A RECONFIGURABLE GPU IMPLEMENTATION FOR TOMLINSON-HARASHIMA PRECODING

Fernando Domene, Sandra Roger, Carla Ramiro, Gema Piñero, and Alberto Gonzalez

Institute of Telecommunications and Multimedia Applications, Universitat Politècnica de València
Camino de Vera s/n, 46022, Valencia, Spain, email: ferdool@iteam.upv.es

ABSTRACT

Fast parallel processing capability of general purpose Graphic Processing Units (GPU) can be exploited to accelerate the precoding calculation needed in spatially multiplexed wireless communication systems. In this paper, a GPU-based implementation of the well-known multiuser Tomlinson-Harashima precoding (THP) scheme combined with a lattice-reduction (LR) stage is presented. The proposed approach allows the LR stage to be switched off when user requirements are achieved by using only THP. Moreover, our GPU implementation provides scalability in the number of subcarriers per symbol, which is a key factor in LTE and 4G wireless standards. Simulation results show that the GPU-based THP implementation performs up to 7 times faster than its CPU-equivalent whereas the LR stage implementation only achieves a speedup of 3. Despite the fact that the LR cannot be as efficiently parallelized as the THP, a speedup of nearly 6 is achieved when both are combined.

Index Terms— Multiuser precoding, Tomlinson-Harashima Precoding, GPU.

1. INTRODUCTION

Multiple-input multiple-output (MIMO) wireless communications have been widely studied during the last decade [1]. MIMO systems provide high spectral efficiency by means of spatially multiplexing as many data streams as transmitting antennas are used in the Base Station (BS). Furthermore, MIMO techniques can be used to enhance the performance of Orthogonal Frequency Division Multiplexing (OFDM) systems by exploiting the spatial domain. MIMO-OFDM allows to transmit different streams over the different subcarriers and, through MIMO precoding, different spatial beams in each one of the subcarriers.

Graphic Processing Units (GPU) have recently become attractive for the efficient implementation of signal processing algorithms for communication systems, such as the soft MIMO detector in [2] or the decoding of LDPC codes [3]. In this paper, we focus on the implementation of the Tomlinson-Harashima precoding (THP) method [4] and its combination with a LLL [5] lattice-reduction stage, called LR-THP. These non-linear techniques achieve a better performance than linear techniques, such as Zero Forcing, at the cost of a higher computational cost. In [6], LR-THP was shown to improve the performance of conventional THP at the expense of an increased computational cost, mainly due to the LLL lattice-reduction stage [5, 7]. To address the above issue, the proposed implementation has been developed using a GPU, since they provide a huge capability of parallel processing and rapid prototyping.

A comparison among the proposed implementations of the algorithms under study is carried out with the conventional execution on a high performance CPU, showing that the GPU highly speeds up the execution of the methods. GPUs also allow for reconfigurability and this advantage has been exploited to propose a reconfigurable THP scheme combining the use of LR. Moreover, since the GPU is more rarely used than the CPU in conventional applications, its use as a co-processor in signal processing systems is very promising.

2. SYSTEM MODEL

Consider a multiuser MIMO-OFDM wireless downlink from a base station (BS) with N antennas to $K \leq N$ single-antenna users. According to technical specifications in LTE Rel. 10 [8], only N_c out of M_{IFFT} subcarriers are used for information purposes, whereas subcarriers placed at both edges of the total bandwidth are void in order to reduce the requirements of analog filters. Under spatial multiplexing precoding, all the users receive their information through the same set of N_c subcarriers. Received symbols by the K users of the system at the mth subcarrier can be grouped in vector $\mathbf{y}[m] \in \mathbb{C}^{K\times 1}$:

$$\mathbf{y}[m] = \mathbf{H}[m]\mathbf{x}[m] + \mathbf{n}[m], \tag{1}$$

where vector $\mathbf{x}[m] \in \mathbb{C}^{N\times 1}$ includes the precoded information symbols and $\mathbf{n}[m] \in \mathbb{C}^{K\times 1}$ is the received noise for the mth subcarrier. Matrix $\mathbf{H}[m] \in \mathbb{C}^{K\times N}$ is the channel matrix and its elements describe the signal fading from each transmitter antenna to each user for the mth subcarrier. We assume block-fading channels, constant on blocks of duration L_{ch} symbols, and changing according to some ergodic statistics from block to block. For practical reasons, we are using

Thanks to the TEC2009-13741 project and to the PROMETEO/2009/013 project for funding.

an equivalent $(2K \times 2N)$-dimensional real-valued representation of equation (1) as described in [7].

3. RECONFIGURABLE TOMLINSON-HARASHIMA PRECODING

In this work we focus on the THP [4] scheme, which uses a simple modulo operator to reduce the transmitted signal power from what it would be if a linear precoding was performed. The THP scheme can be divided into two stages: the preprocessing stage (PPS) and the per-symbol-vector stage (PSVS) [7]. The PPS calculates some matrices independent of the symbol vector which are posteriorly used by the PSVS stage. Thus, the PPS can be done off-line and its results be reused whereas channel does not change. On the other hand, the PSVS collects the calculations performed every time that a new symbol vector is transmitted, which happens L_{ch} times for a given realization of a block-fading channel.

It was shown in [6] that the performance of the THP algorithm can be substantially improved by previously reducing the channel matrix with the LLL algorithm. The performance comparison among both algorithms is shown in Fig.1. Since, depending on the user requirements and channel condition, the performance of the THP can be sufficient with much lower cost than the LR-THP, we propose a reconfigurable scheme that can switch between the conventional THP algorithm and the LR-THP, as depicted in Fig. 2. As reported in [9], reconfigurable approaches are meaningful since they allow for a flexible tradeoff between the energy efficiency and quality of service. Note that, through a switch, the original data $\mathbf{s}$ is replaced by the modified data $\mathbf{R}^{-1}\mathbf{s}$, allowing the use of reduced channel matrix $\mathbf{A}$ in the THP block.

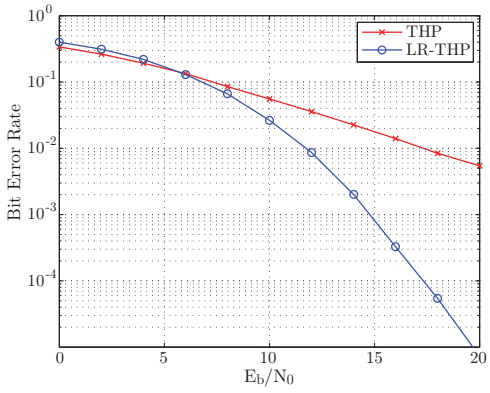

Fig. 1. Bit error rate of the Reconfigurable Tomlinson-Harashima Precoding scheme with and without LR stage.

A detailed description of the THP precoding techniques considered in this work can be found in several works as [7], thus, we will not include such information in this paper. However, we found useful to address the mathematical operations that each technique carries out in the practical reconfigurable setup (see Table 1).

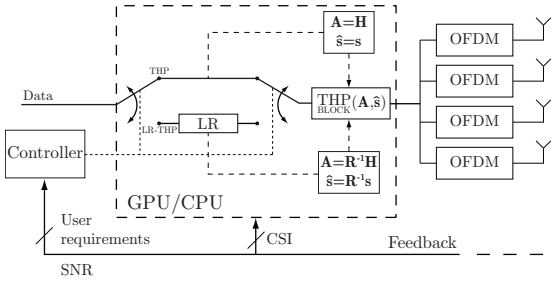

Fig. 2. Reconfigurable Tomlinson-Harashima Precoding scheme.

Table 1. Mathematical operations in the proposed scheme

A.- Preprocessing stage:

1.- switch(algorithm)
  case(THP) : $\mathbf{A} = \mathbf{H}$
  case(LRTHP) : $\mathbf{A} = \mathbf{R}^{-1}\mathbf{H}$

2.- $\mathbf{A} = \mathbf{L}_0\mathbf{Q}_0 = (\mathbf{L}_0\mathbf{G}^{-1})(\mathbf{G}\mathbf{Q}_0) = \mathbf{L}\mathbf{Q}$

3.- $\mathbf{Q}^{+} = (\mathbf{G}\mathbf{Q}_0)^{+} = \mathbf{Q}_0^{\mathrm{T}}\mathbf{G}^{-1}$

B.- Per-symbol-vector processing stage:

1.- switch(algorithm)
  case(THP) : $\hat{\mathbf{s}} = \mathbf{s}$
  case(LRTHP) : $\hat{\mathbf{s}} = \mathbf{R}^{-1}\mathbf{s}$

2.- for $k = 1, \ldots, 2K$
  $\hat{x}_k = \hat{s}_k - \sum_{l=1}^{k-1} \mathrm{L}_{k,l}\tilde{x}_l$
  $\tilde{x}_k = \hat{x}_k \bmod A = \hat{x}_k - A\left\lfloor \frac{\hat{x}_k + A/2}{A} \right\rfloor$
  end

3.- $\mathbf{x} = \mathbf{Q}^{+}\tilde{\mathbf{x}}$

4. IMPLEMENTATION OF THE PRECODERS ON CUDA

4.1. GPU and CUDA

GPUs are becoming a meaningful choice to implement signal processing algorithms due to their massive and intrinsically parallel computational capability. The use of this hardware can achieve high computation throughput by employing many cores to execute the algorithms in parallel, as many works on this topic show [2, 3]. On the other hand, Compute Unified Device Architecture (CUDA) [10] is a software programming model that exploits the massive computation potential offered by GPUs. A GPU can have multiple stream multiprocessors (SM), where each stream multiprocessor consists of 8 pipelined cores if CUDA capability is 1.2 or 1.3, 32 pipelined cores if it is 2.0 [10] or even 48 pipelined cores if it is 2.1. A CUDA device has a large amount of off-chip device memory (global memory) and a fast on-chip memory called (shared memory). In this model, the programmer defines the kernel function which 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, usu-

ally up to 512, and all the blocks within a grid must have the same size. Each thread can select a set of data using its own unique ID and execute the kernel function on the selected set of data.

In the CUDA model, each thread executes the kernel independently. Nevertheless, threads within a block can synchronize through a barrier and write simultaneously to shared memory to share data between them. In contrast, thread blocks are completely independent and can only share data through the global memory once the kernel ends.

We employed for the implementations the Nvidia Tesla C2070 GPU. Its specifications can be seen in Table 2.

Table 2. Nvidia Tesla C2070 features.

Number of stream multiprocessors	14
Number of cores	448
Clock rate	1.15 GHz
Global memory	4 GB
Constant memory	64 kB
Shared memory per block	48 kB

The architecture of this GPU is Fermi, hence it supports the maximum parallelism level with several kernel execution overlapping, data copy and kernel execution overlapping, simultaneous host to device and device to host data copy, etc. The installed CUDA toolkit and SDK version is 4.0 [10].

4.2. Proposed implementation

As we already mentioned, MIMO-OFDM systems can achieve a very high spectral efficiency at the expense of high computational complexity. Although the multiuser precoding stage does not seem a priori very computationally expensive, indeed it is, since the precoding stage must be executed for each subcarrier. In this section, the proposed implementation on GPU is described.

A single kernel is employed for the implementation of either the THP or the LRA-THP algorithm, where the initial input data will differ depending on the selected algorithm. The grid configuration for the kernel (number of blocks and block size) is shown in Fig.3. A bidimensional grid is considered with bidimensional blocks with $N_{th} = 16$ threads per dimension. Since the channel is considered to remain constant during L_{ch} time intervals, each thread block will be in charge of the processing of a subgroup of subcarriers associated to these number of intervals. Thus, the number of subcarriers to be processed by each block can be software-defined at the beginning as $N_{sub} = N_{th}^2/L_{ch}$. For the case considered in this work $N_{sub} = 256/20 = 12$ subcarriers/block. Then, for a certain number of subcarriers the number of blocks per grid is obtained as $N_B = N_c/N_{sub}$, being the number of blocks per dimension $\sqrt{N_B}$.

Before starting the process, the channel matrices and signals to be precoded $(\mathbf{H}, \mathbf{s})$ associated to L_{ch} time intervals and N_c subcarriers are stored in GPU global memory. The preprocessing stage of the algorithm is executed and its output data (matrices $\mathbf{L}$ and $\mathbf{Q}^+$) are stored in GPU shared memory, which allows for a faster access. After this, all threads are synchronized to fetch the data from shared memory and start the per-symbol-vector processing stage. Vector $\mathbf{x}$ is obtained and stored temporally in the registers and at the end copied to global memory for its output.

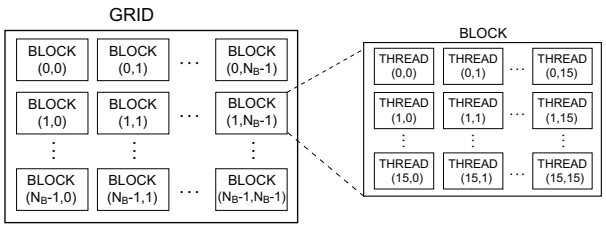

Fig. 3. Grid distribution.

5. RESULTS

In this section, the parallel implementation of the THP and LR-THP algorithms on GPU was compared to their implementations in a high-performance CPU. The selected CPU was an Intel Xeon X5680 at 3.33GHz with 96GB of DDR3 main memory and 12 MB of cache memory running Linux. The compiler used was Intel C compiler (ICC) with the global performance optimization -o3.

Without loss of generality, we consider the same number of transmit and receiving antennas in the system, $N = K$. The transmission is done using N_c subcarriers. As described in LTE Release 10 [8], transmission bandwidths up to 100 MHz can be employed by means of the aggregation of up to five component carriers as the ones in Release 8. Thus, the transmission can be done over a maximum of $5 \times 1200 = 6000$ subcarriers.

Fig. 4 shows the speedup resulting of dividing the computational times to run the algorithms at the GPU by the computational times of the implementations on CPU for a 4×4 system using double precision and different number of subcarriers. It can be observed that, while the GPU-based THP implementation performs up to 7 times faster than its CPU-equivalent, the LRA-THP implementation reaches a speedup up to 5.5. Both speedups are quite promising, but it is interesting to investigate the speedup reduction effect experienced by the LR-THP with respect to the THP. For this purpose, the speedup achieved by the implementation of only the LLL stage was evaluated independently for the same system configurations previously discussed. Results show that this partial speedup ranges between 2 and 3, meaning that this stage acts as a bottleneck for the implementation of the LR-THP.

As described in [10], any flow control instruction can significantly impact the effective instruction throughput

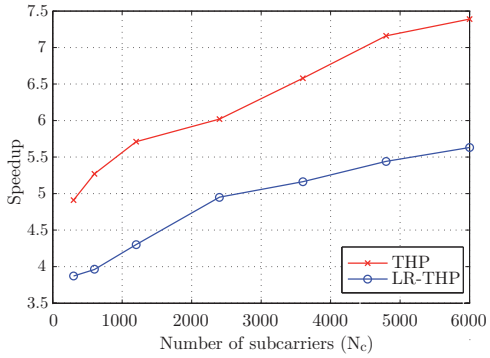

Fig. 4. Speed-up of THP and LR-THP schemes on GPU compared to CPU for $L_{\text{ch}} = 20$, $N = 4$ and 4-QAM.

achieved by the GPU by causing threads of the same warp to diverge (i.e. to follow different execution paths). Indeed this is what happens when the LLL is applied over different channel matrices, since the LLL method contains two *if* statements dependent on the processed channel matrix [5]. As said in [10], this situation causes the serial execution of the different paths, thus increasing the total number of instructions executed for this warp and consequently the computational time.

In addition, the execution time of the THP and LR-THP schemes over GPU is depicted in Fig. 5. It can be seen that LR-THP has a considerably higher execution time than THP, increasing the difference between them as the number of subcarriers increases. Observing the execution time of the LR-stage used in LR-THP scheme, it can be noticed that this stage takes even longer than $L_{\text{ch}} = 20$ runs of the THP. This shows again that the LLL method is not as suitable as the THP for GPU implementation.

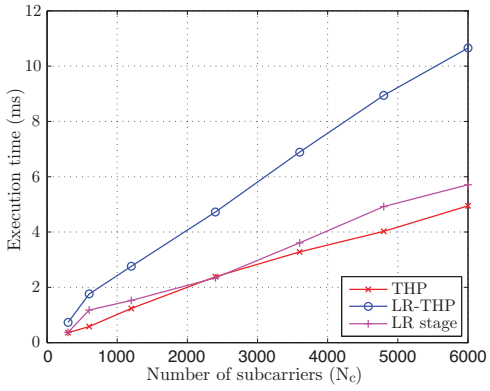

Fig. 5. Execution time of THP and LR-THP schemes over GPU for $L_{\text{ch}} = 20$, $N = 4$ and 4-QAM.

6. CONCLUSION

In this paper, a GPU-based implementation of the well-known Tomlinson-Harashima precoding (THP) scheme combined with a lattice-reduction (LR) stage has been presented. The precoding stage is highly accelerated by processing simultaneously the calculations associated to each subcarrier through forwarding their data to different threads. Moreover, due to the reconfigurable nature of GPUs, it is possible to gate the LR stage off when the user requirements are sufficiently guaranteed by the THP. This way, computational cost and performance can be traded.

The efficiency of the proposed approach was assessed by comparing its computational time to the one taken by an equivalent implementation on a high-performance CPU. Results showed that, while the GPU-based THP implementation performs up to 7 times faster than its CPU-equivalent, the LR stage implementation only achieves a speedup of 3. Despite the fact that the LR cannot be as efficiently parallelized as the THP, a speedup of 5.5 can be achieved when both are combined. These results are promising since there is room for further improvements by optimizing the speedup of the joint scheme.

7. REFERENCES

[1] A. J. Paulraj, D. A. Gore, R. U. Nabar, and H. Bölcskei, "An overview of MIMO communications - a key to Gigabit wireless," *Proceedings of the IEEE*, vol. 92, no. 2, pp. 198–218, Feb. 2004.

[2] M. Wu, Y. Sun, S. Gupta, and J.R. Cavallaro, "Implementation of a high throughput soft MIMO detector on GPU," *Journal of Signal Processing Systems*, Sept. 2010.

[3] G. Falcao, V. Silva, and L. Sousa, "How GPUs can outperform ASICs for fast LDPC decoding," in *Int. Conference on Supercomputing*, Yorktown Heights, New York (USA), 2009.

[4] C. Windpassinger, R.F.H. Fischer, T. Vencel, and J.B. Huber, "Precoding in multiantenna and multiuser communications," *IEEE Transactions on Wireless Communications*, vol. 3, no. 4, pp. 1305–1316, July 2004.

[5] A. Lenstra, H. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Ann.*, vol. 261, pp. 515–534, 1982.

[6] D. Xu, Y. Huang, and L. Yang, "Improved nonlinear multiuser precoding using lattice reduction," *Signal, Image and Video Processing*, vol. 3, no. 1, pp. 47–52, Feb. 2009.

[7] S. Roger, F. Domene, A. Gonzalez, V. Almenar, and G. Piñero, "An evaluation of precoding techniques for multiuser communication systems," in *ISWCS*, York, UK, Sept. 2010.

[8] 3GPP TS 36.201, V10.0.0, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer - General Description," Dec. 2010.

[9] M. Li, D. Novo, B. Bougard, C. Desset, and A. Dejonghe, "A system level algorithmic approach toward energy-aware SDR baseband implementations," in *ICC*, Dresden, Germany, June 2009.

[10] "NVIDIA CUDA C programming guide," *Online at: http://developer.download.nvidia.com/.*