blocks and symbol vectors.

channel realizations, resulting in a block of length $l = \lceil L/F \rceil$ symbols being affected by the same channel realization. It can be seen that multiple symbol vectors have to be processed simultaneously for one received frame.

The *system level* parallelism is implemented by the parallel processing of fading blocks of a received frame. Consequently, the number of kernels launched is equal to the number of independent channel realizations. Every grid assigned to a kernel launches several TBs and the PSD algorithm is executed by the threads of every TB. The configuration of the grids, namely the binding of the TBs and symbol vectors, is critical since this influences the concurrent execution of the kernels. For further details, refer to the discussion of the device level parallelism below.

Different binding strategies among the TBs of one grid and symbol vectors of a fading block are shown in Figs. 4.12 and 4.13. In the first case the number of TBs in one grid is equal to the number of symbol vectors belonging to the same channel matrix. The drawback of this straightforward binding is the high number of TBs because the resources of the GP-GPU will be available for a long time duration only for one kernel. Consequently, the overlapping execution of concurrent kernels is limited. In the second case the number of TBs in a grid is significantly smaller than the number of symbol

DOI:10.15774/PPKE.ITK.2015.010

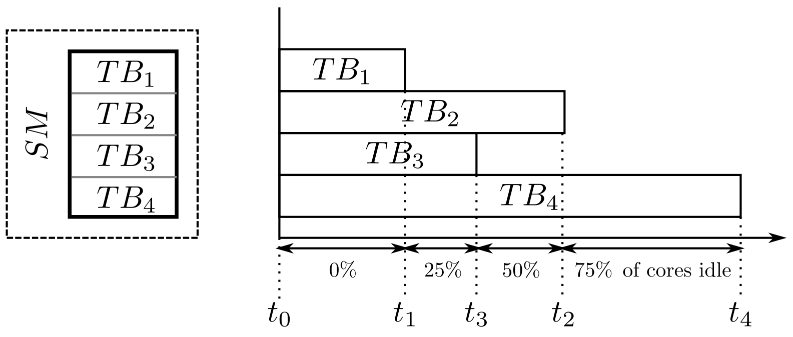


Figure 4.14: A simplified thread block scheduling model on a streaming multiprocessor.

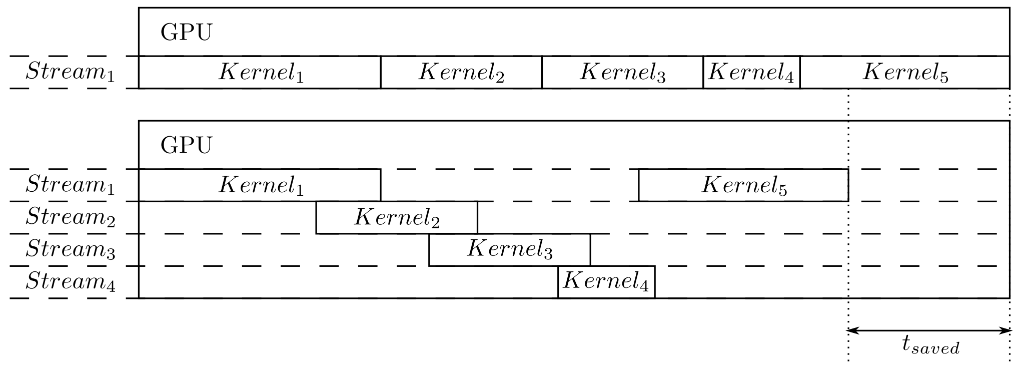


Figure 4.15: The scheduling of kernels using the single stream and multiple stream execution models.

vectors in one group. The work for a TB is dynamically distributed, namely, when the detection of one symbol vector is finished, the PSD algorithm executed by the threads of the TB evaluates the next unprocessed symbol vector. As the detection time of different symbol vectors may differ significantly, the number of symbol vectors to be processed by one TB is also different. Having a lower number of TBs in one grid makes the execution of TBs from other grids possible if there are free GP-GPU resources. The drawback of this approach is the increased complexity of the algorithm caused by the dynamic distribution of the work among the TBs.

The *device level* parallelism in GP-GPUs is achieved by launching multiple kernels simultaneously on different streams. By exploiting the advantage of device level parallelism, a significant decrease in the computational time can be achieved. To demonstrate the importance of overlapping execution of multiple kernels, a simplified TB scheduling is shown in the following. Consider a GP-GPU with only one SM and assume that it is capable of running only four TBs simultaneously as shown in Fig. 4.14. Consider a kernel with a grid configuration of four TBs. The kernel is finished when every TB has completed its task. In this example, the execution of TB_1 is finished at time t_1, upon which 25% of the cores are idle. The worst case is when the execution of TB_2 is finished because

DOI:10.15774/PPKE.ITK.2015.010

Table 4.4: Main characteristics of the GK104 Kepler architecture.

CUDA cores	Threads / Warp	Max warps / SMX	Max threads / SMX	Max TBs / SMX	Max registers / thread	Max threads / TB	Max shared memory / SMX
1536	32	64	2048	16	63	1024	48 Kbytes

75% of the available cores in the SM are idle. Because of the wasted resources, the overall performance is degraded. If a new TB from a different kernel could be launched after the execution of TB_1 is finished, the resources of the GP-GPU would be fully exploited.

The idle time of the cores can be minimized by exploiting the multi-stream features of the selected GP-GPUs. Figure 4.15 shows the scheduling for *single stream* and *multiple streams* execution. The single stream strategy launches the kernels in succession and avoids overlapping execution. As shown in Fig. 4.15, multiple stream exploits the overlapping execution of kernels and minimizes the idle time of the cores. Note, the amount of overlap depends on the occupancy of the kernels and the number of TBs launched in each kernel. In Sec. 4.8.5 the performance of single and multiple-stream strategies are compared and evaluated.

4.8.5 Performance evaluation of the Parallel Sphere Detector algorithm

One of the most important performance metrics of a detector is the BER achieved. The PSD algorithm implements the ML detector and the BER performance of the ML detector was compared with linear, SIC, and tree-search based methods in Figs. 4.2, 4.3, 4.7 and 4.8. Thus, in the following sections the focus is on the detection throughput, scalability and the complexity of the PSD algorithm.

A major challenge in ML detection is handling its varying complexity. Channel matrices with high condition numbers or low SNRs may increase the complexity of the algorithm. Consequently, the running time of different kernels may differ significantly. In order to evaluate the average detection throughput of the PSD algorithm $F = 8000$ independent channel realizations with $l = 1200$ symbol vectors for each channel realization were generated and evaluated. The average throughput is determined based on 9.6×10^6 processed symbol vectors. The performance of the PSD algorithm was evaluated on a GeForce GTX690 GP-GPU built on the Kepler GK104 [92] architecture and compiled with CUDA 5.5. The main parameters of the GK104 architecture are given in Table 4.4.

DOI:10.15774/PPKE.ITK.2015.010

Table 4.5: Parallel Sphere Detector algorithm parameter configurations achieving highest detection throughput with multiple stream kernel executions for 2×2 and 4×4 MIMO systems at 20 dB SNR.

n	m	$\|\Omega\|$	lvl_{nr}	lvl_0	lvl_1	lvl_2	lvl_3	lvl_4	exp_{lvl_0}	exp_{lvl_1}	exp_{lvl_2}	exp_{lvl_3}	tt	$Mbit/s$
2	2	2	1	5	1	0	0	0	1	0	0	0	16	290
2	2	4	1	5	1	0	0	0	1	0	0	0	128	350
2	2	8	2	5	3	1	0	0	1	1	0	0	32	191
4	4	2	2	9	7	1	0	0	1	1	0	0	64	225
4	4	4	3	9	6	4	1	0	1	4	1	0	64	161
4	4	8	4	9	7	5	3	1	1	4	4	4	128	32

4.8.5.1 Average detection throughput and scalability

Another important metric of a MIMO detector is the average detection throughput. The achieved detection throughput by the PSD algorithm is influenced by many factors such as the antenna configuration, modulation order, the realized signal to noise ratio, the parallel architecture, and the algorithm's parameter configuration. In Sec. 4.8 the memory requirements of the PSD algorithm were presented. It was shown that the chosen parameters clearly define the memory requirements. During the parameter optimization for a GP-GPU architecture it is worth choosing the parameters such that the size of the symbol vector buffers do not exceed the shared memory available.

One of the most important quality measures of a communication link is its SNR. The realized SNR at the receiver highly influences the optimal parameter configuration. The following concept lies behind the relationship of the realized SNR and the parameter configuration used: at low SNRs it is more likely that several iterations will be performed by the algorithm on the selected levels lvl_x. The overall algorithm complexity is composed by the expansion and evaluation of the symbol vectors and the path metric based sorting. Since the sorting stage has to be performed more frequently at low SNRs, a better strategy is to perform the sorting with more symbol vectors. This can be carried out by increasing either the value of exp_{lvl_x} or the difference of the adjacent level $lvl_x - lvl_{x+1}$ parameters. With the above modifications more symbol vectors are expanded and evaluated, thus, the probability of finding the ML solution in the first iterations is increased.

At higher SNRs, even with lower exp_{lvl_x} parameters, there is a relatively high probability that the first leaf node found by the PSD algorithm is the ML solution. Consequently, the optimal sphere radius is found, thus, the other symbol vectors can be discarded. The reduced number of iterations results in a reduced number of sortings with fewer elements to sort. This is why the throughput is higher when the SNR is high. Usually, 20 dB SNR is advisable for a reliable communication for $|\Omega| \leq 8$. Consequently,

DOI:10.15774/PPKE.ITK.2015.010

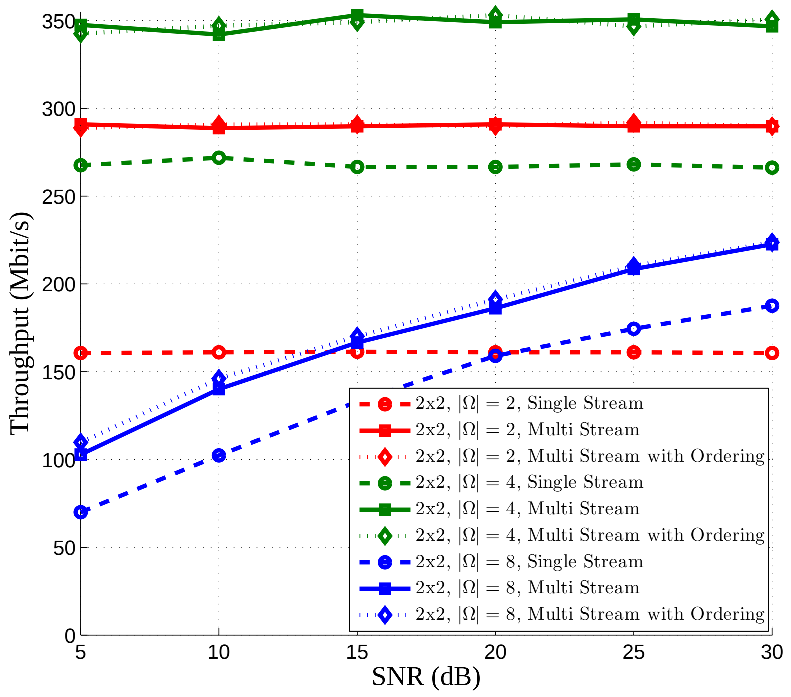


Figure 4.16: The Parallel Sphere Detector average detection throughput for 2×2 MIMO obtained with single stream and multiple stream kernel executions.

the PSD algorithm parameters were optimized for 20 dB SNR.

Because the parameter optimization process depends on many factors, it is performed offline and only once. Table 4.5 summarizes the result of the parameter optimization which were used for the throughput evaluation here for the different MIMO systems.

Average detection throughput was evaluated for 2×2 and 4×4 MIMO systems with symbol sets $|\Omega| = 2, 4$ and 8. In order to measure the effects of device level parallelism

- single stream execution with direct TB binding shown in Fig. 4.12 and
- multiple stream execution with dynamic TB binding depicted in Fig. 4.13

were compared. In the case of direct binding, the number of TBs in a grid was equal to the number of symbol vectors associated to one channel matrix, i.e., TB $= l = 1200$. In the case of dynamic TB binding load distribution grids with 128 TBs were launched on 32 streams.

Figure 4.16 shows that the average detection throughputs do not depend on SNR for 2×2 MIMO systems with $|\Omega| = 2$ and 4. This is due to the low number of symbol vectors to be evaluated on the last tree level. The low number of symbol vectors can be processed simultaneously without computing any partial symbol vector. The throughput is higher for $|\Omega| = 4$ because the number of transmitted bits is doubled compared to the case of

DOI:10.15774/PPKE.ITK.2015.010

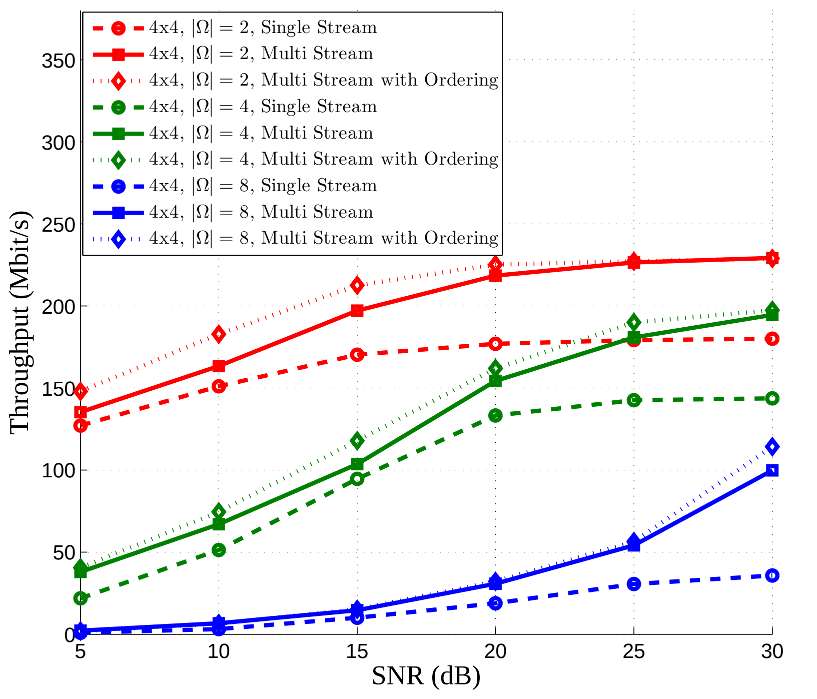


Figure 4.17: The Parallel Sphere Detector average detection throughput for 4×4 MIMO obtained with single stream and multiple stream kernel executions.

$|\Omega| = 2$, but the processing time required is not significantly higher. Multiple stream execution results in an increase of 45% and 25% in the average detection throughput for $|\Omega| = 2$ and $|\Omega| = 4$, respectively. By further increasing the number of antennas or the symbol set size the total number of nodes are exponentially increasing, thus, the detection throughput decreases. The effect of overlapping execution of kernels on multiple streams is shown in Fig. 4.17 for a 4×4 MIMO system with symbol sets $|\Omega| = 2$, 4 and 8. The average detection throughput is increased by $15\% - 30\%$ for $|\Omega| = 2$, 4 and $38\% - 64\%$ for $|\Omega| = 8$.

Furthermore, the average detection throughput achieved with (i) the PSD algorithm implemented on the GTX690 GP-GPU and (ii) the sequential SD executed simultaneously on every thread of an Intel Xeon CPU E5-2650 v3 was compared. The E5-2650 CPU has 10 cores and it is able to run 20 threads. In order to fully exploit the CPU the sequential SD algorithm was simultaneously executed on every single thread of the CPU. Figure 4.18 shows the results of the average detection throughput comparison. In case of smaller search space, i.e., 4×4 MIMO and $|\Omega| = 2$, the speed-up achieved by the GP-GPU is about a factor of two. As the size of the symbol set is increasing the GP-GPU speed-up is also higher. For a 4×4 MIMO system and $|\Omega| = 4$ the throughput

DOI:10.15774/PPKE.ITK.2015.010

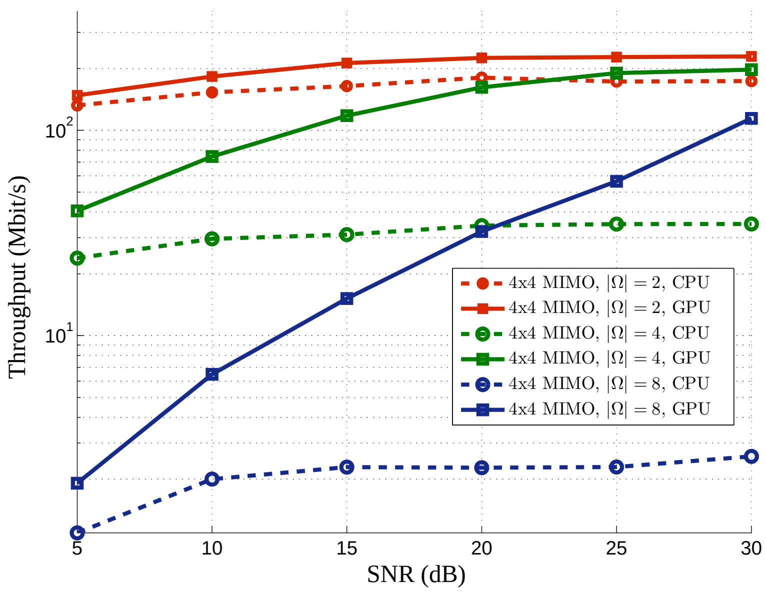


Figure 4.18: The comparison of the average detection throughput of (i) the Parallel Sphere Detector algorithm implemented on a GP-GPU architecture and (ii) the sequential Sphere Detector executed on every thread of a multi-core CPU.

is increased $2-6$ times, and for $|\Omega| = 8$ the throughput is increased $2-50$ times by the GP-GPU.

The scalability of the PSD algorithm on the GTX690 GP-GPU is presented in Fig. 4.19. For the different number of threads a different parameter configuration has to be found in order to achieve maximal throughput. Table 4.6 shows the PSD algorithm parameter configurations achieving highest detection throughput for different number of CUDA threads for 4×4 MIMO systems at 20 dB SNR. The kernels were launched on a single stream and direct TB binding was used. For 4×4 MIMO systems with symbol sets $|\Omega| = 2, 4$ the highest throughput is achieved with a TB configuration of 64 threads and for $|\Omega| = 8$ the highest throughput is achieved with a TB configuration of 128 threads. By further increasing the thread numbers the occupancy of the kernels is degraded resulting in a lower detection throughput.

4.8.5.2 Preprocessing of the channel matrix

Different channel matrix preprocessing algorithms have been proposed in [64], [93] and [56] to improve the performance of MIMO systems. A common preprocessing approach is to adapt the detection order of the spatial streams to the instantaneous channel realization.

In symbol cancellation, the interference generated by the detected components of

DOI:10.15774/PPKE.ITK.2015.010

Table 4.6: Parallel Sphere Detector algorithm parameter configurations achieving highest detection throughput for different number of CUDA threads and single stream kernel execution for 4×4 MIMO systems at 20 dB SNR.

n	m	$\|\Omega\|$	tt	lvl_{nr}	lvl_0	lvl_1	lvl_2	lvl_3	lvl_4	exp_{lvl_0}	exp_{lvl_1}	exp_{lvl_2}	exp_{lvl_3}	$Mbit/s$
4	4	2	8	2	9	5	1	0	0	1	1	0	0	120
4	4	2	16	2	9	5	1	0	0	1	1	0	0	143
4	4	2	32	2	9	6	1	0	0	1	1	0	0	161
4	4	2	64	2	9	7	1	0	0	1	1	0	0	169
4	4	2	128	1	9	1	0	0	0	1	0	0	0	158
4	4	4	32	3	9	6	4	1	0	1	4	1	0	118
4	4	4	64	3	9	6	4	1	0	1	4	1	0	121
4	4	4	128	2	9	5	1	0	0	1	1	0	0	68
4	4	4	256	2	9	5	1	0	0	1	1	0	0	40
4	4	8	32	4	9	7	5	3	1	1	1	1	1	11
4	4	8	64	4	9	7	5	3	1	1	1	1	1	13
4	4	8	128	4	9	7	5	3	1	1	4	4	4	24
4	4	8	256	4	9	7	5	3	1	1	8	8	4	15

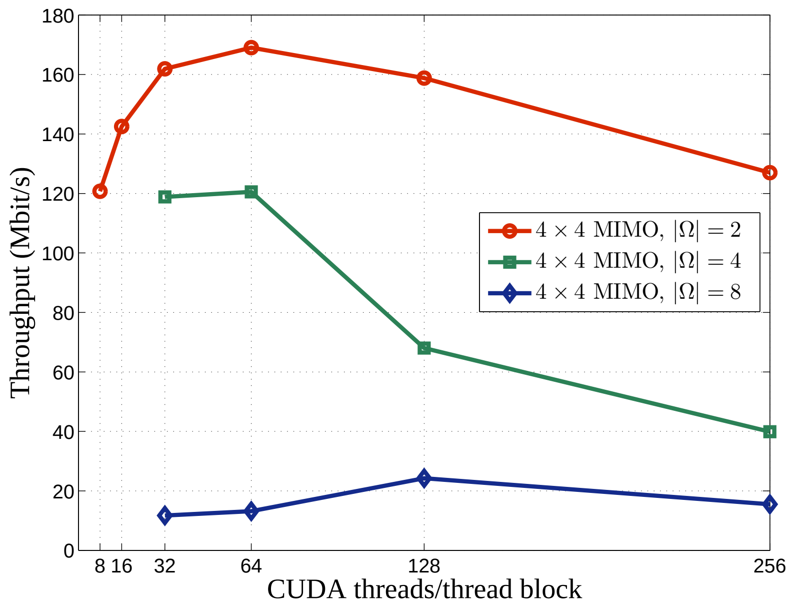


Figure 4.19: The Parallel Sphere Detector average detection throughput for 4×4 MIMO obtained at 20 dB SNR with single stream kernel execution and multiple thread block configurations.

DOI:10.15774/PPKE.ITK.2015.010

$\mathbf{s}_t$ is subtracted from the received symbol vector $\mathbf{y}$, in order to reduce the number of interferers. The best result is achieved if symbols with high post-detection SNR are considered first as discussed in [64]. The FSD introduced in [56] applies an ordering based on the norms of row vectors of the inverse channel matrix.

Channel matrix preprocessing can be used to improve the throughput of the PSD algorithm. The effect of matrix preprocessing based on decreasing ordering of the norms of the row vectors of the inverse channel matrix was evaluated. By applying channel preprocessing an extra increase of $5-10\%$ in average throughput was achieved, as shown in Figs. 4.16 and 4.17.

4.8.5.3 Average complexity per thread

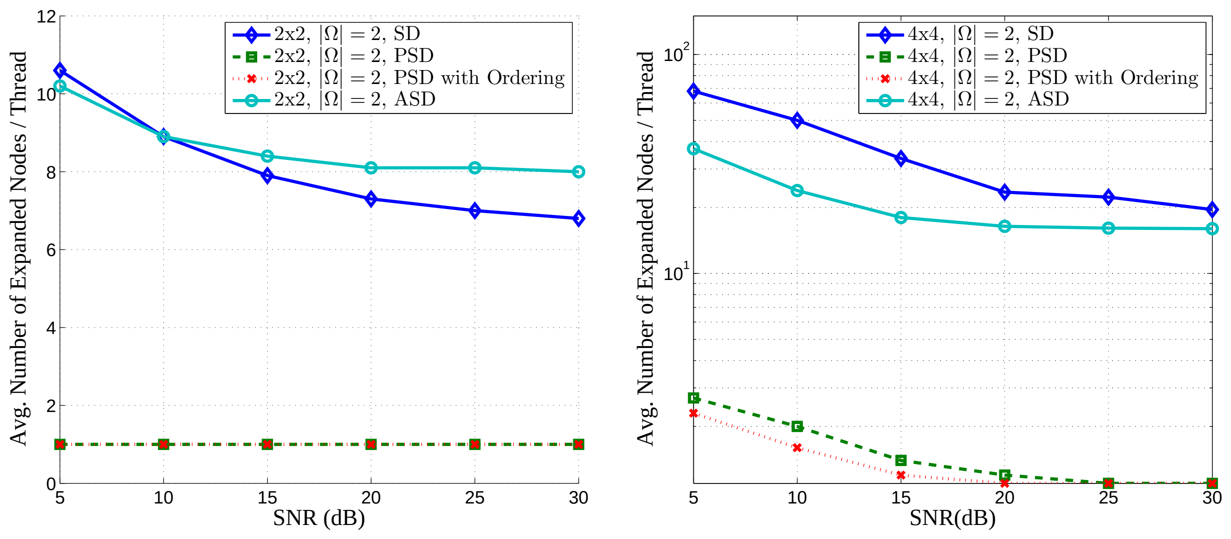


Figure 4.20: Comparison of the average number of expanded nodes per thread for (a) 2×2, (b) 4×4 MIMO and $|\Omega| = 2$ for the sequential, parallel and automatic Sphere Detector algorithms.

Another important metric of SD algorithms is the average number of expanded nodes. Figures 4.20, 4.21 and 4.22 compare the average number of visited nodes in each thread for the SD, PSD and ASD algorithms. The achieved results are compared with that of the ASD algorithm because it has been shown in [77] that ASD expands just the minimum number of nodes as the number of antennas or size of symbol set are increased. Recall, the SD and ASD algorithms are sequential algorithms. Consequently, the tree search can be performed by only a single thread and there is no possibility to expand and evaluate multiple nodes simultaneously. In contrast, the PSD algorithm is able to distribute the work among multiple threads, but the total number of symbol vectors to be expanded

DOI:10.15774/PPKE.ITK.2015.010

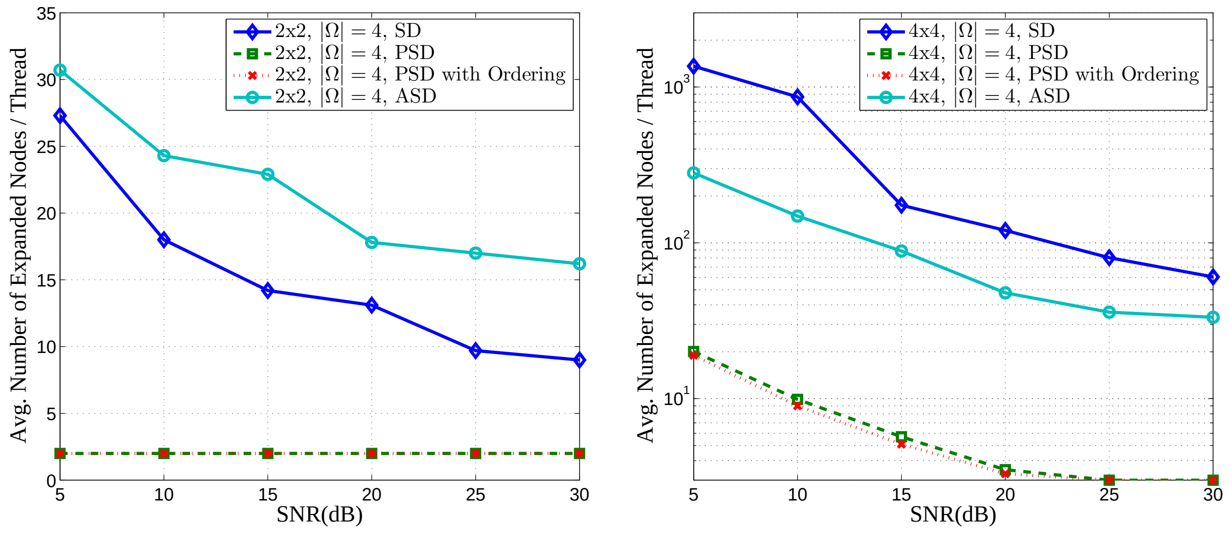


Figure 4.21: Comparison of the average number of expanded nodes per thread for (a) 2×2, (b) 4×4 MIMO and $|\Omega| = 4$ for the sequential, parallel and automatic Sphere Detector algorithms.

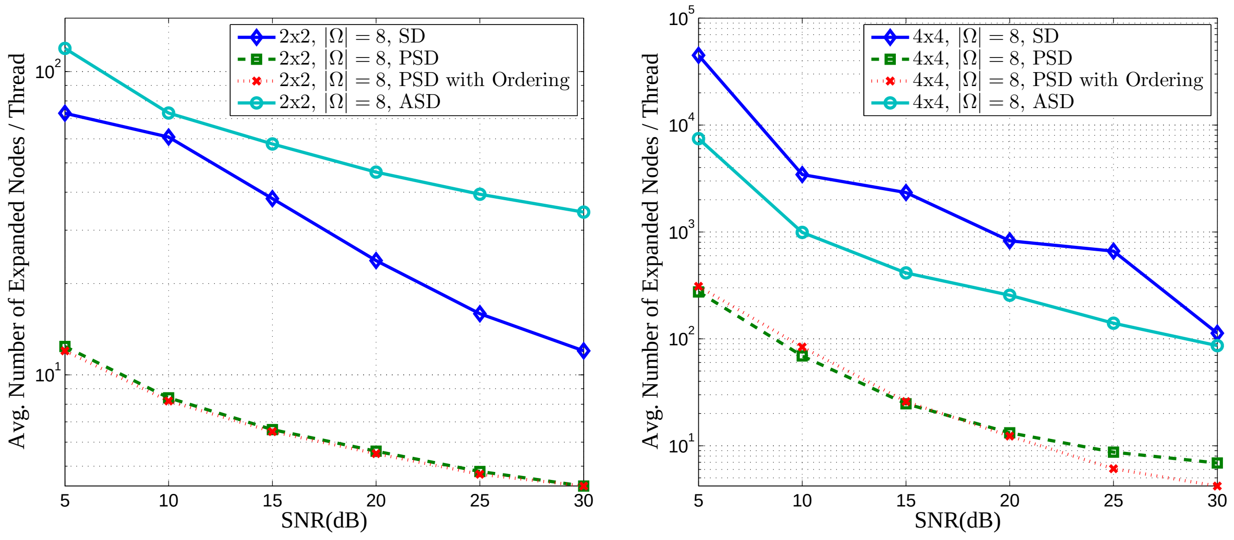


Figure 4.22: Comparison of the average number of expanded nodes per thread for (a) 2×2, (b) 4×4 MIMO and $|\Omega| = 8$ for the sequential, parallel and automatic Sphere Detector algorithms.

and evaluated becomes higher. Table 4.5 shows the number tt of total threads used for different MIMO systems. It is very interesting to compare the achieved results with the theoretical average complexity results discussed in Sec. 4.6.1.3. It can be seen that even the total number of nodes expanded by all of the threads is less than the average complexity for several ε values.

Figures 4.20, 4.21 and 4.22 show that the PSD algorithm requires a significantly lower average number of symbol vectors to be processed by one thread in every MIMO

DOI:10.15774/PPKE.ITK.2015.010

configuration. The signal space of the real-equivalent 4×4 MIMO with symbol set size of $|\tilde{\Omega}| = 64$ symbols has 1.6×10^7 symbol vectors. For an SNR of 5 dB the PSD expands into about 310 nodes per thread while the ASD expands into about 7500 nodes per thread. Consequently, the total workload of a thread running the PSD algorithm is reduced by 96%. For an SNR of 20 dB, the workload of a thread running the PSD algorithm is reduced by 95%. The distribution of work makes the PSD algorithm very efficient despite the fact that the total number of nodes to be processed is higher than that of the SD and ASD algorithms. The increase in the number of nodes is the price to be paid for using a many-core architecture.

4.8.5.4 Comparison of detection throughput and bit error rate performance

As mentioned in the earlier sections a significant trade-off exists between the BER achieved and the computational complexity of the detection algorithm. Many non-optimal detection schemes involve the use of error control coding. Thus, the resulting BER usually is better than in uncoded MIMO systems. This makes a BER performance comparison of uncoded optimal and coded non-optimal detection algorithms difficult.

Guo et al. in [78] presented the BER performance of coded and uncoded hard and soft-output MIMO systems. BER simulations for hard-output 4×4 MIMO system detectors with $|\Omega| = 4$ were presented using a four state rate 1/2 convolutional code and a four state rate 1/2 turbo code. It was shown that the uncoded MIMO system was outperformed by 5 dB at BER $= 10^{-5}$ by the convolutionally coded MIMO system and by 11 dB at BER $= 10^{-5}$ by the turbo coded MIMO system. Turbo coded max-log a posteriori probability (APP) soft-output MIMO detection improves the performance by an additional 2 dB compared to turbo-coded hard-output MIMO detection. Soft-output detection and error control coding schemes are outside scope of the thesis. In [94], [78], [95], [20], [87] further details are available on coded soft-output MIMO detection methods. In summary, the use of error control coding significantly improves the BER which can be further improved by ~ 2 dB with the higher complexity, optimal soft-output detection.

In order to make a fair comparison of the PSD algorithm with its alternatives published in the literature, three groups are compared in Table 4.7.

Performance of ML detection algorithms The first group of implementations compares the performance of hard-output true-ML solutions published in [24], [96], [25], [26] with that of the PSD algorithm. During the development of the PSD algorithm the main objective was to design a parallel algorithm that can exploit the resources of massively

DOI:10.15774/PPKE.ITK.2015.010

Table 4.7: Throughput comparison of existing MIMO detector algorithms.

Reference	BER perf.	Detection output type	Antenna config.	Symbol set size	Throughput [Mbit/s] (SNR [dB])	Technology	Detector algorithm type
[24]	ML	hard	4×4	$\|\Omega\| = 4$	38.4	ASIC	sequential SD
[96]	ML	hard	4×4	$\|\Omega\| = 2$	50	ASIC	exhaustive ML
[25]	ML	hard	4×4	$\|\Omega\| = 4$	73.5 - 145 (20-30 dB)	ASIC 250 nm	sequential SD
[26]	ML	hard	4×4	$\|\Omega\| = 4$	81.5	XC2VP30 FPGA	multi-level parallel SD
This work	ML	hard	2×2	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	290 351 191-223 (20-30 dB)	Geforce GTX 690	multi-level parallel SD (PSD)
	ML	hard	4×4	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	225-228 162-197 32-114 (20-30 dB)	Geforce GTX 690	multi-level parallel SD (PSD)
[22]	near-ML	hard	4×4	$\|\Omega\| = 8$	3.05	GeForce 210	parallel FSD
[97]	near-ML	hard	4×4	$\|\Omega\| = 8$	9.6	GeForce GTX 285	sequential FSDs on several threads
[23]	near max-log APP	soft	2×2	$\|\Omega\| = 4$	16.86	Quadro FX 1700	layered orthogonal lattice detector
	approx. max-log APP	soft	2×2	$\|\Omega\| = 4$	36	Quadro FX 1700	selective spanning fast enumeration
[20]	near max-log APP	soft	4×4	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	211.99 92.31 16.6	Tesla C2070	parallel FSD
[21]	approx. max-log APP	soft	2×2	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	663 269 43	Tesla C1060	multi-pass trellis traversal
	approx. max-log APP	soft	4×4	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	284 120 12	Tesla C1060	multi-pass trellis traversal
[98]	non-ML	hard	4×4	$\|\Omega\| = 8$	37-125	TMS320C6416 DSP	selective spanning fast enumeration
[81]	non-ML	hard	4×4	$\|\Omega\| = 8$	100-732 K=64-5	XC2VP30 FPGA	K-best SD
[87]	approx. max-log APP	soft	4×4	$\|\Omega\| = 2$ $\|\Omega\| = 4$ $\|\Omega\| = 8$	400 200 75	Xilinx XC4VLX160 FPGA	parallel FSD
[86]	near-ML	hard	4×4	$\|\Omega\| = 4$	800	EP2S60F672C3 FPGA	parallel FSD
[99]	non-ML	hard	1x1 to 4×4	$\|\Omega\| = 2$ to $\|\Omega\| = 8$	672-2143	VLSI 130 nm CMOS	multiple SD based cores

DOI:10.15774/PPKE.ITK.2015.010

parallel architectures while implementing hard-output true-ML detection.

In the first group, only the algorithm presented in [26] exploits two levels of parallelism such as: (i) a system level parallelism that implements the concurrent execution of the preprocessing, decoding and the simultaneous detection of symbol vectors, and (ii) a data dependency based low-level parallel structure. In the first group, the PSD algorithm proposed here outperforms all the other parallel and sequential algorithms.

Performance of non-optimal detection algorithms mapped on GP-GPUs In the second group the performance of non-optimal algorithms implemented on GP-GPUs are compared. The approximations used in these algorithms offer significant improvements in detection throughput because the average number of nodes visited during detection is considerably reduced. Note that these algorithms do not implement optimal detection. Consequently, they cannot achieve the theoretically attainable BER performance.

The FSD algorithm overcomes the two main drawbacks of the SD approach: (i) independently of the noise level and the channel condition, the search is performed over only a fixed number of symbol vectors and (ii) it follows predetermined paths down the tree. Consequently, all the paths can be searched in parallel. Furthermore, the BER achieved by the FSD algorithm, depending on the MIMO system size and spatial correlation, differs from the optimal by 0.5 - 1.5 dB.

GP-GPU implementations of the FSD algorithm are given in [22], [97], [20]. In [97] parallelism was achieved by launching several sequential FSDs simultaneously. However, the low detection throughput shows that the sequential algorithm does not benefit from the highly parallel architecture. In [20] a soft-output fully-parallel FSD (FP-FSD) was presented. The BER performance achieved is approximately 1 dB away from the max-log APP reference. The achieved throughput of these implementations is lower than that of the GP-GPU mapping of the PSD algorithm.

In [23] a GPU mapping was proposed for the Selective Spanning Fast Enumeration (SSFE) detector and the Layered Orthogonal Lattice Detector (LORD), but the throughput offered by them was significantly lower compared to the PSD algorithm. By applying rate 1/2 turbo decoding to a 2×2 MIMO system with $|\Omega| = 4$, the achieved Packet Error Rate (PER) of the LORD algorithm was near max-log APP. However, for a PER $=$ 5×10^{-3} the SSFE was 2.5 dB away from the reference max-log APP algorithm.

In [21] a multi-pass trellis traversal detector was proposed. Parallelism is achieved because the edge reductions and path extensions can be done simultaneously for every

DOI:10.15774/PPKE.ITK.2015.010

vertex at each stage. It achieves higher throughputs for 2×2 and 4×4 MIMO systems with symbol set $|\Omega| = 2$. However, in the other cases the PSD outperforms it. The soft-output of the detector is fed to a rate 1/2 low-density parity-check (LDPC) decoder. The achieved BER is 1-1.5 dB away from optimal detection.

Performance of non-optimal detection algorithms mapped on FPGA, VLSI and DSP platforms The performance of non-ML algorithms implemented on FPGA, digital signal processor (DSP) and application-specific integrated circuit (ASIC) architectures are surveyed in group three. In [98], the SSFE algorithm is mapped to a digital signal DSP architecture. The mapping presented is highly parallel. The highest throughput of 125 Mbit/s is achieved with a parameter configuration that is 6 dB away from the optimal ML performance at BER $= 10^{-5}$ and 37 Mbit/s is achieved at 28.5 dB SNR for a BER $= 10^{-5}$ that is 1.5 dB away from the optimal ML performance. The PSD algorithm is three times better when compared with the SSFE mapping achieving the best BER performance.

In [81] an FPGA implementation of an enhanced K-best SD algorithm is given. The achieved throughput and BER performance depends highly on the chosen K parameter. For parameter $K = 64$, the achieved throughput was 100 Mbit/s. However, the achieved BER was 6 dB away from optimal ML detection. For lower K values, the BER performance was significantly degraded. This implementation achieved similar speeds to the PSD algorithm. However, the BER was significantly degraded.

Recently, an FPGA implementation of a parallel soft-output FSD algorithm was presented in [87]. Rate 1/2 turbo decoding was applied and the achieved BER performance was similar to the K-best detector with $K = 16$. It provides similar throughput to the PSD for $|\Omega| = 4$. However, PSD is better for $|\Omega| = 8$ for higher SNR values.

The best implementations were published in [99]. Significant speed-ups were achieved by executing several SD cores in parallel. Each SD core provides a very efficient implementation of the sequential SD algorithm. The variable throughput of the algorithm was fixed by introducing run-time constraints. Consequently, this leads to a constraint on the maximum number of nodes that the SD is allowed to visit, which will clearly prevent the detector from achieving ML performance.

The comparison presented in Table 4.7 shows that the mapping of the PSD algorithm on the GP-GPU achieves the best throughput among the true-ML algorithms and outperforms many of the non-optimal GP-GPU implementations. The detection throughput of the GP-GPU mapping is similar to the non-optimal implementations presented

DOI:10.15774/PPKE.ITK.2015.010

in [98] and [87] but is outperformed by the FPGA implementation presented in [86] and the VLSI implementations published in [99]. However, those solutions implement non-optimal detection.

4.9 Conclusion

This chapter aimed to present several detector methods for MIMO systems using spatial multiplexing. Through the presentation of these methods several mathematical and algorithmical aspects of MIMO detectors were presented and their advantages and drawbacks were discussed. The main result of this chapter was to show how it is possible to enable the efficient usage of multi-core and many-core architectures in wireless MIMO communications systems by solving the hard-output true-ML detection problem. As the complexity of ML detection grows exponentially with both the size of the signal set and the number of antennas, modern MPAs were used to solve this problem. The main drawback of the original SD algorithm is its sequential nature. Thus, running it on MPAs is very inefficient. In order to overcome the limitation of the SD algorithm, the parallel SD algorithm was designed and implemented by exploiting the knowledge present in the literature.

The PSD algorithm is based on a novel hybrid tree traversal where algorithm parallelism is achieved by the efficient combination of DFS and BFS strategies, referred to as hybrid tree search, combined with path metric based parallel sorting at the intermediate stages. The most important feature of the new PSD algorithm is that it assures a good balance between the total number of processed symbol vectors and the extent of parallelism by adjusting its parameters. In modern MPAs complex memory hierarchies are available, enabling the use of smaller but faster memories. The PSD algorithm is able to adjust its memory requirements by the algorithm parameters and the allocated memory is kept constant during the processing. The above mentioned properties of the PSD algorithm make it suitable for a wide range of parallel computing devices. In contrast, the sequential SD algorithm can not fully exploit the resources of a parallel architecture because the generated computational load is always constant.

A higher *system level parallelism* and a GP-GPU specific *device level parallelism* have been identified. System level parallelism is implemented by parallel processing of the fading blocks in each received frame. The equal and dynamic computing load distribution strategies have been designed and it has been shown that a $15 - 64\%$ boost in average detection throughput can be achieved by applying the dynamic distribution of computing

DOI:10.15774/PPKE.ITK.2015.010

load in a multi-stream environment.

Parallel building blocks have been proposed for every stage of the PSD algorithm which facilitates the mapping to different parallel architectures. Based on these building blocks, an efficient implementation on a GeForce GTX 690 GP-GPU has been elaborated.

The MIMO detectors published in the literature were classified in three groups (i) true ML detectors, (ii) GP-GPU based non-optimal detectors and (iii) DSP, ASIC or FPGA based non-optimal detectors. In the latter two solutions some approximations, restrictions are introduced in order to increase the data throughput at the expense of some BER degradation. The performance of the PSD algorithm has been compared against that of the published solutions.

In the first group the average detection throughput of ML implementations, known from the literature, with the GP-GPU mapping of the PSD algorithm were compared. The new PSD algorithm outperformed each of them. In group two, the performance of existing non-optimal GP-GPU implementations have been compared with that of the GP-GPU implementation of the PSD. The PSD outperformed almost every non-optimal GP-GPU implementation. The average detection throughput of the GP-GPU mapping was similar to that of the non-optimal FPGA, DSP and ASIC implementations. Although, the throughput performance of some FPGA and VLSI based non-optimal detectors are better, those solutions suffer from a loss in BER performance.

The average number of expanded nodes per thread was also analyzed. It was shown that the PSD algorithm is doing much less processing in one thread compared to the SD and ASD algorithms. For 4×4 MIMO systems, the work of a thread, i.e., the number of expanded nodes, has been reduced by $90 - 96\%$. Furthermore, the overall node expansion performed by all of the threads is less compared to the theoretical average complexity for several ε values. Consequently, the goal of efficient work distribution was achieved.

DOI:10.15774/PPKE.ITK.2015.010

Chapter 5

Lattice reduction and its applicability to MIMO systems

5.1 Introduction

The application of LR as a preconditioner of various signal processing algorithms plays a key role in several fields. Lattice reduction consists of finding a different basis whose vectors are more orthogonal and shorter, in the sense of Euclidean norm, than the original ones. The Minkowski or Hermite-Korkine-Zolotareff reductions are the techniques that obtain the best performance in terms of reduction, but also the ones with a higher computational cost. Both techniques require the calculation of the shortest lattice vector, which has been proved to be NP-hard (see [100] and references therein).

In order to reduce the computational complexity of LR techniques Lenstra, Lenstra and Lovász (LLL) in [101] proposed the polynomial time LLL algorithm. This algorithm can be seen as a relaxation of Hermite-Korkine-Zolotareff conditions [102] or an extension of Gauss reduction [100] and obtains the reduced basis by applying two different operations over the original basis: size-reduction (linear combination between columns) and column swap. A different structure than that of the LLL algorithm was introduced by Seysen in [103]. While the LLL algorithm concentrates on local optimizations, Seysen's reduction algorithm simultaneously produces a reduced basis and a reduced dual basis for the lattice. Although further reduction techniques have been proposed afterwards, the LLL algorithm is the most used due to the good trade-off between performance and computational complexity.

Regarding the hardware implementation of the LLL algorithm, several solutions can be found in the literature. Implementations that make use of LR to improve the detection

DOI:10.15774/PPKE.ITK.2015.010

performance of multiple antenna systems can be found in [104, 32, 34, 35]. In [104], an LR-aided symbol detector for MIMO and orthogonal frequency division multiple access is implemented using 65 nm ASIC technologies. An FPGA implementation of a variant of the LLL algorithm, Clarkson's algorithm, is presented in [32], main benefit of which is the computational complexity reduction without significant performance loss in MIMO detection. In [33], a hardware-efficient VLSI architecture of the LLL algorithm is implemented, which is used for channel equalization in MIMO communications. More recently, [34] makes use of a Xilinx XC4VLX80-12 FPGA for implementing LR-aided detectors, whereas [35] uses an efficient VLSI design based on a pipelined architecture.

Józsa et al. in [4] proposed the CR-AS-LLL algorithm. The algorithm made an efficient mapping of the algorithm for many-core massively parallel SIMD architectures possible. The mapping exploited low level fine-grained parallelism that was combined with efficient distribution of the work among the processing cores, resulting in minimized idle time of the threads launched.

Based on the parallel block-reduction concept presented in [105], a higher level, coarse-grained parallelism can be applied as an extra level of parallelism. The idea is to subdivide the original lattice basis matrix in several smaller submatrices and perform an independent LR on them followed by a boundary check between adjacent submatrices. Based on the above block-reduction concept Józsa et al. further extended the parallelism in [4] by introducing the MB-LLL algorithm. MB-LLL implements a parallel processing of the submatrices by using the parallel CR-AS-LLL algorithm for the LR of every submatrix. A performance comparison of the CR-AS-LLL and the MB-LLL algorithms on various GP-GPU architectures was presented in [5].

The implementations of the previous references make use of only one architecture to calculate the LR of a basis. A better performance can be obtained by combining different architectures, known as heterogeneous computing [106]. Among the different combinations, the use of CPU and GP-GPU is probably the most popular since they can be found in most of the computers.

The CR-MB-LLL algorithm introduced by Józsa et al. in [4] further reduces the computational complexity of the MB-LLL algorithm. The main idea behind the CR-MB-LLL algorithm is the relaxation of the first LLL condition while executing the LR for the submatrices, resulting in the delay of the Gram-Schmidt coefficients update and by using less costly procedures when performing the boundary checks. The effects of this complexity reduction are evaluated on different architectures.

DOI:10.15774/PPKE.ITK.2015.010

A mapping of the CR-MB-LLL algorithm on a heterogeneous platform consisting of a CPU and a GP-GPU is shown and it is compared with implementations running on a GP-GPU with *dynamic parallelism* (DP) capability and a multi-core CPU architecture. The proposed architecture allows the dynamic scheduling of kernels where the overhead introduced by host-device communication is hidden by the use of CUDA streams. Results show that the CR-MB-LLL algorithm executed on the heterogeneous platform outperforms the DP-based GP-GPU and multi-core CPU implementations. The mapping of algorithms on different parallel architectures is very challenging, since the number of processing cores, latency and size of different memories available and cache size significantly differ.

In this section after a short overview of LR preliminaries, a brief overview of the most important LR algorithms is given and the mapping details of the CR-AS-LLL and CR-MB-LLL algorithms to different parallel architectures is also presented with a special emphasis on the work distribution among the threads and the efficient memory utilization.

5.2 Lattice reduction preliminaries

An m-dimensional lattice is the set of all integer combinations of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ $(m \geq n)$. A compact representation of a lattice basis is to set the basis vectors as columns to a lattice basis matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{R}^{m \times n}$. The integer $n = \dim(\mathrm{span}(\mathbf{B}))$ is called the rank or dimension of the lattice. If the rank n equals the dimension m, then the lattice is called full rank or full dimensional. The real-valued lattice generated by matrix $\mathbf{B}$ is defined as

$$\mathcal{L}(\mathbf{B}) = \left\{ \mathbf{x} \middle| \mathbf{x} = \sum_{i=1}^{n} z_i \cdot \mathbf{b}_i, z_i \in \mathbb{Z}, \mathbf{b}_i \in \mathbb{R}^m \right\}, \tag{5.1}$$

Similarly $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bz} | \mathbf{z} \in \mathbb{Z}^n\}$ can be defined with the matrix-vector multiplication of the lattice basis and the integer input vectors.

In the following the most important transformations, metrics and structures are presented.

1. *Unimodular transformations.* Generally, the columns of any matrix $\tilde{\mathbf{B}}$ can form a basis for $\mathcal{L}(\mathbf{B})$ if and only if a unimodular matrix $\mathbf{T}$ exists that satisfies $\tilde{\mathbf{B}} = \mathbf{BT}$. A unimodular matrix $\mathbf{T} \in \mathbb{Z}^{n \times n}$ is a square integer matrix with $\det(\mathbf{T}) = \pm 1$. Elementary matrix column operations such as reflection, swap and translation can

DOI:10.15774/PPKE.ITK.2015.010

be performed with the help of unimodular matrices.

In case of *reflection* $\tilde{\mathbf{b}}_l = -\mathbf{b}_l$ a specific column is multiplied by -1. The unimodular matrix that carries out the reflection is defined as

$$\mathbf{T}_l = \mathbf{I}_n - 2\mathbf{e}_l\mathbf{e}_l^T, \tag{5.2}$$

where $\mathbf{e}_l$ is an n-dimensional unit vector and the unit value is on dimension l.

Swap is defined as the interchange of two column vectors. The swap of columns k and l according to $\tilde{\mathbf{b}}_l = \mathbf{b}_k$ and $\tilde{\mathbf{b}}_k = \mathbf{b}_l$ is achieved with the postmultiplication of the unimodular matrix

$$\mathbf{T}_{(k,l)} = \mathbf{I}_n - \mathbf{e}_k\mathbf{e}_k^T - \mathbf{e}_l\mathbf{e}_l^T + \mathbf{e}_k\mathbf{e}_l^T + \mathbf{e}_l\mathbf{e}_k^T. \tag{5.3}$$

During *translation* $\tilde{\mathbf{b}}_l = \mathbf{b}_l + \mathbf{b}_k$ one column is added to another column. The unimodular matrix required for this operation is defined as

$$\mathbf{T}_{(k,l)} = \mathbf{I}_n + \mathbf{e}_k\mathbf{e}_l^T. \tag{5.4}$$

In case if the translation operation has to be performed $\mu \in \mathbb{Z}$ times, such as $\tilde{\mathbf{b}}_l = \mathbf{b}_l + \mu\mathbf{b}_k$, the associated unimodular transformation is

$$\mathbf{T}^{\mu}_{(k,l)} = \mathbf{I}_n + \mu\mathbf{e}_k\mathbf{e}_l^T. \tag{5.5}$$

During lattice reduction the above mentioned operations are performed until the lattice basis achieves the requirements of the reduction algorithm.

2. *Fundamental structures.* One fundamental region defined by the lattice basis is the *parallelotope* that is defined as

$$\mathcal{P}(\mathbf{B}) = \left\{ \mathbf{x} \middle| \mathbf{x} = \sum_{i=1}^{n} \phi_i \cdot \mathbf{b}_i, 0 \le \phi_i < 1 \right\}. \tag{5.6}$$

By shifting the parallelotope $\mathcal{P}(\mathbf{B})$ to every lattice point the span of $\mathbf{B}$ is completely covered.

Another important fundamental region is the *Voronoi region.* Given a discrete set

DOI:10.15774/PPKE.ITK.2015.010

of points $\Sigma \in \mathbb{R}^m$ the Voronoi region of a point $\mathbf{y}_i \in \Sigma$ is the closed convex set

$$\mathcal{V}(\mathbf{y}_i) = \left\{ \mathbf{x} \middle| \|\mathbf{x} - \mathbf{y}_i\| \leq \|\mathbf{x} - \mathbf{y}_j\|, \text{ for all } i \neq j \right\}. \tag{5.7}$$

By considering sets of points which form lattices due to the translation symmetry of the lattice the Voronoi regions of all lattice points are congruent. Hence, the Voronoi region of the lattice $\mathcal{L}$ around the origin is defined as

$$\mathcal{V}(\mathcal{L}) = \left\{ \mathbf{x} \middle| \|\mathbf{x}\| \leq \|\mathbf{x} - \mathbf{y}\|, \text{ for all } \mathbf{y} \in \mathcal{L} \right\} \tag{5.8}$$

In contrast to the fundamental parallelotope $\mathcal{P}(\mathbf{B})$, the Voronoi region is a lattice invariant structure meaning that it is independent of the actual lattice basis. In Fig. 5.1 a square, rhombic and hexagonal lattice is shown together with their fundametal parallelotope and the Voronoi region.

3. *Lattice Determinant.* Given lattice $\mathcal{L}$ with basis matrix $\mathbf{B}$ the lattice volume or the lattice determinant is defined as

$$d(\mathcal{L}) = \sqrt{det(\mathbf{B}^T\mathbf{B})}. \tag{5.9}$$

The lattice determinant is independent of the basis. Given a unimodular matrix $\mathbf{T} \in \mathbb{R}^n$ and a lattice basis matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ it can be shown that the postmultiplication with the unimodular matrix does not change the lattice determinant

$$\begin{aligned} \det(\mathbf{B}^T\mathbf{B}) &= \det((\mathbf{BT})^T(\mathbf{BT})) \\ &= \det(\mathbf{T}^T)\det(\mathbf{B}^T\mathbf{B})\det(\mathbf{T}) \\ &= \det(\mathbf{B}^T\mathbf{B}) \end{aligned} \tag{5.10}$$

4. *Orthogonality Defect.* The orthogonality defect $\xi(\mathbf{B})$ of lattice basis $\mathbf{B}$ is defined as

$$\xi(\mathbf{B}) = \frac{1}{d(\mathcal{L})} \prod_{i=1}^{n} \|\mathbf{b}_i\| \tag{5.11}$$

The orthogonality defect measures the degree of orthogonality for a given lattice basis matrix. Given a positive-semidefinite matrix $\mathbf{P} = \mathbf{B}^T\mathbf{B}$ then Hadamard's

DOI:10.15774/PPKE.ITK.2015.010

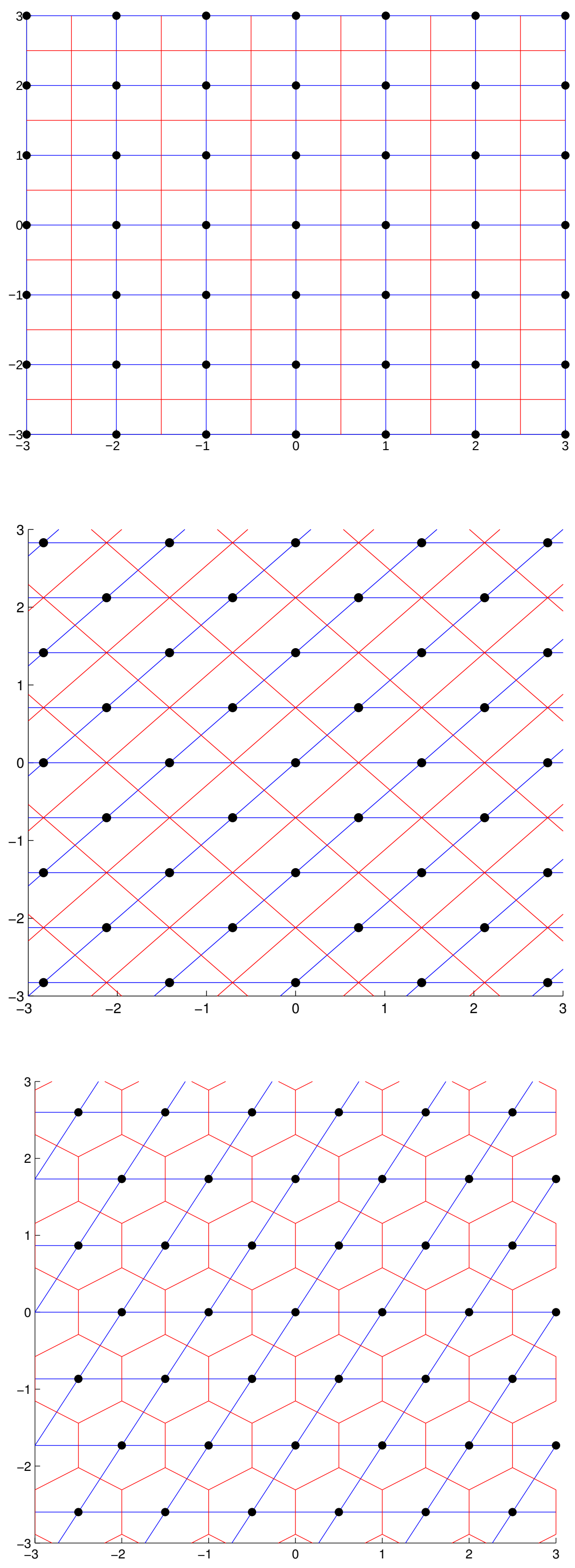


Figure 5.1: Square, rhombic and hexagonal lattices with the fundamental parallelotope structures (blue) and the Voronoi regions (red).

DOI:10.15774/PPKE.ITK.2015.010

inequality can be written as

$$\det(\mathbf{P}) = \det(\mathbf{B}^T\mathbf{B}) \leq \prod_{i=1}^{n} \|\mathbf{b}_i\|^2. \tag{5.12}$$

Based on Hadamard's inequality an upper bound can be defined for the orthogonality defect $\xi(\mathbf{B}) \geq 1$, with equality if and only if the columns of $\mathbf{B}$ are orthogonal to each other.

5. *Successive Minima.* Given an n-dimensional lattice $\mathcal{L}$, the i-th successive minima for $1 \leq i \leq n$ is defined as the radius of the smallest closed ball centered at the origin containing at least i linearly independent lattice vectors. More formally, for any lattice $\mathcal{L}$ let $\lambda_i(\mathcal{L})$ be the i-th successive minimum defined by:

$$\begin{aligned}\lambda_i(\mathcal{L}) = \inf\Bigg\{ \lambda \geq 0 \Bigg| \mathcal{L} &\text{ contains at least } i \text{ linearly independent vectors} \\ &\mathbf{b}_j \text{ for } j = 1, \ldots, i \text{ such that } |\mathbf{b}_j| \leq \lambda \Bigg\}.\end{aligned} \tag{5.13}$$

The shortest nonzero lattice vector of $\mathcal{L}$ (with respect to the Euclidean norm) is denoted as $\lambda_1(\mathcal{L})$.

6. *Dual lattice basis.* If a lattice $\mathcal{L}$ is defined by basis vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ then the dual lattice $\mathcal{L}^d$ of $\mathcal{L}$ is defined by basis vectors $(\mathbf{b}_1^d, \ldots, \mathbf{b}_n^d)$, where

$$\begin{aligned}(\mathbf{b}_i, \mathbf{b}_i^d) &= 1, \\ (\mathbf{b}_i, \mathbf{b}_j^d) &= 0, \qquad \text{for } i \neq j.\end{aligned} \tag{5.14}$$

The $(\cdot, \cdot)$ operator denotes the ordinary inner product on $\mathbb{R}^n$. The above conditions can be satisfied with the help of the right Moore-Penrose pseudoinverse, thus, the dual lattice basis is defined as

$$\mathbf{B}^d = \mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}. \tag{5.15}$$

Geometrically, this means that the dual basis vector $\mathbf{b}_k^d$ is orthogonal to the subspace spanned by the primal basis vectors $\mathbf{b}_1, \cdots, \mathbf{b}_{k-1}, \mathbf{b}_{k+1}, \cdots, \mathbf{b}_n$. The determinant of the dual lattice is easily seen to be given by $d(\mathcal{L}^d) = 1/d(\mathcal{L})$.

7. *Associated orthogonal basis.* Let $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*) \in \mathbb{R}^{n\times n}$ denote the associated orthogonal basis of $\mathbf{B}$, calculated by the Gram-Schmidt orthogonalization process

as follows:

$$\begin{aligned} \mathbf{b}_1^* &= \mathbf{b}_1 \\ \mathbf{b}_i^* &= \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \cdot \mathbf{b}_j^* \qquad \text{for } 2 \le i \le n, \end{aligned} \tag{5.16}$$

where $\mu_{i,j}$ are the Gram-Schmidt coefficients and they are defined as

$$\mu_{i,j} = (\mathbf{b}_i, \mathbf{b}_j^*)/(\mathbf{b}_j^*, \mathbf{b}_j^*) \qquad \text{for } 1 \le j < i \le n. \tag{5.17}$$

With $\mu_{i,i} = 1$ for $1 \le i \le n$ and $\mu_{i,j} = 0$ for $i < j$ the following equation holds

$$\begin{aligned} \mathbf{B} &= \mathbf{B}^* \cdot \mathbf{U} \\ (\mathbf{b}_1, \dots, \mathbf{b}_n) &= (\mathbf{b}_1^*, \dots, \mathbf{b}_n^*) \cdot \begin{pmatrix} 1 & \mu_{2,1} & \mu_{3,1} & \cdots & \mu_{n,1} \\ 0 & 1 & \mu_{3,2} & \cdots & \mu_{n,2} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \end{aligned} \tag{5.18}$$

where matrix $\mathbf{U}$ is upper triangular with unit diagonal and elements above the diagonal are the Gram-Schmidt coefficients $\mu_{i,j}$.

Some papers apply the QR factorization instead of the Gram-Schmidt orthogonalization, because it is numerically more stable. In the following it is showed how the resulting orthogonal matrix $\mathbf{B}^\star$ and the upper triangular matrix $\mathbf{U}$ with unit diagonal can be transformed to matrices $\mathbf{Q}$ and $\mathbf{R}$ that are the results of the QR factorization. By defining the diagonal matrix $\mathbf{D} = \operatorname{diag}(d_i)$, where $d_i = \|\mathbf{b}_i^\star\|$, a further decomposition of $\mathbf{B}^\star = \mathbf{QD}$ is possible. As a result, matrix $\mathbf{R} = \mathbf{DU}$ is defined with the help of diagonal matrix $\mathbf{D}$. As a conclusion the following relations will hold:

- $\mathbf{q}_i = \mathbf{b}_i^\star / \|\mathbf{b}_i^\star\|$,
- $r_{i,i} = d_i = \|\mathbf{b}_i^\star\|$,
- $r_{i,j} = r_{i,i} u_{i,j} = d_i u_{i,j}$.

8. *Complex-valued lattices.* The previous discussion of real-valued point lattices can be generalized to the complex case. Specifically, a complex-valued lattice in the

DOI:10.15774/PPKE.ITK.2015.010

complex space $\mathbb{C}^n$ is defined as

$$\mathcal{L}(\mathbf{B}^c) = \left\{ \mathbf{x}^c \middle| \mathbf{x}^c = \sum_{i=1}^{n} z_i^c \cdot \mathbf{b}_i^c, z_i^c \in \mathbb{Z}_j \right\}, \tag{5.19}$$

where $\mathbf{b}_i \in \mathbb{C}^n$ denotes the complex basis vectors and $\mathbb{Z}_j = \mathbb{Z} + j\mathbb{Z}$ denotes the set of complex integers. The transformation of the complex mapping to real-valued lattice basis and points can be carried out as follows

$$\mathbf{x} = \begin{pmatrix} \Re(\mathbf{x}^c) \\ \Im(\mathbf{x}^c) \end{pmatrix}_{M\times 1}, \mathbf{z} = \begin{pmatrix} \Re(\mathbf{z}^c) \\ \Im(\mathbf{z}^c) \end{pmatrix}_{N\times 1}, \mathbf{B} = \begin{pmatrix} \Re(\mathbf{B}^c) & -\Im(\mathbf{B}^c) \\ \Im(\mathbf{B}^c) & \Re(\mathbf{B}^c) \end{pmatrix}_{M\times N}. \tag{5.20}$$

The above technique is similar to the transformation of the complex-valued MIMO system model to the real-valued MIMO system model as shown in Eq. 3.5. The transformations from complex to real can make the computations more treatable for specific computing architectures, however, in this case the problem dimension is doubled that can lead to an increase in the complexity as well.

The goal of lattice basis reduction is to find, for a given lattice, a basis matrix with favorable properties. The reduction process performs a sequence of unimodular transformations until the proposed reduction criteria is achieved. Depending on the reduction criteria, the result of the lattice reduction process can significantly differ. A favorable basis $\tilde{\mathbf{B}}$ has vectors that are more orthogonal and shorter, in the sense of Euclidean norm, than the original ones. In the following $\tilde{\mathbf{B}}$ will denote an already reduced basis.

The possible metrics that measure the quality of a lattice basis are the condition number $\kappa(\tilde{\mathbf{B}})$, the orthogonality defect $\xi(\tilde{\mathbf{B}})$ and the Seysen criterion $S(\tilde{\mathbf{B}})$. The condition number of an arbitrary lattice basis $\mathbf{B}$ is defined as

$$\kappa(\mathbf{B}) = \sigma_{max}(\mathbf{B})/\sigma_{min}(\mathbf{B}), \tag{5.21}$$

where $\sigma_{max}(\mathbf{B})$ and $\sigma_{min}(\mathbf{B})$ are the maximal and minimal singular values of $\mathbf{B}$, respectively.

5.3 Lattice reduction algorithms

In this section a brief overview on the most important lattice reduction criteria and algorithms are given. The lattice reduction introduced by Hermite, Korkine, and Zolotareff, and Minkowski suffer from an exponential complexity with respect to the lattice

DOI:10.15774/PPKE.ITK.2015.010

dimension, however, they provide the best quality reduction. With the relaxation of the proposed conditions more practical reductions were defined in the LLL algorithm, and a lattice reduction concept based on the dual of the lattice was introduced by Seysen.

The aim of lattice reduction is to construct reduced bases where the basis vectors are shorter and have an improved orthogonality. The main differences between the various lattice reduction algorithms are the imposed conditions, that will influence the achieved orthogonality, the norm of the lattice basis and the computational cost.

5.3.1 Hermite-Korkine-Zolotareff and Minkowski lattice basis reduction

The first algorithm for constructing *Hermite-Korkine-Zolotareff* (HKZ) reduced lattice basis was introduced by Kannan in [107]. Because of its exponential complexity its use in practical systems is not possible, however, it can serve as a theoretical upper bound. A complexity analysis of HKZ reduction in the context of decoding was presented in [108]. Further improvements of the Kanaan's algorithm were presented in [109], [110], [111]. In the following a brief overview is given on Kanaan's HKZ-reduced basis finding algorithm.

The notion of weak reduction for lattices was introduced by Hermite in the context of quadratic forms. The weak reduction states that a lattice basis $\mathbf{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ is called size-reduced if the Gram-Schmidt coefficients $\mu_{i,j}$ satisfy

$$|\mu_{i,j}| \leq 1/2, \text{ for } 1 \leq i < j \leq n, \tag{5.22}$$

or the $r_{i,j}$ elements of its $\mathbf{B} = \mathbf{QR}$ decomposition satisfy

$$|r_{i,j}| \leq 1/2|r_{i,i}|, \text{ for } 1 \leq i < j \leq n. \tag{5.23}$$

This form of the reduction is referred to as the *size reduction.* A compact overview of lattice reduction criteria and algorithms is given in [112].

Later, Korkine and Zolotareff further strengthened the reduction criteria. Before defining the reduction criteria the notion of orthogonal complement and orthogonal projection are defined.

Definition Let W be a subspace of $\mathbb{R}^n$ with $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$ being an orthogonal basis for

DOI:10.15774/PPKE.ITK.2015.010

W. For any vector $\mathbf{v} \in \mathbb{R}^n$ the *orthogonal projection* of $\mathbf{v}$ onto W is defined as

$$\text{proj}_W(\mathbf{v}) = \frac{(\mathbf{u}_1, \mathbf{v})}{(\mathbf{u}_1, \mathbf{u}_1)}\mathbf{u}_1 + \ldots + \frac{(\mathbf{u}_k, \mathbf{v})}{(\mathbf{u}_k, \mathbf{u}_k)}\mathbf{u}_k. \tag{5.24}$$

Definition Let W be a subspace of $\mathbb{R}^n$, $\mathbf{v} \in \mathbb{R}^n$ is *orthogonal* to W if $\mathbf{v}$ is orthogonal to every vector in W. The set of all vectors that are orthogonal to W is called the *orthogonal complement* of W denoted by $\ll \cdots \gg^{\perp}$ and defined as

$$W^{\perp} = \left\{ \mathbf{v} \in \mathbb{R}^n \middle| (\mathbf{v}, \mathbf{w}) = 0 \text{ for all } \mathbf{w} \in W \right\}. \tag{5.25}$$

Let $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ denote a fixed lattice basis. The projection of $\mathbf{a} \in \mathbb{R}^m$ on $\ll \mathbf{b}_1, \ldots, \mathbf{b}_k \gg^{\perp}$ is denoted by $\mathbf{a}(k)$. Similarly, let the projection of $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ on $\ll \mathbf{b}_1, \ldots, \mathbf{b}_k \gg^{\perp}$ be denoted as $\mathcal{L}_k(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. Vectors $\mathbf{b}_k(i)$ can be computed with the Gram-Schmidt orthogonalization process as follows:

$$\mathbf{b}_k(i) = \mathbf{b}_k - \sum_{j=1}^{i-1} \mu_{k,j}\mathbf{b}_j(j), \qquad \text{for } 1 \leq k \leq i \leq n, \tag{5.26}$$

where $\mu_{k,j} = (\mathbf{b}_k, \mathbf{b}_j(j))/(\mathbf{b}_j(j), \mathbf{b}_j(j))$.

Definition A basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is HKZ-reduced if it satisfies the following recursive set of conditions:

- $\mathbf{b}_1$ is a shortest non-zero vector of $\mathcal{L}$ in the Euclidean norm,
- $|\mu_i, 1| \leq 1/2$ for $2 \leq i \leq n$,
- if $\mathcal{L}_2(\mathbf{b}_2(2), \ldots, \mathbf{b}_n(2))$ denotes the projection of $\mathcal{L}_1$ on the orthogonal complement $\ll \mathbf{b}_1 \gg^{\perp}$, then the projections $\mathbf{b}_2(2), \ldots, \mathbf{b}_n(2)$ yield a Korkin-Zolotarev basis of $\mathcal{L}_2$.

Later, Minkowski introduced a stronger reduction criteria, which is now known as Minkowski-reduction. After performing Minkowski-reduction the first vector $\tilde{\mathbf{b}}_1$ of the reduced basis $\tilde{\mathbf{B}}$ is the shortest non-zero vector of the lattice $\mathcal{L}(\tilde{\mathbf{B}})$. Furthermore, every $\tilde{\mathbf{b}}_j$ for $2 \leq j \leq n$ has to be a shortest vector in $\mathcal{L}(\tilde{\mathbf{B}})$ that is linearly independent of $\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_{j-1}$.

Definition An ordered basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is reduced in the sense of Minkowski, or that is a Minkowski reduced basis, if it satisfies the following conditions:

DOI:10.15774/PPKE.ITK.2015.010

- $\mathbf{b}_1$ is a shortest non-zero vector of $\mathcal{L}$ in the Euclidean norm,
- $\mathbf{b}_i$ is a shortest vector among lattice vectors not in the span$(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$, for $i = 2, 3, \ldots, n$.

Algorithms that construct Minkowksi-reduced basis were presented in [109], [113], [114]. These algorithms, similar to the algorithms mentioned in the context of HKZ reduction, suffer from an exponential complexity with respect to the lattice dimension. Thus, their use in practical systems is not feasible.

Before giving the details of the HKZ algorithm the notion of *lifting* has to be introduced. The lifting is the procedure when a vector given in $\mathcal{L}_{k+1}$ is determined with the basis of $\mathcal{L}_k$. Equally, lifting is the search of a vector in $\mathbf{v}(k) \in \mathcal{L}_k$ whose projection on $\mathcal{L}_{k+1}$ is $\mathbf{v}(k+1)$. Let $\mathbf{v}(k+1) = \sum_{i=k+1}^{n} v_i \mathbf{b}_i(k+1) \in \mathcal{L}_{k+1}$ denote the vector that has to be lifted to $\mathcal{L}_k$. Let $\bar{\mathbf{v}}(k) \in \mathcal{L}_k$ denote a vector with the same coordinates as $\mathbf{v}(k+1)$ but with the basis vectors of $\mathcal{L}_{k+1}$ defined as

$$\bar{\mathbf{v}}(k) = \sum_{i=k+1}^{n} v_i \mathbf{b}_i(k). \tag{5.27}$$

In order to get a shortest vector $v(k) \in \mathcal{L}_k$ whose projection on $\mathcal{L}_{k+1}$ is $\mathbf{v}(k+1)$ the subtraction of the common parts is done as

$$\mathbf{v}(k) = \bar{\mathbf{v}}(k) - \mathbf{b}_k(k) \frac{(\bar{\mathbf{v}}(k), \mathbf{b}_k(k))}{\mathbf{b}_k(k), \mathbf{b}_k(k)}. \tag{5.28}$$

Algorithm 7 constructs a HKZ reduced basis. This algorithm was originally introduced by Kanaan in [107] and refined by Helfrich in [109].

5.3.2 The Lenstra-Lenstra-Lovász lattice basis reduction

In [101] Lenstra et al. proposed a polynomial time lattice reduction algorithm. In the literature, this algorithm is referred to as the LLL reduction algorithm. Because the algorithm performs well in practice it is an extensively used technique.

Definition Given a lattice $\mathcal{L}$ with basis $\mathbf{B} = (\mathbf{b}_1, \cdots, \mathbf{b}_n) \in \mathbb{R}^{n \times n}$, associated orthogonal basis $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*) \in \mathbb{R}^{n \times n}$, and Gram-Schmidt coefficients $\mu_{i,j}$, $\mathbf{B}$ is called *LLL-reduced* if the following conditions are satisfied:

$$|\mu_{i,j}| \leq \frac{1}{2} \qquad \text{for } 1 \leq j < i \leq n \tag{5.29}$$

$$\|\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*\|^2 \geq \delta \|\mathbf{b}_{i-1}^*\|^2 \qquad \text{for } 1 < i \leq n,\ \frac{3}{4} \leq \delta < 1. \tag{5.30}$$

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 7 Hermite-Korkin-Zolotareff lattice reduction algorithm

```
1: Input: $(n, \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$
2: Output: $(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ as HKZ reduced basis
3: if $n = 1$ then $\mathbf{b}_1$ is HKZ reduced return
4: $(\mathbf{b}_1, \ldots, \mathbf{b}_n) \leftarrow$ perform LLL lattice reduction on $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ and replace basis ▷ The details of LLL lattice reduction are presented in Sec. 5.3.2
5: $(\mathbf{b}'_2, \ldots, \mathbf{b}'_n) \leftarrow \mathrm{HKZ}(n-1, \mathbf{b}_2(2), \ldots, \mathbf{b}_n(2))$
6: for $i = 2 \to n$ do
7:     $\mathbf{b}_i \leftarrow$ a shortest lattice element whose projection on $\ll \mathbf{b}_1 \gg^{\perp}$ is $\mathbf{b}'_i$ ▷ $\mathbf{b}'_i$ is lifted to $\mathcal{L}_n$ as discussed in Eqs. 5.27 and 5.28
8: end for
9: if $|\mathbf{b}_1|^2 > \frac{4}{3}|\mathbf{b}_2|^2$ then $(\mathbf{b}_1, \mathbf{b}_2) \leftarrow \mathrm{HKZ}(2, \mathbf{b}_1, \mathbf{b}_2)$ end if ▷ With this call the LLL reduction will swap $\mathbf{b}_1$ and $\mathbf{b}_2$
10: Find the shortest lattice vector $\mathbf{v}$ ▷ One possible solution is to launch the SD algorithm to find the closest vector to the origin
11: Construct a lattice basis $(\mathbf{v}, \mathbf{b}_1, \ldots, \mathbf{b}_n)$ with $\mathbf{v}$ being the first column ▷ The *Selectbasis* procedure introduced by Kanaan in [107] constructs the basis in polynomial time
12: $(\mathbf{b}'_2, \ldots, \mathbf{b}'_n) \leftarrow \mathrm{HKZ}(n-1, \mathbf{b}_2(2), \ldots, \mathbf{b}_n(2))$
13: for $i = 2 \to n$ do
14:     $\mathbf{b}_i \leftarrow$ a shortest lattice element whose projection on $\ll \mathbf{b}_1 \gg^{\perp}$ is $\mathbf{b}'_i$
15: end for
```

During the reduction process, local changes are made based on the conditions presented in Eqs. 5.29 and 5.30 in order to achieve a reduced basis. Algorithm 8 gives a detailed overview of the LLL algorithm. Practically, two types of unimodular transformations are performed repeatedly, namely the swap and the repeated translation. Equations 5.3 and 5.5 show how unimodular matrices can be constructed for these type of transformations. After a size reduction or swap the Gram-Schmidt coefficients and the associated orthogonal basis have to be updated.

If a lattice basis satisfies the above conditions with $\delta = 3/4$ then the following bounds can be defined

$$\begin{aligned} |\mathbf{b}_j|^2 \leq 2^{i-1}|\mathbf{b}^*_i| \text{ for } 1 \leq j \leq i \leq n, \\ d(\mathcal{L}) \leq \prod_{i=1}^{n} |\mathbf{b}_i| \leq 2^{n(n-1)/4} d(\mathcal{L}), \\ |\mathbf{b}_1| \leq 2^{(n-1)/4} d(\mathcal{L})^{1/n}. \end{aligned} \tag{5.31}$$

Based on the above the first vector in the reduced lattice basis $\mathbf{b}_1$ satisfies $|\mathbf{b}_1^2| \leq 2^{n-1}|\mathbf{x}|^2$ for every $\mathbf{x} \in \mathcal{L}, \mathbf{x} \neq 0$. Theoretically, the length of $\mathbf{b}_1$ is at most an exponential multiple of the length of the shortest nonzero vector in the lattice. The proof is given in [101].

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 8 The Lenstra-Lenstra-Lovász lattice reduction algorithm

```
Input: $\mathbf{B}$, $\delta$
Output: LLL-reduced basis
Compute $\mathbf{B}^*$ and $\mathbf{U}$ with the Gram-Schmidt algorithm
$k = 2$
while $k \leq n$ do
    SizeReduce($k$,$k-1$)
    if $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$ then
        Swap($k$)
        $k = \max(k-1, 2)$
    else
        for $l = k-2 \to 1$ do
            SizeReduce($k$,$l$)
        end for
        $k = k + 1$
    end if
end while
procedure SizeReduce($k$,$l$)
    if $|\mu_{k,l}| > \frac{1}{2}$ then
        $\mu = \lceil \mu_{k,l} \rfloor$, $\mu_{k,l} = \mu_{k,l} - \mu$, $\mathbf{b}_k = \mathbf{b}_k - \mu \cdot \mathbf{b}_l$
        for $j = 1 \to l-1$ do
            $\mu_{k,j} = \mu_{k,j} - \mu \cdot \mu_{l,j}$
        end for
    end if
end procedure
procedure Swap($k$)
    Swap $\mathbf{b}_k$ with $\mathbf{b}_{k-1}$
    $\mathbf{b}_{k-1}^{*'} = \mathbf{b}_k^* + \mu_{k,k-1}\mathbf{b}_{k-1}^*$
    $\mu'_{k,k-1} = (\mathbf{b}_{k-1}^*, \mathbf{b}_{k-1}^{*'})/\|\mathbf{b}_{k-1}^{*'}\|^2$
    $\mathbf{b}_k^{*'} = \mathbf{b}_{k-1}^* - \mu'_{k,k-1}\mathbf{b}_{k-1}^{*'}$
    for $j = 1 \to k-2$ do
        Swap $\mu_{k,j}$ with $\mu_{k-1,j}$
    end for
    for $i = k+1 \to n$ do
        $\mu'_{i,k-1} = \mu_{i,k-1} \cdot \mu'_{k,k-1} + \mu_{i,k} \cdot \|\mathbf{b}_k^*\|^2/\|\mathbf{b}_{k-1}^{*'}\|^2$
        $\mu'_{i,k} = \mu_{i,k-1} - \mu_{i,k} \cdot \mu_{k,k-1}$
        $\mu_{i,k} = \mu'_{i,k}$, $\mu_{i,k-1} = \mu'_{i,k-1}$
    end for
    $\mathbf{b}_{k-1}^* = \mathbf{b}_{k-1}^{*'}$,$\mathbf{b}_k^* = \mathbf{b}_k^{*'}$,$\mu_{k,k-1} = \mu'_{k,k-1}$
end procedure
```

DOI:10.15774/PPKE.ITK.2015.010

5.3.3 Seysen's lattice basis reduction

In Sec. 5.3.2 it was shown that the LLL algorithm achieves reduced basis by local operations such as swap and size reductions. The reduction concept introduced by Seysen in [103] attempts to globally reduce a lattice basis by performing reduction operations on the lattice basis and its dual basis, respectively. Furthermore, Seysen's reduction algorithm can be successfully applied to solve subset sum problems as well.

Since Seysen's algorithm tries to find a good basis for the dual lattice as well, it is important to see how translation will effect the dual lattice. The definition of the dual lattice basis was given in Eq. 5.14. Recall that during translation a constant multiple of one lattice basis vector is added to another basis vector $\tilde{\mathbf{b}}_l = \mathbf{b}_l + \mu \mathbf{b}_k$. For $j \neq k$ no further operations are required since

$$\begin{aligned}(\tilde{\mathbf{b}}_l, \mathbf{b}_j^*) &= (\mathbf{b}_l + \mu \mathbf{b}_k, \mathbf{b}_j^*) \\ &= (\mathbf{b}_l, \mathbf{b}_j^*) + \mu(\mathbf{b}_k, \mathbf{b}_j^*) = 0.\end{aligned} \tag{5.32}$$

However, when $j = k$, a change must occur in the dual lattice as well, otherwise Eq. 5.14 will not hold. In this case a negative translation is required, thus $\tilde{\mathbf{b}}_k^* = \mathbf{b}_k^* - \mu \mathbf{b}_l^*$. It can be seen that with the negative translation the dual basis satisfies the condition

$$\begin{aligned}(\tilde{\mathbf{b}}_l, \tilde{\mathbf{b}}_k^*) &= (\mathbf{b}_l, \mathbf{b}_k^* - \mu \mathbf{b}_l^*) + \mu(\mathbf{b}_k, \mathbf{b}_k^* - \mu \mathbf{b}_l^*) \\ &= (\mathbf{b}_l, \mathbf{b}_k^*) - \mu(\mathbf{b}_l, \mathbf{b}_l^*) + \mu(\mathbf{b}_k, \mathbf{b}_k^*) - \mu^2(\mathbf{b}_k, \mathbf{b}_l^*) \\ &= 0 - \mu + \mu + 0 = 0.\end{aligned} \tag{5.33}$$

For each lattice basis $\mathbf{B}$ the *associated positive definite quadratic form* is defined as $\mathbf{A} = \mathbf{B}^T\mathbf{B}$. Let $\mathbf{A}^{-1}$ denote the symmetric positive definite inverse of $\mathbf{A}$ and let $a_{i,j}^*$ denote the element of matrix $\mathbf{A}^{-1}$ at row i and column j. A basis transformation $\tilde{\mathbf{B}} = \mathbf{BT}$ with unimodular matrix $\mathbf{T}$ transforms the associated quadratic form as $\tilde{\mathbf{A}} = \mathbf{T}^T\mathbf{AT}$.

Definition For any quadratic form $\mathbf{A}$ with inverse $\mathbf{A}^{-1}$ the *Seysen measure* $S(\mathbf{A})$ is defined as

$$S(\mathbf{A}) = \sum_{i=1}^{n} a_{i,i} \cdot a_{i,i}^* = \sum_{i=1}^{n} \|\mathbf{b}_i\|^2 \|\mathbf{b}_i^*\|^2. \tag{5.34}$$

Definition A lattice basis and its associated quadratic form $\mathbf{A}$ is called S-reduced if $S(\mathbf{A}) \leq S(\mathbf{T}^T\mathbf{AT})$ holds for all $\mathbf{T} \in \mathbb{Z}^{n\times n}$ unimodular matrices.

DOI:10.15774/PPKE.ITK.2015.010

Based on the above definition Seysen in [103] proved the following bounds

$$\begin{aligned}\lambda_i(\mathbf{A})^2 \le a_{i,i} \le S(\mathbf{A})^2 \cdot \lambda_i(\mathbf{A})^2,\\ \frac{1}{S(\mathbf{A})\cdot\lambda_i(\mathbf{A})^2} \le a^*_{i,i} \le \frac{S(\mathbf{A})}{\lambda_i(\mathbf{A})^2}.\end{aligned} \tag{5.35}$$

Since the computation of S-reduced basis is computationally difficult, Seysen introduced the notion of S_2-reduction that is a relaxed version of the S-reduction. Instead of searching for a general unimodular matrix $\mathbf{T}$, the reduction process is implemented with the repeated translation operator $\mathbf{T}^{\mu}_{k,l}$.

Definition A quadratic form is called S_2-reduced if

$$S(\mathbf{A}) \le S(\mathbf{T}^{\mu}_{k,l}\mathbf{A}\mathbf{T}^{\mu}_{l,k}) \tag{5.36}$$

holds for all $k, l, \mu \in \mathbb{Z}$ with $1 \le i \ne j \le n$.

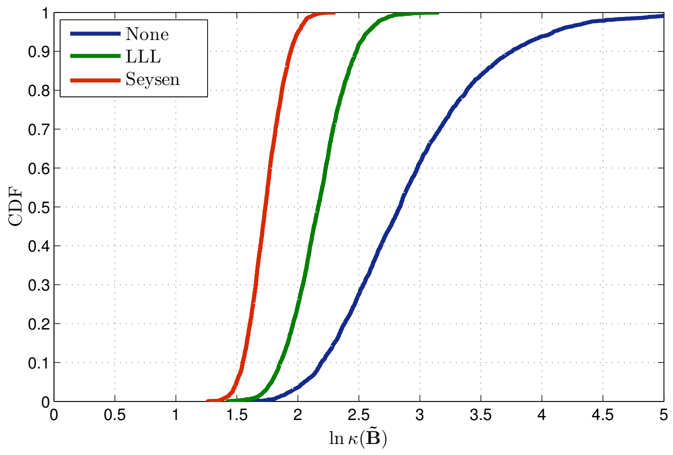
Algorithm 9 Seysen's lattice reduction algorithm

```
1: Input: The associated quadratic form $\mathbf{A}$ of the lattice basis
2: Output: $S_2$-reduced lattice basis
3: while $\mathbf{A}$ is not $S_2$-reduced do
4:     Choose $k, l$ such that $\exists\mu \in \mathbb{Z}$ with
5:         $S(\mathbf{T}^{\mu}_{(\mathbf{l},\mathbf{k})}\mathbf{A}\mathbf{T}^{\mu}_{(\mathbf{k},\mathbf{l})}) \le S(\mathbf{A})$
6:     let $\mu = \lceil \frac{1}{2}(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{(i,j)}}{a_{(i,i)}}) \rfloor$
7:     let $\mathbf{A} = \mathbf{T}^{\mu}_{(l,k)}\mathbf{A}\mathbf{T}^{\mu}_{(k,l)}$.
8: end while
```

In Alg. 9 it is not specified how i and j is selected. The easiest way is to implement two for loops and enumerate all the (i, j) pairs and then repeat the process until $S(\mathbf{A})$ cannot be reduced anymore. There are no known bounds for the Seysen measure of an S_2-reduced basis nor on the length of the shortest nonzero vector in the basis.

In Fig. 5.2 the performance of the LLL and Seysen's lattice reduction is presented. The elements of the lattice basis are assumed to be i.i.d. zero-mean, complex circularly symmetric Gaussian variables with unit variance. The size of the lattice basis is 8×8, however, it is transformed to a real-valued basis based on Eq. 5.20, consequently, 16×16 real-valued basis are reduced. The reduction algorithms significantly decrease the size of the condition number $\kappa(\tilde{\mathbf{B}})$ and the orthogonality defect $\xi(\tilde{\mathbf{B}})$.

DOI:10.15774/PPKE.ITK.2015.010



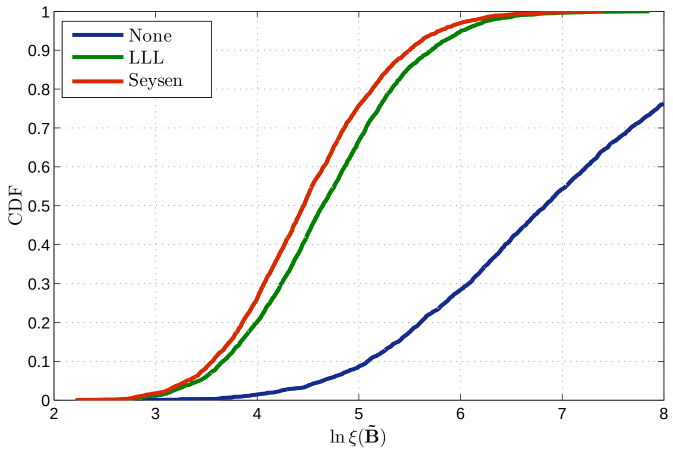


Figure 5.2: The cumulative distribution function of the condition number $\kappa(\tilde{\mathbf{B}})$ and orthogonality defect $\xi(\tilde{\mathbf{B}})$ after Lenstra-Lenstra-Lovász and Seysen lattice reduction for 16×16 zero-mean, unit variance Gaussian random matrices.

5.4 Lattice reduction aided signal processing

In this section application of lattice reduction to MIMO detection and MISO precoding is considered. The equivalent system models are given and the improvement in the BER is shown.

5.4.1 Lattice reduction aided MIMO detection

In [14] Yao et. al discussed how the performance of linear and non-linear detectors can be improved when used in conjuction with LR techniques. The detection techniques discussed in Sec. 4.2 assume that the channel is known by the receiver. Moreover, these techniques heavily rely on the channel matrix. The condition of the channel matrix highly influences the achieved BER. Thus, it is straightforward to regard the channel matrix as a lattice generator basis. Consequently, the condition number and the orthogonality defect of the channel matrix are improved. In [16], [18] it was shown that linear and

DOI:10.15774/PPKE.ITK.2015.010

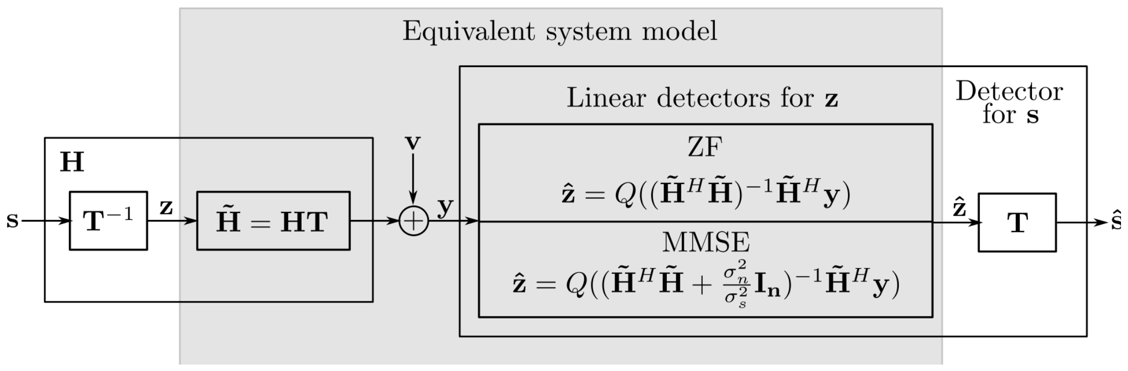

Equivalent system model
Linear detectors for $\mathbf{z}$
Detector for $\mathbf{s}$
$\mathbf{H}$
ZF
$\hat{\mathbf{z}} = Q((\tilde{\mathbf{H}}^H\tilde{\mathbf{H}})^{-1}\tilde{\mathbf{H}}^H\mathbf{y})$
MMSE
$\hat{\mathbf{z}} = Q((\tilde{\mathbf{H}}^H\tilde{\mathbf{H}} + \frac{\sigma_n^2}{\sigma_s^2}\mathbf{I_n})^{-1}\tilde{\mathbf{H}}^H\mathbf{y})$
$\mathbf{s}$
$\mathbf{T}^{-1}$
$\mathbf{z}$
$\tilde{\mathbf{H}} = \mathbf{HT}$
$\mathbf{v}$
$\mathbf{y}$
$\hat{\mathbf{z}}$
$\mathbf{T}$
$\hat{\mathbf{s}}$

Figure 5.3: Equivalent system model of lattice reduction aided MIMO detection.

non-linear detection based on the reduced basis $\tilde{\mathbf{H}}$ achieves full diversity order, which is a significant performance improvement. Instead of using the channel matrix, when computing the detector's weight matrix, it is better to work with the reduced channel matrix. For complex-valued systems, either the corresponding lattice reduction algorithm has to be adapted to the complex-valued case or the equivalent real-valued system model has to be used. In the following the real-valued system is considered.

In the context of the fundamental lattice structures it is straightforward that the Voronoi regions and the ML detection regions are equal and independent from the lattice basis. The ZF decision regions correspond to parallelotope $\mathcal{P}(\mathbf{H})$. After lattice reduction the basis vectors are shorter and more orthogonal, thus, parallelotope $\mathcal{P}(\tilde{\mathbf{H}})$ will be more similar to the Voronoi region $\mathcal{V}(\mathcal{L})$ than $\mathcal{P}(\mathbf{H})$, resulting in a better detector performance. In Fig. 5.1 different lattices were shown. In case of square lattices the Voronoi region and the paralellotopes are similar. Consequently, if the channel matrix happens to be orthogonal ZF detection will provide ML performance.

The equivalent lattice reduced model is shown in Fig. 5.3 and is described as follows

$$\mathbf{y} = \mathbf{HTT}^{-1}\mathbf{s} + \mathbf{n} = \tilde{\mathbf{H}}\mathbf{z} + \mathbf{n}. \tag{5.37}$$

If at the receiver side a linear ZF detector based on the reduced channel is implemented, the detection process is defined as follows

$$\tilde{\mathbf{z}}_{ZF} = \tilde{\mathbf{H}}^{\dagger}\mathbf{y} = \mathbf{z} + \tilde{\mathbf{H}}^{\dagger}\mathbf{n}. \tag{5.38}$$

Because $\tilde{\mathbf{H}}$ is more orthogonal the noise enhancement is reduced. Consequently, a quantization based on $\tilde{\mathbf{z}}_{ZF}$ is more reliable than that of $\tilde{\mathbf{s}}_{ZF}$. After the quantization of $\hat{\mathbf{z}}_{ZF} = Q(\tilde{\mathbf{z}}_{ZF})$, transformation $\hat{\mathbf{s}} = \mathbf{T}\hat{\mathbf{z}}_{ZF}$ is required to return to the original signal

DOI:10.15774/PPKE.ITK.2015.010

space.

The transformation also affects the original constellation. Thus, the decision regions of the transformed constellation differ from the original and the computation of these is very expensive. A suboptimal alternative was presented in [17] with the following steps

- quantize $\tilde{\mathbf{z}}$ with respect to $\mathbb{Z}^n$;
- return to the original symbol space by multiplying with $\mathbf{T}$,
- clip the result with respect to $\mathbb{D}$.

Another problem is related to the quantization of $\tilde{\mathbf{z}}$ with respect to $\mathbb{Z}^n$. As presented in Eq. 3.3 the elements of the M-QAM constellation are normalized and $\mathbb{D} \not\subset \mathbb{Z}$. Since $\mathbf{T}$ is a unimodular matrix it follows that $\mathbf{z} = \mathbf{T}^{-1}\mathbf{s} \notin \mathbb{Z}^n$. Recall, two basis $\mathbf{H}$ and $\tilde{\mathbf{H}}$ describe the same lattice $\mathcal{L}(\mathbf{H}) = \mathcal{L}(\tilde{\mathbf{H}})$ if $\tilde{\mathbf{H}} = \mathbf{HT}$, where $\mathbf{T}$ is unimodular, and if the input symbols are elements of the infinite integer space $\mathbb{Z}^n$.

A scaling and shifting transformation method of $\mathbb{D}$ was presented in [15]. The point is to decompose $\mathbb{D}$ to an integer subset $\mathbb{D}^n_{\mathbb{Z}} \subset \mathbb{Z}^n$ and a constant shifting element that are scaled as follows

$$\mathbb{D}^n = a(\mathbb{D}^n_{\mathbb{Z}} + \frac{1}{2}\mathbf{1}_n), \tag{5.39}$$

where $\mathbf{1}_n$ denotes an n-dimensional vector with unity elements and parameter a was defined in Eq. 3.3. Thus, the transmit signal vector $\mathbf{s} \in \mathbb{D}^n$ can be rewritten as $\mathbf{s} = a(\bar{\mathbf{s}} + \frac{1}{2}\mathbf{1}_n)$ with $\bar{\mathbf{s}} \in \mathbb{D}^n_{\mathbb{Z}}$. Consequently, the transformed signal vector $\mathbf{z}$ can be represented by

$$\mathbf{z} = \mathbf{T}^{-1}\mathbf{s} = a\mathbf{T}^{-1}\left(\bar{\mathbf{s}} + \frac{1}{2}\mathbf{1}^n\right) = a\left(\bar{\mathbf{z}} + \frac{1}{2}\mathbf{T}^{-1}\mathbf{1}^n\right), \tag{5.40}$$

where $\bar{\mathbf{z}} = \mathbf{T}^{-1}\bar{\mathbf{s}} \in \mathbf{T}^{-1}\mathbb{D}^n_{\mathbb{Z}} \subset \mathbb{Z}^n$.

In case of LR-aided linear ZF detection $\tilde{\mathbf{z}}$ is determined as

$$\tilde{\mathbf{z}} = \mathbf{z} + \tilde{\mathbf{H}}^{\dagger}\mathbf{n} = a(\bar{\mathbf{z}} + \frac{1}{2}\mathbf{T}^{-1}\mathbf{1}^n) + \tilde{\mathbf{H}}^{\dagger}\mathbf{n}. \tag{5.41}$$

Thus, before the quantization process $\bar{\mathbf{z}}$ is scaled and shifted

$$\hat{\mathbf{z}} = a\left(Q_{Z^n}\left\{\frac{1}{a}\tilde{\mathbf{z}} - \frac{1}{2}\mathbf{T}^{-1}\mathbf{1}^n\right\} + \frac{1}{2}\mathbf{T}^{-1}\mathbf{1}^n\right). \tag{5.42}$$

The estimation for the transmit signal is $\hat{\mathbf{s}} = \mathbf{T}\hat{\mathbf{z}}$ and can be rewritten as

$$\hat{\mathbf{s}} = a\mathbf{T}Q_{Z^n}\left\{\frac{1}{a}\tilde{\mathbf{z}} - \frac{1}{2}\mathbf{T}^{-1}\mathbf{1}^n\right\} + \frac{a}{2}\mathbf{1}^n. \tag{5.43}$$

DOI:10.15774/PPKE.ITK.2015.010

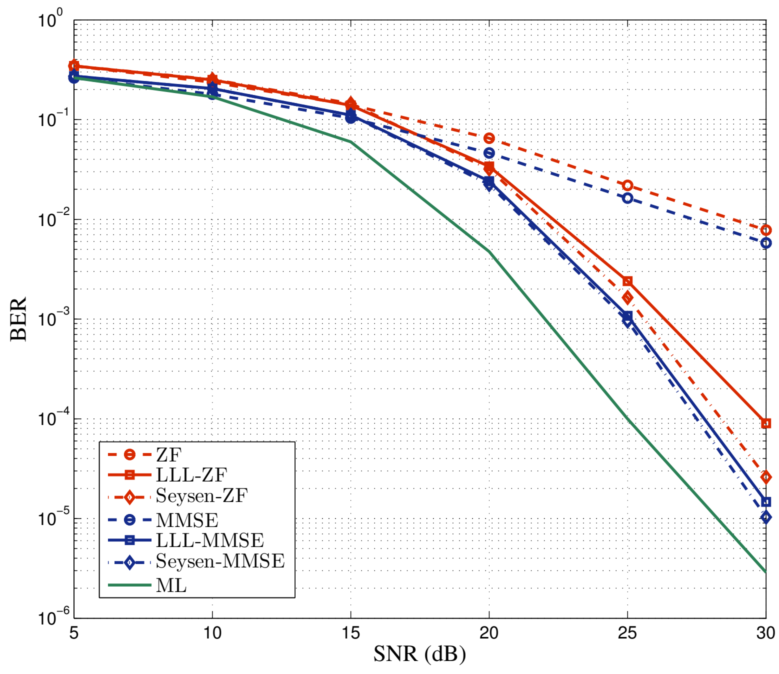


Figure 5.4: Bit error rate of lattice reduction aided linear detectors for 4×4 MIMO systems with 16-QAM symbol constellation.

In Fig. 5.4 it is shown how the LLL and Seysen's LR techniques improve the performance of ZF and MMSE detectors. It is visible that after LR full diversity order is achieved and Seysen's method is slightly better.

5.4.2 Lattice reduction aided MISO precoding

The downlink of an orthogonal frequency-division multiplexing (OFDM) multi-user MISO wireless communication system with a BS equipped with n antennas that serves m MSs equipped with one antenna each is considered in this section. The MSs share the same set of K OFDM subcarriers through spatial multiplexing. The received signal at the MSs on the k-th subcarrier is expressed as $\mathbf{y}^c[k] = \mathbf{H}^c[k]\mathbf{x}^c[k] + \mathbf{n}^c[k]$, where $\mathbf{y}^c[k] \in \mathbb{C}^{m\times 1}$ contains the received symbols for all the MSs, the element $h^c_{ij}[k]$ of the channel matrix $\mathbf{H}^c[k] \in \mathbb{C}^{m\times n}$ represents the complex fading gain from the j-th transmit antenna to the i-th MS. Vector $\mathbf{x}^c[k] \in \mathbb{C}^{n\times 1}$ contains the precoded symbols transmitted by the BS and $\mathbf{n}^c[k] \in \mathbb{C}^{m\times 1}$ is the noise vector on the k-th subcarrier. It is common to use the $M = 2m$ and $N = 2n$ dimensional real-valued equivalent system model as presented in Eq. 3.5.

The multi-user interference must be cancelled at the transmitter. ZF is the simplest precoding technique, but it performs poorly when the channel matrix is badly conditioned

DOI:10.15774/PPKE.ITK.2015.010

[11]. Other non-linear methods have been proposed in the literature to reduce the power of the precoded signal resulting in an increased SNR at the receiver compared to ZF precoding. Since the precoding must be performed for all the subcarriers in the following subcarrier index k is omitted.

Lattice reduction aided Tomlinson-Harashima precoding

Tomlinson-Harashima precoding (THP) is a technique that uses the modulo operation to limit the power of every transmitted symbol [10]. THP can also be used together with lattice reduction. Performing a lattice reduction over the channel matrix, $\tilde{\mathbf{H}} = \mathbf{T}_L\mathbf{H}$ with unimodular matrix $\mathbf{T}_L$ results in the reduced basis $\tilde{\mathbf{H}}$. The LQ-factorization is performed on the reduced basis $\tilde{\mathbf{H}} = \mathbf{L}_0\mathbf{Q}_0 = (\mathbf{L}_0\mathbf{G}^{-1})(\mathbf{G}\mathbf{Q}_0) = \mathbf{L}\mathbf{Q}$, where $\mathbf{L}_0$ is a lower triangular matrix, $\mathbf{Q}_0$ is an orthogonal matrix and $\mathbf{G}$ is a diagonal matrix containing the diagonal of $\mathbf{L}_0$. Transforming the original real-valued vector of data symbols $\mathbf{s} \in \mathbb{R}^{M\times 1}$ with $\mathbf{T}_L$ (step A2.1 in Table 5.1) allows to perform the THP on the reduced matrix $\tilde{\mathbf{H}}$, which shows better orthogonality properties. The different steps of the algorithm are shown in Table 5.1, section A, although a more detailed description can be found in [13], [115]. A possible value of parameter A could be set as $A = 2\sqrt{|\tilde{\Omega}|}$.

Lattice reductiona aided Vector Perturbation precoding

The vector perturbation technique consists of perturbing the original data to reduce the power of the precoded signal [12]. The optimal vector perturbation can be expressed as

$$\mathbf{p} = \arg \min_{\mathbf{p}' \in A\mathbb{Z}^M} \|\mathbf{H}^\dagger(\mathbf{s} + \mathbf{p}')\|^2. \tag{5.44}$$

The calculation of the optimal perturbation is a very computationally demanding task and sub-optimal techniques are usually employed. A brief overview of two different techniques is given in Table 5.1, subsections B and C, respectively. A detailed description of both algorithms can be found in [116], [115]. As shown in [115],the lattice-reduction-aided precoding (LRAP) V-BLAST algorithm shows a better performance but also a higher computational complexity than LRAP linear precoders. Similarly to LR-THP, the step C2.2 in Table 5.1 shows a sequential calculation of the different elements of vector $\tilde{\mathbf{q}}$.

The achieved BER performance for very large multi-user MISO systems with antenna configurations (a) $N = M = 64$ and (b) $N = M = 128$ are shown in Figs. 5.5 and 5.6, where the following algorithms were evaluated: (i) ZF precoding [11], (ii) THP precoding [10] and (iii) the LRAP techniques presented above. The LRAP techniques significantly reduce the BER compared to ZF and THP showing a higher diversity order. Similar

DOI:10.15774/PPKE.ITK.2015.010

Table 5.1: Lattice reduction aided precoding algorithms.

```
A. LR-THP algorithm [13]
A1.- Preprocessing stage:
1.-  $\widetilde{\mathbf{H}} = \mathbf{T}_{\mathrm{L}}\mathbf{H}$
2.-  $\widetilde{\mathbf{H}} = \mathbf{L}_0\mathbf{Q}_0 = (\mathbf{L}_0\mathbf{G}^{-1})(\mathbf{G}\mathbf{Q}_0) = \mathbf{L}\mathbf{Q}$
3.-  $\mathbf{Q}^{\dagger} = (\mathbf{G}\mathbf{Q}_0)^{\dagger} = \mathbf{Q}_0^{\mathrm{T}}\mathbf{G}^{-1}$
A2.- Per-symbol-vector processing stage:
1.-  $\hat{\mathbf{s}} = \mathbf{T}_{\mathrm{L}}\mathbf{s}$
2.-  for $k = 1, \ldots, M$
         $\hat{x}_k = \hat{s}_k - \sum_{l=1}^{k-1} l_{k,l}\tilde{x}_l$
         $\tilde{x}_k = \hat{x}_k \bmod A = \hat{x}_k - A\left\lfloor \frac{\hat{x}_k}{A} + 0.5 \right\rfloor$
     end
3.-  $\mathbf{x} = \mathbf{Q}^{\dagger}\tilde{\mathbf{x}}$

B. LRAP-LIN algorithm [116]
B1.- Preprocessing stage:
1.-  $\mathbf{H}^{\dagger} = \tilde{\mathbf{H}}^{\dagger}\mathbf{T}_R$
B2.- Per-symbol-vector processing stage:
1.-  $\mathbf{p}_{\mathrm{app}} = -A\mathbf{T}_R^{-1}\lceil \frac{\mathbf{T}_R\mathbf{s}}{A} \rfloor$
2.-  $\mathbf{x} = \mathbf{H}^{\dagger}(\mathbf{s} + \mathbf{p}_{\mathrm{app}})$

C. LRAP V-BLAST algorithm [116]
C1.- Preprocessing stage:
1.-  $\mathbf{H}^{\dagger} = \tilde{\mathbf{H}}^{\dagger}\mathbf{T}_{\mathrm{R}}$
2.-  $\mathbf{Q}\tilde{\mathbf{H}}^{\dagger} = \mathbf{L}$
C2.- Per-symbol-vector processing stage:
1.-  $\mathbf{q} = -\mathbf{Q}\mathbf{H}^{\dagger}\mathbf{s}$
2.-  for $k = 1, \ldots, M$
         $\tilde{q}_k = A\lceil \frac{q_k - \sum_{l=1}^{k-1} l_{k,l}\tilde{q}_l}{A} \rfloor$
     end
3.-  $\mathbf{p}_{\mathrm{app}} = \mathbf{T}_{\mathrm{R}}\widetilde{\mathbf{q}}$
4.-  $\mathbf{x} = \mathbf{H}^{\dagger}(\mathbf{s} + \mathbf{p}_{\mathrm{app}})$
```

DOI:10.15774/PPKE.ITK.2015.010

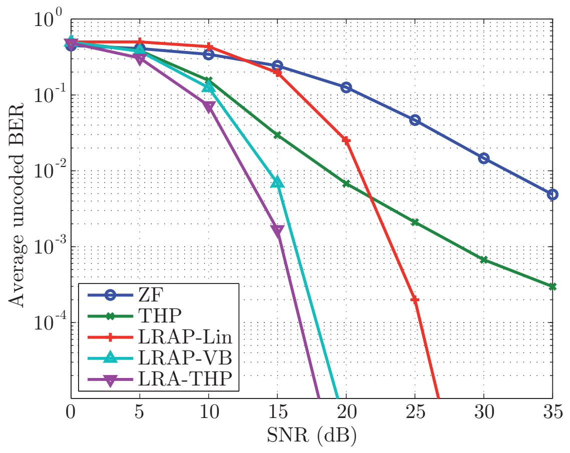


Figure 5.5: Average uncoded bit error rate per subcarrier for 64 × 64 MIMO systems with 4-QAM symbol constellation.

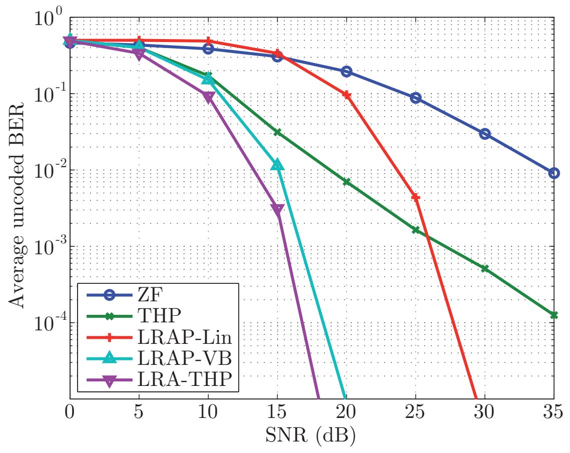


Figure 5.6: Average uncoded bit error rate per subcarrier for 128 × 128 MIMO systems with 4-QAM symbol constellation.

results for smaller systems were presented in [116], [100] where LR-aided techniques also achieved full diversity. Thus, LR has proved to be a decisive technique for improving BER performance also in very large MISO systems.

5.5 Lattice reduction parallelization strategies

In this section the fundamental parallelization possibilities of the LLL algorithm are discussed. Since the LLL algorithm shows a highly sequential behavior, multiple levels of parallelism have to be identified and exploited in order to efficiently parallelize this algorithm. Extensive research of LR techniques has led to important improvements in computational cost, as those achieved thanks to the parallelization of the sequential LLL algorithm. During the design of the parallel LLL algorithm, multiple levels of parallelism were identified.

DOI:10.15774/PPKE.ITK.2015.010

Villard in [117] introduced the concept of the *all swap reduction*, that enables simultaneous basis swaps and served as a basis for future parallel implementations.

In [105] Wetzel introduced the *block reduction* concept. It enables to perform one iteration of the LLL algorithm in parallel by creating non-overlapping sub-groups. This can be regarded as a higher level parallelization possibility. In the following a brief overview of these techniques is given.

5.5.1 The All-Swap lattice reduction algorithm

Basically, LR consists of a succession of swaps between vectors of the basis and some operations to decrease their norms. The order in which the swaps are applied in the LLL algorithm is limiting in a parallel framework. Thus, in [117] first the *any swap reduction* concept was introduced. The LLL algorithm is modified such that there is no restriction in the order in which the swaps are applied. This means that for any k where $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$ is true, basis vectors $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$ are swapped. The order in which the swaps are performed does not affect the final result.

The concept of any swap reduction was further improved and the *all-swap reduction* strategy was introduced. The basic idea is to use several swaps simultaneously at each step of reduction. In order to avoid concurrency issues, the (i) even and (ii) the odd phases are defined. In the odd phase all vectors $\mathbf{b}_k$ and $\mathbf{b}_{k+1}$ are swapped where k is odd and inequality $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$ is fulfilled and in the even phase the same is done for the even k values. At this point $n/2$ processors could work simultaneously.

As a conclusion, the all swap strategy enabled simultaneous basis swaps and served as a good basis for future parallel implementations.

5.5.2 The parallel block reduction concept

In [105] the *parallel block reduction* concept was introduced. Recall, in the LLL algorithm the lattice basis reduction is performed globally at once. However, the main goal of the block reduction is to define m disjoint blocks of size l and perform LLL reduction locally on these blocks. In a lattice basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of size n, $m = \lceil n/l \rceil$ number of blocks of size l are defined. Let $\mathbf{B}_{[\mathbf{k}]} = (\mathbf{b}_{l\cdot(k-1)+1}, \ldots, \mathbf{b}_{l\cdot k})$ denote the k-th lattice basis block, let $\mathbf{B}^*_{[\mathbf{k}]} = (\mathbf{b}^*_{l\cdot(k-1)+1}, \ldots, \mathbf{b}^*_{l\cdot k})$ denote the k-th associated orthogonal lattice basis and the corresponding Gram-Schmidt coefficients are represented by $\mathbf{U}_{[\mathbf{k}]} = U_{(l\cdot(k-1)+1,\ldots,l\cdot k)\times(l\cdot(k-1)+1,\ldots,l\cdot k)}$ an $l \times l$ submatrix of $\mathbf{U}$. After every block is reduced the Gram-Schmidt coefficients are updated and size reduction is performed. Fi-

DOI:10.15774/PPKE.ITK.2015.010

nally, the swap condition is checked at every boundary of the adjacent blocks. In case of a swap the involved blocks have to restart the local LLL reduction.

This approach enables a higher level, coarse grained parallelism, because no frequent synchronization is required. A detailed complexity analysis of the parallel block reduction algorithm was given in [105].

5.6 Parallel lattice reduction algorithms and their mapping to parallel architectures

This section aims to give a description of the newly introduced parallel LR algorithms and concepts by Józsa et al. in [2], [4] and [5]. The results of this section form **Thesis group III**. Three algorithms are presented: (i) the CR-AS-LLL algorithm, (ii) the MB-LLL algorithm and (iii) the CR-MB-LLL algorithm. Mappings to multi-core, many-core architectures and a heterogeneous platform are discussed.

In the CR-AS-LLL algorithm the size reduction delay concept is further refined, thus, an efficient mapping to multi-core and many-core becomes possible. With the improved delay concept the computational complexity is further reduced.

The MB-LLL algorithm allows to split a large matrix in several smaller-sized submatrices where parallel LR is performed in a block-wise manner with the CR-AS-LLL parallel LR algorithm. It is shown how dynamic parallelism can be applied to reduce communication with the CPU.

The CR-MB-LLL further reduces the computational complexity of the MB-LLL algorithm, by relaxing the first LLL condition in the submatrices and with the use of a simplified swap procedure in case of a block boundary swap. A heterogeneous platform is designed in order to achieve better performance.

Finally, the presented algorithms are evaluated and their performance is compared.

5.6.1 The Cost-Reduced All-Swap LLL lattice reduction algorithm

In [118], the concept of delaying the *size reductions* was introduced. In the CR-AS-LLL algorithm, further computational cost is saved by rearranging and delaying the frequently used *size reduction* procedure.

Procedures **SimpleSizeReduce**, **SimpleSwap** and **Swap** are defined in order to give an accurate description of the CR-AS-LLL algorithm.

DOI:10.15774/PPKE.ITK.2015.010

Procedure 1 (SimpleSizeReduce$(\mathbf{B}, k, l)$) *Given a lattice generator matrix* $\mathbf{B}$ *and the associated Gram-Schmidt coefficients matrix* $\mathbf{U}$*, if condition* (5.29) *is not satisfied, i.e.* $|\mu_{k,l}| > \frac{1}{2}$*, the following updates are applied:*

- $\mu = \lceil \mu_{k,l} \rfloor$, $\mu_{k,l} = \mu_{k,l} - \mu$, $\mathbf{b}_k = \mathbf{b}_k - \mu \mathbf{b}_l$.

Procedure 2 (SimpleSwap$(\mathbf{B}, k)$) *Given a lattice generator matrix* $\mathbf{B}$*, the associated orthogonal basis* $\mathbf{B}^*$ *and Gram-Schmidt coefficients matrix* $\mathbf{U}$*, if condition* (5.30) *is not satisfied, or equivalently* $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$*, the following updates are applied:*

- *swap* $\mathbf{b}_k$ *with* $\mathbf{b}_{k-1}$,
- $\mathbf{b}_{k-1}^{*'} = \mathbf{b}_k^* + \mu_{k,k-1}\mathbf{b}_{k-1}^*$, $\mu_{k,k-1}^{'} = (\mathbf{b}_{k-1}^*, \mathbf{b}_{k-1}^{*'})/\|\mathbf{b}_{k-1}^{*'}\|^2$, $\mathbf{b}_k^{*'} = \mathbf{b}_{k-1}^* - \mu_{k,k-1}^{'}\mathbf{b}_{k-1}^{*'}$,
- $\mathbf{b}_{k-1}^* = \mathbf{b}_{k-1}^{*'}$,$\mathbf{b}_k^* = \mathbf{b}_k^{*'}$,$\mu_{k,k-1} = \mu_{k,k-1}^{'}$.

Procedure 3 (Swap$(\mathbf{B}, k)$) *Given a lattice generator matrix* $\mathbf{B}$*, the associated orthogonal basis* $\mathbf{B}^*$ *and Gram-Schmidt coefficients matrix* $\mathbf{U}$*, if condition* (5.30) *is not satisfied, or equivalently* $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$*, the following updates are applied:*

- *perform* **SimpleSwap**(k),
- *swap* $\mu_{k,j}$ *with* $\mu_{k-1,j}$*, for* $1 \le j < k-1$,
- $\begin{pmatrix} \mu_{i,k-1} \\ \mu_{i,k} \end{pmatrix} = \begin{pmatrix} \mu_{i,k-1}\mu_{k,k-1}^{'} + \mu_{i,k}\|\mathbf{b}_k^*\|^2/\|\mathbf{b}_{k-1}^{*'}\|^2 \\ \mu_{i,k-1} - \mu_{i,k}\mu_{k,k-1} \end{pmatrix}$ *for* $k+1 \le i < n$.

In Alg. 10 a detailed description of the CR-AS-LLL algorithm is given. In this case the extent of the parallelism depends on the size of the lattice basis. Note that synchronization is frequently required, however, an efficient distribution of the work among the threads leads to significant decrease in the execution time.

When mapped to GP-GPU the performance of the CR-AS-LLL algorithm depends on the efficiency of the *work distribution* among the available GP-GPU threads and the implementation of the most frequently used operations, such as *dot products*, *size reductions* and *column swaps*. In Alg. 11 the CUDA pseudo-code is presented and Fig. 5.7 presents a possible mapping for the main parts of the CR-AS-LLL algorithm. The kernel is launched with a one dimensional *grid* whose size is determined by the number of lattice basis processed simultaneously. The thread blocks TB(T_x, T_y) launched have a two dimensional configuration, where T_x and T_y denote the number of threads in the x and y dimension. The number of threads T_y is defined based on the size of the original

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 10 The Cost-Reduced All-Swap LLL lattice reduction algorithm

```
1: **Input:** $\mathbf{B}$, $\delta$
2: **Output:** LLL reduced basis
3: Compute $\mathbf{B}^*$ and $\mathbf{U}$ with the Gram-Schmidt algorithm
4: $oddSwap = true$, $evenSwap = true$, $i = 1$
5: while $oddSwap$ or $evenSwap$ do
6:     if $i \mod 2 == 1$ then
7:         $oddSwap = false$, $off = 1$
8:     else
9:         $evenSwap = false$, $off = 0$
10:    end if
11:    for $k = 2 + off$ to $n$ step 2 do        ▷ Embarrassingly parallel for all $k$
12:        Update $\mu_{k,k-1}$
13:        SimpleSizeReduce($k$,$k-1$)        ▷ Only $\mu_{k,k-1}$ is reduced
14:        if $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$ then
15:            SimpleSwap($k$)        ▷ No GS coefficients are updated
16:            if $i \mod 2 == 1$ then
17:                $oddSwap = true$
18:            else
19:                $evenSwap = true$
20:            end if
21:        end if
22:    end for
23:    UpdateGSCoefficients        ▷ Highly parallel
24:    $i = i + 1$
25: end while
26: procedure SimpleSizeReduce($k$,$l$)
27:    if $|\mu_{k,l}| > \frac{1}{2}$ then
28:        $\mu = \lceil \mu_{k,l} \rfloor$, $\mu_{k,l} = \mu_{k,l} - \mu$, $\mathbf{b}_k = \mathbf{b}_k - \mu \cdot \mathbf{b}_l$
29:    end if
30: end procedure
31: procedure SimpleSwap($k$)
32:    Swap $\mathbf{b}_k$ with $\mathbf{b}_{k-1}$
33:    $\mathbf{b}_{k-1}^{*'} = \mathbf{b}_k^* + \mu_{k,k-1}\mathbf{b}_{k-1}^*$
34:    $\mu'_{k,k-1} = (\mathbf{b}_{k-1}^*, \mathbf{b}_{k-1}^{*'})/\|\mathbf{b}_{k-1}^{*'}\|^2$
35:    $\mathbf{b}_k^{*'} = \mathbf{b}_{k-1}^* - \mu'_{k,k-1}\mathbf{b}_{k-1}^{*'}$
36:    $\mathbf{b}_{k-1}^* = \mathbf{b}_{k-1}^{*'}$,$\mathbf{b}_k^* = \mathbf{b}_k^{*'}$,$\mu_{k,k-1} = \mu'_{k,k-1}$
37: end procedure
38: procedure UpdateGSCoefficients
39:    for $i = n - 1 \to 1$ do
40:        for $j = n \to i + 2$ do
41:            $\mu_{j,i} = (\mathbf{b}_j, \mathbf{b}_i^*)/\|\mathbf{b}_i^*\|^2$
42:            SimpleSizeReduce($j$,$i$)
43:        end for
44:    end for
45: end procedure
```

DOI:10.15774/PPKE.ITK.2015.010

Algorithm 11 The pseudocode of the Cost-Reduced All-Swap LLL CUDA kernel - processing of one lattice basis $\mathbf{B}_i$ with a two dimensional thread block configuration $TB(T_x, T_y)$

```
 1: Input: $\mathbf{B}_i$,$\mathbf{B}^*{}_i$,$\mathbf{U}$, $\delta$ and thread identifiers $id_x, id_y$
 2: Output: $\mathbf{B}_i$ as a LLL reduced basis
 3: Definition of shared arrays $buf_1[T_y][T_x], buf_2[T_y][T_x], \mu[T_y]$
 4: Definition of shared variables $odd = true, even = true$ and private variable $off$
 5: Copy the elements above the diagonal from $\mathbf{U}$ to shared array $U_\backslash[n-1]$
 6: while $odd$ or $even$ do                      ▷ $T_x \cdot T_y$ threads are working on the while loop
 7:     $off = (off + 1) \mod 2$
 8:     for $k = id_y * 2 + 1 + off$ to $n$ step $k$ += $T_y * 2$ do
 9:         Call DotProduct($\mathbf{b}^*{}_{k-1}$,$\mathbf{b}^*{}_{k-1}$,$buf_1[id_y][]$), DotProduct($\mathbf{b}_k$,$\mathbf{b}^*{}_{k-1}$,$buf_2[id_y][]$)
10:         Threads with $(id_x == 0)$ set $U_\backslash[k-1] = buf_2[id_y][0]/buf_1[id_y][0]$
11:         if $|U_\backslash[k-1]| > 0.5$ then                              ▷ Check reduction criteria
12:             Threads with $(id_x == 0)$ set $\mu[id_y] = \lceil U_\backslash[k-1] \rfloor$
13:             Call SimpleSizeReduce($\mathbf{b}_k, \mathbf{b}_k, \mathbf{b}_{k-1}, \mu[id_y]$)
14:         end if
15:         Call DotProduct($\mathbf{b}^*{}_k$,$\mathbf{b}^*{}_k$,$buf_2[id_y][]$)
16:         if $buf_2[id_y][0] < (\delta - U_\backslash[k-1]^2) \cdot buf_1[id_y][0]$ then
17:             Call SimpleSwap($\mathbf{b}_k, \mathbf{b}_{k-1}, buf_1[id_y][]$)
18:             Call SimpleSizeReduce($\mathbf{b}^{*\prime}, \mathbf{b}^*_k, \mathbf{b}^*_{(k-1)}, U_\backslash[k-1]$)
19:             Call DotProduct($\mathbf{b}^{*\prime}$,$\mathbf{b}^{*\prime}$,$buf_1[id_y][]$), DotProduct($\mathbf{b}^*_{k-1}$,$\mathbf{b}^{*\prime}$,$buf_2[id_y][]$)
20:             Threads with $(id_x == 0)$ set $U_\backslash[k-1] = buf_2[id_y][0]/buf_1[id_y][0]$ and set $odd$ or $even$ to
    $true$ depending on the $off$ variable
21:             Call SimpleSizeReduce($\mathbf{b}^*_k, \mathbf{b}^*_{(k-1)}, \mathbf{b}^{*\prime}, U_\backslash[k-1]$) and update $\mathbf{b}^*_{(k-1)} = \mathbf{b}^{*\prime}$
22:         end if
23:     end for
24:     Synchronize threads
25: end while
26: Copy the $U_\backslash$ to the diagonal elements of $\mathbf{U}$
27: Update the rest of GS coefficients based on the procedures and methods presented above
28: procedure DotProduct($v_1, v_2, buf[T_x]$)                 ▷ The result is stored in $buf$ at index 0
29:     $buf[id_x] = 0$
30:     for $i = id_x$ to $n$ step $i$ += $T_x$ do $buf[id_x]$ += $v_{1i} \cdot v_{2i}$ end for
31:     for $stride = T_x/2$ to $stride > 0$ step $stride$ >>= 1 do
32:         if $id_x < stride$ then $buf[id_x]$ += $buf[id_y][stride + id_x]$ end if
33:     end for
34: end procedure
35: procedure SimpleSizeReduce($v_1, v_2, v_3, \mu$)
36:     for $i = id_x$ to $n$ step $i$ += $T_x$ do $v_{1i} = v_{2i} - \mu \cdot v_{3i}$ end for
37: end procedure
38: procedure SimpleSwap($v_1, v_2, buf[T_x]$)
39:     for $i = id_x$ to $n$ step $i$ += $T_x$ do
40:         (i.) $buf[id_x] = v_{1i}$, (ii.) $v_{1i} = v_{2i}$, (iii.) $v_{2i} = buf[id_x]$
41:     end for
42: end procedure
```

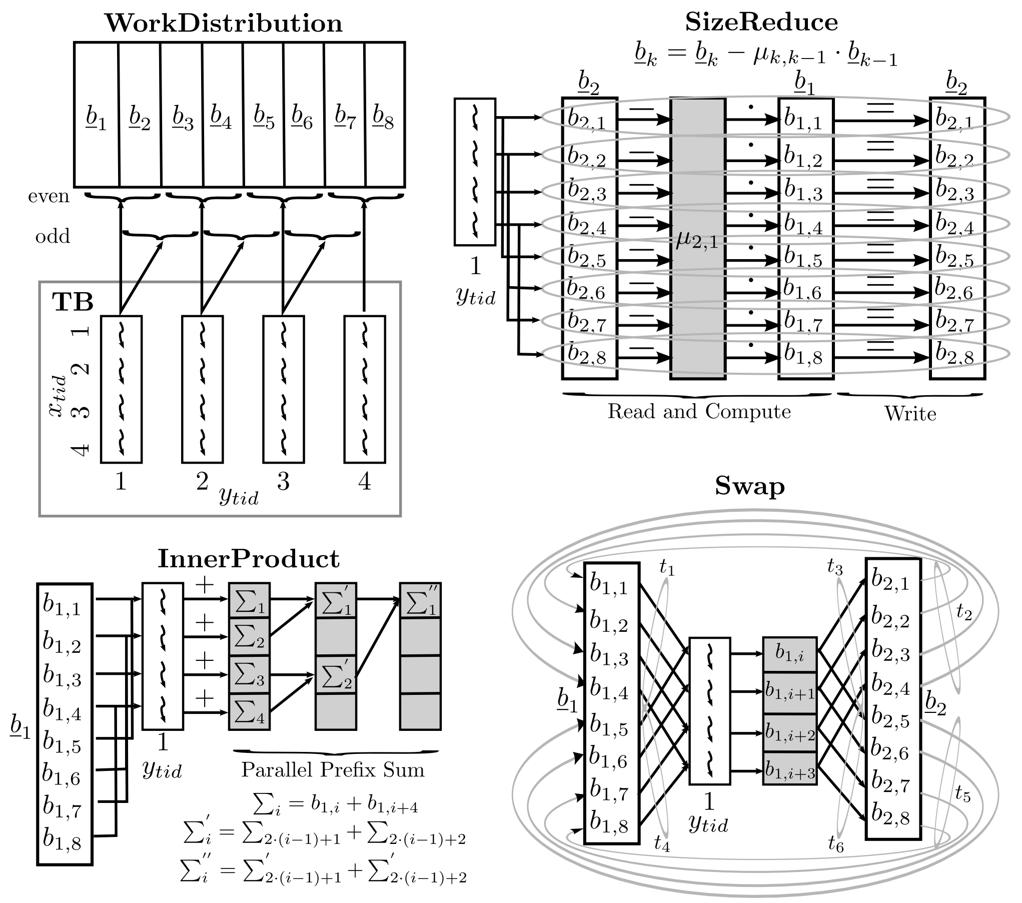

WorkDistribution
even
odd
TB
x_{tid}
y_{tid}
SizeReduce
$\underline{b}_k = \underline{b}_k - \mu_{k,k-1} \cdot \underline{b}_{k-1}$
$\mu_{2,1}$
Read and Compute
Write
InnerProduct
Parallel Prefix Sum
$\sum_i = b_{1,i} + b_{1,i+4}$
$\sum_i' = \sum_{2\cdot(i-1)+1} + \sum_{2\cdot(i-1)+2}$
$\sum_i'' = \sum_{2\cdot(i-1)+1}' + \sum_{2\cdot(i-1)+2}'$
Swap

Figure 5.7: The high-level work distribution among the GP-GPU threads and the mapping of the size reduction, inner product and column swap operations for the Cost-Reduced All-Swap LLL lattice reduction algorithm.

basis, i.e., $T_y = \min(n/2, 32)$. By enabling the usage of $T_x = \min(n, 32)$ threads in the x dimension the threads that belong to the same y dimension will form a *warp*. The maximum of 32 threads in each direction is limited by both the GP-GPU architecture and code optimization. Consequently, the global memory loads and stores issued by the threads of the warp will be coalesced. The y dimension also defines the extent of paralellism. The iteration variable of the for loop is increased in every iteration by $T_y \cdot 2$. In other words, in every phase the threads with the same id_y have to reduce and swap at most $n/(T_y \cdot 2)$ vectors.

The elements of matrices $\mathbf{B}, \mathbf{B}^*, \mathbf{U}$ are stored in the global memory of the GP-GPU. This memory has high latency, but with coalesced access pattern, optimal memory usage can be achieved. Using low latency *shared memory* is also possible, but the size is limited. Because of the limited size it is not possible to load the entire matrices in this low latency memory. Furthermore, the excessive use of shared memory is decreasing the occupancy, resulting in performance degradation. Shared memory is used to store the Gram-Schmidt

DOI:10.15774/PPKE.ITK.2015.010

coefficients $\mu_{i,i-1}$ for $2 \leq i \leq n$ and two shared buffer arrays $buf_1[T_y][T_x]$ and $buf_2[T_y][T_x]$ are allocated in order to efficiently compute the dot products and the vector norms. When computing the inner product the elements of $\mathbf{b}_k$ are read in a coalesced pattern and each thread will sum the corresponding elements in the shared memory buffer. After the sum, the parallel prefix sum pattern is applied to the buffer resulting in the inner product value. In case of the size reduction the corresponding elements are reached in a coalesced pattern and the corresponding $\mu_{k,k-1}$ is read from the shared memory. Because the threads belonging to the same y_{tid} will access the same $\mu_{k,k-1}$ in the shared memory, this will result in a memory broadcast instead of a bank conflict.

Algorithm 12 The OpenMP pseudocode of the Cost-Reduced All-Swap LLL lattice reduction algorithm

```
1: **Input:** $[\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m]$, $[\mathbf{B}_1^*, \mathbf{B}_2^*, \dots, \mathbf{B}_m^*]$, $[\mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_m]$, $\delta$
2: **Output:** $[\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m]$ as LLL reduced basis
3: $maxT \leftarrow$ set the maximum number of available OpenMP threads
4: $simMat \leftarrow$ set the number of matrices processed simultaneously
5: $TPM = maxT/simMat$ ▷ The number of threads for parallel processing one matrix
6: #pragma omp parallel numthreads($simMat$) {
7:   $grp \leftarrow$ set current thread id
8:   $odd = true$, $even = true$, $off = 0$, $i = grp \cdot (m/simMat)$
9:   #pragma omp parallel numthreads($TPM$) shared($odd, even, off$) firstprivate($grp$) {
10:      **while** $(i < (grp + 1) \cdot MPG)$ **do**
11:        **while** $(odd$ or $even)$ **do**
12:          #pragma omp single {
13:            **if** $off == 0$ **then** $odd = false, off = 1$ **else** $even = false, off = 0$ **end if**
14:          }
15:          #pragma omp for reduction(‖:odd,even)
16:          **for** $k = 2 + off$ **to** $n$ **step** 2    ▷ Embarrassingly parallel for all $k$
17:            Update GS coefficient $\mu_{k,k-1}$ and **SimpleSizeReduce**($\mathbf{B_i}$,$k$,$k-1$)
18:            **if** $\|\mathbf{b}_k^*\|^2 < (\delta - \mu_{k,k-1}^2)\|\mathbf{b}_{k-1}^*\|^2$ **then**
19:              Perform **SimpleSwap**($\mathbf{B_i}$,$k$)
20:              **if**$(off == 0)$ **then** $even = true$ **else** $odd = true$ **end if**
21:            **end if**
22:          **end for**
23:        **end while**
24:        #pragma omp barrier
25:        Update all of the GS coefficients of $\mathbf{B_i}$ and perform **SimpleSizeReduce** if necessary    ▷ Highly parallel
26:        #pragma omp single $\{i \leftarrow i + 1, odd = true, even = true\}$
27:      **end while**
28:   }
29: }
```

The OpenMP implementation of the CR-AS-LLL is presented in Alg. 12. Two-level

DOI:10.15774/PPKE.ITK.2015.010

parallelism is implemented based on a nested parallel construct. The outer level parallelism starts the concurrent processing of $simMat$ number of lattice basis and the inner parallel construct is responsible for the parallel LR of a basis with TPM number of threads. As the outer level parallelism is expanded, namely $simMat$ is increased, the number of slave threads for the parallel CR-AS-LLL algorithm that could be forked by the master threads are decreased. In this case the very limited number of CPU threads restrict the exploitation of several levels of parallelism.

Another major difference between the CUDA and OpenMP mapping lies in the implementation of the size reductions, dot products and swaps. In the CUDA implementation, because of the two dimensional TB configuration, T_x number of threads are working in every procedure. For example, in the dot product calculation every thread has to do n/T_x number of multiplications and the result of the multiplication is added to the shared memory buffer. When the execution of all threads finishes, a parallel prefix sum is applied to the buffer in order to compute the dot product. In case of the OpenMP implementation, only one thread is working in the computation of a dot product since the number of threads are limited.

5.6.2 The Modified-Block LLL lattice reduction algorithm

The problem division to several sub-problems that can be executed concurrently can be regarded as one level of parallelism. In addition, if a sub-problem could benefit from a multi-threaded environment it can be regarded as a second level of parallelism. Previous parallel LR implementations, such as the ones presented in [118], [119], [120] have focused only on multi-core architectures. The main drawback of the low number of threads offered by modern CPUs (compared to GP-GPUs) is that low level parallelism can not be efficiently exploited. During an algorithm design, low level parallelism is usually omitted and the levels of parallelism are also restricted. In case of GP-GPUs, the high number of CUDA cores makes the parallel execution of a high number of threads possible leading to significant performance improvements.

The MB-LLL algorithm is designed to take advantage of a highly multi-threaded environment. The MB-LLL algorithm splits the original basis into several sub-problems of lower dimension and performs parallel LLL reduction on them. Because the LLL reduction of the subgroups and the boundaries check can be done independently, no frequent synchronization is required. Thus, coarse grained parallelism is achieved by creating the sub-problems. A detailed description of the MB-LLL algorithm is given in

Alg. 13.

Algorithm 13 The Modified-Block LLL lattice reduction algorithm

```
1: Input: $\mathbf{B}$, $\delta$, block-size $l$
2: Output: LLL reduced basis
3: Compute $\mathbf{B}^*$ and $\mathbf{U}$ with the Gram-Schmidt algorithm
4: $m = \lceil n/l \rceil$                                   ▷ $m$ denotes the number of blocks
5: for $k = 1 \to m$ do                                      ▷ Create the subgroups $\mathbf{B}_{[\mathbf{k}]}$,$\mathbf{B}^*_{[\mathbf{k}]}$,$\mathbf{U}_{[\mathbf{k}]}$
6:     $\mathbf{B}_{[\mathbf{k}]} = (\mathbf{b}_{l\cdot(k-1)+1}, \ldots, \mathbf{b}_{l\cdot k})$
7:     $\mathbf{B}^*_{[\mathbf{k}]} = (\mathbf{b}^*_{l\cdot(k-1)+1}, \ldots, \mathbf{b}^*_{l\cdot k})$
8:     $\mathbf{U}_{[\mathbf{k}]} = U_{(l\cdot(k-1)+1,\ldots,l\cdot k)\times(l\cdot(k-1)+1,\ldots,l\cdot k)}$     ▷ $\mathbf{U}_{[\mathbf{k}]}$ is the $l \times l$ submatrix of $\mathbf{U}$
9:     $echange[k] = true$
10: end for
11: while $\exists k$ such that $exchange[k]$ is true do
12:     for $k = 1 \to m$ do                                 ▷ Embarrassingly parallel
13:         if $exchange[k]$ is true then
14:             LLL($\mathbf{B}_{[\mathbf{k}]}$,$\mathbf{B}^*_{[\mathbf{k}]}$,$\mathbf{U}_{[\mathbf{k}]}$)     ▷ Call CRAS-LLL without performing GS orthogonalization
15:             $group[k] = true$
16:         end if
17:     end for
18:     for $k = 1 \to m-1$ do ▷ Checking the boundaries of the groups, embarrassingly parallel
19:         if $group[k]$ or $group[k+1]$ is true then
20:             Update $\mu_{k\cdot l,k\cdot l-1}$
21:             SimpleSizeReduce($k \cdot l$,$k \cdot l - 1$)
22:             if $\|\mathbf{b}^*_{k\cdot l+1}\|^2 < (\delta - \mu^2_{k\cdot l,k\cdot l-1})\|\mathbf{b}^*_{k\cdot l}\|^2$ then
23:                 for $j = k \cdot l - 1 \to k \cdot l - l + 1$ do     ▷ Prepare the GS coefficients outside the groups
24:                     $\mu_{k\cdot l+1,j} = (\mathbf{b}_{k\cdot l+1}, \mathbf{b}^*_j)/\|\mathbf{b}^*_j\|^2$
25:                 end for
26:                 for $i = k \cdot l + 2 \to k \cdot l + l$ do
27:                     $\mu_{i,k\cdot l} = (\mathbf{b}_i, \mathbf{b}^*_{k\cdot l})/\|\mathbf{b}^*_{k\cdot l}\|^2$
28:                 end for
29:                 Swap($k \cdot l + 1$)          ▷ Update only the GS coef. inside the groups
30:                 $echange[k] = true$, $echange[k+1] = true$
31:             end if
32:         end if
33:     end for
34: end while
35: UpdateGSCoefficients      ▷ Only update the GS coefficients outside the groups
```

The GP-GPU mapping of the MB-LLL algorithm is similar to the one presented in case of the CR-AS-LLL algorithm in Sec. 5.6.1, because the used procedures are performed with a two dimensional TB configuration even in the case of a boundary check. The main difference is that DP [40] enables the launch of new kernels from the GP-GPU without returning the program flow control to the CPU.

The schematic of the kernels scheduling implementing DP is shown in Fig. 5.8. The

DOI:10.15774/PPKE.ITK.2015.010

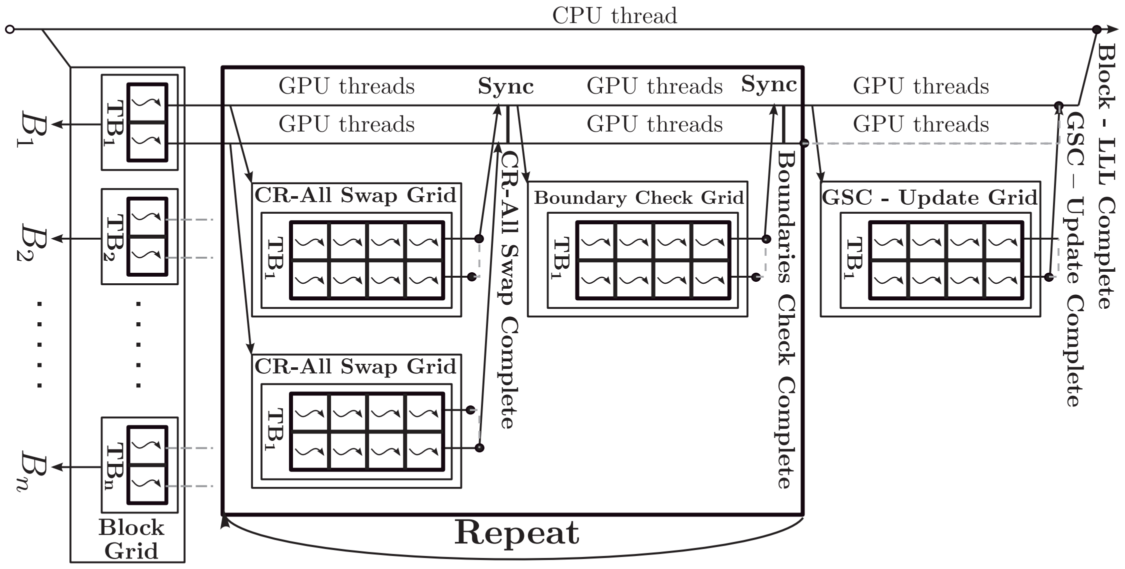


Figure 5.8: Kernels scheduling on a dynamic parallelism enabled GP-GPU for the Modified-Block LLL lattice reduction algorithm.

CPU launches the *Block-LLL* kernel. The size of the grid is equal to the number of matrices that are simultaneously processed and the number of threads in one TB is equal to the number of submatrices. In this case, every thread has to prepare the data for the corresponding submatrices and launch the *CR-AS-LLL* kernel. The *CR-AS-LLL* kernel has to be relaunched if the LLL conditions were broken by a boundary swap, which can be solved by tracking state variables placed in the global memory. When the reduction of the submatrices is over, the *Boundaries Check* kernel is launched. Since the operations performed in this section are dot products and column swaps the thread configuration of the TB is the same as in case of the *CR-AS-LLL* kernel. The *CR-AS-LLL* and *Boundary Check* kernels are repeated until there are no swaps on the boundaries. Because one matrix is assigned to one TB in the parent *Block* kernel, the processing of the different matrices can be done simultaneously despite the variable number of iterations. Finally, the Gram-Schmidt coefficients outside the blocks are updated with the *GSC-Update* kernel and the size-reduction is performed wherever is needed. Note, synchronization of the threads is required only after finishing the LR of the submatrices and after the boundary checks.

5.6.3 The Cost-Reduced Modified-Block LLL lattice reduction algorithm

As stated in the previous sections the MB-LLL algorithm allows to split a large matrix in several smaller submatrices where parallel LR is performed in a block-wise manner

DOI:10.15774/PPKE.ITK.2015.010

with the parallel LR algorithm CR-AS-LLL. Once the LR of the submatrices is finished, the boundaries between adjacent submatrices are checked and finally the Gram-Schmidt coefficients outside the initial groups are updated. The main condition is to keep every submatrix as an LLL-reduced matrix throughout the processing.

The CR-MB-LLL algorithm further reduces the computational complexity of the MB-LLL algorithm. In the MB-LLL algorithm, the submatrices affected by a boundary swap have to be LLL reduced and the Gram-Schmidt coefficients have to be updated. Moreover, in order to fulfill the LLL conditions in the submatrices affected by a boundary swap, the **Swap** procedure has to be performed.

The complexity reduction is achieved by eliminating the GS coefficients update in the submatrices after the execution of the CR-AS-LLL, and with the usage of the **SimpleSwap** procedure instead of **Swap** in case of a boundary swap. Since the GS coefficients are updated only when the ordering condition (5.30) is met for every column vector, the processing time can be considerably reduced.

Algorithm 14 The mapping of the Cost-Reduced Modified-Block LLL lattice reduction algorithm to a heterogeneous platform

```
1: Input: $[\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m]$, $\delta$, block-size $l$, $T$ number of OpenMP threads
2: Output: $[\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m]$ as LLL reduced basis
3: #pragma omp parallel {
4: $mpt = m/T$    ▷ The number of matrices that have to processed by one thread
5: $bpm = n/l$    ▷ The number of blocks per matrix
6: Assign a CUDA $stream_{id}$ to the current CPU thread with identifier $id$
7: Define arrays $matIndD[mpt]$ on the GP-GPU and $matIndH[mpt]$ on the host    ▷ The indexes of the unprocessed matrices are stored in these arrays
8: for $i = 0$ to $mpt$ step $i{+}{+}$ do $matIndH[i] = id \cdot mpt + i$ end for
9: Define arrays $boundaryExchD[mpt \cdot bpm]$ and $boundaryExchH[mpt \cdot bpm]$
10: while $mpt > 0$ do
11:     Asynchronously copy $matIndH$ to $matIndD$ on $stream_{id}$
12:     Launch $CR - AS - LLL$ kernel on $stream_{id}$ with grid size $grid_{lll} = mpt \cdot bpm$ and $TB(T_x, T_y)$    ▷ The CR-AS-LLL is performed on the submatrices, without updating the GS coefficients
13:     Launch the $BoundaryCheck$ kernel on $stream_{id}$ with grid size $grid_{bc} = mpt \cdot (bpm - 1)$ and $TB(B_x, B_y)$    ▷ The LLL conditions (5.29) and (5.30) are checked on the boundary of two adjacent submatrices. In case if the conditions are not met the SimpleSwap is executed instead of the Swap procedure.
14:     Asynchronously copy $boundaryExchD$ to $boundaryExchH$ on stream $id$
15:     Synchronize CPU thread with $stream_{id}$
16:     if There was no boundary exchange for one matrix then
17:         Remove the matrix index from $matIndH$ and $mpt \leftarrow mpt - 1$    ▷ The CPU threads have to process the result of the boundary exchange
18:     end if
19: end while
20: Launch the $GSC - Update$ kernel on $stream_{id}$    ▷ In this kernel all the GS coefficients are updated and size reduction is performed where necessary.
21: }
```

The mapping of the CR-MB-LLL algorithm to a heterogeneous platform is presented

DOI:10.15774/PPKE.ITK.2015.010

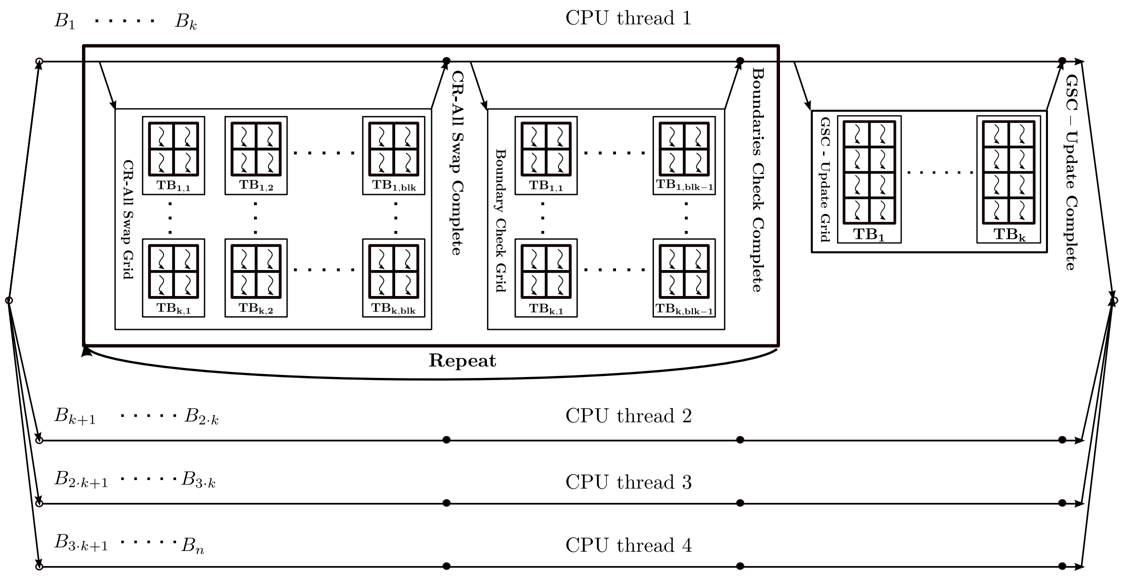


Figure 5.9: Kernels scheduling on the heterogeneous platform for the Cost-Reduced Modified-Block LLL lattice reduction algorithm.

in Alg. 14. The schematic of the heterogeneous platform is shown in Fig. 5.9. The CPU threads launch (i) the *CR-AS-LLL* kernels in order to LLL reduce the submatrices, (ii) launch the *Boundary Check* kernels for checking the LLL conditions at the boundaries of the sub-groups and (iii) launch the *GSC-Update* kernel to update the Gram-Schmidt coefficients and to perform the size reductions wherever it is required. Furthermore, the control logic of the dynamic scheduling is implemented by the CPU threads.

A different CUDA stream is assigned for every CPU thread, making the concurrent kernel execution possible and reducing the idle time of the CUDA cores. Before launching the *CR-AS-LLL* and *Boundary Check* kernels, the CPU thread updates the *matIndD* array placed in the GP-GPU's global memory to specify which matrices need further processing. The size of the grid is dynamically adjusted according to the number of non-processed matrices in every iteration. After the *Boundary Check* kernel is executed, the *boundaryExchH* is updated on the host. Afterwards, the CPU thread checks if the LR of any matrix is finished. If LLL reduced matrices are found the *matIndH* is updated. Consequently, the size of the grids assigned to *CR-AS-LLL* and *Boundary Check* kernels is decreased. The *GSC-Update* kernel starts after all the matrices assigned to one CPU thread are completely processed.

5.6.4 Evaluation results

In this section the performance comparison of the proposed parallel LR algorithms is presented. The computations were done in single-precision floating point arithmetic

DOI:10.15774/PPKE.ITK.2015.010

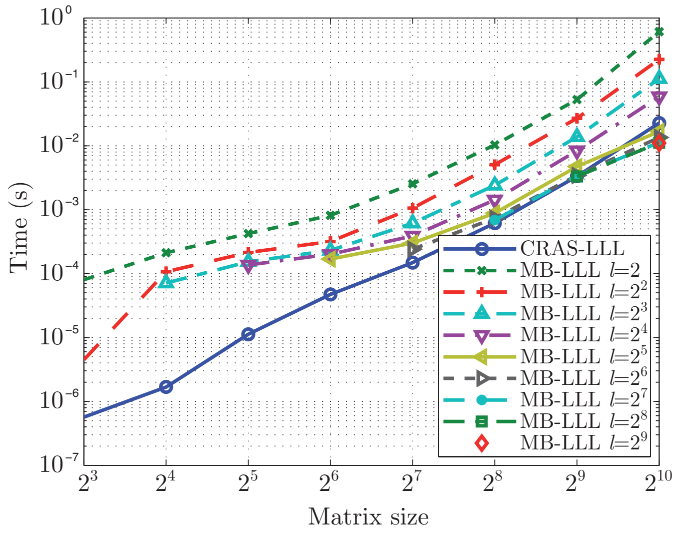


Figure 5.10: Computational time of Cost-Reduced All-Swap LLL and Modified-Block LLL lattice reduction algorithms with different block sizes ranging from $2 - 2^9$ for square matrices of dimensions $2^3 - 2^{10}$.

and parameter $\delta = 0.75$ was used for the LLL condition (5.30). Block-Toeplitz matrices have been considered to evaluate the performance of different implementations. The use of Block-Toeplitz type matrices is motivated by their extensive use in wireless communications [100]. Throughout the simulations the equivalent real-valued lattice basis of size $N = M$ is used. Thus, the different matrix sizes correspond to systems with $n = m$ ranging from 4 to 512.

Figure 5.10 shows the computational time of the CR-AS-LLL and the MB-LLL algorithms launched on a Tesla K20 GP-GPU. In case of MB-LLL different block-size configurations are evaluated. It is observed when comparing CR-AS-LLL with MB-LLL that the block concept used in MB-LLL allows to reduce the computational time for systems with $N \geqslant 2^8$. It is interesting to see that starting from $2^6 - 2^7$ the slope of the MB-LLL curves is slightly increased. The reasons are twofold: (i) the size of the matrix and the coalesced memory access makes possible to fully exploit the memory bandwidth, and (ii) the number of blocks that are not processed in the distinct iterations is getting higher, resulting in the decrease of the processing time.

Figure 5.11 shows the computational times of the MB-LLL algorithm based on three different architectures for different matrix dimensions, where l denotes the size of the processed blocks. The performance measurements were evaluated with all the possible block sizes and the best configuration is shown. The architectures used for the computational time measurements are the Tesla K20 (with DP capability) and an Intel Core i7-3820 processor. The heterogeneous platform clearly outperforms the solutions based

DOI:10.15774/PPKE.ITK.2015.010

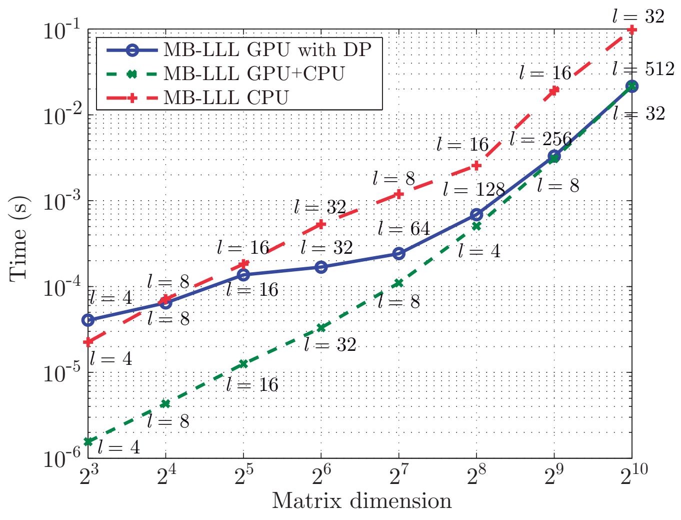


Figure 5.11: Computational time of the Modified-Block LLL lattice reduction algorithm with optimal block size l on different architectures.

Model	Architecture	Die Count	Memory (MB)	SM count	CUDA cores	Shaders clock rate (MHz)	GFLOP/s (FMA)	TDP (W)	Price
GeForce GTX 690	2xGK104	2	2 × 2048	2 × 8	2 × 1536	915	2 × 2810	300	∼ \$1100
Tesla K20	GK110	1	5120	13	2496	705	3519	225	∼ \$3000
Tesla C2075	GF100	1	6144	14	448	1150	1030	238	∼ \$2100

Table 5.2: The comparison of the GTX690, K20 and C2075 GP-GPU architectures.

on DP in the case of small matrices and the CPU for all the cases. The processing times show similar performance for large matrices when the GP-GPU is involved. This gap is caused by the overhead required when launching kernels from kernels with DP and the limited overlapping execution of kernels on different streams. The conclusion is that the data transfer between CPU and GP-GPU required by the heterogeneous system is less time consuming than the overhead of the kernel launch with DP and the limitation of the concurrent execution of kernels on different streams.

The average computational time of the proposed CR-AS-LLL algorithm with the following GP-GPU configurations: (i) 1 × K20, (ii) 2 × K20, (iii) 2 × C2075s and (iv) 1 × GTX690 are compared in Figs. 5.12 and 5.13. A comparison of the hardware components of these GP-GPUs is available in Table 5.2. The GTX690 has the highest number of CUDA cores and achievable FLOP/s. In the case of the C2075, the clock rate of the cores is slightly higher, however, the number of CUDA cores is significantly lower. The multi GP-GPU configurations are used to balance the die number of the K20 and C2075 GP-GPUs with the GTX690 GP-GPU. The average computational time was computed by averaging the processing time of 8000 lattice basis, thus, the same number of thread blocks were defined for the grid. The best results are achieved by the 2 ×

DOI:10.15774/PPKE.ITK.2015.010

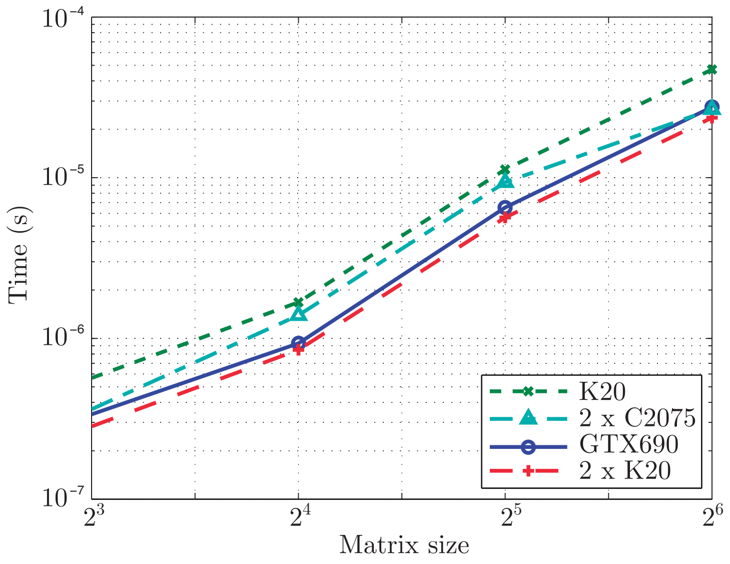


Figure 5.12: Computational time of Cost-Reduced All-Swap LLL lattice reduction algorithm for matrix dimensions $2^3 - 2^6$ on 1 and 2 × Tesla K20, GeForce GTX690 and 2 × Tesla C2075 GP-GPU configurations.

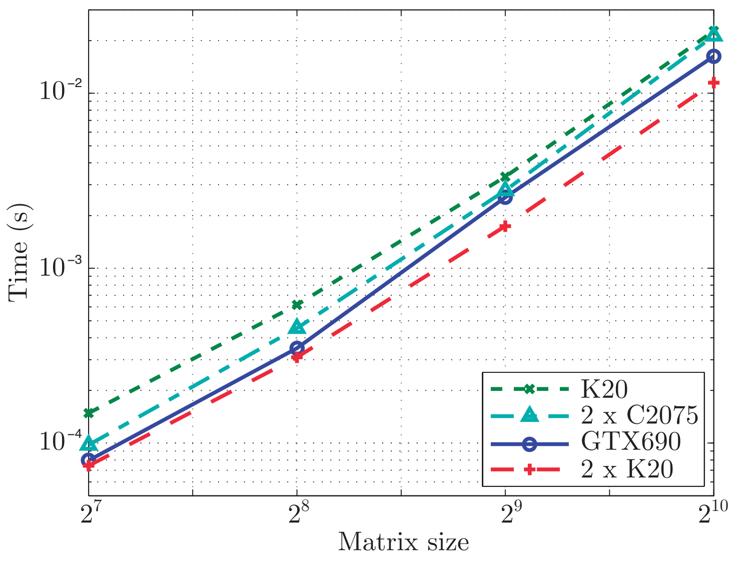


Figure 5.13: Computational time of Cost-Reduced All-Swap LLL lattice reduction algorithm for matrix dimensions $2^7 - 2^{10}$ on 1 and 2 × Tesla K20, GeForce GTX690 and 2 × Tesla C2075 GP-GPU configurations.

DOI:10.15774/PPKE.ITK.2015.010

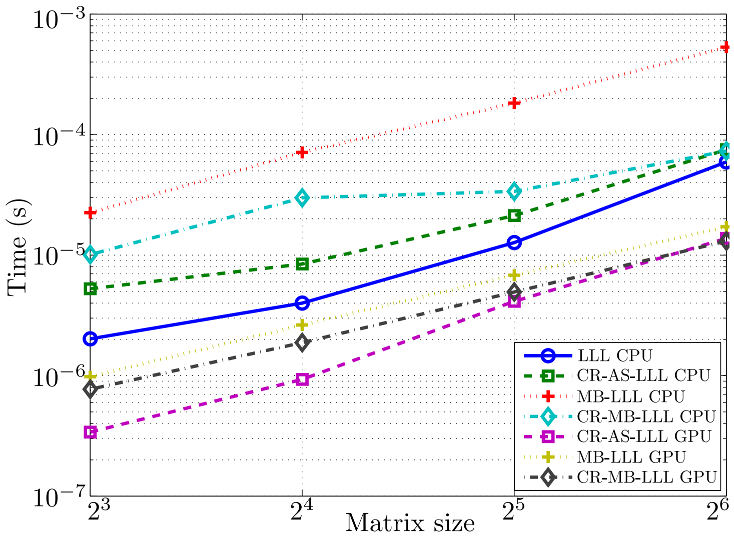


Figure 5.14: Computational time of the LLL, Cost-Reduced All-Swap LLL, Modified-Block LLL and Cost-Reduced Modified-Block LLL algorithms for matrix dimensions $2^3 - 2^6$.

K20 GP-GPU configuration, however, the GTX690 performs better than 1 $\times$ K20. The result achieved with the 2 $\times$ C2075 outperforms the 1 $\times$ K20. This is a surprising result since the FLOP/s achieved by the 2 $\times$ C2075 GP-GPUs are significantly lower compared to the K20 GP-GPU. The reason is the different usage of the L1 cache in the Kepler architecture.

Figures 5.14 and 5.15 compare the average computational time of the LLL, CR-AS-LLL, MB-LLL and CR-MB-LLL algorithms for different matrix dimensions. The algorithms were evaluated on the Intel Core i7-3820 CPU and the GeForce GTX 690 GP-GPU. In [5], it was shown that the GTX 690 has a better performance than the K20 GP-GPU, thus, the DP performance is omitted. The summary of the computational time comparison in case of the GP-GPU is given as follows: (i) the computational time of the CR-MB-LLL is $25 - 40\%$ lower in case of small and medium-sized matrices compared to the MB-LLL algorithm and the performance is similar in case of larger matrices, (ii) the CR-AS-LLL performs better than the CR-MB-LLL in case of small matrices, however, for large matrices the block concept implemented in the CR-MB-LLL achieves 30% speed-up compared to the CR-AS-LLL and (iii) the LR implemented on the GP-GPU architecture outperforms the CPU implementations for every matrix dimension with speed-ups ranging from 6 to 15.

The summary of the computational time comparison in case of the CPU is given as follows: (i) the CR-MB-LLL always outperforms the MB-LLL algorithm with speed-ups ranging from 2 to 7, (ii) the CR-AS-LLL algorithm performs better than the MB-LLL

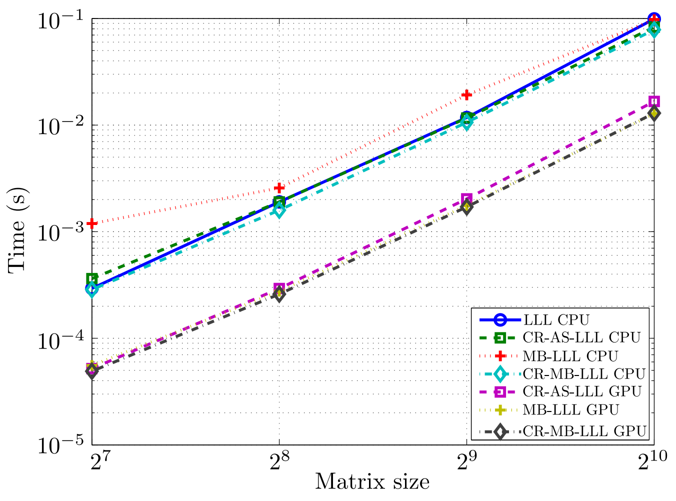


Figure 5.15: Computational time of the LLL, Cost-Reduced All-Swap LLL, Modified-Block LLL and Cost-Reduced Modified-Block LLL algorithms for matrix dimensions $2^7 - 2^{10}$.

and CR-MB-LLL for matrices with low dimensions ($2^3 - 2^6$), (iii) the computational time of the CR-MB-LLL is $10 - 20\%$ lower in case of larger matrices compared to the CR-AS-LLL and (iv) for smaller matrices the sequential LLL algorithm performed on every thread of the CPU is better than the parallel algorithms. However, for higher dimensions, despite the overhead introduced by the cache coherency, thread management and synchronization, the parallel CR-AS-LLL and CR-MB-LLL algorithms achieve a smaller computational time.

A surprising result is while the CR-MB-LLL achieves a significant speed-up compared to the MB-LLL for the CPU architecture, the same is not true for GP-GPU architecture. This is due to several reasons. The computational complexity reductions for the CR-MB-LLL affect only the *CR-AS-LLL* and *Boundary Check* kernels. However, in case of large matrices the *GSC-Update* kernel is taking the major part of the processing time. This kernel has to access the global memory frequently and these accesses have a high latency. In case of the CPU, this problem is alleviated by the high speed memory access and the large amount of available cache.

The performance of LR mostly depends on the precision of the computation, the size and type of the basis matrix and the architecture used. In Table 5.3 performance of existing implementations are presented. Previous research mostly focused on small matrices. In [119] performance measures for higher dimension matrices are presented as well, however, the total runtime of the algorithm is not specified. The parallelization and efficient implementation of the parallel algorithms allow to reduce the computational

DOI:10.15774/PPKE.ITK.2015.010

Ref	Algorithm	Architecture	4×4	8×8	64×64	1024×1024
[32]	Clarkson's Algorithm	Virtex-II-Pro FPGA	4.2×10^{-6}	x	x	x
[33]	Complex LLL	Virtex-5 FPGA	0.79×10^{-6}	x	x	x
[121]	Brun's Algorithm	ASIC 250 nM	0.07×10^{-6}	x	x	x
[122]	Reverse Siegel LLL	Virtex-4 FPGA	0.18×10^{-6}	x	x	x
[122]	Reverse Siegel LLL	ASIC 130 nM	0.04×10^{-6}	x	x	x
[120]	SB-LLL	ADRES	0.17×10^{-6}	x	x	x
This work	CR-AS-LLL	GTX690 GP-GPU	x	0.33×10^{-6}	1.37×10^{-5}	1.67×10^{-2}
This work	CR-MB-LLL	GTX690 GP-GPU + Intel i7-3820 CPU	x	0.77×10^{-6}	1.30×10^{-5}	1.28×10^{-2}

Table 5.3: Performance comparison of the Cost-Reduced All-Swap LLL and the Cost-Reduced Modified-Block LLL algorithms with existing lattice reduction implementations.

time compared to other schemes in the literature [122], [32].

5.7 Conclusion

In this chapter the importance of LR and its applicability to MIMO systems was discussed. After a brief overview of the fundamental definitions, the most important lattice reduction algorithms were discussed. The performance of two polynomial time lattice reduction algorithms the LLL and Seysen's algorithm were evaluated in terms of condition number and orthogonality defect. It was shown that improving the lattice basis, by constructing a shorter and more orthogonal basis, significant BER performance improvement is achieved in detection and precoding. It was shown that full diversity order is achieved even for linear detection when lattice reduction is applied. Different precoding techniques that make use of lattice reduction have also been evaluated for a very large multi-user MISO system. Lattice reduction aided precoding techniques were shown to obtain a significantly higher performance in terms of BER with respect to non LRAP techniques.

Because the LLL algorithm is widely used in several fields, the goal was to design parallel LR algorithms that could benefit from highly parallel architectures such as GP-GPUs and can exploit the possibilities of heterogeneous platforms that consist in the cooperation of modern CPUs and GP-GPUs. Efficient algorithm design and implementations of the parallel LLL algorithm for many-core architectures were presented.

First, the CR-AS-LLL algorithm was introduced, which is built on the all-swap concept. The delay of size reductions was further improved, thus, the complexity of the algorithm was reduced by omitting unnecessary size reduction operations.

The high number of threads available in the GP-GPU enabled the exploitation of several levels of parallelism. As a result, the combination of the parallel block concept with

DOI:10.15774/PPKE.ITK.2015.010

the parallel LLL processing of the blocks, implemented by the CR-AS-LLL algorithm, resulted in the MB-LLL algorithm. The MB-LLL slightly outperformed the CR-AS-LLL algorithm for large matrices. The MB-LLL mapping exploited the dynamic parallelism capability of the GP-GPU.

Finally, the CR-MB-LLL was discussed. The idea behind the CR-MB-LLL algorithm is the relaxation of the first LLL condition for the submatrices, resulting in the delay of the Gram-Schmidt coefficients update when executing LR, and a simplified, less costly swap procedure when performing the boundary checks. The CR-MB-LLL algorithm has been evaluated on several architectures: a multi-core architecture, a GP-GPU with dynamic parallelism capability and a heterogeneous platform based on a CPU and GP-GPU. Results show that mapping the CR-MB-LLL algorithm on the heterogeneous architecture reduces the computational time by 30% compared to the CR-AS-LLL in case of large matrices, whereas implementations involving GP-GPUs achieve speed-up factors from $6 - 15$ compared to the multi-core CPU architecture. The MB-LLL algorithm achieves speed-up factors ranging from $5 - 25$ when launched on the proposed heterogeneous platform compared to the DP-based GP-GPU implementation for matrix dimensions $2^3 - 2^6$.

It was shown that the efficiency of the CR-MB-LLL is significantly affected by the architectures used. The CR-MB-LLL is $1.5 - 7$ times faster compared to the MB-LLL algorithm when launched on multi-core CPU architecture, however the CR-MB-LLL is at most 1.4 times faster compared to the MB-LLL when launched on the GP-GPU architecture. This is mainly because the computational complexity reductions introduced in the CR-MB-LLL algorithm affect the *CR-AS-LLL* and *Boundary Check* kernels. However, in case of large matrices the *GSC-Update* kernel is taking the major part of the processing time with frequent accesses to the global memory of the GP-GPU. In case of the CPU, the memory access has a lower latency and the available cache for CPU is significantly bigger, causing different speed-ups of the same algorithm on the different architectures.

The proposed algorithms were compared with implementations available in the literature and the average computational times are significantly lower compared to previous implementations. Thus, the goal of efficient algorithm design and implementation was achieved.

DOI:10.15774/PPKE.ITK.2015.010

Chapter 6

Theses of the Dissertation

This chapter gives a concise summary of the main scientific contributions of this dissertation as well as the methods and tools used, and briefly discusses the applicability of the results.

6.1 Methods and tools

The goal of my research was to solve computationally demanding signal processing problems in the field of wireless communications with modern MPAs, such as GP-GPUs, and multi-core CPUs. The main challenge was to identify and develop the *mathematical and algorithmic transformations* of the sequential, high-complexity problems in such a way that an efficient mapping to these parallel architectures became possible.

In the first part of my thesis I consider the optimal hard-output *ML detection* in MIMO systems. The complexity of the ML detector increases exponentially with the number of antennas and the modulation order. In order to significantly reduce the complexity, the *SD algorithm* was proposed in [69] and applied in a decoding context in [70]. The fundamental aim of the SD algorithm is to restrict the search to lattice points that lie within a certain sphere around a given received symbol vector. Reducing the search space will not affect the detection quality, because the closest lattice point inside the sphere will also be the closest lattice point for the whole lattice.

During detection the optimum search path for the symbol vectors is different. Since different parts of the search tree are explored by the detection algorithm, a variable processing time is expected. In order to moderate the effects of the variable complexity (i) *a column norm based ordering method* shown in [64] and (ii) *a dynamic computing load distribution* strategy were applied. Specifying the order of symbol detection, based

DOI:10.15774/PPKE.ITK.2015.010

on metrics involving the channel matrix, was shown to lead to less computations. The probability of choosing the right search path on the top levels of the tree can be increased by first detecting symbols with higher post-detection SNR or SINR. Consequently, a non-optimal symbol detection on a lower level does not lead to a major step-back on the tree.

The variable processing time of the symbol vectors leads to an imbalance in the execution time of the thread blocks of a kernel. Until the execution of the thread blocks is not finished the GP-GPU resources allocated to a kernel will not be freed. The long-time resource allocation prevents the overlapping execution of multiple kernels on different streams. With a dynamic load balancing method the tail effect is negligible, thus, the overlapping execution of multiple kernels becomes possible and the goal of alleviating the variable processing time is achieved.

In the second part of my thesis the focus is on a powerful preprocessing tool, namely the *LR method* [100]. Lattice reduction aims to find a "better" basis whose vectors are more orthogonal and shorter, in the sense of Euclidean norm, than the original ones. Lattice reduction improves the condition number, the orthogonality defect and the Seysen measure. Several LR algorithms exist in the literature that differ in computational complexity and achieved performance. However, the most extensively used polynomial-time algorithm is the *Lenstra-Lenstra-Lovász (LLL) algorithm* introduced in [101]. Because of its wide applicability and several favorable properties, my research focused on improving this method and making it suitable for MPAs.

In [14] it was shown that the performance of *linear and non-linear detectors* can be improved when used in conjunction with LR techniques and full diversity order is achieved with the reduced basis. Since many detection schemes heavily rely on the usage of the channel matrix, it is straightforward to regard the channel matrix as a lattice generator matrix.

In MISO systems the multi-user interference must be canceled at the transmitter, this method is referred to as *precoding*. According to [10] linear methods, such as Zero-Forcing precoding, and non-linear methods, such as Tomlinson-Harashima precoding and vector perturbation techniques perform better if the channel matrix is not badly conditioned. Moreover, full diversity is achieved even for very large systems.

The tools used to solve the above mentioned computationally challenging signal processing tasks were modern *multi-core CPUs*, such as Intel Core i7-3820, Intel Xeon X5680, Intel Xeon E5-2650 v3, and *massively parallel architectures*, such as NVIDIA GeForce GTX 690 and NVIDIA Tesla C2075 and K20 GP-GPUs. A number of parallel program-

DOI:10.15774/PPKE.ITK.2015.010

ming models were employed to support the hierarchical parallelism present in modern computer systems. For *coarse-grained shared memory parallelism* I used simultaneous multithreading using *OpenMP*. For *fine-grained parallelism* Single Instruction Multiple Threads (SIMT) was implemented using *CUDA*.

6.2 New scientific results

Thesis group I. Design of new parallel Sphere Detector algorithms achieving hard-output true-ML performance and their efficient mapping to multi-core and many-core architectures.

(Related articles [1], [3].)

Thesis I.a

I proposed a new Parallel Sphere Detector (PSD) algorithm to achieve true-ML bit error rate performance in hard-output MIMO detection. The high degree of parallelism of the PSD algorithm is based on a novel hybrid tree traversal where depth-first search and breadth-first search methods are efficiently combined, furthermore, at each intermediate stage, path metric based parallel sorting networks are employed to achieve a faster convergence. I showed that the PSD algorithm achieves an efficient work distribution in a highly multi-threaded environment reducing the number of visited tree nodes by a single thread with $88\% - 96\%$*, and the speed-up factor of the detection throughput of the PSD algorithm in* 4×4 *MIMO systems is* $2 - 50$ *times higher for different signal-to-noise ratios compared to the sequential case.*

The real-valued MIMO system model is described as

$$\mathbf{y} = \mathbf{H}\mathbf{s}_t + \mathbf{v}$$

where $\mathbf{y} \in \mathbb{R}^M$ is the received symbol vector, $\mathbf{v} \in \mathbb{R}^M$ is the additive channel noise, $\mathbf{s}_t \in \Omega^N$ is the transmitted symbol vector, Ω is the symbol set, and the superposition of the transmitted symbols is modeled by the channel matrix $\mathbf{H} \in \mathbb{R}^{M \times N}$. The optimal

DOI:10.15774/PPKE.ITK.2015.010

hard-output ML detector is defined as

$$\hat{\mathbf{s}}_{ML} = \arg\min_{\mathbf{s}\in\Omega^N} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2.$$

The ML estimate of the transmitted symbol vector is found by solving an integer least-squares problem which is analogous to finding the closest lattice point of lattice $\mathbf{\Lambda} = \{\mathbf{H}\mathbf{s}|\mathbf{s} \in \Omega^N\}$ to a given point $\mathbf{y}$ as discussed in [62], [68].

With the unconstrained least-squares solution $\hat{\mathbf{s}} = \mathbf{H}^\dagger\mathbf{y}$, where $\mathbf{H}^\dagger$ denotes the Moore–Penrose pseudoinverse, and the QR factorization of the channel defined as $\mathbf{H} = \mathbf{QR}$, the ML detection problem can be reformulated as

$$\hat{\mathbf{s}}_{ML} = \arg\min_{\mathbf{s}\in\Omega^N} \|\mathbf{R}(\mathbf{s} - \hat{\mathbf{s}})\|^2.$$

The lattice point $\mathbf{Hs}$ is included by the sphere $S(\mathbf{y}, d)$ with center point $\mathbf{y}$ and radius d if the following inequality is satisfied $\|\mathbf{R}(\mathbf{s} - \hat{\mathbf{s}})\|^2 \leqslant d^2$,

$$\left\| \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1N} \\ 0 & r_{22} & \cdots & r_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{NN} \end{pmatrix} \begin{pmatrix} s_1 - \hat{s}_1 \\ s_2 - \hat{s}_2 \\ \vdots \\ s_N - \hat{s}_N \end{pmatrix} \right\|^2 \leqslant d^2.$$

Starting with dimension N, the elements of the symbol set are inserted into the partial symbol vector and the inequality condition is evaluated. The search process is analogous to a depth-first tree search that is highly sequential, consequently, this problem cannot be efficiently solved in a multi-threaded environment.

The PSD completely eliminates the sequential parts of the SD algorithm. The tree traversal of the PSD algorithm is implemented by a novel hybrid tree search method, where the algorithm parallelism is assured by the efficient combination of depth-first search and breadth-first search algorithms. Because of the hybrid tree search only distinct levels of the tree are evaluated that are denoted by the parameter lvl_x. On these levels the number exp_{lvl_x} of partial symbol vectors are expanded simultaneously. During the expansion of a partial symbol vector $(lvl_{x-1} - lvl_x)$ number of new symbols are added to the original symbol vector. The simultaneous expansion of $exp_{lvl_{x-1}}$ number of partial symbol vectors on level lvl_{x-1} will create $eval_{lvl_x} = exp_{lvl_{x-1}} \cdot |\Omega|^{(lvl_{x-1} - lvl_x)}$ number of new partial symbol vectors on level lvl_x. Note, that a hybrid search is realized at this point, because with parameters exp_{lvl_x} the extent of the breadth search, while with

DOI:10.15774/PPKE.ITK.2015.010

parameters lvl_x the extent of the depth search is controlled. Since several new (partial) symbol vectors are created after the expansion stage, the parallel path metric update becomes possible, thus, the resources of an MPA can be efficiently exploited.

Figures 4.20, 4.21, 4.22 show the average number of expanded nodes per thread for different MIMO systems and symbol set configurations. The signal space of real-equivalent 8×8 MIMO with symbol set size of $|\Omega| = 8$ symbols has about 1.6×10^7 symbol vectors. For an SNR of 5 dB the PSD expands into about 310 nodes per thread while the *Automatic Sphere Detector* expands into about 7500 nodes per thread. Consequently, the total workload of a thread running the PSD algorithm is reduced by 96%. For an SNR of 20 dB, the workload of a thread running the PSD algorithm is reduced by 95%.

In Fig. 4.18 the average detection throughput achieved with (i) the PSD algorithm implemented on the GTX690 GP-GPU and (ii) the sequential SD executed simultaneously on every thread of an Intel Xeon CPU E5-2650 v3 was compared. At 30 dB SNR for a 4×4 MIMO and $|\Omega| = 4$ the detection throughput is increased by 6 times, and for $|\Omega| = 8$ the throughput is increased by 50 times by the GP-GPU.

Thesis I.b.

I defined highly parallel, dynamic building blocks for the Expansion and Evaluation pipeline of the PSD algorithm as a function of available parallelism. Based on the building blocks, I identified a set of parameters that determine the extent of parallelism and memory footprint. I showed that the achieved average detection throughput of the GP-GPU mapping outperformed every existing true-ML detector and many non-ML GP-GPU, ASIC, DSP and FPGA implementations.

Throughout the detection process the most heavily used operations are the vector expansion and evaluation. In order to remove every possible bottleneck and to make a parallel implementation possible, I have introduced the EEP. The stages of the EEP are defined as: (i) the *Preparatory Block*, (ii) the *Selecting, Mapping and Merging Block*, (iii) the *Path Metric Evaluation Block*, and (iv) the *Searching or Sorting Block* as shown in Fig. 4.10.

In the *Preparatory Block* virtual identifiers are computed simultaneously by tt number of threads where the k-th thread is denoted by t_{id}^k. The virtual identifiers are computed

DOI:10.15774/PPKE.ITK.2015.010

in the following manner:

$$VT^{k}_{lvl_x} = \{vt_{lvl_x} | vt_{lvl_x} = (t^{k}_{id} + n \cdot tt) \mod |\Omega|^{(lvl_{x-1} - lvl_x)},$$
$$n = 0 : \lceil eval_{lvl_x}/tt \rceil - 1\},$$

$$VB^{k}_{lvl_x} = \{vb_{lvl_x} | vb_{lvl_x} = \lfloor (t^{k}_{id} + n \cdot tt)/|\Omega|^{(lvl_{x-1} - lvl_x)} \rfloor,$$
$$n = 0 : \lceil eval_{lvl_x}/tt \rceil - 1\}.$$

In the *Selecting, Mapping and Merging* block previously evaluated partial symbol vectors are selected and further expanded. In the *Selecting* phase, previously evaluated partial symbol vectors $\mathbf{s}^{N}_{lvl_{x-1}}$ are selected based on the thread's virtual block identifiers $vb_{lvl_x} \in VB^{k}_{lvl_x}$. In the *Mapping* phase the virtual thread identifiers $vt_{lvl_x} \in VT^{k}_{lvl_x}$ are mapped to $\mathbf{s}^{lvl_{x-1}-1}_{lvl_x}$ partial symbol vectors. Finally, in the *Merging* phase each selected vector $\mathbf{s}^{N}_{lvl_{x-1}}$ and mapped symbol vector $\mathbf{s}^{lvl_{x-1}-1}_{lvl_x}$ is merged as $\mathbf{s}^{N<j>}_{lvl_x} = (\mathbf{s}^{lvl_{x-1}-1<j>}_{lvl_x}, \mathbf{s}^{N<j>}_{lvl_{x-1}})$.

In the *Path Metric Evaluation* block, the path metric of the expanded partial symbol vectors is updated. This is one of the most time-consuming steps, however, to reduce the time required the path metrics are updated in parallel by several threads.

The *Searching or Sorting* block of the EEP is one of the most important stages during the detection. Depending on the level of processing, either sorting or a minimum search is performed. The minimum search is applied only when the detection has reached the last processing level, while sorting is applied on all other levels. The sorting is done with the use of sorting networks [89], [88], [90]. Due to their data-independent structure, their operation sequence is completely rigid. This property makes this algorithm parallelizable for the GP-GPU architecture. The minimum search algorithm relies on the parallel prefix sum algorithm [91].

As a result, I elaborated a highly parallel expansion and evaluation pipeline where no frequent thread synchronization is required. This enables a very efficient utilization of an MPA. In Table 4.7 I compared the average detection throughput of the PSD algorithm achieved with optimal ML implementations known from the literature. The PSD algorithm outperformed each of them. Further comparison was made with non-optimal FPGA, DSP, ASIC and GP-GPU implementations. The average detection throughput of the PSD was better in the majority of cases. Although, some FPGA and VLSI based non-optimal detectors showed a better performance, but those solutions suffer from a loss in BER performance.

DOI:10.15774/PPKE.ITK.2015.010

Thesis I.c.

I proposed a dynamic computing load scheduling algorithm that combines in a very efficient manner the system level and device level parallelism. The result of the elaborated scheduling is a dynamic binding between the symbol vectors and the thread blocks, that allows to configure grids with significantly less thread blocks. By reducing the size of the grids, the resources of the streaming multiprocessors are shared between several grids, thus, the concurrent executions of kernels on multiple streams are enhanced. Thereby, the idle time of the processing units, caused by the variable complexity of the symbol detection, is minimized and the average detection throughput achieved is increased.

The *system level* parallelism is implemented by the parallel processing of fading blocks of a received frame. Consequently, the number of kernels launched is equal to the number of independent channel realizations. Every grid assigned to a kernel launches several TBs and the detection of the symbol vectors associated to one channel realization is done by the threads of the TBs. The configuration of the grids, namely the binding of the TBs and symbol vectors, is critical since this influences the concurrent execution of the kernels.

A straightforward binding requires a high number of TBs, because the resources of the GP-GPU will be available for a long time duration only for one kernel, thus, the concurrent execution of the kernels of different streams is limited. By reducing the number of TBs and keeping the load constant, the varying detection time of the different symbol vectors could amplify the tail effect. This means that only a few TBs of a grid are working and the resources of the streaming multiprocessors are not freed up.

In the proposed dynamic computing load scheduling algorithm the number of TBs in a grid is significantly smaller compared to the straightforward binding case. The work for a TB is dynamically distributed, namely, when the detection of one symbol vector is finished, the PSD algorithm executed by the threads of the TB evaluates the next unprocessed symbol vector. By means of this technique, the tail effect introduced by the varying processing time of different symbol vectors is balanced. As a result, the *device level* parallelism, namely, the concurrent execution of multiple kernels on different streams, is enhanced.

The effect of dynamic computing load scheduling is shown in Fig. 4.17 for a 4×4 MIMO system with symbol sets $|\Omega| = 2$, 4 and 8. An increase of $15\% - 30\%$ for $|\Omega| = 2$, 4 and $38\% - 64\%$ for $|\Omega| = 8$ of average detection throughput is observed.

DOI:10.15774/PPKE.ITK.2015.010

Thesis group II. Channel preprocessing techniques for true-ML hard-output MIMO detection.

(Related articles [1], [3].)

Thesis II.a.

I experimentally proved that the computational complexity of the PSD algorithm is reduced considerably by defining the detection order based on the inverse channel row norms. The aim of ordering is to detect symbols with lower signal strength on levels where a full breadth-first search is performed. This approach maximizes the probability that the best path metric partial symbol vector is the optimal choice on these levels. I showed that the applied inverse channel based row norm ordering increases the average detection throughput and decreases the number of expanded nodes.

Detectors based on successive interference cancellation are seriously influenced by the order of detected symbols. In case if the detected symbol is different from the symbol sent then symbol cancellation introduces noise instead of lowering the number of interferers. Several ordering metrics have been introduced in the literature [64]. The most important ordering metrics are based on the (i) signal-to-interference plus noise ratio (SINR), (ii) signal-to-noise ratio (SNR), and (iii) channel matrix column norms.

The metrics based on SINR and SNR involve complex computations. A simpler metric based on the column norms of the channel matrix can be represented as:

$$\mathbf{y} = \mathbf{H}\mathbf{s}_t + \mathbf{v} = \mathbf{h}_1 s_1 + \mathbf{h}_2 s_2 + \cdots + \mathbf{h}_n s_n + \mathbf{v} \tag{6.1}$$

where $\mathbf{h}_i$ represents the i-th column of the channel matrix $\mathbf{H}$. The ordering metric is based on the norms of the column vectors $\|\mathbf{h}_i\|$. As a result, the received signal strength is proportional with the ordering metric.

Algorithms based on SIC require to detect the strongest symbols first. However, the PSD starts the detection process with the symbols having the lowest metric, because at the top of the tree a full breadth-first search is performed and the search is continued with the best path metric symbol vectors. Since every possibility is examined the error probability introduced by the lower signal strength is minimized.

The effect of matrix preprocessing based on decreasing ordering of the norms of the row vectors of the inverse channel matrix was evaluated. By applying channel prepro-

DOI:10.15774/PPKE.ITK.2015.010

cessing an extra increase of $5 - 10\%$ in average detection throughput was achieved, as shown in Fig. 4.17.

Thesis group III. Complexity reduced parallel Lattice Reduction algorithms mapped to massively parallel and heterogeneous platforms.

(Related articles [2], [4], [5].)

Thesis III.a.

I proposed a parallel Cost-Reduced All-Swap LLL (CR-AS-LLL) lattice reduction algorithm where the cost reduction consists in delaying the update of the off-diagonal Gram-Schmidt coefficients when the size reductions and column swaps are performed. I elaborated a GP-GPU mapping of the CR-AS-LLL algorithm relying on a two-dimensional thread block configuration. I showed that efficient work distribution, memory access, inner product and size reduction computation are achieved with the proposed mapping. The average computational time of the GP-GPU mapping achieves one order of magnitude improvement compared to the multi-core CPU mapping.

After every size reduction or column swap the Gram-Schmidt coefficients are updated in the original parallel All-Swap LLL algorithm. However, a lot of unnecessary computations are performed, because the frequent size reductions and column swaps change the value of the Gram-Schmidt coefficients several times. In the proposed CR-AS-LLL algorithm only the $\mu_{k,k-1}$ Gram-Schmidt coefficients are updated regularly because the evaluation of the LLL conditions depend only on these parameters. The rest off the coefficients are updated after finishing the swaps and size reductions operations.

When mapped to GP-GPU, the performance of the CR-AS-LLL algorithm depends on the efficiency of the *work distribution* among the available GP-GPU threads and the implementation of the most frequently used operations, such as *dot products*, *size reductions* and *column swaps*. Figure 5.7 presents a possible mapping for the main parts of the CR-AS-LLL algorithm. The kernel is launched with a one dimensional *grid* whose size is determined by the number of lattice basis processed simultaneously. The thread blocks TB(T_x, T_y) launched have a two dimensional configuration, where T_x and T_y denote the number of threads in the x and y dimension. The number of threads T_y is defined based on the size of the original basis, i.e., $T_y = \min(n/2, 32)$. By enabling the usage of $T_x = \min(n, 32)$ threads in the x dimension, the threads, that belong to the same

DOI:10.15774/PPKE.ITK.2015.010

y dimension, will form a *warp*. Consequently, the elements of matrices $\mathbf{B}, \mathbf{B}^*$ stored in the global memory are accessed through the coalesced memory pattern exploiting the available memory bandwidth. The size of the low latency *shared memory* is limited. Thus, only those Gram-Schmidt coefficients are stored in this memory which are required to evaluate the LLL conditions. Shared memory also plays an important role in computing the dot products and in the column swap procedures.

Figures 5.14 and 5.15 compare the average computational time of the CR-AS-LLL mapped on a GP-GPU and a CPU. The GP-GPU outperforms the CPU for every matrix dimension with speed-up ranging from 6 to 15.

Thesis III.b.

I proposed the Cost-Reduced Modified-Block LLL (CR-MB-LLL) algorithm where two levels of parallelism are identified and exploited enhancing the lattice reduction of higher dimensional lattice basis. The higher level parallelism follows the block reduction concept where the original lattice basis is divided into several smaller sized sub-matrices and, on the lower level, the parallel lattice reduction of the sub-matrices is done by the CR-AS-LLL algorithm. I showed that for large matrices the CR-MB-LLL algorithm is more efficient than the CR-AS-LLL algorithm.

The problem division to several sub-problems that can be executed concurrently can be regarded as one level of parallelism. In addition, if a sub-problem could benefit from a multi-threaded environment it can be regarded as a second level of parallelism. Previous parallel LR implementations, such as the ones presented in [118], [119], [120] have focused only on multi-core architectures. The main drawback of the low number of threads offered by modern CPUs (compared to GP-GPUs) is that low-level parallelism cannot be exploited in an efficient manner. During the algorithm design, low-level parallelism is usually omitted and the levels of parallelism are also restricted. In case of GP-GPUs, the high number of CUDA cores makes the parallel execution of a high number of threads possible offering significant performance improvements.

The CR-MB-LLL algorithm is designed to exploit the benefits of a highly multi-threaded environment. The CR-MB-LLL algorithm splits the original basis into several sub-problems with lower dimension and performs parallel LLL reduction on them. Because the LLL reduction of the subgroups and the boundaries check can be done independently, no frequent synchronization is required. Thus, coarse grained parallelism is achieved by creating the sub-problems. The GP-GPU mapping of the CR-MB-LLL

DOI:10.15774/PPKE.ITK.2015.010

algorithm is similar to the one presented in case of the CR-AS-LLL algorithm because the procedures used are performed with a two dimensional TB configuration even in the case of a boundary check.

The CR-MB-LLL algorithm reduces further the computational complexity of the MB-LLL algorithm. In the MB-LLL algorithm, the submatrices affected by boundary swap have to be LLL reduced and the Gram-Schmidt coefficients have to be updated. The complexity reduction in the CR-MB-LLL algorithm is achieved by eliminating the GS coefficients update in the submatrices after the execution of the CR-AS-LLL and with the simplified swap procedure.

As shown in Figs. 5.14 and 5.15, the computational time of the CR-MB-LLL is $25 - 40\%$ lower in case of small and medium-sized matrices compared to the MB-LLL algorithm. Furthermore, the block concept implemented in the CR-MB-LLL achieves 30% speed-up for large matrices compared to the CR-AS-LLL.

Thesis III.c.

I proposed a heterogeneous platform and a suitable mapping for the Cost-Reduced Modified-Block LLL algorithm where the scheduling of kernels is implemented by a CPU and the processing tasks are executed by GP-GPU kernels. I compared the performance of the proposed heterogeneous platform with a dynamic parallelism based GP-GPU mapping and a parallel CPU implementation. I showed that the average computational time is better by one order of magnitude for smaller and middle sized matrices when a heterogeneous platform is used.

The schematic of the heterogeneous platform is shown in Fig. 5.9. The CPU threads launch (i) the *CR-AS-LLL* kernels in order to LLL reduce the sub-matrices, (ii) the *Boundary Check* kernels for checking the LLL conditions at the boundaries of the sub-groups and (iii) launch the *Coefficients Update* kernel to update the Gram-Schmidt coefficients and to perform the size reductions wherever it is required.

The control logic of the dynamic scheduling is implemented by the CPU threads. A different CUDA stream is assigned for every CPU thread, making the concurrent kernel execution possible and reducing the idle time of the CUDA cores. The status of the sub-matrices is updated continuously in the GP-GPU global memory and it is communicated to the CPU, thus, the size of the grids assigned to *CR-AS-LLL* and *Boundary Check* kernels is dynamically adjusted according to the number of modified sub-matrices in every iteration. The *Coefficients Update* kernel starts after all the matrices assigned to

DOI:10.15774/PPKE.ITK.2015.010

one CPU thread are completely processed.

Figure 5.11 shows the computational times of the MB-LLL algorithm based on three different architectures for different matrix dimensions. The performance was evaluated on a Tesla K20 GP-GPU and an Intel Core i7-3820 processor. The heterogeneous platform clearly outperforms the solutions based on dynamic parallelism in the case of small matrices and shows similar performance for large matrices. The CPU implementation is outperformed for all of the cases. The conclusion is that the data transfer between CPU and GP-GPU required by the heterogeneous system is less time consuming than the overhead of the kernel launch with dynamic parallelism and the limitation of the concurrent execution of kernels on different streams.

6.3 Applicability of the results

Lattice reduction is a powerful concept for solving diverse problems involving point lattices. It is a topic of great interest, both as a theoretical tool and as a practical technique. Since point lattices and lattice reduction plays a key role in numerous fields of applications, my goal was to enhance the performance of the polynomial-time LLL lattice reduction algorithm.

The results presented in Thesis group III. prove that my goal was successfully achieved, since I reduced the complexity of the LLL algorithm, I identified and exploited several levels of parallelism that lead to efficient algorithm mapping to different parallel architectures and heterogeneous platforms. By exploiting the resources of this powerful architectures the processing time of the LR was significantly decreased. The following enumeration gives a brief summary where the results of Thesis group III. can be applied.

- In the field of *wireless communications* my results could enhance: (i) the equalization of frequency-selective channels [123], (ii) the equalization in precoded orthogonal frequency division multiplexing systems [124], (iii) the source and channel coding in scenarios with multiple terminals [125], and the preprocessing of sphere decoding [61]. When used in conjunction with LR methods, lower complexity linear and non-linear detection and precoding methods achieve full diversity order [14], [10]. The computational complexity of these methods is mostly determined by the preprocessing LR algorithm, however, my results presented in Thesis group III. significantly reduce the complexity of the LLL algorithm, achieving better processing times.
- My results can be applied in the field of *image processing* for improving the speed of radar imaging, magnetic resonance imaging and color space estimation in JPEG

DOI:10.15774/PPKE.ITK.2015.010

images as shown [126] and [127].

- In the field of *combinatorial mathematics* it is possible to phrase many different problems as questions about lattices. Lattice problems arise in integer programming [107], subset sum problems [67], factoring polynomials with rational coefficients [101], and diophantine approximation just to name a few of them. My results presented in Thesis group III. could speed-up the solution of these problems.
- As shown in [128] methods based on LR have been used in *cryptography* where the processing time has a critical role.

Research in information theory has revealed that important improvements can be achieved in data rate when multiple antennas are applied at both the transmitter and receiver sides [8]. Unfortunately, with the increased performance the complexity of the associated signal processing problems is also increased. The complexity of the optimal ML detection in MIMO systems increases exponentially with the number of transmit antennas and modulation order, thus, its use in practical systems is prohibitive. The SD algorithm was developed and refined in [69], [67], [61] in order to significantly reduce the search space. However, the sequential components of the SD algorithm are a serious limitation in a parallel environment.

In Thesis group I. with the PSD algorithm, I proposed a highly parallel algorithm that eliminated the sequential components and bottlenecks of the SD algorithm and the efficient mapping to massively parallel architectures could be realized. In Thesis group II., I further improved the performance of the PSD algorithm by defining a detection ordering based on the inverse channel matrix row norms. These results made possible to significantly improve the computation time of the optimal BER curves in larger MIMO systems under different circumstances that was very time-consuming until now.

It was shown that the SD algorithm is analogous to the closest lattice point (CLP) problem, or equivalently, the shortest vector problem (SVP) [61], [62], [71]. Since optimal LR techniques, such as the Minkowski and Hermite-Korkine-Zolotareff LR algorthms, iterativetly perform CLP searches and cryptography problems can be traced back to CLP and SVP problems, my results presented in Thesis groups I. and II. can be applied to enhance the solution of these problems.

DOI:10.15774/PPKE.ITK.2015.010

Bibliography

Author's journal publications

[1] **Csaba M. Józsa**, Géza Kolumbán, Antonio M. Vidal, Francisco J. Martínez-Zaldívar, and Alberto González. "Parallel Sphere Detector algorithm providing optimal MIMO detection on massively parallel architectures". In: *Concurrency and Computation: Practice and Experience* (2015). DOI: `10.1002/cpe.3488`.

[2] **Csaba M. Józsa**, Fernando Domene, Antonio M. Vidal, Gema Piñero, and Alberto González. "High performance lattice reduction on heterogeneous computing platform". In: *The Journal of Supercomputing* (2014), pp. 1–14. ISSN: 0920-8542. DOI: `10.1007/s11227-014-1201-2`.

Author's conference publications

[3] **Csaba M. Józsa**, Géza Kolumbán, Antonio M. Vidal, Francisco-José Martínez-Zaldívar, and Alberto González. "New Parallel Sphere Detector Algorithm Providing High-Throughput for Optimal MIMO Detection". In: *2013 International Conference on Computational Science (ICCS 2013)*. Vol. 18. Barcelona, Spain, 2013, pp. 2432 –2435. DOI: `http://dx.doi.org/10.1016/j.procs.2013.05.417`.

[4] **Csaba M. Józsa**, Fernando Domene, Gema Piñero, Alberto González, and Antonio M. Vidal. "Efficient GPU implementation of Lattice-Reduction-Aided Multiuser Precoding". In: *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on.* Ilmenau, Germany, Aug. 2013, pp. 1–5. ISBN: 978-3-8007-3529-7.

[5] Fernando Domene, **Csaba M. Józsa**, Antonio M. Vidal, Gema Piñero, and Alberto González. "Performance analysis of a parallel Lattice Reduction algorithm on many-core architectures". In: *The 13th International Conference on Compu-*

DOI:10.15774/PPKE.ITK.2015.010

tational and Mathematical Methods in Science and Engineering (CMMSE 2013). Vol. 2. Almeria, Spain, June 2013, pp. 535–542. ISBN: 978-84-616-2723-3.

[6] Tamás Krébesz, **Csaba M. Józsa**, and Géza Kolumbán. “New carrier generation techniques and their influence on bit energy in UWB radio”. In: *Circuit Theory and Design (ECCTD), 2011 20th European Conference on.* IEEE. Aug. 2011, pp. 801–804. DOI: 10.1109/ECCTD.2011.6043838.

[7] Tamás Krébesz, Géza Kolumbán, and **Csaba M. Józsa**. “Ultra-wideband impulse radio based on pulse compression technique”. In: *Circuit Theory and Design (ECCTD), 2011 20th European Conference on.* IEEE. Aug. 2011, pp. 797–800. DOI: 10.1109/ECCTD.2011.6043839.

Related publications

[8] Emre Telatar. “Capacity of Multi-antenna Gaussian Channels”. In: *European Transactions on Telecommunications* 10.6 (1999), pp. 585–595. ISSN: 1541-8251.

[9] Ezio Biglieri, Robert Calderbank, Anthony Constantinides, Andrea Goldsmith, Arogyaswami Paulraj, and H. Vincent Poor. *MIMO Wireless Communications.* New York, NY, USA: Cambridge University Press, 2007. ISBN: 0521873282.

[10] Christoph Windpassinger, Robert FH Fischer, Tomáš Vencel, and Johannes B Huber. “Precoding in multiantenna and multiuser communications”. In: *IEEE Trans. Wireless Commun.* 3.4 (2004), pp. 1305–1316.

[11] C.B. Peel, B.M. Hochwald, and A.L. Swindlehurst. “A vector-perturbation technique for near-capacity multiantenna multiuser communication - Part I: channel inversion and regularization”. In: *IEEE Trans. Commun.* 53.1 (2005), pp. 195–202.

[12] B.M. Hochwald, C.B. Peel, and A.L. Swindlehurst. “A vector-perturbation technique for near-capacity multiantenna multiuser communication - Part II: perturbation”. In: *IEEE Trans. Commun.* 53.3 (2005), pp. 537–544.

[13] Daofeng Xu, Yongming Huang, and Luxi Yang. “Improved nonlinear multiuser precoding using lattice reduction”. In: *Signal, image and video processing* 3.1 (2009), pp. 47–52.

DOI:10.15774/PPKE.ITK.2015.010

[14] Huan Yao and Gregory W. Wornell. “Lattice-reduction-aided detectors for MIMO communication systems”. In: *Global Telecommunications Conference, 2002. GLOBECOM ’02. IEEE.* Vol. 1. Nov. 2002, pp. 424–428.

[15] C. Windpassinger and R.F.H. Fischer. “Low-complexity near-maximum-likelihood detection and precoding for MIMO systems using lattice reduction”. In: *Information Theory Workshop, 2003. Proceedings. 2003 IEEE.* Mar. 2003, pp. 345–348.

[16] M. Taherzadeh, A. Mobasher, and A.K. Khandani. “LLL Reduction Achieves the Receive Diversity in MIMO Decoding”. In: *Information Theory, IEEE Transactions on* 53.12 (Dec. 2007), pp. 4801–4805. ISSN: 0018-9448.

[17] C. Studer, D. Seethaler, and H. Bolcskei. “Finite lattice-size effects in MIMO detection”. In: *Signals, Systems and Computers, 2008 42nd Asilomar Conference on.* Oct. 2008, pp. 2032–2037.

[18] Xiaoli Ma and Wei Zhang. “Performance analysis for MIMO systems with lattice-reduction aided linear equalization”. In: *Communications, IEEE Transactions on* 56.2 (Feb. 2008), pp. 309–318. ISSN: 0090-6778.

[19] Wen-mei W. Hwu. *GPU Computing Gems Emerald Edition.* 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[20] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A.M. Vidal. “Fully Parallel GPU Implementation of a Fixed-Complexity Soft-Output MIMO Detector”. In: *Vehicular Technology, IEEE Transactions on* 61.8 (Oct. 2012), pp. 3796–3800.

[21] Michael Wu, Yang Sun, Siddharth Gupta, and Joseph R. Cavallaro. “Implementation of a High Throughput Soft MIMO Detector on GPU”. In: *J. Signal Process. Syst.* 64.1 (July 2011), pp. 123–136. ISSN: 1939-8018.

[22] Wang Hongyuan and Chen Muyi. “A Fixed-Complexity Sphere Decoder for MIMO Systems on Graphics Processing Units”. In: *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on.* Dec. 2010.

[23] T. Nylanden, J. Janhunen, O. Silven, and M. Juntti. “A GPU implementation for two MIMO-OFDM detectors”. In: *Embedded Computer Systems (SAMOS), 2010 International Conference on.* July 2010, pp. 293–300.

[24] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, and G. Knagge. “Silicon complexity for maximum likelihood MIMO detection using spherical decoding”. In: *Solid-State Circuits, IEEE Journal of* 39.9 (2004), pp. 1544–1552.

DOI:10.15774/PPKE.ITK.2015.010

[25] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei. “VLSI implementation of MIMO detection using the sphere decoding algorithm”. In: *Solid-State Circuits, IEEE Journal of* 40.7 (2005), pp. 1566–1577.

[26] X. Huang, C. Liang, and J. Ma. “System architecture and implementation of MIMO sphere decoders on FPGA”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16.2 (2008), pp. 188–197.

[27] Rongchun Li, Yong Dou, Dan Zou, Shi Wang, and Ying Zhang. “Efficient graphics processing unit based layered decoders for quasicyclic low-density parity-check codes”. In: *Concurrency and Computation: Practice and Experience* 27.1 (2013), pp. 29–46. ISSN: 1532-0634.

[28] Rongchun Li, Yong Dou, and Dan Zou. “Efficient parallel implementation of three-point viterbi decoding algorithm on CPU, GPU, and FPGA”. In: *Concurrency and Computation: Practice and Experience* 26.3 (2014), pp. 821–840.

[29] Fernando Domene, Sandra Roger, Carla Ramiro, Gema Pinero, and Alberto Gonzalez. “A reconfigurable GPU implementation for Tomlinson-Harashima precoding”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on.* 2012.

[30] J. Kim, Seungheon Hyeon, and Seungwon Choi. “Implementation of an SDR system using graphics processing unit”. In: *Communications Magazine, IEEE* 48.3 (2010), pp. 156–162. ISSN: 0163-6804.

[31] Chiyoung Ahn et al. “Implementation of an SDR system using an MPI-based GPU cluster for WiMAX and LTE”. English. In: *Analog Integrated Circuits and Signal Processing* 73.2 (2012), pp. 569–582. ISSN: 0925-1030.

[32] Luis G Barbero, David L Milliner, T Ratnarajah, John R Barry, and Colin Cowan. “Rapid Prototyping of Clarkson’s Lattice Reduction for MIMO Detection”. In: *Communications, 2009. ICC’09. IEEE International Conference on.* 2009, pp. 1–5.

[33] Brian Gestner, Wei Zhang, Xiaoli Ma, and David V Anderson. “VLSI implementation of a lattice reduction algorithm for low-complexity equalization”. In: *Circuits and Systems for Communications, 2008. ICCSC 2008. 4th IEEE International Conference on.* 2008, pp. 643–647.

DOI:10.15774/PPKE.ITK.2015.010

[34] B. Gestner, Wei Zhang, Xiaoli Ma, and D.V. Anderson. "Lattice Reduction for MIMO Detection: From Theoretical Analysis to Hardware Realization". In: *Circuits and Systems I: Regular Papers, IEEE Transactions on* 58.4 (Apr. 2011), pp. 813–826. ISSN: 1549-8328.

[35] M. Shabany, A. Youssef, and G. Gulak. "High-Throughput 0.13-μm CMOS Lattice Reduction Core Supporting 880 Mb/s Detection". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 21.5 (May 2013), pp. 848–861. ISSN: 1063-8210.

[36] M. Flynn. "Very high-speed computing systems". In: *Proceedings of the IEEE* 54.12 (Dec. 1966), pp. 1901–1909. ISSN: 0018-9219.

[37] M. Flynn. "Some Computer Organizations and Their Effectiveness". In: *Computers, IEEE Transactions on* C-21.9 (1972), pp. 948–960. ISSN: 0018-9340.

[38] Michael Flynn. "Flynn's Taxonomy". English. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Springer US, 2011, pp. 689–697. ISBN: 978-0-387-09765-7.

[39] NVIDIA Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110*. 2012.

[40] NVIDIA Corporation. *CUDA C Programming Guide*. http://docs.nvidia.com/cuda/cuda-c-programming-guide/. 2012.

[41] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007. ISBN: 0262533022, 9780262533027.

[42] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. 2nd. (Revised). Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262692155.

[43] Khronos OpenCL Working Group. *The OpenCL Specification, version 1.0.29*. 2008. URL: `http://khronos.org/registry/cl/specs/opencl-1.0.29.pdf`.

[44] Andrea Goldsmith. *Wireless Communications*. New York, NY, USA: Cambridge University Press, 2005. ISBN: 0521837162.

[45] S. Alamouti. "A simple transmit diversity technique for wireless communications". In: *Selected Areas in Communications, IEEE Journal on* 16.8 (Oct. 1998), pp. 1451–1458. ISSN: 0733-8716.

DOI:10.15774/PPKE.ITK.2015.010

[46] Jiann-Ching Guey, M.P. Fitz, M.R. Bell, and Wen-Yi Kuo. “Signal design for transmitter diversity wireless communication systems over Rayleigh fading channels”. In: *Communications, IEEE Transactions on* 47.4 (Apr. 1999), pp. 527–537. ISSN: 0090-6778.

[47] Vahid Tarokh, N. Seshadri, and A.R. Calderbank. “Space-time codes for high data rate wireless communication: performance criterion and code construction”. In: *Information Theory, IEEE Transactions on* 44.2 (Mar. 1998), pp. 744–765. ISSN: 0018-9448.

[48] Vahid Tarokh, Hamid Jafarkhani, and A.R. Calderbank. “Space-time block codes from orthogonal designs”. In: *Information Theory, IEEE Transactions on* 45.5 (July 1999), pp. 1456–1467. ISSN: 0018-9448.

[49] Gerard J. Foschini. “Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas”. In: *Bell Labs Technical Journal* 1.2 (1996), pp. 41–59. ISSN: 1089-7089.

[50] M. Arar and A. Yongacoglu. “Parallel low-complexity MIMO detection algorithm using QR decomposition and Alamouti space-time code”. In: *Wireless Conference (EW), 2010 European*. Apr. 2010, pp. 141–148.

[51] Vahid Tarokh, A. Naguib, N. Seshadri, and A.R. Calderbank. “Combined array processing and space-time coding”. In: *Information Theory, IEEE Transactions on* 45.4 (May 1999), pp. 1121–1128.

[52] Meixia Tao and R.S. Cheng. “Generalized layered space-time codes for high data rate wireless communications”. In: *Wireless Communications, IEEE Transactions on* 3.4 (July 2004), pp. 1067–1075.

[53] Claude Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27 (1948), pp. 379–423.

[54] S.K. Jayaweera and H.V. Poor. “Capacity of multiple-antenna systems with both receiver and transmitter channel state information”. In: *Information Theory, IEEE Transactions on* 49.10 (Oct. 2003), pp. 2697–2709. ISSN: 0018-9448.

[55] G.J. Foschini and M.J. Gans. “On Limits of Wireless Communications in a Fading Environment when Using Multiple Antennas”. English. In: *Wireless Personal Communications* 6.3 (1998), pp. 311–335. ISSN: 0929-6212.

DOI:10.15774/PPKE.ITK.2015.010

[56] L.G. Barbero and J.S. Thompson. “Fixing the Complexity of the Sphere Decoder for MIMO Detection”. In: *Wireless Communications, IEEE Transactions on* 7.6 (June 2008), pp. 2131–2142.

[57] M.S. Khairy, C. Mehlfuhrer, and M. Rupp. “Boosting sphere decoding speed through Graphic Processing Units”. In: *Wireless Conference (EW), 2010 European*. IEEE. 2010, pp. 99–104.

[58] Mostafa El-Khamy, Mostafa Medra, and Hassan M. ElKamchouchi. “Reduced complexity list sphere decoding for MIMO systems”. In: *Digital Signal Processing* 0 (2013). ISSN: 1051-2004.

[59] Chiao-En Chen and Wei-Ho Chung. “Computationally efficient near-optimal combined antenna selection algorithms for V-BLAST systems”. In: *Digital Signal Processing* 23.1 (2013), pp. 375 –381. ISSN: 1051-2004.

[60] Gianmarco Romano, Domenico Ciuonzo, Pierluigi Salvo Rossi, and Francesco Palmieri. “Low-complexity dominance-based sphere decoder for MIMO systems”. In: *Signal Processing* 93.9 (2013), pp. 2500 –2509. ISSN: 0165-1684.

[61] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. “Closest point search in lattices”. In: *Information Theory, IEEE Transactions on* 48.8 (2002).

[62] M.O. Damen, H. El Gamal, and G. Caire. “On maximum-likelihood detection and the search for the closest lattice point”. In: *Information Theory, IEEE Transactions on* 49.10 (2003), pp. 2389–2402.

[63] A.D. Murugan, H. El Gamal, M.O. Damen, and G. Caire. “A unified framework for tree search decoding: rediscovering the sequential decoder”. In: *Information Theory, IEEE Transactions on* 52.3 (2006), pp. 933–953.

[64] P.W. Wolniansky, G.J. Foschini, G.D. Golden, and R. Valenzuela. “V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channel”. In: *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*. IEEE. Sept. 1998, pp. 295–300.

[65] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*. Vol. 671. Springer Science & Business Media, 2002.

[66] U. Fincke and M. Pohst. “Improved Methods for Calculating Vectors of Short Length in a Lattice, Including a Complexity Analysis”. In: *Mathematics of Computation* 44.170 (1985), pp. 463–471.

DOI:10.15774/PPKE.ITK.2015.010

[67] C. P. Schnorr and M. Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Mathematical Programming* 66 (1994), pp. 181–199.

[68] J. H. Conway, N. J. A. Sloane, and E. Bannai. *Sphere-packings, lattices, and groups*. New York, NY, USA: Springer-Verlag, Inc., 1987.

[69] M. Pohst. "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications". In: *ACM SIGSAM Bulletin* 15.1 (1981), pp. 37–44.

[70] E. Viterbo and E. Biglieri. "A universal decoding algorithm for lattice codes". In: *14 Colloque sur le traitement du signal et des images, FRA, 1993*. GRETSI, Groupe d'Etudes du Traitement du Signal et des Images. 1993.

[71] B. Hassibi and H. Vikalo. "On the sphere-decoding algorithm I. Expected complexity". In: *Signal Processing, IEEE Transactions on* 53.8 (2005).

[72] H. Vikalo and B. Hassibi. "On the sphere-decoding algorithm II. Generalizations, second-order statistics, and applications to communications". In: *Signal Processing, IEEE Transactions on* 53.8 (2005), pp. 2819–2834.

[73] J. Jalden and B. Ottersten. "On the complexity of sphere decoding in digital communications". In: *Signal Processing, IEEE Transactions on* 53.4 (2005), pp. 1474 –1484.

[74] J. Fink, S. Roger, A. Gonzalez, V. Almenar, and V.M. Garcia. "Complexity assessment of sphere decoding methods for MIMO detection". In: *Signal Processing and Information Technology (ISSPIT), 2009 IEEE International Symposium on.* Dec. 2009, pp. 9–14.

[75] Markus Myllylä, Markku Juntti, and Joseph R. Cavallaro. "Implementation aspects of list sphere decoder algorithms for MIMO-OFDM systems". In: *Signal Processing* 90.10 (2010), pp. 2863 –2876. ISSN: 0165-1684.

[76] P. van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Report. Department of Mathematics. University of Amsterdam. Department, Univ., 1981.

[77] Su, K. "Efficient Maximum Likelihood Detection for Communication over Multiple Input Multiple Output Channels". MA thesis. University of Cambridge, 2005.

DOI:10.15774/PPKE.ITK.2015.010

[78] Zhan Guo and P. Nilsson. “Algorithm and implementation of the K-best sphere decoding for MIMO detection”. In: *Selected Areas in Communications, IEEE Journal on* 24.3 (Mar. 2006), pp. 491–503. ISSN: 0733-8716.

[79] Sizhong Chen, Tong Zhang, and Yan Xin. “Relaxed K-Best MIMO Signal Detector Design and VLSI Implementation”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 15.3 (2007), pp. 328–337.

[80] S. Mondal, K.N. Salama, and W.H. Ali. “A novel approach for K-best MIMO detection and its VLSI implementation”. In: *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on.* May 2008, pp. 936–939.

[81] S. Mondal, A. Eltawil, Chung-An Shen, and K.N. Salama. “Design and Implementation of a Sort-Free K-Best Sphere Decoder”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18.10 (Oct. 2010), pp. 1497–1501.

[82] Yi Hsuan Wu, Yu Ting Liu, Hsiu-Chi Chang, Yen-Chin Liao, and Hsie-Chia Chang. “Early-Pruned K-Best Sphere Decoding Algorithm Based on Radius Constraints”. In: *Communications, 2008. ICC ’08. IEEE International Conference on.* May 2008, pp. 4496–4500.

[83] Chung-An Shen and A.M. Eltawil. “A Radius Adaptive K-Best Decoder With Early Termination: Algorithm and VLSI Architecture”. In: *Circuits and Systems I: Regular Papers, IEEE Transactions on* 57.9 (Sept. 2010), pp. 2476–2486. ISSN: 1549-8328.

[84] K.C. Lai, J.J. Jia, and L.W. Lin. “Hybrid Tree Search Algorithms for Detection in Spatial Multiplexing Systems”. In: *Vehicular Technology, IEEE Transactions on* 99 (2011).

[85] J. Jalden, L.G. Barbero, B. Ottersten, and J.S. Thompson. “Full Diversity Detection in MIMO Systems with a Fixed-Complexity Sphere Decoder”. In: *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on.* Vol. 3. Apr. 2007, pp. 49–52.

[86] M.S. Khairy, M.M. Abdallah, and S. E-D Habib. “Efficient FPGA Implementation of MIMO Decoder for Mobile WiMAX System”. In: *Communications, 2009. ICC ’09. IEEE International Conference on.* June 2009, pp. 1–5.

[87] Qi Qi and Chaitali Chakrabarti. “Parallel High Throughput Soft-Output Sphere Decoding Algorithm”. English. In: *Journal of Signal Processing Systems* 68.2 (2012), pp. 217–231. ISSN: 1939-8018.

DOI:10.15774/PPKE.ITK.2015.010

[88] P. Kipfer and R. Westermann. "GPU Gems". In: vol. 2. Addison Wesley Professional, 2005. Chap. 46, pp. 733–746.

[89] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems).* Addison-Wesley Professional, 2005.

[90] K. E. Batcher. "Sorting networks and their applications". In: 1968, pp. 307–314.

[91] Hubert Nguyen. *GPU Gems 3.* First. Addison-Wesley Professional, 2007.

[92] NVIDIA Corporation. *GTX 680 Kepler (GK104) Whitepaper.* 2012.

[93] D Wübben, J Rinas, R Böhnke, V Kühn, and KD Kammeyer. "Efficient algorithm for detecting layered space-time codes". In: *Proceedings of the 4th International ITG Conference on Source and Channel Coding (SCC).* 2002.

[94] B.M. Hochwald and S. ten Brink. "Achieving near-capacity on a multiple-antenna channel". In: *Communications, IEEE Transactions on* 51.3 (Mar. 2003), pp. 389–399. ISSN: 0090-6778.

[95] C. Studer, A. Burg, and H. Bolcskei. "Soft-output sphere decoding: algorithms and VLSI implementation". In: *Selected Areas in Communications, IEEE Journal on* 26.2 (Feb. 2008), pp. 290–300. ISSN: 0733-8716.

[96] N. Felber, W. Fichtner, and A. Burg. "A 50 MBPS 4x4 maximum likelihood decoder for multiple-input multiple-output systems with QPSK modulation". In: *Icecs 2003: Proceedings Of The 2003 10Th Ieee International Conference On Electronics,Circuits And Systems.* Vol. 1. IEEE. Dec. 2003, pp. 332–335.

[97] M.S. Khairy, C. Mehlfuhrer, and M. Rupp. "Boosting sphere decoding speed through Graphic Processing Units". In: *Wireless Conference (EW), 2010 European.* Apr. 2010, pp. 99–104.

[98] Min Li, B. Bougard, Weiyu Xu, D. Novo, L. Van der Perre, and F. Catthoor. "Optimizing Near-ML MIMO Detector for SDR Baseband on Parallel Programmable Architectures". In: *Design, Automation and Test in Europe, 2008. DATE '08.* Mar. 2008, pp. 444–449.

[99] Christoph Studer, Markus Wenk, and Andreas Burg. "VLSI Implementation of Hard- and Soft-Output Sphere Decoding for Wide-Band MIMO Systems". In: *VLSI-SoC: Forward-Looking Trends in IC and Systems Design.* Vol. 373. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2012, pp. 128–154.

DOI:10.15774/PPKE.ITK.2015.010

[100] D. Wubben, D. Seethaler, J. Jalden, and G. Matz. “Lattice Reduction”. In: *Signal Processing Magazine, IEEE* 28.3 (May 2011), pp. 70–91. ISSN: 1053-5888.

[101] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. “Factoring polynomials with rational coefficients”. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[102] Murray R Bremner. *Lattice basis reduction: An introduction to the LLL algorithm and its applications.* CRC Press, 2012.

[103] Martin Seysen. “Simultaneous reduction of a lattice basis and its reciprocal basis”. In: *Combinatorica* 13.3 (1993), pp. 363–376. ISSN: 0209-9683.

[104] Di Wu, Johan Eilert, and Dake Liu. “A programmable lattice-reduction aided detector for MIMO-OFDMA”. In: *Circuits and Systems for Communications, 2008. ICCSC 2008. 4th IEEE International Conference on.* 2008, pp. 293–297.

[105] Susanne Wetzel. “An efficient parallel block-reduction algorithm”. In: *Algorithmic Number Theory.* Ed. by JoeP. Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 323–337. ISBN: 978-3-540-64657-0.

[106] Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli. “State-of-the-art in heterogeneous computing”. In: *Sci. Program.* 18.1 (Jan. 2010), pp. 1–33. ISSN: 1058-9244.

[107] Ravi Kannan. “Improved algorithms for integer programming and related lattice problems”. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing.* ACM. 1983, pp. 193–206.

[108] A.H. Banihashemi and A.K. Khandani. “On the complexity of decoding lattices using the Korkin-Zolotarev reduced basis”. In: *Information Theory, IEEE Transactions on* 44.1 (Jan. 1998), pp. 162–171. ISSN: 0018-9448.

[109] Bettina Helfrich. “Algorithms to construct Minkowski reduced and Hermite reduced lattice bases”. In: *Theoretical Computer Science* 41 (1985), pp. 125–139.

[110] Ravi Kannan. “Minkowski’s Convex Body Theorem and Integer Programming”. In: *Math. Oper. Res.* 12.3 (Aug. 1987), pp. 415–440. ISSN: 0364-765X.

[111] Wen Zhang, Sanzheng Qiao, and Yimin Wei. “HKZ and Minkowski Reduction Algorithms for Lattice-Reduction-Aided MIMO Detection”. In: *Signal Processing, IEEE Transactions on* 60.11 (Nov. 2012), pp. 5963–5976. ISSN: 1053-587X.

DOI:10.15774/PPKE.ITK.2015.010

[112] Phong Q. Nguyen and Brigitte Valle. *The LLL Algorithm: Survey and Applications.* 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 3642022944, 9783642022944.

[113] L. Afflerbach and H. Grothe. "Calculation of Minkowski-reduced lattice bases". English. In: *Computing* 35.3-4 (1985), pp. 269–276. ISSN: 0010-485X.

[114] Phong Q. Nguyen and Damien Stehlé. "Low-dimensional Lattice Basis Reduction Revisited". In: *ACM Trans. Algorithms* 5.4 (Nov. 2009), pp. 1–48. ISSN: 1549-6325.

[115] S. Roger, F. Domene, A. Gonzalez, V. Almenar, and G. Piñero. "An evaluation of precoding techniques for multiuser communication systems". In: *Proc. 7th Int Wireless Communication Systems (ISWCS) Symp.* 2010.

[116] Christoph Windpassinger, Robert FH Fischer, and Johannes B Huber. "Lattice-reduction-aided broadcast precoding". In: *IEEE Trans. Commun.* 52.12 (2004), pp. 2057–2060.

[117] Gilles Villard. "Parallel lattice basis reduction". In: *Papers from the international symposium on Symbolic and algebraic computation.* ISSAC '92. New York, NY, USA: ACM, 1992. ISBN: 0-89791-489-9.

[118] Yixian Luo and Sanzheng Qiao. "A parallel LLL algorithm". In: *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering.* 2011, pp. 93–101.

[119] W. Backes and S. Wetzel. "Parallel Lattice Basis Reduction - The Road to Many-Core". In: *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on.* 2011.

[120] U. Ahmad, A. Amin, Min Li, S. Pollin, L. Van der Perre, and F. Catthoor. "Scalable Block-Based Parallel Lattice Reduction Algorithm for an SDR Baseband Processor". In: *Communications (ICC), 2011 IEEE International Conference on.* 2011.

[121] A. Burg, D. Seethaler, and G. Matz. "VLSI Implementation of a Lattice-Reduction Algorithm for Multi-Antenna Broadcast Precoding". In: *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on.* 2007, pp. 673–676.

[122] L. Bruderer, C. Studer, M. Wenk, D. Seethaler, and A. Burg. "VLSI implementation of a low-complexity LLL lattice reduction algorithm for MIMO detection". In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on.* 2010.

DOI:10.15774/PPKE.ITK.2015.010

[123] Wai Ho Mow. “Maximum likelihood sequence estimation from the lattice viewpoint”. In: *Information Theory, IEEE Transactions on* 40.5 (Sept. 1994), pp. 1591–1600. ISSN: 0018-9448.

[124] Xiaoli Ma, Wei Zhang, and A. Swami. “Lattice-reduction aided equalization for OFDM systems”. In: *Wireless Communications, IEEE Transactions on* 8.4 (Apr. 2009), pp. 1608–1613. ISSN: 1536-1276.

[125] R. Zamir, S. Shamai, and U. Erez. “Nested linear/lattice codes for structured multiterminal binning”. In: *Information Theory, IEEE Transactions on* 48.6 (2002), pp. 1250–1276. ISSN: 0018-9448.

[126] A. Hassibi and S. Boyd. “Integer parameter estimation in linear models with applications to GPS”. In: *Signal Processing, IEEE Transactions on* 46.11 (Nov. 1998), pp. 2938–2952. ISSN: 1053-587X.

[127] R.N. Neelamani, R.G. Baraniuk, and Ricardo de Queiroz. “Compression color space estimation of JPEG images using lattice basis reduction”. In: *Image Processing, 2001. Proceedings. 2001 International Conference on.* Vol. 1. 2001, pp. 890–893.

[128] PhongQ. Nguyen and Jacques Stern. “Lattice Reduction in Cryptology: An Update”. English. In: *Algorithmic Number Theory.* Ed. by Wieb Bosma. Vol. 1838. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 85–112. ISBN: 978-3-540-67695-9.