

Software Viterbi Decoder with SSE4 Parallel Processing Instructions for Software DVB-T Receiver

Shu-Ming Tseng
Department of Electronic
Engineering
National Taipei University
of Technology
Taipei, Taiwan
shuming@ntut.edu.tw

Yu-Chin Kuo
Department of Electronic
Engineering
National Taipei University
of Technology
Taipei, Taiwan
ken.yckuo@gmail.com

Yen-Chih Ku
Department of Electronic
Engineering
National Taipei University
of Technology
Taipei, Taiwan
t5418506@ntut.edu.tw

Yueh-Teng Hsu
Lite-On Technology
Corporation
Taipei, Taiwan
matthew.hsu@liteon.com

Abstract—In this paper, we discuss the procedures how to make Viterbi decoder faster. The implementation in Intel CPU with SSE4 parallel processing instruction sets and some other methods achieves the decoding speed 47.05 Mbps (0.64 Mbps originally). The DVB-T mode used in Taiwan needs 13.27 Mbps to achieve real-time reception, so our implementation of software Viterbi decoder takes only 28% CPU loading.

Keywords—software defined radio; Viterbi algorithm; parallel processing

I. INTRODUCTION

Recently, the Software Radio (SR) has become a popular in software Digital Audio Broadcasting (DAB) [1] and software Global Position System (GPS) [2]. As the operation of the computer has speeded up, it is possible to use the software to do the demodulation, decoding, and playback at real time. The SR has the following advantage: saving the hardware cost, flexible expansion, enhanced maintainability. We hope to implement the SR which is used for DVB-T signal. Therefore, we have to implement the software of the Viterbi decoder which is fast enough to decode in real time.

Because of these advantages, we are investigating an experimental software DVB-T receiver, which is an extension of our previous work on DAB [1]. According to the DVB-T standard [3], the useful bit rate of convolution code is 13.27 Mbps (16QAM, guard interval 1/4, code rate 2/3 for non-hierarchical system for 8 MHz channels.).

In order to speed-up program the Viterbi decoder for DVB-T in a systematic way, we use the Performance Explorer in Microsoft Visual Studio 2008 (Team Suite edition) to analyze the slowest parts of our programs. Then we improve the slowest parts by various ways described below. First we use the Single Instruction Multiple Data Stream (SIMD). Many people have used these instructions to speed up their program [4]-[7], but we choose the latest Intel CPU which contains the Streaming SIMD Extension 4 (SSE4) instructions. The new Intel CPU provides more SIMD instructions, and we use them to be our tool to saving the CPU operation time [9]-[13]. Second, we reduce the

branches of the program. We found that the branch commands (IF, FOR, etc.) takes a lot of times, so we try to reduce them as much as possible. Third, we modified the data structure of the function blocks of Viterbi decoder.

Finally, our software Viterbi decoder has decoding speed 47.05 Mbps which has satisfied our request. In the rest of this paper, we present our implementation in details.

II. SOFTWARE AND HARDWARE ENVIRONMENT

In this implementation, the hardware list is shown in table I. In order to use all the registers of the 64-bit CPU, we choose the Windows Vista (Ultimate 64-bit vision) to be our operation system. We use the Performance Explorer in Microsoft Visual Studio 2008 (Team Suite edition) to analyze the slowest parts of our programs. Then we improve the slowest parts by various ways described later.

The all estimations of efficiency use the Performance Explorer which calculates the time (T) of decoding 261120 bits of DVT-T. And then we can get the decoding speed (V) by

$$V \text{ (bits/s)} = 261120 \text{ (bits)} / T \text{ (ms)} \quad (1)$$

TABLE I. HARDWARE LIST

Item	Model
CPU	Intel Core 2 Duo E8400(3.0GHz)
Memory	DDR2 800 1GB x 2
Mainboard	Gigabyte GA-P31-DS3L
Graphic Card	NVIDIA GeForce 7300 GT(256M)

III. THE FUNCTION BLOCKS OF VITERBI DECODER AND SYSTEMATIC APPROACH OF SOFTWARE OPTIMIZATION

The Viterbi decoder consists of branch metric unit (BMU), add-compare-select unit (ASCU), path metric memory unit (PMMU), and survivor memory unit (SMU). And its procedures flow chart is shown in Fig. 1. In order to improve the bit error rate (BER) of our decoder, we will also use the results of the channel estimation to help the decoder [8]. And then carry on the work of the optimization in order to improve our decoder speed based on this program. The

procedures of the optimization are shown in Fig. 2. After finding out the bottleneck of the procedures by the Performance Explorer, we will improve the bottleneck of the procedure until the efficiency of the procedure accords with our need.

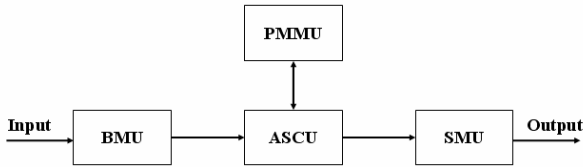


Figure 1. The procedure of Viterbi decoder flow chart.

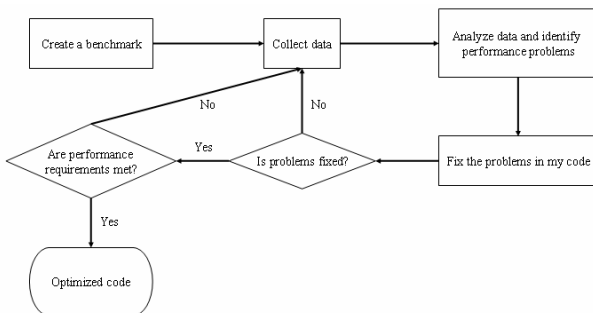


Figure 2. The systematic approach of software optimization.

IV. SSE4 INSTRUCTIONS

The most new processors provide SIMD instructions (as shown in Fig. 3), and many people have used these instructions to speed up their program [4]-[7]. The majority's CPU either use Intel or use AMD now, and they all have the Streaming SIMD Extension (SSE), Streaming SIMD Extension 2 (SSE2), and Streaming SIMD Extension 3 (SSE3) instructions. But we choose the latest Intel CPU which contains the Streaming SIMD Extension 4 (SSE4) instructions. The new Intel CPU provides more SIMD instructions, and we use them to be our tool to saving the CPU operation time [9]-[13]. We can let some operations use these instructions to combine, and it will make the efficiency of the procedure raise.

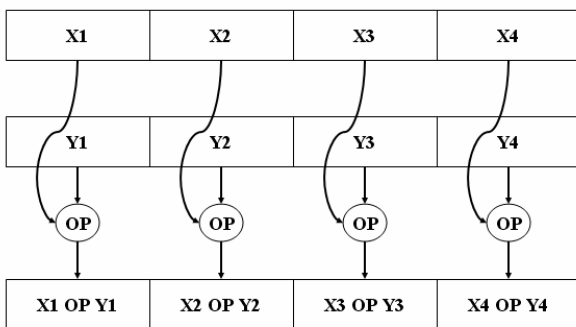


Figure 3. SIMD Model.

There are two instructions of SSE4 for us to improve the Viterbi decoder: 1. The "PMINUSW" instruction. 2. The "PHMINPOSUW" instruction.

1. Using the "PMINUSW" instruction:

In Viterbi decoder, we always set a threshold for the metric value. If the metric exceed the threshold, the metric would be subtracted by the minimum of metric. If we use the unsigned data type to replace the signed data type, the number of times of the condition would become half.

In the past, it is more effective to use "PMINSW" to compare and find the minimum of the metric data. But the instruction limit the data type of metric data to signed word. So we use the "PMINUSW" of SSE4 to replace the "PMINSW". It can be used to compare and find the minimum of metric data which data type is unsigned word. And we can use this instruction to reduce the metric overflow.

2. Using the "PHMINPOSUW":

Another new instruction of SSE4 is "PHMINPOSUW". It is use to find the minimum and the index of the minimum. We use it to find out the decoding path.

The Fig. 4 shows the comparison between using SSE3 and SSE4 instructions. After such improvement, the performance more originally improved by 17.4%.

Function Name	Elapsed Inclusive Time
viterbi_softSSE4(int,short *,short *,	6.67
viterbi_softSSE3(int,short *,short *,	7.83

Figure 4. The comparison between using SSE3 and SSE4 instructions.

V. TO REDUCE THE BRANCH OF THE PROGRAM

When using containing the judgment operations in the procedure (like if, for, while etc.), the execution efficiency of the procedure will drop. After we reduce these operations in the program, the efficiency of the procedures would be improved.

In general, most people like to use for loop in their C program to deal with the continued and similar procedure. There are many for loop in our original Viterbi decoder program. In order to reduce all the loops in our program, we need a large number of C or assembly code to replace. And we write some program to produce those C or assembly code. For example, there is a loop in our original Viterbi decoder (shown in Fig. 5), and we write a program to produce a large C code to replace it (the new procedure is shown in Fig. 6). After such improvement, the performance more originally improved by 86.9% (shown in Fig. 7).

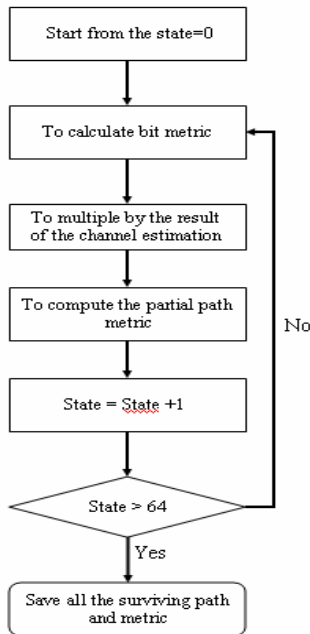


Figure 5. The original loop in our program.

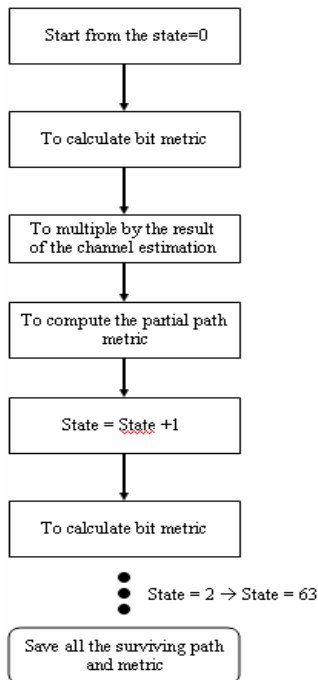


Figure 6. The modified procedure in our program.

Function Name	Elapsed Inclusive Time
viterbi_soft71(int,short *,short *,sh	133.40
viterbi_soft70(int,short *,short *,sh	249.30

Figure 7. The performance comparison result between viterbi_soft70 (before modified) and viterbi_soft71 (after modified).

VI. TO FIND OUT THE MORE EFFICIENT METHOD

In the writing of the program, the same result can often be got through different ways. Therefore, find out an algorithm of performing faster to design program, will speed up the operation of the procedures.

In last section, we reduce the loop but keep the same structure. In fact, there are some other structures can be used to replace the original structures. For example, we find the C code in the last section can be improved by changed the structure. There are only 4 types of bit metric and 2 types of channel estimation results in every 2 coded bits. We can modify the structure by combined these computations (shown in Fig. 8). And we also can use assembly and SSE instructions to implement it. After such improvement, the performance more originally improved by 77.9% (shown in Fig. 9).

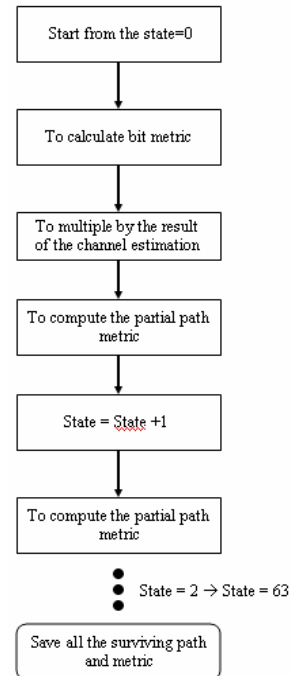


Figure 8. The new procedure in our program.

Function Name	Elapsed Inclusive Time
viterbi_soft73(int,short *,short *,sh	75.15
viterbi_soft71(int,short *,short *,sh	133.74

Figure 9. The performance comparison result between viterbi_soft71 (before modified) and viterbi_soft73 (after modified).

VII. TO REWRITE THE WHOLE VITERBI DECODER PROGRAM WITH ASSEMBLY LANGUAGE

There are sixteen XMM registers in our 64-bit CPU. The x32 compiler only allows us to use eight XMM registers. So we must use x64 compiler to use all XMM registers. And we can use the sixteen to reduce the operation of moving data from registers to memories or moving data from memories to registers. But it doesn't allow us to use inline

assembly by using x64 compiler. We have to rewrite the whole program with assembly to adapt to the x64 compiler.

We use the additional eight XMM registers to hold the metric data to reduce the movements between memories and registers. After such improvement, the performance more originally improved by 21.8% (shown in Fig. 10). And the decoding speed eventually is 47.05 Mbps (261120 bits/5.55 ms=47.05 Mbps).

Function Name	Elapsed Inclusive Time
viterbi_softasm04	5.55

Figure 10. The performance result of our final Viterbi decoder with assembly language.

VIII. CONCLUSION

The most people only use SIMD instructions which the modern CPU provided to accelerate their programs [4]-[7], so the performance improved limitedly. Besides using SIMD, we also utilize some other skills to let the performance be improved greatly. After we continued to do the optimization of our program for several months, the decoding rate has been upgraded from 0.64 Mbps to 47.05 Mbps. The performance is up to 73.5x faster than our original C program. The current decoding rate is more than the useful bit rate of the DVB-T system, so the decoder can decode the DVB-T signal immediately and it only takes 28% CPU loading. In the future, we will carry on the optimization of other parts of our SR for DVB-T until the whole performance of our SR accords with our need.

ACKNOWLEDGMENT

This work was supported in part by National Science Council, Taiwan, under grant NSC 96-2622-E-027-015-CC3.

REFERENCES

- [1] Shu-Ming Tseng, Yao-Teng Hsu, Meng-Chou Chang, and Hsiao-Lung Chan, "A Notebook PC Based Real-Time Software Radio DAB Receiver," *IEICE Transactions on Communications*, vol. E89B, No. 12, Dec. 2006, pp. 3208-3214.
- [2] Nobuaki Kubo, Shunichiro Kondo, Akio Yasuda, "Evaluation of code multipath mitigation using a software GPS receiver," *IEICE Trans. Commun.*, Vol. E88-B, No.11, Nov. 2005, pp. 4204-4211.
- [3] Digital Video Broadcasting (DVB); Fram structure, channel coding and modulation for digital terrestrial television, European Standard (EN) 300 744 V1.5.1, European Telecommunications Standards Institute (ETSI), Nov. 2004.
- [4] S. Kanthak, et al., "Using SIMD instructions for fast likelihood calculation in LVCSR," in *Proc. ICASSP*, vol. 3, June 2000, pp. 1531-1534.
- [5] Y.F. Fung, M.F. Ercan, T.K. Ho, and W.L. Cheung, "Parallel Solution for Railway Power Network Simulation," in *Proc. of PACRIM 2001*, Victoria Canada, Aug. 2001, pp. 413-416.
- [6] A. Servetti, A. Rinotti, and J.C. De Martin, "Fast implementation of the MPEG-4 AAC main and low complexity decoder," in *Proc. ICASSP*, vol. 5, May 2004, pp. 249-252.
- [7] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Performance Comparison of SIMD Implementations of the Discrete Wavelet Transform," *Proc. IEEE Int. Conf. Application-Specific Systems, Architecture Processors (ASAP '05)*, 2005, pp. 393-398.
- [8] Akay, E. and Ayanoglu, E., "High Performance Viterbi Decoder for OFDM Systems," in *Proc. IEEE Vehicular Technology Conf. (VTC)*, 2004, pp. 323-327.
- [9] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol. 20, No. 4, 2000, pp. 47-57.
- [10] "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1," <http://download.intel.com/design/processor/manuals/253665.pdf>
- [11] "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A," <http://download.intel.com/design/processor/manuals/253666.pdf>
- [12] "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B," <http://download.intel.com/design/processor/manuals/253667.pdf>
- [13] "Intel® SSE4 Programming Reference," [http://softwarecommunity.intel.com/isn/Downloads/Intel SSE4 Programming Reference.pdf](http://softwarecommunity.intel.com/isn/Downloads/Intel%20SSE4%20Programming%20Reference.pdf)