GPU-Based Low-Precision Detection Approach for Massive MIMO Systems

Adel Dabah[1(✉)], Hatem Ltaief[1], Zouheir Rezki[2], Slim Alouini[1], and David Keyes[1]

[1] Division of Computer, Electrical, and Mathematical Sciences and Engineering, King Abdullah University of Science and Technology, Thuwal, Jeddah 23955, Saudi Arabia
{Adel.Dabah.1,Hatem.Ltaief,slim.alouini,David.Keyes}@kaust.edu.sa

[2] University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064, USA
zrezki@ucsc.edu

Abstract. Massive Multiple-Input Multiple-Output (M-MIMO) uses hundreds of antennas in mobile communications base stations to increase the amount of transmitted data and the number of connected devices in 5G and beyond. However, M-MIMO systems increase the complexity of recovering the transmitted data (detection phase). To address this challenge, we leverage low-precision arithmetic in recent NVIDIA GPUs to improve the latency/scalability/accuracy of M-MIMO detection. We propose a GPU tree-based detection algorithm that aggregates multiple tree levels and formulates the computation as a matrix multiplication operation followed by a square-norm calculation and sorting (reduction) phase. This process is repeated until reaching the last level of the detection tree. The obtained results show near-optimal data detection with a 10× speedup compared to a two-socket 28-core IceLake CPU implementation. We further deploy low-precision arithmetic operations. We show that moving from single-precision 32-bit floating-point arithmetic (FP32) to half-precision 16-bit representation (FP16) does not affect the accuracy performance while translating into an additional 1.7× speedup. In addition, exploiting 8-bit integer representation results in an acceptable error rate degradation that can be compensated by increasing the number of aggregated levels. In addition, we propose a multi-GPU version that computes the matrix-multiplication operation of subsequent iterations in parallel. This latter operation represents more than 80% of the elapsed time for dense constellations. Results with four A100 GPUs show an additional 2.3× relative speedup compared to our single GPU version. The achieved accuracy/scalability balance may accelerate the deployment of this technology and promote low-precision GPU computations within the wireless communication community.

Keywords: GPU MIMO detection · Low-Precision Arithmetic

A. Bhatele et al. (Eds.): ISC High Performance 2023, LNCS 13948, pp. 144–163, 2023.
https://doi.org/10.1007/978-3-031-32041-5_8

1 Introduction

GPU accelerators enable an increase in computational power by lowering the arithmetic precision. This paper demonstrates the gains achievable by using GPUs with various arithmetic precision to meet the requirements of Next-Generation mobile communication networks in general and Massive Multiple-Input Multiple-Output (M-MIMO) detection in particular. M-MIMO technology is a key enabling technology for 5G and 6G mobile communication networks. It uses hundreds of antennas to send and receive data [7,11]. However, when increasing the number of antennas, the signal detection phase, which estimates the transmitted data, becomes a bottleneck, with an exponential complexity in the number of transmit-antennas for optimal detection. In this context, our goal is to speed up this phase using multiple GPUs with various arithmetic precision. Two main categories of detection methods exist, i.e., linear and nonlinear algorithms. On the one hand, linear detection algorithms operate under real-time constraints but fail to estimate the transmitted data correctly due to noise. On the other hand, nonlinear algorithms, e.g., the Sphere Decoder (SD) [6,8], give an excellent estimation of the transmitted data but require significant execution time. Nonlinear algorithms operate on a search tree that models all possible combinations of the transmitted data. Each path is defined by a set of symbols (data), from which a distance from the received signal can be calculated. The detection goal is to find the path with the shortest distance representing the originally transmitted data.

In this paper, we introduce a low-precision multi-level approach. It iteratively extends one path with several symbols representing the best combination in terms of distance within a window until we reach a complete path (solution). At each iteration, the algorithm combines successive levels within a window and computes all distances via a matrix-matrix multiplication exploiting tensor core capabilities in recent NVIDIA GPUs. The matrix shape is short and wide in dimensions representing the number of levels and all possible paths in a window. We then calculate the square norm and launch a sorting (reduction) phase to select the best extension. By increasing the number of aggregated levels, we improve the accuracy, but this comes at the price of higher complexity, there being a trade-off between complexity and accuracy. To mitigate the complexity and maintain good accuracy, we first exploit low-precision arithmetic (i.e., FP16 and INT8) and engage NVIDIA tensor cores with fast matrix engines. We report results on A100 GPU and achieve a 10× speedup compared to multicore CPU implementation on a two-socket 28-core Intel IceLake CPU. In addition, exploiting low-precision gives an additional 1.7× speedup without impacting the accuracy in terms of error rate. To further reduce the complexity, we propose a multi-GPU version that improves the complexity by reducing the matrix multiplication time, representing more than 80% of the global execution time for dense constellations. The idea is to overlap all matrix multiplication operations performed during the detection process since they are entirely independent and can be processed in an embarrassingly parallel fashion using multiple GPUs. This breaks the inherent sequential behavior of the detection phase, which results

in an additional 2.3× improvement. Overall, we achieved up to 40× relative speedup compared to our multi-CPU implementation.

The rest of the paper is structured as follows. Section 2 introduces basic mobile communication concepts. Section 3 reviews the literature on high-performance MIMO processing. Section 4 presents the system model. Section 5 describes the details of our multi-level approach and its implementation. Results and discussions about the complexity and performance are given in Sect. 7. Finally, Sect. 8 concludes this paper and highlights our future plans.

2 Brief Background

M-MIMO incorporates hundreds of antennas in telecommunication base stations to enhance the quality of service for several 5G applications, from video streaming and gaming to self-driving cars and smart cities. The more antennas we integrate, the more data we can send (resp. receive) simultaneously. i.e., One on each antenna.

2.1 Modulation

Modulation is the act of changing a signal to transmit data. It represents a collection of symbols that can be sent directly on one antenna in one transaction. A symbol is represented by a complex number, i.e., real and imaginary parts. The number of symbols in a given modulation is defined as 2^b, where b is the number of bits encapsulated in a symbol. For instance, in Binary Phase-shift keying (BPSK) modulation, one bit is sent per symbol ($b = 1$). Therefore, this modulation includes two symbols (1,0) and (−1,0). In 64 Quadrature Amplitude Modulation (64-QAM), we can send six bits per symbol ($b = 6$). This represents 64 symbols in total. The higher the modulation, the better the data rate. However, it also increases the communication system's error rate and complexity.

2.2 Signal to Noise Ratio (SNR)

The SNR measures the relevant signal strength in decibels (dB) compared to the noise signal that can get in the way. Therefore, the higher the SNR, the better the communication system. A high-SNR value indicates that the user is close to the transmit antenna. A user can be assigned a specific modulation based on the SNR value. For instance, a BPSK modulation in the low-SNR regime versus a 64-QAM modulation if the user has a high SNR.

2.3 Error Rate and Time Complexity

The error rate performance is a ratio between transmitted data and the one recovered correctly at the receiver side. The error rate varies according to the detection algorithm used. The lower the error rate, the better the communication system is. In general, 10^{-2} uncoded symbol error rate (SER) is considered an

acceptable error performance for many applications. For systems with powerful error correction codes, the previous SER readily translates into $10^{-5} - 10^{-6}$ SER error performance. The detection latency also depends on the complexity of the detection algorithm and the application area. In general, 10 ms is considered an acceptable latency for mobile communications. A good detection algorithm achieves a good trade-off between complexity and error rate performance.

For more information about communication science and engineering, please refer to e.g., [13].

3 Related Work

Many researchers have exploited multi-core CPUs and GPUs to accelerate non-linear detection algorithms.

Chen and Leib [4] propose a GPU-based Fixed Complexity Sphere Decoder. The authors reported a relative speedup of 7× for large MIMO systems. However, the time of the approach is an order of magnitude higher compared to 10 ms requirements.

Arfaoui *et al.* [3] propose a GPU-based SD algorithm in which a Breadth-First Search (BFS) exploration strategy is used to increase the GPU resource occupancy. However, BFS increases the complexity, especially in low SNR region. The authors reported excellent error rate performance. However, the proposed approach has a scalability limitation, especially for large constellations due to the exponential complexity of the SD.

Husmann *et al.* [9] propose a flexible parallel decoder for MIMO systems using GPU and field-programmable gate array (FPGA) architectures. Their algorithm contains two phases. A first pre-processing phase chooses parts of the SD search tree to explore, and a second phase maps each of the chosen parts of the SD tree to a single processing element (GPU or FPGA). The results are presented for a maximum of a 12×12 MIMO system using 64-QAM modulation.

Nikitopoulos *et al.* [10] propose the design and implementation of a parallel multi-search SD approach using multicore CPU and Very-Large-Scale Integration (VLSI) architectures. After the pre-processing phase, in which they obtain a processing order of the tree branches, the authors split the search tree into several sub-trees. Each sub-tree is then mapped on a processing element and explored using a depth-first strategy. However, the authors do not consider the load-balancing problem, which may arise in modulations with dense constellations. The authors approximate results for a maximum of 16×16 MIMO system using 64-QAM modulation.

Dabah *et al.* [2,5] propose a parallel multi-agent approximate approach that runs simultaneously a single agent with a SD instance while the remaining agents execute concurrent k-best algorithm to accelerate SD search tree.

Despite the decent error rate performance, the above-proposed methods still suffer from scalability limitations. For example, the largest MIMO configuration reported in the above works is for 32 antennas under 10 ms requirements, which is far from massive MIMO potential. In fact, all works mentioned above are based

on the SD algorithm, which has an exponential complexity $(2^b)^M$, with M the number of antennas and b the number of bits per symbol. Our GPU multi-level algorithm has a linear complexity $M(2^b)^L$, where L is the number of combined levels $L \in \{1, .., 4\}$. As a result, we report a good error rate performance for up to 100×100 antennas while maintaining an excellent error rate under real-time requirements.

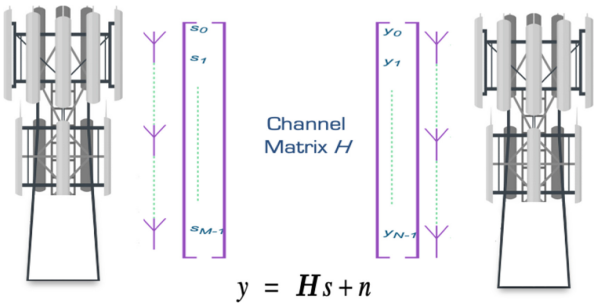

Fig. 1. Example of a MIMO system where the vector $\boldsymbol{s}$ is transmitted by M transmitter antennas via a channel matrix $\boldsymbol{H}$. The received vector $\boldsymbol{y}$ is a collection of N receiver antennas' observations.

4 System Model

In this paper, we consider a MIMO system consisting of M transmit antennas and N receive antennas, as depicted in Fig. 1. The transmitter sends M data streams simultaneously to a receiver using multiple antennas via a flat-fading channel. i.e., We consider a small-scale fading channel which is a standard model in the literature [3]. The following equation describes the base-band MIMO model: $y = \boldsymbol{H}s + n$, where the vector $\boldsymbol{y} = [y_0, ..., y_{N-1}]^T$ represents the received signal. $\boldsymbol{H}$ is an $N \times M$ channel matrix, where each element h_{ij} is a complex Gaussian random variable, with mean 0 and variance 1, that models the fading gain between the j-th transmitter and i-th receiver. The vector $\boldsymbol{s} = [s_0, ..., s_{M-1}]$ represents the transmitted vector, where $\boldsymbol{s_i}$ belongs to a finite alphabet set denoted by Ω. The input s is subject to an average power constraint ρ, i.e., $E[\|s\|^2] \leq \rho$. Finally, $\boldsymbol{n} = [n_0, ..., n_{N-1}]^T$ represents the additive white Gaussian noise with zero mean and covariance $\boldsymbol{I}_N$, where $\boldsymbol{I}_N$ designates the identity matrix of size N. With regard to the noise and channel normalization, the average power ρ also designates the average SNR per receive antenna. For convenience, let us consider $\mathcal{S}$ as the set of all possible combinations of the transmitted vector $\boldsymbol{s}$. The possible number of combinations corresponds to the complexity of the MIMO system and it is calculated as follows: $|\mathcal{S}|=|\Omega|^M$.

The two main options for decoding the received signal are linear decoders characterized by low complexity and poor error rate performance and nonlinear

(optimal or near-optimal) decoders characterized by good error rate quality but relatively high complexity.

Linear decoders multiply and map the received signal using a matrix denoted by $\boldsymbol{H_{inv}}$ $(M \times N)$, obtained from the channel matrix $\boldsymbol{H}$. The most common linear decoders are Zero Forcing (ZF) and Minimum Mean Square Error (MMSE). As for nonlinear decoders, the Maximum Likelihood [12] is the optimal decoder, exhibiting prohibitive complexity. It calculates *a posteriori* probability for each possible transmitted vector $\boldsymbol{s} \in \boldsymbol{S}$. In other words, the algorithm performs a brute-force exploration of the entire search space, as shown in Eq. 1:

$$\hat{\boldsymbol{s}} = \boldsymbol{arg} \min_{\boldsymbol{s} \in \boldsymbol{S}} ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}||^{\boldsymbol{2}}. \tag{1}$$

The SD algorithm [1,14] mimics the ML decoder, but limits the search for the optimal solution inside a sphere of radius r set initially by the user, as shown in the Eq. 2:

$$||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}||^{\boldsymbol{2}} < r, \quad \text{where} \quad \boldsymbol{s} \in \boldsymbol{S}. \tag{2}$$

The radius may then be updated subsequently at runtime to further prune the search space and reduce the complexity.

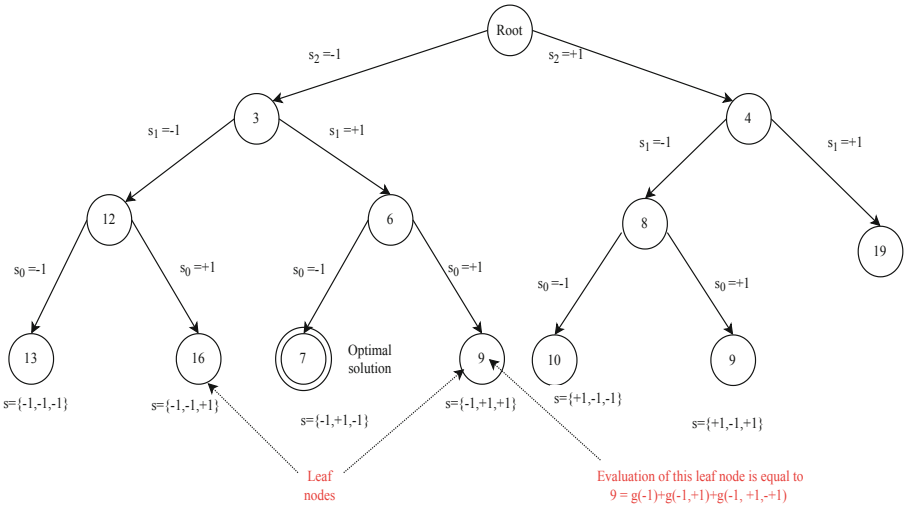

Fig. 2. Detection search tree for a MIMO system with three transmit antennas. One symbol is fixed at each level.

4.1 Tree-Based Representation

The problem in Eq. 2 can be transformed into another equivalent problem by performing the $\boldsymbol{QR}$ decomposition of the channel matrix $\boldsymbol{H}$ as follows:

$$\begin{aligned} ||\boldsymbol{y} - \boldsymbol{Hs}||^2 &= ||\boldsymbol{y} - \boldsymbol{QRs}||^2 \\ &= ||\bar{\boldsymbol{y}} - \boldsymbol{Rs}||^2, \text{ where } \bar{\boldsymbol{y}} = \boldsymbol{Q}^H \boldsymbol{y} \end{aligned}$$

where $\boldsymbol{Q} \in \boldsymbol{C}^{N \times N}$ is an orthogonal matrix and $\boldsymbol{R} \in \boldsymbol{C}^{N \times M}$ is an upper triangular matrix.

Thus, finding the supposed transmitted vector ($\hat{s}$) in Eq. (1) is equivalent to solving the following minimization problem:

$$\min \sum_{k=1}^{M} g_k(\boldsymbol{s_{M-1}, ..., s_{M-k}}), \quad where \tag{3}$$

$$g_k(s_{M-1}, ..., s_{M-k}) = ||\bar{y}_{M-k} - \sum_{i=M-k}^{M-1} (r_{(M-k),i} \boldsymbol{s_i})||^2 \tag{4}$$

where (4) represents the partial distance (PD) of a search tree node (path). Indeed, this latter formulation of the problem allows to model all possible combinations of the transmitted vector as a search tree with M layers. To find the path with the minimum distance from the received signal, the SD performs a tree exploration to retrieve the best path.

Algorithm 1: Multi-Level Detection Pseudo-code.

Require: $\bar{y}$, R
$P = \{\}$ *solution vector* L *number of aggregated levels* M *number of antennas*
1: **while** $|P|! = M$ **do**
2: Generate partial paths P_i $i \in \{1, \Omega^L\}$
3: Calculate PD_i $i \in \{1, \Omega^L\}$
4: Locate P_m such that $PD_m = \min\{PD_i,\ i \in \{1, \Omega^L\}\}$
5: $P = P \cup P_m$
6: **end while**
7: **return** P

5 Multi-level Approach

This section describes a multi-level approach that relies on two factors to keep real-time requirements and a good error rate. The first factor is algorithmic based on our multi-level technique. The second factor is efficiently exploiting the computing power of GPU resources and its large number of processing elements.

As depicted in Algorithm 1, our approach operates on a search tree with M levels (number of transmit antennas) and constructs only one solution named P (complete path). Usually, one symbol is detected at each level, starting from symbol S_{m-1} at level 1 to finally reach symbol S_0 at level M. Our idea is to combine the detection of multiple and successive symbols simultaneously. Despite the increase in the number of successors from $|\Omega|$ to $|\Omega|^L$, combining the detection of L symbols increases the accuracy in terms of error rate performance and reduces the number of iterations of our multi-level approach from M to M/L. Starting from a partial path P (initially empty), our approach creates $|\Omega|^L$ partial paths (P_i / i=1,.., $|\Omega|^L$) that extend P with all possible combinations of L symbols. After that, we calculate the partial distance (PD_i) for each partial path P_i using Eq. (5) Next, we replace P with the best partial path P_i in terms of partial distance (minimum PD_i). We repeat this process until reaching the last level of the tree, where we return P as an approximate solution to the MIMO detection problem.

Increasing the constellation size to 64-QAM (transmitting six bits per symbol) increases the error probability to fall into neighboring symbols instead of the transmitted one due to the noise. Our approach overcomes this issue by using coefficients of the next lower levels to confirm which of these symbols is the right one.

6 GPU-Based Multi-level Approaches

Increasing the aggregated levels increases the accuracy. However, it also increases the complexity. To keep practical time complexity and good error rate even for large constellations, we exploit low precision tensor core capacity in recent GPU hardware. All parts of our Multi-level approach are implemented and executed on GPU to avoid all data-transfer over the slow PCIe bus.

6.1 GPU Multi-level

We formulate our multi-level algorithm as a linear algebra operation that computes the PD (evaluation) of all partial paths simultaneously and then chooses the best one for the next iteration. Indeed, our algorithm is implemented to avoid: (1) thread divergence, especially in generating the partial paths; (2) increasing the compute portion of the algorithm by reformulating this process as matrix algebra operation $\boldsymbol{A} * \boldsymbol{B} + \alpha \boldsymbol{C}$; and finally (3) relying on a reduction process to find the best candidate for following iterations. More detail on efficiently exploiting GPU resources in general, and half-precision in particular, is given in what follows.

Complex to Real Transformation. Wireless communication data are modeled as complex numbers. In order to exploit low-precision arithmetic, we must perform a transformation from complex to real because there is no GPU support for low-precision computation for complex numbers. There are two ways

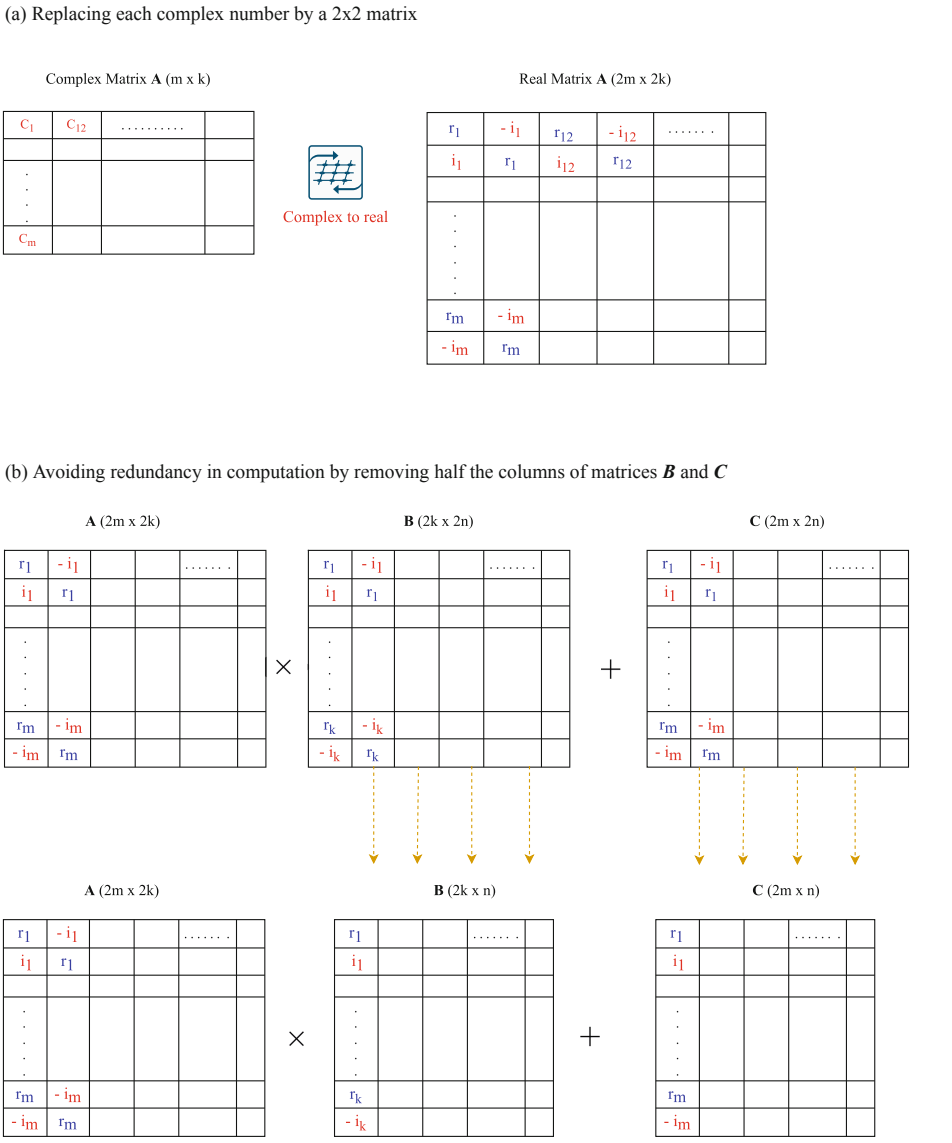

Fig. 3. Complex to real transformation.

to do the transformation. In the first way, we split a complex matrix into two matrices, one matrix representing the real part and the other one representing the imaginary part. This option creates an overhead of managing two matrices instead of one, thus inducing an overhead in computation and memory access. The other interesting option (preferred) is to replace each complex number with a 2×2 matrix. This option is depicted in Fig. 3 (a). Therefore, Matrix $\boldsymbol{A}$ with

m rows and k columns will be transformed to a matrix with $2*m$ rows and $2*k$ columns. We do the same for matrices $\boldsymbol{B}$ and $\boldsymbol{C}$.

Matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ can have millions of columns, inducing a huge number of floating-point operations (FLOPS) and memory access. We can notice that matrix $\boldsymbol{C}$ (multiplication result) has duplicated information (r_1,i_1), and ($-i_1$,r_1) (See Fig. 3 (b)). Here, we exploit this redundancy to cut down by half the number of flops and memory accesses in the multiplication. In this way, we reduce the size of matrix $\boldsymbol{B}$ from $2k$ * $2n$ to $2k$ * n. Similarly, for matrix $\boldsymbol{C}$. This is important since the number of columns of matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ can reach several million.

Avoiding Thread Divergence. Thread divergence appears when threads within the same warp don't follow the same instruction path (if-else), resulting in negative performance consequences. The thread divergence situation is known when exploring trees on GPU since the branching process has many if-else instructions.

Exploring a search tree and generating partial paths (successor nodes) at each iteration represents a bottleneck on GPU since it involves many if-else conditions. To answer this issue when generating partial paths (all possible combinations of L symbols), we divide a partial path into two parts. A part common with all partial paths (from root to node x) and a distinct part that is unique for each partial path. For instance, the partial paths in Fig. 4 second iteration have two parts: a common part (marked in red) from root to node x, followed by the unique part for each partial path. The distinct part for all partial paths is represented by a matrix $\boldsymbol{B}$. This latter contains all possible combinations of L symbols such that each column represents a partial path. This matrix is generated once and does not change from one iteration to another. The only thing that changes from one iteration to another is the common part modeled as a vector Vc.

On the one hand, this decomposition allows to avoid thread divergence situations. On the other hand, it also allows reducing the size (memory and flops) of matrix $\boldsymbol{B}$ (resp. $\boldsymbol{C}$). Without the aforementioned decomposition, the common part will be duplicated for all partial paths $|\Omega|^L$, which can reach millions.

New Incremental Evaluation: The evaluation for each partial path is calculated using Eq. 4. To increase the arithmetic intensity of our algorithm, we grouped the evaluation for all partial paths as a matrix multiplication as follows: $\boldsymbol{A}*\boldsymbol{B}+\alpha\boldsymbol{C}$.

$$E_{Pi} = E_p + \sum_{k=0}^{L-1} \|C_{ki} + V_k\| \tag{5}$$

The evaluation of a partial path P_i is the evaluation of the constructed path P (calculated in the previous iteration) plus the square norm over column $C_i + V$. Following the decomposition we did earlier, the evaluation is divided into two parts, i.e., matrix-matrix multiplication and matrix-vector multiplication. The square matrix $\boldsymbol{A}$ is obtained from matrix $\boldsymbol{R}$ ($\boldsymbol{QR}$ decomposition page 6.), such that it contains the rows of L fixed symbols in the current iteration. The matrix $\boldsymbol{B}$ is defined in the earlier section as all possible combinations of the L symbols, which can reach millions of columns. Finally, the matrix $\boldsymbol{C}$ represents the

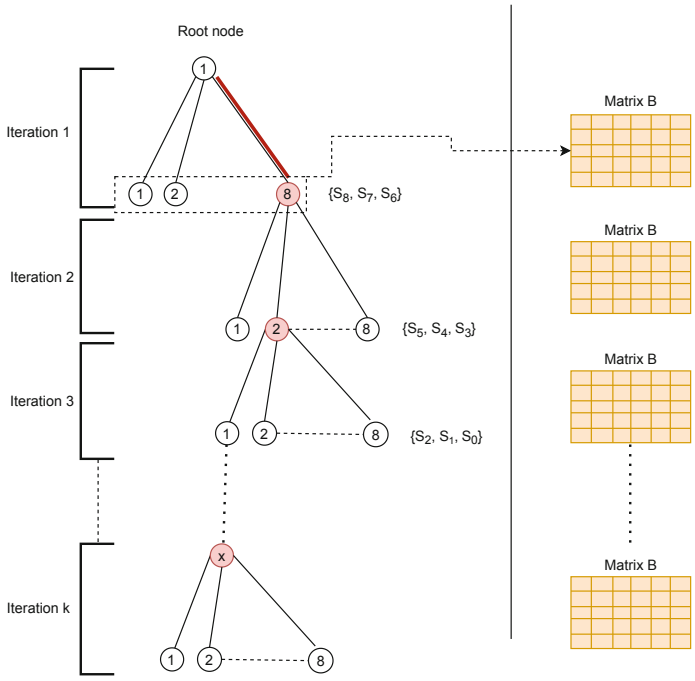

Fig. 4. Branching scheme of the multi-level approach.

elements of $\bar{y}$ corresponding to the L symbols in the current iteration duplicated $|\Omega|^L$ times.

The other part of the evaluation is a matrix-vector multiplication that multiplies Matrix $\boldsymbol{A'}$ obtained from matrix $\boldsymbol{R}$ with the common vector Vc defined earlier. *Sorting (Reduction):* After the evaluation phase, our algorithm chooses the best partial path in terms of evaluation, i.e., distance from the received signal.

6.2 Multi-GPU Version

As earlier stated, the multiplication used to compute the evaluation for each partial path ($\boldsymbol{A} * \boldsymbol{B} + \alpha \boldsymbol{C}$) requires nearly 80% of execution time (see Fig. 6). In addition to using low-precision mode for computing the above multiplication, we aim to accelerate this phase further using multiple GPUs. Thanks to our path decomposition to avoid thread divergence, matrix $\boldsymbol{B}$ remains the same from one iteration to another. In addition, matrix $\boldsymbol{A}$ for each iteration is known in advance. The idea behind this multi-GPU version is to overlap all the multiplication used during the detection process using multiple GPUs. As depicted in Fig. 5, all multiplications from different iterations are performed on multiple GPUs at the same time. This reduces all the matrix multiplication operations to the complexity of one multiplication. The only phases that need to be done sequentially are the norm calculation and min.

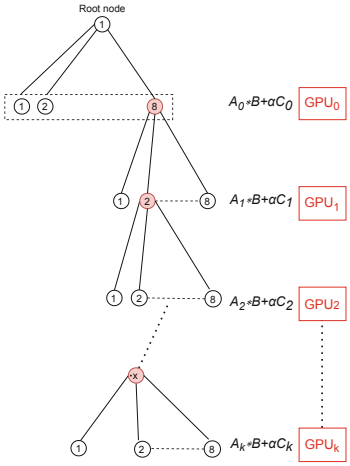

Fig. 5. Multi-GPU version where the matrix multiplication operations during the whole detection process are performed simultaneously on several GPUs.

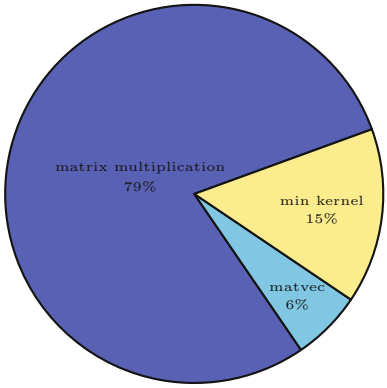

Fig. 6. Time partition of different kernels (single precision) of our approach for a 100×100 MIMO system with 64-QAM modulation and four levels.

7 Results and Discussions

In the following, we conduct experiments to assess our GPU-based approach's accuracy (error rate) and complexity. For that, we use MMSE linear detection and the optimal GPU-SD in [3]. The exponential complexity of the SD prevents it from dealing with large MIMO systems. For this reason, we include its performance for a small MIMO system. We perform our experiments using a server with four NVIDIA A100 GPUs with 40GB and a two-socket Intel IceLake CPU 2 GHz with 28 CPU-core and 1024 GB of main memory. For all the experiments, we consider the case of perfect channel state information. This means that the channel matrix is known only at the receiver. Each experiment uses randomly generated symbols (data set). As a result, the data sets are different from one execution to another which is close to real wireless data. All level three BLAS operations are performed using the vendor-optimized cuBLAS library.

Figure 7 illustrates the impact of increasing the number of combined levels on the error rate and complexity of our ML approach. We compare our results with the accuracy of the optimal SD algorithm to show how far we are from optimal results. Despite the attractive latency of the MMSE algorithm, this latter has poor error rate performance, which makes it not suitable for M-MIMO. The first observation from sub-figure (a) is the good impact of increasing the number of levels on the error rate performance. Indeed, the accuracy of our multi-level technique is quite close to the performance of the optimal GPU-SD [3] when using four and five levels. However, if we look at the complexity (sub-figure (b)),

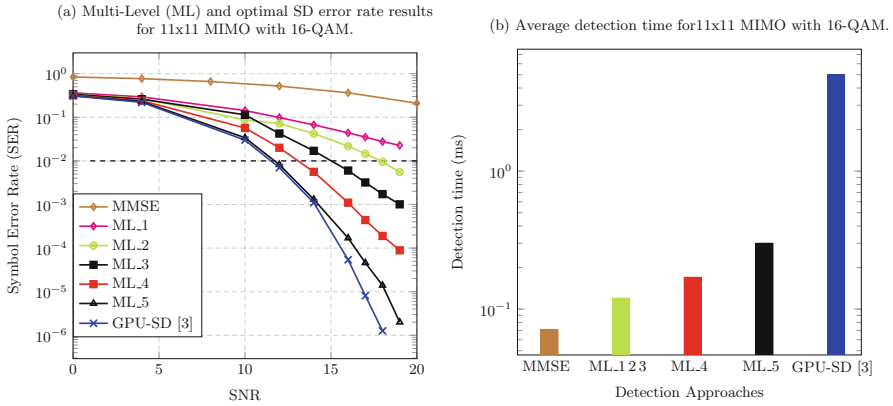

Fig. 7. Accuracy and latency results of our GPU ML approach compared to the linear MMSE and the optimal SD results.

we can see a significant gap in complexity between the two approaches. Indeed, The GPU-SD [3] has high latency since it enumerates all possible combinations of the transmitted signal inside a given radius, which results in a massive number of explored paths. This is not the case with our approach, which combines multiple levels to target the best path in the search tree. This results in a limited number of explored nodes (low latency) while achieving high accuracy. On average, our approach (ML_5) is 40x faster than GPU-SD [3] for this small configuration. By increasing the number of levels of our approach from one to four, we reach the acceptable accuracy (10^{-2}) at 13 dB instead of 22 dB, thus saving 9 dB in power consumption with a slight increase in complexity. This represents a good accuracy/complexity balance for communication users. Thus, increasing the number of levels in our approach is crucial for achieving better accuracy. However, the complexity increases accordingly. To scale the number of antennas while keeping reasonable complexity, we exploit tensor-core capability in recent GPUs.

Figure 6 shows the time partition of our GPU kernels for a 100×100 MIMO system with 64 QAM modulation and four levels. The matrix-matrix multiplication required to evaluate partial paths represents 76% of the total execution time. In this configuration, we have 16777216 partial paths that need to be evaluated as matrix-matrix multiplication with m, k, and n equal to 8, 8, and 16777216, resp. As a result, lowering the time complexity of our approach requires reducing the complexity of the matrix multiplication operation. To achieve this goal and study the impact of low-precision data structure on the wireless communication field in general and MIMO detection in particular, we exploit FP16 and INT8 as follows.

Figure 8 shows the error rate performance of our approach using different arithmetic precisions (FP32, FP16, and INT8) using respectively three and four combined levels (ML_3, ML_4) for a 100×100 MIMO with 64-QAM modulation.

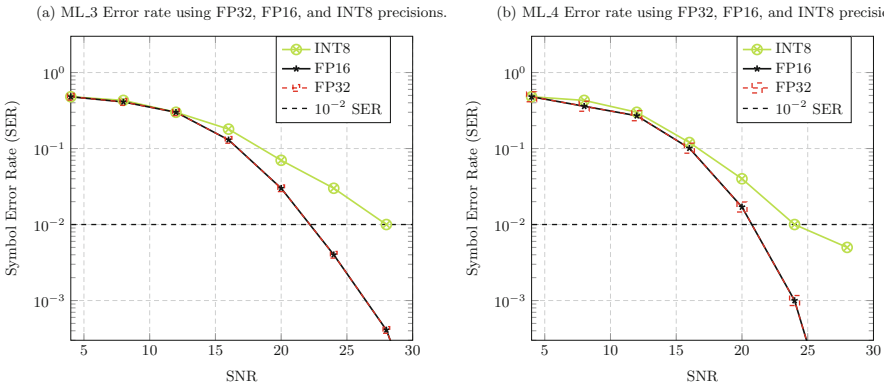

Fig. 8. Error rate performance using different arithmetic precision from float 32 bits precision to the smallest integer 8 bits for a 100×100 MIMO with 64-QAM modulation. Sub-figures (a) and (b) give the results of our algorithm with three levels (ML_3) and four levels (ML_4), respectively.

The interesting observation from both sub-figures (a) and (b) is that our approach performs well and can support precision loss even when using the smallest representation of 8 bits (size of a register). Indeed, we can see from the two sub-figures that passing from FP32 to FP16 representation of matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ has no effect on the accuracy for all SNR regions. This means that the conversion, multiplication, and accumulation in FP16 does not lead to accuracy loss. In turn, this means that the multiplication and accumulation operations performed during the GEMM are all within the range of FP16. Furthermore, when moving to the smallest representation that can fit into a register(INT8), we see a moderate variation in error rate performance. This means we are losing some useful information. Since the accumulation for INT8 is done in integer 32 bits, the precision loss occurs when converting matrix $\boldsymbol{A}$ from Float 32 bits to Integer 8 bits. Indeed, when scaling up matrix $\boldsymbol{A}$, we may be out of the INT8 range (−127,127), which affects the accuracy in error rate, especially in the high SNR region. Figure 9 shows the effect of increasing the scaling number in the float_to_INT8 conversion of matrix $\boldsymbol{A}$ on the accuracy. We can identify two phases, a first one where increasing the scaling number improves the accuracy, and a second phase where increasing the scaling number negatively affects the accuracy. Indeed, a large scaling number leads to integer values out of the INT8 range (−127, 127). Thus, all values above (resp. under) 127 (resp. −127) are represented by 127 (resp. −127). Therefore, we lose useful information, which explains the decrease in accuracy.

It seems that increasing the number of levels positively impacts the accuracy of the INT8 version. Figure 10 investigates this behavior for 100×100 MIMO with 16-QAM modulation. It shows the impact of increasing the number of levels on the accuracy of the INT8 version in terms of error rate performance. We can see clearly the good impact of increasing the number of levels on the accuracy

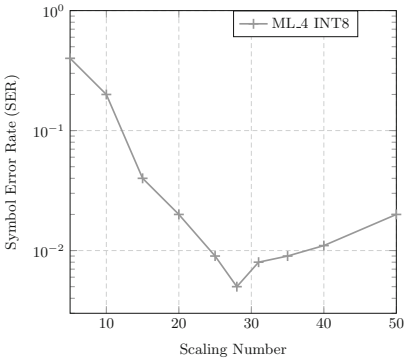

Fig. 9. Impact of scaling number in the float_to_INT8 conversion on the error rate performance for a 100×100 MIMO 64-QAM modulation using ML_4 and SNR = 28 dB.

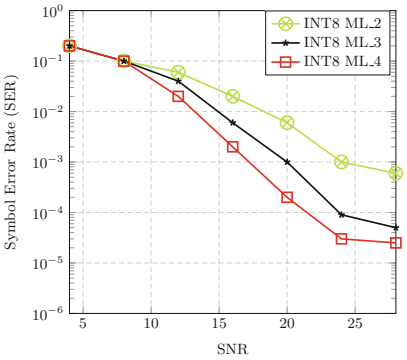

Fig. 10. Impact of multi-level technique in supporting integer 8-bits accuracy for 100×100 MIMO with 16-QAM modulation.

of the INT8 version by saving up to five dB in power consumption, which is an important aspect in the wireless communication domain.

From Fig. 8, we can identify two SNR regions. Low SNR region between 0 and 16 dB, where INT8 version has a similar error rate compared to FP16 and FP32 versions. After that, a second region begins where we can see the loss in error rate performance of the INT8 version. On the one hand, reducing the precision in the low SNR region affects the chosen path. However, this is not visible since the error rate is very high in this region, even when using the FP32 version. On the other hand, reducing the precision has a visible impact on the accuracy of MIMO detection. However, Fig. 10 shows that combining more levels (ideally four levels) reduces the impact of precision loss. Indeed, increasing the number of levels increases the difference in terms of evaluation between the optimal path and neighboring symbols. This compensates for the precision loss in this SNR region.

Figure 11 gives a general view of INT8 performance for different modulations from BPSK where we send only one bit per antenna, to 64-QAM where six bits are sent together per antenna. We can see from Fig. 11 the limited impact of precision loss on the accuracy of MIMO detection for all modulation and SNR regions. We can see that the more dense the constellation, the more impact of precision loss. Indeed, increasing the constellation size increases the error probability and increases the impact of precision loss since this lost information can influence the chosen path.

Figure 12 shows the impact of using different arithmetic precision on the time complexity of our approach using four levels. We can see that 1.7× improvement in complexity going from FP32 (32 bits) to FP16 (16 bits) without any impact on the accuracy, as we saw earlier. We can also see that INT8 precision does not

significantly impact time complexity due to the limited support in CUDA driver 11.6. Indeed, tensor cores are currently not activated for the non-transpose cases when launching this specific CUDA INT8 GEMM kernel. However, even when using half-precision, which has mature support on the GPU hardware, we are not close to the theoretical 18× speedup. Our hypothesis is that the shape of the matrix for our approach deeply affects the performance gain using tensor cores. Figure 13 investigates this and shows the performance gain using driver matrix multiplication with two kinds of matrices. The first is the short and wide matrix shape from our MIMO multi-level detection, i.e., $A(8 \times 8)$, $B(8 \times 16M)$. The second kind is a square matrix $A(4k \times 4k)$ and $B(4k \times 4k)$.

Figure 13 confirms our suggestion that the shape of the matrices significantly impacts the improvement factor using tensor cores. Indeed, with a square shape of $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, we are getting close to the theoretical peak performance using both FP32 and FP16, with an improvement factor around 15×. On the other hand, the low performance achieved by the wide and short matrices (MIMO shape) is explained by two reasons. The main reason is that this latter shape of matrices engenders a memory-bound regime of execution with an Arithmetic Intensity (AI) in flops per byte of only 4 compared to an AI of 682 square shape matrices. Such matrix shape does not engender enough data reuse for such an operation to be in the compute-bound regime of execution, as usually noticed for traditional square matrix-matrix multiplication. The same conclusion is also valid for INT8 precision.

In addition to using low-precision, we exploit multiple GPUs to overlap the matrix multiplication performed during the detection process. Figure 14 shows

FP32, FP16

Modulation / SNR	0	4	8	12	16	20	24	28	32
64-QAM	4.8E01	4.8E01	4.3E01	2.7E01	1.0E01	1.0E02	1.0E03	5.0E05	4.0E06
16-QAM	3.0E01	2.1E01	7.0E02	1.9E02	1.6E03	2.0E04	3.0E05	3.0E05	3.0E05
4-QAM	2.4E02	3.0E03	2.0E04	2.0E05	1E08	1E08	1E08	1E08	1E08
BPSK	1E08	1E08	1E08	1E08	1E08	1E08	1E08	1E08	1E08

INT8 with FP32 accumulation

Modulation / SNR	0	4	8	12	16	20	24	28	32
64-QAM	4.8E01	4.8E01	4.3E01	2.7E01	1.0E01	4.0E02	1.0E02	5.0E03	4.0E03
16-QAM	3.0E01	2.0E01	1.0E01	2.0E02	2.0E03	2.0E04	3.0E05	3.0E05	3.0E05
4-QAM	2.4E02	4.0E03	2.0E04	2.0E05	1E08	1E08	1E08	1E08	1E08
BPSK	1E08	1E08	1E08	1E08	1E08	1E08	1E08	1E08	1E08

Fig. 11. ML_4 error rate heat-map using low-precision arithmetic for different modulations and SNR values for a 100 × 100 with 64-QAM modulation.

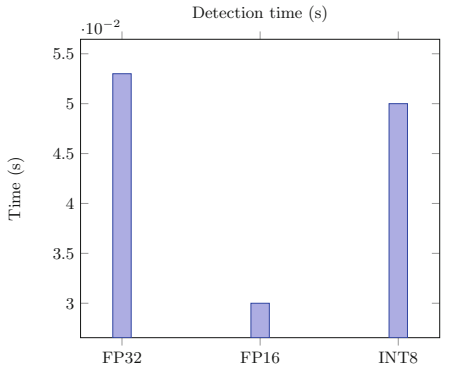

Fig. 12. Detection latency using ML_4 with different arithmetic precision for a 100 × 100 MIMO with 64-QAM modulation.

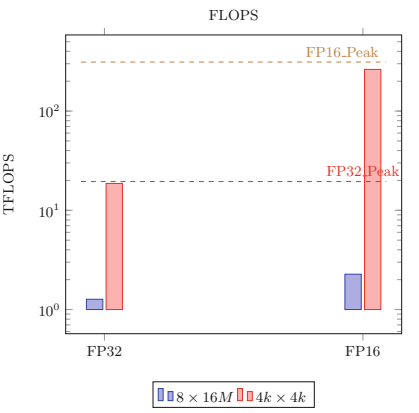

Fig. 13. FLOPs using ML_4 with different arithmetic precision.

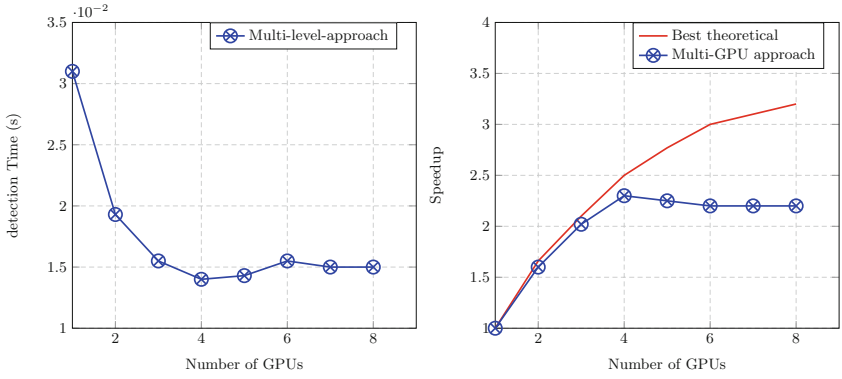

Fig. 14. complexity versus the number of GPUs for our multi-GPU approach for a 100 × 100 MIMO system with 64-QAM using four levels.

the impact of scaling the number of GPUs to further reduce the time-to-solution of the main kernel, i.e., the matrix-matrix multiplication. This latter represents more than 80% of the elapsed time for dense constellations. The idea is to execute matrix-matrix multiplications from subsequent iterations using multiple GPUs simultaneously. However, the remaining 20% of the code must be executed sequentially, which may impede strong scaling performance. Figure 14 shows the complexity (a) and speedup (b) of our multi-GPU approach for a 100 × 100 MIMO system with 64-QAM modulation and four levels. Sub-figure (b) shows the theoretical best speedup (red curve) and achieved speedup (blue curve) by our multi-GPU approach. We can notice two regimes: the first between one and four GPUs, where the complexity decreases, and the second between four and eight GPUs, where increasing GPUs has no effect on the complexity. The first

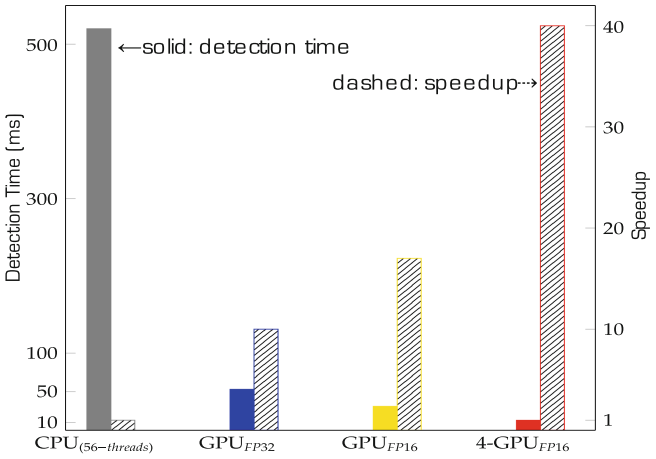

Fig. 15. Complexity and speedup of our proposed approaches for a 100×100 MIMO with 64-QAM modulation.

regime is characterized by a rapid decrease in complexity and a near-optimal speedup. This is due to a high compute-to-communication ratio. After that, increasing the number of GPUs increases the communication-to-computation ratio, which indicates that the increased communication neutralizes the benefit achieved by overlapping addition iterations. Despite the fact that matrix multiplication represents 80% of the execution time, we still need to perform the norm and min kernels. This results in synchronization and data transfer between GPUs. Indeed, the communication when using one to four GPUs is performed using the high-speed NVLink interconnect, whereas increasing the number of GPUs further leads the communication through the slow PCIe bus. This increases significantly the communication, which neutralizes the gain from overlapping more multiplication operations for this particular data set. Adding more levels will allow supporting more than four GPUs; however, the complexity will increase beyond the acceptable threshold for mobile communication.

Figure 15 shows the overall performance against a multi-core CPU implementation on IceLake architecture for a 100×100 MIMO with 64-QAM modulation (ML_4). For a fair comparison, the CPU implementation is also based on real matrix representation as explained in Fig. 3. The best performance for the parallel CPU version is reached around 30 threads and remains the same up to 56 threads. Solid fill indicates the time to solution, while dashed bars report the speedup achieved. Figure 15 shows that going from the multi-CPU version with 30 physical threads FP32 to one GPU A100 with FP32 single precision leads to $10\times$ improvement in complexity. Moreover, exploiting half-precision arithmetic (FP16) pushes the speedup to $17\times$. Furthermore, our multi-GPU version is $2.3\times$ faster than the single-GPU version with half-precision mode. In total, our multi-GPU version is $40\times$ faster than the parallel CPU implementation. As a result, we achieve a good complexity/accuracy trade-off.

Regarding power consumption, our approach requires an average of 78 W which is below the 90 W cap imposed by wireless vendors.

8 Conclusion and Perspectives

Recent GPUs have fast tensor-core operations that leverage low-precision arithmetic to achieve performance gain. This paper exploits this capability to overcome M-MIMO detection overhead for a large number of antennas. In this paper, we demonstrate the positive impact of low-precision arithmetic operations (32 bits, 16 bits, and 8 bits) on the complexity (1.7×) while maintaining a good accuracy performance of our multi-level detection algorithm. To further reduce the complexity while maintaining the same accuracy performance, we proposed a multi-GPU approach that overlaps the matrix-multiplication operations on subsequent iterations. This resulted in an additional 2.3× speedup. To summarize, we have improved the complexity by a factor of 4× compared to a single-precision single-GPU approach and 40× compared to the multi-core CPU implementation on a two-socket 28-core IceLake.

In future work, we will investigate the potential gain of a Field-Programmable Gate Array (FPGA) on both complexity and power consumption.

References

1. Agrell, E., Eriksson, T., Vardy, A., Zeger, K.: Closest point search in lattices. IEEE Trans. Inf. Theory **48**(8), 2201–2214 (2002)
2. Alouini, M.S., Keyes, D.E., Ltaief, H., Dabah, A., Rezki, Z.: Massive multiple-input multiple-output system and method (14 Dec 2021). US Patent 11,201,645
3. Arfaoui, M.A., Ltaief, H., Rezki, Z., Alouini, M.S., Keyes, D.: Efficient sphere detector algorithm for massive MIMO using GPU hardware accelerator. Procedia Comput. Sci. **80**, 2169–2180 (2016)
4. Chen, T., Leib, H.: GPU acceleration for fixed complexity sphere decoder in large MIMO uplink systems. In: IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE 2015), pp. 771–777. IEEE (2015)
5. Dabah, A., Ltaief, H., Rezki, Z., Arfaoui, M.A., Alouini, M.S., Keyes, D.: Performance/complexity trade-offs of the sphere decoder algorithm for massive MIMO systems. arXiv preprint arXiv:2002.09561 (2020). To be submitted
6. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comput. **44**(170), 463–471 (1985)
7. Foschini, G.J.: Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. Bell Labs Tech. J. **1**(2), 41–59 (1996)
8. Hassibi, B., Vikalo, H.: On the sphere-decoding algorithm I. expected complexity. IEEE Trans. Signal Process. **53**(8), 2806–2818 (2005)
9. Husmann, C., Georgis, G., Nikitopoulos, K., Jamieson, K.: FlexCore: massively parallel and flexible processing for large MIMO access points. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2017), pp. 197–211 (2017)

10. Nikitopoulos, K., Georgis, G., Jayawardena, C., Chatzipanagiotis, D., Tafazolli, R.: Massively parallel tree search for high-dimensional sphere decoders. IEEE Trans. Parallel Distrib. Syst. **30**(10), 2309–2325 (2018)
11. Paulraj, A.J., Kailath, T.: Increasing capacity in wireless broadcast systems using distributed transmission/directional reception (DTDR) (6 Sep 1994). US Patent 5,345,599
12. Simon, M.K., Alouini, M.S.: Digital Communication over Fading Channels (Wiley Series in Telecommunications and Signal Processing), 2nd edn. Wiley-IEEE Press, New York (2004)
13. Sklar, B., et al.: Digital Communications, vol. 2. Prentice Hall, Upper Saddle River (2001)
14. Viterbo, E., Boutros, J.: A universal lattice code decoder for fading channels. IEEE Trans. Inf. Theory **45**(5), 1639–1642 (1999)