Implementation of an LDPC Decoder on GPU of Mobile Devices

Roohollah Amiri

April 30, 2016

Abstract

Low density parity check (LDPC) codes have been extensively applied in mobile communication systems due to their excellent error correcting capabilities. However, their wide 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 has made it possible to develop software solutions. Here, we propose a multi-stream LDPC decoder that uses both ARM and GPU processors of a mobile device to achieve efficient real-time decoding. The proposed solution is implemented on an NVIDIA development board, where our results indicate that we can reduce the load on either the GPU or the ARM processor through the proposed structure.

1 Introduction

Originally proposed by Robert Gallager in 1962 [1] and rediscovered by MacKay and Neal in 1996 [2] Low Density Parity Check (LDPC) codes have been adopted by a wide range of applications including many communication standards such as IEEE 802.11n, 10 Gigabit Ethernet (IEEE 802.3an), Long Term Evolution (LTE) and DVB-S2. Recently, Chung and Richardson [3] showed that a class of LDPC codes can approach the Shannon limit to within 0.0045 dB. However, the error correcting strength of these 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 developement cost, low programming flexibility [7] and it is very challenging to design decoder hardware that

supports various standards and multiple data rates [8]. On the other hand, iterative LDPC decoding schemes based on the sum-product algorithm (SPA) can be fully parallelized, leading to high-speed decoding [3]. For these reasons, designers have recently focused on software implementations of LDPC decoders on multi/many-core devices [9] to meet the performance requirements of current communication systems through software defined radio (SDR). In terms of multi-core architectures, researchers have used CPUs [10, 11], graphics processing units (GPUs) [5, 9, 12], and advanced RISC machine (ARM) [11, 13] architectures to develop high throughput, low latency SDRs.

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 programm multiple data (SPMD), and single instruction multiple threads (SIMT). The general purpose multi-core processors replicate a single core in a homogeneous way, typically with an x86 instruction set, and provide shared memory hardware mechanisms[9]. 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 an effective and relatively straightforward approach for programming general-purpose multi-cores. 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 efficiently implement a high throughput LDPC decoder. Although the power consumption of x86 implementations are incompatible with most of the embedded mobile systems, which makes them useful for simulation purposes only.

Mainly due to the demands for visualization technology in the gaming industry, the performance of GPUs has significantly improved over the last decade. With many cores driven by a considerable memory bandwidth, recent GPUs are also being targeted for solving computationally intensive algorithms in a multi-threaded 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 to develop LDPC decoders on GPUs. As an example, the authors in [17] have achieved almost 1Gbps of throughput 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 - small processors, tiny memory, and low power. In other words, the limited available resources must be used in the most effective and efficient fashion.

To solve this issue, ARM-based SDR systems have been proposed in recent years [11,13] with the goal of developing an SDR based LDPC decoder that pro-

vides 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 report proposes an LDPC decoder for an embedded device which exploits GPU resources on the device. The structure of the proposed decoder is based on multiple GPU 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 will be limited.

The remainder of the report is structured as follows. Section 2 briefly introduces the LDPC error correcting codes and their decoding algorithms. Then the proposed heterogeneous algorithm on embedded mobile targets is described in Section 3. Finally, section 4 gives experimental results and some comparisons with other ARM implementations.

2 LDPC codes and their Decoding Processes

LDPC codes are a class of linear block codes with a very sparse parity check matrix called H-matrix. Their main advantage is that they provide a performance which is very close to that of the channel capacity for various wireless channels. Furthermore, they are suited for implementations that make heavy use of parallelism [20].

Here, we present a very 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].

[1]The reader is refered to [20] for more information.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{1}$$

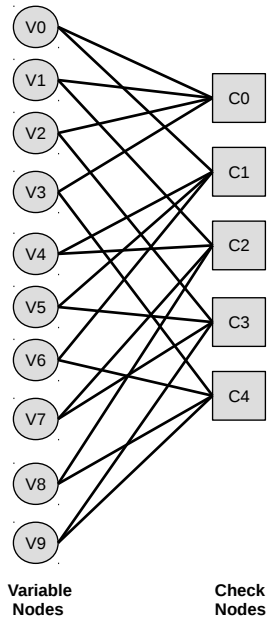

Figure 1: An example of Tanner graph

The general decoding algorithm of LDPC codes is based on the standard two-phase message passing (TPMP) principle described in [9]. 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. Due to transcendental operations and relying of message passing algorithm to the estimation of noise standard deviation, in practice Min-Sum (MS) variants are prefered by designers [13]. This algorithm is provided in Algorithm 1.

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 update loop of the variable nodes, will not start until all check nodes are updated. This characteristic affects the efficiency of parallel implementation of such algorithm.

Algorithm 1 Min-Sum algorithm

Loop 1: Initialization
for all $m \in C, n \in V$ **do**
  $Lq_{nm} = 0$
end for
for all $t = 1 \rightarrow (iter\_max)$ **do**
  Loop 2: LLR of message CN_m to VN_n
  for all $m \in C, n \in V$ **do**
    $\alpha_{nm} \triangleq sign(Lq_{nm})$,
    $\beta_{nm} \triangleq |Lq_{nm}|$,
    $Lr_{mn} = \prod_{n' \in N(m) \backslash n} \alpha_{n'm} \min_{n' \in N(m) \backslash n} \beta_{n'm}$.
  end for
  Loop 3: LLR of message VN_n to CN_m
  for all $m \in C, n \in V$ **do**
    $Lq_{nm} = LP_n + \sum_{m' \in M(n) \backslash m} Lr_{m'n}$.
  end for
end for
Loop 4: Hard decision from soft-values
for all $n \in V$ **do**
  $LQ_n = LP_n + \sum_{m' \in M(n)} Lr_{m'n}$,
  $\forall n, \hat{c} = [LQ_n] > 0$.
end for

Due to poor parallel mapping of Min-Sum algorithm, more efficient schedules, such as horizontal layered-based decoding algorithm, are proposed which allow updated imformation to be utilized more quickly in the algorithm, thus, speeding up decoding[18, 21]. 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 2x faster convergence for a given error rate. By using this method they have reduced memory bits usage by 45-50%. The layered decoding (Algorithm 2), which is used in the proposed algorithm here, can be summarized as follow:

- (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.

Algorithm 2 Horizontal Layered Min-Sum algorithm

Loop 1: Initialization
for all $m \in C, n \in N(m)$ **do**
  $Lr_{mn}^{(0)} = 0$
end for
for all $t = 1 \rightarrow (iter\_max)$ **do**
  Loop 2: For each check node
  for all $m \in C$ **do**
    for all $n \in N(m)$ **do**
      $Lr_{nm}^{(t)} = E_n - Lr_{nm}^{(t-1)}$
    end for
    for all $n \in N(m)$ **do**
      $\alpha_{nm} \triangleq sign(Lr_{nm}^{(t)})$,
      $\beta_{nm} \triangleq |Lr_{nm}^{(t)}|$,
      $Lr_{mn}^{(t)} = \prod_{n' \in N(m) \setminus n} \alpha_{n'm} \min_{n' \in N(m) \setminus n} \beta_{n'm}$.
    end for
    for all $n \in N(m)$ **do**
      $E_n = Lr_{nm}^{(t)} + Lr_{mn}^{(t)}$
    end for
  end for
  Loop 3: Hard decision
  for all $n \in V$ **do**
    $\forall n, \hat{c} = [E_n] > 0$
  end for
end for

This project implements 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 less resources of the ARM processor is used for decoding compared to similar work in [13]. Thus, there will be processing power left for other applications that need to be supported by the ARM processor. On the other hand, since the GPU and ARM of a mobile device are sitting on a same die, the latency issues in [17] are improved.

3 Algorithm Mapping

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

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

Considering these, 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 proposed algorithm.

3.1 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:

- First parallelism level is located inside the check node computations (Algorithm 2, loops located at line 8, 11 and 16). It is possible to execute such computations in parallel. However, this atomic parallelism level is hard to exploit due to the low complexity of computations.
- Second parallelism level is located at the check node level (Algorithm 2, line 7). 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.
- Third 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 3.4 the parallel decoding of F frames is referred to kernel 2 for sake of simplicity.

3.2 Target Architecture

In this study, we focus on mobile embedded devices equipped with ARM and GPU processors. One example is Jetson K1 SoCs which consists of a 4-core Cortex-A15 and an NVIDIA GK20a GPU processor.

An ARM Cortex-A15 processor is composed of multiple integer and floating point pipeline stages and is capable of running multiple concurrent threads. As in terms of GPU features, concurrent kernel execution capability of this device is most used in the proposed mapping. To achieve high throughput performance on such a low-power embedded processors, SIMD and SIMT programming models are exploited in the proposed LDPC decoder. Although, the proposed structure is applicable to other heterogeneous structures too, and is not limited to embedded devices.

3.3 Data Interleaving/Deinterleaving

Recal that the implementation of the parallel frame processing is subject to massive irregular memory access due to the structure of H-matrix. As a matter of fact, to process the same VN_i 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. This reordering is shown in Fig.2 [4]. In the proposed structure, interleaving and deinterleaving of frames are called kernel 1 and kernel 3.

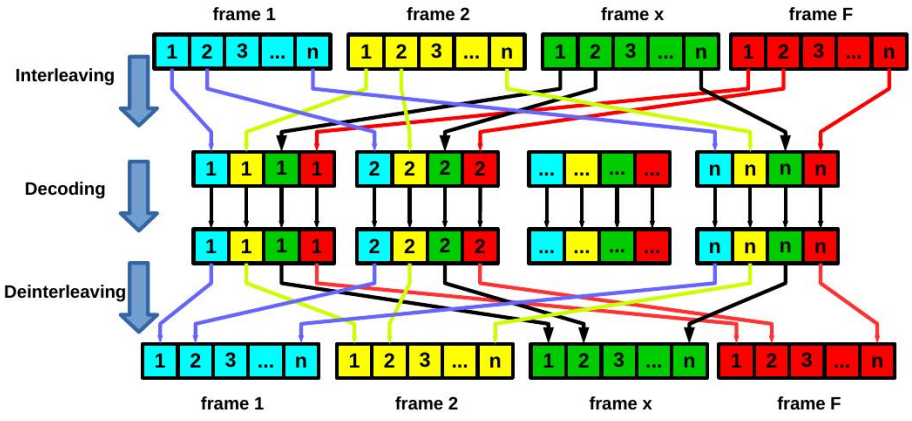

Figure 2: Data interleaving/deinterleaving process[4]

3.4 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×F frames at the same time. The whole LDPC decoder system model is shown in Fig.3.

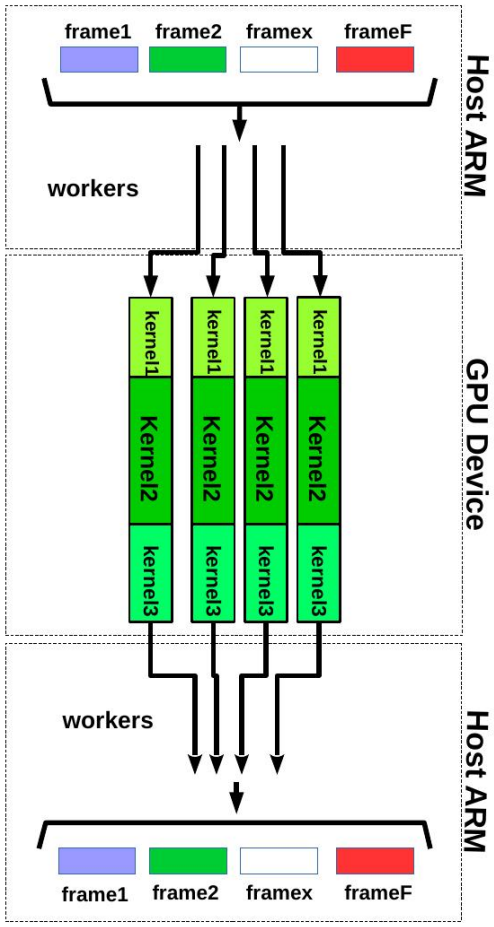

Figure 3: LDCP Decoder Data Flow

4 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.

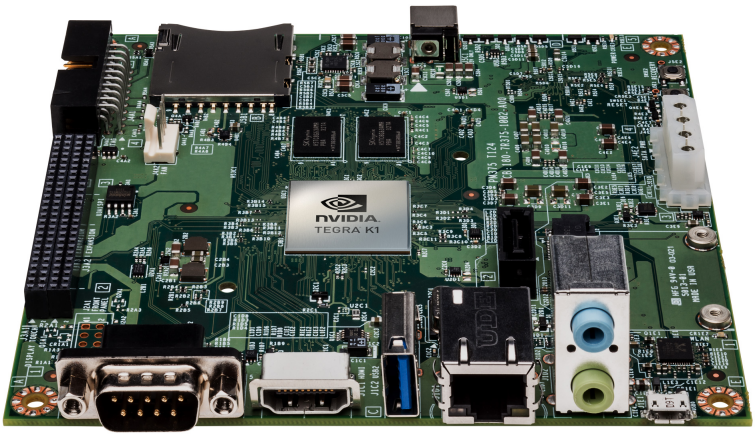

Figure 4: Tegra-TK1 Development Board

4.1 Performance Evaluation of Proposed Algorithm

First set of experiments was performed to achieve air throughput using different codes. The codes have different frame lengths: 576 to 9972. The air throughputs are provided in Fig.5 when {1,3} threads are used to handle {1,3} GPU streams. Measurements are performed for LDPC decoders that execute 10 layered-base decoding iterations.

Performance of one stream (or one thread) achieves 25 Mbps, while with 3 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 about 2×2.4 ms, interleaving steps need about 2×5 ms and decoding takes about 150 ms. So 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, which has been reduced here to 4.8 ms because of the architecture of embedded mobile device. On the other hand, with introducing 3 streams to GPU, its processing capacity is used more effectively which results to about 30% throughput improvement in most of our experiments.

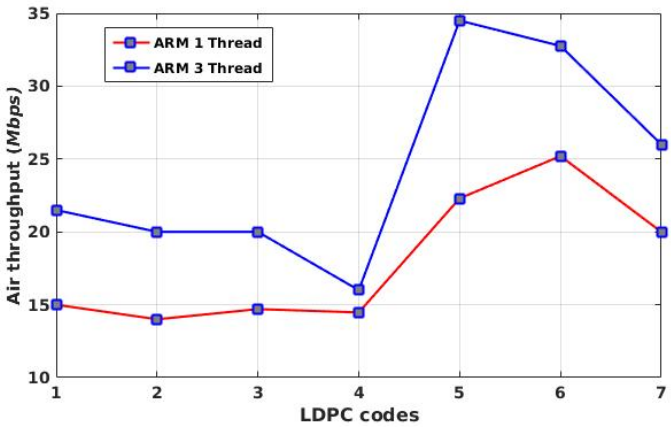

Figure 5: Measured throughputs for 10 layered decoding iterations (1-7 LDPC codes: $576 \times 288, 1024 \times 512, 1200 \times 600, 1944 \times 722, 4000 \times 2000, 8000 \times 4000, 9972 \times 4086$)

4.2 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 1. 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 1, 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 1: Throughput (Mbps) Comparison With Related Work

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

4.3 Performance Comparison on Different GPU Devices

GPU devices have different characteristics: amount of stream multiprocessors, CUDA cores, working frequencies, etc. A GPU based algorithm should have the scalability to use all the processing capability of a GPU device. In Table 2, 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 performance in Table 2 shows 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.

Table 2: Throughput (*Mbps*) of algorithm on different GPUs

code	target	iter	*Mbps* (1-thread)	*Mbps* (3-thread)
(576,288)	**GT540M**	**10**	**23**	**33**
		5	**44**	**61**
	K620	**10**	**23**	**30**
		5	**45**	**61**
	GTX680	**10**	**94**	**127**
		5	**163**	**217**
	TeslaK20	**10**	**66**	**90**
		5	**123**	**165**
(2304,1152)	**GT540M**	**10**	**25**	**37**
		5	**47**	**63**
	K620	**10**	**23**	**31.5**
		5	**47**	**63**
	GTX680	**10**	**94**	**132**
		5	**170**	**226**
	TeslaK20	**10**	**66**	**94**
		5	**127**	**170**
(4000,2000)	**GT540M**	**10**	**27**	**34**
		5	**27**	**37**
	K620	**10**	**24**	**32**
		5	**44**	**60**
	GTX680	**10**	**98**	**131**
		5	**164**	**230**
	TeslaK20	**10**	**73**	**98**
		5	**139**	**196**

5 Conclusion

An 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 desirable throughputs on embedded mobile

devices. Experimental results demonstrate that 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.

References

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 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 Communications Letters*, 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 Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 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 Communications Letters*, 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 Journal on Wireless Communications and Networking*, vol. 2011, no. 1, pp. 1–8, 2011. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2011-172

[9] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 309–322, Feb 2011.

[10] 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

[11] S. Grnroos and J. Bjrkqvist, "Performance evaluation of LDPC decoding on a general purpose mobile cpu," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, Dec 2013, pp. 1278–1281.

[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 Communications Letters*, vol. 19, no. 11, pp. 1861–1864, Nov 2015.

[14] H. Kim and R. Bond, "Multicore software technologies," *IEEE Signal Processing Magazine*, 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 (Scientific and Engineering Computation)*. 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. [Online]. Available: http://doi.acm.org/10.1145/2538668

[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] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, Oct 2004, pp. 107–112.