Heterogeneous LDPC Decoder Algorithm on ARM and GPU of Mobile Devices

Roohollah Amiri
Department of Electrical and
Computer Engineering
Boise State University
Email: roohollahamiri@boisestate.edu

Hani Mehrpouyan
Department of Electrical and
Computer Engineering
Boise State University
Email: 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 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, unlike prior solutions that are based on either graphic processing units (GPUs) or advanced RISC machine (ARM) architectures, we propose a heterogeneous LDPC decoder that uses both the ARM and GPU processors of a mobile device to achieve efficient real-time decoding. The proposed solution is implemented on an NVIDIA development kit, where our results indicate that we can reduce the load on either the GPU or the ARM processor through the proposed heterogeneous structure, which in turn allows these resources to support other applications, simultaneously.**

I. 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 WiFi (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 decoding algorithms are computationally-intensive and need powerful computer architecture 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 fully be 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). As in terms of multi-core architectures, researchers have used CPUs [10], [11], 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 peorformance 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 a 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 OpenMP [15] which provides an effective and relatively straightforward approach for programming general-purpose multi-cores. In [8], OpenMP is used to generate address patterns as a part of their LDPC decoder structure. 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.

Mainly due to the demands for visualization technology in the gaming industry, the performance of graphics processing units (GPUs) has significantly improved over the last decade. With many cores driven by a considerable memory bandwidth, recent GPUs are 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], [19] and Open Computing Language (OpenCL) [18] platforms to develop LDPC decoders on GPUs.

Due to large computing capacity of CPUs and GPUs, software based LDPC decoders have met the required throughputs of communication standards, although in wireless communication the end user's device is ususally a mobile device with an embedded system inside. Unlike today's powerful multi-core systems, embedded systems must operate on limited resources - small processors, tiny memory and low power. With an embedded system, it is about doing as much as you can with as little as you can. 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 provides high throughput and low latency on a low-power embedded system. The authors in [13] have used ARM processor's SIMD and SIMT programming models to implement an LDPC decoder that is based on parallel decoding of data frames. 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 but mobile devices need to support multiple applications simultaneously. 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 considers limited resources of a mobile device and proposes an algorithm to reach high throughput and low latency decoding on an embedded device.

The contributions of this paper are twofold. First, it proposes an LDPC decoder for an embedded device which exploits ARM SIMD units and GPU of the device together. This allows these resources to also support other applications on a mobile device. Second, as a consequence of this, by using the GPU of a mobile device, less memory from ARM processor is being used which is typically limited in an embedded system. The remainder of the paper is structured as follows. Section II briefly introduces the LDPC code family and its decoding algorithms. Then the proposed heterogeneous algorithm on embedded targets is described in Section III. Finally, IV gives experimental results and some comparisons with other ARM implementations.

II. 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 the channel capacity for a lot of different channels and linear time complex algorithms for decoding. Furthermore, they are suited for implementations that make heavy use of parallelism.

Basically there are two ways to represent LDPC codes. Like all linear block codes they can be described by their H-matrix, while they can be represented by a Tanner graph which is a bipartite graph. An LDPC graph consists of a set of variable nodes (VNs) , a set of check nodes (CNs), 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. Figure II illustrate the equivalent Tanner graph for $(10, 5)$ LDPC code with H-matrix in (1).

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

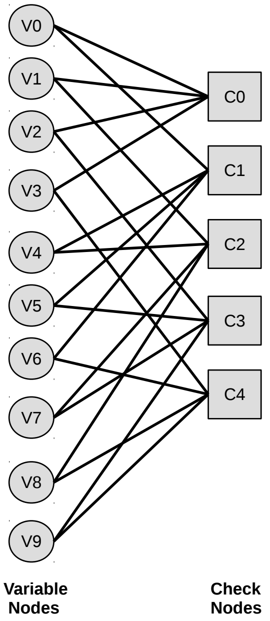

Fig. 1. An example of Tanner graph

Many works as in [9], [11], [17], [18] focused on mapping LDPC decoders on multi-core architectures. Most of these works are based on the standard two-phase message passing (TPMP) schedule 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.

More efficient layered schedules, such as horizontal layered-based decoding algorithm, allow updated imformation to be utilized more quickly in the algorithm, thus, speeding up the decoding [19], [21]. In fact, the H-matrix can be viewed as a layered graph that is decoded sequentially. The work

Algorithm Flooding Min-Sum algorithm

```
Kernel 1: Initialization
for all $m \in C, n \in V$ do
    $Lq_{nm} = 0$
end for
for all $t = 1 \rightarrow (iter\_max)$ do
    Kernel 2: LLR of message $CN_m$ to $BN_n$
    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
    Kernel 3: LLR of message $BN_n$ 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
Kernel 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
```

Algorithm Horizontal Layered Min-Sum algorithm

```
Kernel 1: Initialization
for all $m \in C, n \in V$ do
    $Lr_{mn} = 0$
end for
for all $t = 1 \rightarrow (iter\_max)$ do
    Kernel 2: Process each CN one after another
    for all $m \in C$ do
        for all $n \in V$ do
            $\alpha_{nm} \triangleq sign(Lq_{nm})$,
            $\beta_{nm} \triangleq |Lq_{nm}|$,
            $Lr'_{mn} = \prod_{n' \in N(m) \backslash n} \alpha_{n'm} \min_{n' \in N(m) \backslash n} \beta_{n'm}$.
            {$Lr'_{mn}$ is the new calculated massage to $BN_n$}
        end for
    end for
    for all $n \in V$ do
        $LQ'_n = LQ_n - Lr_{mn} + Lr'_{mn}$
    end for
end for
Kernel 4: Hard decision from soft-values
for all $n \in V$ do
    $\forall n, \hat{c} = [LQ_n] > 0$.
end for
```

in [17] has applied a form of layered belief propagation to irregular LDPC codes to reach 2x faster convergence in a given error rate. By using this method they have reduced memory bits usage by 45-50%. The layered decoding (Algorithm 2) can be summarized as follow: all values for the check node computations are computed using variable node massages linked to them. Once, a check node is calculated, the corresponding variable nodes are updated immediately after receiving massages. This process is repeated to the maximum number of iterations.

In algorithm 1, TPMP, the kernels 2 and 3 are updated by seperate processing and passed to each other iteratively. It means that variable nodes update, will not start until all check nodes are updated. Consdering that, algorithm 2, horizontal layered decoding, is composed of a single loop kernel with some data dependencies between consecutive loop iterations and in each iteration, the horizontal layers are processed sequentially from the top to the bottom layer [22].

Although layered algorithm is composed of a single loop kernel composed to two sequential kernels in standard algorithms, it has an irregular memory access as a major limitation. To solve this irregular memory access, a data interleaving/deinterleaving process is used before and after the decoding process in [13], [17], which is used in the proposing algorithm too. In [17], the GPU decoder achieves high throughputs but its latency that goes beyond seconds makes it suitable for simulation purposes only. On the other hand, the ARM decoder proposed in [13] uses all computing resources (4 existing cores) for LDPC decoding and does not take advantage of GPU processing on mobile devices. This paper uses one single core of ARM and the GPU device of a mobile processor to implement a high throughput and low latency LDPC decoder. By using one core of the ARM processor, there will be processing power for other applications of a mobile device and less memory of ARM processor will be used for decoding. 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.

III. 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: (i) the number of computations respect to number of memory access is low; (ii) the data reusing between consecutive computations is low; (iii) it requires a large set of irregular memory access due to H-matrix structure [4]. Considering these, an software-based decoder should take advantage of different parallelism levels offered by the target architecture to achieve high throughput efficiency. In this section, first the target architecture's features are presented. Then the decoding algorithm is detailed.

A. Target architecture

In this study, we focused on embedded devices equipped with ARM and NVIDIA GPU processors. One example is Jetson K1 SoCs which contains a 4-core Cortex-A15 and a Kepler GPU processor.

ARM Cortex-A15 processor is composed of 15 integer/17-25 floating point pipeline stages. To limit the level of memory access latency, multi-level caches were introduced. Cortex-A15 has 32 KB data + 32 KB instruction L1 cache per core and an integrated low-latency L2 cache controller, up to 4 MB per cluster. Each core includes a NEON SIMD unit which can execute up to 16 computations on 8-bit data simultaneously. Tegra K1 features a single SMX (in a single GPC), which amounts to 192 CUDA cores using Kepler technology. Concurrent kernel execution capability of this device is used in the proposed mapping. To achieve high throughput performance on such a low-power embedded processors, the two programming models are exploited in the proposed LDPC decoder. In the next subsections, we detailed the different parallelism levels along with the implementation choices.

B. Parallelism levels in the proposed algorithm

To achieve high throughput performance, a software LDPC decoder has to exploit computation parallelism for taking advantage of multi-core architectures. Different parallelism levels are present in a layered decoding algorithm. **First parallelism level** is located inside the CN computations (Algorithm 2, loops located at line 8 and 15). 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 CN level (Algorithm 2, line 7). Two CN computations can be done in parallel if there is data dependency. Thus this level is hard to exploit and inefficient. **Third parallelism level** is located at frame level (Complete Algorithm 2). A same computation is sequence is executed over consecutive frames. This approach provides an efficient parallel processing algorithm. So we have used single instruction multiple data (SIMD) programming model to decode F frames in parallel.

C. Data Interleaving/deinterleaving

The implementation of the parallel frame processing is subject to massive irregular memory access due to 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 decoding stage. So channel frames coming from F frames are reordered to achieve an aligned memory data structure. This reordering is done on SIMD units of ARM processor. This reordering is shown in Figure 2.

D. Stream based parallelism

SIMT programming model is used to decode W sets of F frames concurrently, with W the number of concurrent streams on the GPU device. This multi-core programming is specified by CUDA API. Each gpu stream is controlled by a *pthread* called *worker*. Each *worker* is responsible for its own sets of frames. By using stream based processing, the system can decode WxF frames at the same time. The whole LDPC decoder system model is shown in Figure 3.

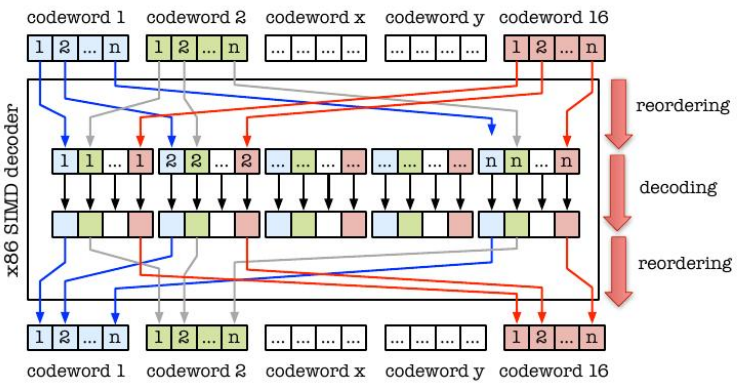

Fig. 2. Data interleaving/deinterleaving process

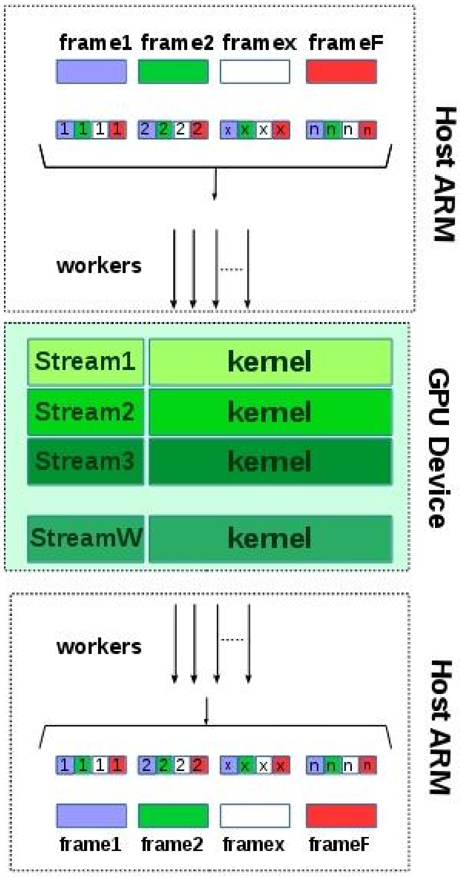

Fig. 3. LDCP Decoder Data Flow

IV. Experimental Results

The experiments were carried out by decoding LDPC codes using NVIDIA Tegra K1 Socs. The programs compiled with 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-gdacac96.

V. Conclusion

The conclusion goes here.

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*, 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] 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.

[19] 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

[20] Cuda homepage. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[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.

[22] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic ldpc codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985–994, August 2009.