# PARALLELIZATION ALGORITHM FOR TURBO DECODING AND ITS IMPLEMENTATION ON GPU FOR SDR-BASED LTE SYSTEM

Saehee Bang (HY-SDR Research Center, Hanyang Univ., Seoul, South Korea; say_0618@dsplab.hanyang.ac.kr); Chiyoung Ahn (HY-SDR Research Center, Hanyang Univ., Seoul, South Korea; ahncy@dsplab.hanyang.ac.kr); Sungsoo Ahn (Myongji College; ssan@mjc.ac.kr); and Seungwon Choi* (corresponding author, HY-SDR Research Center, Hanyang Univ., Seoul, South Korea; choi@ieee.org)

## ABSTRACT

In this paper, we first presented a parallelized turbo decoding algorithm, which is widely used as a tool for Forward Error Correction (FEC) . Then, the proposed parallelized algorithm was implemented on a Graphic Processor Unit (GPU) board. We analyzed the performance of implemented turbo decoder on an Software Defined Radio (SDR) -based Long Term Evolution (LTE) system. Turbo codes have been adopted in many communication standards such as Worldwide Interoperability for Microwave Access (WiMAX) , Wideband Code Division Multiple Access (WCDMA) , LTE, etc. However, since the Maximum a Posteriori (MAP) decoder, which is a core part of turbo decoder, needs excessive memory requirements and heavy computational complexity, implementation of turbo decoder on SDR system brings about many severe difficulties in practice. The proposed parallelization algorithm tremendously reduces the decoding time caused by the pair of MAP decoders included in turbo decoder.

## 1. INTRODUCTION

Forward Error Correction (FEC) technique for wireless signal environments needs a fast and high quality processing capability in order to be able to retrieve the transmit information from distorted receive data. Turbo code [1] has been known as a good error correction method providing Shannon's limit [2] with a relatively simple structure.

Despite these wonderful features, however, implementation of turbo decoder suffers from tremendous amount of computational load and large delay time, which brings about many practical problems in real-time processing.

A way of overcoming the problem of heavy computational load is to employ a high speed processor that is capable of parallel processing.

Recently, Graphic Processor Unit (GPU) has been introduced as a Single Instruction Multiple Data (SIMD) parallel processor which supports various applications including high speed floating-point parallel operations for 3-dimensional graphic processing [3].

In addition, the high-speed applications for GPU are even more accelerated with the C-based high-level language, Compute unified device architecture (CUDA). Indeed, GPU is a lot more flexible than Field Programmable Gate Array (FPGA) and a lot faster than Digital Signal Processor (DSP).

In this paper, we propose a novel parallelization algorithm for high-speed turbo decoder using CUDA that is appropriate for SIMD architecture. The proposed algorithm is implemented on a GPU board of NVIDIA GeForce GTX 260. Section 2 shows basic architecture of the implemented turbo decoder adopting proposed algorithm, while Section 3 introduces the proposed algorithm and system implementation using the GPU. Section 4 demonstrates the system performance obtained from various experimental tests, and Section 5 concludes this paper.

## 2. TURBO DECODING ALGORITHM

The terminology, turbo, originates from the operation principle of the decoder which enhances its performance by having its output fed-back to its input for iterative processing, which is very similar to the principle of turbo engine adopted in vehicles. This section summarizes the basic concept and operational principle of turbo decoder.

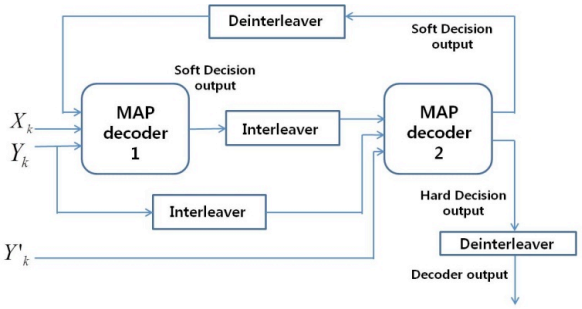


**Figure 1. Architecture of turbo decoder**

Figure 1 illustrates a general architecture of turbo decoder. Turbo decoder input is Log Likelihood Ratio (LLR) output, while the output of turbo decoder is binary bits of hard decision. Basically, turbo decoder consists of two MAP decoders, interleaver, and deinterleaver. MAP decoders shown at the front and back part of Figure 1 is composed of SISO (Soft Input Soft Output) decoder of which the output is used repeatedly for iterative decoding.

The first MAP decoder generates soft decision information using the received information bits and parity bits, which are used by the second MAP decoder as an input after rearranging the bit order through interleaving. The second MAP decoder also generates soft decision information using the received bits that are rearranged by interleaver, parity bits, and the output of the first MAP decoder. The output of the second MAP decoder is feedback to the first MAP decoder for the iterative decoding. This procedure is repeated by a preset number of iterations in order to obtain a desired BER (Bit Error Rate).

## 3. IMPLEMENTATION OF PARALLELIZATION ON GPU

### 3.1 CUDA (Compute Unified Device Architecture)

In a GPU, there are a lot of Arithmetic Logic Units (ALUs) for 3-dimensional graphic processing. CUDA is a C-based extended high-level programming environment for using in various general applications by appropriately managing the large number of ALUs of GPU. Thread in CUDA environment denotes a computing unit. Since CUDA supports as many threads as the number of computing units not the number of cores, GPU can exhibit an extremely high-quality parallel processing capability through an efficient program coding with the multiple threads [4].

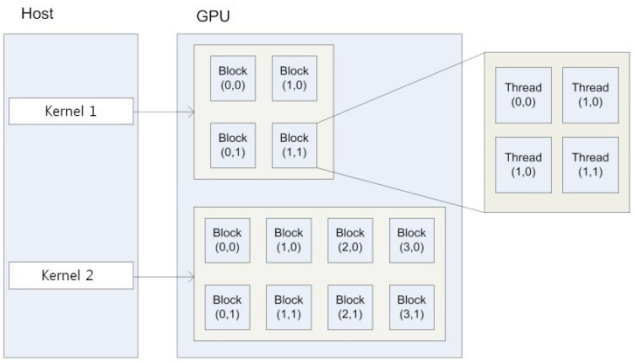


**Figure 2. CUDA kernel architecture**

Figure 2 illustrates CUDA kernel architecture [5]. As shown in Figure 2, GPU consists of a number of thread blocks, while block identifier (ID) and thread ID are properly assigned to each block and thread for managing the blocks and threads. As all the threads in a block execute the same instruction, programmer can control the operations to be performed at each thread using block ID and thread ID. Consequently, GPU is equipped with a SIMD architecture of which the multiple data can be controlled and processed through a single instruction. It also implies that GPU is advantageous for providing an efficient parallelism using multiple components each of which is performing the same operation, although it is impossible to perform different operations at each of the multiple components in parallel like in FPGA.

### 3.2 PARALLEL ALGORITHM

For parallelizing the turbo decoder, we consider a method of partitioning the entire data block into many sub-blocks [6].However, since the procedure of computing the information sequence of each sub-block is dependent upon one another, which would cause the initial metrics of each sub-block to become inaccurate, arbitrary partitioning will severely degrade the performance of turbo decoder. Once the initial metrics become inaccurate due to partitioning, error is spread into all the stages of trellis procedure such that the performance degradation becomes worse and worse.

In order to resolve the problem of error spreading, one could increase the training sequence length to enhance the reliability of initial metrics. However, to increase the length of training sequence brings about excessive time delay and extra hardware complexity. Another way of resolving the error spreading is to use the metrics obtained in between adjacent sub-blocks [7]. However, that algorithm suffers from severe performance degradation when the sub-block is too short. In addition, the very first decoding generates inaccurate soft decision, which causes decoding loss iteratively.

Considering all the problems mentioned above, we propose a novel method of partitioning the data block into many sub-

blocks in such a way that the algorithm of using the metrics in between adjacent sub-blocks is improved for maintaining a good performance with as short sub-block as possible, which brings correspondingly short decoding delay.

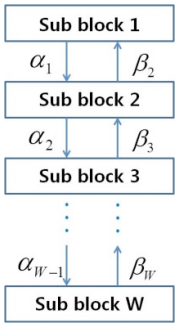


**Figure 3. Initial state value computation in the proposed algorithm**

Figure 3 illustrates how the initial state values are computed at each of sub-blocks. As shown in Figure 3, each of adjacent sub-blocks computes the forward static metrics and backward static metrics to provide $\alpha$ and $\beta$ for the next sub-block to compute the forward and backward static metrics iteratively. Using the architecture shown in Figure 3, since the next sub-block is provided the starting static metrics from the previous sub-block, decoder output can be generated rapidly with a high accuracy. Consequently, performance degradation due to partitioning can be minimized, maintaining a very short decoding delay with a short sub-block length.

Proposed algorithm performs the decoding procedure as follows.

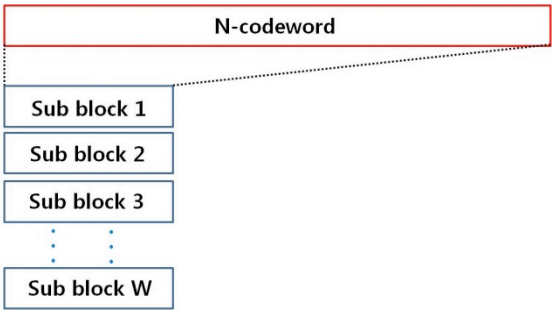


**Figure 4. Decoding of N-code word with W partitioned sub-blocks**

N-codeword input is partitioned into W sub-blocks as shown in Figure 4.At each pair of adjacent sub-blocks, one sub-block computes forward static metrics first while the other sub-block computes the backward static metrics first. Using (1) and (2), obtained from branch metrics shown in (3), forward and backward static metrics are computed and the results are stored.

$$\alpha_k(s_k) = \max^*_{s_{k-1}\in K}(\alpha_{k-1}(s_{k-1}) + \gamma(s_{k-1}, s_k)) \quad (1)$$

$$\beta_k(s_k) = \max^*_{s_{k+1}\in K}(\beta_{k+1}(s_{k+1}) + \gamma(s_{k+1}, s_k)) \quad (2)$$

where

$$\gamma_k(s_{k-1}, s_k) = \left(L_c(y_k^s) + L_a(y_k^s)\right)u_k + L_c(y_k^p)p_k \quad (3)$$

At each pair of sub-blocks mentioned in step 2), the sub-block computed the forward static metrics now computes the backward static metrics, and vice versa. Note that the values for $\alpha$ and $\beta$ are handed over to be used as starting static metrics as depicted in Figure 3.

While performing step (3), each sub-block computes the output value as shown in (4).

$$L_e(k) = \max^*_{s_k\in K}(\Lambda(s_k|u_b = 0)) - \max^*_{s_k\in K}(\Lambda(s_k|u_b = 1) - L_a(y_k^s) - L_c(y_k^s) \quad (4)$$

The output obtained in step (4) is provided to decoder input after interleaving or de-interleaving in order to repeat the procedures from step (2) by preset number of iterations. After performing the above procedures by preset number of iterations, final decoder output is obtained through a hard decision.

In the proposed algorithm, since the static metrics at the boundary of adjacent blocks are used as starting static metrics, the effective length of training sequence is increased by the sub-block length at the adjacent block. Due to the increased length of training sequence, which will enhance the reliability of starting static metrics, the final performance is improved. Consequently, performance degradation due to short training sequence can be reduced. It rather enhances the decoding delay tremendously.

In short, by exchanging the static metrics between two processors associated with adjacent sub-blocks, the proposed algorithm has resolved the inherent problem of block partitioning method, i.e., performance degradation due to the small size of sub-blocks.

### 3.3 IMPLEMENTATION

MAP decoder takes the largest part in the operation delay of turbo decoder. Among the operations in the MAP decoder, operations of (1) and (2) are the major part of MAP decoder. Considering that the operations for (1) and (2) are iterative, which means that the current values are determined by previous values, we have performed the computation of (1) and (2) using a parallel processing through the method of exchanging the computed values of those two equations after partitioning the entire data block into many sub-blocks.

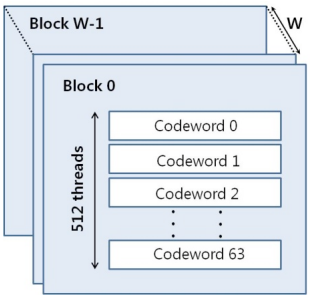


**Figure 5. Operational architecture of GPU parallelization for MAP decoder with 64 sub-block**

Figure 5 illustrates kernel architecture designed for CUDA programming of our implemented system. As shown in Figure 5, parallelization processing of turbo decoding has been performed efficiently by assigning as many threads in each CUDA block as the number of states in each sub-block, while CUDA blocks are assigned as many as the number of sub-blocks of turbo decoder

In our implemented system, interleaving has also been performed in high-speed operation using parallelization technique. For this, the number of threads should be as large as possible for the parallelized operation. Through the parallelization coding, GPU can perform the turbo decoding as discussed above. Since GPU is a floating-point processor, the turbo decoding can be processed using large enough soft-decision values.

## 4. PERFORMANCE EVALUATION

The operation time of our implemented system has been measured using NVIDIA GeForce GTX260 to be compared to the turbo decoder implemented with Texas Instruments' TMS320C6201 [8].

For the experimental tests, a 1/3 coding rate turbo code prepared in accordance with 3GPP release 9 [9] has been adopted, while some algorithm to calculate the LLR values [9] is used for the soft input of turbo decoder.

For the partitioning, 6144-bit data block is divided into a single 6144-bit sub-block, 32 of 192-bit sub-blocks, 64 of 96-bit sub-blocks, and 128 of 48-bit sub-blocks.

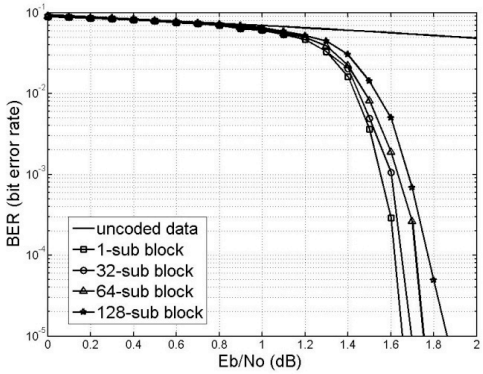


**Figure 6. BER performance**

Figure 6 illustrates the BER performance of proposed method according to various number of sub-blocks. As shown in the figure, performance is degraded as the number of sub-blocks is increased, which causes the block size to be decreased.

Table 1 shows the operation time required for turbo decoder according to the various number of sub-blocks.

| Number of Sub-block | Total Processing Time for 6144 bits | Processing Speed |
|---|---|---|
| 1 | 10.24ms | 600Kbits/s |
| 32 | 4.096ms | 1.5Mbits/s |
| 64 | 2.833ms | 2.2Mbits/s |
| 128 | 1.708ms | 3.6Mbits/s |

**Table 1. Operation time taken for different number of sub-blocks**

From table 1, our implemented system is 2.5, 3.6 and 6 times faster than the turbo decoder implemented with Texas Instruments' TMS320C6201 providing 500k bits/s [8] of processing speed when the number of sub-blocks is 32, 64 and 128, respectively.

<64 sub-blocks>

| Method | GPU time (μs) | %GPU time |
|---|---|---|
| Interleaver+deinterleaver | 19.104 | 0.674 |
| Memcpy | 10.576 | 0.373 |
| Other calculation | 96.587 | 3.409 |
| MAP decoder | 2706.96 | 95.543 |
| Total time | 2833.227 | 100 |

<128 sub-blocks>

| Method | GPU time (μs) | %GPU time |
|---|---|---|
| Interleaver+deinterleaver | 19.104 | 1.118 |
| Memcpy | 10.576 | 0.619 |
| Other calculation | 91.414 | 5.352 |
| MAP decoder | 1586.96 | 92.91 |
| Total time | 1708.054 | 100 |

**Table 2. Computation time for the parallelized Turbo decoder**

Table 2 shows a computation time taken for each operation required in our implemented system, which has been measured using CUDA Visual Profiler provided by NVIDIA. In the table, "Memcpy" denotes the procedure of memory copy between Central Processing Unit (CPU) and GPU, while "other calculation" denotes the procedure of data processing of decoder input and hard decision for the final decoder output. In addition, "% GPU time" denotes the portion of operation time taken by each function in the entire GPU processing time.

Table 2 illustrates the portion of MAP decoder in the entire processing time required to turbo decoder as a function of the number of sub-blocks.

From Table 2, it can be observed that the total processing time is reduced by about 1.6 times as the number of sub-blocks is increased by twice. Since the operation is not partitioned at bridge areas where the values are exchanged between sub-blocks and where the sub-blocks are divided and merged, the operation time is enhanced only by 1.6 times instead of 2 times while the number of sub-blocks is increased by twice. As the parallel processing in GPU can be performed using a large number of threads simultaneously, the speed-up operation can be further enhanced as the number of sub-blocks is increased.

## 5. CONCLUSION

We designed and implemented a parallel processing turbo decoder by fully exploiting the parallel processing capability of GPU.

Operation speed in our implemented system has been shown to be 1.5, 2.2 and 3.6 Mbits/s when the number of sub-blocks is 32, 64 and 128, respectively, which means the processing time can be saved further as the number of sub-blocks is increased. The partitioning into 128 sub-blocks provides about 6 times faster processing time compared to a normal turbo decoder, the 0.15dB degradation seems to be tolerable. It has also been found that our implemented system with 128 sub-blocks is about 6 times faster than TMS320C6201's turbo decoder. From various experimental tests, we conclude that the proposed turbo decoder which fully exploits the parallel processing capability with the sub-block partitioning is suitable for speed-up operation of modern communications including LTE.

## ACKNOWLEDGEMENTS

This research was supported by the ICT Standardization program of MKE(The Ministry of Knowledge Economy)

## 6. REFERENCES

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding : Turbo-codes (1)," IEEE International Conference on Communications. ICC, Vol.2,1993

[2] C. E. Shannon, "A Mathematical Theory of Information" Bell Systems Technical Journal, Vol.27, 1948

[3] NVIDIA; "NVIDIA CUDA C Programming Guide", Ver.4.2, 2012

[4] Kim, J., Seungheon Hyeon, Seungwon Choi, "Implementation of an SDR system using graphics processing unit", IEEE Communication Magazine, Vol.48, 2010.

[5] NVIDIA; "NVIDIA CUDA Programming Guide", Ver.2.2, 2010

[6] Jae-Ming Hsu and chin and Chin-Liang Wang, "A parallel decoding scheme for Turbo codes," IEEE International Symposium on Circuits and Systems, Vol.4, 1998

[7] Seokyun Yun, Yeheskel Bar-Ness.Y, "Parallel MAP algorithm for low latency turbo decoding," IEEE Communication Letters, Vol.6, 2002

[8] Yuansheng Song, Gongyuan Liu, Huiyang, "The Implementation of Turbo Decoder on DSP in WCDMA System," IEEE International Conference on Wireless Communicationm Networking and Mobile Computing, Vol.2, 2005.

[9] 3G Generation Partnership Project(3GPP); Technical Specification Group Radio Access Network; **Evolved** Universal Terrestrial Radio Access (E-UTRA); multiplexing and channel coding(Release 9)

[10] Surendra Raju, M., Annavajjala, R., & Chockalingam, A., "BER analysis of QAM on fading channels with transmit diversity," IEEE Transactions on Wireless Communications, Vol.5, 2006