Fast Viterbi Decoder Algorithms for Multi-Core System

ZILONG JU

Master's Degree Project
Stockholm, Sweden

XR-EE-SB 2012:014
Fast Viterbi Decoder Algorithms for Multi-Core System

Abstract

In this thesis, fast Viterbi Decoder algorithms for a multi-core system are studied. New parallel Viterbi algorithms for decoding convolutional codes are proposed based on tail biting trellises. The performances of the new algorithms are first evaluated by MATLAB and then Eagle (E-UTRA algorithms for LTE) link level simulations where the optimal parameter settings are obtained based on various simulations. One of the algorithms is proposed for implementation in the product due to its good BLER performance and low implementation complexity.

The new parallel algorithm is then implemented on target DSPs for Ericsson internal multi-core system to decode the PUSCH (Physical Uplink Shared Channel) CQI (Channel Quality Indicator) in LTE (Long Term Evolution). And the performance of the new algorithm in the real multi-core system is compared against the current implementation regarding both cycle and memory consumption. As a fast decoder, the proposed parallel Viterbi decoder is computationally efficient which reduces significantly the decoding latency and solves memory limitation problems on DSP.

Keywords

Parallel algorithm, Viterbi algorithm, Tail biting convolutional code, Multi-core system
Acknowledgements

The work presented in this thesis was carried out at Ericsson LTE baseband design department, Stockholm.

In the first place, I would like to express my sincerest gratitude to my supervisor Magnus Björklund at Ericsson for his valuable advice and consistent encouragement. It has been a pleasure working with him and I deeply appreciate his valuable feedback and insightful comments.

I gratefully thank my examiner and supervisor Joakim Jaldén at KTH for his outstanding administrative and academic support, which improved the quality of the work.

Many thanks go in to Peter Björk, my manager at Ericsson LTE Design, for offering me the opportunity to carry out my Master thesis at Ericsson and continuously supporting me during my thesis and job searching. I would also like to extend my thanks to the colleagues at Ericsson LTE and, in particular, Alan Said, Simon Gaude, Oskar Mauritz, and Fredrik Huss for their valuable discussions and cheerful support.

It is a pleasure to express thanks to my friends in Sweden for helping me and enjoying time with me.

Finally, I would like to express my special gratitude to my family for their love and support. Words cannot express my thanks to you.
Contents:

1 Introduction ................................................................................................................................. 7
   1.1 Background ............................................................................................................................. 7
   1.2 Previous Work ....................................................................................................................... 7
   1.3 Objectives ............................................................................................................................. 8
   1.4 Research Approach ............................................................................................................... 8
   1.5 Thesis Outline ....................................................................................................................... 9

2 Viterbi algorithms ..................................................................................................................... 10
   2.1 Introduction ............................................................................................................................ 10
   2.2 Convolutional Codes ............................................................................................................ 10
      2.2.1 Convolutional encoder ................................................................................................. 11
      2.2.2 Trellis diagram ............................................................................................................. 12
   2.3 Viterbi Algorithm .................................................................................................................. 12
      2.3.1 Implementation ............................................................................................................. 13
   2.4 Tail-biting Convolutional Code ............................................................................................ 14
   2.5 Wrap-Around Viterbi Algorithm ........................................................................................ 15

3 Parallel Viterbi Algorithms ..................................................................................................... 16
   3.1 Overview of Parallel Processing .......................................................................................... 16
   3.2 Block-wise Viterbi Algorithm ............................................................................................. 16
   3.3 Parallel Wrap-around Viterbi Algorithm .............................................................................. 18
      3.3.1 Subblock structure 1 ..................................................................................................... 18
      3.3.2 Subblock structure 2 ..................................................................................................... 20
   3.4 Parallel Viterbi Algorithm Based on Tail Biting Structure .................................................. 22
   3.5 Summary .............................................................................................................................. 22

4 Parameter Study ...................................................................................................................... 23
   4.1 Size of Subblock .................................................................................................................. 23
   4.2 Size of Alpha section .......................................................................................................... 23
   4.3 Size of Beta section ............................................................................................................. 24
   4.4 Maximum number of iterations ......................................................................................... 25
   4.5 Conclusion ........................................................................................................................... 25

5 Eagle Simulations ..................................................................................................................... 27
   5.1 Eagle Simulator .................................................................................................................... 27
   5.2 Eagle implementation .......................................................................................................... 27
   5.3 Performance Evaluations ..................................................................................................... 28
      5.3.1 Simulation Settings ....................................................................................................... 28
      5.3.2 Location of simulation results ...................................................................................... 28
      5.3.3 Parameter study in Parallel Viterbi Algorithms ........................................................... 29
      5.3.4 Simulation result for AWGN channel ......................................................................... 33
      5.3.5 Simulation result for ETU70 channel .......................................................................... 34
   5.4 Testdata Generation ............................................................................................................ 36
      5.4.1 Simulation Settings ....................................................................................................... 36
      5.4.2 Location of simulation results ...................................................................................... 37
   5.5 Conclusion ........................................................................................................................... 37
6 Multi-Core Implementation
   6.1 Block of implementation
   6.2 Complexity Analysis
   6.3 Memory Consuming Analysis
   6.4 Conclusion

7 Conclusions
   7.1 Conclusions
   7.2 Future work

8 References
List of Figures

Figure 2.1: Rate 1/3 convolutional encoder................................................................. 11
Figure 2.2: A trellis diagram.................................................................................... 12
Figure 2.3: Three basic units of a Viterbi decoder.................................................... 13
Figure 2.4: Path metric calculation.......................................................................... 14
Figure 3.1: Parallel processing on a multi-core system......................................... 16
Figure 3.2: Data structure of subblock based on block-wise method ....... 17
Figure 3.3: Subblock partition................................................................................ 18
Figure 3.4: Data structure of new concatenated subblock ...................................... 19
Figure 3.5: Parallel Wrap-around Viterbi algorithm for subblock 1 ....................... 19
Figure 3.6: Data structure of middle subblock....................................................... 20
Figure 3.7: Parallel Wrap-around Viterbi algorithm for subblock 2 ....................... 21
Figure 3.8: Data structure of VD algorithm based on tail-biting structure........ 22
Figure 4.1: BLER performance with different values of Alpha............................ 24
Figure 4.2: BLER performance with different values of Beta............................... 25
Figure 5.1: BLER performance of Viterbi decoder, AWGN channel ..................... 29
Figure 5.2: BLER performance of Parallel Wrap-around VA, AWGN channel .... 30
Figure 5.3: BLER performance of Parallel VA, tail-biting structure, AWGN channel 31
Figure 5.4: BLER performance with optimal parameter settings......................... 32
Figure 5.5: BLER performance zoomed in around BLER=0.01 ....................... 32
Figure 5.6: BLER performance comparison, AWGN channel......................... 34
Figure 6.1: System configuration ......................................................................... 39
Figure 6.2: Cycle costs in wrap-around Viterbi decoder ...................................... 40
Figure 6.3: Cycle costs of parallel Viterbi decoder............................................. 41

Abbreviations

ACS             Add-Compare-Select
AWGN  Additive White Gaussian Noise
BLER  Block Error Rate
BM  Branch Metric
DSP  Digital Signal Processor
Eagle  E-UTRA algorithms for LTE
ETU  External Typical Urban
BPG  Baseband Performance Group
SISO  Sink Source
TD  Test Data
VA  Viterbi Algorithm
VR  Verification Report
VS  Verification Specification
VD  Viterbi Decoder
CQI  Channel Quality Indicator
CRC  Cyclic Redundancy Check
LTE  Long Term Evolution
PUSCH  Physical Uplink Shared Channel
1 Introduction

1.1 Background

Long Term Evolution (LTE) is the latest radio access technology specified by the 3rd Generation Partnership Project (3GPP), whose objectives include improved capacity, lower latency, higher throughput, increased spectrum efficiency and better coverage.

In LTE, the channel status reports are sent over the Physical Uplink Shared Channel (PUSCH), consisting of a Channel Quality Indicator (CQI) and a Precoding Matrix Indicator (PMI) which are encoded with tail-biting convolutional code [2]. Tail biting convolutional code, where the initial state of the encoder is given by the last bits of the data block to be encoded.

The Viterbi decoder is a maximum-likelihood decoder for block and stream convolutional code which is currently used in many areas, for example, Mobile Communications and Digital Video Broadcasting. Because of its simple and highly parallel structure, the Viterbi algorithm becomes for efficient hardware implementation. However, for large constrain length and data stream, the computational complexity and decoding latency cannot be ignored. In this case, the decoding delay and the amount of memory required for decoding long information sequence become a problem. The decoding cannot be started until the whole sequence is received, and all the surviving paths have to be stored in limited size of memories.

In 3GPP Release 10, as the number of sub-bands increases, the number of encoded CQI bits increases up to 350 bits, which makes latency requirements harder to fulfill. This means that optimizations of the Viterbi decoder implementation are of great importance, i.e., the parallelization of the Viterbi decoder implementation is highly needed.

1.2 Previous Work

Several methods for decoding convolutional code have been studied in recent years.

An alternative algorithm, the Lazy Viterbi algorithm, has been proposed recently [9]. This works by not expanding any nodes until it really needs to, and usually manages to get away with doing a lot less work (in software) than the ordinary Viterbi algorithm for the same result. However, it is not so easy to parallelize in hardware.

The Viterbi algorithm has been extended to operate with a deterministic finite automaton in order to quickly generate the trellis with state transitions pointing back at variable amount of history [10].
Wrap-around Viterbi decoding can be used to decode tail-biting convolutional codes. However, it is only valid for decoding small size of data. As the data size increases, the decoding latency becomes a problem and makes it not feasible in practice [8].

The Viterbi decoding solution, called the limited acquisition and survivor depth, pure feed forward parallel Viterbi decoder has been studied by G. Fetweis and H. Meyr [11].

However, most of these decoding methods are designed at the hardware level which means that the decoding of different codes may require complete changes of hardware architectures.

1.3 Objectives

It can be seen from previous section that by using general Viterbi decoder for large constrain length data steam, the computational complexity and decoding latency is a big problem when implementing it in the real communication systems. Therefore, investigation of a fast Viterbi decoder is of significant importance, especially for the LTE system which aims at providing significantly higher data rates and lower latencies for delay critical services. For the large size data sequence, the trace back loop in Viterbi decoder can cause a significant delay. To solve this problem, the parallel Viterbi algorithm should be studied and implemented.

The objective of this thesis is to theoretically investigate how to parallelize the Viterbi decoding algorithm over several DSPs in a multi-core environment. MATLAB simulations need to be done to find the optimal parameters for the new algorithm. And a link level simulator is used to verify the BLER performance of the algorithm under different scenarios. In addition, the new decoding algorithm is implemented on target DSPs for a multi-core system and the performance regarding cycle and memory consumptions is studied and evaluated against current decoding algorithms.

1.4 Research Approach

In this thesis, the theoretical investigation of a new Viterbi decoding algorithm is done first. It is based on the Wrap-around Viterbi decoder by introducing the parallelization to reduce computational complexity and decoding delay.

After the algorithm investigation, the new algorithm is first studied by using MATLAB simulations to find out the optimal parameters for the parallel Viterbi decoder for decoding 384 bits CQI_CRC with AWGN channel model.

Then the new algorithm with tuned parameters is verified by using EAGLE (E-UTRA algorithms for LTE) which is a C++ based link level simulator with a collection of libraries and algorithm models for testing and verifying LTE baseband functionality.
The algorithm verified by the EAGLE simulator is implemented on the target DSPs for Ericsson Multi-Core system in C and Assembly language. The decoded CQI bits on the target DSPs are required to be bit exactly the same as the EAGLE simulation results. The total required memories and cycles per state and bits in new algorithm also are compared to them in original Wrap-around Viterbi decoder.

1.5 Thesis Outline

The thesis report is organized as follow.

Chapter 2 presents the general concepts of convolutional codes and tail-biting convolutional codes. Solutions to decode convolutional codes, including the Viterbi decoder, and the Wrap-around Viterbi are also given in this chapter.

Chapter 3 presents two proposed parallel Viterbi decoder algorithms, the optimal parameter setting and MATLAB simulation results.

Chapter 4 presents the performance results: BLER under AWGN and ETU70 in EAGLE. The results are compared to the Wrap-around Viterbi decoder. Test data for Hardware implementation is also generated by Eagle in this chapter.

Chapter 5 gives a solution for the implementation of the Parallel Viterbi decoder for multi-core system. The cycles cost and the decoding latency for decoding 384 bits CQI_CRC in the Parallel Viterbi decoder is compared to, the Wrap-around Viterbi decoder.

Conclusions and future work are stated in Chapter 6.
2 Viterbi algorithms

The scope of this chapter is to describe the basic convolutional codes, tail biting convolutional codes and their decoding solution.

There are several approaches to decode convolutional codes including sequential decoding and maximum likelihood decoding. The Viterbi algorithm is the best known implementation of the maximum likelihood decoding solution.

2.1 Introduction

In telecommunication, a convolutional code is a type of Forward Error Correction (FEC) code where some carefully designed redundant information is added to the data transmitted through the channel. It is commonly used to correct errors in noisy channels.

Compare to the simple encoding procedure in convolutional encoding, the decoding of a convolutional code is a much more complex task. Several classes of algorithms exist for this purpose:

1. **Threshold decoding** is the simplest one of them. But it can only be successfully applied to the specific classes of convolutional codes. The performance is also far from optimal [4].

2. **Sequential decoding** is a class of algorithms performing much better than threshold algorithms. The biggest advantage is that the decoding complexity is virtually independent from the length of the particular code. Although sequential algorithms are also suboptimal, they are successfully used with very long codes, where no other algorithm can be acceptable. The main drawback of sequential decoding is unpredictable decoding latency [4].

3. **Viterbi decoding** is an optimal algorithm for decoding a convolutional code in a maximum-likelihood sense. Its main drawback is that the decoding complexity grows exponentially with the code constraint length. So, it can be utilized only for codes with a relatively short constraint length[4].

2.2 Convolutional Codes

Convolutional codes were first introduced by Elias in 1955 [13].

Like the other error correcting codes, a convolutional code adds some structured redundant information to the data and uses this information to correct errors. Convolutional codes are usually specified by three parameters $(N, K, M)$

\[ N = \text{number of output bits} \]
\( K \) = number of input bits
\( M \) = number of memory registers

In some cases, we can also use constrain length to present the number of bits in the encoder memory that affects the generation of the \( N \) output bits. The constrain length of the code \( L \) is defined by

\[
L = K \times (M-1)
\]

The code rate is expressed as a ratio of the number of input information bits to the number of output by the convolutional encoder.

\[
R = \frac{K}{N}
\]

In this thesis, we deal with \((3, 1, 7)\) convolutional code.

### 2.2.1 Convolutional encoder

In communication systems, a binary convolutional encoder can be seen as one or more shift registers and multiple XOR gates. The information sequence is the input to the encoder. A combination of registers' cells that forms one of the output streams is defined by a polynomial.

For the \((3, 1, 7)\) convolutional code in this thesis, the generator polynomials are defined as:

\[
G_0 = 133_{(octal)}, \quad G_1 = 171_{(octal)}, \quad G_2 = 165_{(octal)}
\]

The encoder output corresponds to the first, second and third parity stream respectively as shown in Figure 2.1 [2].

![Figure 2.1: Rate 1/3 convolutional encoder](image-url)
2.2.2 Trellis diagram

A trellis is one of the most convenient ways to visualize the principle of the decoding algorithms of the convolutional codes. We can see a convolutional encoder as a finite state machine. Each state corresponds to some value of the encoder’s memory register. In trellis diagram, the system is modeled as finite number of states. The total number of the states is $2^M$, where $M$ is the number of encoder registers.

An example of trellis diagram can be seen in Figure 2.2.

![Trellis Diagram](image)

Each node in the trellis represents the memory register state. For tail biting convolutional code (3, 1, 7) with 384 information bits, there are 64 states and 384 stages. The initial state and the final state should be the same, because of tail biting properties.

The free distance, which is a minimum hamming distance between two different encoded binary sequences, is an important property of the convolutional code. It influences the correcting capability of a convolutional code which is the number of closely located errors that the decoder is able to correct.

2.3 Viterbi Algorithm

The Viterbi algorithm was developed by Andrew J. Viterbi and published in the paper “Error Bounds for Convolutional codes and an Asymptotically Optimum decoding algorithm”, IEEE Transactions on Information Theory, Volume IT-13, pages 260.-269, in April, 1967. [1]

The algorithm makes a number of assumptions to satisfy the first order hidden Markov model.

1. First, both the observed events and hidden events must be in a sequence. This sequence often corresponds to time (stage).
2. Second, these two sequences need to be aligned, and an instance of an observed event needs to correspond to exactly one instance of a hidden event.

3. Third, computing the most likely hidden sequence up to a certain point $t$ must depend only on the observed event at point $t$, and the most likely sequence at point $t-1$.

### 2.3.1 Implementation

A Viterbi decoder usually consists of the following three basic units: Branch metric (BMU), path metric (PMU) and traceback (TBU) as shown in Figure 2.3.

**Figure 2.3: Three basic units of a Viterbi decoder**

1. **Branch Metric Unit:**

   A branch metric unit is to calculate branch metrics which are the distances between the received symbols pair of bits and all possible pairs in the codebook. There are two different methods for calculating branch metrics:

   - A *Hard decision* decoder where the Hamming distance between the received pair of bits and the ideal pair is measured.
   
   - A *soft decision* decoder where a branch metric is measured using the Euclidean distance.

2. **Path Metric Unit:**

   For every encoder state, a path metric unit calculates a metric for the survivor path ending in this state (a survivor path is a path with the minimum metric).

   Path metrics are calculated using a procedure called ACS (Add-Compare-Select) unit. It contains two steps and is repeated for every encoder state.

   1. Add – For a given state, we know two states on the previous stage which can move to this state, and the output bit pairs that correspond to these transitions. To calculate new path metrics, we add the previous path metrics with the corresponding branch metrics.
   
   2. Compare and select – we get two paths ending in a given state from the first step. The best survivor path is chosen.
3. Trace Back Unit

The Trace-Back Unit is a backward processing algorithm and requires the decisions from the ACS to be stored in a memory.

Trace Back Unit restores a maximum-likelihood best survivor path from the decisions made by Path Metric Unit. Since it is done in inverse direction, a Viterbi decoder comprises a FILO (first-in-last-out) buffer to reconstruct a correct order.

The general approach to trace back is to accumulate path metrics for up to five times of the constraint length, find the node with the largest accumulated cost, and begin traceback from this node.

2.4 Tail-biting Convolutional Code

The most common convolutional codes without tail biting are generated by an encoder which adds $M$ dummy symbols at the end of the information sequence at starts and ends in predefined states. This results in a loss in rate of the code. When transmitting short packets of data, the loss is too big.

There is another solution called the tail biting convolutional code in which the encoder starts in the state given by the $M$ last symbols of the information sequence. Hence, the encoder starts and ends in the same state and thus the loss in rate of the code is less in this case. However, the decoder gets more complicated to be implemented since the starting and ending states of the encoder are unknown to the decoder.
A path in a tail biting trellis is a valid codeword if and only if its starting and ending states are the same. Since there are multiple starting states, a transmitted codeword can start from any of its starting states which is unknown to the receiver. Therefore, any tail biting trellis-based decoding algorithms must have a mechanism to estimate the starting state for each transmission of a codeword.

In our case, the initial value of the shift register of the encoder shall be set to the values corresponding to the last 6 information bits in the input stream so that the initial and final states of the shift register are the same.

2.5 Wrap-Around Viterbi Algorithm

The Wrap-Around Viterbi decoder is an iterative algorithm for decoding tail biting convolutional codes. It is not an optimal Maximum Likelihood Decoding algorithm. However, simulation results show that this algorithm achieves near optimum error performance with only 2 to 4 iterations for many convolutional codes.

This algorithm is devised based on processing a tail biting trellis T repeatedly in a continuous manner with the Viterbi algorithm. Each Viterbi processing of T is called an iteration. The algorithm consists of a sequence of decoding iterations. The available soft information is transferred from current iteration to the next iteration. The algorithm starts the decoding process from all the states with the same initial state metrics at the first iteration. At the end of each iteration, if no decoding decision is made, the metrics of the ending states are used as the starting state metrics of the next iteration. This is called wrap-around Viterbi decoding [8].

When the decoding latency is an important factor, the Wrap-around Viterbi algorithm is only suitable for decoding codes of small size because the decoding delay increases a lot when the length of the code increases.
3 Parallel Viterbi Algorithms

In this chapter, we investigate two parallel Viterbi algorithms for decoding a large size of tail biting convolutional codes, aiming at achieving lower decoding delay and decreased, computational complexity based on their tail-biting property.

3.1 Overview of Parallel Processing

A parallel algorithm is an algorithm which can be executed a piece at a time on many different processing devices. The outputs from each piece are then combined in the end to get the correct result.

In a multi-core system, multiple processing elements can be simultaneously used to achieve low latency and high throughput. There are different ways of parallel processing, e.g., instruction-level parallelism, bit-level parallelism and data parallelism, which should be chosen based on the platform architecture. For the Multi-Core system in this thesis, parallelism on data structure is the best choice.

Figure 3.1 shows the concept of parallel processing on a multi-core system. The problem is decomposed into independent subtasks, and then dispatched into each processing element to execute its part of the algorithm simultaneously with the others, finally put back together again at the end to get the correct result.

![Figure 3.1: Parallel processing on a multi-core system](image)

3.2 Block-wise Viterbi Algorithm

The parallel architectures can be used to divide the input data to several small sub-blocks, and solve it in parallel to reduce the decoding latency.
For terminated trellis diagrams, the start and end states are known. And we can make a decision on the overall best path at the end of the trellis. The decoding latency for the Viterbi algorithm is then proportional to the length of the trellis. However, in some applications like decoding a continuous sequence of information bits or the decoding which starts and ends in the middle part, the knowledge of the start/end states is not known. By using the acquisition and truncation properties of the Viterbi algorithm, these convolutional code sequences with unknown start and end state can be correctly decoded.

Figure 3.2 shows decoding of middle part subblock by using block-wise method. In order to correctly decode subblocks, additional data sections need to be added both before and after desired decoding data section. We denote them as Alpha section and Beta section shown in Figure 3.2.

If all n paths at time B₃ are traced back in time, it is highly probable that they merge at time M₄. When a block of A+M+E step has been processed, the decisions of the first A steps have to be discarded as unreliable. In the last B steps, the unique path branches out to N paths from which, after processing further steps of the trellis, one of the n will be chosen as the optimum path of the time interval (M₁, M₄). Solely by exploiting this algorithmic knowledge, new parallel VD architectures were recently derived. [12]

![Figure 3.2: Data structure of subblock based on block-wise method](image)

Viterbi decoding processing starts from the first bits of Alpha section, these additional Alpha sections is used for the acquisition of the path metrics. If Alpha value is sufficient, there is high probability of getting correct initial path metric of state m₁. Notice that the decisions of the Alpha sections have to be discarded as unreliable.

The Beta sections are obtained with correct path metrics, and P paths are correctly decoded, where P is the number of states. However there are not unique. We have needed extra information to find the most likelihood survivor from these N survivor paths.
By using the block-wise method, the convolutional encoded data can be divided into three different types of subblocks: the start subblock, the middle block and the end subblock.

![Figure 3.3: Subblock partition](image)

Here, the middle part subblock consists of three basic data sections: Alpha part, Reliable part, and Beta part.

However, there is no acquisition information part for the first subblock and the last subblock. Without the correct start and ending state information, the decoding cannot be done correctly. For tail-biting convolutional codes, the tail-biting property can be used to estimate/determine correct start and end states of tail-biting convolutional codes. The details will be further described in section 3.3, section 3.4

### 3.3 Parallel Wrap-around Viterbi Algorithm

This method increases the probability to find out the correct initial metrics of the first subblock by using wrap-around Viterbi decoder. There are two different types of subblocks based on the wrap-around Viterbi decoder and the block-wise method.

#### 3.3.1 Subblock structure 1

This structure is based on wrap-around Viterbi decoder. The tail-biting check needs the initial state and the ending state. The head part and the end part should be processed in one thread. In order to process it in parallel, the tail-biting part, start state and end state should be put into one subtask. Figure 3.4 shows the solution, where the middle part can be seen as the unique decoded part. The new concatenated tail-biting subblock can use wrap-around algorithm for decoding.
The flow chart of the algorithm for one subblock based on tail-biting check is shown in Figure 3.5.

Figure 3.5: Parallel Wrap-around Viterbi algorithm for subblock 1
The algorithm for a new concatenated subblock consists of the following steps:

1. Initialization: start from all the states in the first stage with the same initial state metric.

2. For 1<i<l, execute the i-th decoding iteration with the last state metric in the previous iteration. At the section boundary location, compare the metrics of all the L branch surviving paths that terminate at the ending states.

3. Select the best path as the winning path of the current iteration. If this best path is the tail biting path, go to Step 5. Otherwise, update the best path and go to the next step.

4. Find the best surviving tail biting path. Compare the metrics of the best surviving tail biting path. Check iteration number condition, i < l. If true, set l = i+1 and go to Step 2. Otherwise, go to Step 6.

5. Output the best path as the decoded codeword.

6. Output the best tail biting path as the decoded code word if it exists. Otherwise output the best path as the decoded word.

7. Remove the middle acquisition part, send to concatenating step.

3.3.2 Subblock structure 2

This structure is based on block-wise method and shown in Figure 3.6.

![Figure 3.6: Data structure of middle subblock](image)

The algorithm can be formulated as general Viterbi decoding processing which is shown in Figure 3.7.
The algorithm for mid-subblock consists of the following steps:

1. Initialize all the start states in the first stage with the same initial state metric.

2. Execute the Viterbi decoding process. At the ending stage, compare the metrics of all the L branch surviving paths that terminates at the ending states.

3. Find the best surviving path for each state.

4. Select the best path as the winning path.

5. Trace back from the best path, output it as the decoded codeword. (Note that only the Beta section and the Reliable section are traced back)

6. Remove the Beta acquisition section and send the Reliable part to concatenating step.
3.4 Parallel Viterbi Algorithm Based on Tail Biting Structure

By the tail-biting structure and the block-wise method, we can find the correct initial state and the optimum survivor path for the last state.

The tail-biting structure uses the last few bits as the memory register of the first state. Then the memory register states can be seen as a ring structure by adding the end section of the last subblock to the head of the first state. Since the initial state and ending state are the same in tail-biting convolutional code and the decoding only depends on the current state of the memory register, we can use the last few data bits to determine the initial state of the first subblock.

Figure 3.8: Data structure of VD algorithm based on tail-biting structure

This parallel algorithm has the same subblock structure. Therefore, it is easy to program in a multi-core system. It can also obtain a great speedup gain simply by adding more processors. The flow chart of the algorithm is the same as which is shown in Figure 3.7 in section 3.3.2.

3.5 Summary

A method for decoding a convolutional code block without knowledge of the start/end states is introduced in this chapter. Two parallel Viterbi algorithms for decoding tail-biting convolutional code are proposed in this chapter: an algorithm based on tail-biting check and an algorithm based on the tail-biting structure. Both of the algorithms can find the correct initial state and the maximum-likelihood survivor path at the last state.
4 Parameter Study

The performance of an algorithm is directly affected by parameter settings. This section studies parameters of the parallel Viterbi decoder. MATLAB simulations are done to see how the parameters affect the system performance. All the simulations in this section are done for the Parallel Viterbi algorithm based on tail biting structure, under AWGN channels and 1000 blocks are run to get enough statistics.

4.1 Size of Subblock

The number of sub-blocks affects the decoding speed and memory consumption. With the increment of the number of sub-blocks, the decoding speed is increased and usage of memory per sub-block is decreased which will give better performance. However, the total number of sub-blocks is limited by the number of DSPs in the multi-core system.

Currently, the Wrap-around Viterbi decoder supports the number of CQI information bits up to 68 bits. And in this thesis, the size of each sub-block is set to be 64 bits which is reasonable and practical.

4.2 Size of Alpha section

Alpha part in the subblocks is used for the acquisition of the path metrics. With a large size of Alpha section, more reasonable path metrics can be obtained. However, it also costs more computational cycles and memories to store the path metrics for Alpha section.

With MATLAB simulation, different values of Alpha section size are tested for 8, 16 and 32 bits. Note that the value of Beta is set as 32 bits. It can be seen from Figure 4.1 that 32 bits performs better than other values regarding the BLER performance. The performance is improved with the increment of Alpha value.
4.3 Size of Beta section

Similarly as Alpha, with a large value of Beta, BLER performance will be increased.

With MATLAB simulations, different values of Beta are tested for 8, 16 and 32 bits. Note that the value of Alpha is set as 32 bits. It can be seen from Figure 4.2 that 32 bits performs better than other values regarding the BLER performance.
Figure 4.2: BLER performance with different values of Beta

4.4 Maximum number of iterations

The wrap-around Viterbi algorithm has been applied to decode various blocks and convolutional codes based on their tail biting trellises. This algorithm achieves near optimum MLD error performance for all the codes being decoded with a maximum number of 2 to 4 iterations. For convolutional codes with $L \geq M$, where $L$ is the code size, $M$ is the memory order, 2 iterations are enough for the wrap-around Viterbi algorithm to achieve near optimum error performance [8].

4.5 Conclusion

It can be concluded that the BLER performance improves when the size of Alpha and Beta section increase. Regarding the maximum number of iterations in parallel wrap around Viterbi algorithm, 2 iterations are enough to achieve near optimum error performance.
The parameter settings for new algorithms are also studied in Matlab to get the optimal performance for each algorithm. However, the MATLAB simulation in this chapter is simplified which is just used to investigate the performances of the proposed algorithms. More simulations are done in Eagle in the next chapter.
5 Eagle Simulations

In this chapter, we use a link level simulator Eagle to evaluate the BLER performance of the two proposed parallel Viterbi algorithms in 3.3 and 3.4. The aim is to take one of them as the final solution for the target multi-core system, and to generate testdata for the DSP implementation in chapter 6.

5.1 Eagle Simulator

Eagle (E-UTRA Algorithms for LTE) is a Ericsson internal link level simulator, which is intended for performance optimization and verification of LTE layer 1 algorithms. It can be used to generate data at various points in the encoder and decoder chain. And it is a bit exact simulator which can be used to verify the new algorithm by physical layer design, nodal integration testing, etc. More information about it can be found in [5].

In this thesis, Eagle simulations have been performed. Specifically, two tasks are done with Eagle simulator:

- Performance evaluation: Find out the optimal parameters for the two proposed parallel Viterbi decoder algorithms. Carry out the performance test to evaluate the performance of the two algorithms and choose a better one.

- Generate testdata to verify the results for the target DSP.

5.2 Eagle implementation

Since Eagle simulator doesn’t support multiprocessing, each parallelized subtask is implemented in sequence to verify BLER performance of the new parallel Viterbi decoder algorithm.

The newly added/affected files are:

- ul_conv_decoder_wa.cpp:
  Wrap-around Viterbi Algorithm, can be used as the reference of non-parallel solution.

- ul_conv_decoder_para.cpp:
  Parallel Viterbi Algorithm based on tail biting state check.

- ul_conv_decoder_para_v2.cpp:
  Parallel Viterbi Algorithm without tail-biting property.

- ul_conv_decoder_para_v3.cpp:
Parallel Viterbi Algorithm based on tail biting structure.

5.3 Performance Evaluations

The performance tests are done for evaluating the new algorithms under different channels and the results are presented in this section. The reason of using BLER to evaluate the algorithm performance is that BER is not applicable here because Cyclic Redundancy Check (CRC) fails if any of the CQI bits is decoded incorrectly. So BLER is used to evaluate the new algorithm performance in this thesis.

5.3.1 Simulation Settings

The test case specification file VS used for running the simulations is:

/vobs/bts_common/ebpg/dev/eagle/vs/vs_perftest_pushc_cqi.xls

Some parameters are listed in Table 5.1:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoder. model</td>
<td>'wrap_around_viterbi', 'parallel_viterbi_v1', 'parallel_viterbi_v2', 'parallel_viterbi_v3'</td>
</tr>
<tr>
<td>include channel</td>
<td>AWGN, ETU70</td>
</tr>
<tr>
<td>MCS</td>
<td>25rb_gqsk</td>
</tr>
<tr>
<td>Number of frames</td>
<td>1000 (AWGN), 2500 (ETU70)</td>
</tr>
<tr>
<td>SNR</td>
<td>[M, M+N, M+2N, M+3N, M+4N, M+5N, M+6N]</td>
</tr>
<tr>
<td>Number of CQI bits</td>
<td>376</td>
</tr>
<tr>
<td>Simulation type</td>
<td>BLER</td>
</tr>
</tbody>
</table>

The number of frames is dependent on what BLER level to achieve and how many error events should be included to have a good statistics. Therefore, 1000 frames and 2500 frames are run for the AWGN and the ETU70 channel, respectively.

5.3.2 Location of simulation results

The simulation results are stored:

/bts_common/ebpg/sim/studies/studies_ejuzilo_pushc_cqi/r2_hw2_75bb_fix/eagle3.3.0/cqi
5.3.3 Parameter study in Parallel Viterbi Algorithms

We studied parameter settings in parallel Viterbi algorithms by using Eagle simulator. Note that for confidential reasons, some parameters like SNR values are normalized.

1. Parallel Viterbi algorithm without using tail biting property:

Figure 5.1 shows the BLER performance of the parallel Viterbi decoder without tail biting property under AWGN channel. It can be seen as verifying the performance of the block-wise method.

![Figure 5.1: BLER performance of Viterbi decoder, AWGN channel](image)

There are no exact BLER requirements on PUSCH CQI. However, we are usually more interested in the area where BLER = 0.01. From Figure 5.1, we can see that the decoder algorithm provides better BLER performance with the increment of the value of Alpha and Beta.

2. Parallel Wrap-around Viterbi decoder:

Figure 5.2 shows the BLER performance of the parallel Wrap-around Viterbi algorithm under AWGN channel.
From Figure 5.2, we can see the similar tendency, i.e., the increment of the value of Alpha and Beta increases BLER performance. When looking at the part where BLER = 0.1, the algorithm provides best performance with Alpha = 32, Beta = 32. However, with alpha = 8, beta = 8, the BLER is above 0.01 even with good channel quality.

3. Parallel Viterbi algorithm with tail biting structure

Figure 5.3 shows the BLER performance of parallel Viterbi algorithm with tail biting structure under AWGN channel.
Figure 5.3: BLER performance of Parallel VA, tail-biting structure, AWGN channel

From Figure 5.3, we can see the similar tendency, i.e., the increment of the value of Alpha and Beta increases BLER performance. With Alpha = 32, Beta = 32, the algorithm provides best performance. And all the other values of Alpha and Beta can provide good BLER performance.

4. Comparison with optimal parameter settings

Now, we compare the performance of the three parallel Viterbi algorithms and the non-parallel wrap-around Viterbi decoder algorithm with the optimal parameter setting Alpha = 32, Beta = 32, Number of maximum iterations = 2 as shown in Figure 5.4. The area around BLER = 0.01 is zoomed in and shown in Figure 5.5.
**Figure 5.4: BLER performance with optimal parameter settings**

**Figure 5.5: BLER performance zoomed in around BLER=0.01**
From the Figure 5.4 and Figure 5.5, we can see that by using the tail biting property, parallel Viterbi decoders can achieve the same level BLER performance as the non-parallel Wrap-around Viterbi decoder. The parallel Viterbi algorithm based on tail-biting structure performs better than other proposed algorithms.

However, with optimal parameter setting of Alpha = 32 and Beta = 32, the total computational complexity increases. Considering the BLER performance and computational complexity, Alpha = 8 and Beta = 16 can be a good parameter setting solution for implementation in the multi-core system. And this setting is used for further performance verification under different channels in the following sections.

5.3.4 Simulation result for AWGN channel

In this section, the performances of three Viterbi decoding algorithms under AWGN Channel are studied.

- Wrap-around Viterbi algorithm with number of maximum iteration = 2.
- Parallel Wrap-around VA, Alpha=8, Beta=16, I =2
- Parallel VA based on tail biting structure, Alpha=8, Beta=16

The simulated frame number is 1000 corresponding to 10000 times decoding of CQI.

The simulation results are shown in Figure 5.6. It can be seen that all of the three algorithms provide good BLER performance under AWGN channel with a good channel quality. However, when it comes to the area where BLER = 0.01, the parallel Viterbi algorithms perform slightly worse than Wrap-around Viterbi decoder.
Figure 5.6: BLER performance comparison, AWGN channel

More specifically, to get BLER $< 10^{-2}$, wrap-around Viterbi decoder requires $\text{SNR} = M+1.25 \text{ dB}$, parallel Viterbi decoder based on tail biting structure requires $\text{SNR} = M+1.55 \text{ dB}$ and parallel Wrap-around Viterbi algorithm requires $\text{SNR} = M+1.75 \text{ dB}$, respectively. This is shown in Table 5.2 below.

**Table 5.2 parameter settings for Eagle performance tests**

<table>
<thead>
<tr>
<th>Decoder algorithm</th>
<th>SNR requirement for BLER 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrap-around Viterbi algorithm</td>
<td>$M+1.25 \text{ dB}$</td>
</tr>
<tr>
<td>Parallel Viterbi algorithm, tail biting structure</td>
<td>$M+1.55 \text{ dB}$</td>
</tr>
<tr>
<td>Parallel Wrap-around Viterbi algorithm</td>
<td>$M+1.75 \text{ dB}$</td>
</tr>
</tbody>
</table>

5.3.5 **Simulation result for ETU70 channel**

The simulations are also executed for ETU 70 (External Typical Urban) channel to see if the new parallel Viterbi algorithms are sensitive in this case.
The simulation numbers are set to 2500 frames, since we need to capture enough time fading cycles in ETU channel. The SNR range also changed to \([N+1,N+7]\).

The other parameter settings are the same as the AWGN channel case in section 5.3.4.

Table 5.3 BLER performance comparison, ETU 70 channel

With ETU70 channel, the simulation results are similar to in AWGN channel. Parallel Viterbi algorithm is slightly worse than wrap-around Viterbi algorithm case as shown in .

More specifically, to get BLER < \(10^{-2}\), wrap-around Viterbi decoder requires \(SNR = N+3.5\) dB, parallel Wrap-around Viterbi algorithm based on tail biting structure requires \(SNR = N+3.6\) dB and parallel Wrap-around Viterbi algorithm requires \(SNR = N+4\) dB, respectively. This is shown in Table 5.3 below.

However, considering that we can get high speedup gain in parallel Viterbi algorithm case, these differences are acceptable.
Table 5.3 parameter settings for Eagle performance tests

<table>
<thead>
<tr>
<th>Decoder algorithm</th>
<th>SNR requirement for BLER 0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrap-around Viterbi algorithm</td>
<td>N+3.5 dB</td>
</tr>
<tr>
<td>Parallel Viterbi algorithm based on tail biting structure</td>
<td>N+3.6 dB</td>
</tr>
<tr>
<td>Parallel Wrap-around Viterbi algorithm</td>
<td>N+4 dB</td>
</tr>
</tbody>
</table>

5.4 Testdata Generation

In order to verify the algorithm implementation on target DSP, testdata is generated by Eagle. It is typically a batch of short tests intended to produce input/output data for bit exact verification between Eagle models and target implementation.

The verification of the target code comprises of the following steps:

- Generate test data i.e. Eagle generates input file F1 and output file F2.
- Run target code to read file F1 and generate file F3.
- Compare files F2 and F3. Pass if files are equal.

From the performance evaluation above, we decided to use Parallel Viterbi decoder based on tail biting structure, with Alpha=8, Beta=16 as the final implementation solution for decoding CQI on PUSCH for Multi-core system.

5.4.1 Simulation Settings

The parameter setting file VS used for generating testdata is located at:

/vobs/bts_common/ebpg/dev/eagle/vs/vs_testdata_up_ul_cqi.xls
/vobs/bts_common/ebpg/dev/eagle/vs/vs_testdata_up_ul_cqi.iop

Some parameters are listed in Table 5.4 below.

Table 5.4: Parameters for Testdata Simulation

<table>
<thead>
<tr>
<th>Test case</th>
<th>'wrap_around_viterbi', 'parallel_viterbi_v3'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>ETU70</td>
</tr>
<tr>
<td>Number of frames</td>
<td>1</td>
</tr>
<tr>
<td>SNR</td>
<td>-5 and 5 dB</td>
</tr>
</tbody>
</table>
Only one frame simulation is needed for testdata generation. Since the number of iterations in the wrap-around Viterbi decoder depends on the channel quality, two batches of testdata are generated for different channel properties as below:

1. ETU 70, SNR = 5 dB: Wrap-around Viterbi decoder doesn’t run the second iteration for good channels which decreases the computational complexity.

2. ETU 70, SNR = -5 dB: Wrap-around Viterbi decoder runs the second iteration to find the best tail biting path under bad channels. It increases the computational complexity in Viterbi trial step in programming.

5.4.2 Location of simulation results

The testdata is stored at:
/proj/ebpg/sim/testdata/testdata_up_ul_cqi/

Test bench specific Eagle variables:

- EAGLE_VS_CQI "testdata_up_ul_cqi"
- EAGLE_BASELINE_CQI "r2_hw2_75bb_fix"
- EAGLE_VERSION_CQI "eagle3.3.0_ejuzilo"

5.5 Conclusion

In this chapter, the two parallel VD algorithms are further studied by using Eagle simulator to get a precise performance evaluation. In addition, testdata for the implementation on target DSP is generated by Eagle.

Parallel wrap-around Viterbi algorithm uses wrap-around iteration to find out the initial metrics of the first state. And parallel solution based on tail-biting structure uses tail-biting, block-wise method to find out the unique initial metrics of the first state. Both of the algorithms can achieve a good BLER performance. However, the latter algorithm has a simpler data structure and hence leads to a simpler implementation. Therefore, the latter algorithm is chosen for implementation on target DSP.
From an implementation point of view, the parallel Viterbi algorithm based on tail biting structure has a unique and simple structure which is easy to be implemented. The other solution with wrap-around VA consists of tail-biting subblock and block-wise subblock. It is a kind of adaptive solution where the total computation cost depends on the channel quality. This is further discussed in the next chapter.
6 Multi-Core Implementation

This chapter presents the DSP implementation of the parallel Viterbi decoder algorithm based on tail biting structure. The implementation is done on Ericsson multi-core system. The cycle and memory costs are studied for the new parallel VD algorithm.

6.1 Block of implementation

To implement the solution proposed in the previous section, a number of identical parallel Viterbi decoders are required. The complete system configuration is shown in Figure 6.1. The soft values of CQI information bits with CRC will be divided to several subblocks. Then GDE dispatches these thread to each DSPs to execute the Viterbi decoding processing. When all the threads are finished, the results will be concatenated to get the final decoded result.

![Figure 6.1: System configuration](image)

6.2 Complexity Analysis

The testdata generated in section 4.4 is used to verify the correctness of the DSP implementation. Meanwhile, the complexity and decoding latency of the proposed parallel Viterbi algorithm is studied.
First, the Wrap-around Viterbi algorithm case is studied. Figure 6.2 below shows the cycle costs for decoding 384 bit of CQI_CRC information report. For confidential purposes, the cycles here are normalized with an $x$, the same rules are used for the following parts of the thesis report.

![Diagram](Image)

**Figure 6.2: Cycle costs in wrap-around Viterbi decoder**
There are two sets of simulation results for the wrap-around Viterbi decoder algorithm. As discussed in section 4, there can be different iterations to find the best tail biting survivor path depending on channel quality. With bad channel quality, if the decoder couldn’t find the best tail biting survivor in the first iteration, it will execute the second iteration processing. On the other hand, only one iteration of Viterbi Trial is needed to find the best path if the channel is good and this saves many cycles. Both of the above cases are studied and shown in Figure 6.2. The results of cycle costs will be used as a reference to study the cycle costs of the proposed parallel Viterbi decoder algorithm.

Figure 6.3 shows the cycle costs for decoding 384 bits of CQI_CRC information report by using proposed parallel Viterbi decoder algorithm.
It can be seen that the total number of cycles for decoding 384 bits CQI_CRC is between wrap-around Viterbi decoder using 1 iteration and 2 iterations. The explanation is that each of subblock has additional acquisition section Alpha = 8 and Beta =16. Therefore, in Viterbi decoding step, 384*1.375= 528 bits of CQI_CRC is processed in total. However, since it doesn’t execute tail-biting state check, it can save the cycle consumption in this step and no extra iteration steps are needed even under bad channel. The total cycle cost is not reduced, but the cycle cost per each DSP can be reduced to 148.6x cycles. Comparing to non-parallel wrap-around Viterbi decoder algorithm, the new proposed parallel Viterbi decoder algorithm has great benefits by decreasing the decoding latency. The cycle costs analysis can be seen in Table 6.1.

Table 6.1: Cycle costs comparison

<table>
<thead>
<tr>
<th></th>
<th>Wrap-Around Viterbi I=1</th>
<th>Wrap-Around Viterbi I=2</th>
<th>Parallel Viterbi Total</th>
<th>Parallel Viterbi Per DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle cost: (cycle)</td>
<td>88259.02</td>
<td>120144.08</td>
<td>126519.96x</td>
<td>21086.66x</td>
</tr>
<tr>
<td>Latency: (cycle/bit/state)</td>
<td>4.19x</td>
<td>5.70x</td>
<td>5.99x</td>
<td>1x</td>
</tr>
</tbody>
</table>

So by doing parallel processing, the decoding latency is significantly reduced. For our case of 384 bits CQI_CRC bits, by using 6 subblocks, Alpha=8, Beta=16, the final decoding latency is reduced:

\[
\frac{(4.19x-1x)}{4.19x} \times 100\% = 76%
\]

\[
\frac{(5.70x-1x)}{5.70x} \times 100\% = 82%
\]

The cycle cost depends on the number of subblocks and it can be further optimized for different cases, i.e., different length of CQI_CRC bits.
6.3 Memory Consuming Analysis

For DSP implementation, the memory usage is also an important factor to verify an algorithm. In a multi-core system, there is limited size of memory unit available on each DSP. Since all path metrics until trace back step need to be stored, we don't have enough memory to store all the data if the CQI size is very large. This can be a problem with limited memory on one DSP. However, it can be easily fixed by parallel Viterbi algorithm. Although the total memory costs are almost the same in both cases, the parallel solution split the memory usage to different DSPs which makes it feasible to decode even the large size of CQI bits. In another word, the memory usage per subblock is decreased. In addition, the memory can be released and hence reused after the complete each subtasks.

6.4 Conclusion

In this chapter, we compare the proposed parallel Viterbi algorithm with the original wrap-around Viterbi algorithm regarding computation complexity, decoding latency and the memory usage.

In serial processing algorithm, the decoding latency and memory requirement can be traded for computation complexity. But with parallel processing, we can get a significant performance gain of decoding latency, memory requirement and implementation complexity. On system level, it's more complex. But since the tail-biting check step is removed in each subtask, the algorithm is simplified.
7 Conclusions

7.1 Conclusions

The parallel Viterbi decoding method proposed in this thesis can be used for decoding large size of tail biting convolutional codes.

The approaches presented here are based on the acquisition and truncation properties of the Viterbi algorithm and tail biting. Two parallel Viterbi decoder algorithms which can obtain correct initial path metrics of the start and end state are studied in this thesis, i.e., Parallel Wrap-around Viterbi algorithm and Parallel Viterbi algorithm based on tail biting structure.

The performances of the two algorithms are first studied in the Eagle simulator. It can be seen that both of the two parallel Viterbi decoder algorithms with optimal parameter settings can provide as good BLER performance as the non-parallel Wrap-around Viterbi decoder algorithm. The difference of BLER performance between the two proposed algorithms is small. However, the algorithm based on tail biting structure has slightly better BLER performance. When it comes to implementation complexity and algorithm flexibility, the parallel Viterbi decoder based on tail biting structure also has better performance. Therefore, it is proposed to be the final solution to be implemented on target DSP for decoding PUSCH CQI reports in the LTE system.

The implementation of the proposed algorithm is done on the multi-core system. It can be seen from the results that the proposed parallel Viterbi decoder obtains a great speedup gain because of parallel computation. It can be used for decoding large size of data with low decoding latency. In addition, the limitation of memory problem on each DSP is solved by dividing and dispatching jobs to different DSPs.

7.2 Future work

The performance can be improved by using an adaptive method where the values of Alpha, Beta and the length of subblock are adaptive to the length of the CQI information bits on PUSCH.

To get even better BLER performance, the proposed two algorithms, i.e. Parallel Wrap-around Viterbi algorithm and based on tail biting structure, can be combined. Although the complexity will be increased, there can be some improvement of BLER performance.
8 References


[6] 3GPP TS 36.211: "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation".


