Noname manuscript No.
(will be inserted by the editor)

Parallel SUMIS Soft Detector for Large MIMO Systems on Multicore and GPU

Carla Ramiro · M. Ángeles Simarro · Alberto Gonzalez · Antonio M. Vidal

Received: date / Accepted: date

Abstract The number of transmit and receive antennas is an important factor that affects the performance and complexity of a MIMO system. A MIMO systems with very large number of antennas is a promising candidate technology for next generations of wireless systems. However, the vast majority of the methods proposed for conventional MIMO system are not suitable for large dimensions. In this context, the use of High Performance Computing (HPC) systems, such us multicore CPUs and Grapfhics Processing Units (GPUs) has become attractive for efficient implementation of parallel signal processing algorithms with high computational requirements. In the present work two practical parallel approaches of the *Subspace Marginalization with Interference Suppression* (SUMIS) detector for large MIMO systems have been proposed. Both approaches have been evaluated and compared in terms of performance and complexity with other detectors for different system parameters.

Keywords Large MIMO systems · SUMIS · high order constellation · GPU · low-complexity detection

Carla Ramiro
Institute on Robotics and Information and Communication Technologies, Universitat de València
E-mail: cramiro@gmail.com

M. Ángeles Simarro
Institute of Telecommunications and Multimedia Applications, Universitat Politècnica de València
E-mail: mdesiha@iteam.upv.es

Alberto Gonzalez
Institute of Telecommunications and Multimedia Applications, Universitat Politècnica de València
E-mail: agonzal@dcom.upv.es
Antonio M. Vidal
Department of Information Systems and Computation, Universitat Politècnica de València
E-mail: avidal@dsic.upv.es

1 Introduction

Multiple-Input Multiple-Output (MIMO) systems provide significant capacity improvement using multiple antennas at both sides of digital communication systems. This technology has become essential for wireless communications and has been incorporated into many communication wireless standards. An emerging research area are so-called Large MIMO systems, often referred to as massive MIMO systems. It can be defined as those systems that use very large number of antennas, e.g., one hundred ore more . The price to pay is increased complexity and energy consumption at both ends. Particularly the MIMO detection problem is generally computationally very expensive to deal with. Thus, an adequate balance between efficiency and complexity is critical, especially in large MIMO systems [1] and large sizes constellation.

The optimal detector, which solves the MIMO detection problem optimally, computes the log-likelihood ratios (LLRs) values exactly and holds prohibitively high computational complexity, which grows with the size of the signal constellation and the number of antennas. In the above context, several detectors and exhibit different trade-offs between complexity and performance have been recently proposed. "*Single Tree Search*" (STS) [2] and "*Repeated Tree Search*" (RTS) [3] algorithms are the most common detectors which achieve the max-log approximation exactly. Sub-optimal max-log algorithms reduce the complexity at the expense of a certain performance loss. On the other hand, "*Partial Marginalization*" (PM) [4] and SUMIS [5] algorithms represent an intermediate approach between the optimal detector and its max-log approximation. The SUMIS algorithm offers a good trade-off between exact and approximate computation of the LLR values and a given complexity. Even so, SUMIS detector can be the bottleneck for the overall system performance if large number of antennas or high order constellations are used.

This paper aims to reduce the computational cost of the SUMIS method, not only from a theoretical point of view, but through its scalable and versatile implementation for efficient processing thereof e.g. multicore processors and Graphic Processing Units (GPUs). This allows to guarantee the SUMIS detection performance in large MIMO systems with higher throughput. In [6] a parallel implementation based on multicore processors was proposed. In the present paper a parallel version based on GPU is presented and more results with respect to the implementation based on multicore processors are provided. Furthermore, the performance in terms of Bit Error Rate (BER) and Mutual Information (MI) has been evaluated for large MIMO system.

The rest of the paper is organized as follows. The MIMO system model and a brief review of the SUMIS detector are given in Section 2. The details of the proposed parallel algorithms on multicore and GPU are described in Section 3. The performance evaluation and the implementation results are given in Section 4. Finally, the conclusions are presented in Section 5.

2 Background

2.1 System Model

Let us consider a complex-valued MIMO system model, using n_T transmit and n_R receive antennas with $n_T \leq n_R$. The information bits are encoded using an error-correcting code and then interleaved and mapped to symbols. Each symbol s_j contains $k = \log_2(M)$ encoded and interleaved bits and is taken independently from the M-ary constellation Ω. The corresponding bits are denoted by $s_{j,b}$, where the indices refer to the *bth* bit associated with the *jth* symbol. The relation between the received vector, $\mathbf{y}_c \in \mathbb{C}^{n_R}$ and the associated transmitted symbol vector, $\mathbf{s}_c \in \mathbb{C}^{n_T}$, can be expressed as $\mathbf{y}_c = \mathbf{H}_c\mathbf{s}_c + \mathbf{v}_c$, where $\mathbf{H}_c \in \mathbb{C}^{n_R \times n_T}$denotes a fading channel matrix with independent elements $h_{j,i} \sim \mathcal{N}(0,1)$ and it is assumed to be perfectly known by the receiver. Vector $\mathbf{v}_c \sim \mathcal{N}(\mathbf{0}, \frac{N_o}{2}\mathbf{I})$ denotes an additive Gaussian noise (AWGN). Since separable complex-value constellation can be considered (M-QAM), we can easily transform the $(n_R \times n_T)$-dimensional complex model into an equivalent $(2n_R \times 2n_T)$-dimensional real-valued representation as described in [6]. Thereby, the real model can be described as

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

The real model is assumed throughout the rest of the paper. Let $\sqrt{M}$-PAM with $P_M = \{-\sqrt{M}+1, \cdots, -1, 1, \cdots, \sqrt{M}-1\}$ be the real-valued representation of a M-QAM constellation where $\Omega = \{a + bj : a, b \in P_M\}$.

At the receiver side, for each of the encoded and interleaved bit $s_{j,b}$ the demodulator computes soft information in form of LLR values which expresses how likely is the hypothesis that the $s_{j,b}$ bit is equal to 1 or 0. Assuming equal a priori probabilities and using Bayes' theorem, the LLR values can be rewritten as

$$L_{j,b} = \log \frac{\sum_{\mathbf{s} \epsilon \chi^1_{j,b}} \exp(-\frac{1}{N_0}\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2)}{\sum_{\mathbf{s} \epsilon \chi^0_{j,b}} \exp(-\frac{1}{N_0}\|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2)}, \tag{2}$$

where $\chi^u_{j,b}$ denotes the set of possible transmitted sequences for which $s_{j,b}$ bit is equal to u. The computational cost of (2) increases exponentially with n_T and polynomially with M [5]. Thus, the exact MIMO detection scheme becomes prohibitive. The most common approach to cope with this limitation is the max-log approximation [7].

A new approach to compute (2) is proposed in other works [4][5] as an alternative to max-log approximation. The main idea is to employ the following partitioning model, which is based on (1),

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{v} = [\overline{\mathbf{H}} \quad \widetilde{\mathbf{H}}] \quad [\overline{\mathbf{s}}^T \quad \widetilde{\mathbf{s}}^T]^T + \mathbf{v} = \overline{\mathbf{H}}\overline{\mathbf{s}} + \widetilde{\mathbf{H}}\widetilde{\mathbf{s}} + \mathbf{v} \tag{3}$$

where $\overline{\mathbf{H}} \in \mathbb{R}^{2n_R \times n_s}$, $\widetilde{\mathbf{H}} \in \mathbb{R}^{2n_R \times (2n_T - n_s)}$, $\overline{\mathbf{s}} \in \Omega^{n_s}$ and $\widetilde{\mathbf{s}} \in \Omega^{2n_T - n_s}$ for fixed $n_s \in 1, \cdots, 2n_T$. The partitioned model carries out intrinsically an optimal permutation of the columns of $\mathbf{H}$ that determines $\overline{\mathbf{H}}$ and $\widetilde{\mathbf{H}}$ [5].

2.2 SUMIS algorithm review

SUMIS [5] algorithm is composed by two main stages and employs the partitioning model (3). This partition uses a permutation based on $\mathbf{H}^T\mathbf{H}$. Here we give a brief revision of SUMIS algorithm.

Stage I: The algorithm begins with the partitioned model (3) denoting the new model as

$$\overline{\mathbf{y}} = \overline{\mathbf{H}}\overline{\mathbf{s}} + \mathbf{e} \tag{4}$$

where $\mathbf{e} = \widetilde{\mathbf{H}}\widetilde{\mathbf{s}} + \mathbf{v}$ is a Gaussian stochastic vector $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ with $\mathbf{Q} = \widetilde{\mathbf{H}}\widetilde{\mathbf{H}}^T + \frac{N_o}{2}\mathbf{I}$. We compute the approximate $\lambda_{j,b}$ LLR using the next operator $\|\mathbf{x}\|^2_{\mathbf{Q}} \triangleq \mathbf{x}^T\mathbf{Q}^{-1}\mathbf{x}$, as

$$\lambda_{j,b} = \log \frac{\sum_{\overline{\mathbf{s}}\epsilon\chi^0_{j,b}} \exp(-\frac{1}{2}\|\mathbf{y} - \overline{\mathbf{H}}\overline{\mathbf{s}}\|^2_Q)}{\sum_{\overline{\mathbf{s}}\epsilon\chi^1_{j,b}} \exp(-\frac{1}{2}\|\mathbf{y} - \overline{\mathbf{H}}\overline{\mathbf{s}}\|^2_Q)}. \tag{5}$$

Stage I is performed for all bits $b = 1, \cdots, k$ in all symbols $j = 1, \cdots, n_T$.
Stage II: In the second stage, the LLR values are computed again over a new model given by

$$\mathbf{y}' \triangleq \mathbf{y} - \widetilde{\mathbf{H}}\mathbb{E}\{\widetilde{\mathbf{s}}|\mathbf{y}\} \approx \overline{\mathbf{H}}\overline{\mathbf{s}} + \mathbf{n}', \tag{6}$$

where $\mathbb{E}\{\widetilde{\mathbf{s}}|\mathbf{y}\}$ is the conditional expected value of vector $\widetilde{\mathbf{s}}$, and $n' \backsim \mathcal{N}(\mathbf{0}, \mathbf{Q}')$ with $\mathbf{Q}' \triangleq \widetilde{\mathbf{H}}\widetilde{\mathbf{\Upsilon}}\widetilde{\mathbf{H}}^T + \frac{N_o}{2}\mathbf{I}$, being $\widetilde{\mathbf{\Upsilon}}$ the conditional covariance matrix of $\widetilde{\mathbf{s}}$. Hence, the refined LLR values can be computed as

$$L_{j,b} \approx \log \frac{\sum_{\overline{\mathbf{s}}\epsilon\chi^0_{j,b}} \exp(-\frac{1}{2}\|\mathbf{y}' - \overline{\mathbf{H}}\overline{\mathbf{s}}\|^2_{Q'})}{\sum_{\overline{\mathbf{s}}\epsilon\chi^1_{j,b}} \exp(-\frac{1}{2}\|\mathbf{y}' - \overline{\mathbf{H}}\overline{\mathbf{s}}\|^2_{Q'})}. \tag{7}$$

Note that the processing per bit can be performed in parallel.

3 Proposed Parallelization

The use of the last generation of High Performance Computing (HPC) systems such as multi-core CPUs and Graphics Processing Units (GPUs) has become attractive for the efficient implementation of parallel signal processing algorithms with high computational requirements [8][9]. The implementation of advanced algorithms able to use both architectures is crucial in MIMO research, since it allows to fully exploit the capabilities of the modern computer architectures and to reduce the response time of computationally expensive problems.

The SUMIS algorithm computes the $\lambda_{j,b}$ and $L_{j,b}$ values using (5) and (7) respectively, where the number of elements in the two summations over $\overline{\mathbf{s}}$ is M^{n_s}. Therefore, SUMIS algorithm has to calculate the expression $\exp(-\frac{1}{2}\|\mathbf{y} - \overline{\mathbf{H}}\overline{\mathbf{s}}\|^2_Q)$ a number of times M^{n_s} to achieve the total $\lambda_{j,b}$ or $L_{j,b}$ values in each stage. For this reason almost the 98% of the total signal detection time is consumed to compute these terms. Therefore, the parallelization is focused on reducing the computational cost of the $\exp(-\frac{1}{2}\|\mathbf{y}-\overline{\mathbf{H}}\overline{\mathbf{s}}\|_Q^2)$ terms in (5) and (7).

In [5], Appendix A, an optimization of the SUMIS algorithm is explained. Following this approach the next expression can be derived by simple matrix manipulations:

$$\|\mathbf{y}-\overline{\mathbf{H}}\overline{\mathbf{s}}\|_Q^2 = (\mathbf{A}\mathbf{y}-\overline{\mathbf{s}})^T\mathbf{B}(\mathbf{A}\mathbf{y}-\overline{\mathbf{s}}) + \delta_1 - \delta_2, \tag{8}$$

By renaming: $\mathbf{C} = \mathbf{Q}^{-1}\overline{\mathbf{H}}$, $\mathbf{D} = \overline{\mathbf{H}}^T\mathbf{Q}^{-1}$, $\mathbf{B} = \mathbf{D}\overline{\mathbf{H}}$, $\mathbf{A} = \mathbf{B}^{-1}\mathbf{D}$, $\delta_1 = \mathbf{y}^T\mathbf{Q}^{-1}\mathbf{y}$ and $\delta_2 = \mathbf{y}^T\mathbf{C}\mathbf{A}\mathbf{y}$.
The terms δ_1 and δ_2 in (8) do not depend on $\overline{\mathbf{s}}$. Equation (8) can be computed using the Linear Algebra PACKage (LAPACK) [10].

For the multicore implementation we are using the Intel Math Kernel Library (MKL) [11], which is composed of several optimized math routines and it is optimized specifically for Intel processors. For the GPU implementation we use the cuBLAS Library [12], which is a GPU-accelerated version of the complete standard BLAS library.

3.1 Multi-threaded implementation

During the last years, the main microprocessors manufacturers such as Intel or AMD have been focused on developing faster and smarter chips. Their purpose is to get maximum performance with minimal consumption by integrating multiple processing units (called cores) onto a single processor. Thus, these cores can process simultaneously multiple tasks although at a lower clock rate.

For the parallelization of the SUMIS algorithm we assume a MIMD computer (i.e. multiple instruction, multiple data) with shared memory. This system has p processors which share a common central memory. The MKL multi-threaded version uses the OpenMP application programming interface [13] to distribute the computation among the different cores. We can parallelize each MKL function by selecting the desired number of threads before calling the function. However, the performance of these functions greatly depends on the size of the channel matrix. Even though we are considering channel matrix sizes larger than usual used in MIMO communication systems, they are still too small to fully exploit the MKL performance. Previous results show how increasing the number of threads barely manage to accelerate the sequential version. For this reason we have chosen to parallelize at a higher level. To clarify how it has been carried out the distribution of tasks, we have presented in Algorithm 1 the complete pseudo-code related to the SUMIS detector. The OpenMP pragma ***omp parallel for*** in line 7 distributes the for loop iterations among the threads and therefore the SUMIS processing is performed per jth-symbol in parallel.

```
   Input   : $\mathbf{H}$, $\mathbf{y}$, $n_s \in \{1, \ldots, n_T\}$
   Output: Log Likelihood Ratios ($\mathbf{L}$)
 1 Calculate $\mathbf{H}^T\mathbf{H}$
 2 /* Stage I */
 3 for $j = 1, \ldots, n_T$ do
 4     Decide upon a partitioning in (3) $\mathbf{H} = [\overline{\mathbf{H}} \quad \widetilde{\mathbf{H}}]$ based on $\mathbf{H}^T\mathbf{H}$
 5 end
 6 Set $\widetilde{\boldsymbol{\Upsilon}} = \mathbf{I}$
 7 #pragma omp parallel for
 8 for $j = 1, \ldots, n_T$ do
 9     Variance matrix $\mathbf{Q} = \widetilde{\mathbf{H}}\widetilde{\boldsymbol{\Upsilon}}\widetilde{\mathbf{H}}^T + \frac{N_o}{2}\mathbf{I}$
10     for $p = 1, \ldots, M^{n_s}$ do
11         $\omega[j]_p = \exp(-\frac{1}{2}\|\mathbf{y} - \overline{\mathbf{H}}\bar{\mathbf{s}}_{:,p}\|^2_Q)$ such as (8)
12     end
13     for $b = 1, \ldots, k$ do
14         $\lambda_{j,b} = \log \frac{\sum_{p:\bar{\mathbf{s}}_{:,p} \epsilon \chi^0_{j,b}} w[j]_p}{\sum_{p:\bar{\mathbf{s}}_{:,p} \epsilon \chi^1_{j,b}} w[j]_p}$
15     end
16 end
17 /* Stage II */
18 for $j = 1, \ldots, n_T$ do
19     $\mathbb{E}\{s_j|\mathbf{y}\} \triangleq \sum_{\mathbf{s} \in \Omega} s \prod_{b=1}^{k} \frac{1}{1+e^{(-2s_{j,b}+1)\lambda_{j,b}}}$
20     Suppress the interfering vector $\mathbf{y}' \triangleq \mathbf{y} - \widetilde{\mathbf{H}}\mathbb{E}\{\widetilde{\mathbf{s}}|\mathbf{y}\}$
21     Calculate $\widetilde{\boldsymbol{\Upsilon}} = \mathbb{E}\{\text{diag}(\widetilde{\mathbf{s}})^2|\mathbf{y}\} - \mathbb{E}\{\text{diag}(\widetilde{\mathbf{s}})|\mathbf{y}\}^2$
22 end
23 Repeat steps 7 to 15 with [$\mathbf{y}'$, $\mathbf{Q}'$, $L_{j,b}$] instead of [$\mathbf{y}$, $\mathbf{Q}$, $\lambda_{j,b}$]
```

Algorithm 1: SUMIS pseudo-code OpenMP implementation.

3.2 CUDA implementation

Compute Unified Device Architecture (CUDA) [14] is a software programming model that exploits the massive computation potential offered by GPUs. A GPU can have multiple stream multiprocessors (SM) with a certain number of pipelined cores each. In contrast to the multicores, the GPUs follow a single-instruction multiple-threads (SIMT) programming model, that is, a single set of instructions is executed on different data sets. In this model, the programmer defines the kernel function that contains a set of common operations. At runtime, the kernel is called from the main central processing unit (CPU) and spawns a large number of threads blocks, which is called grid. Each thread block contains multiple threads, usually up to 1024, and all the blocks within a grid must share the same size. Blocks and grids can be one-dimensional, two-dimensional or tri-dimensional but they must not exceed a certain size stated in the GPU's specifications. Each thread can select a set of data using its own unique ID and executes independently the kernel function on the selected set of data. Threads of the same block can share data between them by using the shared memory. However, threads of different blocks are independent and should use global memory to share data once all threads have finished running the full kernel. At is said above, the $\exp(-\frac{1}{2}\|\mathbf{y}-\overline{\mathbf{H}}\bar{\mathbf{s}}\|_Q^2)$ terms have been parallelized for the LLR calculation as in equation (8) using the equivalent cuBLAS [12] functions.

Batched functions are intended to be used for matrices of small sizes where the launch overhead is a significant factor. For small sizes, this kind of functions improves significantly the performance compared to making calls to its corresponding cublas<t>*name* routine. However, matrix-vector and vector-vector functions are not yet available for batched computation. For this reason we have implemented three functions in CUDA. These functions explained bellow.

– cudaDgemvBatched: The kernel performs the matrix-vector multiplications of an array of matrices and vectors $z[n] = A[n]*y[n]$ for $n = 0, ..., n\_batch-1$. Each A variable is a $(n_{rows} \times n_{cols})$ matrix, z is a $(1 \times n_{rows})$ vector and y is a $(n_{cols} \times 1)$ vector. Each thread determines the element to process by using its unique ID using: its block index (blockIdx.x), its thread index (threadIdx.x) and the number of threads per block (blockDim.x). A bidimensional grid configuration with $N_{Bx} = N_B$, $N_{By} = N_B$ blocks per dimension has been considered for kernels. The number of blocks N_B depends on the number of threads per dimension, which are denoted by N_{tx} and N_{ty}, respectively. Then the value of N_B is obtained as

$$N_B = \left\lceil \sqrt{\frac{n_{th}}{N_{tx} \times N_{ty}}} \right\rceil \tag{9}$$

where n_{th} is the number of CUDA threads, in this case $n_{th} = n\_batch = n_T$. Each thread calculates its unique identifier (n) and executes the kernel on the selected set of data independently.
– cudaDgevmvBatched: The kernel performs the vector-matrix-vector multiplications of an array of matrices and vectors $\delta[n] = z[n] * A[n] * y[n]$ for $n = 0, ..., n\_batch - 1$. Each A variable is a $(n_{rows} \times n_{cols})$ matrix, z is a $(1 \times n_{rows})$ vector, y is a $(n_{cols} \times 1)$ vector and $delta$ is a $(1 \times n\_batch)$ vector. This function is used to compute δ_1 and δ_2 values in (8). In this case the number of threads is $n_{th} = n\_batch = n_T$.
– cudaDgevmvBatched_v2: This function is similar to the previous *cudaDgevmvBatched* kernel. In these case the number of threads is $n_{th} = n_T \times M^{n_s}$ since we need to compute $(\mathbf{Ay}-\bar{\mathbf{s}})^T\mathbf{B}(\mathbf{Ay}-\bar{\mathbf{s}})$ for each $\bar{\mathbf{s}} \in \Omega^{n_s}$. All vectors $\mathbf{v} = \mathbf{Ay} - \bar{\mathbf{s}}$ are previously computed. This function performs the vector-matrix-vector multiplications of an array of matrices and vectors $\omega[n]_p = \exp(-\frac{1}{2}(v[n]'_{:,p} * B[n] * v[n]_{:,p} + \delta_1[n] - \delta_2[n]))$ for $n = 0, ..., n_T - 1$ and $p = 0, ..., M^{n_s} - 1$.

Other operations in (5) and (7) show low complexity compared to the previous terms and does not have a parallel pattern (see lines 13-15 Alg.1). Moreover, the so-called *warp divergence* [14] plays an important role in the performance. In CUDA, threads are executed in warps of 32 threads, with all threads in the warp executing the same instruction at the same time. However, in this part different threads in a warp need to cope with different tasks. The GPU hardware is not capable of executing if and else statements at the same time. CUDA serializes the different execution paths to generate correct code. For this reason, these operations have not been parallelized.

4 Results

In the following sub-sections, we compare the performance of the proposed multicore and GPU implementation of the SUMIS soft-output detector (with $n_s = 3$) in terms of bit error rate, mutual information, computational complexity and speedup. The transmitted symbols are assumed to be independent and uniformly distributed. The transmitted bits are encoded using a 1/2 LDPC code of codeword size 648 bits, which is available from http://www.csl.cornell.edu /vstuder/software ldpc.html and implements a LDCPC code from the IEEE 802.11n wireless LAN standard and the sum-product algorithm option has been chosen as the decoding option. A machine with one Nvidia Tesla K40C GPUs and two multicore Intel processors has been employed. Each multicore is an Intel Xeon CPU E5-2697 at 2.70 GHz with 12 cores per CPU. The most relevant features and specifications of the GPU are listed in Table 4.

Number of stream multiprocessors	15
Number of cores	2880
Clock Rate	0.75GHz
Global Memory	12 GB
Memory Clock Speed	6GHz

Table 1 Nvidia Tesla K40c especifications

4.1 BER performance, Mutual Information and Complexity

In this paper, we evaluate the performance and complexity of SUMIS for higher number of antennas and constellation order. The use of exact and max-log algorithms, such as RTS or STS, become prohibitive for this number of antennas. On the other hand, the " *Fully Parallel Fixed-complexity Sphere decoding*" (FPFSD) [15] for moderate number of antennas achieves almost max-log behavior at reasonable computing cost, for this reason it has been chosen for comparison. A linear detector with reduced complexity such as Soft "*Minimum Mean Square Error*" (MMSE) has also been chosen.

BER performance is represented for SUMIS, Soft MMSE and FPFSD algorithms in Fig. 1. The curves show clearly that the SUMIS detector performs much better than the other algorithms for 64-QAM and 256-QAM constellations. It is noteworthy that SUMIS algorithm exhibits an improvement up to 5 dB in SNR with respect to the FPFSD.

In addition, we also provide a detector comparison in terms of mutual information [16]. In Fig. 2 the minimum SNR required to achieve a given MI

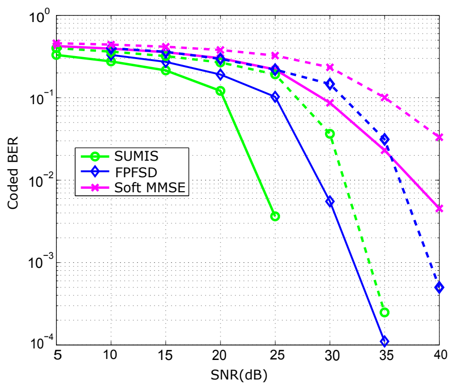

Fig. 1 BER as a function of SNR for the $N_T = N_R = 200$ in (1 with the LDPC code of rate 1/2. The dashed curves show the performance for a 64-QAM constellation and the solid curves show the performance for a 256-QAM constellation.

is referred as the "minimum SNR" for that MI. This Figure represents that minimum depending on the number of antennas. The results in Fig. 2 clearly show that the SUMIS detector achieves a MI equal to 1 at lower SNR. It's worth noting that the "minimum SNR" scales more linearly with the increase in overall antennas for the SUMIS algorithm than for the other algorithms. The behavior is the same for higher constellation orders.

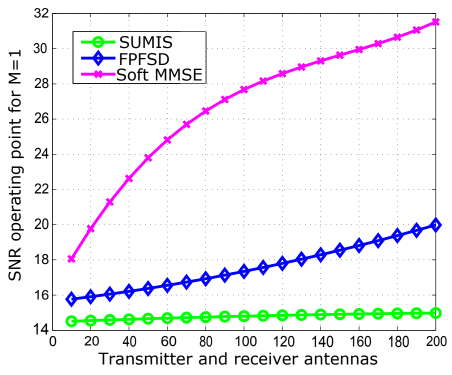

Fig. 2 Minimum SNR required to achieve a value of 1 in the Mutual Information parameter. The Mutual Information is calculated between the LLR values obtained after decoding and the transmitted coded bits. The simulation have been carried out for a 16-QAM constellation and the LDPC code of rate 1/2.

The performance for the three different algorithms has been illustrated by Figs. 1-2. On the other hand, the computational cost is represented in Fig. 3 in terms of FLOPS. Fig. 3 curves represent the number of FLOPS depending on the number of antennas for a 256-QAM system. The bottleneck of the computational cost in the three analyzed algorithms is due to the number of antennas of the system ($\mathcal{O}(N_T^3)$ for the three algorithms). A variation in the constellation order affects only slightly the results. Fig. 3 illustrates that the computational cost of SUMIS shows the same order of magnitude as a linear method for Large MIMO systems.

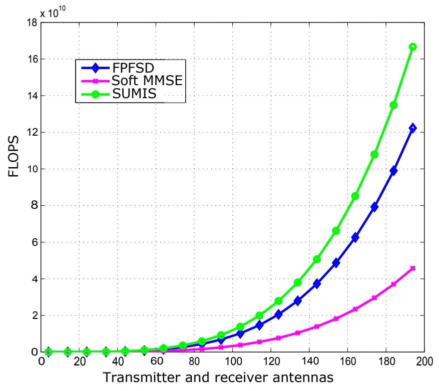

Fig. 3 FLOPS as a function of the number of antennas with the LDPC code of rate 1/2 and a 256-QAM constellation.

4.2 Speedup

We measure the execution time of the detection, with different number of antennas and constellation sizes to evaluate the SUMIS efficiency of the proposed parallel prototypes. The speedup (S_P) has been computed as the ratio between the time execution of the *"reference SUMIS algorithm"* (T_S), and the execution time of the OpenMP and the GPU versions (T_P). The MKL implementation with the *-mkl=sequential* compiler option was selected for the *"reference SUMIS algorithm"*.

Tables 2 and 3 illustrate the execution time and speedup of the different configurations. It can be observed that, generally, the higher system sizes achieve the higher speedup for the OpenMP version. The soft detection can be accelerated up to 9 times.

The experimental GPU measurements in Tab. 3 show how CUDA version fails to accelerate the sequential version when the number of antennas and the constellation order is small. This is due to the lower complexity of the detector with these parameters. This problem gradually disappears when the complexity of the detection stage increases, for example when the number of transmitter antennas n_T is 200 and M is 1024. In this scenario, where very large MIMO arrays are considered, the CUDA detector is up to 4 times faster than the sequential version.

T_P \ S_P	M=16	M=64	M=256	M=1024
$n_T = 8$	0.04 \ 1.50	0.08 \ 1.50	0.21 \ 2.67	1.35 \ 3.19
$n_T = 24$	0.18 \ 1.33	0.25 \ 1.52	0.63 \ 2.70	2.67 \ 4.92
$n_T = 48$	0.67 \ 1.85	1.06 \ 1.47	1.79 \ 2.35	5.57 \ 4.88
$n_T = 100$	2.19 \ 5.85	3.06 \ 4.36	4.24 \ 4.55	13.08 \ 5.13
$n_T = 200$	17.00 \ 9.33	18.80 \ 8.53	21.86 \ 7.53	33.26 \ 7.83

Table 2 Multicore Execution Times in milliseconds (T_P) and Speedup (S_P) for the OpenMP SUMIS algorithm with different number of antennas and different QAM constellation.

It is interesting to note how both parallel versions allow to boost the performance of the system with a speed comparable to a low-complexity linear detector such as MMSE. For example, with $n_T = 200$ and 256-QAM, the complexity of SUMIS ($\approx 16 \cdot 10^{10}$) is 4 times higher than the MMSE detector ($\approx 4 \cdot 10^{10}$) (see Fig. 3). However, by using the parallel implementations, the complexity of the SUMIS detector is reduced to $\approx 2 \cdot 10^{10}$ for the OpenMP version and $\approx 4 \cdot 10^{10}$ for the GPU version. Thus, we can detect signals with a similar and even higher throughput than the MMSE detector with much better BER.

T_P \ S_p	M=16	M=64	M=256	M=1024
$n_T = 8$	0.27 \ 0.22	0.25 \ 0.48	0.51 \ 1.10	1.52 \ 2.82
$n_T = 24$	0.18 \ 1.33	0.46 \ 0.82	0.63 \ 2.70	1.52 \ 8.64
$n_T = 48$	1.29 \ 0.96	1.49 \ 1.05	2.36 \ 1.77	3.50 \ 7.76
$n_T = 100$	5.85 \ 2.19	6.26 \ 2.13	7.30 \ 2.64	21.10 \ 3.18
$n_T = 200$	39.30 \ 4.04	40.30 \ 3.97	42.31 \ 3.89	70.80 \ 3.68

Table 3 GPU Execution Times in milliseconds (T_P) and Speedup (S_P) for the CUDA SUMIS algorithm with different number of antennas and different QAM constellation.

5 Conclusion

In this paper, we have proposed and analyzed two parallel SUMIS Soft Detector implementations. Furthermore, the SUMIS algorithm has been compared with the FPFSD and MMSE detectors, considering very large MIMO systems reaching up to 200 transmitter/receiver antennas and up to 1024-QAM constellations. This comparison shows the robust, versatile and scalable behavior of the SUMIS algorithm. This detector behaves much better than the other detectors in terms of BER, achieving up to 5 dB improvement in SNR compared to FPFSD.

Currently, we are still far from reaching the speeds required by the IEEE 802.11n wireless LAN standard, which makes the implementation unfeasible for on line use. However, these approaches allow to reduce considerably the complexity of the simulation of large MIMO systems with scalable quasi optimal soft detector, opening the door to foresee new technology performance faster than by conventional simulation.

Acknowledgment

This work has been partially supported by the Spanish MINECO Grant RACHEL TEC2013-47141-C4-4-R, the PROMETEO FASE II 2014/003 project and FPU AP-2012/71274.

References

1. F. Rusek, D. Persson, B.K. Lau, E. G. Larsson, T.L. Marzetta, O. Edfors, and F. Tufvesson. Scaling up MIMO: Opportunities and challenges with very large arrays. *IEEE Signal Proc. Magazine*, 30(1):40–60, Jan. 2013.
2. C. Studer, A. Burg, and H. Bölcskei. Soft-output sphere decoding: algorithms and VLSI implementation. *IEEE J. Sel. Areas Commun.*, 26(2):290–300, 2008.
3. Renqiu Wang and Georgios B Giannakis. Approaching MIMO channel capacity with reduced-complexity soft sphere decoding. *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, 3:1620–1625, 2004.
4. D. Persson and E. G. Larsson. Partial Marginalization soft MIMO detection with higher order constellations. *IEEE Trans. on Signal Proccesing*, 59(1):453–458, Jan. 2011.
5. M. Čirkić and E. G. Larsson. SUMIS: Near-optimal soft-in soft-out MIMO detection with low and fixed complexity. *Signal Processing, IEEE Transactions on*, 62(12):3084–3097, June 2014.
6. Alberto Gonzalez C. Ramiro, M. Ángeles Simarro and Antonio M. Vidal. Parallel SUMIS Soft Detector for MIMO systems on Multicore. *Proceedings of the 17th Internationa Conference on Computational and Mathematical Methods in Science and Engineering*, V:1729–1736, 2017.
7. B.M. Hochwald and S. ten Brink. Achieving Near-Capacity on a Multiple-Antenna Channel. *IEEE Trans. Commun.*, 51:389–399, 2003.
8. Kaipeng Li, Bei Yin, Michael Wu, Joseph R Cavallaro, and Christoph Studer. Accelerating massive MIMO uplink detection on GPU for SDR systems. *Circuits and Systems Conference (DCAS), 2015 IEEE Dallas*, pages 1–4, 2015.
9. Di Wu, Johan Eilert, and Dake Liu. Implementation of a high-speed MIMO soft-output symbol detector for software defined radio. *Journal of Signal Processing Systems*, 63(1):27–37, 2011.
10. Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, A McKenney, and D Sorensen. LAPACK users' guide. 9, 1999.
11. online at: `https://software.intel.com/en-us/articles/mkl-reference-manual`. *Intel MKL Reference Manual (2015) [Online].*
12. online at: `http://docs.nvidia.com/cuda/cublas`. *cuBLAS Documentation (2015) [Online].*
13. Leonardo Dagum and Rameshm Enon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.
14. online at: `https://developer.nvidia.com/cuda-toolkit`. *CUDA Toolkit Documentation, Version 7.5 (2015) [Online].*
15. Sandra Roger, Carla Ramiro, Alberto Gonzalez, Vicenc Almenar, and Antonio M Vidal. Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector. *IEEE Transactions on Vehicular Technology*, 61(8):3796–3800, 2012.
16. Martin Senst, Gerd Ascheid, and Helge Lüders. Performance evaluation of the markov chain monte carlo MIMO detector based on mutual information. *Communications (ICC), 2010 IEEE International Conference on*, pages 1–6, 2010.