Profiling of GNU Radio DVB-S2X transmitter using multi-core CPU and hardware accelerators

Haris Turkmanović
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
haris@etf.bg.ac.rs

Dragomir El Mezeni
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
elmezeni@etf.bg.ac.rs

Vladimir L. Petrović
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
petrovicv@etf.bg.ac.rs

Lazar Saranovac
School of Electrical Engineering
University of Belgrade
Belgrade, Serbia
laza@etf.bg.ac.rs

***Abstract*—Software defined radio (SDR) brought flexibility and easier development to the design of telecommunication systems. However, achieving real-time performance with SDR using general purpose processors (GPP) is still a challenging topic. We have examined performance of SDR DVB-S2X transmitter implemented in GNU Radio framework using a multi-core processor with 32 cores. We have found that GNU Radio framework cannot fully utilize this processor since overheads of parallelization become bottleneck. When FEC blocks are offloaded to hardware accelerator, transmitter achieved the largest throughput for just 8 CPU cores. This means that there exists optimal number of processing cores for specific SDR architecture. Maximal achieved throughput for accelerated DVB-S2X transmitter was 3.4 Gbps. Also, accelerated architecture provides throughput of 1.8 Gbps for 4 CPU cores which is higher than 1.3 Gbps achieved for 32 CPU cores and full software implementation.**

***Keywords*—CPU utilization, DVB-S2X, GNU radio, hardware acceleration, software-defined radio**

I. Introduction

Software defined radio (SDR) introduced unprecedented flexibility in the design of modern telecommunication systems. It is just recently that real-time signal processing became feasible by using only general purpose processors (GPP). Multiple CPU cores, vector instructions and other means of acceleration are necessary to achieve the desired performance. However, in radio applications with strict requirements for latency, throughput, and power consumption, the SDR approach is still challenging. This means that some kind of trade-off between hardware-like performance and software-like flexibility should be achieved. Usual approach for commercial applications is to use custom optimized SDR solution such as Open Air interface for 5G NR and LTE [1]. There are also proposals for domain specific languages targeting SDR applications that can better utilize available hardware resources [2].

The research presented in this paper has been using GNU Radio. Since increased flexibility usually induces performance degradations, it is challenging to achieve real-time performance with GNU Radio on GPP. Nevertheless, there are many advantages in using such general framework when developing and researching new applications, ranging from community support to easier debugging and reconfiguration of existing designs.

There are dedicated benchmarks developed to enable easier selection of appropriate processor that can achieve the best GNU Radio performance. Another approach is to optimize the processing itself to better use available hardware resources. The question we are trying to answer in this paper is: "*How can we accelerate GNU Radio processing on multi-core GPP and what benefits can hardware acceleration bring?*".

The first step in answering the previous question is analysis of GNU Radio framework and overheads it introduces. In GNU Radio framework each block is implemented in the separate processing thread. Dedicated scheduler is responsible for deploying these threads to the actual processing cores. However, even though the work is split to multiple threads, the parallelization is usually not that much efficient as synchronization is needed between different processing blocks. Bloessl et al [3] analyzed GNU Radio throughput for a large number of simple processing blocks. They concluded that throughput scales linearly with increasing the number of blocks. However, they also found that significant improvements can be achieved by manual processing organization, thus avoiding framework synchronization and scheduling mechanisms. Becker et al [4] used GNU Radio for real-time wireless signal classification. They found that by using alternative synchronization mechanisms, based on Qt Signals and Slots, performance improvement of up to 78× can be achieved. They also used vector instructions to accelerate separate processing blocks.

In order to efficiently use multiple processing cores, sufficient level of parallelism should be exposed. Recently, Grayver et al [5] presented DVB-S2 demodulator with 4GHz bandwidth using multiple GPP servers. To efficiently use multiple-cores they used multiple data-path processing chains to extract parallelism. Miller also used this way of processing parallelization to develop HDR QPSK modem using GNU Radio [6]. Cassange et al [2] demonstrated usage of a custom domain specific language with dedicated synchronization and parallelization mechanisms. They demonstrated several times better DVB-S2 receiver throughput when compared to the gr-dvbs2rx GNU Radio solution.

In this paper we demonstrate mechanisms how to achieve the HDR performance on a multi-core processor on the GNU Radio DVB-S2X transmitter use case, and analyze the impact of dedicated hardware accelerators on throughput

Digital Object Identifier: 10.1109/TELFOR56187.2022.9983695; Final, published article available at: https://ieeexplore.ieee.org/document/9983695

and CPU utilization. Section 2 gives a brief overview of GNU radio framework and its synchronization and scheduling mechanisms. Section 3 describes implementation of accelerated DVB-S2X transmitter architecture. In section 4 we evaluate performance of fully software and hardware accelerated architectures. Conclusion is given in the section 5.

II. GNU Radio framework

GNU Radio represents a framework that can be used for development of different SDR applications. Beside library of already designed DSP blocks with standardized interfaces framework also supports development of custom DSP blocks. These blocks are called out of tree (OOT) modules and can be used to implement new functionalities, which increases the framework flexibility.

Signal processing blocks chain in GNU Radio is called “flowgraph”. Each flowgraph contains at least one source and one sink block. Data flows from source to sink through all DSP blocks in the chain. Every DSP block has input and output buffers and is implemented in a separate processing thread. GNU Radio uses operating system scheduler for managing thread executions. By using *affinity* parameter of DSP block, thread can be statically scheduled to the specific CPU core. The thread is scheduled for execution if there is enough data in the input and enough space in the output buffer. Otherwise, processing is stalled and neighboring blocks are notified via message passing mechanism. Hence, each block in a chain waits for previous block to fill its input buffer in order to start processing. Consequently, the entire chain throughput is determined by the slowest block.

Since each block is implemented in a separate thread, flowgraph can be scheduled on a multi-core processor. On the other hand, such low granularity can lead to frequent context switching and increase the processing overhead. One recommendation for optimization is to combine several functionalities in one larger block [7]. Efficient use of multiple cores assumes that sufficient number of threads can be executed independently.

III. Acceleration of DVB-S2X transmitter

A. *DVB-S2X transmitter GNU radio chain*

Flowgraph for DVB-S2X transmitter is shown in Fig. 1. The transmitter is implemented using GNU Radio framework v3.8 on Ubuntu 20.04 using AMD Ryzen 9 5950X with 32 processing cores. Maximal throughput that can be achieved for a single transmitter chain is 131 Mbps with processor utilization of just 160%. This is equivalent to 2 CPU cores, which means that GNU Radio scheduler cannot efficiently utilize multi-core processor when single serial flowgraph is used. The reason for this behavior is that many simple tasks will be blocked while waiting for more complex tasks to be completed. In the DVB-S2X transmitter case, these complex tasks are forward error correction (FEC) encoders, LDPC and BCH [8]. They take almost 70% of total processor utilization. Also, the flowgraph from Fig. 1 is relatively small, having only 6 processing blocks, and scheduler is not able to efficiently use the remaining CPU cores.

B. *Parallel architecture of DVB-S2X transmitter*

In order to increase the CPU utilization and throughput several parallel processing branches can be used [5]. Each parallel branch is processing a different codeword. *Interleave* and *deinterleave* blocks from GNU Radio library are used for transferring codewords from serial stream at the input to parallel branches and to combine results into a single serial stream at the output. Parallel architecture of DVB-S2X transmitter for the case of four branches is presented in Fig. 2.

Since blocks from different branches are working independently, they can be scheduled on different processing cores more efficiently. The goal of the first experiment in this study was to find an optimal number of parallel branches for DVB-S2X transmitter in the case when different numbers of CPU cores were available.

C. *Accelerated DVB-S2X transmitter*

Even though the throughput can be significantly increased by using parallel branches, the most complex blocks like FEC encoders still represent a bottleneck for each branch. To overcome this, the transmitter can be

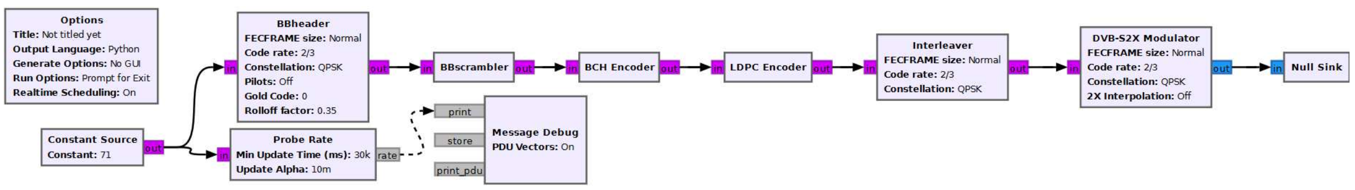


Fig. 1. DVB-S2X transmitter flowgraph

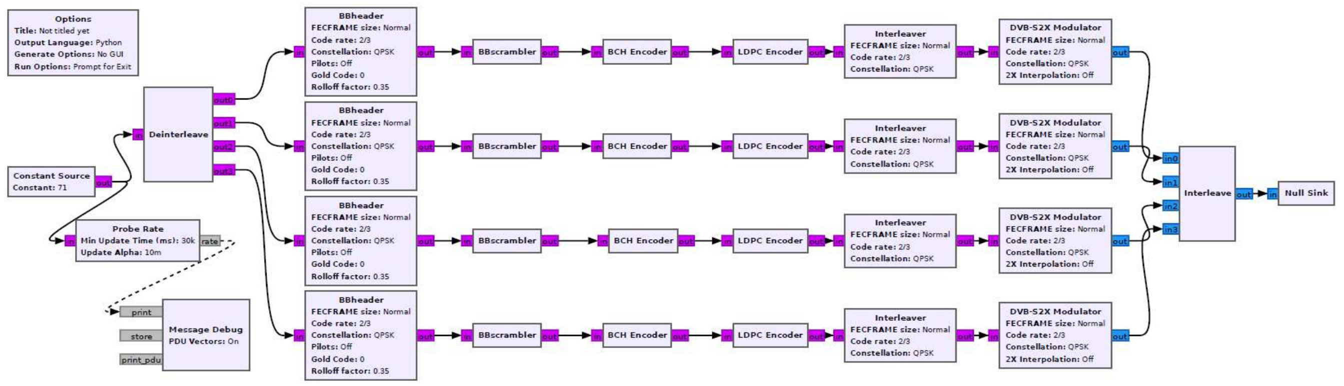
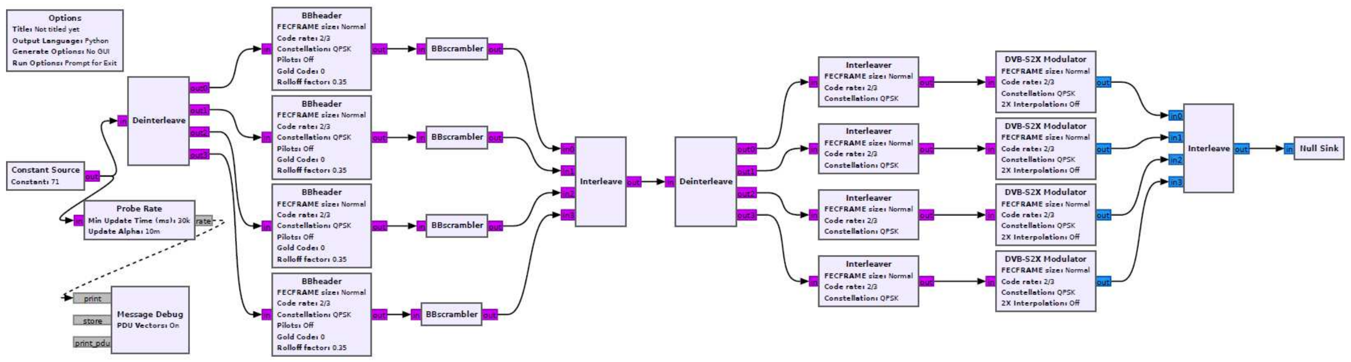


Fig. 2. Parallel DVB-S2X transmitter flowgraph


Options
Title: Not titled yet
Output Language: Python
Generate Options: No GUI
Run Options: Prompt for Exit

Constant Source
Constant: 71

Probe Rate
Min Update Time (ms): 30k
Update Alpha: 10m

Message Debug
PDU Vectors: On

BBheader
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
Pilots: Off
Gold Code: 0
Rolloff factor: 0.35

BBheader
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
Pilots: Off
Gold Code: 0
Rolloff factor: 0.35

BBheader
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
Pilots: Off
Gold Code: 0
Rolloff factor: 0.35

BBheader
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
Pilots: Off
Gold Code: 0
Rolloff factor: 0.35

Interleaver
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK

Interleaver
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK

Interleaver
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK

Interleaver
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK

DVB-S2X Modulator
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
2X Interpolation: Off

DVB-S2X Modulator
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
2X Interpolation: Off

DVB-S2X Modulator
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
2X Interpolation: Off

DVB-S2X Modulator
FECFRAME size: Normal
Code rate: 2/3
Constellation: QPSK
2X Interpolation: Off

Fig. 3. DVB-S2X transmitter with emulation of accelerated FEC blocks

further accelerated by offloading these blocks to dedicated hardware accelerators, eg. PCIe FPGA cards or GPU cards. If accelerator is fast enough, only one can be used to process data from all parallel branches. *Interleave* and *detinterlevae* blocks again can be used between parallel and serial stream conversions. These components also emulate data copying to and from the accelerator. If accelerator processes stream of data with the same throughput as the rest of software chain, then it can be modeled as a simple connection between *interleave* and *deinterleave* components. Architecture of DVB-S2X transmitter with accelerated FEC blocks is shown in Fig. 3.

By accelerating the most complex blocks, the rest of the flowgraph becomes more balanced and can be efficiently scheduled. The goal of the second experiment was to find the number of CPU cores needed to achieve the maximal throughput of the transmitter with FEC acceleration.

IV. Results and Duiscussion

In order to evaluate performance of the examined architectures CPU utilization and achieved transmitter throughput were measured. CPU utilization was measured by using *htop* tool and it is expressed in percent ranging from 0 to 3200 in case of 32 cores. Number of cores available to GNU Radio application can be limited by using *taskset* command in Linux. Throughput is measured directly from GNU Radio by using Probe Rate block. Running average gain was set to 0.01 and each measurement was taken by averaging results obtained from different runs of GNU Radio applications. This way the influence of different states of a cache and different states of operating system can be reduced. GNU Radio scheduler was set to Normal priority, and VOLK acceleration is not used.

Throughput and CPU utilization for parallel transmitter architectures with different number of parallel branches, with different number of available CPU cores, are shown in Fig. 4 and Fig. 5 respectively.

Maximal throughput for parallel DVB-S2X transmitter was 1.3 Gbps and it is achieved for 20 parallel branches in case when all 32 processing cores were available. However, slightly lower throughput of 1.2 Gbps can be achieved for the same number of parallel branches by using just 16 processor cores. Hence, for this architecture it is worth considering the usage of simpler processor or freeing processor resources for some other processing tasks.

Increasing number of parallel branches exposes more parallelism and enables higher throughput. However, when processing load exceeds available resources CPU utilization enters saturation. In the case of 4 CPU cores, utilization

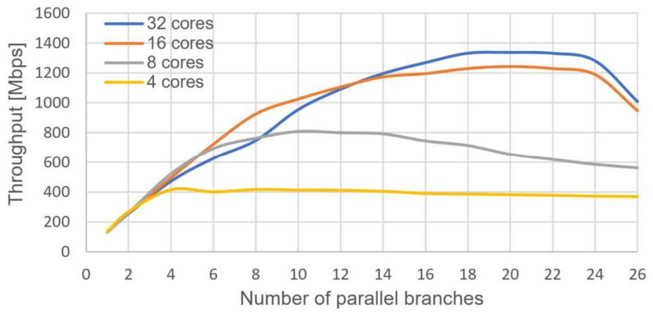


Fig. 4.Throughput for parallel DVB-S2X transmitter

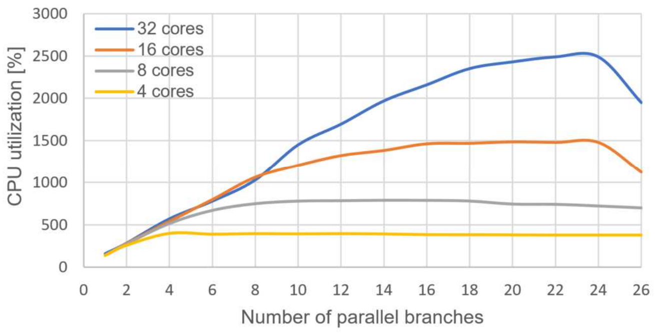


Fig. 5. CPU utilization for parallel DVB-S2X transmitter

reaches 400% for 4 parallel branches and throughput increase stops. When there are 8 CPU cores available, CPU utilization reaches 785% for 10 parallel branches and stays in this high utilization region. For 16 and 32 CPU cores, CPU utilization saturates before CPU reaching maximal values of 1600% and 3200% and even starts to drop when number of parallel branches is further increased. The reason for this behavior is a new bottleneck introduced in parallel architectures. Namely, although there are parallel branches enabling larger throughput, *Interleave* and *deinterleave* components still operate in serial manner. Their load increases with throughput and they become bottlenecks when more than 24 parallel branches are used. *Interleave* and *deinterleave* components can be implemented as simple data switches, and thus their complexity should be minimal. In software they just need to forward data pointers for current codeword to the appropriate processing branch. However, in GNU Radio, these blocks are copying all data elements from memory to component buffers. Copy operation takes significant amount of time especially in case with large number of parallel branches. Hence, implementing OOT module that work with pointers instead of using library components can potentially increase throughput.

For smaller number of parallel branches lower number of processing cores can provide higher throughput. For example, in the case of 6 parallel branches, throughput of 692 Mbps can be achieved with 8 processing cores while it drops to 626 Mbps when all 32 cores are used. This can be explained by inefficiency of GNU Radio scheduler when number of cores largely exceeds processing load. In this case, scheduler will more frequently switch processing tasks from one core to another, introducing additional overhead and thus decreasing the throughput. This can be confirmed by statically scheduling processing blocks using *affinity* parameter of GNU Radio blocks. In this case increasing number of cores will not decrease the throughput. However, throughput achieved with static scheduling is lower than by dynamic scheduling mechanism.

Throughput and CPU utilization for accelerated DVB-S2X transmitter architecture are shown in Fig. 6 and Fig. 7 respectively.

Maximal throughput for accelerated DVB-S2X transmitter was 3.4 Gbps and it is achieved for 8 parallel branches in a case when just 8 processing cores were used. Furthermore, the achieved throughput for 8 cores is larger than for any other number of cores for all configurations of parallel branches. This result is very interesting and shows that there exists optimal architecture for a specific flowgraph. In accelerated DVB-S2X transmitter there are 4 simple blocks in parallel branches and 4 serial blocks (*interleave* and *deinterleave*). This architecture can optimally fit 8 cores when serial tasks are implemented in the separate cores while parallel blocks share the remaining 4 cores. Since number of cores is limited there will be less thread and context switching. By examining CPU utilization, we can notice that in the case of 8 cores, it saturates to lower level than in the case of DVB-S2X transmitter without acceleration.

When only 4 CPU cores are available, the utilization reaches 379% for 4 parallel branches and maximal achieved throughput is 1.8 Gbps. This is still higher than maximal throughput achieved for all-software transmitter with 32 cores. For 16 CPU cores maximal achieved throughput is 2.9 Gbps for 6 parallel branches, while throughput of 2.2 Gbps is achieved for 8 parallel branches and 32 CPU cores. Further increase of parallelism leads to lower throughput since *interleave* and *deinterleave* components become bottlenecks.

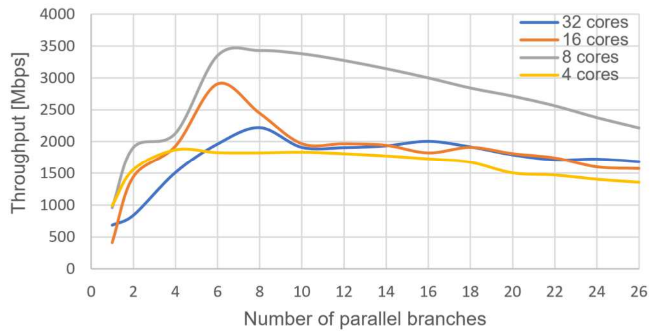


Fig. 6. Throughput for accelerated DVB-S2X transmitter

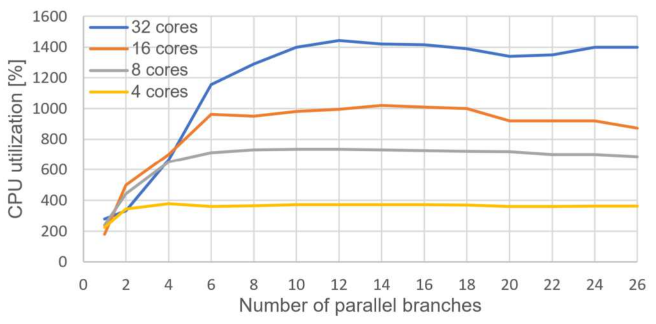


Fig. 7. CPU utilization for accelerated DVB-S2X transmitter

V. Conclusion

In this paper we examined the influence of varying number of parallel branches and number of available CPU cores to the performance of DVB-S2X transmitter with and without hardware acceleration.

We found that in case of fully software implementation, transmitter achieves throughput of 1.2 Gbps for 16 CPU cores and 20 parallel branches. This is just a slight penalty when compared to the case when all 32 cores were used. Hence, there is a limit of number of parallel branches and processing cores for this architecture.

When FEC modules are accelerated in hardware, aforementioned effects are much stronger since the remaining flowgraph is more balanced. Maximal throughput of 3.4 Gbps is achieved for just 8 parallel branches and 8 processing cores. Accelerated architecture provides larger throughput for 4 cores than software architecture with all 32 cores.

Accelerating DVB-S2X transmitter by implementing FEC encoders in hardware can significantly increase a throughput and enable usage of less powerful processors which are usually available in SoC solutions.

Acknowledgment

The authors acknowledge the support of the Science Fund of the Republic of Serbia, grant No 7750284, Hybrid Integrated Satellite and Terrestrial Access Network - hi-STAR.

References

[1] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet. "OpenAirInterface: A flexible platform for 5G research." ACM SIGCOMM Computer Communication Review 44, no. 5, 2014, pp. 33-38.

[2] A. Cassagne, R. Tajan, O. Aumage, C. Leroux, D. Barthou and C. Jégo. "A DSEL for High Throughput and Low Latency Software-Defined Radio on Multicore CPUs." arXiv preprint arXiv:2206.06147 2022.

[3] B. Bloessl, M. Müller and M. Hollick. "Benchmarking and Profiling the GNURadio Scheduler." In Proceedings of the GNU Radio Conference, vol. 4, no. 1. 2019.

[4] C. Becker, A. Baset, S. Kasera, K. Derr and S. Ramirez, "Experiences with using GNU Radio for real-time wireless signal classification." In Proceedings of the GNU Radio Conference, vol. 3, no. 1. 2018.

[5] E. Grayver and A. Utter. "Extreme Software Defined Radio–GHz in Real Time." In 2020 IEEE Aerospace Conference, 2020, pp. 1-10.

[6] D. Miller, "Demonstration of GNU Radio High Data Rate QPSK Modem at 15.0 Mbps Real-Time with Multi-Core General Purpose Processor", In Proceedings of the GNU Radio Conference, 2022.

[7] T. W. Rondeau, O. Holland, H. Bogucka, and A. Medeisis. "On the GNU radio ecosystem." Opportunistic Spectrum Sharing and White Space Access: The Practical Reality, 2015, pp. 25-48.

[8] ETSI, "Digital Video Broadcasting (DVB)", https://www.dvb.org/standards/dvb-s2