Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2013

New Parallel Sphere Detector algorithm providing high-throughput for optimal MIMO detection

Csaba M. Józsa[a], Géza Kolumbán[a], Antonio M. Vidal[b], Francisco-Jose Martínez-Zaldívar[c], Alberto González[c]

[a]*Faculty of Information Technology, Pázmány Péter Catholic University, Práter str. 50/A, 1083 Budapest, Hungary*
[b]*Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n, 46022 València, Spain*
[c]*Departamento de Comunicaciones, Universitat Politècnica de València, Camino de Vera s/n, 46022 València, Spain*

Abstract

Multiple–input multiple-output (MIMO) technology has attracted considerable attention in wireless communications, because it offers significant increases in data throughput and link range without additional bandwidth requirements or increased transmit power. The price that has to be paid is the need for the increased computation power. Manycore architectures offer great opportunities for researchers, because the computational performance offered by these devices are in some cases far beyond the general purpose processors. Recent researches have proved that General Purpose Graphics Processing Units (GP-GPUs) are able to solve computationally intensive tasks in a very efficient manner. MIMO detection techniques can vary significantly in complexity and detection performance. Finding the optimal Maximum Likelihood (ML) solution with high throughput was limited by the computational performance. In order to achieve high throughput non-ML algorithms were introduced, having degraded detection performance and lower complexity. In this paper we present a new parallel algorithm, inspired by the Sphere Detector (SD) algorithm, which can efficiently solve the ML detection of the MIMO systems with high throughput on parallel architectures. We also give an overview on how it is possible to map the algorithm onto GP-GPUs, however different parallel architectures are also suitable for adapting the presented algorithm.

Keywords: GPU, MIMO, ML detection, Sphere Detector

1. Introduction

The most important factors implicated in the development of wireless communications are the need for higher link throughput, network capacity and improved reliability. The limiting factors of such systems are usually equipment cost, radio propagation and frequency spectrum. Research in Information Theory has revealed that important improvements can be achieved in data rate when multiple antennas are applied at both the receiver and transmitter side. The key feature of multiple-transmit multiple-receive antenna, i.e. Multiple-Input Multiple-Output (MIMO), systems is the ability to turn multipath propagation, traditionally a pitfall of wireless transmissions, into a benefit for the user. The success of MIMO lies in the fact that the performance of wireless systems is improved by orders of magnitude at no cost of extra spectrum. The MIMO techniques can increase the robustness of wireless communication systems by transmitting different representations of the same data stream (by means of coding) on the different transmit branches, or they can achieve a higher throughput by transmitting independent data streams on

Email addresses: jozsa.csaba@itk.ppke.hu (Csaba M. Józsa), kolumban.geza@itk.ppke.hu (Géza Kolumbán), avidal@dsic.upv.es (Antonio M. Vidal), fjmartin@dcom.upv.es (Francisco-Jose Martínez-Zaldívar), agonzal@dcom.upv.es (Alberto González)

different transmit branches simultaneously and at the same carrier frequency. The price that has to be paid is the increased complexity of the different hardware components and algorithms. The complexity of the detector algorithms used in different receiver structures depend on many factors, such as antenna mapping, channel, coding, etc.

The manycore parallel architectures, such as GP-GPUs or field programmable gate arrays (FPGAs) are getting a prominent role in computational sciences because of their general purpose, high computational performance and cheap price. The trend is that market leading smart phones are using sophisticated GP-GPUs, moreover the importance and usage of high-performance GP-GPU clusters is highly increasing. In several scientific fields, research shows that using these powerful devices, significantly better results can be achieved. In this paper we are presenting an efficient mapping of the SD algorithm onto GP-GPUs, with the purpose of finding the optimal, maximum likelihood (ML) solution of the detection process. Many papers [1], [2], [3] are available in the literature focusing on finding a solution close to ML with a significant decrease in the computational complexity, however our goal is to find the optimal ML solution. The drawback of the imposed condition is that we have to deal with a significantly higher complexity compared to algorithms that are searching for non-optimal solution.

In Section 2 we present the MIMO system model, after that in Section 3 we describe the well known SD algorithm by showing how it is possible to reduce its complexity without degrading the quality of detection. In Section 4 a brief overview is given of the CUDA programming model. Section 5 gives a description about the proposed Parallel Sphere Detector (PSD) algorithm and in Section 6 we present the results achieved by the PSD.

2. System Model

A MIMO system consists of n transmit and m receive antennas. The transmit antennas are sending a complex signal vector $\tilde{s}$ of size n during one symbol period. Assuming rich-scattering and flat-fading channel over one symbol period, the system model is given by the following equation:

$$\tilde{\mathbf{y}} = \tilde{\mathbf{H}}\tilde{\mathbf{s}} + \tilde{\mathbf{v}} \tag{1}$$

where $\tilde{\mathbf{s}} = [\tilde{s}_1, \tilde{s}_2, .., \tilde{s}_n]^T$ is the transmitted symbol vector, where each component is drawn from a complex symbol set $\tilde{\Omega}$, $\tilde{\mathbf{y}} = [\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_m]$ is the received complex symbol vector and $\tilde{\mathbf{v}} = [\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_m]$ is an independent and identically distributed (i.i.d) $CN(0, K)$ circular symmetric complex multivariate random variable where the covariance matrix $K = \sigma_n^2 I_m$, and the entries $\tilde{h}_{ij}$ of the channel matrix $\tilde{H}$ are assumed to be i.i.d zero-mean complex Gaussian variables with unit variance.

The optimal solution for the system model (1) is:

$$\tilde{s}_{ML} = \underset{\tilde{s}\epsilon\tilde{\Omega}^{N_t}}{\arg\min} \|\tilde{y} - \tilde{H}\tilde{s}\|^2. \tag{2}$$

In order to simplify the problem description, we transform the original complex representation of the system model (1) into a real valued system model, at the cost of increasing dimension:

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{v} \tag{3}$$

where $\mathbf{y}_{2\times m,1} = \begin{bmatrix} \Re(\tilde{y}) \\ \Im(\tilde{y}) \end{bmatrix}$, $\mathbf{s}_{2\times n,1} = \begin{bmatrix} \Re(\tilde{s}) \\ \Im(\tilde{s}) \end{bmatrix}$, $\mathbf{v}_{2\times m,1} = \begin{bmatrix} \Re(\tilde{v}) \\ \Im(\tilde{v}) \end{bmatrix}$, and $\mathbf{H}_{2\times m,2\times n} = \begin{bmatrix} \Re(\tilde{H}) & -\Im(\tilde{H}) \\ \Im(\tilde{H}) & \Re(\tilde{H}) \end{bmatrix}$.

For the real valued system the ML metric is:

$$s_{ML} = \underset{s\epsilon\Omega^{2\times N_t}}{\arg\min} \|y - Hs\|^2 \tag{4}$$

where y, H, s, s_{ML} are real valued and Ω is real valued signal set of size Q. From equation (4) it can be seen that the maximum likelihood estimate of the symbol vector is found by solving an integer least-squares (ILS) problem, which is analogous to finding the closest lattice point of lattice $\Lambda = \{Hs : s\epsilon\mathbb{Z}^{N_t}\}$ to a given point y [4], [5]. In lattice theory this problem is often called the closest lattice point search (CLPS) [6], [7]. The exhaustive search implementation of ML decoding, or the enumeration of all lattice points has a complexity that grows exponentially with the number of elements in the signal set Ω or with the number of the antennas, thus the required computational performance would be unattainable. For general lattices the problem was shown to be NP-hard [8]. However, significant complexity reduction can be achieved by exploiting the structure of the lattice (e.g., [9], [10]).

3. The Sphere Detector Algorithm

The fundamental aim of the Sphere Detector algorithm is to restrict the search to lattice points $s \epsilon \Omega^{2n}$ that lie in a certain sphere of radius d around a given vector. Reducing the search space will not affect the performance of the detection, 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.

The complexity analysis of the SD algorithm has been thoroughly investigated by the researchers, for a few good examples refer to [11], [12], [13], [14], etc. The result of investigation is that the exact solution of the decoding process outperforms even the best heuristics. It has been shown that the complexity of the sphere detector algorithm is directly proportional to the number of explored lattice points. The search space is highly influenced by the chosen sphere radius. Choosing a small radius may result in an empty sphere, but the choice of a too large radius may lead to an increased complexity. It is known from lattice theory that the covering radius of the lattice is the smallest radius of spheres placed on the lattice points that will cover the entire space. For an arbitrary lattice Λ the search of the covering radius require a number of steps that grows exponentially [15] with the dimension of the lattice, i.e. the number of transmit and receive antennas.

The definition of the center point $\hat{s}$ of the sphere $S(\hat{s}, d)$ is usually defined as the unconstrained least-squares solution $\hat{s} = Wy$, where $W = (H^T H)^{-1} H^T$ is the pseudo-inverse of the channel matrix. In order to take advantage of the search space reduction, a good enumeration strategy is needed. However, the individual checking of each lattice point, if it is included in the given sphere, is not efficient solution because it is equal with the exhaustive search of the lattice. The benefit of the SD algorithm lies in the enumeration of the lattice points.

Without any loss of generality we assume that $n = m$ and the channel matrix has full rank. Furthermore we assume perfect channel state information (CSI) at the receiver side. Taking the unconstrained least-squares solution $\hat{s}$ of the real system shown in (3) the ML solution (4) can be determined by a different metric as follows:

$$\|y - Hs\|^2 = (s - \hat{s})^T H^T H (s - \hat{s}) \tag{5}$$

Applying the QR factorization to the real channel matrix $H = QR$, the ML solution can be formulated as follows:

$$\begin{aligned} s_{ML} &= \arg\min_s \|y - Hs\|^2 = \arg\min_s (s - \hat{s})^T H^T H (s - \hat{s}) \\ &= \arg\min_s (s - \hat{s})^T (QR)^T (QR)(s - \hat{s}) = \arg\min_s \|R(s - \hat{s})\|^2 \end{aligned} \tag{6}$$

where matrix Q is orthogonal and matrix R is upper triangular. Then the condition $s \in S(\hat{s}, d)$ can be formulated as:

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

Exploiting the upper triangular property of matrix R instead of enumerating all the possible symbol combinations a recursion can be defined based on the dependency hierarchy of the terms. In order to give a deeper insight, let $s_i^m \triangleq (s_i, s_{i+1}, \cdots, s_m)^T$ denote the last $m - i + 1$ components of the vector s and let $M(s_i^m) = \sum_{j=i}^{m} \left| \sum_{k=j}^{m} r_{jk}(s_k - \hat{s}_k) \right|^2$ define the metric of the partial symbol vector s_i^m. Having a bottom up selection of the possible symbols for the symbol vector, in every iteration one symbol $s_i \in \Omega$ is being selected and added to the partial symbol vector $s_{i+1}, \cdots, s_m$. As a result s_i^m is defined and the metric $M(s_i^m)$ can be applied. If the conditions are met, namely $M(s_i^m) < d^2$, a new symbol s_{i-1} has to be selected for the next dimension, contrary the previously chosen symbol s_i is discarded and a new symbol has to be chosen from the signal set. A solution is found if a complete symbol vector s_1^m is satisfying the condition $M(s_1^m) < d^2$. The solution with the smallest metric is the ML solution. Choosing a small radius may not include the solution, so the process has to be restarted with a higher radius. A detailed description of the SD algorithm can be found in [16].

It is important to point out that the Sphere Detector algorithm is equivalent to a bounded tree search. The continuous change of vector s_i^m is analogous of a depth-first tree traversal process. The partial symbol vectors s_i^m can be regarded as tree nodes at level i, the full symbol vectors s_1^m are the leaves of the tree and the weight of each node is defined by the symbol vector metric $M(s_i^m)$. Figure 1 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 four receive antennas were used.

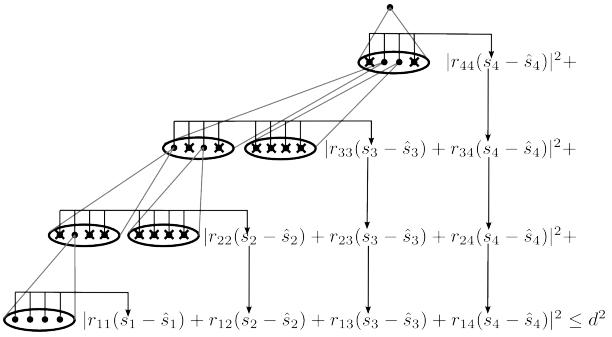

Fig. 1: Sphere decoding - tree pruning

4. The CUDA programming model

The programming of the GP-GPU devices became popular when Nvidia launched the Compute Unified Device Architecture (CUDA) parallel programming model. Traditional CPUs are able to execute a few threads very quickly, however GP-GPUs, due to their parallel architecture, are able to support the execution of thousands of threads with a slower speed. At the time of writing the most advanced GP-GPU is the GeForce GTX 690 built on the new Kepler GK104 [17] architecture. Thus the device parameters given in the following are referring to the previously mentioned GP-GPU.

An extensive description of CUDA programming and optimization techniques can be found in [18]. The main entry points of a GP-GPU programs are called *kernels*. These kernels are executed N times in parallel by N different *CUDA threads*. CUDA threads are grouped in *thread blocks*. In the Kepler architecture the number of threads in a thread block are limited to 1024, however multiple equally-shaped thread blocks can be executed by a kernel. A *grid* is a collection of thread blocks. Either the threads in the thread block, either the thread blocks in the grid can have a one-dimensional, two-dimensional, or three-dimensional ordering. The maximum x dimension of the grid is $2^{32} - 1$. The ordering is motivated by the problem to be solved, thus when launching a kernel the grid size, the number threads in a thread block and the ordering dimensions have to be defined by the programmer.

The cooperation between the threads is realized with the help of multiple memory spaces. In CUDA the following hierarchy of memory levels is defined:

1. Private memory - each thread has its own private memory and this memory is only visible to the respective thread. The private memory usually maps to the registers thus this is the fastest memory. However, there are situations when large data structures are mapped to local memory. The maximum number of registers per thread for the GK104 architecture is 63.
2. Shared memory - each thread block has its own shared memory and this memory is only visible to all threads in the block. The shared memory is an on-chip memory with low latency and high bandwidth. Equally-sized memory modules, called banks, serves the different memory requests. By avoiding bank conflicts high throughput can be achieved.
3. Global memory - the communication between thread blocks is possible by using the global memory. The global memory is an off-chip memory, has high latency, thus whenever is possible its use should be avoided or specific patterns (coalescing) should be used. Global memory can be accessed by the host and it is persistent across kernel launches that were initiated by the same application.
4. Constant and texture memory - these are read-only off-chip cached memories. They are visible for all of the thread blocks.

There are situations when there is a need to wait for the results of other threads. Therefore, threads within a block may be *synchronized*. In order to continue the execution each thread has to reach the barrier. There is no similar possibility to synchronize thread blocks in a grid, however, when a kernel finishes its execution it can be regarded as a global synchronization of the thread blocks.

The Nvidia GP-GPU architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). The blocks 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 multiprocessor is defined as the number of active threads divided by the maximum number of threads. The maximum number of blocks running simultaneously on a multiprocessor is 16, however this can be limited by the maximum number of warps, the registers or the amount of shared memory used per SM. In order to concurrently execute hundreds of threads,

the SMs employ the *Single-Instruction, Multiple-Thread (SIMT)* architecture. Groups of 32 threads, called *warps*, are executed together. A warp executes one common instruction at a time. In the case of data-dependent branching, the warp will serially execute each branch path taken. In order to achieve full efficiency divergence should be avoided.

5. The Parallel Sphere Detector algorithm

Many signal processing algorithms used in complex communications systems have a parallel structure that can be implemented on GP-GPU platform in a very efficient manner. It was shown in Section 4 that computations can be divided in several thread blocks and each thread block has multiple threads running in parallel. The analysis and identification of different levels of parallelism is crucial, the better the mapping the better the system performance. In case of PSD algorithm two levels of parallelism can be defined, (i) a higher level, system level parallelism, (ii) a lower level, algorithm level parallelism.

Next generation wireless communication standards will take the advantage of MIMO-OFDM [19] technology, because of its the high spectral efficiency that can be achieved. The parallel nature of the wireless system makes possible the processing of multiple symbols at once. In order to keep the processing units busy and to exploit fully the advantage of GP-GPU resources, a mapping between the received signal symbol vectors and the thread blocks is defined, that belongs to the system level parallelism. Consequently, the number of thread blocks launched for a kernel is equal to the number of simultaneously processed symbol vectors and each thread block detects only one symbol vector. The system level parallelism was defined as a bounding between a received symbol vector and a thread block, however the algorithm level parallelism is the effective distribution of the work for the threads in a thread block in order to complete the detection process.

Section 3 concluded that the SD algorithm can be regarded as a branch and bound tree search problem. Khairy et al. showed in [3] that significant speed-ups can be achieved by executing multiple sphere decoders simultaneously with the conventional sequential algorithm. However, to achieve peak performance on the GP-GPU it is mandatory to redesign the sequential algorithm using several effective design patterns and taking the limitations imposed by the architecture into considerations. The mapping in this case becomes more complex. During the design process of the PSD algorithm the key motivating factors are as follows:

1. Ensure that we do not have to restart the algorithm with a higher radius. In order to achieve this, we will define the initial radius $d^2 = \infty$. This initial condition assures that there is no need for time consuming algorithms. (An alternative solution for the initial radius could be the Zero Forcing (ZF) radius. The ZF radius is equal to the distance between the Babai estimate, which is obtained by slicing each element of $\hat{s}$ to the closest integer, and the received vector y.)
2. Having a radius of infinity means that the SD algorithm will be equal with an exhaustive search. Finding a leaf node in the tree it is possible to adjust the radius to the metric of the found leaf node $d^2 = M(s_1^m)$. Thus the search space can be significantly reduced. The problem that has to be solved, is to find a small metric leaf node as fast as possible.
   Recall, the two fundamental tree search algorithms are the breadth-first (BF) search and the depth-first (DF) search techniques [20]. The SD algorithm is based on a DF strategy. Finding the least cost leaf in the case of a large tree could take excessively long time. Considering that the initial radius $d = \infty$, one of the benefits of the DF search is that it will find a leaf really fast, moreover the amount of memory used will be proportional to the number of levels of the tree. The drawback is the high possibility of finding a leaf node with a big metric, thus the search space will not decrease efficiently. In the case of the BF search the result will be the least cost node, but the memory requirements may be to excessive and the criteria of finding a leaf node as fast as possible are not met.
   Lai et al. in [21] have examined different hybrid tree search algorithms and they could achieve significantly lower complexity with a moderate increase in the memory need. The speed-up was achieved because the decoding process started with a BF search and it was continued as a DF search based on the extracted nodes branch metric. Thus a leaf node could be found in a more efficient manner. A hybrid searching scheme seems that will efficiently solve the imposed conditions.
3. Using the radius update strategy, the expected value of the symbol vectors analyzed is decreased significantly. The task is to make the radius update available for every thread in a thread block.
4. In order to get fast detection, the access of the global memory has to be minimized. Hence all the data needed for the detection should be placed in a shared memory and registers. Unfortunately, the size of the shared memory and number of registers are limited. The goal is to take into consideration the memory limitation in such a way that the occupancy of the GP-GPU will not be affected by the amount of the shared memory or registers.

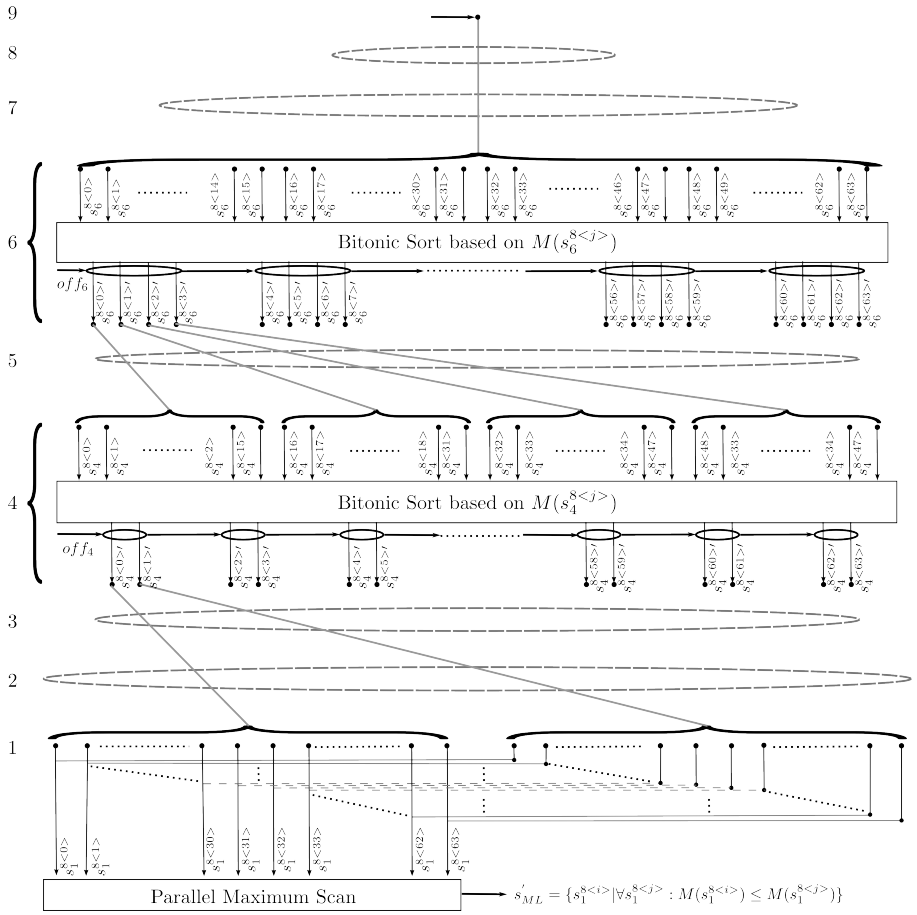

Fig. 2: PSD algorithm structure for a 4x4 MIMO system with 16-QAM modulation

5.1. Implementing PSD algorithm on CUDA

The system level parallelism is exposed by mapping each received symbol vector y to a thread block in the grid. Having a high number of thread blocks is important, because it keeps the GPU utilized. The algorithm level parallelism is more difficult, but significant speed-up can be achieved by redesigning the sequential algorithm. In order to define the parallel implementation of the sphere decoding algorithm, the following parameters are defined:

1. **tt** - total number of threads, used in a thread block to decode one received symbol vector;
2. $\mathbf{t_{id}^{k}}$ - thread with identifier **k** in a given thread block;
3. $\mathbf{lvl_{nr}}$ - PSD algorithm parameter specifying the number of levels used for BF search strategy (i.e. 1,2,3,...);
4. $\mathbf{lvl_x}$ - levels assigned for BF search;
5. $\mathbf{paths_{lvl_x}}$ - the number of symbol vectors selected on level $\mathbf{lvl_x}$ for simultaneous process with DF search strategy;
6. $\mathbf{exp_{lvl_x}}$ - the number of simultaneously expanded nodes on level $\mathbf{lvl_x}$;
7. $\mathbf{s_{lvl_x}^{N<j>}}$ - symbol vector on level $\mathbf{lvl_x}$, where $< \mathbf{j} >$ is an optional parameter showing that the symbol vector is placed in the buffer $\mathbf{buf_{lvl_x}}$ at index **j**;
8. $\mathbf{M(s_{lvl_x}^{N<j>})} \equiv \mathbf{M_{lvl_x}^{<j>}}$ - the path metric of symbol vector $\mathbf{s_{lvl_x}^{N<j>}}$, where $< \mathbf{j} >$ is an optional parameter showing that the symbol vector's path metric is placed in the buffer $\mathbf{bufPM_{lvl_x}}$ at index **j**;
9. $\mathbf{buf_{lvl_x}}$ - shared memory buffer containing the symbol vectors $\mathbf{s_{lvl_x}^{N<j>}}$ for level $\mathbf{lvl_x}$, where $0 \leq j < \mathbf{exp_{lvl_x}}$;
10. $\mathbf{bufPM_{lvl_x}}$ - shared memory buffer containing the path metric of the symbol vectors $\mathbf{M(s_{lvl_x}^{N<j>})}$ for level $\mathbf{lvl_x}$, where $0 \leq j < \mathbf{exp_{lvl_x}}$.

The parameters that define the mapping of the algorithm are $\mathbf{lvl_{nr}}$, $\mathbf{lvl_x}$ and $\mathbf{paths_{lvl_x}}$. These parameters influence the memory usage of a thread block, the speed of reaching a leaf node, the metric of the first leaf node, the iterations needed for finding the optimal solution, namely the performance of the algorithm. It is important to notice that different antenna configurations and modulation order needs different parameters, because the tree depth and branching factor are changing. The basic definitions and relations are as follows:

1. The root level is equal to: $\mathbf{lvl_0} = 2 \cdot m + 1$. The number of receive antennas is multiplied by 2, because of the complex to real transformation given by Equation 3.
2. The branching factor is equal to the square root of the modulation order.
3. $\mathbf{lvl_x} > \mathbf{lvl_{x+1}}$ and the last level is always $\mathbf{lvl_{lvl_{nr}}} = 1$
4. The root node is always expanded, thus $\mathbf{paths_{lvl_0}} = 1$
5. $\mathbf{exp_{lvl_x}} = \mathbf{paths_{lvl_{x-1}}} * |\Omega|^{(\mathbf{lvl_{x-1}} - \mathbf{lvl_x})}$

Table 1a enumerates a few valid parameter sets of different configurations for demonstrative purposes.

Table 1: Various PSD algorithm configurations

(a) Valid system configurations for the PSD algorithm.

Configuration	1	2	3	4	5	6
Antennas	2x2	2x2	2x2	4x4	4x4	4x4
Symbol set size	2	4	8	4	8	8
$\mathbf{lvl_{nr}}$	2	2	2	3	4	4
$\mathbf{lvl_0}$	5	5	5	9	9	9
$\mathbf{lvl_1}$	2	2	2	6	7	7
$\mathbf{lvl_2}$	1	1	1	4	6	6
$\mathbf{lvl_3}$	0	0	0	1	3	2
$\mathbf{lvl_4}$	0	0	0	0	1	1
$\mathbf{paths_{lvl_0}}$	1	1	1	1	1	1
$\mathbf{paths_{lvl_1}}$	4	4	4	4	2	2
$\mathbf{paths_{lvl_2}}$	0	0	0	2	3	3
$\mathbf{paths_{lvl_3}}$	0	0	0	0	4	4
$\mathbf{exp_{lvl_1}}$	8	64	256	64	64	64
$\mathbf{exp_{lvl_2}}$	8	16	32	64	16	16
$\mathbf{exp_{lvl_3}}$	0	0	0	128	768	12288
$\mathbf{exp_{lvl_4}}$	0	0	0	0	256	32
$\sum_{x=1}^{\mathbf{lvl_{nr}}} \mathbf{exp_{lvl_x}}$	16	80	288	256	1104	12400

(b) Analyzed systems for the PSD algorithm.

Antennas	2x2	2x2	2x2	4x4	4x4	4x4
Symbol set size	2	4	8	2	4	8
tt	8	16	64	32	64	64
$\mathbf{lvl_{nr}}$	2	2	2	2	3	4
$\mathbf{lvl_0}$	5	5	5	9	9	9
$\mathbf{lvl_1}$	2	3	3	4	6	7
$\mathbf{lvl_2}$	1	1	1	1	4	5
$\mathbf{lvl_3}$	0	0	0	0	1	3
$\mathbf{lvl_4}$	0	0	0	0	0	1
$\mathbf{paths_{lvl_0}}$	1	1	1	1	1	1
$\mathbf{paths_{lvl_1}}$	4	1	1	4	4	1
$\mathbf{paths_{lvl_2}}$	0	0	0	0	1	1
$\mathbf{paths_{lvl_3}}$	0	0	0	0	0	1

The amount of shared memory used by a thread block is proportional to the sum $\sum_{x=1}^{\mathbf{lvl_{nr}}} \mathbf{exp_{lvl_x}}$ of the expanded nodes at different level. Configurations 1, 2 and 3 are having the same parameters $\mathbf{lvl_x}$, $\mathbf{paths_{lvl_x}}$, but because the modulation order is different a significant change in memory requirements can be observed. Comparing configurations 3 and 4 it can be seen that the different antenna configurations and modulation order have almost the same memory requirements. Configurations 5 and 6 prove that a slight change in the parameters may change significantly the required memory size. The excessive use of shared memory can degrade occupancy, thus the number of thread blocks running on one SM will decrease and the achieved throughput will be reduced. The amount of shared memory that can be assigned to one thread block keeping the occupancy high is different. Thus finding the optimal parameters depends not only on the MIMO system configuration but also on the type of GP-GPU chosen. This contribution does not consider the methods used to find the optimal parameters.

Parameters of different system configurations studied here are given in Table 1a. Figure 2 shows the PSD schematic for Configuration 4. The levels referred below are identified on the left side of the figure.

The detection starts at $\mathbf{lvl_0} = 9$, which is the root node of the tree.

Level 7 and 8 are not analyzed. These levels can be skipped because the GP-GPU offers a high parallel computing capability and a high number of threads can be launched simultaneously. If at a given level the number of threads launched in one thread block is equal to that of the expanded nodes then these nodes can be processed simultaneously.

The first node expansion happens at $\mathbf{lvl_1} = 6$. At this level $\mathbf{exp_{lvl_1}} = 64$ nodes have to be expanded. During the node expansion both the partial symbol vector and the metric associated to the node in the tree have to be determined.

Next stage applies the Bitonic Sort method [22]. The sorting key is the metric $\mathbf{M}(\mathbf{s_6^{8<j>}})$ of the partial symbol vectors. This stage is mainly motivated by the condition of finding a leaf node with small metric. When moving towards to the next level $\mathbf{lvl_2} = 4$ the $\mathbf{paths_{lvl_1}} = 4$ best metric nodes are selected from the previous level $\mathbf{lvl_1} = 6$. Recall, the smaller the metric, the faster the convergence. The "greedy behavior" of Bitonic Sort method ensures that a leaf with small metric is found.

The selection, expansion and sorting discussed above are repeated until the the last level $\mathbf{lvl_3} = 1$ is reached. Upon reaching the last level, the symbol vector characterized by the best metric has to be found. At level $\mathbf{lvl_1} = 1$, instead of the parallel sorting design pattern approach, a maximum search based on the parallel reduce pattern [23] is applied because of its much lower computing cost.

Then the best metric symbol vector s_1^8 is compared to the one found in the previous iteration, if there is any. After the comparison the sphere radius is adjusted and the search for the best ML candidate is continued in a similar manner.

Algorithm 1 shows the new PSD proposed here. The pseudo-code gives an insight how it the parallelism of the GP-GPU architecture is exploited during the detection process. To make the pseudo-code more understandable the details of optimization of CUDA kernel are not shown.

The skeleton of the algorithm is formed by three procedures. The procedure called *VARIABLES DEFINITION AND INITIALIZATION* is the main entry point of the algorithm. The macro definition of the algorithm parameters $\mathbf{lvl_{nr}}$, $\mathbf{lvl_x}$, $\mathbf{paths_{lvl_x}}$ is symbolical in the sense that these parameters are defined outside the kernel.

After the parameter setup, the *PROCESS* is started on $\mathbf{lvl_1} = 9$. In the algorithm, *PROCESS* adjusts the offset and level parameters. Note, this procedure controls the flow of the tree in the vertical direction. The tasks of procedure *EXPAND AND EVALUATE* are to (i) prepare the partial symbol vectors for each level, (ii) calculate the metric of each symbol vector, (iii) perform sorting based on the calculated path metrics and (iv) find the candidate for the best solution.

6. Performance results

In order to measure the performance of the PSD algorithm several systems have been analyzed. The parameter settings for the different system configurations are given in Table 1b. As stated before, finding the optimal parameter configurations is not subject of this paper. The parameter selection was motivated by:

1. The number of expanded nodes at each level is equal to the number of threads working on one thread block.
2. Minimizing the expanded paths at a specific level by choosing small values for $\mathbf{paths_{lvl_x}}$.

The measurements were done in a GeForce GTX 690 GP-GPU based on the Kepler GK104 architecture. A major issue of ML detection is its varying complexity and this is mostly caused by channel matrices with high condition numbers. As a result the running time of different kernels can be significantly different. When measuring the average throughput of the PSD algorithm in order to get realistic results we tested each system configuration by simulating 20000 channels and 1200 symbol vectors in each channel. The detection throughput results are presented in Figure 3.

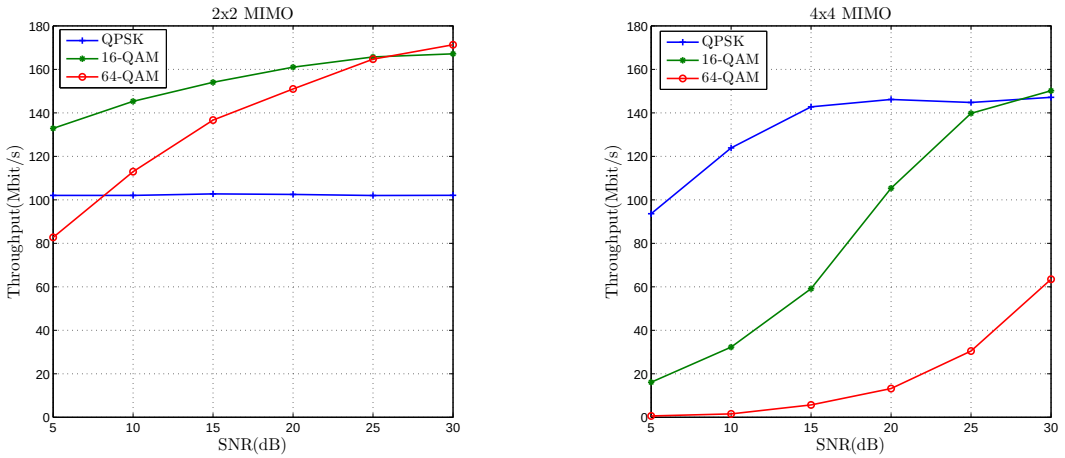

Fig. 3: PSD algorithm average throughput (a) 2x2 MIMO; (b) 4x4 MIMO.

The throughput of the new parallel algorithm mapped onto GP-GPU proposed in this contribution is compared with that of known from the literature in Table 2. We took into consideration only those results which were focusing on finding the optimal solution. It can be seen that the proposed PSD algorithm proposed here outperforms each of them.

Algorithm 1 The PSD algorithm GP-GPU kernel pseudo-code

Require: $\hat{s}, R$
  procedure VARIABLES DEFINITION AND INITIALIZATION
    define lvl_nr, lvl_x, $paths_x$ as macros for $x \in \{0...lvl_{nr}\}$
    statically allocating shared memory for buf_{lvl_x}, $bufPM_{lvl_x}$ for $x \in \{0...lvl_{nr}\}$
    shared $M(s_{ML}) \leftarrow \infty$, shared $s_{ML} \leftarrow \{\}$, shared $d^2 \leftarrow \infty$
    PROCESS(lvl_1,$off_{lvl_0} \leftarrow 0$)
  end procedure
  procedure PROCESS(lvl_x,$off_{lvl_{x-1}}$)
    if $off_{lvl_{x-1}} < exp_{lvl_{x-1}}$ **then**
      EXPAND AND EVALUATE
      $off_{lvl_{x-1}} \leftarrow off_{lvl_{x-1}} + paths_{lvl_{x-1}}$
      return PROCESS($lvl_{x+1}, off_{lvl_x} \leftarrow 0$)
    else
      if $x > 1$ **then**
        $off_{lvl_{x-1}} \leftarrow 0$
        return PROCESS($lvl_{x-1}, off_{lvl_{x-2}}$)
      else
        Transfer s_{ML} to global memory
        return
      end if
    end if
  end procedure
  procedure EXPAND AND EVALUATE
    Select corresponding path $s_{lvl_{x-1}}^{N<l>}$ from buffer $buf_{lvl_{x-1}}$ with the help of $off_{lvl_{x-1}}$
    if $M(s_{lvl_{x-1}}^{N<l>}) < d^2$ **then**
      Determine partial symbol vector $s_{lvl_x}^{lvl_{x-1}-1<j>}$ for the previously selected path
      Merge selected path $s_{lvl_{x-1}}^{N<l>}$ with partial symbol vector $s_{lvl_x}^{lvl_{x-1}-1<j>}$ resulting in $s_{lvl_x}^{N<j>}$
      Store $s_{lvl_x}^{N<j>}$ in buf_{lvl_x} at index j
      Compute path metric for the merged symbol vector $M(s_{lvl_x}^{N<j>})$
      Store $M(s_{lvl_x}^{N<j>})$ in $bufPM_{lvl_x}$ at index j
    else
      Store ∞ in $bufPM_{lvl_x}$ at index j
    end if
    syncthreads
    if $lvl_x \neq 1$ **then**
      Bitonic sort buf_{lvl_x} based on the corresponding path metric buffer $bufPM_{lvl_x}$
    else
      Find the best path metric M' in $bufPM_{lvl_x}$ and the associated symbol vector $s_1^{N'}$ with a parallel max reduction
      if $M' < M(s_{ML})$ **then**
        $s_{ML} = s_1^{N'}$
        Update radius $d^2 \leftarrow M(s_{ML})$
      end if
    end if
    syncthreads
  end procedure

Table 2: Throughput comparison of the SD algorithm

Reference	[24]	[25]	[26]	[27]	PSD	PSD
Antennas	4x4					
Modulation	16-QAM	QPSK	16-QAM	16-QAM	16-QAM	QPSK
BER performance	Full-ML					
Technology	ASIC	ASIC	ASIC	FPGA	GPU	GPU
Throughput	38 Mbps	50 Mbps	73 Mbps @SNR = 20 dB	81.5 Mbps @SNR = 20 dB	105 Mbps @SNR = 20 dB	146 Mbps @SNR = 20 dB

The results of our investigations shows that ML detection can achieve high throughput when is being mapped on a GP-GPU. In order to achieve this high throughput we proposed a new algorithm, which can be easily adapted for parallel architectures, because the parameters can adjust the memory requirements and the extent of the parallelism, furthermore we presented an effective mapping of the proposed PSD algorithm onto a GeForce GTX 690 GP-GPU. As stated in Section 1 with careful design and implementation and due to the high computational performance of the GP-GPUs, heavy signal detection algorithms can be solved with high speed.

Acknowledgments

The support of the grants TÁMOP-4.2.1.B-11/2/KMR-2011-0002, TÁMOP-4.2.2/B-10/1-2010-0014 at Pázmány Péter Catholic University, PAID-05-11-2733 at Universitat Politècnica de València, PROMETEO/2009/013 from Generalitat Valenciana and CICYT TEC2009-13741 from the Spanish Government is gratefully acknowledged.

References

[1] L. Barbero, J. Thompson, Fixing the Complexity of the Sphere Decoder for MIMO Detection, Wireless Communications, IEEE Transactions on 7 (6) (2008) 2131 –2142. doi:10.1109/TWC.2008.060378.
[2] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, A. Vidal, Fully Parallel GPU Implementation of a Fixed-Complexity Soft-Output MIMO Detector, Vehicular Technology, IEEE Transactions on 61 (8) (2012) 3796 –3800. doi:10.1109/TVT.2012.2210576.
[3] M. Khairy, C. Mehlfuhrer, M. Rupp, Boosting sphere decoding speed through Graphic Processing Units, in: Wireless Conference (EW), 2010 European, IEEE, 2010, pp. 99–104.
[4] M. Damen, H. El Gamal, G. Caire, On maximum-likelihood detection and the search for the closest lattice point, Information Theory, IEEE Transactions on 49 (10) (2003) 2389–2402.
[5] J. H. Conway, N. J. A. Sloane, E. Bannai, Sphere-packings, lattices, and groups, Springer-Verlag, Inc., New York, NY, USA, 1987.
[6] A. Murugan, H. El Gamal, M. Damen, G. Caire, A unified framework for tree search decoding: rediscovering the sequential decoder, Information Theory, IEEE Transactions on 52 (3) (2006) 933–953.
[7] E. Agrell, T. Eriksson, A. Vardy, K. Zeger, Closest point search in lattices, Information Theory, IEEE Transactions on 48 (8).
[8] D. Micciancio, S. Goldwasser, Complexity of lattice problems: a cryptographic perspective, The Kluwer international series in engineering and computer science, Kluwer Academic, 2002.
[9] U. Fincke, M. Pohst, Improved Methods for Calculating Vectors of Short Length in a Lattice, Including a Complexity Analysis, Mathematics of Computation 44 (170) (1985) pp. 463–471.
[10] C. P. Schnorr, M. Euchner, Lattice basis reduction: Improved practical algorithms and solving subset sum problems, Mathematical Programming 66 (1994) 181–199, 10.1007/BF01581144.
[11] H. Vikalo, B. Hassibi, On the sphere-decoding algorithm II. Generalizations, second-order statistics, and applications to communications, Signal Processing, IEEE Transactions on 53 (8) (2005) 2819–2834.
[12] B. Hassibi, H. Vikalo, On the sphere-decoding algorithm I. Expected complexity, Signal Processing, IEEE Transactions on 53 (8).
[13] J. Jalden, B. Ottersten, On the complexity of sphere decoding in digital communications, Signal Processing, IEEE Transactions on 53 (4) (2005) 1474 – 1484. doi:10.1109/TSP.2005.843746.
[14] J. Fink, S. Roger, A. Gonzalez, V. Almenar, V. Garciay, Complexity assessment of sphere decoding methods for MIMO detection, in: Signal Processing and Information Technology (ISSPIT), 2009 IEEE International Symposium on, 2009, pp. 9 –14. doi:10.1109/ISSPIT.2009.5407544.
[15] P. van Emde-Boas, Another NP-complete partition problem and the complexity of computing short vectors in a lattice, 1981.
[16] E. Viterbo, J. Boutros, A universal lattice code decoder for fading channels, Information Theory, IEEE Transactions on 45 (5) (1999) 1639 –1642. doi:10.1109/18.771234.
[17] NVIDIA Corporation, GTX 680 Kepler (GK104) Whitepaper, `http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf` (2012).
[18] NVIDIA Corporation, CUDA C Programming Guide, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/` (2012).
[19] G. Stuber, J. Barry, S. McLaughlin, Y. Li, M. Ingram, T. Pratt, Broadband MIMO-OFDM wireless communications, Proceedings of the IEEE 92 (2) (2004) 271 – 294. doi:10.1109/JPROC.2003.821912.
[20] D. E. Knuth, The art of computer programming, volume 3: (2nd ed.) sorting and searching, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
[21] K. Lai, J. Jia, L. Lin, Hybrid Tree Search Algorithms for Detection in Spatial Multiplexing Systems, Vehicular Technology, IEEE Transactions on (99) (2011) 1–1.
[22] M. Pharr, R. Fernando, GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems), Addison-Wesley Professional, 2005.
[23] H. Nguyen, GPU Gems 3, 1st Edition, Addison-Wesley Professional, 2007.
[24] D. Garrett, L. Davis, S. ten Brink, B. Hochwald, G. Knagge, Silicon complexity for maximum likelihood MIMO detection using spherical decoding, Solid-State Circuits, IEEE Journal of 39 (9) (2004) 1544–1552.
[25] A. Burg, N. Felber, W. Fichtner, A 50 Mbps 4 times;4 maximum likelihood decoder for multiple-input multiple-output systems with QPSK modulation, in: Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003 10th IEEE International Conference on, Vol. 1, 2003, pp. 332 – 335 Vol.1. doi:10.1109/ICECS.2003.1302044.
[26] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, H. Bolcskei, VLSI implementation of MIMO detection using the sphere decoding algorithm, Solid-State Circuits, IEEE Journal of 40 (7) (2005) 1566–1577.
[27] X. Huang, C. Liang, J. Ma, System architecture and implementation of MIMO sphere decoders on FPGA, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16 (2) (2008) 188–197.