GPU-accelerated SDR Implementation of a Multi-User Detector for Satellite Return Links

Chen Tang, Francisco Lázaro Blasco, *Student Member, IEEE*

***Abstract*—In the past decade new satellite applications have emerged, which require a bidirectional satellite link. Due to the scarcity and high cost of satellite frequency spectrum, it is very important to utilize the available spectrum as efficiently as possible. The efficient usage of the spectrum in the satellite return link is a challenging task, especially if multiple users are present. In previous works Multi-User Detection (MUD) techniques have been widely studied to increase the spectral efficiency of the satellite return link. However, due to the high computational complexity and its sensitivity to synchronization and channel estimation errors, only few implementations of MUD for satellite communications exist. In this paper a novel Graphics Processing Unit (GPU)-based Software Defined Radio (SDR) implementation of a MUD receiver for transparent satellite return link is presented, which uses iterative channel estimation and decoding. In addition to its high flexibility and low cost, with the GPU acceleration our SDR MUD receiver implementation achieves a decoding throughput of 290 Kbps, which is sufficient to operate in real time in satellite return links.**

***Keywords*—muti-user detection, satellite return link, successive interference cancellation, software defined radio, GPU**

I. Introduction

Satellite communication systems are a very good solution when it comes to delivering service to a broad area, since they do not require the deployment of costly terrestrial infrastructures. Traditionally, satellite systems have been used to provide television and radio broadcasting services, which are unidirectional services. However, in the past decade new applications have emerged, which require a bidirectional communication. The most prominent example is internet access for rural areas, where using the satellite service is commonly the only possibility. Moreover, there is a trend towards introducing interactive applications in satellite television services. Hence, it is necessary to provide the user terminals with transmission capabilities through satellite return links, while still maintaining a low cost. Such cheap user terminals have a very limited transmission power and can only transmit with low data rates (at most in the order of 100 kbps). Moreover, the scarcity of frequency spectrum drives satellite operators to use higher frequency bands, where phase noise becomes more problematic for cheap user terminals with low stability oscillators. Therefore, it is challenging to provide an efficient spectrum access in satellite return link in the presence of strong phase noise.

The satellite return link can be considered as a multi-user communication problem. In most of today's systems, the multiple access issue is solved by orthogonalization, such as the case of GSM, in which TDMA is used, or Digital Video Broadcasting - Return Channel via Satellite (DVB-RCS), where Multi Frequency TDMA (MF-TDMA) is used. This means that users are assigned with different resources so that they do not interfere with each other. In our implementation we employ MUD techniques to boost the spectral efficiency in a satellite return link, in which users are assigned with the same resources and their transmissions interfere. Although there is an enormous amount of literature in the field of MUD, there are actually only a few practical implementations for satellite systems [1] [2]. In [1] an implementation of MUD for the return link of a satellite system is presented, which is not designed for the presence of strong phase noise. In [2] a MUD receiver for spread spectrum Aloha is presented. In contrast to [1] and [2], our system is designed to operate in the presence of strong phase noise without employing spreading techniques.

The reasons for the low adoption rate of MUD techniques are twofold. The first reason is the high computational complexity. The second reason is the sensitivity of MUD techniques to synchronization and channel estimation errors. The latter is even more problematic, when the channel is time variant because of phase noise.

Due to the high computational complexity of MUD techniques, implementations are commonly restricted to use programmable hardware devices, such as Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC). However, the development of such hardware devices is very costly and implementations are not flexible to accommodate changes in the communication standards. From the perspective of economic cost and flexibility, SDR implementations of the MUD receiver have big advantages compared to conventional hardware implementations. The software of the SDR implementation can be easily modified to support new radio protocols or system requirements. Another advantage of SDR is its low implementation cost. However, the weak point of SDR implementations is their low throughput, especially when complex MUD algorithms have to be implemented completely in software.

In this paper we present a novel SDR implementation of the MUD receiver, which utilizes GPU acceleration and achieves a decoding throughput of 290 Kbps. The implementation is able

Chen Tang and Francisco Lázaro Blasco are with the Institute of Communications and Navigation, German Aerospace Center (DLR), Oberpfaffenhofen, 82234 Wessling, Germany. Email:{Chen.Tang,Francisco.LazaroBlasco}@dlr.de

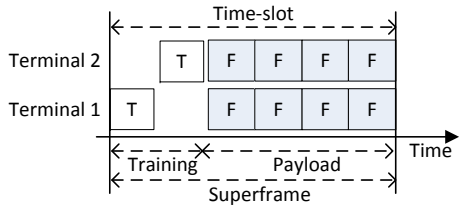

Fig. 1. Transmission inside one time-slot. The block denoted by T represent the training part and the blocks denoted with F represent payload frames.

to achieve a spectral efficiency of 1.1 b/s/Hz in the presence of strong phase noise. The receiver presented in this work uses Successive Interference Cancellation (SIC), iterative channel estimation and Low Density Parity Check (LDPC) channel decoding to increase the spectral efficiency as much as possible and still achieves a decoding throughput that allows real time processing for satellite return links. In this paper a MUD receiver for two users is considered. However, since SIC is used in our system, it can be extended to support more users in a straightforward way.

The article is structured as follows. Section II describes our system model and MUD receiver design. Section III gives a general introduction from the implementation perspective of our MUD receiver. Then, more details of the GPU-based implementation of our MUD receiver are introduced in IV. Afterwards, the simulation results of the Packet Error Rate (PER) and the processing time of our GPU-based SDR MUD implementation are presented in V. Finally Section VI draws the conclusions.

II. System Model and Receiver Design

In this paper we focus on a satellite return link scenario based on MF-TDMA, as it is the case for DVB-RCS. As the transmission scheme the scheduler assigns the same frequency and time-slot to K user terminals, which are assumed to have limited transmission power and low transmission data rate, as introduced in the previous section. In our system the simplified case $K = 2$ is considered. Terminals are assumed to be roughly synchronized. Fig. 1 shows how the transmission of the two terminals is organized inside one time-slot. In each time-slot each terminal transmits a super-frame, which consists of a training part and a payload part. In the training part the first terminal transmits while the second is silent, then the second terminal transmits and the first terminal is silent. This interference free part will be used by the burst demodulator for detection and synchronization purposes. In the second part terminals transmit a number of data frames that collide, which requires MUD to decode.

If only the data part of the time-slot is considered, the received signal $y(t)$ at time instant t can be expressed as:

$$y(t) = \sum_{k=0}^{K-1} \sum_{l=0}^{L} h_k(t - lT_s) x_k(l) p_k(t - lT_s - \tau_k) + n(t) \quad (1)$$

where K is the number of users, L is the number of symbols in a frame, $h_k(t)$ is the complex channel coefficient for user k at time instant t, $x_k(l)$ is the l-th transmitted symbol from user k, $p_k(t)$ is the transmit pulse shape filter of user k, T_s is the symbol duration, τ_k represents the delay of user k with respect to the start of the frame and $n(t)$ is the complex valued additive white Gaussian noise. In our case the channel coefficient is induced by phase noise, $h_k(t) = |h_k| e^{j\phi_k(t)}$. $|h_k|$ is assumed to be constant, while the variation of $\phi(t)$ can be modelled as a random-walk (Wiener) process [3]. Under this assumption the evolution of the channel phase for discrete time instants can be expressed as:

$$\theta_k(t - lT_s) = \theta_k(l) = \theta_k(l-1) + \Delta_k(l) \quad (2)$$

where $\Delta_k(l)$ are i.i.d zero mean real Gaussian random variables with variance ρ_p^2. We will assume that the transmitted symbols $x_k(l)$ belong to an M-PSK alphabet. Bit interleaved coded modulation will be used together with a Soft-Input Soft-Output (SISO) channel decoder.

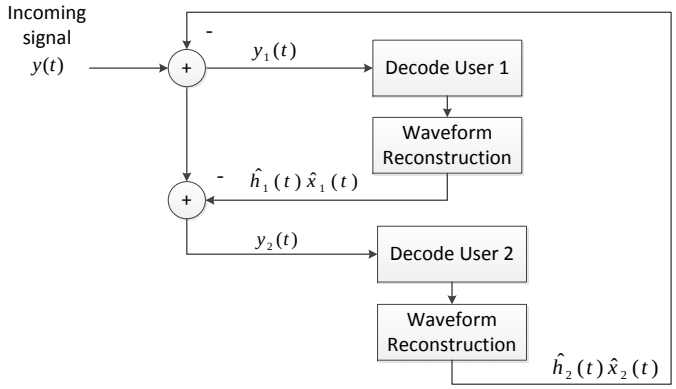

Fig. 2. SIC decoding scheme

The MUD receiver employed is based on SIC [4], as shown in Fig 2. The reason of choosing SIC is that although it is not optimal, its computational complexity increases linearly with the number of system users. SIC is based on a simple idea: User 1 is decoded treating the interference from the user 2 as Gaussian noise. After decoding user 1, its waveform is reconstructed and removed from the received signal. If this reconstruction is perfect, user 2 is now free of interference and can be decoded. In order to reduce the PER, a multi-stage SIC can be used [4], where the SIC process is repeated for several times and each execution of SIC is denoted as a stage. If the user 1 signal cannot be recovered error free after the first stage, a second stage is carried out by reconstructing the signal from user 2 and cancelling it from the received signal on user 1. User 1 will now have a lower interference and can hopefully be decoded correctly. This process is repeated until the maximum number of stages is achieved, or until both users are decoded correctly. Based on SIC, our simplified case with $K = 2$ number of users can be easily extended to support more user terminals.

SIC is very sensitive to channel estimation errors. In order to improve the performance of the SIC receiver, an iterative channel estimation and decoding algorithm is used. The basic idea of such an algorithm is to use not only pilot symbols

but also data symbols to estimate the channel. In our receiver the Expectation-Maximization (EM) algorithm is applied as described in [5], where the E-step corresponds with the channel decoding and the M-step with the channel estimation. The equations used for the M-step are the following [4]:

$$|\hat{h_k}| = \frac{\sum_{t_1=0}^{L-1} Re\left(\hat{x}_k^*\left(t_1\right) y_k\left(t_1\right)\right)}{\sum_{t_1=0}^{L-1} |\hat{x}_k^2\left(t_1\right)|} \tag{3}$$

$$\hat{\theta}_k\left(t_1\right) = \text{Arg} \sum_{t_1=t-W}^{t+W} \hat{x}_k^*\left(l\right) y_k\left(l\right) \tag{4}$$

$$\hat{h}_k\left(t_1\right) = |\hat{h}|_k e^{j\hat{\theta}_k(t_1)} \tag{5}$$

$$\hat{\sigma}_k^2 = \frac{1}{2L} \sum_{t_1=0}^{L-1} |y_k\left(t_1\right) - \hat{h}_k\left(t_1\right) \hat{x}_k\left(t_1\right)|^2 \tag{6}$$

where y_k is the input signal of the single user receiver of the user k, which is the received signal minus the reconstructed signal from the other users. $\hat{x}_k$ is the output of the SISO channel decoder. In order to estimate the channel amplitude $|\hat{h}_k|$ for each user and the noise level $\hat{\sigma}_k^2$, the assumption is made that these magnitudes stay constant over the duration of the whole frame L. For the estimation of the channel phase $\hat{\theta}_k\left(t_1\right)$ at discrete time instant t_1, the channel phase is assumed to be constant over a window of $2W+1$ samples centered on t_1.

Prior to SIC decoding it is necessary to perform frequency, frame and time synchronization. In order to make synchronization easier, a training part is introduced in each superframe, which is free of multiuser interference, see Fig. 1. Using this training part, it is possible to perform frequency, frame and timing estimation using single user synchronization algorithms. In particular, in our implementation the Oerder & Meyr algorithm [6] is used for time synchronization. Timing synchronization is not affected by phase noise. However phase noise does have a negative impact in frequency and frame synchronization. For the frame synchronization the Villanti *et al.* algorithm [7] is used, which was specially developed to perform frame estimation in the presence of phase uncertainty due to doppler or phase noise. This algorithm breaks the synchronization sequence into smaller subsequences and performs correlation of the incoming signal with these subsequences. Then the metrics obtained are summed up with the correlation. Frequency estimation is done using the Rife and Boorstyn algorithm [8]. This algorithm estimates the carrier frequency by performing a FFT of the received signal. Its performance is affected by the presence of phase noise, but it achieves the Cramer-Rao lower bound.

III. SDR-BASED MUD RECEIVER

In general, the current data rates used in satellite return links are moderate/low, e.g. the baud-rate used in our system for both user terminals is 62500 symbols per second. Thus, it is feasible for a SDR-based MUD receiver to operate in real time. The real time threshold $T_{real-time}$ is defined as:

$$T_{process} \leq T_{real-time} = T_{frame} \tag{7}$$

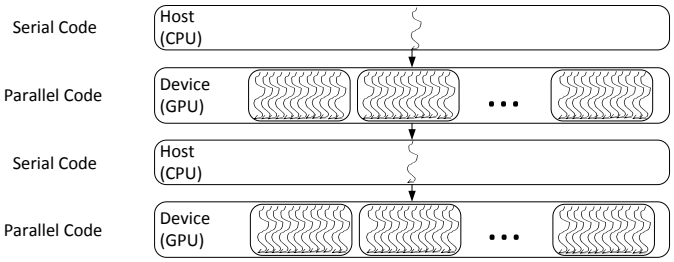

Fig. 3. GPGPU implementation structure

i.e. the average processing time $T_{process}$ of a data frame is lower than its frame duration T_{frame}. In our case, each frame is composed of 4800 data symbols, 63 header symbols and 440 pilot symbols, which results in $T_{frame} \approx 85ms$. However, due to the computational complexity of the algorithms used in our MUD receiver, it is still challenging to achieve a real time SDR-based MUD system even with such a moderate data rate.

Conventionally programmable hardware devices, such as Digital Signal Processors (DSPs) and FPGAs together with Radio Frequency (RF) transceiver are used to implement a SDR system. For instance, the real time implementations of MUD receiver by using dedicated DSPs and FPGAs were presented in [9] [10] [11]. A number of dedicated hardware implementations of LDPC decoders have also been proposed in the past few years [12] [13]. However, the drawbacks of high cost, considerable development effort and lack of flexibility of FPGAs and DSPs restrict the deployment of SDR technologies [14]. As an example, dedicated hardware-based LDPC implementations have to start a new design and development process to adapt to a different coding design, since the structure of the LDPC decoder changes according to the parity check matrix.

In recent years owing to the tremendous parallel computational power and very high memory bandwidth, general-purpose computing on graphics processing unit (GPGPU) has become an attractive alternative to FPGA and DSP to achieve high-performance SDR systems. GPUs are originally designed to optimize the floating-point calculations for image and video rendering. GPGPU technique utilizes a GPU to perform general computations in applications traditionally handled by the CPU. The implementation of a GPGPU in a conventional CPU-based system is illustrated in Fig. 3. The parts of the application that need to be executed serially are implemented on CPU. The parts that are very computationally intensive and can be parallelized are deployed on the GPU.

State of the art high-end GPUs consist of thousands of processing cores, which can provide over 3 tera-floating operation-per-second (TFLOP) computing capability [15]. Moreover, with novel GPGPU architectures, such as Compute Unified Device Architecture (CUDA) introduced by Nvidia [15], applications running on a GPUs can be programmed with high-level C-like languages, which makes it easier to implement and modify a GPU-driven SDR system. Therefore, due to its high computing capability, low cost and high flexibility, GPGPU has become a very promising solution for modern SDR systems.

For instance, many SDR blocks with high computational complexity and inherent parallelism, such as channel estimation, channel decoding and digital filters can be implemented on GPU. In [16] a high throughput SDR-based multiple-input and multiple-output (MIMO) detector is implemented on GPU. GPU-accelerated LDPC decoder has also been widely studied [17] [18] [19] [20].

In this work we adopt CUDA to exploit GPU to accelerate the SDR implementation of our MUD system. The implementation details of our GPU-accelerated MUD receiver will be given in the next section.

IV. Implementation of GPU-accelerated MUD receiver

Based on the receiver design introduced in Sec. II, all components of our MUD receiver are implemented in software, except for a reconfigurable hardware for sampling and analog-to-digital conversion. As some of the algorithms of the MUD receiver used in our work have very high computational complexity, we exploit the CUDA programming model to accelerate our SDR implementation on GPU.

As the first step of a GPU implementation using CUDA, it is important to assess the application to locate the main processing bottlenecks that can be parallelized and accelerated by GPU [21]. After some performance profilings are carried out, the main processing bottlenecks of the MUD receiver can be summarized as follows:

- Channel decoding: In our system, an irregular LDPC channel code is used. The codeword length is 4800 symbols for both user terminals and the code rates are $2/3$ and $1/2$ for user terminal 1 and 2 respectively. The Min-Sum LDPC decoding algorithm [22] is implemented. The maximum number of iteration is set to 50. Although the min-sum algorithm is considered as one of the most efficient algorithms used for LDPC decoding with lower workload than the well known Sum-Product algorithm (SPA) [23], its computational complexity is still quite significant. Despite the use of early termination techniques, in which the decoding stops when a valid codeword is detected, the LDPC channel decoder is still one of the main processing bottlenecks.
- Channel estimation: The EM channel estimation we deployed is based on filtering operations in the time domain, as given in Eq. (3), (4) and (5). In our case, a long data set with thousands-points FFT has to be calculated, which is also a very intensive computation. Moreover, it has to iterate several times with the channel decoder.
- Downsampling and interference cancellation: in our implementation oversampling factor 8 is used. Due to this high oversampling factor, interference cancellation and downsampling take a substantial amount of time.
- Data transfer between host (CPU) and device (GPU) memory: Because of the EM algorithm we employed, the iterative channel estimation and channel decoding has to be repeated several times, in our case 4 EM iterations are used. Thus, if there is any memory copy between host (CPU) and device (GPU) during the EM iterations, such a data transfer has to be repeated for 4 times. In order to achieve better processing performance, it is important to minimize such data transfer between host and device [21], as the peak theoretical GPU memory bandwidth is much higher than the peak bandwidth between host (CPU) memory and device (GPU) memory.

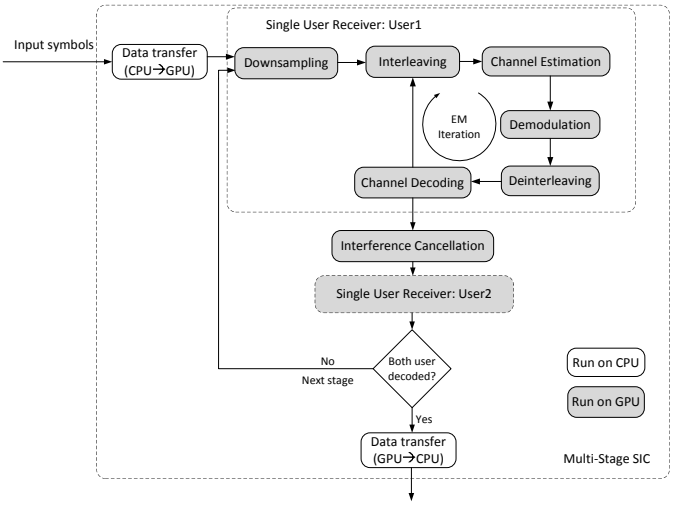

Fig. 4. Multi-Stage SIC MUD receiver on GPU

Based on this assessment of our MUD receiver, we decided to implement all of the components of each single user receiver together with the interference cancellation on GPU, as illustrated in Fig. 4. Although some components of the single user receiver demonstrate little speedup from the GPU implementation, the benefit of running all components on GPU is to minimize the latency of intermediate data transfer between the host and device memory.

In order to have a benchmark for the processing performance, we duplicate the implementation of our MUD receiver in C++ running on CPU. The simulation results are given in the next section.

V. Simulation results

In this section the simulation results in terms of PER and the processing performance of our GPU-based implementation of the MUD receiver are presented. These results are compared with the CPU-based implementation. The specifications of the devices are given in Tab. I. As introduced before, only two satellite terminals are assumed to share the satellite return link at the same time. The results presented in this section correspond to a scenario, in which the power of user 1 is 3 dB higher than the power of user 2, which is a realistic setting for a satellite return link. The number of SIC stages is fixed to 3. The phase noise of the terminals resulted in a standard deviation of 2.29° for the phase increments between consecutive symbols.

The PER performance of the GPU-based implementation for each user terminal is presented in Fig. 5. On the other hand, the

TABLE I. IMPLEMENTATION DEVICES

	CPU	GPU
Platform	Intel Xeon(R) E5620	Nvidia Tesla C2070
Cores	4 (used only one core)	448
Clock rate	2.4GHz	1.15GHz

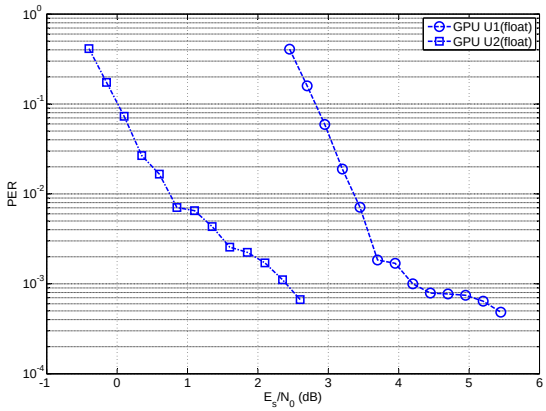

Fig. 5. PER of the GPU-based MUD implementation

CPU-based implementation gives the same PER performance with the GPU-based solution. Thus, it is omitted in Fig. 5 for the sake of clarity.

The total processing time of the MUD implementation is defined as:

$$T_{process} = T_{ds} + T_{ch\_est} + T_{ch\_dec} + T_{ic} + T_{misc} \quad (8)$$

which consists of the time for downsampling T_{ds}, channel estimation T_{ch_est}, channel decoding T_{ch_dec}, interference cancellation T_{ic} and T_{misc} that includes memory operations and the remaining miscellaneous components of the MUD given in Fig. 4. As given in Eq. (7), the real time processing threshold can be defined as:

$$T_{real-time} = 85ms \quad (9)$$

Tab. II shows a comparison of the processing time between the CPU-based and GPU-based implementations for both user terminals to decode one frame. The total processing time reduces with the increase of SNR. In the case of higher SNR, due to the early termination scheme we used, the channel decoder can skip the remaining decoding iterations and finish the decoding process as long as the valid codeword is detected. On the other hand, with the lower PER in the high SNR range, there is a high probability that both users can already be decoded properly in the first SIC stage, in which case the successive SIC stages can be skipped. The processing time consumed by the downsampling and the interference cancellation stays nearly constant with different SNRs.

Compared to the CPU-based solution, with the GPU acceleration, the processing time of the MUD receiver can be significantly reduced, especially for the part of the LDPC channel decoder. The channel decoding can be sped up by a factor 10 by use of the GPU. A channel decoding throughput of 2.0 Mbps is achieved for terminal 1 (code rate $2/3$) with $E_s/N_0 = 3.45$ dB and 1.9 Mbps for terminal 2 (code rate $1/2$) with $E_s/N_0 = 0.60$ dB. The parts of the downsampling and interference cancellation of MUD receiver can also be parallelized and accelerated by GPU. Because of the GPU initialization and the latency of memory transfer between CPU and GPU, the *miscellaneous* part of the GPU-based implementation shown in Tab. II takes more time to process. From this simulation result we can see that our GPU-accelerated MUD solution can perform much better than the real time processing requirement defined in Eq. (9) in terms of the overall processing time for both user terminals to decode each packet, even for the SNR as low as 2.70 dB for terminal 1 and -0.15 dB for terminal 2.

TABLE II. COMPARISON OF TOTAL PROCESSING TIME OF THE MUD IMPLEMENTATION IN SINGLE PRECISION

E_s/N_0 (dB)		$T_{ds}(\%)$	$T_{ch\_est}(\%)$	$T_{ch\_dec}(\%)$	$T_{ic}(\%)$	$T_{misc}(\%)$	$T_{process}(ms)$
User1: 2.45 User2: -0.40	CPU	4.5	7.8	77.9	7.6	2.2	542.76
	GPU	4.7	15.8	46.2	10.31	23.0	82.81
	Speedup factor	6.3	3.2	11.0	4.9	0.6	6.5
User1: 3.45 User2: 0.60	CPU	7.8	11.8	62.0	14.3	4.0	147.12
	GPU	7.2	19.8	21.4	16.7	34.9	28.63
	Speedup factor	5.6	3.1	14.9	4.4	0.6	5.1
User1: 5.45 User2: 2.60	CPU	16.8	15.4	28.9	25.9	13.0	65.25
	GPU	8.5	19.8	11.0	18.2	42.5	19.5
	Speedup factor	6.6	2.6	8.8	4.7	1.0	3.3

Our GPU-accelerated LDPC channel decoder could be further optimized by using a coalesced GPU memory access pattern [17] [18] [19] [20]. However, in order to realize coalesced GPU memory access for LDPC decoder, a complex translating array for the data address transformation is needed, which increases the implementation complexity. On the other hand, the performance of the GPU-based LDPC channel decoder can also be increased by using fast GPU on-chip cached memory [20]. But the state of the art GPUs like the one we used (Nvidia Tesla C2070) support cache for the off-chip global memory. Therefore, it is of little benefit to use on-chip cached memory in our implementation. Moreover, the further optimization of LDPC channel decoder on GPU would only provide a marginal gain on the total processing time, since the LDPC decoding only amounts to less than 20% of the total processing time of the MUD receiver in most of operational SNRs.

VI. CONCLUSION

In this article, we described the system design and the implementation of a MUD receiver using SIC for a satellite return link based on MF-TDMA. The receiver is able to cope with the imperfections of cheap user terminals and increase the spectral efficiency of satellite return links. With the acceleration of an affordable commercial GPU, our SDR implementation of the MUD receiver is able to operate in real time. Moreover, compared to the dedicated hardware implementation, our SDR-based solution has higher flexibility and lower development cost.

VII. ACKNOWLEDGEMENTS

The authors were supported by the Space Agency of the German Aerospace Center and the Federal Ministry of Economics and Technology based of on the agreement of the German Bundestag, under support code 50 YB 0905. The

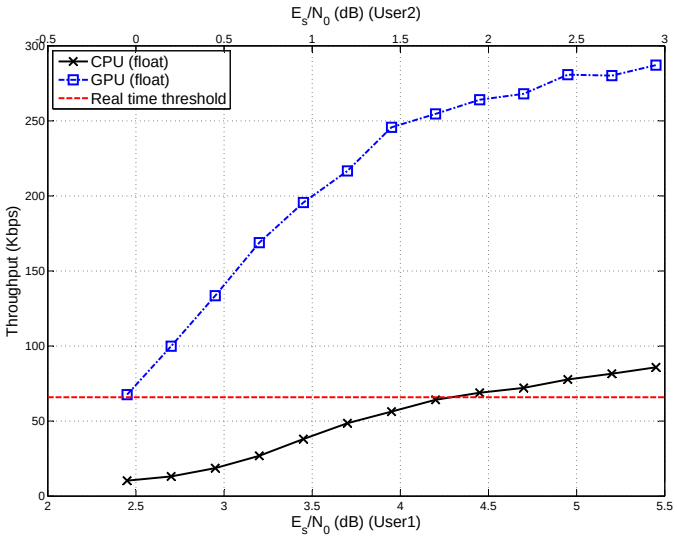

Fig. 6. Throughput of the MUD implementation

authors would like to acknowledge Federico Clazzer and Dr. Svilen Dimitrov for the useful discussions.

REFERENCES

[1] P. Fines, P. Febvre, and E. Christofylaki, "Turbo-code division multiple access: capacity enhancement of mobile satellite systems using narrow-band multiuser detection," in *Proc. of 10th International Workshop on Signal Processing for Space Communications*, Rhodes Island, Greece, Oct. 2008.

[2] F. Basile and G. Mendola, "Satellite hub communication system GPU based," in *Proc. of GPU Technology Conference*, San Jose, CA, USA, May 2012.

[3] G. Colavolpe, A. Barbieri, and G. Caire, "Algorithms for iterative decoding in the presence of strong phase noise," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 9, pp. 1748–1757, 2005.

[4] F. Lázaro Blasco and F. Rossetto, "On the derivation of optimal partial successive interference cancellation," in *Proc. of IEEE Globecomm*, Houston, Texas, US, Nov. 2011.

[5] M. Kobayashi, J. Boutros, and G. Gaire, "Successive interference cancellation with SISO decoding and EM channel estimation," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 8, pp. 1450–1460, Aug. 2001.

[6] M. Oerder and H. Meyr, "Digital square and filter timing recovery," *IEEE Trans. Commun.*, vol. 36, pp. 605–612, May 1988.

[7] M. Villanti, P. Salmi, and G. E. Corazza, "Differential post detection integration techniques for robust code acquisition," *IEEE Trans. Commun.*, vol. 55, no. 11, pp. 2172–2184, Nov. 2007.

[8] D. Rife and R. Boorstyn, "Single tone parameter estimation from discrete-time observations," *IEEE Trans. Inf. Theory*, 1974.

[9] S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, "Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers," *IEEE Trans. Wireless Commun.*, vol. 1, no. 3, pp. 468–479, Jul. 2002.

[10] J. Tranquilli, J. Farkas, J. Niedzwiecki, B. Pierce, L. Brothers, and J. Debardelaben, "Real time implementation of a multiuser detection enabled ad-hoc network," in *Proc. of Military Communications Conference*, San Diego, California, USA, Nov. 2008.

[11] I. Seskar and N. Mandayam, "A software radio architecture for linear multiuser detection," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 5, pp. 814–823, May 1999.

[12] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 6, Jun. 2007.

[13] S. Muller, M. Schreger, M. Kabutz, M. Alles, F. Kienle, and N. Wehn, "A novel LDPC decoder for DVB-S2 IP," in *Proc. of Design, Automation and Test in Europe*, Nice, France, Apr. 2009.

[14] J. Kim, S. Hyeon, S. Choi, and H. University, "Implementation of an sdr system using graphics processing unit," *IEEE Commun. Mag.*, pp. 156–162, Mar. 2010.

[15] Nvidia, "Nvidia CUDA Zone." [Online]. Available: https://developer.nvidia.com/category/zone/cuda-zone

[16] M. Wu, Y. Sun, S. Gupta, and J. Cavallaro, "Implementation of a high throughput soft MIMO detector on GPU," *Journal of Signal Processing Systems*, vol. 64, no. 1, pp. 123–136, Jul. 2011.

[17] G. Falcao, L. Sousa, and V. Silva, "Massively ldpc decoding on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 309–322, Feb. 2011.

[18] C. Chang, Y. Chang, M. Huang, and H. B., "Accelerating regular ldpc code decoders on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ*, vol. 4, no. 3, pp. 653–659, 2011.

[19] G. Wang, M. Wu, Y. Sun, and C. J.R., "A massively parallel implementation of QC-LDPC decoder on GPU," in *Proc. of IEEE 9th Symposium on Application Specific Processors*, San Diego, California, USA, 2011, pp. 82–85.

[20] S. Kang and J. Moon, "Parallel LDPC decoder implementation on GPU based on unbalanced memory coalescing," in *Proc. of IEEE Int. Conf. on Commun.*, Ottawa, Canada, Jun. 2012.

[21] Nvidia, "CUDA C Best Practices Guide," Oct. 2012. [Online]. Available: {http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf}

[22] N. Wiberg, "Codes and decoding on general graphs," *Ph.D. dissertation, U.Linkping*, 1996.

[23] S. Lin and D. Costello, *Error Control Coding*. Prentice Hall, 2004.