Multi-Stream LDPC Decoder on GPU of Mobile Devices

Roohollah Amiri
Electrical and Computer Engineering
Boise State University, Idaho, USA
roohollahamiri@u.boisestate.edu

Hani Mehrpouyan
Electrical and Computer Engineering
Boise State University, Idaho, USA
hanimehrpouyan@boisestate.edu

***Abstract*—Low-density parity check (LDPC) codes have been extensively applied in mobile communication systems due to their excellent error correcting capabilities. However, their broad adoption has been hindered by the high complexity of the LDPC decoder. Although to date, dedicated hardware has been used to implement low latency LDPC decoders, recent advancements in the architecture of mobile processors have made it possible to develop software solutions. In this paper, we propose a multi-stream LDPC decoder designed for a mobile device. The proposed decoder uses graphics processing unit (GPU) of a mobile device to achieve efficient real-time decoding. The proposed solution is implemented on an NVIDIA Tegra board as a system on a chip (SoC), where our results indicate that we can control the load on the central processing units through the multi-stream structure.**

***Index Terms*—Parallel and Distributed Algorithms, Multiprocessor Architectures, LDPC Decoder, GPU Processing.**

I. Introduction

Low-density parity check (LDPC) codes were originally proposed by Robert Gallager in 1962 [1] and rediscovered by MacKay and Neal in 1996 [2]. LDPC codes have been adopted by a wide range of communication standards such as IEEE 802.11n, 10 Gigabit Ethernet (IEEE 802.3an), Long Term Evolution (LTE), and DVB-S2. Chung and Richardson [3] showed that a class of LDPC codes could approach the Shannon limit to within 0.0045 dB. However, the error correcting strength of LDPC codes comes at the cost of very high decoding complexity [4]. Moreover, to date, there are no closed-form solutions to determine the performance of LDPC codes in various wireless channels and systems. Thus, performance evaluation is typically carried out via simulations on computers or dedicated hardwares [5].

Since LDPC decoders are computationally-intensive and need powerful computer architectures to result in low latency and high throughput, to date, most LDPC decoders are implemented using application-specific integrated circuits (ASIC) or field-programmable gate array (FPGA) circuits [6]. However, their high speed often comes at a price of high development cost and low programming flexibility [7]. Further, it is very challenging to design decoder hardware that supports various standards and multiple data rates [8]. Decoding of LDPC codes is implemented via belief propagation also known as sum-product algorithm (SPA). One advantage of iterative schemes based on the SPA is that it could be parallelized based on the architecture of the code graph [3]. In recent years, researchers have used multi-core architectures such as CPUs [9], [10], graphics processing units (GPUs) [5], [11], [12], and advanced RISC machines (ARMs) [10], [13] to develop high throughput and low latency software-defined radio (SDR) applications. Therefore, designers have recently focused on software implementations of LDPC decoders on multi/many-core devices [11] to meet the performance requirements of current communication systems.

In microarchitectures, increasing clock frequencies to obtain faster processing performance has reached the limits of silicon-based architectures. Hence, to achieve gains in processing performance, other techniques based on parallel processing is being investigated [4]. Todays' multi-core architectures support single instruction multiple data (SIMD), single program multiple data (SPMD), and single instruction multiple threads (SIMT). The general purpose multi-core processors homogeneously replicate a single core, typically with an x86 instruction set, and provide shared memory hardware mechanisms [11]. Such multi-core structures can be programmed at a high level by using different software technologies [14] such as Open Multi-Processing (OpenMP) [15] which provides a practical and relatively straightforward approach for general-purpose programming. On the other hand, newer microarchitectures are trying to provide larger SIMD units for vector processing like streaming SIMD extensions (SSE), advanced vector extensions (AVX), and AVX2 [16] on Intel Architectures. In [4], the authors have used Intel SSE/AVX2 SIMD units to implement a high throughput LDPC decoder efficiently. Meanwhile, the power consumption of x86 implementations is incompatible with most of the embedded mobile systems, which makes them useful for simulation purposes only.

Over the last decade, the performance of GPUs has significantly improved mainly due to the demands for visualization technology in the gaming industry. Recent GPUs are composed of many cores which are driven by considerable memory bandwidth. Therefore, they are also being targeted for solving computationally intensive algorithms in a multithreaded and highly parallel fashion. Hence, researchers in the high-performance computing field are applying GPUs to general-purpose applications (GPGPU). Pertaining to the field of communication, researchers have used Compute Unified Device Architecture (CUDA) from NVIDIA [5], [8], [12], [17], [18] and Open Computing Language (OpenCL) [19] platforms

978-1-7281-0554-3/19/$31.002019 IEEE

to develop LDPC decoders on GPUs. As an example, the authors in [17] have achieved almost 1 Gbps of decoding throughput for LDPC codes on GPU devices. Although these works can achieve extremely high throughputs, their latency beyond seconds, their high power consumption, and their cost make them incompatible with embedded mobile devices. The devices of the end users usually have limited access to a large power source. As such, these devices must operate on limited resources as small processors, tiny memory, and low power budget. In other words, the limited available resources must be used most effectively and efficiently.

ARM-based SDR systems have been proposed in recent years [10], [13] with the goal of developing an SDR based LDPC decoder that provides high throughput and low latency on a low-power embedded system. The authors in [13] have used the ARM processor's SIMD and SIMT programming models to implement an LDPC decoder. This approach allows reaching high throughput while maintaining low-latency. However, the proposed ARM-based solution in [13] is based on the assumption that the ARM processor is solely used for LDPC decoding. However, mobile devices need to support multiple applications simultaneously, and the processing resources cannot be extensively dedicated to the LDPC decoder. Moreover, recent works in SDR LDPC embedded systems are missing the fact that today's mobile devices have powerful CUDA enabled GPUs which can play a significant role as a computing resource in an embedded system.

This paper proposes a GPU-based LDPC decoder for an embedded device. The structure of the proposed decoder is based on multiple data streams which first makes it scalable to other architectures, and second, the process imposed by the decoding can be controlled by choosing the appropriate number of data streams that are sent to the GPU device. Moreover, since the ARM and GPU of an embedded device are collocated on the same die, the latency issues associated with a GPU implementation is limited.

The remainder of the paper is structured as follows. Section II briefly introduces the LDPC error correcting codes and their decoding algorithms. Then the proposed heterogeneous algorithm on embedded mobile targets is described in Section III. Section IV gives experimental results and finally, Section V concludes the paper.

II. LDPC codes and their Decoding Processes

LDPC codes are a class of linear block codes with a sparse parity check matrix called H-matrix. Their main advantage is that they provide a performance which is close to that of the channel capacity for various wireless channels. Furthermore, the decoding process of LDPC codes is suited for implementations that make heavy use of parallelism [20]. Here, we present a brief background on LDPC codes[1]. There are two ways to represent LDPC codes. Like all linear block codes, they can be described by their H-matrix, while they can also be represented by a Tanner graph which is a bipartite graph. An LDPC graph consists of a set of variable nodes, a set of check nodes, and a set of edges E. Each edge connects a variable node to a check node. For example, when the (i, j) element of an H-matrix is '1', the ith check node is connected to the jth variable node of the equivalent Tanner graph. Fig. 1 illustrates the equivalent Tanner graph for a 10 variable nodes and 5 check nodes, $(10, 5)$, LDPC code with H-matrix in (1) [20].

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad (1)$$

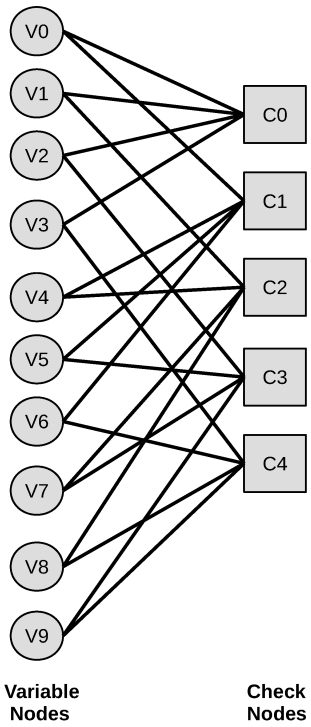


Fig. 1: Tanner graph of the H-matrix in (1)

The general decoding algorithm of LDPC codes is based on the standard two-phase message passing (TPMP) principle described in [11]. This algorithm works in two phases. In the first phase, all the variable nodes send messages to their neighboring parity check nodes, and in the second phase, the parity check nodes send messages to their neighboring variable nodes. One practical variant of message passing algorithms is Min-Sum algorithm which is preferred by designers [13]. The general steps taken in the Min-Sum algorithm are provided in Algorithm 1. In Algorithm 1, LLR stands for log-likelihood ratio, CN_m and VN_n denote the mth check node and the nth variable node, respectively.

[1]The reader is referred to [21] for more information.

Algorithm 1 Min-Sum algorithm

1: **Loop 1:** Initialization
2: **for all** $t = 1 \rightarrow$ (Max Iterations) **do**
3: **Loop 2:** LLR of message CN_m to VN_n
4: **Loop 3:** LLR of message VN_n to CN_m
5: **Loop 4:** Hard decision from soft-values
6: **end for**

One major drawback of Algorithm 1 is that Loops 2 and 3 are updated by separate processing and passed to each other iteratively. It means that the update loop of the variable nodes does not start until all check nodes are updated. This characteristic affects the efficiency of parallel implementation of such an algorithm.

Due to the poor parallel mapping of the Min-Sum algorithm, more efficient schedules, such as horizontal layered-based decoding algorithm, are proposed which allow updated information to be utilized more quickly in the algorithm, thus, speeding up decoding [18], [22]. In fact, the H-matrix can be viewed as a layered graph that is decoded sequentially. The work in [17] has applied a form of layered belief propagation to irregular LDPC codes to reach 2 times faster convergence for a given error rate. By using this method, the memory bits usage is reduced by 45% to 50%. The layered decoding algorithm is denoted as Algorithm 2 and can be summarized as follows:

1) All values for the check node computations are computed using variable node messages linked to them.
2) Once, a check node is calculated, the corresponding variable nodes are updated immediately after receiving messages.
3) This process is repeated to the maximum number of iterations.

In this paper, we propose a multi-stream structure for implementing the layered decoding of LDPC codes on the GPU device of a mobile processor with high throughput and low latency performance. By using GPU device as the processing unit, significantly fewer resources of the ARM processor is used for decoding compared to similar work in [13]. Thus, the ARM processor gains more processing power for other applications running on the device. On the other hand, since the GPU and ARM of a mobile device are sitting on the same die, the latency issues in [17] are improved.

III. Algorithm Mapping

An efficient implementation of the layered decoding algorithm is a challenging task. The concerning programming drawbacks of this algorithm are as follows:

1) The number of computations for the number of memory access is low.
2) The data reuse between consecutive computations is low.
3) It requires a large set of random memory access due to the sparse nature of the H-matrix [4].

Therefore, a software-based decoder should take advantage of different parallelism levels offered by the target architecture to achieve high throughput efficiency. In this section, we detail the different parallelism levels, target architecture and the structure of the proposed algorithm.

A. Parallelism Levels in the Proposed Algorithm

To achieve high throughput performance, a software-based LDPC decoder has to exploit computational parallelism for taking advantage of multi-core architectures. Different parallelism levels are present in a layered decoding algorithm, which include:

1) First parallelism level is located inside the check node computations. Executing such computations in parallel is possible. However, this atomic parallelism level is hard to exploit due to the low complexity of computations. On the other hand, two check node computations can be done in parallel if there is no data dependency. Since this is rarely true, this level is hard to exploit and inefficient.
2) Second parallelism level is located at the frame level (complete execution of Algorithm 2). The same computation sequence is executed over consecutive frames. This approach provides an efficient parallel processing algorithm.

Hence, here, we use the SIMD programming model to decode F frames in parallel. In subsection III-C the parallel decoding of F frames is referred to as kernel 2 for the sake of simplicity.

B. Data Interleaving/Deinterleaving

Recall that the implementation of the parallel frame processing is subject to massive irregular memory access due to the structure of H-matrix. To process the same VN_n element of the F frames at the same time, non-contiguous memory access would affect performance. To solve this issue, a data interleaving process has to be performed before and after the decoding stage to ensure that each set of F frames are reordered to achieve an aligned memory data structure. We use the same procedure as in [4] and the reordering is shown in Fig. 2. In the proposed structure, interleaving and deinterleaving of frames are called kernel 1 and kernel 3.

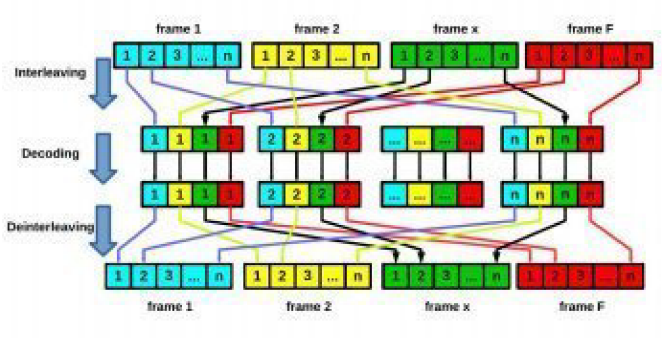


Fig. 2: Data interleaving/deinterleaving process [4]

C. Multi Stream Parallelism

The SIMT programming model is used to decode W sets of F frames concurrently, with W denoting the number of

concurrent streams on the GPU device. This multi-core programming is specified by the CUDA API. Each GPU stream is controlled by a *pthread* called *worker* on the host machine (which is an ARM in this case). Each *worker* is responsible for its own sets of frames. By using stream-based processing, the system can decode $W \times F$ frames at the same time. The whole LDPC decoder system model is shown in Fig. 3.

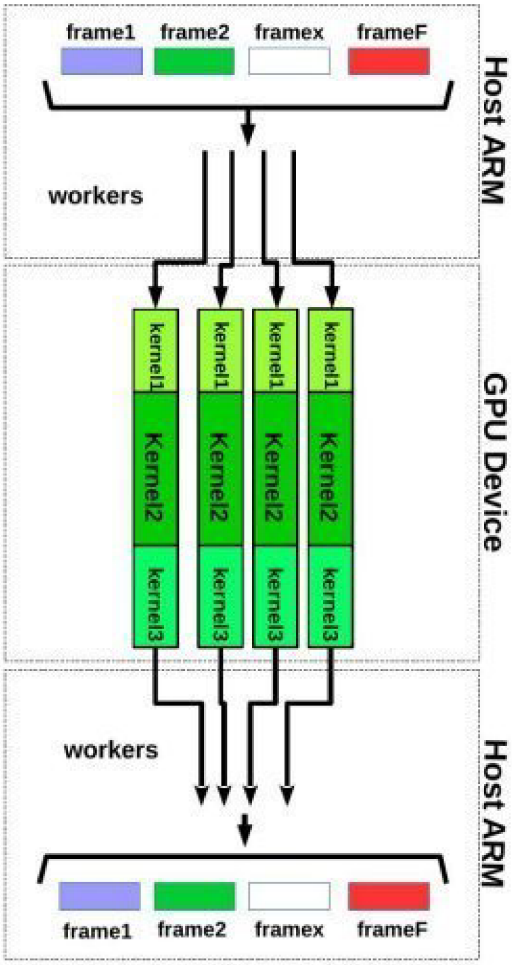


Fig. 3: LDCP decoder data flow

IV. Experimental Results

The experiments were carried out by decoding LDPC codes using NVIDIA Tegra K1 SoCs and various other structures to show scalability. The programs were compiled via GCC-4.8 and CUDA 6.5. The TK1 is composed of 4 Cortex-A15 ARM processors and one NVIDIA Kepler "GK20a" GPU with 192 SM3.2 CUDA cores. The host platform uses a GNU/Linux kernel 3.10.40.

A. Performance Evaluation of the Proposed Algorithm

The first set of experiments evaluates the decoding throughput of different LDPC codes. The codes have different frame lengths: 576 to 9972. The results are provided in Fig. 5 when one or three threads are used to handle one or three GPU

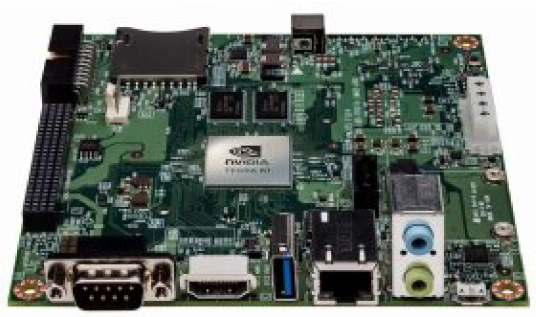


Fig. 4: Tegra-TK1 development board

streams. Measurements are performed for LDPC decoders that execute 10 layered-base decoding iterations.

One stream decoding achieves 25 Mbps, while with three streams it can be as high as 35 Mbps. For a $(4000, 2000)$ LDPC code and one thread, data transfer takes about 2×2.4 ms, interleaving steps need about 2×5 ms and decoding takes about 150 ms. For the same code with 3 threads, data transfer takes approximately 2×2.4 ms, interleaving steps need about 2×5 ms and decoding takes about 150 ms. Therefore, by introducing more streams to GPU device, its performance does not degrade. In comparison, the latency, i.e., the time for data transfer between the host and GPU device in [17] is about 20 ms, is reduced to 4.8 ms because of the architecture of the embedded mobile device. On the other hand, with introducing three streams to GPU, its processing capacity is used more effectively which results to about 30% throughput improvement in most of our experiments.

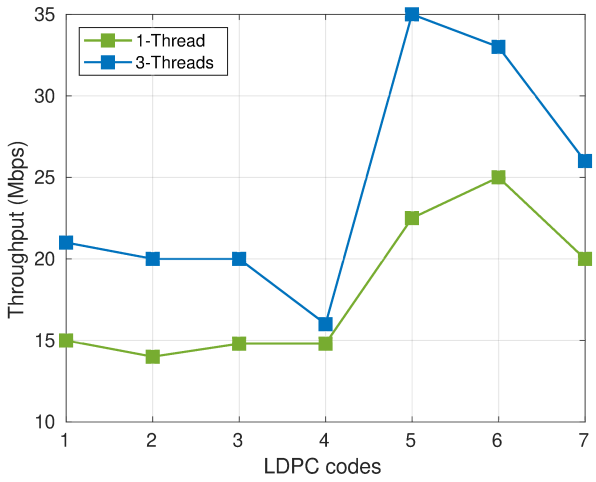


Fig. 5: Measured throughputs for 10 layered decoding iterations ($1 - 7$ LDPC codes: $576 \times 288, 1024 \times 512, 1200 \times 600, 1944 \times 722, 4000 \times 2000, 8000 \times 4000, 9972 \times 4086$)

B. Performance Comparison with Related Works

To demonstrate the efficiency of the proposed ARM decoder, its throughput was compared to the ARM related work in [13]. In [13], ARM SIMD units are used to perform vector

data processing in parallel frame decoding. In the experiment, the throughput of the proposed decoder is compared to that of [13] while using 1 thread for the work in [13] and 3 threads in the proposed algorithm. This selection is motivated by the fact that the 1 thread from [13] uses a 100% of a core while the 3 threads for the proposed algorithm only uses 8% of each core resulting in an overall utilization of 24%. 10-iteration decoding performed on Tegra-K1 board gives us the results as shown in Table I. The work in [13] can achieve much higher throughputs by using more threads on the ARM processor, but by introducing each thread, the whole capacity of one more ARM core is used for decoding. In Table I, it is shown that the proposed algorithm can achieve the similar throughput to that of [13] when using 24% of ARM processing power and using its GPU device. Although, by using more powerful GPU device, the algorithm can achieve much higher throughputs which has been shown in next subsection. This shows that the proposed algorithm is scalable across platforms.

TABLE I: Throughput (Mbps) Comparison With Related Work

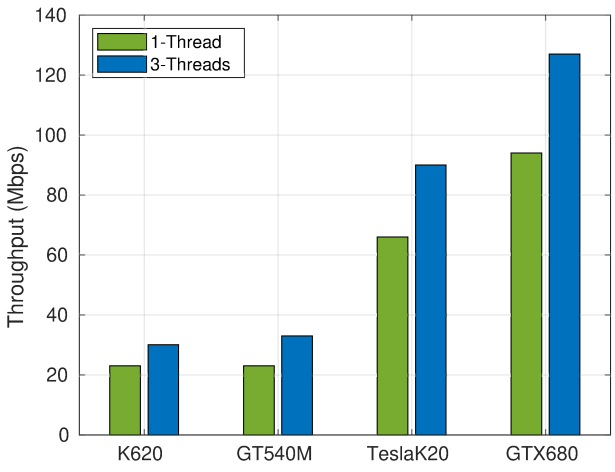
	ARM decoder [13], 1 thread		**Proposed decoder, 3 thread**	
code	**(Mbps)**	**Processes used**	**(Mbps)**	**Processes used**
(4000,2000)	**35**	**100%**	**34.5**	***24%***
(8000,4000)	**34**	**100%**	**33**	***24%***

C. Performance Comparison on Different GPU Devices

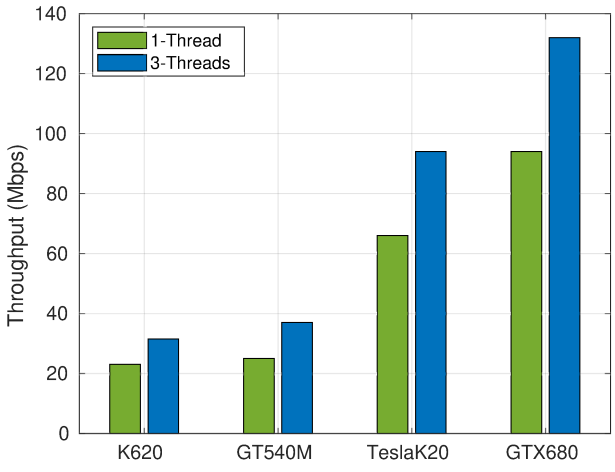
GPU devices have different characteristics such as the number of stream multiprocessors, CUDA cores, and working frequencies. A GPU based algorithm should have the scalability to use all the processing capability of a GPU device. The proposed algorithm has been executed on multiple GPU devices. GT540M and K620 are considered as mid-range and GTX680, and TeslaK20 are considered as high power GPU devices. The algorithm is executed for three code lengths as $(576, 288)$, $(2304, 1152)$ and $(4000, 2000)$. The performance is shown for 10 and 5 iterations in two sets of figures in Fig. 6 and Fig. 7. These figures show that the proposed algorithm can achieve up to 230 Mbps performance across devices. In these set of experiments, an x86 CPU processor is the host.

V. Conclusion

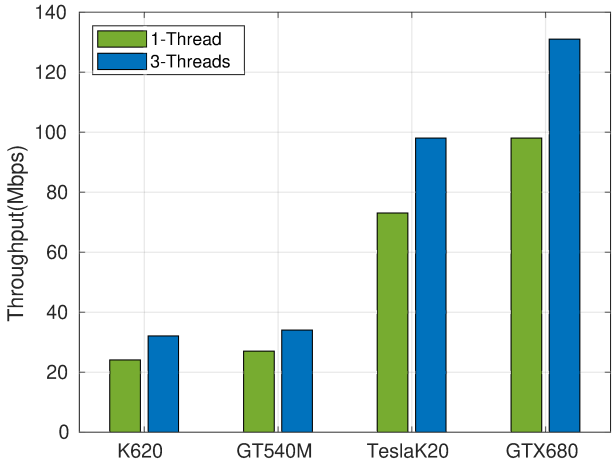
A stream-based approach for GPU-based LDPC decoding on embedded devices was introduced in this paper. This algorithm is based on running multiple concurrent kernels on GPU devices to utilize their processing capacity and freeing up resources on the ARM processor of mobile devices. Our results show that this approach helps to achieve higher throughputs on embedded mobile devices. Experimental results demonstrate that the proposed algorithm is scalable and can achieve high throughputs on multiple GPU devices. Moreover, the proposed algorithm structure provides a trade-off for the operating system to choose between performance and resource management by selecting various values for the number of streams that are used for decoding.



(a) code=(576,288)

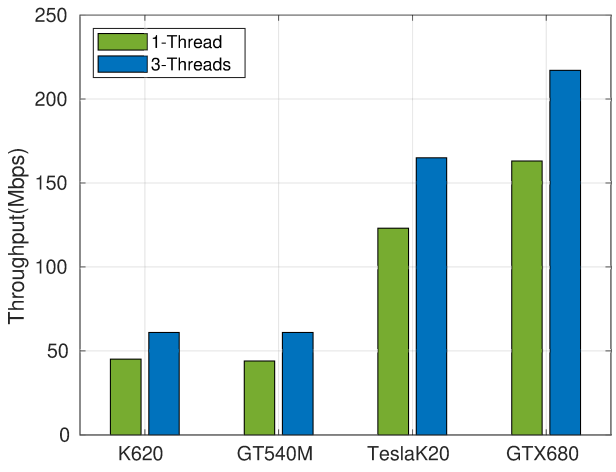


(b) code=(2304,1152)

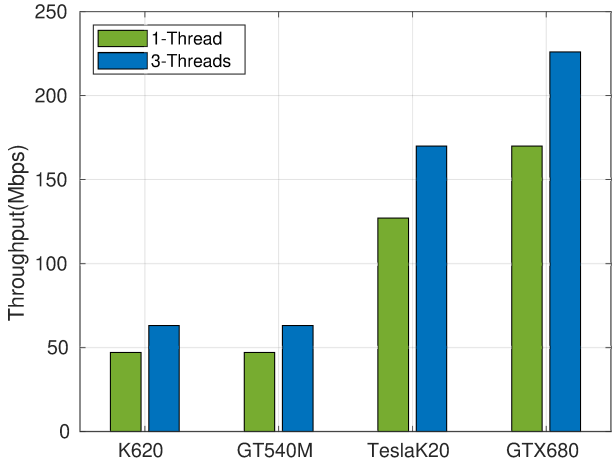


(c) code=(4000,2000)

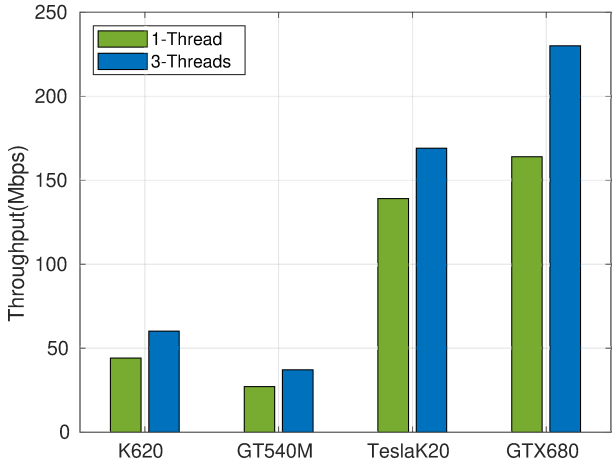
Fig. 6: 10 iteration experiment



(a) code=(576,288)



(b) code=(2304,1152)



(c) code=(4000,2000)

Fig. 7: 5 iteration experiment

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan 1962.

[2] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar 1997.

[3] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb 2001.

[4] B. L. Gal and C. Jego, "High-throughput multi-core LDPC decoders based on x86 processor," *IEEE Trans. Parallel Distrib. Syst.*, vol. PP, no. 99, pp. 1–1, May 2015.

[5] S. Kang and J. Moon, "Parallel LDPC decoder implementation on GPU based on unbalanced memory coalescing," in *Communications (ICC), 2012 IEEE International Conference on Proc*, June 2012, pp. 3692–3697.

[6] J. Andrade, G. Falcao, and V. Silva, "Flexible design of wide-pipeline-based WiMAX QC-LDPC decoder architectures on FPGAs using high-level synthesis," *Electronics Letters*, vol. 50, no. 11, pp. 839–840, May 2014.

[7] Y. Hou, R. Liu, H. Peng, and L. Zhao, "High throughput pipeline decoder for LDPC convolutional codes on GPU," *IEEE Commun. Lett.*, vol. 19, no. 12, pp. 2066–2069, Dec 2015.

[8] J.-Y. Park and K.-S. Chung, "Parallel LDPC decoding using CUDA and OpenMP," *EURASIP JWCN*, vol. 2011, no. 1, pp. 1–8, Nov. 2011.

[9] S. Grönroos, K. Nybom, and J. Björkqvist, "Efficient GPU and cpu-based LDPC decoders for long codewords," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 2, pp. 583–595, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10470-012-9895-7

[10] S. Grnroos and J. Bjrkqvist, "Performance evaluation of LDPC decoding on a general purpose mobile cpu," *in Proc. IEEE GlobalSIP*, pp. 1278–1281, Dec 2013.

[11] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 309–322, Feb 2011.

[12] G. Wang, M. Wu, B. Yin, and J. R. Cavallaro, "High throughput low latency LDPC decoding on GPU for SDR systems," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, Dec 2013, pp. 1258–1261.

[13] B. L. Gal and C. Jego, "High-throughput LDPC decoder on low-power embedded processors," *IEEE Commun. Lett.*, vol. 19, no. 11, pp. 1861–1864, Nov 2015.

[14] H. Kim and R. Bond, "Multicore software technologies," *IEEE Signal Processing Mag.*, vol. 26, no. 6, pp. 80–89, November 2009.

[15] B. Chapman, G. Jost, and R. v. d. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2007.

[16] M. Deilmann, "A guide to auto-vectorization with intel c++ compilers," *Intel Corporation*, April 2012.

[17] B. L. Gal, C. Jego, and J. Crenne, "A high throughput efficient approach for decoding LDPC codes onto GPU devices," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 29–32, June 2014.

[18] B. L. Gal and C. Jego, "Gpu-like on-chip system for decoding LDPC codes," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4, pp. 95:1–95:19, Mar. 2014.

[19] G. Falcao, V. Silva, L. Sousa, and J. Andrade, "Portable LDPC decoding on multicores using opencl [applications corner]," *IEEE Signal Processing Magazine*, vol. 29, no. 4, pp. 81–109, July 2012.

[20] D. J. Costello Jr, "An introduction to low-density parity check codes," 2009.

[21] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[22] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," *in Proc. IEEE SiPS*, pp. 107–112, Oct 2004.