High Performance Channel Decoders on CELL Broadband Engine for WiMAX System

Xiang Chen[1], Ming Zhao[1], Juntao Zhao[1], Jianwen Chen[2], Jing Wang[1]
1- Research Institute of Information Technology, Tsinghua University, Beijing, China.
Email: chenxiang98@gmail.com (Corresponding author)
2- Electrical Engineering Department, UCLA, Los Angeles, USA.
Email: jianwen.chen@gmail.com

Received: June 2010 Revised: October 2010 Accepted: December 2010

ABSTRACT:
Wireless baseband processing, which is characterized by high computational complexity and high data throughput, is regarded as the most challenging issue for software radio (SR) systems, especially for the General Purpose Processor (GPP)-based SR systems. To overcome this implementation difficulty in SR systems, the multicore architecture has been proposed as the GPP-based SR platform, for example, multicore Central Processing Unit (CPU), Graphic Processing Unit (GPU) and Cell processors. In this paper, the Cell processor is considered as the core component in the GPP-based SR platform. The channel decoding modules for convolutional, Turbo and Low-density parity-check (LDPC) codes of WiMAX systems are investigated and efficiently implemented on Cell processor. With a single Synergistic Processor Element (SPE) running at 3.2GHz, the implemented channel decoders can throughput up to 30Mbps, 1.36Mbps and 1.71Mbps for the above three codes, respectively. Moreover, the decoding modules can be easily integrated to the SR system and provide a highly integrated SR solution.

KEYWORDS: Software Radio, Convolutional codes, Turbo code, LDPC codes, WiMAX, Multi-Core.

1. INTRODUCTION[1]

Software Radio (SR) technology brings the flexibility, cost efficiency and lower power to drive communications forward. SR has wide-reaching benefits that are realized by service providers, product developers, and through to end users. However the SR application is always restricted by the performance of the hardware platform. In recent years, the multi-core technology has developed rapidly and is currently the trend of the microprocessor development. Multi-core processor, with high-frequency and low-power consumption, is able to provide a whole wireless system SR solution with high performance and good adaptability [1, 2], which is also called General Purpose Processor (GPP)-based SR systems.

WiMAX, stands 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 4G standard recently by ITU[3].

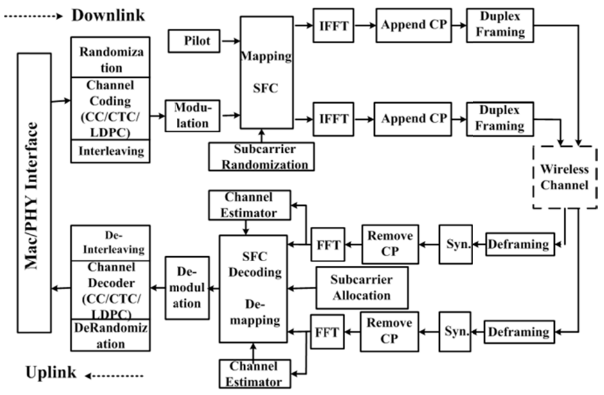

Fig. 1. WiMAX Physical Layer System Structure

In this paper, a basic WiMAX baseband SR system based on Cell Broadband Engine (B.E.) is considered [2][4], which is also based on the multi-core technology. The block diagram of the WiMAX Base Sation (BS) physical layer baseband transceiver (Orthogonal Frequency-Division Multiple Access

[1] This work is partially supported by IBM OCR and SUR programs, National Basic Research Program of China (2007CB31060), National Science and Technology Pillar Program (2008BAH30B09), National Natural Science Foundation of China (60832008), National S&T Major Project (2009ZX03003-010), China 863 Project (2009AA011501) and PCSIRT.

(OFDMA) mode) is illustrated in Fig.1. In Fig.1, 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 Synergistic Processor Element (SPE) decrementers to evaluate the computation complexity of each module. And the decrementers performance of each module is provided in Table 1. But in [2], only the Convolutional Codes (CC) with tail-biting is adopted, which can't meet all the system requirements especially in the multi-path fading channels from the performance point of view.

Table 1. WiMAX modules computational complexity

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

Moreover, from Table 1, it can be noticed that the Viterbi decoding is the most computation intensive module in the whole SR baseband system. Even when some other powerful channel coding schemes are considered (for example, Turbo and Low-density parity-check (LDPC) codes), the computational complexity will be much higher. Actually, in the conventional radio system, channel decoding is usually implemented with ASIC chips, FPGA or optimized hardware accelerators. However, under the concept of GPP-based SR systems, the channel decoding 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 the channel decoding module (CC, Turbo or LDPC codes) can support the system requirements for throughput, the baseband processing can totally handle with software and a highly integrated SR system solution can be achieved.

In this paper, we will study the channel decoding algorithms for CC, Turbo and LDPC codes of WiMAX baseband system. For each type of channel coding schemes (CC, Turbo or LDPC codes), we will firstly analyze the computational complexity of different decoding algorithms in detail for efficient and parallel implementations on Cell B.E. At the same time, we will emphasis on the optimization methodology for different decoding algorithms on Cell B.E., which can also be used as references for other modules optimization on Cell platform.

The rest of this paper is organized as follows. Sections 2 briefly introduce the structure of Cell B.E. processor. From Section 3 to Section 5, the Viterbi decoding for tail-biting CC, the MAX-Log-MAP decoding algorithm for Turbo codes and the offset-Belief Propagation (BP) algorithm for LDPC codes are investigated and implemented on Cell B.E. platform. Then in Section 6, the implementation results on Cell B.E. are given to verify the efficient design on these three types of channel decoders for SR WiMAX baseband systems. Section 7 concludes the paper.

2. STRUCTURE OF CELL B.E. PROCESSOR

In this section, we will firstly review the structure of Cell B.E. processor.

The Cell B.E. processor is the result of collaboration between Sony, Toshiba, and IBM known as STI [4]. As depicted in Fig.2, the Cell B.E. 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.

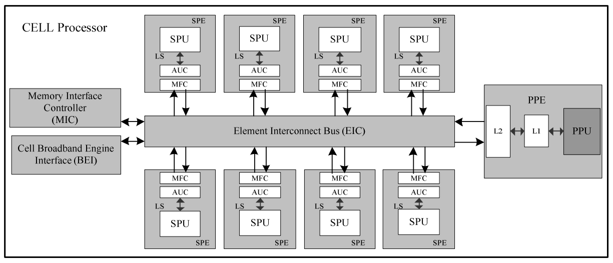
CELL Processor
SPE
SPU
LS
AUC
MFC
Memory Interface Controller (MIC)
Cell Broadband Engine Interface (BEI)
Element Interconnect Bus (EIC)
PPE
L2
L1
PPU

Fig. 2. Block diagram of Cell B.E. 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.

Due to the powerful computational capability and abundant SIMP resources of the Cell B.E. processor, it has been considered as one of the candidates of GPP-based SR platform [2] for WiMAX systems. Therefore, in the following three sections, we will focus on the implementation of channel decoding algorithms for CC, Turbo and LDPC codes over this multi-core platform.

3. VITERBI DECODING ALGORITHM AND IMPLEMENTATION FOR TAIL-BITING CONVOLUTIONAL CODES

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 [3]. Two generator polynomials codes are specified as, G1=171 (OCT) and G2=133 (OCT) .

In this section, we will study the Viterbi decoding algorithm for WiMAX baseband system. Firstly, Subsection 3.1 will briefly introduce the algorithm for the tail-biting convolutional codes. Subsection 3.2 will describe our considerations and techniques to achieve the peak performance of the Viterbi decoding algorithm on the Cell processor.

3.1. Viterbi Decoding Algorithm

Tail-biting convolutional encoding can avoid transferring additional data bits, but it slightly increases 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 [5].

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

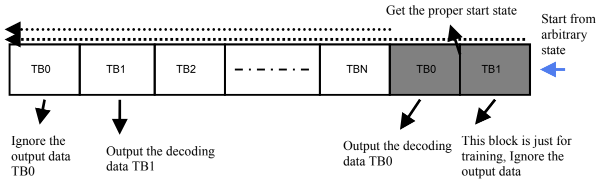
Get the proper start state
Start from arbitrary state
TB0
TB1
TB2
TBN
TB0
TB1
Ignore the output data TB0
Output the decoding data TB1
Output the decoding data TB0
This block is just for training, Ignore the output data

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 1 are 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 degradation from maximum likelihood decoding, the size of the block should be greater than 4K, where K is the constraint length [5]. In our system, the value 72 is selected as the constraint length.

3.2. Parallelize the Viterbi Decoding on Cell

To get the best throughput performance, we manually tune highly parallel code in the following ways.

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 [7]. 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 SPE processes fetch groups one at a time [7]. 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 for CC

The SPE 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 Add-Compare-Select (ACS) operation [9]. 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 SIMD optimization 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. Here, we can also call it the full-state-parallel algorithm for the Viterbi decoding.

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.

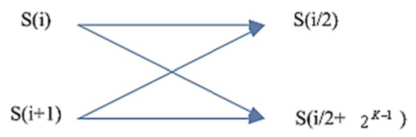

Fig. 4. Butterfly operation in Viterbi decoding

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 [8].

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 mentioned in Section 2, the Cell B.E. 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 [6].

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.

4. MAX-LOG-MAP DECODING ALGORITHM AND IMPLEMENTATION FOR TURBO CODES

In this section, the Turbo decoder on Cell B.E. will be studied. Two parallel decoding methods, referred to as Parallel Block Decoding (PBD) and Parallel State Decoding (PSD), are presented to achieve high throughput and high performance based on the Cell B.E. platform. In addition, the decoder is also optimized based on the programming characteristic of SPE.

4.1. MAX-log-MAP Decoding Algorithm

The iterative Turbo decoder consists of two component Soft-Input Soft-Output (SISO) decoders serially concatenated via an interleaver, identical to the one in the Turbo encoder, as shown in Fig. 5 [9].

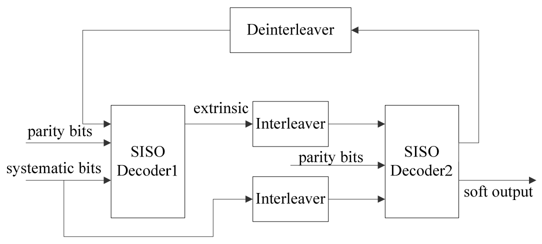

Fig. 5. Structure of the Turbo decoder

When the Maximum A Posteriori (MAP) algorithm is applied to each SISO decoder, the Log-Likelihood Ratio (LLR) for each double-binary pair can be expressed as follows:

$$L(A_k, B_k) = \ln \frac{P(A_k = a, B_k = b \mid y)}{P(A_k = 0, B_k = 0 \mid y)}$$

$$= \ln \frac{\sum_{A_k = a, B_k = b} \alpha_k(s_k) * r_{k+1}(s_k, s_{k+1}) * \beta_{k+1}(s_{k+1})}{\sum_{A_k = 0, B_k = 0} \alpha_k(s_k) * r_{k+1}(s_k, s_{k+1}) * \beta_{k+1}(s_{k+1})}$$

where (a,b) are (0,1), (1,0) or (1,1).

However, the MAP decoding algorithm requires large memory and a large number of operations involving exponentiations and multiplications, which is likely to be considered too difficult for implementation, especially in the SR system on Cell B.E. Thereby here we choose the MAX-log-MAP algorithm to replace the MAP algorithm, which has acceptable performance with much lower computational complexity and memory consumption [9].

In the MAX-log-MAP algorithm, the output of each SISO decoder, representing the extrinsic LLR, is expressed as follows:

$$\lambda(A_k, B_k) = \max_{(s_k, s_{k+1}, z)} \{\alpha_k(s_k) + \gamma_{k+1}(s_k, s_{k+1}) + \beta_{k+1}(s_{k+1})\} - \max_{(s_k, s_{k+1}, 00)} \{\alpha_k(s_k) + \gamma_{k+1}(s_k, s_{k+1}) + \beta_{k+1}(s_{k+1})\} \quad (1)$$

where z∈ ϕ={01,10,11}.

The extrinsic LLR $\lambda(A_k, B_k)$ of SISO decoder is interleaved or de-interleaved and then fed to the next SISO decoder as the priori information $L_{e,IN}^{(z)}$.

The computation of the LLR can be broken into calculation of three metrics: the forward state metric $\alpha_k(s_k)$, the branch metric $\gamma_{k+1}(s_k, s_{k+1})$, and the backward state metric $\beta_{k+1}(s_{k+1})$. Denote s_k and s_{k+1} as the start and the end states, respectively. Then $\alpha_k(s_k)$, $\beta_k(s_k)$ and $\gamma_k(s_k, s_{k+1})$ can be defined as follows:

$$\alpha_k(s_k) = \max_{s_{k-1} \in s_1} \{\alpha_{k-1}(s_{k-1}) + \gamma(s_{k-1}, s_k)\} , \quad (2)$$

$$\beta_k(s_k) = \max_{s_{k+1} \in s_2} \{\beta_{k+1}(s_{k+1}) + \gamma(s_k, s_{k+1})\} , \quad (3)$$

$$\gamma_k(s_k, s_{k+1}) = \ln[P(y_k \mid x_k) * P(A_k, B_k = z)]$$

$$= \frac{L_c}{2}(x_k^{s_1} y_k^{s_1} + x_k^{s_2} y_k^{s_2} + x_k^{p_1} y_k^{p_1} + x_k^{p_2} y_k^{p_2}) + L_{e,IN}^{(z)} , \quad (4)$$

where:

- s_1 is the set of states at time k-1 connected to the state s_k. s_2 is the set of states at time k+1 connected to the state s_k.
- $z \in \phi = \{00, 01, 10, 11\}$.
- (A_k, B_k) is the input symbol consisting of two bits. $P(A_k, B_k)$ is a priori probability of (A_k, B_k).
- x_k and y_k are the transmitted and received codewords respectively associated with (A_k, B_k).
- Superscripts p and s respectively denote the parity bits and systematic bits.
- $L_{e,IN}^{(z)}$ is the priori information obtained from the other SISO decoder.

The code is assumed to be modulated by BPSK and transmitted through an AWGN channel with noise variance σ^2. In this case, the Turbo decoding based on the Max-log-MAP algorithm is independent of SNR, therefore $L_c = 2/\sigma^2$ can usually be set to a constant value.

4.2. Turbo Decoder Implementation on Cell

In this part, we will describe the implementation methods of MAX-log-MAP decoding algorithms on a single SPE. To make good use of this SIMD feature, we present two parallel decoding methods on the implementation of the SISO decoder, which are Parallel Block Decoding (PBD) and Parallel State Decoding (PSD), respectively.

A. PBD Implementation

Firstly, we will introduce the PBD implementation method. In PBD method, the frame size is assumed to be M bits, which can be divided into N sub-blocks with equal length. Each sub-block of the frame can be decoded in parallel structure independently [10].

As for the implementation, without loss of generality, the soft-input data is quantized by 8 bits and occupies two bytes. So one 128-bits wide vector can contain eight soft-input data at maximum. The above data mapping method of PBD algorithm is shown in Fig. 6, where data[i] denotes the soft-input of the decoder, i=0,1,…,2*M/N. Thus, data[i] of each sub-block is read into one vector and dealt with in parallel. Simultaneously, values of α and β for each sub-block are calculated according to (2), (3) and (4) in Section Ⅱ. At last, the LLR is obtained by (1) in parallel and independently.

Since the calculations of α and β metrics may be started somewhere in the middle of the frame, they must be initialized. Fig. 7 shows the initialization value passing scheme for PBD algorithm. For simplicity, we only demonstrate two iterations in Fig.7. In practice, the number of iterations is chosen by the trade-off between the Bit Error Ratio (BER) performance and decoding throughput.

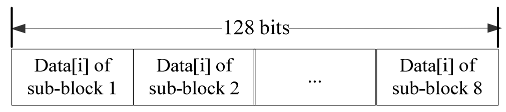

Fig. 6. Data mapping for the PBD algorithm

Assuming k bits couples are contained in each sub-block. As shown in Fig.7, during the parallel decoding, α_k of the sub-block n (n=1, 2,…, N-1) is saved as α_0 of the sub-block n+1 for the next iteration. For the last sub-block N, α_k is saved as α_0 of the sub-block 1 for the next iteration. Similarly, β_0 of the sub-block n (n=2, 3,…, N) is saved as β_k of the sub-block n-1 for the next iteration, and for the sub-block 1, β_0 is saved as β_k of the sub-block N for the next iteration.

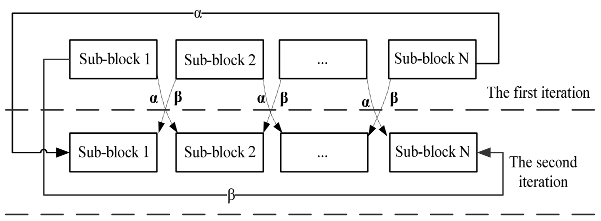

Fig. 7. Metric initialization value passing scheme for PBD algorithm

B. PSD Implementation

Except the parallel decoding structure of PBD algorithm according to the sub-blocks in one frame, another parallel structure based on the decoding states, referred to as PSD algorithm, is presented in this subsection. Assuming the state number of the Turbo encoder is D, in the PSD algorithm, all states of α and β are calculated in parallel. Since the CTC encoder used in WiMAX system has three registers [3], there are eight states in total, i.e., D=8. The state transition diagram of this CTC is given in Fig.8 [9].

In Fig.8, we can see that, for each specific time k, both of α and β have eight states. With each α or β corresponding to each state represented by 16 bits, one 128-bits-wide vector can contain all the values of α or β corresponding to all the eight states. The data mapping scheme for α in the PSD algorithm be shown in the following Fig. 9. The mapping scheme for β is also similar to that for α.

Then the values of α and β for eight states are calculated in parallel structure simultaneously, according to (2), (3) and (4) in Section Ⅱ. In order to calculate the α and β in parallel, the SPE shuttle instruction should be used frequently, to adjust the positions of the eight elements in one vector in each iteration.

Since the local store, i.e., local memory of one single SPE is limited to 256K Bytes (KB), the extrinsic LLR can be calculated during the calculation of α to save the memory usage. Consequently, with the frame length 2*N, the memory usage for metric α is reduced from (N+1)*8*2 bytes to 8*2 bytes.

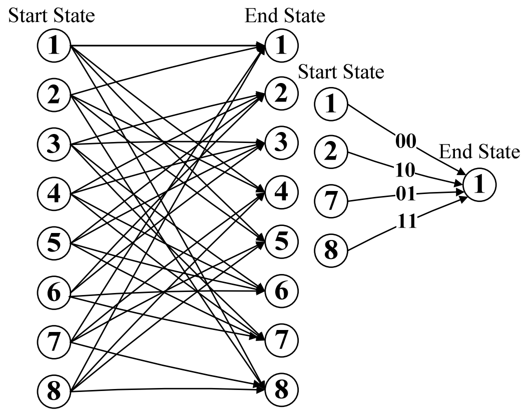

Fig. 8. State transition diagram of CTC in WiMAX system

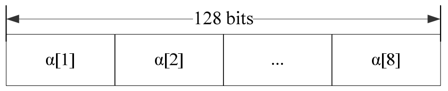

Fig. 9. Data mapping for the PSD algorithm

C. Turbo Decoder Optimization on Cell B.E.

To achieve the best throughput performance, the application programming codes are also optimized as Subsection 3.2. Moreover, for Turbo decoder on Cell B.E, last but not least, we use more memory to help to speed up the calculations. The local store of the SPE is only 256KB, which contains the program, stack, local data structures, and the DMA buffers. During our implementation in one single SPE, the memory occupation of the program and global data for the Turbo decoder is about 135KB, then the remainder 121KB can be used for memory spending (stack allocation etc.) during the program running. It is confirmed that, when the frame size is equal or less than 4800 bits, the maximum SPE local sore consumption during the program execution will not exceed 256KB.

As for the use of a full Cell chip for Turbo decoder, we also consider the methodology as mentioned in Subsection 3.2.F.

5. OFFSET BP-BASED DECODING ALGORITHM AND IMPLEMENTATION FOR LDPC CODES

In this section, the LDPC decoder on Cell will be studied, which is also optimized based on the programming characteristic of SPE.

5.1. Offset BP-based Decoding Algorithm

In this part, the Offset BP-based [11] decoding Algorithm is applied to the LDPC decoding. In the following depiction, H represents the check matrix. n represents the length of codeword. $C = (c_1, c_2, ..., c_n)$ represents the codeword, $\widehat{C} = (\widehat{c}_1, \widehat{c}_2, ..., \widehat{c}_n)$ represents the codeword after decoding. $L(c_i)$ represents the soft initial information for variable node i. $L(r_{ji})$ represents the information transmitted from check node j to variable node i. $L(q_{ij})$ represents the information t transmitted from variable node i to check node j. $L(q_i)$ represents the collected information by the variable node i. a_{ij} and $b_{i'j}$ are expressed as follows:

$$\begin{aligned} \alpha_{ij} &= sign(L(q_{ij})) \\ \beta_{i'j} &= abs(L(q_{ij})) \end{aligned} \tag{5}$$

The steps of Offset BP-based decoding algorithm are listed as follows:

1. Set the original information

$$L(q_{ij}) = L(c_i) \tag{6}$$

Set the Iteration variable Iter=0

2. Update Check Node information

$$L(r_{ji}) = \left(\prod_{i' \in R_j \backslash i} \alpha_{i'j} \right) \max \left[\min_{i' \in R_j \backslash i} \beta_{i'j} - offset, 0 \right] \tag{7}$$

3. Update Variable Node information

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i}) \tag{8}$$

4. Iter=Iter+1. If variable Iter is bigger than the max Iterations, go to Step 6. Otherwise go to Step 2.

5. Output $\widehat{C} = (\widehat{c}_1, \widehat{c}_2, ..., \widehat{c}_n)$

5.2. Parallelize the Offset BP-based LDPC Decoding on Cell

In this section, we will describe considerations and techniques of the implementation methods of Offset BP-based decoding algorithm on Cell B.E.

A. Computation Data Type Choice for LDPC

Since the smaller bit width of data type is, the faster decoding speed is along with more information loss, it's important to choose the quantization scheme to maximize the decoding speed and guarantee the BER performance. We contrast the BER performance with LDPC codes in WiMAX systems between different quantization schemes: char (8 bits), short (16bits) and float (32bits). The answer is illustrated in Fig.10.

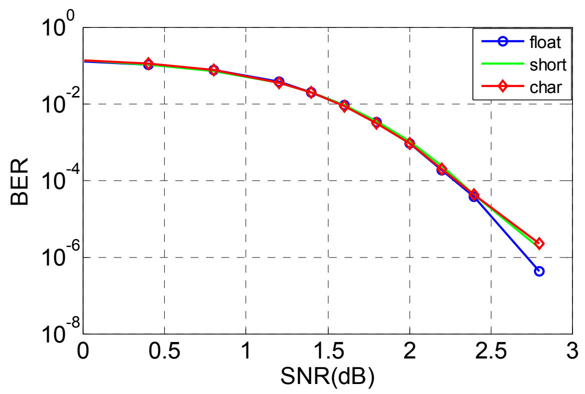

Fig. 10. BER performance comparison for different quantization schemes

The frame length of the LDPC code is 960 bits, the coding rate is ½, and the iteration number of decoding is 15.The LDPC decoding is implemented on the CPU using the Offset BP-based decoding algorithm. The code is assumed to be modulated by BPSK and transmitted through an AWGN channel.

According to Fig.10, we can see that the BER performance between char type and float type is quite small. So we decide to choose the char type.

B. Parallel Mode Considerations for LDPC

The main operations of LDPC decoding are Updating Check Node information and Updating Variable Node information which are equivalent to the row and column operations of the check matrix H [11]. It's naturally to choose the parallel mode in which updates 16 rows or columns at a time. However this method is proved to be failed in implementation on Cell. As the check matrix is a sparse matrix, we only store the non-zero elements and the elements are stored in the vector mode which means every vector stores 16 elements and every vector operates 16 elements. If we update 16 rows or columns at a time, for each vector operation there will be 16 scalar operations to find the 16 elements in different vectors which will be a disaster for the decoding speed.

So different parallel strategies used in Viterbi and Turbo decoders, we consider another parallel mode, in which 16 LDPC blocks are decoded at the same time as in Fig.11. In this parallel mode, the operations will all be vector operations and the computation ability of SPE can be sufficiently used.

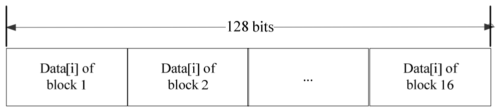

Fig.11. LDPC decoder parallel mode

C. Memory Options and Other Issues for LDPC

As we have chosen the above parallel mode and computation data type, we can compute the maximum used memory in SPE. We find that if the code length n is not bigger than 1248, the memory usage including the program, stack, local data structures, and the DMA buffers is less than 256KB, which is enough in one SPE.

When using each SPE to perform part of the process necessary for WiMAX system, with only one SPE core actually running the LDPC decoding. We consider a parallel mode similarly with what we use in one SPE. For now, every SPE decodes 16 LDPC blocks at a time. When we use the full 8 SPEs, we can decode 128 LDPC blocks at a time.

The application programming codes can also be further optimized as Subsection 3.2.

6. BER AND THROUGH PERFORMANCE

In this section, we will present some results, including BER and throughput performance, to justify the efficacy of the proposed parallel decoding algorithms for different channel coding schemes of WiMAX systems on Cell B.E.

6.1. Performance Results of Viterbi Decoding

The BER performance of CC is illustrated in Fig.12 for 3-bit and 4-bit soft input, respectively. 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 SPE number.

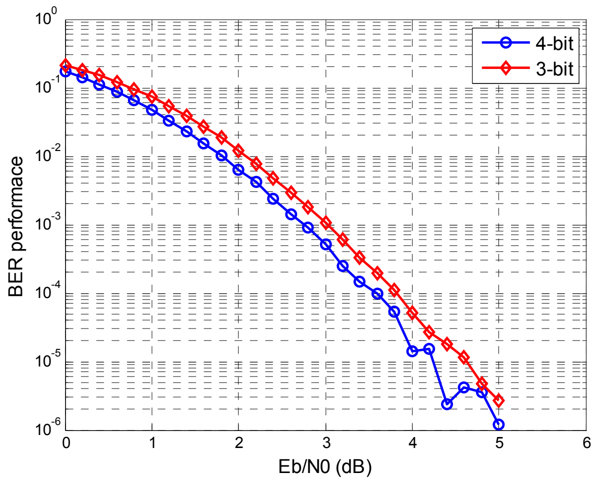

Fig. 12. BER performance for the Viterbi Decoding

6.2. Performance Results of Turbo Decoding

A. Throughput and Latency Results

Fig. 13 shows the throughput of the two parallel Turbo decoding methods for different frame length. For the tradeoff between BER performance and throughput, when the frame length is equal or shorter than 192 bits, the iteration number is set to be 6; otherwise, the iteration number is set to be 5.

As shown in Fig. 13, we can find that when the frame length is equal or longer than 192 bits, the PBD algorithm has obviously higher throughput than the PSD algorithm. However, as the frame length decreases, the throughput gap between these two algorithms is getting smaller. That is because the PBD algorithm is constrained by the BER performance. When the frame length gets shorter, to keep the acceptable BER performance, the number of sub-blocks has to be smaller, which deduces the efficiency of the SIMD operation and thus deduces the throughput.

Therefore, when the frame length is less than 192 bits, the PSD algorithm will be preferred; otherwise, the PBD is preferred [13].

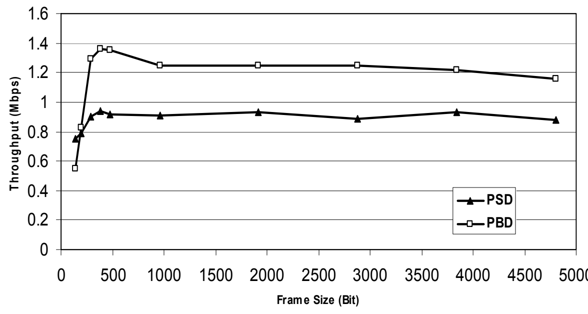

Fig. 13. Throughput comparison of the two parallel methods

Without loss of generality, the decoding latency of frame length 480 bits is given. The iteration number is 5. For PSD, the decoding latency is 0.52 ms. For PBD, the decoding latency is 0.36 ms.

B. BER Proformance of Turbo Decoding

In this part, some BER and Frame Error Ratio (FER) results of the PBD algorithm will be shown. The results of the PSD algorithm are omitted due to the same BER performance as the non-block algorithm.

Firstly, Fig. 14 and Fig. 15 show the BER and FER performance of different frame and block length for the PBD algorithm and non-block algorithm respectively.

From Fig. 14 and Fig. 15, we can find that for the same frame length, as the block length becomes shorter, the BER and FER performance gets worse. The PBD algorithm offers the similar BER and FER performance with the non-block algorithm when the frame length is equal or longer than 192 bits.

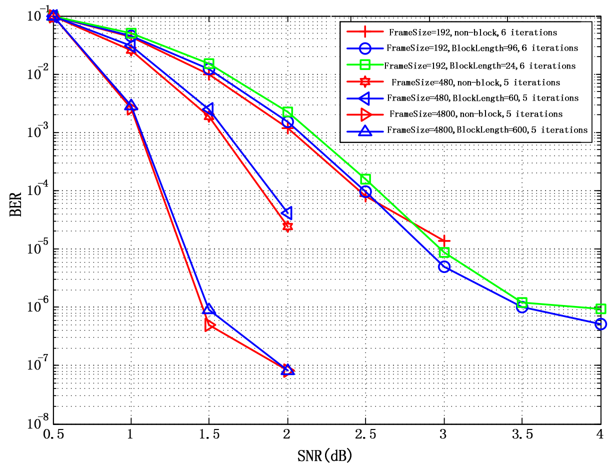

Fig. 14. BER performance for the PBD algorithm

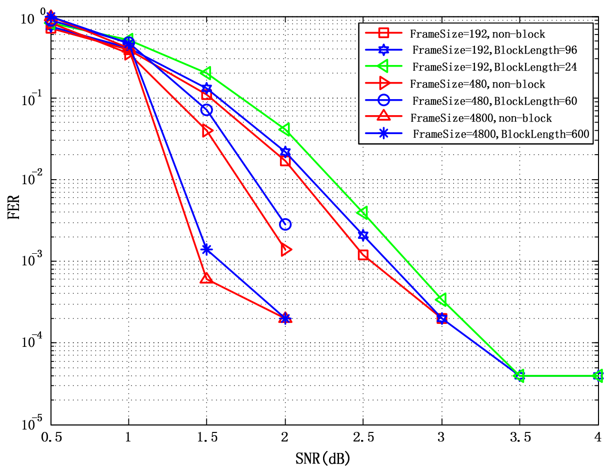

Fig. 15. FER performance for the PBD algorithm

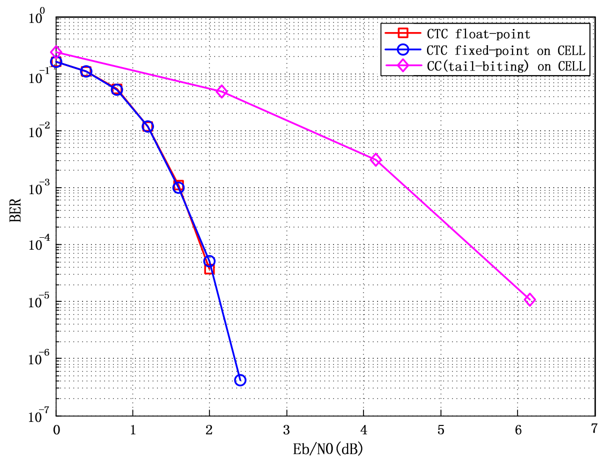

Fig. 16. WiMAX BER performance on Cell B.E.

Furthermore, the PBD Turbo decoder with BPSK modulation is concatenated into the WiMAX SR system as shown in Fig.1. Fig. 16 shows the system BER performance comparisons in AWGN channel between the CC and CTC. The CTC is decoded by the

PBD algorithm. The frame length of the CTC code is 480 bits, the coding rate is ½, and the iteration number of decoding is 5. From Fig.16, we can see that the coding gain of the CTC compared to the CC is about 4dB at the BER 10^{-5}.

6.3. Performance Results of LDPC Decoding

The BER performance of LDPC codes on Cell B.E. is illustrated in Fig.17 for float and char input, respectively. The frame length of the LDPC code is 960 bits, the coding rate is ½, and the iteration number of decoding is 15. The LDPC decoding is based on the Offset BP-based algorithm. BPSK modulation and AWGN channel are assumed.

The peak throughput with float input is 0.53Mbps and 1.71Mbps for char input on a single SPE in contrast with the 0.08Mbps' peak throughput on CPU. The CPU is Intel(R) Pentium(R) Dual E2180 with two cores both working at 2.0GHZ. When we use a whole Cell chip, the throughput for char input will be greater than 13Mbps. If we use the decoding strategy described above, the minimum decoding time delay is 0.59us, while the maximum decoding time delay depends on the queuing model of arriving service.

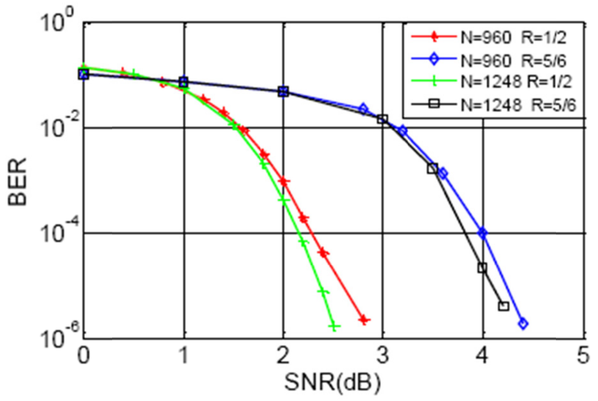

Fig. 17. BER performance of LDPC codes on Cell

7. CONCLUSIONGS

In this paper, the channel decoding algorithms are investigated on multi-core Cell B.E. platform for WiMAX SR systems. Three types of channel coding schemes, including CC, Turbo and LDPC codes, are all implemented on Cell B.E. processor according to their different characters.

For the Viterbi decoding of CC, based on the full-state-parallel algorithm, the decoding throughput can reach up to 30Mb/s on single SPE running at 3.2GHz. For the MAX-log-MAP decoding of Turbo codes, two parallel decoding algorithms, so-called the PSD and PBD, are presented. The testing results show that, the throughput of the Turbo decoder can reach up to 1.36Mbps. For the offset BP-based algorithm of LDPC coes, we decode 16 LDPC blocks simultaneity on a single SPE and the throughput can achieve 1.71Mbps.

Moreover, the channel decoders, which are all optimized in parallel on Cell B.E. platform, can be easily integrated to the whole SR WiMAX system and can provide a highly integrated SR solution. The related optimization methodology in this module design can be extended to other modules on Cell platform.

All of these channel decoder modules have be packaged to an open channel coding library for SR systems on Cell platform [12], which provides a novel idea to implement a whole GPP-based SR wireless baseband system over multi-core platform.

8. ACKNOWLEDGMENT

The authors gratefully acknowledge Yonghua Lin, Huili Guo, Junjie Lai for their contributions on this paper. This work was partly presented at the 2009 International Conference on Wireless Communications and Signal Processing, Nanjing, China, November 13-15, 2009, and partly presented at the 1st International Workshop on Software Radio Technology, Beijing, China, October 16-17, 2008.

REFERENCES

[1] Kun Tan, Jiansong Zhang, Ji Fang, **"Sora: High Performance Software Radio Using General Purpose Multi-core Processors,"** in *6th USENIX Symposium on Networked Systems Design & Implementation 2009, USENIX*, 2009.

[2] Jianwen Chen, Qing Wang, Zhenbo Zhu, Yonghua Lin, **"An Efficient Software Radio Framework for WiMAX Physical Layer on Cell Multicore Platform,"** in *ICC2009*, in Dresden, Germany.

[3] IEEE Std 802.16e-2005, **"Part 16: Air Interface for Fixed, Mobile Broadband Wireless Access Systems Amendment2:Physical, Medium Access Control Layer for Combined Fixed, and Mobile Operation in Licensed Bands,"** Feb., 2006.

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

[5] Wonjin Sung, In-Kyung Kim, **"Performance of a fixed delay decoding scheme for tail biting convolutional codes,"** *30th Asilomar Conference on Signals, Systems and Computers*, Vol. 1, Issue 3-6, Nov. 1996, pp.704 – 708.

[6] Junjie Lai, Jianwen Chen, **"High Performance Viterbi Decoder on Cell BE,"** *Proc. the 1st International Workshop on Software Radio Technology (SRT2008)*, October 16-17, 2008.

[7] IBM, Cell Broadband Engine Programming Handbook, Version 1.1, April 24, 2007.

[8] IBM, **C/C++ Language Extensions for Cell Broadband Engine Architecture,** Version 2.5, Feb.2008.

[9] Shu Lin, Daniel J. Costello, **Error Control Coding (2nd Edition)**, Prentice Hall PTR, New York, 2004.

[10] Jonathan Roth and Naraig Manjikian, Subramania Sudharsanan, **"Performance Optimization and Parallelization of Turbo Decoding for Software-**

defined Radio," *Electrical and Computer Engineering, 2009 CCECE '09 Canadian Conference*, pp. 804 – 809, May 3-6, 2009.

[11] Yuan Dongfeng, Zhang Haigang, **"The theory and applications of LDCP code"** (in Chinese), *the People's Posts and Telecommunications Press*, pp77-84,April 2008

[12] Open source software: http://sourceforge.net/projects/openwireless/

[13] Huili Guo, Juntao Zhao, Jianwen Chen, Xiang Chen, Jing Wang, **"High Performance Turbo Decoder on CELL BE for WiMAX System"**, *the 2009 International Conference on Wireless Communications and Signal Processing, Nanjing, China*, November 13-15, 2009.