Pushing the Limits of Energy Efficiency for Non-Binary LDPC Decoders on GPUs and FPGAs

Srinivasan Subramaniyan, Oscar Ferraz*, Ashuthosh M R‡, Santosh Krishna‡, Guohui Wang†, Joseph R. Cavallaro†, Vitor Silva*, Gabriel Falcao* and Madhura Purnaprajna
Amrita Vishwa Vidyapeetham, Department of Computer Science and Engineering, Bangalore, India
University of Coimbra, Instituto de Telecomunicações, Department of Electrical and Computer Engineering, Portugal
‡PES Institute of Technology , Department of Electronics and Communication Engineering , Bangalore South Campus, India
†Rice University, Department of Electrical and Computer Engineering, US
{srinivasansubramaniam74, ashuthoshmrofficial, santhosh.bk97, robertwgh}@gmail.com {oscar.ferraz, vitor, gff}@co.it.pt, cavallar@rice.edu, p_madhura@blr.amrita.edu

***Abstract*—Signal processing hardware designers of Low-Density Parity-Check (LDPC) decoders used in modern optical communications are confronted with the need to perform multi-parametric design space exploration, targeting very high throughput (hundreds of Mbit/s) and low-power systems. This work addresses the needs of current designers of dedicated GF(2^m) NB-LDPC decoders that necessitate robust approaches for dealing with the ever-increasing demand for higher BER performance. The constraints pose tremendous pressure on the on-chip design of irregular data structures and micro-circuit implementation for supporting the complex Galois field mathematics and communications of hundreds of check nodes with hundreds of variable node processors. We have developed kernels targeting GPU and FPGA (HLS and its equivalent RTL) descriptions of this class of complex circuits for comparing area, frequency of operation, latency, parallelism and throughput. Exploiting techniques such as using custom bit-widths, pipelining, loop-unrolling, array-partitioning and the replication of compute units, results in considerably faster design cycles and demands less non-recurring engineering effort. We report a throughput performance of 800 Mbps for the FPGA case.**

***Index Terms*—design space exploration; roofline model; high-throughput; parallelism; low-power; FPGAs; GPUs; RTL; HLS; NB-LDPC decoding.**

I. Introduction

Next-generation optical communications will demand bit error rates (BER) in the range of 10^{-15} for long-distance communications (hundreds to thousands of kilometres) for optical fibre submarine cable systems or between base stations (e.g., ITU-T G.975). Here two distinct worlds have to coexist, viz., 1) coding theorists that propose and test new codes, in particular exploiting the powerful low-density parity-check (LDPC) codes [1], and 2) hardware system developers who deploy them in normal operating modes, achieving high throughput [2] under limited power and BER constraints.

Compared to binary codes [3], non-binary (NB) LDPC codes allow higher coding performances and lower BER. Moreover, for achieving such BER performance, code block lengths can be smaller as compared with the binary counterpart [3]. For this class of non-binary algorithms, the challenges are: *i)* hardware and computational complexity increase as they operate on the Galois field domain; *ii)* operations are based on intensive and irregular accesses to memory; *iii)* the standards demand low latency. In order to achieve high-throughput and low-power performance, both key for success, design space exploration has to be done quickly and efficiently. This paper deeply analyses the roofline models of all optimisation options in order to quantify how close they are from theoretical peak performance for GPU and FPGA devices. Moreover, the paper addresses the optimized use of HLS and RTL descriptions on reconfigurable FPGA technology for simultaneously achieving high throughput performance.

We have made the following **contributions**:

- High-throughput NB-LDPC decoder implementations on GPU and FPGA;
- **HLS-based design space exploration** for dealing with the high number of **degrees of freedom** that are a characteristic of these algorithms;
- Custom **RTL** design to compare the performance of these algorithms with HLS solutions;
- **Roofline model** based comparison of CPU / GPU / FPGA platforms for throughput and energy efficiency.

II. Min-Max Algorithm

The Min-Max algorithm described by V. Savin [3], proposes an approach to NB-LDPC decoding. By using proper metrics, this decoding algorithm is capable of achieving low-complexity and quasi-optimal performance. Binary codes are rather easy to express in hardware in bit-level logic. On the other hand, non-binary codes are harder to solve, becoming a multi-valued logic search space for finding the minimum probability symbol.

The Tanner graph represents the connections between checks nodes and variable nodes. Mathematically, the Tanner graph is represented by **H**. Given the probabilities ($\gamma_n(a)$), the variable node message ($\alpha_{m,n}(a)$) is initialised. The Row-wise check node processing (CNP) starts by calculating the forward and backward metrics ($F(a)$, $B(a)$) and finishes in the check node message computation ($\beta_{m,n}(a)$). The metrics are initialised by attributing the first element of $\alpha_{m,n}(a)$ on

978-1-7281-8099-1/20/$31.00 ©2020 IEEE

Algorithm 1: Min-Max decoding algorithm

{**Initialisation:**}
$\gamma_n(a) = \frac{ln(Pr(c_n=s_n|channel))}{ln(Pr(c_n=a|channel)}$
$\alpha_{m,n}(a) = \gamma_n(a)$
while ($\mathbf{c_n}\,\mathbf{H}^T \neq \mathbf{0} \wedge$ it $<$ I) {c_n-decoded word; I-Max. no. of iterations.}
do
  {**Check node processing:**}
  $F_0(a) = \alpha_{m,n_0}(h^{-1}_{m,n_0}a)$
  $F_i(a) = \min_{a'+h_{m,n_i}a''=a}(max(F_{i-1}(a'), \alpha_{m,n_i}(a'')))$
  $B_{dc-1}(a) = \alpha_{m,n_{(dc-1)}}(h^{-1}_{m,n_{(dc-1)}}a)$
  $B_i(a) = \min_{a'+h_{m,n_i}a''=a}(max(B_{i+1}(a'), \alpha_{m,n_i}(a'')))$
  $\beta_{m,n_0}(a) = B_1(h_{m,n_0}a)$
  $\beta_{m,n_{(dc-1)}}(a) = F_{(dc-2)}(h_{m,n_{dc-1}}a)$
  $\beta_{m,n_i}(a) = \min_{a'+a''=a}(max(F_{i-1}(h_{m,n_{i-1}}a'), B_{i+1}(h_{m,n_{i+1}}a')))$
  {**Variable node processing:**}
  $\alpha'_{m,n}(a) = \gamma_n(a) + \sum_{m'\in M(n)\setminus\{m\}} \beta_{m',n}(a)$
  $\alpha'_{m,n} = \min_{a\in GF(q)} \alpha'_{m,n}(a)$
  $\alpha_{m,n}(a) = \alpha'_{m,n}(a) - \alpha'_{m,n}$
  {**Tentative decoding:**}
  $\widetilde{\gamma}_n(a) = \gamma_n(a) + \sum_{m\in M(n)} \beta_{m,n}(a)$
  $c_n = arg \min_{a\in GF(q)}(\widetilde{\gamma}_n(a))$
end while

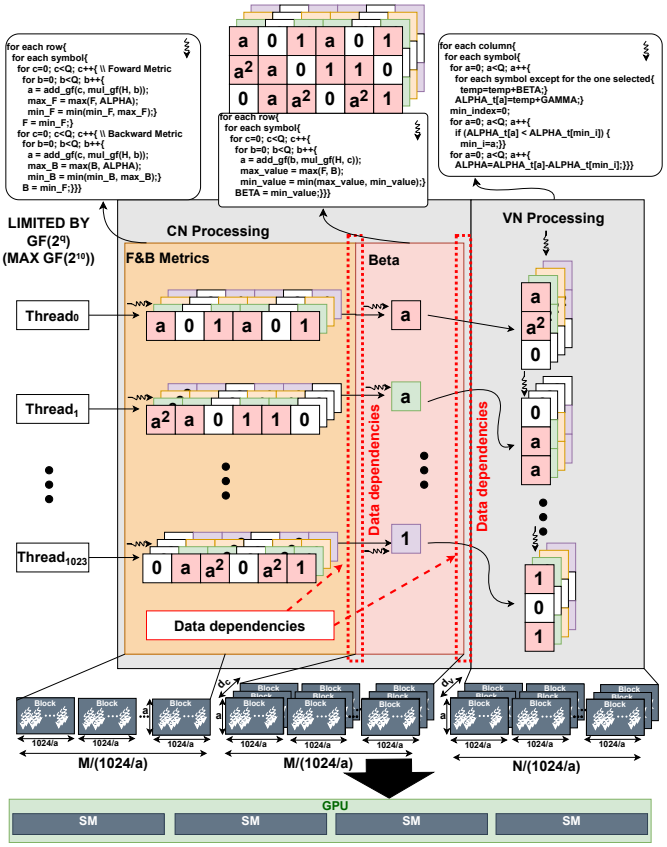

Fig. 1: Min-Max algorithm mapping to a GPU architecture. Forward and backward metrics are row-parallel while VNP is column-parallel. The beta kernel is symbol-parallel.

each row to $F(a)$ and last element to $B(a)$. The next elements of each metric are calculated using the previous ones, from the initial to final element in $F(a)$, and backwards on the $B(a)$ metric. To initialise $\beta_{m,n}(a)$, the first element of $B(a)$ and last of $F(a)$ are attributed to the initial and final elements of $\beta_{m,n}(a)$. The remaining elements calculate the minimum of the maximum for the different symbols on the Galois field.

The variable node processing (VNP) is calculated column-wise and uses $\gamma_n(a)$ and $\beta_{m,n}(a)$ to compute a temporary variable node message ($\alpha'_{m,n}(a)$). From this variable and for each element in *GF(q)*, it is determined the most likely symbol (min operation). For the algorithm to be computationally stable, the most likely symbol is subtracted from $\alpha'_{m,n}(a)$, producing the variable node message ($\alpha_{m,n}(a)$) to be used in the next iteration of the check node processing. In the tentative decoding, if a codeword satisfies the check equation ($\mathbf{c}_n^T\mathbf{H} = \mathbf{0}$), the decoding process successfully terminates. Otherwise, the CNP and VNP are executed until they verify a valid check equation or a maximum number of iterations is reached. As the order of the field increases, the number of symbols increments by powers of 2, causing an exponential growth of the used memory easily occupying several GigaBytes of memory, turning the system infeasible for higher Galois fields. The use of CSR (compressed sparse row) and CSC (compressed sparse column) data formats allows compressing the **H** by up to 95%. FPGAs can use less memory than GPUs since they allow for custom bit lengths to be used in contrast with GPUs, which have a granularity of bytes.

III. Mapping Parallelism on Embedded Devices

The input probabilities (log-likelihood ratios [3]) received from the channel were converted from double precision to integer, reducing the computational complexity, and also the pressure on the memory subsystem. The FPGA implementations use 5-bit for data representation. The GPU does not allow single-bit manipulation and uses 8-bit for representing probabilities. Reducing precision does not affect error calculation since computations use Galois field arithmetic.

A. GPU: Multi-threading and fast local memory access

The mapping of the Min-Max decoding algorithm on the GPU is performed used 3 parallel kernels: the $F(a)$ and $B(a)$ metrics kernel, the CNP kernel and the VNP kernel as shown in Fig 1. The figure represents the pseudo-code for each thread executing in the kernel, as well as an example of a NB-LDPC **H** matrix in the top of the figure.

The main constraints to parallelisation of the algorithm are data dependencies present between the forward and backward metrics and the beta kernels and CNP and VNP. In order to maximise the occupancy rate, each block is configured to run 1024 threads with dimension of $\frac{1024}{a}$ by a, being a the size of the Galois field. The $F(a)$ and $B(a)$ metrics kernel, represented from line 6 to 9 of Algorithm 1, executes

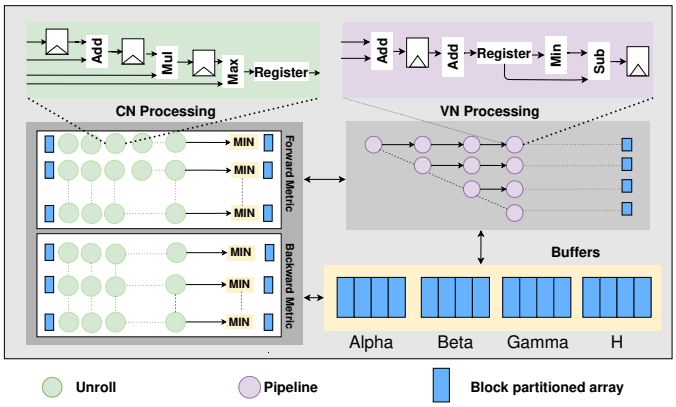

Fig. 2: Block Diagram of the implementation of Min-Max algorithm in FPGAs. The CNP represents row wise operation and the VNP represents column wise operations.The CNP is unrolled and the VNP is pipelined.

horizontal slices from the **H** matrix running $\frac{M}{\frac{1024}{a}}$ blocks in one dimension. The CNP kernel (line 10 to 12 of Algorithm 1) executes $\frac{M}{\frac{1024}{a}}$ by d_c blocks with individual threads processing individual symbols since inter-symbol data dependencies are non-existent. The VNP kernel (line 14 to 16 of Algorithm 1), executes vertical-parallel slices in $\frac{N}{\frac{1024}{a}}$ by d_v blocks. These blocks are automatically scheduled and executed by the GPU.

The scalability of this algorithm can be challenging. For codes with dimension higher than 1024 (M> 1024, N> 1024) it is hard to implement a competitive solution since in CUDA, blocks can only execute with a maximum of 1024 threads. The use of shared memory poses a barrier to scalability. Different devices have different amounts of shared memory and compiling the same kernel in different devices, might result in different SM configuration which could be detrimental in some devices. However, the kernels need to be tailored for the target GPU device, ensuring scalability. The GPU implementation is considered as a parallel baseline version and used for comparing different implementations.

B. FPGA: Reduced bitwidth, pipelining, loop unrolling and block partitioning

Fig. 2 shows the FPGA implementation of the Min-Max algorithm. For FPGAs, we used both the HLS and RTL implementations. The primary goal for the HLS approach was to produce a low latency implementation of the Min-Max algorithm. With high-throughput in mind, this work exploits the high-level of parallelism available in FPGAs, namely by identifying appropriate concurrent execution blocks in the code. For the HLS implementation, we used a combination of loop unrolling and pipelining in both the blocks along with block partitioning of $\beta_{m,n}(a)$,$\alpha_{m,n}(a)$ in both the dimension, storing them in registers as shown in Fig. 2. For the RTL implementation, in the check node block there are column dependencies within the $F(a)$ and $B(a)$ metrics. It takes $2\times(a+1)$ cycles to assign values to the first and last columns of $F(a)$ and $B(a)$ wherein, a cycles are spent to initialise Galois field elements with the BRAM taking 1 cycle. Followed by this, sequentially, the other columns of the $F(a)$ and $B(a)$ are computed, taking $(d_c-1)\times(4+a)$ cycles for each column. In the last stage, outputting $\beta_{m,n}(a)$ requires a cycles. Hence the total number of cycles is $19+(7\times a)$ for the CNP. In the VNP block, $3\times a$ cycles are required to process the obtained $\beta_{m,n}(a)$, one clock pulse to find the minimum of the temporary $\alpha'_{m,n}$ and a cycles to update $\alpha_{m,n}(a)$. The total number of the VNP block is $(a\times 3)+(a+1)$. The total number of cycles required for the entire design running 10 iterations can be found as $10\times((11\times a)+20)$.

IV. Experimental Setup

The GPU version of the code was validated on an Nvidia Jetson TX2 equipped with a low-power Nvidia GP10B GPU, using 256 cores (1.3 GHz) [4]. This implementation is written in CUDA. Each experiment is executed 20 times and the values presented in this paper are averaged. The FPGA device used for conducting experiments in HLS and RTL is the Virtex Ultrascale+ XCVU13p-FSGA2577-1-i. Xilinx SDx 2019.1 was used on the Nimbix online platform with 192 GB RAM (20 CPU cores). C/RTL co-simulation is used to ascertain the functional correctness of the design in HLS and RTL post-synthesis simulation was done for functional validation of the design. For HLS, FPGA resource utilisation, power and energy consumption were obtained after exporting the IP into the Vivado synthesis tool.

V. Experiments and Results

All experimental results were obtained by executing the decoder for 10 iterations on GF(2^2) and GF(2^3). To build the ***H*** matrix of dimension (384,256), it was used a process described by authors Kasai et al [5]. This method ensures a regular NB-LDPC code with a rate of $1/3$, $d_c=3$ and $d_v=2$ [6], [7]. The encoded codeword is composed of zeros embedded in additive white Gaussian noise (AWGN) with an SNR of 2dB.

A. GPU

The GPU uses multi-threading, running 1024 in multiple blocks. The kernels also exploit fast memory accesses through the use of shared memory (48KB available for each block). These kernels use full SM occupancy for all Galois fields except for the first kernel in GF(2^2). Throughput for GF(2^2) and GF(2^3) on the GPU was 1.856 and 2.335 Mbps, respectively. Throughput performance tends to decrease for higher Galois fields as the decoder's complexity increases.

B. FPGA

The Min-Max algorithm is implemented for GF(2^2) and GF(2^3). In the HLS implementation an *m_axi interface* was used as an interface to the Min-Max module and eight optimisation strategies were applied to a sequential version of the C code. Table I lists these optimisation indices.

Index I is the sequential version with the applied CSR/CSC compression. Index II uses block partitioning of the matrices α, β and unrolling of the innermost loops to extract parallelism. Index III introduces pipelining the VNP in addition to

TABLE I: Description of Optimisations Implemented

Index	Description
I	C sequential version
II	Array block partitioning + loop unrolling of inner loops
III	II + pipelining the VN alone
IV	Pipelining the CN alone with complete array partitioning
V	Pipelining the CN + II
VI	Pipelining the CN with complete array partitioning + unrolling inner loops in VN
VII	Pipelining the VN & CN with complete array partitioning
VIII	Implementation using Custom bit allocation + using instances + VII

the optimisations used in Index II. Index IV pipelines the CNP alone with complete array partitioning for the input matrix γ. Index V pipelines the CNP in addition to the optimisations used in index II. Index VI pipelines the CNP with complete array partitioning of all the matrices and unrolling the inner loops in the VNP. Index VII pipelines both VNP and CNP with complete array partitioning of all the matrices. For index VIII, the code was restructured to expose parallelism, custom bit-width were assigned (using ap_int library) for the Galois field operators and multi-dimensional array partition applied, in addition to Index VII.

For the HLS implementation for GF(2^2), the highest throughput was obtained for the optimisation index VIII at 91.92 Mbps. For GF(2^3) the optimisation index VIII surpassed performance at 92.6 Mbps as shown in Fig. 4. Optimisation Index VIII had high throughput due to the high extent of parallelism as described in section III-B. Fig. 3 shows the variation of frequency for various Galois fields. For GF(2^2) and GF(2^3),the highest frequency of operation were 571.42 MHz and 469.48 MHz respectively. The high frequency was obtained due to the presence of Flip-Flops. Table II shows the area utilisation for different optimisation solution for all the Galois fields. For GF(2^2) and GF(2^3), the highest area utilisation was obtained for optimisation index VIII. Compared to the other optimisations, index VIII shows high resource utilisation due to mapping of arrays into registers in both dimensions.

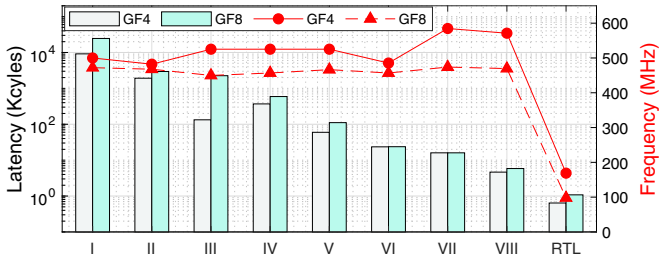

Fig. 3: Comparison of Min-Max algorithm for all Galois fields in FPGAs in terms of latency and frequency for varying optimisation indices. The left axis is represented in logarithmic scale. The bars represent latency and the lines represent frequency.

The state-machine implementation of the custom RTL code was analysed for GF(2^2) and GF(2^3). The current version of code was optimised for latency using parallel access to

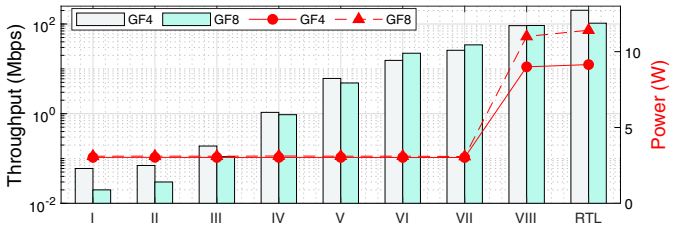

Fig. 4: Comparison of Min-Max algorithm for all Galois fields for FPGAs in terms of throughput and power variation for varying optimisation indices. The left axis is represented in logarithmic scale. The bars represent throughput and the lines represent power.

data (α and β) stored in BRAMs. The latency and throughput for the RTL design are plotted along with the best-case HLS implementation reports in both Fig. 3 and 4. The maximum frequency of operation for the RTL implementation for GF(2^2) was obtained at 168.94 MHz. The highest throughput obtained in RTL is also for GF(2^2), at 202.74 Mbps. From Table II, it can be seen that as Galois fields increase, the area utilisation increases proportionally. The RTL implementation is optimised for parallel memory access through multiple BRAM instantiation, which results in low latency, high frequency and BRAM utilisation.

Throughput and power levels for GF(2^2) and GF(2^3) are given in Fig. 4. High LUT utilisation on account of parallel instantiation of Galois field operators results in a corresponding increase in power for GF(2^3).

C. Roofline model analysis

The roofline model in Fig. 5 gives a comprehensive overview of the performance bottlenecks in a system. Here, we compare the design space exploration of the Min-Max algorithm on: (1) diverse micro-architectures, viz., CPU, GPU and FPGA and (2) across Galois fields GF(2^2) and GF(2^3). All the implementations lie in the compute-bound region.

The number of executed operations is obtained by running the *nvcc* profiler. For a given application we assume that the number of operations remains constant across the three devices. Data size refers to the non-zero elements of **H** and in RTL it is smaller since it uses fewer bits to represent symbols.

The single-thread CPU performance decreases as the Galois field grows due to the increase of data. The GPU shows peak performance for GF(2^3), where utilisation is at its peak, and shared memory and thread configuration fit the best occupancy rate. The use of shared and constant memory increases the performance for the same arithmetic intensity. For FPGAs, the peak performance is observed for GF(2^2) in RTL and GF(2^3) in HLS. In RTL, for GF(2^3), the performance decreases due to implementation complexity and area increase. For HLS, Fig. 5 shows an increase in both performance and arithmetic intensity as the Galois field is incremented. The increase in performance is linear, as it uses more resources. Using loop unrolling, array partitioning and pipelining increases the performance for the same number of operations. Using reduced precision instead of

TABLE II: Area Utilisation for HLS and RTL implementation.

	GF(2^2)			GF(2^3)		
Index	BRAM (%)	FF (%)	LUT (%)	BRAM (%)	FF (%)	LUT (%)
I	18 (0.6)	3440 (0.1)	5900 (0.3)	30 (1.1)	5721 (0.1)	8939 (0.5)
II	23 (0.9)	8371 (0.2)	10200 (0.6)	35 (1.3)	11403 (0.3)	15105 (0.8)
III	24 (0.9)	9804 (0.3)	11527 (0.7)	36 (1.3)	13856 (0.4)	18895 (1.1))
IV	24 (0.8)	11584 (0.4)	13319 (0.7)	40 (1.4)	22479 (0.6)	32286 (1.8)
V	18 (0.8)	24431 (0.7)	23500 (1.3)	36 (1.3)	34233 (0.9)	44775 (2.5)
VI	24 (0.9)	11458 (0.3)	13034 (0.7)	40 (1.4)	22828 (0.6)	32315 (1.8)
VII	24 (0.1)	11489 (0.3)	13073 (0.8)	40 (1.4)	22999 (0.6)	32649 (1.8)
VIII	**48 (1.78)**	**137508 (3.97)**	**637639 (38)**	**96(3.56)**	**275016(7.94)**	**1275278(76)**
RTL	768 (28.6)	83932 (2.7)	333587 (19.3)	768 (28.6)	188586 (5.46)	1380989 (79.92)

double or single floating-point precision reduces the memory requirement for HLS and RTL.

D. Energy and efficiency analysis

Table III shows the energy consumption for the CPU, GPU and FPGA. The RTL implementation achieves lower energy consumption compared to the other implementations. On Jetson TX2 the power measurements are written to a file by the OS. Reading sampling rates must be higher than 1 second to increase resource utilisation [4]. It was assumed a realistic mean power of 5 W. For the FPGA as the size of the Galois field increases more energy is consumed.

The peak energy efficiency for GF(2^2) is obtained for the RTL implementation, at 22.16 (Mbps/W), while for GF(2^3) it is achieved 9.14 (Mbps/W), as shown in Table III. Among all the devices, the FPGA shows higher performance while requiring low energy consumption.

E. Discussion

RTL is more suitable to manually introduce parallelism in the hardware. The current implementation minimises latency, hence directly improves throughput. Optimising the Galois field operators for higher frequencies would result in an increase in overall throughput. HLS exploits micro-architectural design space exploration. Replicating the decoder IP core to accommodate the entire FPGA area, with a small area reserved for orchestration/control, synchronisation and communication, increases data-parallelism and throughput.

With the area utilisation as in Table II for GF(2^2), the RTL design can be replicated 4 times, allowing multi-codeword decoding. This replication of the decoder cores will give the overall throughput of RTL to be 812 Mbps, which surpasses the current state-of-the-art [11].

TABLE III: Energy consumption and efficiency for Min-max algorithm on CPU, GPU and FPGA for HLS and RTL.

	Energy consumption (uJ)				Efficiency (Mbps/W)			
	CPU[1]	GPU[1]	HLS	RTL	CPU[1]	GPU[1]	HLS	RTL
GF(2^2)	5180	2068	73	**34**	0.15	0.37	10.22	**22.16**
GF(2^3)	16390	2465	136	**126**	0.07	0.47	8.42	**9.14**

[1]Assuming 5 W running on the Jetson TX2 [4].

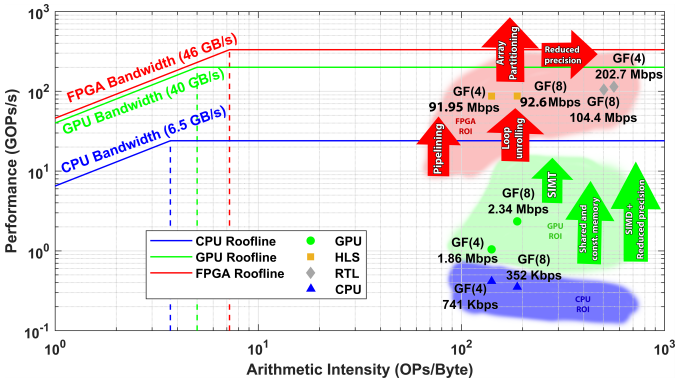

Fig. 5: Roofline models for the Denver 2 CPU, GPU, HLS and RTL with the associated performance.

VI. Related Work

Table IV gives an overview of existing FPGA implementations of NB-LDPC decoders. In terms of throughput, the custom hardware developed using Trellis Min-Max algorithm [11] outperforms all other implementations. The second best algorithmic implementation is the weighted bit-reliability based NB-LDPC decoding algorithm [12], followed by the full bit-reliability-based one. The iterative reliability-based majority-logic NB-LDPC decoding algorithm [9] is capable of achieving 90 Mbps and the enhanced algorithm can achieve at least tens of times higher efficiency as compared to the Min-Max algorithm. Comparatively, reliability-based majority-logic NB-LDPC decoding algorithm has better performance with reduced complexity. The enhanced (E-) IHRB algorithm uses a small amount of additional hardware to further improve throughput [9]. Next in performance is the implementation of the extended min-sum algorithm [8] that is able to achieve up to 50 Mbps for GF(2^2) through the design of a semi-parallel custom architecture that overlaps check node and variable node calculations. Nevertheless, all the above implementations [3], [8]–[17] primarily involve custom RTL design, which implies significant design and verification time. Our focus in this paper has been to apply both high-level synthesis techniques and register transfer level language to perform a comprehensive design space exploration across a range of Galois fields. The possibility to translate high-level specifications into FPGA performance analysis gives us the freedom to quickly explore the impact on optimisation techniques on different Galois fields. In this context, our implementation outperforms the HLS-based FFT-SPA implementation [6] by about 80× for GF(2^2) and 97× for GF(2^3) respectively. The reduction in

TABLE IV: Comparison of FPGA-based implementation of NB-LDPC decoders

	Algorithm	Paper	FPGA	Code	Galois field	Frequency (MHz)	Throughput (Mbps)	Iterations
Custom RTL	Extended Min-Sum	[8]	Virtex-4	(972,486)	GF(2^2)	131	50	NA
	Majority logic	[9]	Virtex-5	(403, 226)	GF(2^5)	117.6	90.68	25
	Enhanced Majority logic	[9]	Virtex-5	(403, 226)	GF(2^5)	109.8	84.67	25
	FFT-SPA with layered scheduling	[10]	Virtex-5	(512, 256)	GF(2^2)	100	33	1
				(256, 128)	GF(2^4)	100	13.2	1
				(128, 64)	GF(2^8)	100	1.56	1
	Trellis Min-Max	[11]	Virtex-7	(2304,2048)	GF(2^4)	226	**630**	10
	Weighted bit-reliability based	[12]	Virtex-5	(403,226)	GF(2^5)	106.3	118.98	15
	Full bit-reliability-based	[12]	Virtex-5	(403,226)	GF(2^5)	85.52	95.73	15
	Min-Max	[13]	Virtex-2	(744,653)	GF(2^5)	106	**9.3**	15
	Min-Max	**[This work]**	Virtex Ultrascale+	(384,256)	GF(2^2)	168.9	**202.7**	10
					GF(2^3)	97.9	104.38	10
HLS	FFT-SPA	[6]	Virtex-7	(384,256)	GF(2^2)	250	1.17	10
					GF(2^3)	250	0.95	10
					GF(2^4)	216	0.66	10
	Min-Max	**[This work]**	Virtex Ultrascale+	(384,256)	GF(2^2)	571.4	91.95	10
					GF(2^3)	461.48	**92.6**	10

algorithmic complexity in the Min-Max algorithm and the upgrade in device architectures add to the improvement in throughput.

VII. Conclusions

In this paper, we compare the performance of a GPU-based approach and FPGA designs developed using RTL and HLS design space exploration. HLS sits in the middle of GPUs and RTL in terms of development effort and performance, as shown by the roofline model comparison.

The parallel instantiation of multiple RTL decoders with a frequency optimised approach can lead to an aggregate throughput surpassing 800 Mbps, which, for the best of our knowledge, clearly surpasses the state-of-the-art, except, of course, for ASIC-based designs. In terms of energy, both FPGA-based techniques are similar, but RTL tends to achieve superior efficiency (Mbps/W) for all Galois fields as compared to the HLS version. Future work addresses more efficient architectures for higher Galois Fields (above GF(2^5)).

VIII. Acknowledgements

ECHO is a joint work supported under the Indo-Portugal Bilateral Scientific and Technological Cooperation funded by Instituto de Telecomunicações and Fundação para a Ciência e a Tecnologia in Portugal (UIDB/EEA/50008/2020 and PTDC/ EEI-HAC/30485/2017) and the Department of Science and Technology (INT/PORTUGAL/P-12/2017), Government of India. This work was also supported under "Just-in-time Customisation: Research Platform for Accelerating Algorithms-to-Systems" funded within the Early Career Research Award by Science and Engineering Research Board, India (ECR/2016/001765).

References

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[2] O. Ferraz, S. Subramaniyan, G. Wang, J. R. Cavallaro, G. Falcão, and M. Purnaprajna, "Gbit/s Non-Binary LDPC Decoders: High-Throughput using High-Level Specifications," in *IEEE International Symp. on Field-Programmable Custom Computing Machines (FCCM)*, May 2020.

[3] V. Savin, "Min-Max decoding for non binary LDPC codes," in *2008 IEEE Int. Symp. on Inf. Theory*, July 2008, pp. 960–964.

[4] Nvidia, "Data Sheet, Nvidia Jetson TX2 Series System-on-Module," accessed: May. 2020 [Online]. Available: https://developer.nvidia.com/embedded/dlc/jetson-tx2-module-datasheet.

[5] K. Kasai, R. Matsumoto, and K. Sakaniwa, "Information reconciliation for QKD with rate-compatible non-binary LDPC codes," in *2010 Int. Symp. On Inf. Theory Its Applications*, 2010, pp. 922–927.

[6] J. Andrade, N. George, K. Karras, D. N. Novo, V. Silva, P. I. Ienne, and G. Falcão, "From Low-architectural Expertise Up to High-throughput Non-binary LDPC Decoders: Optimization Guidelines using High-level Synthesis," in *International Conf. on Field Programmable Logic and Applications - FPL*, vol. 1, September 2015, pp. 1–8.

[7] ——, "Fast Design Space Exploration using Vivado HLS: Non-Binary LDPC Decoders," in *IEEE International Symp. on Field-Programmable Custom Computing Machines (FCCM)*, May 2015.

[8] Yue Sun, Yuyang Zhang, Jianhao Hu, and Zhongpei Zhang, "FPGA implementation of nonbinary quasi-cyclic LDPC decoder based on EMS algorithm," in *2009 International Conference on Communications, Circuits and Systems*, July 2009, pp. 1061–1065.

[9] X. Zhang, F. Cai, and S. Lin, "Low-Complexity Reliability-Based Message-Passing Decoder Architectures for Non-Binary LDPC Codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 1938–1950, Nov 2012.

[10] T. Lehnigk-Emden and N. Wehn, "Complexity evaluation of non-binary Galois field LDPC code decoders," in *2010 6th International Symposium on Turbo Codes Iterative Information Processing*, Sep. 2010, pp. 53–57.

[11] J. O. Lacruz, F. García-Herrero, M. J. Canet, J. Valls, and A. Pérez-Pascual, "A 630 Mbps non-binary LDPC decoder for FPGA," in *IEEE Int. Symp. on Circuits and Syst. (ISCAS)*, May 2015, pp. 1989–1992.

[12] Q. Huang and S. Yuan, "Bit Reliability-Based Decoders for Non-Binary LDPC Codes," *IEEE Transactions on Communications*, vol. 64, no. 1, pp. 38–48, 2016.

[13] X. Zhang and F. Cai, "Reduced-Complexity Decoder Architecture for Non-Binary LDPC Codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1229–1238, July 2011.

[14] C. Spagnol, W. Marnane, and E. Popovici, "FPGA Implementations of LDPC over GF(2m) Decoders," in *2007 IEEE Workshop on Signal Processing Systems*, Oct 2007, pp. 273–278.

[15] F. García-Herrero, M. J. Canet, and J. Valls, "High-speed NB-LDPC decoder for wireless applications," in *2013 International Symposium on Intelligent Signal Processing and Communication Systems*, Nov 2013, pp. 215–220.

[16] G. Wang, H. Shen, B. Yin, M. Wu, Y. Sun, and J. R. Cavallaro, "Parallel non-binary LDPC decoding on GPU," in *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, Nov 2012, pp. 1277–1281.

[17] D. Zou and I. B. Djordjevic, "FPGA implementation of concatenated non-binary QC-LDPC codes for high-speed optical transport," *Optics express*, vol. 23, no. 11, pp. 14 501–14 509, 2015.