Speed Optimization and Application for ATSC Receivers for GNU Radio

Joshua Lilly

joshualilly@me.com

March 12, 2016

Secrete Word/s: Polar Codes are the coolest codes

Introduction

Synopsis

The goal of this project is to optimize the processing of Advanced Television Systems Committee (ATSC) signals in GNU radio's gr-dtv component. Currently, it is possible to process ATSC signals in GNU radio in multiple ways. The first is to create a capture of the signal and, in non-real time, demodulate the signal. The second way is to do real time processing of the signal using GNU radio's USRP "block". The second way is currently possible on high-end servers; however, demodulating on less powerful desktop computers is very cumbersome. This project will focus on optimizing the ATSC signal processing by using multiple parallel Viterbi decoders, which will be implemented in the Vector Optimized Library of Kernels (VOLK), to distribute the load. After these algorithms are implemented, in depth profiling of gr-dtv will be done using perf to find other areas within gr-dtv which are exceptionally computationally intensive. The techniques necessary to effectively make conclusions about the performance of gr-dtv will be documented to allow others to follow the techniques used within gr-dtv elsewhere when profiling GNU radio. These findings will be compared and contrasted with other components from VOLK to further reduce bottlenecks and increase overall performance of gr-dtv using VOLK.

Benefits to the Community

This improvement could result in signification benefits to the community. In addition to the improvements in speed in GNU radio, the VOLK library will also be positively impacted by the addition of Viterbi decoders and demux algorithms. These additions will make it easier for anyone who wants to learn the basics if HDTV signals. HDTV signals will be more accessible to the community as a whole due to the reduced the necessity of the costly professional grade hardware required to decode real-time signals. It will open doors to possibilities of doing much more with less and it will lay the groundwork for future optimizations over all parts of GNU radio. This will also aid people like security researchers who wish to understand the intricacies of real time signal transmission to explore possible exploits of the signals.

Details on ATSC, Viterbi Algorithms, and Viterbi Decoders

ATSC is the set of standards used to define how digital televisions is transmitted over satellite and cable television networks, primarily in the United States and Canada. GNU radio has implemented an ATSC demodulator that allows ATSC signals to be decoded to MPEG files. Currently, due to high demand on hardware resources, this functionality is not capable in real time on standard desktop computers. One possible way around this is to do more work in parallel. This is where the Viterbi algorithm comes into play. The Viterbi algorithm is used in finding the most likely sequence of possible states. This algorithm is commonly used in Hidden Markov Models. The Viterbi algorithm works from statistical techniques, allowing the model to deduce a hidden state given an

observed sequence, providing that it is possible to statistically link transitions between hidden states. This algorithm applies to numerous areas such as speech recognition and, in decoding, Code Division Multiple Access (CDMA) as used by cellular carriers such as Verizon and Sprint. Global Systems for Mobile phones (GSM) is used by AT&T and T-Mobile. However, in this case the Viterbi algorithm will be used to in what is known as a Viterbi decoder. A Viterbi decoder, when implemented in software, uses the Viterbi algorithm to decode a given bit stream. One of the primary areas of implementation for Viterbi decoders is in ATSC decoders. Viterbi decoders play a role in ATSC systems. They sit on the receiving end of a given stream and correct bit errors by retracing the path through the trellis diagram that has the highest probability (Digital video and audio broadcasting). The Viterbi decoders will be used in a Symetric Instruction Multiple Data (SIMD) way of processing, meaning the same operation will be performed on multiple data points at the same time.

Deliverables

- Deliverable 0: A functional and optimized Viterbi decoder implemented within VOLK written in C++.
- Deliverable 1: A functional and optimized demux algorithm within VOLK written in C++.
- Deliverable 2: Documentation on how both algorithms were implemented.
- Deliverable 3: QA tests to test both algorithms written in C++.
- Deliverable 4: Additional example code for gr-dtv to demonstrate how to use and leverage the Viterbi and demux algorithms written in C++, if necessary.
- Deliverable 5: Profile of bottlenecks within gr-dtv using perf compared and contrasted with parts of VOLK that are potential optimizations to the current code base.
- Deliverable 6: Application of Viterbi decoders and other parts of VOLK where possible increases in performance are seen written in C++.
- Deliverable 7: Generalized documentation for how to use perf to find performance heavy areas elsewhere in GNU radio.

Preliminary Planning and Timeline

This summer, I plan on taking a class in artificial intelligence at my George Mason University and also working a part time job in order to pay tuition. These activities will take away some time from coding on this project. However, I am confident that I can still commit at no less than 32 hours per week on this project and during some weeks I believe I can commit the desired 40 hours. Additionally, my family typically takes a vacation during the summer, during which time my contributions would reach around ~3-6 hours during that time. Although this may seem like a lot to bite off, I am highly intrinsically motivated and work hard to meet my goals. In the past, I have worked a full time job while attending school full time, taking between 15-17 credits per semester. Additionally,

when I set my mind on a task, I work tirelessly until I accomplish it. I plan on continuing this work until the deliverables have been achieved, which if necessary could extend past the summer.

Pre-GSoC (Current day – May 12)

- Read through all documentation on relevant parts of GNU radio
- Explore GNU radio code, SDR and make some flow diagrams!
- Build some real time radios using an SDR
- Work on minor bug fixes and other features to get into the code
- Communicate with project mentors about current bugs and things I can already start doing to learn the necessary pieces of VOLK and GNU radio
- Read up on Viterbi decoders, SIMD, and demuxing as much as possible
- Implement some basic programs utilizing basic SIMD, and Viterbi decoders

Week 1 (May 9 – May 15)

- Begin profiling gr-dtv to find bottle necks
- Begin basic documentation for profiling gr-dtv
- Work with VOLK and begin writing demux algorithm in VOLK
    - Specifically the work on the demuxing algorithm will involve devising how data will flow into the demuxing algorithm and specifically designing the structure and interface of a demuxing class or namespace or finding pieces in VOLK that already would suffice to house the necessary functions.

Week 2 (May 16 – May 22)

- Continue working on profiling and adding to documentation on profiling
- Continue working in VOLK to get a working demux algorithm
- Start writing basic unit tests to check correctness of the portions of the demuxing algorithm that are in place.
- Publish all tests and code to obtain community feedback on the implementation thus far

Week 3 (May 27 – June 3)

- Finish profiling of gr-dtv and publish findings to the community to obtain feedback of findings
- Publish a detailed, and, as generalized as possible, list of the processes taken to profile gr-dtv in order to identify bottlenecks which will make for easy replication elsewhere in the GNU radio.
- Continue work on the code for the demuxing algorithm and continue writing tests to verify correctness of code written thus far.

Week 4 (May 23 – May 29)

- Continue working on the demuxing algorithm and writing tests to verify correctness
- Begin the design of the Viterbi decoders by introducing the classes or namespaces needed to handle the Viterbi decoder algorithm structures in VOLK

- Write “stub” Viterbi decoders that will start handling the demuxed streams and which will eventually be used to Segway into writing the Viterbi decoders.

Week 5 (May 30 – June 5)

- Finish the demuxing algorithm and unit tests.
- Publish the demuxing algorithm to the community and obtain feedback from the community as well as mentors.
- Start working on implementation of the Viterbi algorithm in VOLK, which will be the backbone of the decoder. To begin, the branch metric unit portion of the Viterbi decoder will be started. This will utilize the bit streams produced from the demux algorithm and basic calculations of the distance between ideal streams and the actual streams will be calculated.

Week 6 (June 6 – June 12)

- Respond to the feedback on the demux algorithms and continue working to finalize and optimize the code.
- Work out algorithms for determining the distances between bit streams and implement code to handle distance calculations in the incoming, now demuxed, bit streams.
- Work out the interface between the branch metric unit portion of the Viterbi decoder and the next portion of the decoder, which will be the add compare select unit—a two stage approach will be used.
- Write unit tests to prove correctness of the Viterbi decoder thus far.

Week 7 (June 13 – June 19)

- Plan add compare select recursion and begin implementation of the necessary logic and classes
- Add unit tests to verify correctness as code is produced.

Week 8 (June 20 – June 26)

- Continue working out the details of the of the add compare select unit and implementing code for it.
- Continue writing tests to verify produced code.
- Submit code to the community for opinions.
- Begin working out the details and interface of the third stage of the Viterbi decoder, which will be the decoding of the survivor path.

Week 9 (June 27 – July 3)

- Continue working on the Viterbi decoder in VOLK and producing test to verify correctness

Week 10 (July 4 – July 10)

- Polish up all stages of the Viterbi decoder and unit tests
- Work on optimization of the Viterbi decoder and publish code to the community for feedback.

Week 11 (July 11 – July 17)
- Finish the Viterbi decoder and tests.

Week 12 (July 18 – July 24)

**** This will be a built in catch up week in case of vacation, exams, or feature slip, specifically with the Viterbi decoder.**

- Providing no slips this will be a week where the gr-dtv code is revisited and evaluated based on the profiling done in the beginning to find places to begin apply the new Viterbi decoder.

Week 13 (July 25 – July 31)
- Begin implementation of the Viterbi decoder in gr-dtv using the metrics obtained in the first weeks
- Write unit tests to verify code is still correct in gr-dtv
- Locate positions where numerous Viterbi decoders implemented with SIMD would speed up computation and being implementing them

Week 14 (August 1 – August 7)
- Continue implementing the Viterbi decoders utilizing SIMD, the demuxing algorithm, and the Viterbi decoder.
- Profile speed ups in gr-dtv

Week 15 (August 8 – August 15)
- Continue implementing code to increase performance in gr-dtv
- Profile code in gr-dtv and compare and contrast results.
- Publish speed increases to the community
- Finish up documentation
- Use the applicable unit test written throughout the summer as QA test and the remaining test as example usage of the new algorithms.

And beyond…
- Continue working on finding ways to improve optimality of GNU radio using VOLK.
- Tie up loose ends from the summer.

My Background

Name: Joshua Lilly
Residence: Fairfax, VA
University/Status: George Mason University, 3rd year undergraduate.
Google Hangouts name: joshualilly91
Current coding projects: https://github.com/joshu0991/Robot_2.0

I currently attend George Mason University, where I study Computer Science. I have an interest in artificial intelligence, computer networks, RF, and robotics. I have worked on building a few different robots using the Raspberry Pi, which use sensors to gather data on surroundings. I have also written a basic speech-to-text and text-to-speech application using CMUSphinx, which has allowed me to gain basic working knowledge of Viterbi algorithms and Hidden Markov models. I have basic knowledge of DSP, and SIMD programming and about 3 years of experience programming in C++. My C++ knowledge includes a working knowledge of Boost, including libraries like Spirit. I also have basic working knowledge of MPL, cmake, protobufs, gtest, TBB, and CUDA. Overall, I have been programming for about 6 years. During this time I have learned multiple languages, including Java, C/C++, Groovy, Javascript, and Python. This is the first open source project that I will be contributing to; however, I do have basic knowledge of how to work on a team due to my experience working with a team of developers at my internship.

Conclusion

My plan for implementation of the necessary algorithms allows for a reasonable amount of work to be done for a newcomer to open source projects over a given amount of time. During the allotted time for the Summer of Code, I hope to help GNU radio increase its performance to allow for everyday community members to have the capability to demodulate ATSC signals in real-time. This will allow for a more streamlined experience when researching ATSC signals and bring forth new capabilities to the VOLK library.

Sources

15, Marc Flores October, and 2013 Mobile phones. N.d.

Verizon CDMA Explained. TechRadar. http://www.techradar.com/us/news/phone-and-communications/mobile-phones/verizon-cdma-explained-1189551, accessed March 14, 2016.

A Practical Guide to SSE SIMD with C++. N.d. http://sci.tuomastonteri.fi/programming/sse, accessed March 15, 2016.

ATSC Standards. 2016

Wikipedia, the Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=ATSC_standards&oldid=709535172, accessed March 14, 2016.

Center for History and New Media. N.d.

Digital Video and Audio Broadcasting Technology: A Practical Engineering Guide. Springer Science & Business Media.

Gnuradio/volk. N.d.

GitHub. https://github.com/gnuradio/volk, accessed March 14, 2016.

Microsoft PowerPoint - lecture_14 - lecture_14.pdf. N.d.

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-973-communication-system-design-spring-2006/lecture-notes/lecture_14.pdf, accessed March 15, 2016.

SIMD 2016.

Wikipedia, the Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=SIMD&oldid=703240939, accessed March 14, 2016.

Vector Optimized Library of Kernels. N.d. http://libvolk.org/, accessed March 14, 2016.

Viterbi Algorithm. 2016.

Wikipedia, the Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Viterbi_algorithm&oldid=708223911, accessed March 14, 2016.

Wikipedia, the Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Viterbi_decoder&oldid=701073046, accessed March 14, 2016.