COVER PAGE

SPEED UP OF LINK SIMULATOR USING GPU FOR SDR SYSTEMS

Kiwook Son(HY-SDR Research Center, Hanyang Univ., Seoul, Korea: kiukk@dsplab.hanyang.ac.kr);
Hyunseok Lee (HY-SDR Research Center, Hanyang Univ., Seoul, Korea; yundol@dsplab.hanyang.ac.kr);
June Kim (HY-SDR Research Center, Hanyang Univ., Seoul, Korea; nzneer@dsplab.hanyang.ac.kr);
Seungheon Hyeon (HY-SDR Research Center, Hanyang Univ., Seoul, Korea; sheon@ieee.org); and
Seungwon Choi(HY-SDR Research Center, Hanyang Univ., Seoul, Korea; choi@dsplab.hanyang.ac.kr)

ABSTRACT

Link simulator is a major tool for verifying algorithms and/or inspecting performance of implemented communication systems. As the complexity and throughput of communication system increases, reducing the operation speed is emerged as a key issue of the link simulator. In this paper, we propose a novel architecture for the link simulator which is implemented using a graphic processing unit (GPU). In our experimental tests, the GPU-based link simulator provides 64 times faster operation speed compared with conventional link simulator which uses centre processing unit (CPU) only.

1. INTRODUCTION

In various kinds of link simulators, the function of which is verification of algorithms and/or performance inspection of implemented system, the operation speed is the key issue that determines the quality of the link simulator. The operation speed is even more important in software defined radio (SDR) system because it has to support the multi-mode functionality. In this paper, we introduce a method of accelerating the operation speed of general-purpose link simulators for SDR systems.

In many cases, the operation time has been a main burden of the entire research and development of link simulators. Recognizing the importance of speed-up of operation time, we propose a new scheme of using the GPU for implementing the link simulators which perform with the functionality of SDR. The GPU is optimized for vector processing because it usually consists of multiple processors. Each processor in GPU is composed of a set of threads. For reducing the simulation time, we break the link simulator into a set of waveform components. Each waveform component which can be implemented very efficiently through the vector processing is realized in GPU and the other functions are operated in CPU separately.

To verify the performance of the proposed scheme, we implement a WiBro 1x1 SISO link simulator and measure the simulation time with / without GPU processing. As a result, the operation time with GPU has been found to be 70 times faster compared with the case of CPU only processing without GPU.

The rest of this paper is organized as follows. The overview of the GPU is provided in Section 2. The proposed architecture of the link simulator using GPU is described in Section 3. In Section 4, we implement the link simulator in accordance with the proposed architecture. In Section 5, performance of the implemented link simulator is analyzed in terms of elapsed time followed by conclusion in Section 6.

2. OVERVIEW OF GPU

The GPU is massively parallel processor which can be optimized with a proper vector processing. As shown in Figure 1, the GPU consists of plural thread processors each of which includes 8 threads and shared memory for sharing data among the threads. The thread processors load/store data from/to a global memory [1].

The thread execution manager (TEM) manages thread allocation and execution as follows. The input assembler compiles a function from external host (e.g. CPU) into "kernel" which is an object for GPU. The function has the information about the number of threads for creating the kernel. As shown in Figure 2, using the information of number of threads, the TEM allocates a kernel instruction into multiple threads for parallel processing [2]. Consequently, since GPU is built based on the single instruction multiple data (SIMD) architecture which enables an instruction to process multiple data in parallel, the GPU is optimized for the vector processing [3].

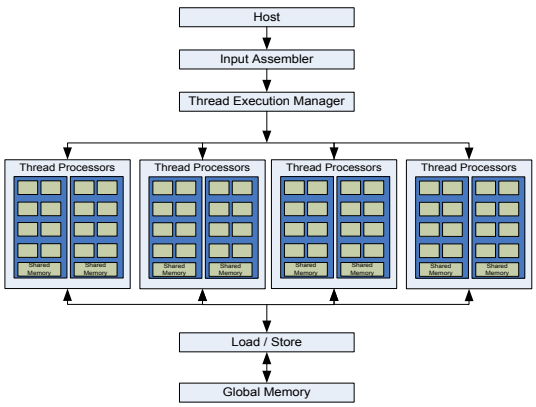

Figure 1 Architecture of GPU

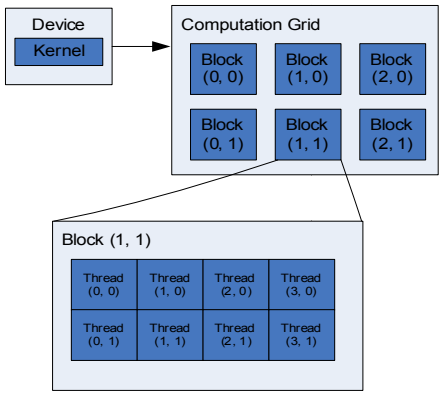

Figure 2 Structure of Kernel

Moreover, by providing software development kit (SDK), the GPU expands their working area into other sites which requires large scale calculation such as fluid dynamics, financial simulation, communication, etc. In other words, the GPU could be utilized not only for 3D-graphic processing but also for various other signal processings.

3. PROPOSED ARCHITECTURE FOR GPU BASED LINK SIMULATOR

The proposed GPU-based link simulator is divided into three parts, CPU application, Framework, and GPU signal processing, as shown in Figure 3. The CPU application is for managing the entire link simulator (such as setting the simulation parameters up and executing the link simulator). Framework includes application programming interfaces for controlling GPU, and the GPU signal processing is for performing most calculations required in the simulator.

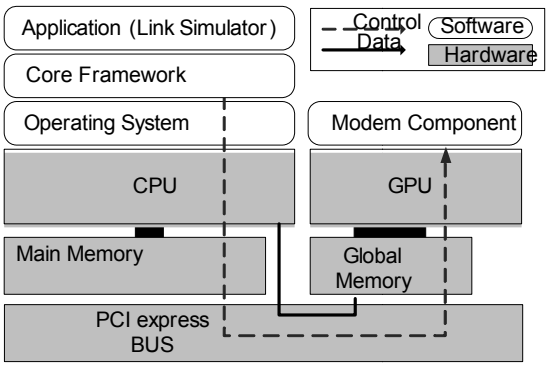

Figure 3 Proposed Architecture for GPU based Link Simulator

Data transfer between the CPU and GPU is performed via Peripheral Component Interconnect (PCI) express bus with direct memory access (DMA) scheme.

4. APPLICATION TO WIBRO LINK SIMULATOR

Figure 4 shows the software structure of GPU-based WiBro link simulator [4]. The link simulator consists of 4 parts, input signal generation, fading channel coefficient generation, WiBro modem, and bit error rates (BER) calculation part. The input signals and fading channel coefficients generation, and calculating BER parts are operated in CPU. The modem component which includes IFFT/FFT, channel estimation, and channel coding/decoding components is programmed into GPU, where each component of GPU is mapped to the kernel.

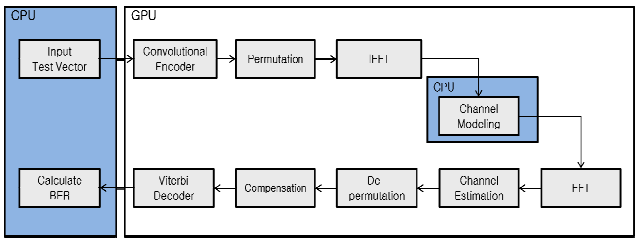

Figure 4 Software Structure of WiBro Link Simulator

The link simulator operates in the following procedures. Firstly, CPU copies the input signals generated by itself to GPU via PCI express, and then GPU encodes them in accordance with WiBro protocols. The encoded data are sent back to CPU for fading channel emulation. The faded signals are fed into FFT component in GPU and then the GPU retrieves the signal and transfers the data into CPU for calculating BER. Detail of each component is depicted as follows.

The transmit part of the link simulator consists of three components, such as convolutional encoder, permutator, and IFFT as shown in Figure 4. Constraint length of the convolutional encoder (CC) in our link simulator has been set to 3 as shown in Figure 5. Since the output symbol of the CC is dependent upon the current and some past inputs, the encoding procedure for the FEC block could not be performed in parallel. However, since there are no dependencies among the FEC blocks, we deploy plural CC's into each thread of the kernel.

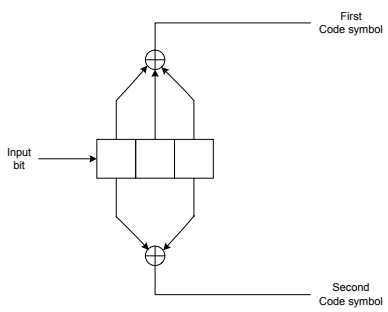

Figure 5 Convolutional Code Encoder (K=3)

In receive part of the link simulator, Viterbi decoder is used for decoding the convolutional encoded data. The Viterbi decoder is the most widely used algorithms to decode the convolutional code. The Viterbi decoder uses a trellis diagram of convolutional code for decoding as shown in Figure 6. In the trellis diagram, the vertical axis represents state while the horizontal axis denotes time.

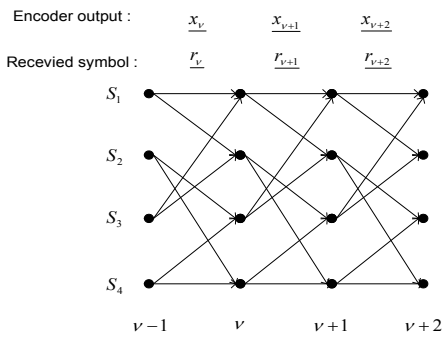

Figure 6 Trellis Diagram of Convolutional Code

Viterbi decoder consists of two parts. One is add-compare-selection (ACS) part and the other is trace-back part. Operating of ACS takes almost all the Viterbi decoder processing time. ACS is for calculating the branch metric and adding the branch metric with the path metric. During the state transition in the trellis diagram, each branch metric is independent of another metric. Therefore we allocate as many threads as the number of states into the decoding block and create as many decoding blocks as the length of convolutional code as shown in Figure 7. Each thread stores the survival path which is calculated from path metric and branch metric [5].

After finding survival path, traceback should be conducted. Unfortunately, it is impossible to parallelize the traceback procedure.

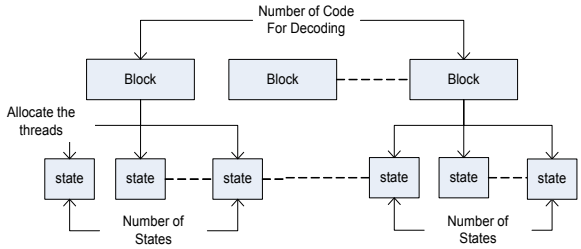

Figure 7 Kernel for Viterbi Decoder

4. PERFORMANCE ANALYSIS

In this section, we present the performance of the GPU link simulator. We use GeForce 9800 GTX+ provided by NVDIA to ascertain performance of the GPU link simulator. Also we implement CPU link simulator, programmed by C language, to compare the performance to each other. The system parameters for the link simulator are as follows;

- Modulation : QPSK
- FFT size : 1024 points
- Coding Rate : 1/2
- Channel : AWGN
- Channel Estimation : Linear Interpolation

Figure 8 and 9 illustrate the elapsed time of the each component in GPU-based and CPU-based link simulator, respectively. As shown in the figures, most of the processing time was consumed by FEC component. Therefore FEC processing time is the key factor for accelerating speed of the link simulator.

The GPU-based link simulator requires 1.994ms for processing a single WiBro frame, while CPU-based simulator takes 128.34ms. In other words, operation time of GPU link simulator is 64 times faster than CPU link simulator.

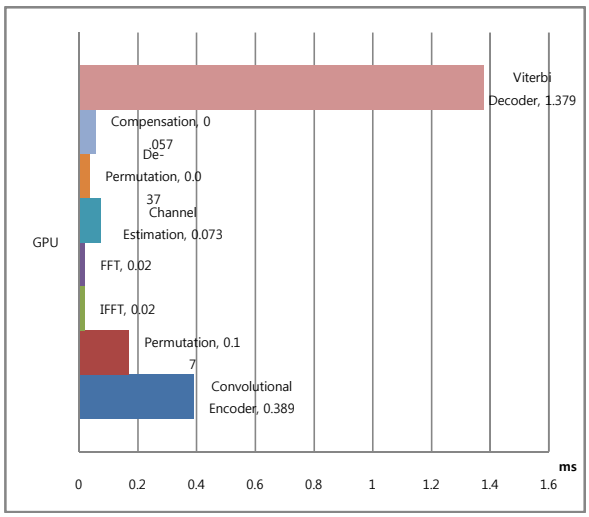

Figure 8 Elapsed Time for each Component in GPU Based Link Simulator

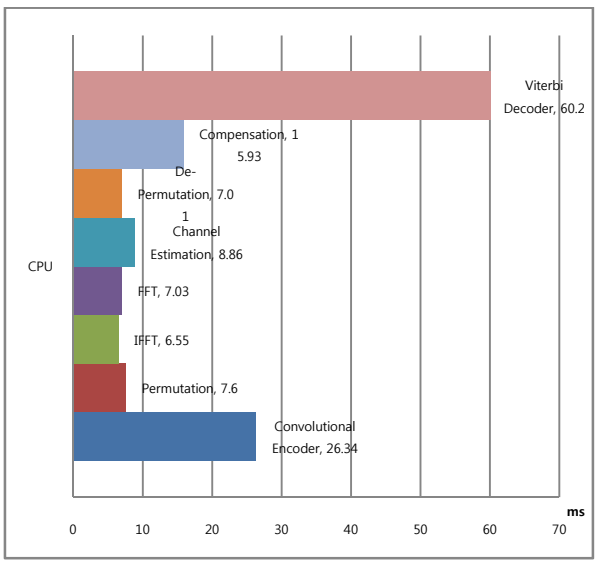

Figure 9 Elapsed Time for each Component in CPU Based Link Simulator

5. CONCLUSION

Most PCs used at home or office have GPU for 3D graphic processing. If we use this powerful computing resource, we can shorten our research period. Especially, in wireless communication engineering, the simulator is an important tool to analyze the performance of the implemented systems. However, most simulators require excessive waiting times for acquiring the results, due to its intensive computation. GPU can be a solution for this problem because of its powerful computation capacity.

In this paper, we proposed a novel architecture for accelerating a link simulator with GPU. In addition, parallelization scheme, which is a key issue in implementing GPU-based link simulator, is presented. Using this characteristic of GPU, we improve the performance of GPU-based link simulator by up to 64 times faster than CPU-based link simulator.

ACKNOWLEDGEMENT

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute for Information Technology Advancement) (IITA-2009-C1090-0902-0003)

7. REFERENCES

[1] Tom R. Halfhill, “Parallel Processing With CUDA”, Micro Processor Report, 2008.01.28

[2] NVIDIA Cooperation “NVIDIA CUDA Compute Unified Device Architecture Programming Guide”

[3] Michael Weiss, Compass, Inc. “Strip Mining on SIMD Architecture”, ACM, pp.234-243, 1991

[4] IEEE, Part 16 : Air Interface for Broadband Wireless Access System, IEEE, 2007, New York

[5] G Fettweis, H Meyr, “Parallel Viterbi Algorithm Implementation: Breaking the ACS Bottleneck” IEEE Transactions on Communications, vol.31, No.8, Aug 1989