RC24644 (C0809-010) September 19, 2008
Electrical Engineering

IBM Research Report

High Performance Viterbi Decoder on Cell/B.E.

Junjie Lai
Department of Electronic Engineering
Tsinghua University
Beijing
P.R. China

Jianwen Chen
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.China

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

High Performance Viterbi Decoder on Cell/B.E.

Junjie Lai
Department of Electronic Engineering
Tsinghua University
Beijing, China
laijj02@mails.tsinghua.edu.cn

Jianwen Chen
IBM China Research Laboratory
IBM
Beijing, China
jianwenc@cn.ibm.com

***Abstract*—Viterbi decoding is widely used in many radio systems. Because of the large computation complexity, it is usually implemented with ASIC chips, FPGA, or optimized hardware accelerators. With the rapid development of the multicore technology, the multicore platforms become a reasonable choice for the software radio (SR) systems. The Cell Broadband Engine processor is a state-of-art multi-core platform designed by Sony, Toshiba, and IBM. In this paper, we present a 64-state soft input viterbi decoder for WiMAX SR Baseband system based on the Cell processor. With one SPE running at 3.2GHz, the viterbi decoder can achieve the throughput up to 30Mb/s to decode the tail-biting convolutional code. The performance demonstrates that the proposed Viterbi decoding implementation is very efficient. Moreover, the viterbi decoder can be easily integrated to the SR system and can provide a highly integrated SR solution. The optimization methodology in this module design can be extended to other modules on CELL platform**

Keywords- Viterbi decoding, WiMAX, Tail-biting, Cell, Multi-core

I. INTRODUCTION

To meet both the performance and the adaptability requirements of the evolving wireless systems, software radio (SR) is introduced. The infrastructures of the traditional wireless baseband systems are also required to adapt to the emerging SR techniques. At the same time, the multicore technology is developing rapidly. The multicore systems can provide high computation capacity and huge throughput, which suits the wireless baseband system well. Obviously, if the whole wireless baseband system can be integrated on one multicore platform with software, the cost for development and management will be reduced greatly. Moreover, it is easy to support multiple wireless standards.

WiMAX, short for Worldwide Interoperability for Microwave Access, is a metropolitan wireless standard ratified by the IEEE, the Institute of Electrical and Electronics Engineers, under the name IEEE-802.16. It can be used in many applications, including the "last mile" broadband connections and offering the mobile client machines with the internet connections, and WiMAX has been approved as a 3G standard today by ITU.

Based on the concept of software radio, we have implemented a WiMAX SR Baseband system on the Cell processor. The block diagram of the BS physical layer baseband transceiver (OFDMA mode) is illustrated in Fig.1.

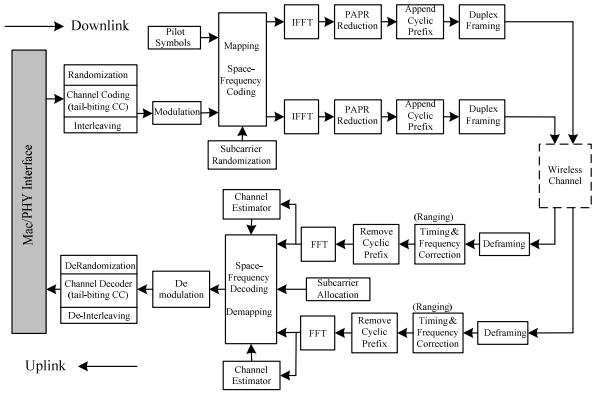

Fig. 1 WiMAX Physical Layer System Structure

The downlink of the system consists of randomization, channel coding, interleaving, modulation, map constellation, space-frequency block coding (SFBC), IFFT, cyclic prefix(CP) insertion and duplex framing; the uplink consists of deframing, timing frequency correction, CP remove, FFT, channel estimation, space-frequency block decoding, demodulation, deinterleaving, channel decoding, derandomization. We use SPU decrementers to evaluate the computation complexity of each module. And the decrementers performance of each module is provided in Table I.

TABLE I
WIMAX MODULES COMPUTATION COMPLEXITY

Component	Decrementers
Channel coding	666
Interleave	1225
IFFT	1551
Channel Estimation	783
SFBC	1484
De-Interleaving	1287
Viterbi Decoding	3204

From Table I, it can be noticed that the Viterbi decoding is the most computation intensive module in the whole SR baseband system. In the conventional radio system, Viterbi decoding is usually implemented with ASIC chips, FPGA or optimized hardware accelerators. However, under the concept of software radio, this module is preferred to be implemented with software and it is the most challenging part in the SR baseband system. On the other hand, only when Viterbi decoding can support the system requirement for throughput, the baseband processing can totally handled with software and a highly integrated SR system solution can be achieved.

In this paper, we will study the Viterbi decoding algorithm for WiMAX Baseband System. The computation complexity of Viterbi decoding is analyzed in detail for an efficient implementation on Cell/B.E. At the same time, we will emphasis on the optimization methodology for Viterbi decoding on CELL/B.E., which can also be used as references for other modules optimization on CELL platform.

Viterbi algorithm is the optimal solution for Convolutional Encoding. The tail-biting convolutional encoding method can eliminate the transfer data rate loss by the extra tail bits introduced by the conventional convolutional code. And tail-biting convolutional encoding, which has the rate of 1/2, a constraint length of 7, is the mandatory channel coding scheme used in WiMAX systems [1]. Two generator polynomials codes are specified as, G1=171 (OCT) and G2=133 (OCT) .

The Cell Broadband Engine processor is the result of collaboration between Sony, Toshiba, and IBM known as STI [2]. As depicted in Fig.2, the Cell BE Processor is a heterogeneous processor with one PowerPC Processor Element (PPE) and eight Synergistic Processor Elements (SPEs). The PPE which contains a 64-bit PowerPC Architecture core runs the operating system and is mainly responsible for controlling the behavior of all the SPEs. The eight SPEs are in-order single-instruction, multiple-data (SIMD) processor elements optimized for compute-intensive work. Each SPE has 256KB local memory for instructions and data, and 128 128-bit register file. Each SPE has two pipelines and can issue and complete up to two instructions each cycle. At 3.2GHz each SPE can give a theoretical 25.6 GFLOPS of single precision performance. All these processor elements are connected by the element interconnect bus (EIB). The EIB transfers data between these processor elements, the main memory and the IO interface. At 3.2GHz it could offer a theoretical peak bandwidth up to 204.8 GB/s.

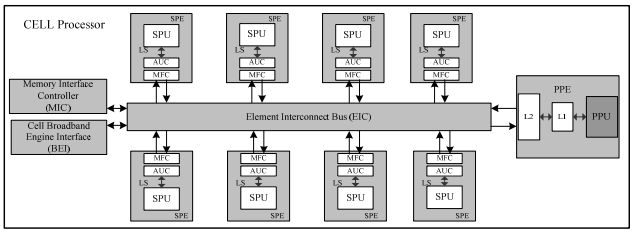

Fig. 2 Block Diagram of Cell Processor

Data transactions between the SPE's local memory and the main memory are via DMA operations. The DMA operation supports aligned transfer size of 1, 2, 4, 8, and 16 bytes and multiple of 16 bytes and can move up to 16KB at a time. With the double-buffer techniques, the DMA transfer latency could be covered by the application execution.

The rest of this paper is organized as follows. Section II will briefly introduce the algorithm for the tail-biting convolutional codes. Section III will describe our considerations and techniques to achieve the peak performance of the viterbi decoding algorithm on the Cell processor. In section IV, the BER and other performance information is provided. Section V presents the conclusions.

II. DECODING ALGORITHM FOR THE TAIL-BITING CONVOLUTIONAL CODES

Tail-biting convolutional encoding can avoid transferring additional data bits, but slightly increased the decoding complexity. There are many ways to decode the tail-biting convolutional codes. Some methods have iterative structures which cannot guarantee a fixed delay. The basic algorithm used here is introduced by Wonjin Sung and In-Kyung Kim, called a fixed delay decoding scheme for tail-biting convolutional codes [3].

The decoding algorithm, which we have changed a little to suite the architecture of the system, is illustrated in Fig.3.

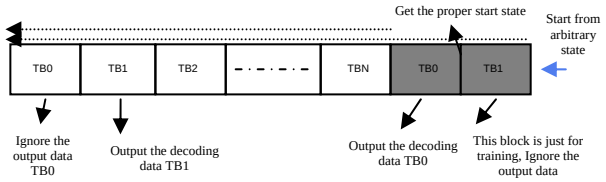

Fig. 3 Tail-biting code Viterbi Decoding

The data block which need to be decoded is separated into smaller blocks, as Block 0, Block 1, ..., Block N.

Firstly, Block 0 and Block 1are attached at the end of Block N and we get a new data vector VN.

Secondly, decode VN using the conventional viterbi algorithm and find the minimum path metric at the end of VN.

Finally, trace from the end of VN back to the original Block 1 by the path with the best path metric. At this procedure, discard the decoded bits of the Block 1 on the tail, and reorganize the bits decoded from the attached Block 0 to the beginning.

To get a negligible degration from maximum likelihood decoding, the size of the block should be greater than 4K, where K is the constraint length [3]. In our system, the value 72 is selected as the constraint length.

III. PARALLELIZE THE DECODING ALGORITHM ON THE CELL PROCESSOR

Viterbi Decoding is the bottleneck of the uplink WiMAX system. To get the best throughput performance, we handily tune highly parallel code.

A. General Considerations of Using One SPE

In order to get the best performance, the application code needs to comply with the architecture and features of the SPE.

Firstly, SPE's natural operand type is 128-bit quadword or vector. A vector is an instruction operand containing a set of data elements packed into a one-dimensional array [4]. SPE's scalar operation performance is very poor because in order to accomplish a scalar operation, the SPE has to pack the scalar data into a vector, and after the operation, unpack the vector data to get the final scalar result. Therefore we need to use vector data type if possible, and sometimes we even need to

change the algorithm or the data memory layout to use vector operations.

Secondly, SPE is an in-order processor element, and SPE issues all instructions in program order. If there is dependency between two adjacent instructions, the later one has to wait to be issued until the former one completes. And this could lead to a huge performance loss. Thus we need to diminish the branch operations and decrease the dependency among the nearby instructions.

Thirdly, each SPE has two dual-issue execution pipelines, referred to as even pipeline and odd pipeline. Each of SPE's six execution units belongs to one of the two pipelines. A doubleword-aligned instruction pair called a fetch group. A fetch group can have one or two valid instructions. The SPU processes fetch groups one at a time [4]. So the SPE can complete up to two instructions per cycle. If the first instruction of a fetch group can be issued while the second one cannot, the first instruction is issued to the proper execution pipeline and the second instruction is held. We need to put instructions issuable to different pipelines together, then after compiling, they could be dual issued.

B. *Computation Data Type Choice*

The SPU hardware supports the following data types:

- Byte—8 bits
- Halfword—16 bits
- Word—32 bits
- Doubleword—64 bits
- Quadword—128 bits

The SPE can finish one vector operation per instruction. Since one vector comprises 4 word type elements, we can finish four word type data operations per instruction. Similarly, we can handle 8 halfword type data operations or 16 byte type data operations per instruction.

The main calculation in the viterbi algorithm is the ACS operation. Comparing the decoder input with the recreated encoder output, we have the number of disagreements, as the branch metric. Then we accumulate the branch metrics as path metrics, and make decisions to select the most likely state transition sequence.

The input of the viterbi decoder is 3 or 4 bits, so every branch metric is 4 or 5 bits long in a 1/2 encoding rate. Thus we could represent the branch metrics and the path metrics with one byte. However, after the accumulation operations, there could be overflows. So we need overflow control scheme in the implementation as described in part F.

The theoretical peak byte operations per second could be 3.2G*16*2=102.4G due to the simdization and the dual pipelines.

C. *Data Memroy Layout and Vectorization*

In get the decoder output, we need to trace the best path back to the beginning. The best path is decided by choosing the final state which has the least path metrics. Then obtain the input sequence by placing a 0 at the decoder output each time we choose an upper transition, and a 1 output each time we choose the lower transition.

So during the ACS operations, the decisions at every transition and the final path metrics need to be stored. With a constraint length of 7, we have 64 different encoding register states. Then the dimension of the decision matrix is 64*L, where L is the length of the length of the ACS input sequence. The dimension of the input is 2*L and the branch table which contains the recreated encoding output is 2*32, due to the 1/2 encoding rate.

The 64 decision in each stage can be stored in 4 vectors. We need 4 vectors to store the branch table and need to splat the input scalar data pair to form 4 vectors to compare with the recreated encoding output vector to get the branch metric. The data of the branch table and the decision matrix should be 128-bit aligned. Otherwise segmentation fault will occur while loading the data in local memory to the registers.

After the data arrangement described, all the ACS operations can be achieved in vector format except the loop control and the overflow control, and high data level parallelism can be obtained.

D. *Butterfly operations*

The butterfly operations, as depicted in Fig.4, can change the relative memory positions of those variables which store the path metrics and the transfer decisions. In order to have the uniform processing pattern, we have to rearrange the data each time that we have processed one decoder input pair.

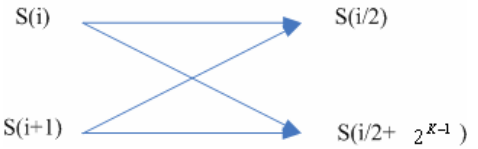

Fig. 4 Butterfly operation in viterbi decoding

The reorganizing operations can easily be implemented by the shuffle operations of the SPE. The shuffle operations select bytes from two source registers and place selected bytes in a target register, and the byte selection operations are controlled by a third source register [5].

Therefore, the reorganizing in the viterbi butterfly operations can be implemented with two types of shuffle operations and total 8 instructions each stage for both the path metrics and the decisions reorganizing.

E. *Loop-unrolling and Overflow Control*

Loop-unrolling is an effective way to decrease the jumping operations and the dependency between instructions and gives the compiler more chance to optimize the code. Because the SPE is in-order processor element, the effect of loop-unrolling is more obvious than other out-of-order processors.

Since the path metric is stored in one byte and each branch metric could be 3 or 4 bits, we need to add overflow control to prevent the path metric values from saturation. The method of the overflow control used here is to check the path metric of state 0, and if the metric exceeds a threshold, we check all the 64 path metrics and find the minimum metric, and then subtract the minimum metric from all the path metrics. Similarly, all the overflow control operations here are implemented with vector instructions except the exceeding judgment.

However, to use the loop-unrolling optimization method can affect the choice of the exceeding threshold. After the experimental test, we finally choose 128 for 4-bit soft input with 2 times of unrolling and 128 for 3-bit soft input with 4 times of unrolling as the thresholds

F. Considerations of Using the Resource of a Full Cell Chip

As described above, the Cell BE processor has 8 SPEs and 1 PPE. Use each SPE to perform part of the process necessary for WiMAX system, with only one SPE core actually running the tail-biting viterbi decoding. This is a type of multi-core operation on Cell platform to implement the WiMAX baseband system. The advantage of this approach is that each SPE has part of the processing codes and there are more memory left on the local store for the data buffer. The disadvantage is that we need to do some more work to achieve a better load balance on 8 SPEs.

Another type of multi-core operation could be that each SPE perform a whole processing task of one frame and PPE is in charge of distributing the data frames to different SPEs. The advantage of this approach is that the load on every SPE is symmetrical and the disadvantage is that code size of each SPE program could be very large.

IV. PERFORMANCE SUMMARY

The BER performance is illustrated in Fig.5 for 3-bit and 4-bit soft input separately. And the peak throughput with 3-bit soft input is 32.5 Mbps and 31Mbps for 4-bit soft input on a single SPE. Additionally, the throughput of the viterbi decoder can increase linearly with the increase of the SPU number.

V. CONCLUSION

This paper presented a 64-state soft input viterbi decoder for WiMAX SR Baseband system on the emerging heterogeneous multicore Cell platform. We focused on the parallelization and optimization for Viterbi decoding on CELL/B.E. The performance shows that with one SPE, the decoding throughput of tail-biting convolutional code can reach up to 30Mb/s. It can be easily integrated to the SR system and provide a highly integrated SR system solution. The design considerations and optimization methodology can also be extended to other SR modules on CELL platform.

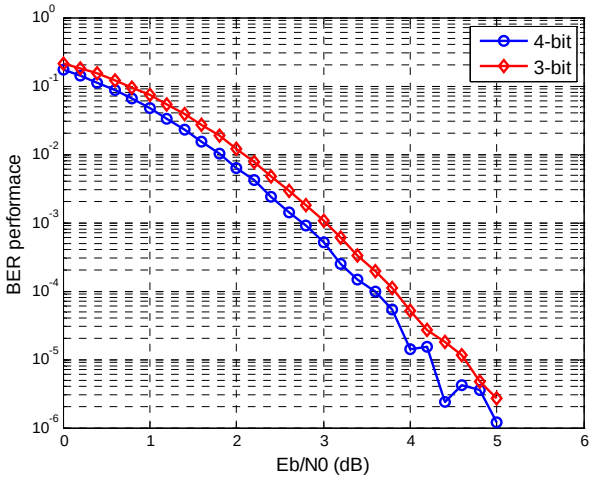

Fig. 5 BER performance for the Viterbi Decoding

VI. FUTURE WORK

For the channel coding of WiMAX physical layer, the convolutional encoding is the mandatory, the Turbo Code encoding is the optional. But in the practical WiMAX CBEs and basestaions, Turbo Code is always supported. So we have realized the initially optimized turbo code on Cell platform. And in the near future, we will try to improve it for an accepted performance. Besides, we also plan to realize high efficient modules for LDPC encoding and decoding on Cell platform. All of these modules will be packaged to an open channel coding library for SR systems on Cell platform.

REFERENCES

[1] IEEE Std 802.16-2004 “Part 16: Air Interface for Fixed Broadband Wireless Access Systems”, June.2004

[2] IBM Cell Broadband Engine resource center website (http://www.ibm.com/developerworks/power/cell/)

[3] Wonjin Sung; In-Kyung Kim, “Performance of a fixed delay decoding scheme for tail biting convolutional codes,” Signals, Systems and Computers, 1996. 1996 Conference Record of the Thirtieth Asilomar Conference on Volume 1, Issue , 3-6 Nov 1996 Page(s):704 - 708 vol.1

[4] IBM, “Cell Broadband Engine Programming Handbook,” Version 1.1, April.2007

[5] IBM, “C/C++ Language Extensions for Cell Broadband Engine Architecture,” Version 2.5, Feb.2008

.