A Tiling-Scheme Viterbi Decoder in Software Defined Radio for GPUs

Chih-Sheng Lin† Wei-Lun Liu‡ Wei-Ting Yeh‡ Li-Wen Chang♯ Wen-Mei W.Hwu♯
Sao-Jie Chen* Pao-Ann Hsiung†
†National Chung Cheng University, Chiayi, Taiwan ‡National Chiao Tung University, Hsinchu, Taiwan
♯University of Illinois at Urbana-Champaign, Illinois, USA *National Taiwan University, Taipei, Taiwan
†{lcs98p,pahsiung}@cs.ccu.edu.tw ‡{aobocean.ee96,yeahweiting.cm96}@nctu.edu.tw
♯{lchang20, w-hwu}@illinois.edu *csj@cc.ee.ntu.edu.tw

***Abstract*—In this paper, we propose a parallel design of Viterbi decoder for Software-Defined Radio (SDR). Our method implements a divide-and-conquer approach by tiling decoding sequences, performing independent speculated Viterbi decoding, and merging partial candidate paths into the final path. For each independent Viterbi decoding, the best path is selected by calculating Hamming distances trellis-by-trellis in parallel.**

Our method shows up to 14.6x speedup on an NVIDIA 8800 GTX over a sequential C implementation on a 2.4GHz Intel Core 2 CPU. Also, compared with existing GPU-based implementation in [3], our method outperforms up to 2.5x.

***Index Terms*—Software-Defined Radio (SDR), Viterbi Decoder, Graphics Processing Units (GPUs), Compute Unified Device Architecture (CUDA)**

I. Introduction

Enormous progress has been made in developing wireless communication standards with the goals of high data rate, long transmission distance, and robustness. Advanced standards such as Global System for Mobile communication (GSM), Worldband Code Division Multiple Access (W-CDMA), Wi-Fi (IEEE802.11), and Worldwide Interoperability for Microwave Access (WiMAX, IEEE 802.16) have been mostly designed in Application-Specific Integrated Circuits (ASICs). However, the capability of this kind of platform is limited to specific communication systems, and cannot conform to multiple standards with reconfigurability for different kinds of demands in the future. With the tendency of current and ongoing standards, each standard requires a reconfigurable, high-performance, and low-cost platform to satisfy the diverse demands from users anywhere and any time.

The term software-defined radio (SDR) was defined by Mitola in 1991 [2]. SDR is designed for reconfigurable wireless standards equipment which conforms to current and future standards. Most of the recent works on SDR are implemented on field programmable gate arrays (FPGAs) or digital signal processors (DSPs). These kinds of platforms lack the features of high computation power or low cost in comparison with graphics processing units (GPUs) for commercial deployment of SDR systems. Furthermore, developing SDR systems in FPGAs and DSPs requires SDR designers to learn how to program particular embedded architectures without a sound and complete development environment.

As a massively parallel computing device, GPU is a candidate SDR platform for signal processing most of the physical layer (PHY) blocks such as convolutional encoder, interleaver and fast Fourier transformer (FFT) in an SDR system. Furthermore, the compute unified device architecture (CUDA) [4] programming model developed by NVIDIA corporation provides SDR developers a programming interface to design systems efficiently. Recently, a multiple-input multiple-output (MIMO) detector [7][8] was implemented on GPUs for recovering received radio frequency (RF) signals with high throughput in a multi-antenna SDR system. Kim et. al. [3] proposed two kinds of GPU-based SDR architectures with high performance and flexibility. They also used GPUs to implement each operational PHY block in a WiMAX transceiver for high performance in comparison with Texas Instruments (TI) DSP.

By analyzing the number of computational operations of each PHY block in current wireless standards, Viterbi decoder is the most computationally intensive component in the receiver of an SDR system. The existing GPU implementation of Viterbi decoder is based on selecting the best path among candidates by calculating their Hamming distances trellis-by-trellis. Although these paths can be searched between two neighboring trellises in parallel, parallelism is still limited by the overheads caused by data dependencies from previous trellises [1]. In this paper, we propose a design based on the concept "divide-and-conquer" with a merging mechanism to eliminate dependencies as much as possible among trellises.

The remaining sections are organized as follows. In Section 2, we give an overview of CUDA and a brief background of Viterbi decoder in SDR systems. In Section 3, we introduce our proposed Viterbi decoder for GPUs. In Section 4, we evaluate our implementation and compare with the existing GPU implementation. Then, we draw a conclusion in Section 5.

II. Background

A. Compute Unified Device Architecture

Compute Unified Device Architecture (CUDA) is a GPU programming model developed by NVIDIA. CUDA provides a

978-1-4244-6252-0/11/$26.00 ©2011 IEEE

programming interface [4] to utilize the highly-parallel nature of GPUs and hide the complexity of controlling GPUs. A programmer specifies a kernel as a series of instructions and a data set, then the kernel is executed by thread blocks, each of which consists of a number of threads with unique IDs. For independent executions among different thread blocks, synchronization is required to avoid the race condition. Note that it is more expensive for synchronizing thread blocks than synchronizing threads in a thread block.

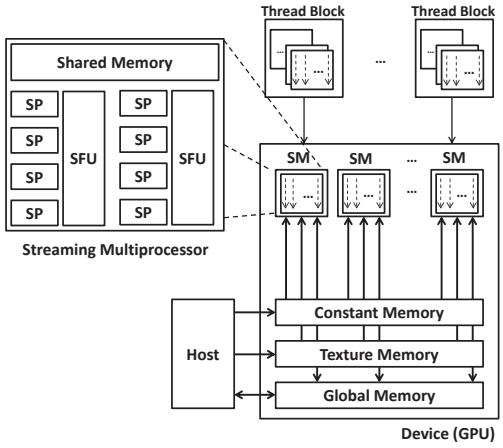

Fig. 1. CUDA Architecture.

At runtime, thread blocks are distributed to streaming multiprocessors (SMs) as shown in Fig. 1. Precisely, a thread block is divided into warps of 32 threads. Each warp is executed by an SM within 4 cycles if the input data is cached for computing. For utilizing an SM efficiently, the threads of each warp should execute the same instructions. Otherwise, it will be executed sequentially with overhead for stalling if threads execute different instructions. This scenario is called warp divergence.

As to the type of memory as listed in Table I, each kind of memory is designed from different read or write speeds, hardware, and programming scopes. Global memory, as a bridge between host and device, is shared among all thread blocks. Constant and Texture memories are read-only memories for faster memory accesses. In a thread block, the most essential component is shared memory. Being different from registers used only by a thread, shared memory allows threads in a block to communicate with each other to reduce the overhead of accessing global memory.

B. Viterbi Decoder

Viterbi decoding algorithm (VDA) [5][6] was proposed as a maximum likelihood and asymptotically optimal decoding algorithm for convolutional codes. When receiving signal sequences from the demodulator in a receiver, Viterbi decoder finds the branch with the best metric of an optimal path between two neighboring trellises by calculating its Hamming distance using the following Equation 1.

TABLE I
COMPARISONS OF MEMORY TYPE

Attributes	**Register**	**Shared Memory**	**Const./Text. Memory**	**Global Memory**
Scope	Thread	Block	Grid	Grid
Hardware	On-chip	On-Chip	DRAM	DRAM
Access Type	Read-Write	Read-Write	Read-Only	Read-Write
Access Latency	Immediate	4 cycles	Immediate	400-600 cycles

$$Y = \arg\max_{X} Pr(X, Z), \tag{1}$$

where X is one of the possible decoding sequences, Y is the most likely decoding sequence, and Z is the received signal sequence.

Being different from the forwarding algorithm, VDA reduces the memory space for only recording the survivor, which is the path with the best metric, instead of all possible paths. Moreover, VDA reduces operations by eliminating the least likely path in each code trellis. If multiple paths have the same metric, VDA chooses one of them randomly as the survivor.

The most essential part of VDA is the selection of the survivor for finding the path with a maximum likelihood estimation (MLE). For the most hardware implementation of VDA, the key component is an add-compare-select (ACS) unit which repeatedly accumulates Hamming distance, compares all viable paths, and selects the survivor for every state of a trellis. Another concern is to efficiently record survivor paths especially in an embedded system with limited memory space. As to software implementations, the critical goal is to achieve high throughput on general-purpose processors. The existing GPU implementation is to select a survivor at every state in the next trellis in parallel. However, the performance is bounded by the data dependencies among trellises.

For the wireless communication standards as mentioned in the previous section, various coding schemes are employed. In this paper, we mainly focus our proposed method on the GSM standard using convolutional encoder with constraint length (K) 5 and coding rate (r) 1/2 as shown in Fig. 2.

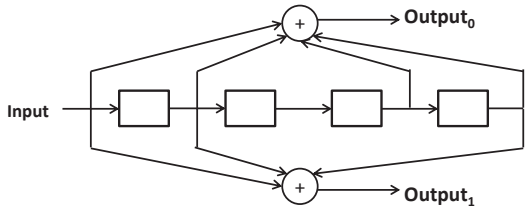

Fig. 2. Convolutional encoder (K = 5, r = 1/2).

III. TILED VITERBI DECODING ALGORITHM

Based on the concept of "divide-and-conquer", we propose a tiled Viterbi decoding algorithm (TVDA) as shown in Figure

3. A forward error correction (FEC) block may be divided into N_C chunks. With N_F FECs, after partitioning, we have N_F-by-N_C independent chunks in total. Each chunk will be traversed independently for storing candidate paths by their Hamming distance with corresponding input sequences, and then candidate paths between neighboring chunks are selected for merging. The decoding sequence will be obtained after merging all N_C chunks of a FEC block in parallel.

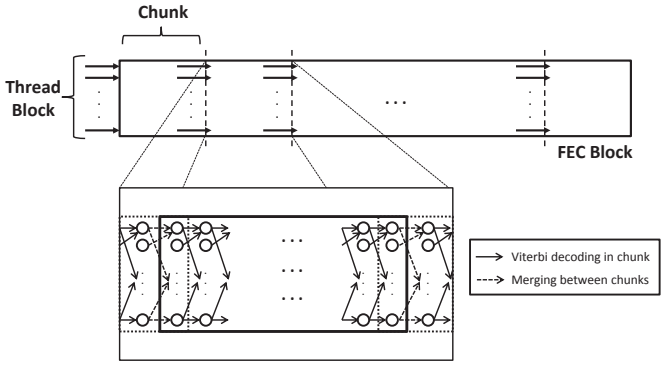

Fig. 3. The concept of TVDA.

TVDA consists of two phases: First, as shown in Algorithm 1, Viterbi decoding will process every chunk for local distance by each thread block in parallel. Second, as shown in Algorithm 2, chunks will be merged for obtaining global distance.

The following four components are for calculating the Hamming distance with corresponding input:

- *State transition matrix* (ST): $st_{p,q}$ captures the connection of state transition between State p in previous trellis and State q in current trellis. $st_{p,q} = 1$ represents the existence of a state transition, $st_{p,q} = 0$ otherwise.
- *Hamming distance matrix* (HD): $hd_{p,q,c}$ represents the Hamming distance between State p in previous trellis and State q in current trellis with a corresponding input c.
- *Local distance*: $d^i_{p,q}$ represents the accumulated Hamming distance from p in Trellis $(i-1)$ and State q in Trellis i.
- *Global distance*: D^i_p represents the accumulated Hamming distance of State p in last trellis of Chunk i.

Every state in the first trellis of the current chunk is a candidate connector from the previous chunk. To acheive the global MLE when merging chunks, TVDA takes every state in the first trellis as a start state except the first chunk of every FEC block. Note that when N_C is equal to 1, our method is reduced into [3]. In that case, the merging phase does not need to be executed.

IV. Experimental Results

We evaluated our decoder by comparing it with a previous non-tiled GPU implementation on a Linux platform with 4GB DDR2 memory and Intel Core 2 Quad Q6600 running at 2.4GHz. In our experiment, we used NVIDIA Geforce 8800 GTX, which has 128 stream processors, running at 1.35 GHz with 768MB of DDR3 memory.

Algorithm 1 Viterbi decoding in chunk

1: Initialize d^0 for every states in first trellis of each chunk
2: **for** $i = 1$ to ($chunkLength$-1) **do**
3: Fetch input c
4: Fetch st and hd from constant memory
5: Fetch $d^{i-1}_{p,q}$ from shared memory
6: $d^i_{q,r} = d^{i-1}_{p,q} + st_{q,r} \times hd_{q,r,c}$
7: **if** $d^i_{q',r} < d^i_{q,r}$ **then**
8: $q' = q$
9: **end if**
10: Store $d^i_{q',r}$ in shared memory and q' in global memory
11: Local Sync
12: **end for**
13: $D^{chunkIndex}_r = d^{chunkLength-1}_{q',r}$

Algorithm 2 Chunk merging

1: **for** $i = 1$ to totalChunk-1 **do**
2: Fetch D^{i-1}_p from shared memory
3: **if** $D^{i-1}_{p'} + D^i_q < D^{i-1}_p + D^i_q$ **then**
4: $D^i_q = D^{i-1}_{p'} + D^i_q$
5: **end if**
6: Local Sync
7: **end for**

Our testing scenario is shown in Fig. 4. The host randomly generates data bits and encodes them into FEC blocks. Then, FEC blocks will pass through the channel with AWGN noise to the device. First, we have to consider BER results of our proposed method because our method eliminates partial dependencies among chunks. In our experiments, the BER results of our method are as well as standard VDA.

Fig. 5 illustrates the speedups of Kim's parallel VDA [3] and our TVDA on a GPU, over a sequential VDA on a CPU. Our method is 1.4x to 14.6x faster than the sequential CPU implementation. Compared with Kim's work, ours shows up to 2.5x speedup on a GPU. Particularly, when N_F is small, our method performs much well over Kim's. That is because our method can extract more parallelism within a FEC. While Kim's method only get parallelism from independent FECs, our method has parallelism from both independent FECs and chunks in FECs.

Note that the number of received FECs which are to be processed in a time depends on the type of wireless application. For one kind of applications such as voice call service, the latency must be short to maintain Quality-of-Service (QoS) for a user. That is, small amount of FECs need to be processed in a time when they are received for reducing the latency. According to this scenario, our method works much well over Kim's. The other kind of applications such as video streaming, large amount of received FECs are processed for high quality. In this case, our method works as well as Kim's.

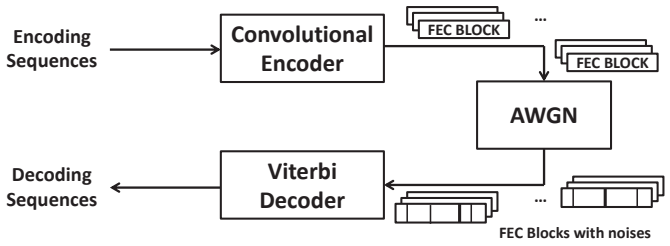

Fig. 4. Testing Scenario.

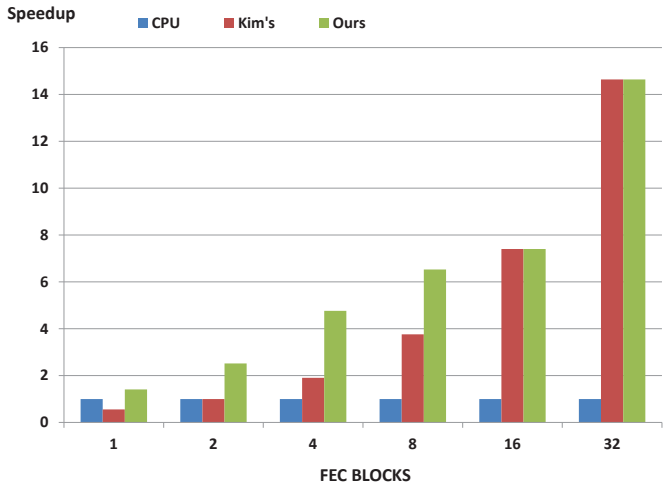

Fig. 5. Performance Comparison.

V. Conclusions

This paper presented a tiling-scheme Viterbi decoding algorithm and its GPU implementation for convolutional encoder with small constraint length. The previous GPU implementation of Viterbi decoder is based on selecting the best path among candidates by calculating their Hamming distances trellis-by-trellis. However, the parallelism is limited by the overhead caused by data dependencies from previous trellises. Our proposed algorithm is designed for eliminating partial dependencies among trellises and recovering results to a MLE. The experimental results showed that our proposed design is 2.5x faster than the previous GPU implementation.

Acknowledgement

The authors would like to thank the members of IMPACT group in UIUC. This work is supported by National Science Council, Taiwan, ROC, under Grant NSC97-2220-E002-032.

References

[1] J. Chase, B. Nelson, J. Bodily, Z. Wei, and D.J. Lee. Real-time optical flow calculations on FPGA and GPU architectures: A comparison study. pages 173–182, 2008.

[2] J. Mitola. *Cognitive Radio Architecture: The Engineering Foundations of Radio XML*. Sepetmber 2006.

[3] J. Kim, S. Hyeon, and S. Choi. Implementation of an SDR system using graphics processing unit. *Communications Magazine, IEEE*, 48(3):156–162, 2010.

[4] NVIDIA Corporation. *NVIDIA CUDA Programming Guide*. 2010.

[5] A. Viterbi. Convolutional codes and their performance in communication systems. *IEEE Transactions on Communication Technology*, 19(5):751–772, 1971.

[6] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 2002.

[7] M. Wu, Y. Sun, and J.R. Cavallaro. Reconfigurable real-time MIMO detector on GPU. pages 690–694, 2010.

[8] 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*, pages 1–14, 2010.