EURASIP Journal on
Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH **Open Access**

Systematic construction, verification and implementation methodology for LDPC codes

Hui Yu*, Jing Cui, Yixiang Wang and Yibin Yang

Abstract

In this article, a novel and systematic Low-density parity-check (LDPC) code construction, verification and implementation methodology is proposed. The methodology is composed by the simulated annealing based LDPC code constructor, the GPU based high-speed code selector, the ant colony optimization based pipeline scheduler and the FPGA-based hardware implementer. Compared to the traditional ways, this methodology enables us to construct both decoding-performance-aware and hardware-efficiency-aware LDPC codes in a short time. Simulation results show that the generated codes have much less cycles (length 6 cycles eliminated) and memory conflicts (75% reduction on idle clocks), while having no BER performance loss compared to WiMAX codes. Additionally, the simulation speeds up by 490 times under float precision against CPU and a net throughput 24.5 Mbps is achieved. Finally, a net throughput 1.2 Gbps (bit-throughput 2.4 Gbps) multi-mode LDPC decoder is implemented on FPGA, with completely on-the-fly configurations and less than 0.2 dB BER performance loss.

Keywords: low-density parity-check codes, simulated annealing, ant colony optimization, graphic processing unit, decoder architecture

1. Introduction

Low-density parity-check (LDPC) code is first proposed by Gallager [1] and rediscovered by Mackay and Neal since they introduce Tanner Graph [2] into LDPC code [3]. LDPC code with soft decoding algorithms on Tanner Graph can achieve outstanding capacity and approach Shannon limit over noisy channels at moderate decoding complexity [4]. Most algorithms root from the famous believe propagation (BP) algorithm, such as min-sum algorithm (MSA) with simplified calculation, modified MSA (MMSA) [5] with improved BER performance and layered versions [6] with fast decoding convergence.

The existence of "cycle" in Tanner Graph is a critical constraint of the above algorithms, as it breaks the "message independence hypothesis" and degrades the BER performance. As a result, "girth" becomes an importance metric of estimating the performance of the LDPC code. The progressive edge-growth (PEG) algorithm [7] is a girth-aware construction method that tries to make shortest cycle as large as possible. Approximate cycle extrinsic (ACE) message degree constraint is further combined into PEG [8] to lower error floor. However, these performance-aware methods do not take hardware implementation into account, which usually result in low efficiency or high complexity.

As to the decoder implementation, the fully-parallel architecture [9] is first proposed for achieving the highest decoding throughput, but the hardware complexity due to the routing overhead is very high. The semi-parallel layered decoder [10] is then proposed to achieve the trade-off between hardware complexity and decoding throughput. Memory conflict is a critical problem for layered decoder, which is modeled as a single-layer traveling salesman problem (TSP) in [11]. However, this model ignores "element permutation", i.e., the order assignment of the edges in each layer, and its search does not cover the entire solution space. Further, fully-parallel graphic processing unit (GPU) based implementation is also proposed in [12].

In this article, a novel and systematic LDPC code construction, verification, and implementation methodology is proposed, and a software and hardware platform is implemented, which is composed by four modules as shown in Figure 1. The simulated annealing (SA) based LDPC code constructor continuously constructs good candidate codes. The BER performance of the generated

* Correspondence: yuhui@sjtu.edu.cn
Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, P. R. China

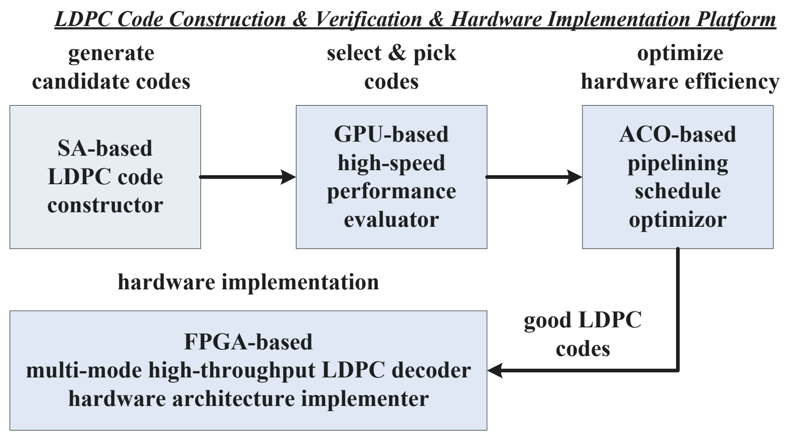
LDPC Code Construction & Verification & Hardware Implementation Platform
generate candidate codes
select & pick codes
optimize hardware efficiency
SA-based LDPC code constructor
GPU-based high-speed performance evaluator
ACO-based pipelining schedule optimizor
hardware implementation
good LDPC codes
FPGA-based multi-mode high-throughput LDPC decoder hardware architecture implementer

Figure 1 LDPC code construction and verification platform.

codes, especially the error floor, is then evaluated by the high-speed GPU based simulation platform. Next, the hardware pipeline of the selected codes are optimized by the ant colony optimization (ACO) based scheduling algorithm, which can reduce much of the memory conflicts. Finally, detailed implementation schemes are proposed, i.e., *reconfigurable switch network* (adopted by [13]), *offset-threshold decoding, split-row MMSA core, early-stopping scheme* and *multi-block scheme*, and the corresponding multi-mode high-throughput decoder of the optimized codes is implemented on FPGA. The novelties of the proposed methodology are listed as follows:

- Compared to traditional methods (PEG, ACE), the SA-based constructor takes both decoding performance and hardware efficiency into consideration during construction process.
- Compared to existed work [11], the ACO-based scheduler covers both layer and element permutation, and maps the problem to a double-layer TSP, which is a complete solution and can provide better pipelining schedule.
- Compared to existed works, the GPU-based evaluator first implements the semi-parallel layered architecture on GPU. The obtained net throughput is similar to the highest report [12] (about 25 Mbps), while the proposed scheme has higher precision and better BER performance. Further, we put the whole coding and decoding system into GPU rather than a single decoder.
- Compared to existed FPGA or ASIC implementations [14-16], the proposed multi-mode high-throughput decoder not only supports multiple modes with completely on-the-fly configurations, but also has a performance loss within 0.2 dB against float precision and 20 iterations, and a stable *net-throughput* 721.58 Mbps under code rate 1/2 and 20 iterations. With early-stopping scheme, a *net-throughput* 1.2 Gbps is further achieved on Stratix III FPGA.

The remainder of this paper is organized as follows. Section 2 presents the background of our research. Sections 3, 4, and 5 introduces the ACO based pipeline scheduler, the SA based code constructor and the GPU based performance evaluator, respectively, followed by hardware implementation schemes and issues of the multi-mode high-throughput LDPC decoder discussed in Section 6. Simulation results are provided in Section 7 and hardware implementation results are given in Section 8. Finally, Section 9 concludes this article.

2. Background

2.1. LDPC codes and Tanner graph

An LDPC code is a special linear block code, characterized by a sparse parity-check matrix $\mathbf{H}$ with dimensions $M \times N$; $\mathbf{H}_{j,i}= 1$ if code bit i is involved in parity-check equation j, and 0 otherwise. An LDPC code is usually described by its Tanner Graph, a bipartite graph defined on the code bit set $\mathbb{R}$ and parity-check equation set $\mathbb{C}$, whose elements are called a "bit node" and a "check node", respectively. An edge is assigned between bit node BN_i and check node CN_j if $\mathbf{H}_{j,i} = 1$. A simple 4×6 LDPC code and the corresponding Tanner Graph is shown in Figure 2.

Quasi-cyclic LDPC codes (QC-LDPC) is a popular class of structured LDPC codes, which is defined by its base matrix $\mathbf{H}^b$, whose elements satisfying $-1 \le \mathbf{H}^b_{j,i} < z_f . z_f$ is called the expansion factor. Each element in the base matrix should be further expanded to a $z_f \times z_f$ matrix to obtain $\mathbf{H}$. The elements $\mathbf{H}^b_{j,i} = -1$ are expanded to zero matrices, while $L(Q_i) = L(c_i) + \sum_{j' \in c_i} L(r_{j'i}) = L(q_{ij}) + L(r_{ji})$

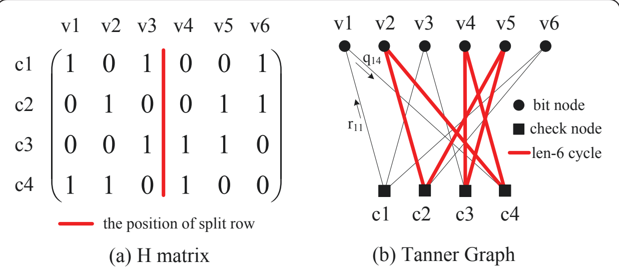

Figure 2 H matrix and Tanner Graph of LDPC.

are expanded to a cyclic-shift identity matrices with permutation factors $\mathbf{H}^b_{j,i} \geq 0$. QC-LDPC is naturally available for layered algorithms, whose j-th row is exactly layer j. We call the "1"s of j-th row as the set $p = \mathbf{H}^b_{j,i}$. See Figure 3 for an example of a 4 × 6 base matrix with $z_f = 4$.

2.2. The BP algorithm and effect of cycle

The BP algorithm is a general soft decoding scheme for codes described by Tanner Graph. It can be viewed as the process of iterative message exchange between bit nodes and check nodes. For each iteration, each bit node or check node collects the messages passed from its neighborhood, updates its own message and passes the updated message back to its neighborhood. BP algorithm has many modified versions, such as log-domain BP, MSA, and layered BP. All of them originate from the basic log-domain message passing equations, given as follows.

$$\{\mathbf{H}^b_{j,i} \,|\, \mathbf{H}^b_{j,i} \geq 0\} \tag{1}$$

$$L(q_{ij}) = L(Q_i) - L(r_{ji}) \tag{2}$$

$$L(Q_i) = L(c_i) + \sum_{j' \in c_i} L(r_{j'i}) = L(q_{ij}) + L(r_{ji}) \tag{3}$$

where $L(c_i)$ is the initial channel message, $L(q_{ij})$ is the message passing from BN_i to CN_j, $L(r_{ji})$ is the message of inverse direction, and $L(Q_i)$ is the a-posteriori of bit node BN_i. $\mathcal{C}_i$ is the neighbor set of BN_i, $\mathcal{R}_j$ is the neighbor set of CN_j. $\Phi(x) = \log \frac{e^x + 1}{e^x - 1}$. These equations can also be applied in layered BP, the difference is that the $L(q_{ij})$ and $L(r_{ji})$ should be updated in each layer of the iteration.

The above equations requires the independence of all the messages $L(q_{i'j})$, $i' \in \mathcal{R}_j$ and $\mathbf{H}^b_{j,k}$. However, the existence of "cycle" in Tanner Graph invalidates this independence assumption, thus degrades the BER performance of BP algorithm. A length 6 cycle is shown with bold lines in Figure 2. In this case, if BP algorithm proceeds for more than three iterations, the receive messages of the involved bit nodes v_2,v_4,v_5 will partly contain its own message sent three iterations before. For this reason, the minimum cycle length in the Tanner Graph, called "girth", has a strong relationship with its BER performance, and is considered as an important metric in LDPC code construction algorithms (PEG, ACE) [7,8].

2.3. Decoder architecture and memory conflict

The semi-parallel structure with layered MMSA core is a popular decoder architecture due to its good tradeoff among low complexity, high BER performance and high throughput. As shown in Figure 4, the main components in the top-level architecture include an LLRSUM RAM storing $L(Q_i)$, an LLREX RAM storing $L(r_{ji})$ and a layered MMSA core pipeline. The two RAMs should be readable and writable. Old values of $L(Q_i)$ and $L(r_{ji})$ are read, and new values are calculated through the pipeline and written back to RAMs. For QC-LDPC codes, the values are processed layer by layer, and the "1"s in each layer is processed one by one.

Memory conflict is a critical problem that constrains the throughput of the semi-parallel decoder. Essentially, memory conflict occurs when the read-after-write (RAW) dependency of $L(Q_i)$ is violated. Note that the new value of $L(Q_i)$ will not be written back to RAM until the pipelined calculation finishes. If $L(Q_i)$ is again needed during this calculation period, the old value will be read, while the new one is still under processing, see $L(Q_6)$ in Figure 4. This case happens when the layers j and $j + l$ have "1"s in the same position i $(\mathbf{H}^b_{j,i} \geq 0, \mathbf{H}^b_{j+l,i} \geq 0)$. We call it a gap-$l$ conflict.

Memory conflict slows the decoding convergence and thus reduces the BER performance. The traditional method of handling memory conflict is to insert idle clocks in the pipeline, with the cost of throughput reduction. It's obvious that the smaller l, the more idle clocks should be inserted, since the pipeline need to wait at least K stages before writing back the new values. Usually, the number of gap-1, gap-2, gap-3

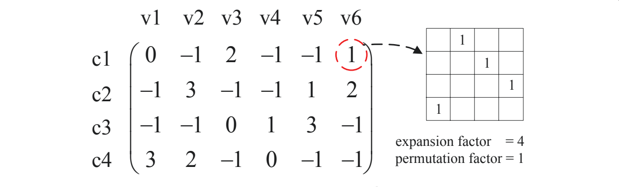

Figure 3 A simple 4 × 6 base matrix H^b with z_f = 4.

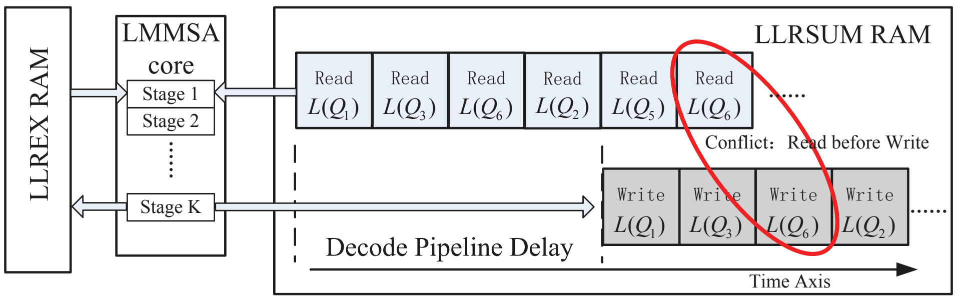

Figure 4 The layered MMSA decoder architecture and memory conflict.

conflicts, denote c_1, c_2 and c_3, are considered as the metrics of measuring memory conflict.

3. The ACO-based pipelining scheduler

In this section, we propose the ACO-based pipeline scheduling algorithm to minimize memory conflict. We first formulate this problem, then map it to the double-layered TSP and finally use ACO to solve it.

3.1. Problem formulation

Consider a QC LDPC code described by its base matrix **H** with dimensions $M \times N$. Thus, there are M layers. Denote w_m,$1 \leq m \leq M$ as the number of elements ("1"s) in m-th layer. Denote $h_{m,n}$, $1 \leq n \leq w_m$ as the column index in **H** of the n-th element, m-th layer. Additionally, we assume the core pipeline is K stages.

As discussed above, the decoder processes all the "1"s in **H** exactly once by processing layer-by-layer in each iteration, and element-by-element in each layer. However, the order can be arbitrary, which enables us to schedule the elements carefully to minimize memory conflict. We have two ways to solve it.

- *Layer permutation:* We can assign which layer to be processed first and which to be next. If two layers i,j have 1s at totally different positions, i.e., such j,l do not exist that $h_{i,k} = h_{j,l}$, they tend to be assigned as the adjacent layers with no conflict.
- *Element permutation:* In a certain layer, we can assign which element to be processed first and which to be next. If two adjacent layers i,j still have conflict, i.e., $h_{i,k} = h_{j,l}$ for some k,l, then we can assign element k to be first in layer i, and l to be last in layer j. By this way, we increase the time interval between the conflicting elements k and l.

Therefore, the memory conflict minimization problem is exactly a scheduling problem, in which layer permutation and element permutation should be designed to minimize the number of idle pipeline clock insertions. We denote layer permutation as $m \rightarrow \lambda_m$, $1 \leq m$, $\lambda_m \leq M$, and element permutation of layer m as $n \rightarrow \mu_{m,n}$, $1 \leq n,\mu_{m,n} \leq w_m$.

Based on the above definitions, a memory conflict occurs between layer i, element k and layer j, element l if the following conditions are satisfied: (1) layers i,j are assigned to be adjacent, i.e., $\lambda_j = \lambda_i + 1$; (2) $h_{i,k} = 1$ and $h_{j,l} = 1$; (3) the pipeline time interval is less than pipeline stages, i.e., $w_i - \mu_{i,k} + \mu_{j,l} \leq K$. Further, we define the "conflict set" $\mathcal{C}$ as $\mathcal{C}(i,j) = \{(k,l) | \text{elements } (i,k) \text{ and } (j,l) \text{ cause a memory conflict}\}$, and the "conflict stages", also the minimum number of idle clocks inserted due to this conflict, as

$$c(i,k;j,l) = \max\{K - (w_i - \mu_{i,k} + \mu_{j,l}), 0\} \tag{4}$$

3.2. The double-layered TSP

This part introduces the mapping from the above memory conflict minimization problem to a double-layered TSP. TSP is a famous NP-hard problem, in which the salesman should find the shortest path to visit all the n cities exactly once and finally return to the starting point. Denote $d_{i,j}$ as the distance between city i and city j. TSP can be mathematically described as follows: given distance matrix $\mathbf{D} = [d_{i,j}]_{n\times n}$, find the optimal permutation of the city indices x_1,x_2, ...,x_n *to* minimize the loop distance,

$$\min \left(\sum_{i=1}^{n-1} d_{x_i,x_i+1} + d_{x_n,x1} \right) \tag{5}$$

Compared to layer permutation which can contribute most part of the memory conflict reduction, element permutation only deals with minor changes for the optimization when layer permutation is already determined. Therefore, we map the problem to a double-layered TSP, where layer permutation is mapped to the first layer, and element permutation is mapped to the second layer based on result of the first layer. Details are described as follows:

- *Layer permutation layer:* In this layer we only deal with layer permutation. We define the "distance", also "cost" between layers i and j as the minimum number of idle clocks inserted before the processing of layer j. If more conflict position pairs exist, i.e., $\left|\mathcal{C}(i,j)\right| > 1$, then we should take the maximum one. Thus in this layer, the distance matrix should be defined by

$$d_{i,j} = \max_{(k,l)\varepsilon\mathcal{C}(i,j)} \mathcal{C}(i,k;j,l) \tag{6}$$

and the target function remains the same as (5).

- *Element permutation layer:* In this layer we inherit the layer permutation result, and map element permutation of each layer to an independent TSP. In the TSP for layer i, we fix the schedule of the prior layer p ($\lambda_p = \lambda_i - 1$) and next layer q ($\lambda_q = \lambda_i + 1$), and only tune the elements of layer i. We define the "distance" $d_{k,l}$ as the change on the number of idle clocks if element k is assigned to the position l, i.e., $\mu_{i,k} = l$. Note that element k can conflict with layer p or q, and $d_{k,l}$ varies by different conflict cases, given by

$$d_{k,l} = \begin{cases} 0 & \text{both conflict or neither conflict} \\ k - l & k \text{ only conflict with layer } p \\ l - k & k \text{ only conflict with layer } q \end{cases} \tag{7}$$

Since the largest $d_{k,l}$ becomes the bottleneck of element permutation, the target function should change to the following max form:

$$\min\ \max\{d_{x_1,x_2}, d_{x_2,x_3}, \ldots, d_{x_{n-1},x_n},\ d_{x_n,x_1}\} \tag{8}$$

3.3. The ACO-based algorithm

This part introduces the ACO based algorithm to solve the double-layered TSP discussed above. ACO is a heuristic algorithm to solve computational problems which can be reduced to finding good paths through graphs. Its idea originates from mimicking the behavior of ants seeking a path between their colony and a source of food. ACO is especially suitable for solving TSP.

Algorithm 1 [see Additional file 1] gives the ACO-based double-layered memory conflict minimization algorithm. First we try layer permutation LAYER1_MAX times, and for each layer permutation, we try element permutation for LAYER2_MAX times. We record the pipeline schedule with smallest idle clocks as the best solution for this algorithm.

The detailed ACO algorithm for TSP is described in Algorithm 2. We try SOL_MAX solutions, and for each solution, all ants should finish CYCLE_MAX cycles, in which the shortest cycle is recorded as the best solution. One ant cycle is finished in VERTEX_NUM ant-move steps, where one step is consist of four sub-steps: *Ant Choose*, *Ant Move*, *Local Update and Global Update*. Further, the *Bonus* is rewarded to the shortest cycle. All specific parameters (e.g., p and ϕ) are referred to the suggestion of [17].

4. The SA-based code constructor

In this section, we propose a joint optimized construction algorithm that takes both performance and efficiency into consideration during construction the **H** matrix of the LDPC code. We first give the SA based framework and then discuss the details of the algorithm.

4.1. Problem formulation

We now deal with the classic code construction problem. Given the code length N, code rate R, and perhaps other constraints such as QC-RA type (e.g., WiMAX, DVB-S2), or fixed degree distribution (optimized by density evolution), we should construct a "good" LDPC code described by its **H** matrix that meets practical need. The word "good" here mainly have the following two metrics.

- *High performance*, which means the code should have high coding gain and good BER/BLER performance, including early water-fall region, low error floor and anti-fading ability. This is strongly related to large girth, large ACE spectrum, few trapping sets, and etc.
- *High efficiency*, which means the implementation of the encoder and decoder should have moderate complexity, and high throughput. This is strongly related to QC-RA type, high degree of parallelism, short decoding pipeline, few memory conflicts, and etc.

Traditional construction methods mainly focus on high performance of the code, such as PEG and ACE, which motivates us to find a joint optimized construction method concerning both performance and efficiency.

4.2. The double-stage SA framework

In this part, we introduce the double-stage SA [18] based framework for the joint optimized construction problem. SA is a generic probabilistic metaheuristic for the global optimization problem which should locate a good approximation to the global optimum of a given function in a large search space. Since our search space is a large 0-1 matrix space, denoted as $\{0, 1\}^{M\times N}$, SA is very useful for this problem.

Note that the performance metric is the more important metric for LDPC construction compared with efficiency metric. Therefore, we divide the algorithm into two stages, aiming at performance and efficiency, respectively, and regard performance as the major stage that should be satisfied first. For a specific target measured by "performance energy" e_1 and "efficiency energy" e_2, we set two thresholds: upper bound $e_{1h} = e_1$, and lower bound $e_{1l} < e_1$. The algorithm enters in the second stage when the current performance energy is less than e_{1l}. At the second stage, the algorithm ensures the performance energy to be not larger than e_{1h}, and try to reduce the e_2. Algorithm 3 shows the details.

4.3. Details of the algorithm

This part discusses the details of the important functions and configurations of Algorithm 3.

- *sample_temperature* is the temperature sampling function, decreasing with k. It can be an exponential form $\alpha e^{-\beta k}$.
- *prob* is the accept probability function of the new search point *h_new*. If *h_new* is better (*E_new* <*E*), it returns 1, otherwise, it decreases with *E_new*−*E*, and increases with t. It can be an exponential form $\alpha e^{-\beta}$ $(E\_new-E)/t$
- *perf_energy* is the performance energy function. It evaluates the performance related factors of the matrix h, and gives a lower energy for better performance. Typically, we can calculate the number of length-l cycles c_l, then calculate a total cost given by $\sum_l w_l c_l$, where w_l is the cost weight of a length-l cycle, decreasing with l.
- *effi_energy* is the efficiency energy function, similar as *perf_energy* except that it gives a lower energy for higher efficiency. Typically, we can calculate the the number of gap-l memory conflicts c_l, then calculate a total cost given by $\sum_l w_l c_l$, where w_l is the cost weight of a layer gap l conflict, decreasing with l.
- *perf_neighbor* searches for a neighbor of h in the matrix space when aiming at performance, which is based on minor changes of h. For QC LDPC, we can define three atomic operations for the base matrix $\mathbf{H}^b$ as follows.
    - *Horizontal swap:* For chosen row i,j and column k, l, swap values of $\mathbf{H}^b_{i,k}$ and $\mathbf{H}^b_{i,l}$, then swap values of $\mathbf{H}^b_{j,k}$ and $\mathbf{H}^b_{j,l}$.
    - *Vertical swap:* For chosen row i,j and column k, l, swap values of $\mathbf{H}^b_{i,k}$ and $\mathbf{H}^b_{j,k}$, then swap values of $\mathbf{H}^b_{i,l}$ and $\mathbf{H}^b_{j,l}$.
    - *Permutation change*: Change the permutation factor for chosen element $\mathbf{H}^b_{i,k}$.

  For a higher temperature t, we allow the neighbor searching process to search in a wider space. This is done by performing the atomic operations more times.
- *effi_neighbor* searches for a neighbor of h in the matrix space when aiming at efficiency. This is similar as *perf_neighbor*, however, typically we should remove the permutation change operation, as it does nothing to help reduce conflicts.

5. The GPU-based performance evaluator

In this section, we introduce the implementation of high-speed LDPC verification platform based on compute unified device architecture (CUDA) supported GPUs. We first give the architecture and algorithm on GPU, and then talk about some details.

5.1. Motivation and architecture

Compute unified device architecture is NVIDIA's parallel computing architecture. It enables dramatic increases in computing performance by executing multiple parallel independent and cooperated threads on GPU, thus is particularly suitable for the Monte Carlo model. The BER simulation of LDPC code is Monte Carlo since it collects huge amount of bit error statistics of the same decoding process, especially in the error floor region when the BER is low (10^{-7} to 10^{-10}). This motivates us to implement the verification platform on GPU where many decoders run parallel like hardware such as ASIC/FPGA to provide statistics.

Figure 5 shows our GPU architecture. CPU is used as the controller, which puts the code into GPU constant memory, raises the GPU kernels and gets back the statistics. While in GPU grid, we implement the whole coding system for each GPU block, including source generator, LDPC encoder, AWGN channel, LDPC decoder and statistics. Our decoding algorithm is layered MMSA. In each GPU block, we assign z_f threads to calculate new LLRSUM and LLREX of the z_f rows in each layer, where z_f is the expansion factor of QC LDPC. The z_f threads cooperate to complete the decoding job.

5.2. Algorithm and procedure

This part introduces the procedure that implements the GPU simulation, given by Algorithm 4. $P \times Q$ blocks run parallel, each simulating an individual coding system, where P is the number of multiprocessors (MP) on the device and Q is the number of cores per MP. In each system, z_f threads cooperatively do the job of encoding, channel and decoding. When decoding, the threads process data layer after layer, each thread performing

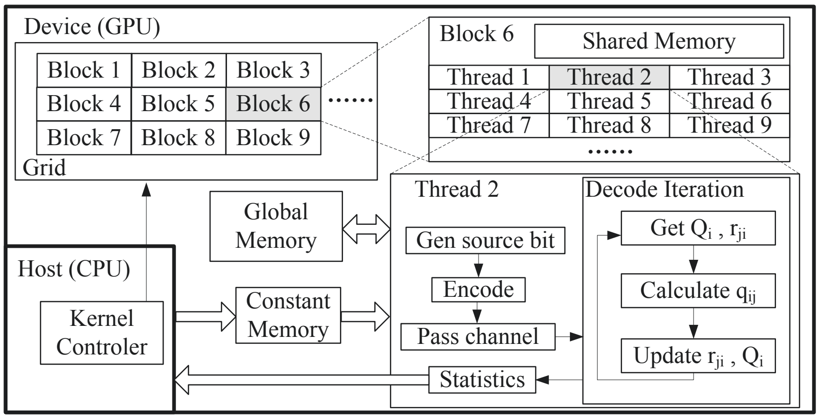

Figure 5 GPU architecture of the BER simulation for LDPC code.

LMMSA for one row of this layer. The procedure ends up with the statistics of $P \times Q$ LDPC blocks.

5.3. Details and instructions

- Ensure "coalesced access" when reading or writing global memory, or the operation will be auto-serialized. In our algorithm, the adjacent threads should access adjacent $L(Q_i)$ and $L(r_{ji})$.
- Shared memory and registers are fast yet limited resources and their use should be carefully planned. In our algorithm, we store $L(Q_i)$ in shared memory and $L(r_{ji})$ in registers due to the lack of resources.
- Make sure all the $P \times Q$ cores are running. This calls for careful assignment of limited resources (i.e., warps, shared memory, registers). In our case, we limit the registers per thread to 16 and threads per block to 128, or some of the Q cores on each MP will "starve" and be disabled.

6. Hardware implementation schemes

6.1. Top-level hardware architecture

Our goal is to implement a multi-mode high-throughput QC-LDPC decoder, which can support multiple code rates and expansion factors on-the-fly. The proposed decoder consists of three main parts, namely, the interface part, the execution part and the control part. The top level architecture is shown in Figure 6.

The interface part buffers the input and output data as well as handling the configuration commands. In the execution part, the LLRSUM and LLREX are read out from the RAMs, updated in the Σ parallel LMMSA cores, and written back to the RAMs, thus forming the *LLRSUM loop* and the *LLREX loop*, as marked red in Figure 6. The control part generates control signals, including port control, LLRSUM control, LLREX control and iteration control.

Note that the *reconfigurable switch network* is designed in the LLRSUM loop to support multi-mode feature. As to achieve high-throughput, we propose the *split-row MMSA core*, the *early-stopping scheme* and the *multi-block scheme*. The split-row core has two data inputs and two data outputs, hence it also "splits" the LLRSUM RAM and LLREX RAM into two parts, meanwhile, two identical switch networks are needed to shuffle the data simultaneously. We also propose the *offset-threshold decoding scheme* to improve BER/BLER performance. The above five techniques are described in detail as follows.

6.2. The reconfigurable switch network

A switch network is an S-input, S-output hardware structure that can put the input signals in the arbitrary order at the output. Formally, given input signals $x_1,x_2,...,x_S$ with data width W, the output of switch network has the form $x_{a_1}, x_{a_2}, ..., x_{a_S}$ where $a_1,a_2,...,a_S$ is any desired permutation of 1,2,...,S. For the design of reconfigurable LDPC decoders, two special kinds of output order are more important, described as follows.

- *Full cyclic-shift:* The output has the cyclic-shift form of the total S inputs, i.e., x_c, x_{c+1},...x_S, x_1,x_2...,x_{c-1}, where $1 \le c \le S$.
- *Partial cyclic-shift:* The output has the cyclic-shift form of the first p inputs, while other signal can be in arbitrary order, i.e., i.e., x_c, x_{c+1},...x_p, x_1,x_2...,x_{c-1}, x_*,...x_*, where $1 \le c<p <S$, and x_* can be any signal from x_{p+1} to x_S.

For the implementation of QC-LDPC decoder, the switch network is an essential module. Suppose $H^b_{j,i} \neq H^b_{k,i} \geq 0, j < k$, and for any $j < l < k$, $H^b_{l,i} = -1$, then the same data is involved in the processing of the above two "1"s, i.e., LLRSUM and LLREX of $BN_{i \times Zf}$ to

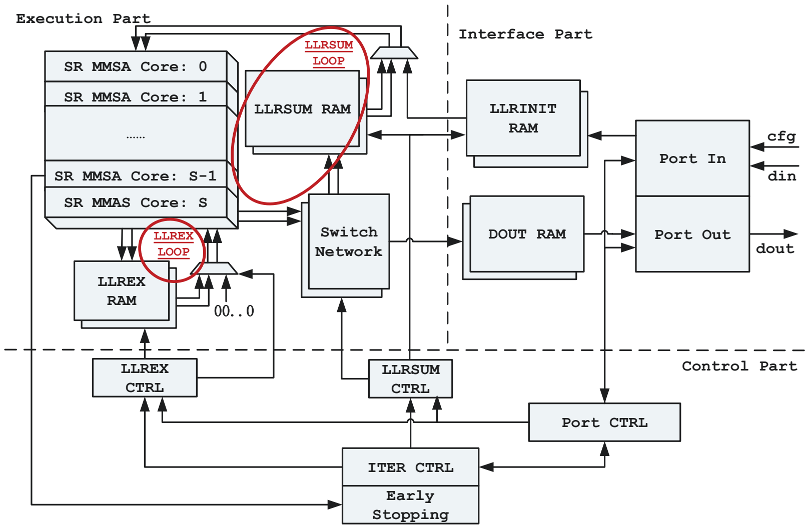

Figure 6 **Top-level multi-mode high-throughput LDPC decoder architecture**.

$BN_{(i+1)\times z_f-1}$. However, after processing $H^b_{j,i}$, the above data should be cyclic-shifted to ensure correct order for the processing $H^b_{j,k}$ which corresponds to *the full cyclic-shift* case with

$$S = z_f, \quad c = (H^v_{k,i} - H^b_{j,i} + S) \mod S \tag{9}$$

Further, in the case of multiple expansion factors, such as WiMAX [19] (z_f = 24: 4: 96), the *partial cyclic-shift* is required with

$$S = z_f^{\max}, \quad p = z_f, \quad c = (H^b_{k,i} - H^b_{j,i} + p) \mod p \tag{10}$$

The existing schemes to implement switch networks include the MS-CS network [20] and Benes network [13,21,22]. The former structure can handle the case when S is not a power of 2, while the latter is proved more efficient in area and gate count. In [13], the most efficient on-the-fly generation method of control signals is proposed. Therefore, we adopt the Benes network proposed in [13] for our decoder. The structure is shown in Figure 7 and the features is given in Table 1.

6.3. The offset-threshold decoding scheme

In this part, we propose the offset-threshold decoding method, which is adopted in our decoder architecture. Unlike existed modifications of MSA [5], the proposed scheme uses an offset-threshold correction to further improve the BER/BLER performance.

The traditional MSA is a simplified version of BP, by replacing the complicated Equation (2) with simple min operation, shown as follows.

$$\text{sgn}\ (L(r_{ji})) = \left(\prod_{i' \in \mathcal{R}_{j\backslash i}} \text{sgn}\ (L(q_{i'j})) \right) \tag{11}$$

$$\text{abs}(L(r_{ji})) = \min_{i' \in \mathcal{R}_{j\backslash i}} \left(\left| L(q_{i'j}) \right| \right) \tag{12}$$

In [5], the normalized and offset MMSA schemes (13) (14) are proposed to compensate the loss of the above approximation, described as follows.

$$\text{abs}(L(r_{ji})) = \alpha \cdot \text{abs}(L(r_{ji})) \tag{13}$$

$$\text{abs}(L(r_{ji})) = \max\left(\text{abs}(L(r_{ji})) - \beta, 0\right) \tag{14}$$

In our simulation, for BLER, the offset MMSA performs better than normalized one. However, as to BER, both schemes show error floor at 10^{-6}, as shown in Figure 8. The problem here is that, for most cases, the offset MMSA works well, while in a few cases, the decoding fails with many bit errors in one block. The intuitive explanation of such phenomenon is existence of extremely large likelihoods ($L(q_{ij})$, $L(r_{ji})$). In high SNR region, the $L(q_{ij})$ likelihoods converge fast to a large value, for both correct and wrong bits in some cases. The wrong bits not only remain wrong, but also propagate large $L(r_{ji})$ to other bits, resulting in more wrong bits and finally failure of decoding. For this reason, we need to set threshold upon offset MMSA to limit the likelihoods of becoming extremely large, which leads to the proposed offset-threshold scheme, done by the following equation.

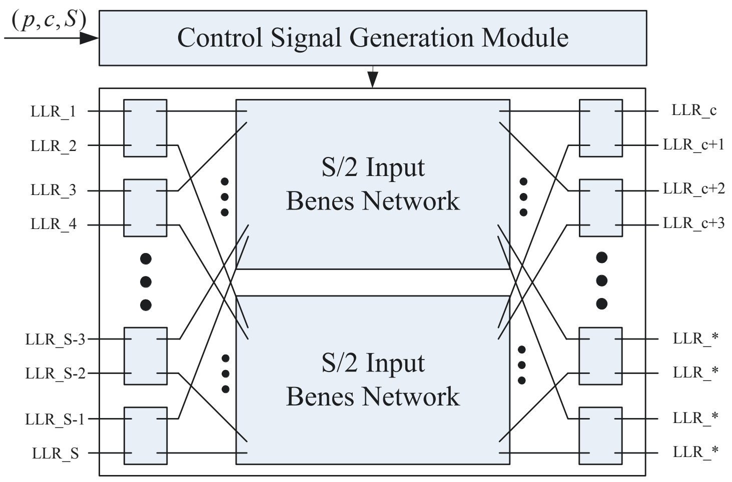

Figure 7 The structure of Benes network.

$$\mathrm{abs}(L(r_{ji})) = \min\left(\max(\mathrm{abs}(L(r_{ji})) - \beta, 0), \gamma\right) \quad (15)$$

The difference between traditional MSA, normalized MMSA, offset MMSA and offset-threshold MMSA is shown in Figure 9. Simulation result (Figure 8) shows that the proposed scheme has lowest error floor (10^{-8}) among the above schemes, while achieving good BLER performance as offset MMSA.

6.4. The split-row MMSA core

This part presents the split-row MMSA core. In traditional semi-parallel structure with layered MMSA core (see Figure 4), since the "1"s in j-th row will be processed one by one to find the minimum and sub-minimum of all $L(q_{ij})$, the decoding stages K for one iteration is proportional to the number of "1"s in each row of the base matrix $\mathbf{H}^b$. The idea is that, if k "1"s can be processed at the same time, the decoding time of one iteration will be shortened by a factor of k, and the throughput will have a gain of k. This is done by split-row scheme, which vertically splits $\mathbf{H}^b$ into multiple part. The "1"s in each part are processed simultaneously to find the local minimum, and the results are merged together. In this way, for $\mathbf{H}^b$ with maximum row weight w, the minimum and sub-minimum can be obtained in w/k clocks. See Figure 2 as an example, we split the 4×6 $\mathbf{H}^b$ into two parts, each has one or two "1"s in every row. The corresponding architecture with $k = 2$ is shown in Figure 10. The LLRSUM ($L(Q_i)$), LLREX ($L(r_{ji})$) and LLR ($L(q_{ij})$) of the left part and right part are stored in two individual RAM/FIFOs, respectively. Two minimum/sub-minimum finders pass result to the merger for final comparison, thus approximately shorten the process pipeline by half. Note that the split position must exist for the code $\mathbf{H}^b$ such that each row in each part contains nearly the same number of "1"s. Otherwise, we need RAMs with multiple read ports and write ports, which is not practical for FPGA implementation.

Table 1 Features of reconfigurable Benes network

Scale	$W = 9$, $S = 256$, 15 stages, 128×15 MUX
Support	$p = 64{:}8{:}256$, $1 \le c \le p$
Resource	20866 LE, 388 memory bits
Clock	100 MHz for Cyclone II, 240 MHz for Stratix III
Delay	Two clocks for control signals, four clocks for output

6.5. The early-stopping scheme

This part introduces the early-stoping scheme applied in our decoder. In practical scenario, the decoding process often gets to convergence much earlier than the preset maximum iterations is reached, especially under favorable transmission conditions when SNR is large. Thus, if the decoder can terminate the decoding iterations as

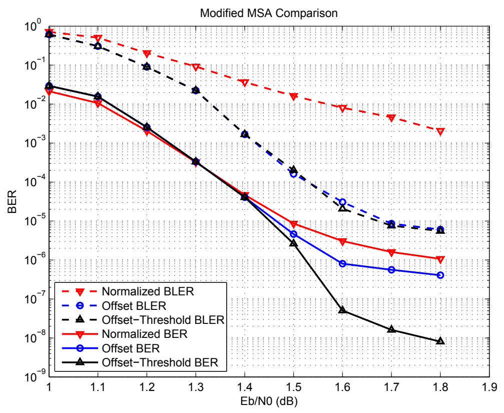

Figure 8 BER/BLER performance of different MSA schemes.

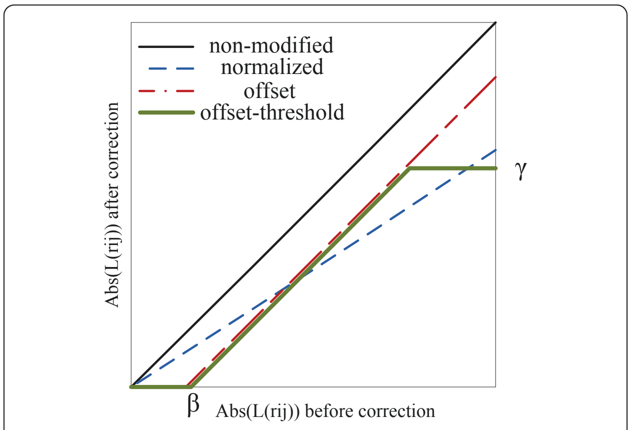

Figure 9 Comparison of the correction types of different MSA schemes.

soon as it detect the convergence, the power of the circuit can be reduced as well as the decoding delay. Throughout is also increased if the system dynamically adjusts the transmission rate according to statistics of average iteration numbers under current channel state.

Traditional stopping criterions focus on whether the code can be decoded successfully or not, which either cost too much extra resource to store iteration parameters, such as HDA [23] and NSPC [24], or use floating-point calculation to evaluate the current iteration situation, such as VNR [25] and CMM [23]. All these methods are not suitable for the hardware implementation.

Here, we propose a simple and effective scheme to detect the convergence of the decoding. The "convergence" means at some time, all of the hard decisions sgn ($L(Q_i)$) satisfy the check equations, The detection of convergence usually demands parallel calculation of each equation, However, due to the layered structure (QC) and lack of the hardware resources, we can use a semi-parallel algorithm to implement iteration-stopping module, which evaluates one layer (z_f equations) simultaneously. If the number of the continuously successful-check layers reach a threshold ω, the module will trigger a signal meaning the decoding have got to convergence and the iteration can be stopped.

One important issue of Algorithm 5 is the estimation of threshold ω. The BER/BLER performances and average iteration times of different ω are shown in Figure 11, where the stopping criterion of ideal iteration is $\mathbf{H}\mathbf{c}^T = 0$. We choose $\omega = 2.5 \times M$ to achieve tradeoff between time and performance. In this case, if the average iteration times is I_{ave} (ideal iteration case), the decoding terminates at approximately $I_{ave} + 2$ iterations.

6.6. The multi-block scheme

Suppose the LDPC code with code length N and expansion factor z_f still has serious memory conflicts though being optimized by our SA and ACO algorithms, which is common for large z_f and relatively small N. To address this problem, we propose a hardware method called "multi-block" to further avoid memory conflicts and increase pipeline efficiency. The "multi-block" scheme is explain as follows.

We construct a new matrix $\mathbf{H}_\nu$ by the parity-check matrix $\mathbf{H}$ with M rows:

$$\mathbf{H}_\upsilon = \begin{pmatrix} \mathbf{H} & 0 \\ 0 & \mathbf{H} \end{pmatrix} \tag{16}$$

Here "virtual matrix" $\mathbf{H}_\nu$ is the combination of two codes $\mathbf{H}$ without "cross constraint" (edge between nodes from different codes in Tanner Graph) between each other. Suppose v_1, v_2 are any two legal encoded blocks satisfying that

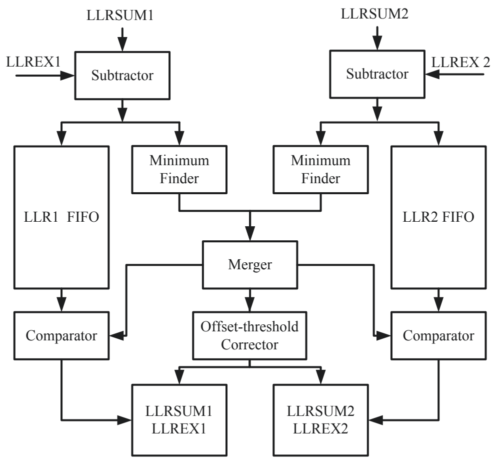

Figure 10 The architecture of split-row MMSA core.

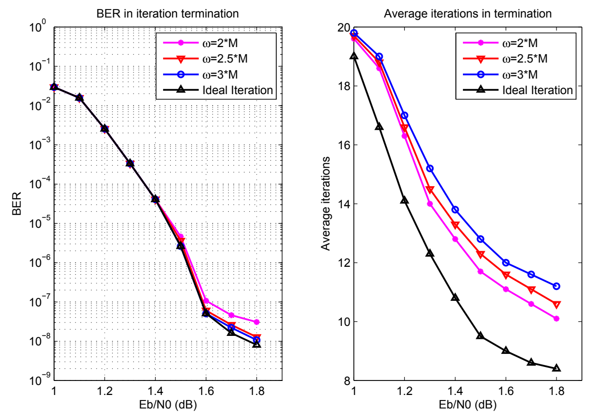

Figure 11 BER/BLER performance and average iterations under different ω.

$$\mathrm{v}_1.\mathbf{H}^{\mathrm{T}} = 0 \quad \mathrm{v}_2.\mathbf{H}^{\mathrm{T}} = 0 \tag{17}$$

Thus the vector (v$_1$ v$_2$) is also one legal block for $\mathbf{H}_v$:

$$\left(\mathrm{v}_1\ \mathrm{v}_2\right).\mathbf{H}_v^{\mathrm{T}} = \left(\mathrm{v}_1\ \mathrm{v}_2\right).\begin{pmatrix}\mathbf{H}^{\mathrm{T}} & 0\\ 0 & \mathbf{H}^{\mathrm{T}}\end{pmatrix} = 0 \tag{18}$$

The key observation is that there are no memory conflicts between the two codes **H** due to the diagonal form of $\mathbf{H}_v$. This enables us to reorder and combine the decoding schedule of the two codes to reduce memory conflict of each code. We rewrite **H** and $\mathbf{H}_v$ as follows:

$$\mathbf{H} = \begin{pmatrix}\mathbf{H}_1\\ \vdots\\ \mathbf{H}_M\end{pmatrix} \quad \mathbf{H}_{\mathrm{vopt}} = \begin{pmatrix}\mathbf{H}_1^{(1)} & 0\\ 0 & \mathbf{H}_1^{(2)}\\ \vdots & \vdots\\ \mathbf{H}_M^{(1)} & 0\\ 0 & \mathbf{H}_M^{(2)}\end{pmatrix} \tag{19}$$

where $\mathbf{H}_i^{(j)}$ denotes the i-th row of the j-th code. The decoding schedule is given by above equation, i.e., $\mathbf{H}_i^{(1)}$ comes first, followed by $\mathbf{H}_i^{(2)}$, and then $\mathbf{H}_{i+1}^{(1)}$, and so on so forth. The benefit of this "multi-block" scheme is that the insertion of $\mathbf{H}_i^{(2)}$ provides extra stages for the conflicts between $\mathbf{H}_i^{(1)}$ and $\mathbf{H}_{i+1}^{(1)}$.

To sum up, the "multi-block" scheme changes any gap-l memory conflict to gap-($2l$ − 1), thus can improve the pipeline efficiency significantly. Meanwhile, it demands no extra logic resources (LE) for the design, but may double the memory bits for buffering two encoded blocks. Since the depth of memory is not fully used on our FPGA, the proposed method can make full use of it with no extra resource cost.

7. Numerical simulation

In this section, we show how our platform produces "good" LDPC codes with outstanding decoding performance and hardware efficiency. For comparison, we target on the WiMAX LDPC code (N = 2304, R = 0.5, z_f = 96). We use the same parameters and degree distributions as WiMAX for our SA-based constructor. We set "cycle" as performance metric and memory conflict as efficiency metric. The performance of one of the candidate codes and the WiMAX code are listed in Table 2. The candidate code has much less length-6/8 cycles and gap-1/2/3 memory conflict. Usually, the candidate codes can eliminate length-6 cycles and gap-1 conflicts, which ensures a larger-than-or-equal-to 8 girth and no conflict under short pipeline (when $K \le w_m$).

Table 2 Cycle and conflict performance of the two codes

	Candidate code	WiMAX code
Cycle:length 6/8	0/55	5/150
Conflict:gap 1/2/3	0/3/9	5/11/15
Pipeline occupancy	Before ACO: 76/88	Only layer
	After ACO: 76/81	permu.: 76/96

We simulate the candidate code and WiMAX code through the GPU platform. The BER/BLER performance is shown in Figure 12, while the platform parameters and throughput are listed in Table 3. The water-fall region and the error floor of our candidate code is almost the same as WiMAX code. For speed comparison, we also include the fastest result that ever reported [12]. The "net throughput" is defined by the decoded "message bits" per second, given by:

$$\text{net throughput} = \frac{P\cdot Q\cdot N\cdot R}{t} \tag{20}$$

where t is the consumed time for running through the GPU kernel (for us is Algorithm 4). As shown in Table 3, our GPU platform speeds up 490 times against CPU and achieves a net throughput 24.5 Mbps. Further, our throughput approaches the fastest one, while providing better precision (floating-point vs. 8 bit fixed-point) for the simulation.

Finally, we optimize the pipeline schedule by ACO-based scheduler, shown in Table 2. The "pipeline occupancy" is given by running/total clocks required for one iteration. For the candidate code, the number of idle clock insertions after ACO is 5, compared with 12 before ACO, achieving a 58.3% reduction. While for WiMAX code, 20 idle clock insertions remain required after layer-permutation-only (single-layer) scheme

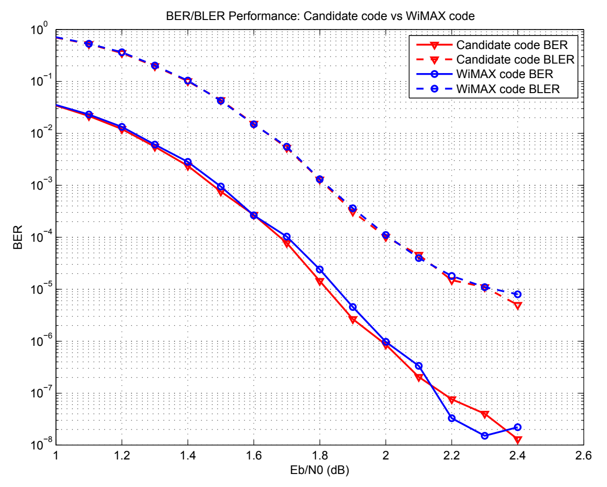

Figure 12 BER and BLER performance of the two codes.

Table 3 Parameters and performance: GPU vs CPU (20 iterations)

	GPU (ours)	CPU	GPU [12]
Platform	NV. GTX260	Intel Core2 Quad	NV. 8800GTX
Clock frequency	1.24GHz	2.66 GHz	1.35 GHz
Decoding method	Semi-parallel LMMSA	Semi-parallel LMMSA	Full-parallel BP
Blocks×threads	216 × 96	1	128 × 256
Net throughput	24.5 Mbps	50 Kbps	25Mbps
Precision	Floating-point	Floating-point	8-bit fixed-point

proposed by [11]. In this case, the double-layered ACO achieves a 75% reduction against the single-layer scheme (5 vs. 20 idle clocks).

8. The multi-mode high-throughput decoder

Based on the above techniques, namely, *reconfigurable switch network*, *offset-threshold decoding*, *split-row MMSA core*, *early-stoping scheme* and *multi-block scheme*, we implement the multi-mode high-throughput LDPC decoder on Altera Stratix III FPGA. The proposed decoder supports 27 modes, including nine different code lengths and three different code rates, and maximum 31 iterations. The configurations for code length, code rate, and iteration number are completely on-the-fly. Further, it has a BER gap less than 0.2 dB against floating-point LMMSA, while achieving a stable net-throughput 721.58 Mbps under code rate $R = 1/2$ and 20 iterations (corresponding to a bit-throughput 1.44 Gbps). With early-stopping module working, the net-throughput can boost up to 1.2 Gbps (bit-throughput 2.4 Gbps), which is calculated under average 12 iterations. The features are listed in Table 4.

One great advantage of the proposed multi-mode high-throughput LDPC decoder is that more modes can be supported with only more memory bits consumed and no architecture level change. Since the *reconfigurable switch network* supports all expansion factors $z_f \leq 256$, and the layered MMSA cores supports arbitrary QC-LDPC codes, more code lengths and code rates are naturally supported, for example, the WiMAX codes (z_f = 24: 4: 96, R = 1/2, 2/3, 3/4, 5/6, 114 modes in total). The only cost is that more memory bits are required to store the new base matrices $\mathbf{H}^b$.

Table 4 Features of the multi-mode high-throughput decoder

FPGA platform	Altera Stratix III EP3SL340F1517C2
Decoding scheme	Layered offset-threshold MSA
Modes supported	9 × 3 = 27 modes
Code length	N = 1536:768:6144 (z_f = 64:32:256)
Code rate	R = 1/2,2/3,3/4 ($\mathbf{H}^b$:12 × 24, 8 × 24, 6 × 24)
Iteration number	iter = 1–31, 20 recommended
Resources usage	149, 976 LE, 3, 157, 136 bits memory
BER performance	gap ≤ 0.2 dB vs. 20 iteration float LMMSA
Clock setup	225.58MHz
Stable net throughput	721.58 Mbps (z_f = 256, R = 1/2, iter = 20)
Max. net throughput	1.2 Gbps (early-stopping, iter = 12 ave.)

9. Conclusion

In this article, a novel LDPC code construction, verification, and implementation methodology is proposed, which can produce LDPC codes with both good decoding performance and high hardware efficiency. Additionally, a GPU verification platform is built that can accelerate 490× speed against CPU and a multi-mode high-throughput decoder is implemented on FPGA, achieving a net-throughput 1.2 Gbps and performance loss within 0.2 dB.

Additional material

Additional file 1: Algorithm. This file contains Algorithm 1, Memory conflict minimization algorithm; Algorithm 2, ACO algorithm for TSP; Algorithm 3, The SA based LDPC construction framework; Algorithm 4, The GPU based LDPC simulation; and Algorithm 5, Semi-parallel early-stopping algorithm.

Acknowledgements
This paper is partially sponsored by the Shanghai Basic Research Key Project (No. 11DZ1500206) and the National Key Project of China (No. 2011ZX03001-002-01).

Competing interests
The authors declare that they have no competing interests.

Received: 15 May 2011 Accepted: 6 March 2012
Published: 6 March 2012

References

1. R Gallager, Low-density parity-check codes. IRE Trans. Inf. Theory. **8**(1), 21–28 (1962). doi:10.1109/TIT.1962.1057683
2. R Tanner, A recursive approach to low complexity codes. IEEE Trans. Inf. Theory. **27**(9), 533–547 (1981)
3. D MacKay, Good error-correcting codes based on very sparse matrices. IEEE Trans. Inf. Theory. **45**(3), 399–431 (1999)
4. T Richardson, M Shokrollahi, R Urbanke, Design of capacity approaching irregular low-density parity-check codes. IEEE Trans. Inf. Theory. **47**(2), 619–637 (2001). doi:10.1109/18.910578
5. J Chen, RM Tanner, C Jones, L Yan Li, Improved min-sum decoding algorithms for irregular LDPC codes, in *Proc. ISIT*, (Adelaide, 2005), pp. 449–453
6. DE Hocevar, A reduced complexity decoder architecture via layered decoding of LDPC codes, in *IEEE workshop on SiPS*, pp. 107–112 (2004)
7. Y Hu, E Eleftheriou, DM Arnold, Regular and irregular progressive edge growth Tanner graphs. IEEE Trans. Inf. Theory. **51**(1), 386–398 (2005)
8. D Vukobratovic, V Senk, Generalized ACE constrained progressive Eedge-growth LDPC code design. IEEE Comm. Lett. **12**(1), 32–34 (2008)
9. AJ Blanksby, CJ Howland, A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder. J. Solid State Circ. **37**(3), 404–412 (2002). doi:10.1109/4.987093
10. Z Cui, Z Wang, Y Liu, High-throughput layered LDPC decoding architecture. IEEE Trans. VLSI Syst. **17**(4), 582–587 (2009)
11. C Marchand, J Dore, L Canencia, E Boutillon, Conflict resolution for pipelined layered LDPC decoders, in *IEEE workshop on SiPS*, (Tampere, 2009), pp. 220–225

12. G Falcao, V Silva, L Sousa, How GPUs can outperform ASICs for fast LDPC decoding, in *Proc. international conf on Supercomputing*, (New York, 2009), pp. 390–399
13. J Lin, Z Wang, Effcient shuffle network architecture and application for WiMAX LDPC decoders, in *IEEE Trans. on Circuits and Systems*. **56**(3), 215–219 (2009)
14. KK Gunnam, GS Choi, MB Yeary, M Atiquzzaman, VLSI architectures for layered decoding for irregular LDPC codes of WiMax, in *IEEE International Conference on Communications*, (Glasgow, 2007), pp. 4542–4547
15. T Brack, M Alles, F Kienle, N Wehn, A synthesizable IP core for WIMAX 802.16E LDPC code decodings, in *IEEE Inter. Symp. on Personal, Indoor and Mobile Radio Comm*, (Helsinki, 2006), pp. 1–5
16. K Tzu-Chieh, AN Willson, A flexible decoder IC for WiMAX QC-LDPC codes, in *Custom Integrated Circuits Conference*, (San Jose, 2008), pp. 527–530
17. M Dorigo, LM Gambardella, Ant colonies for the travelling salesman problem. Biosystems. **43**(2), 73–81 (1997). doi:10.1016/S0303-2647(97)01708-5
18. S Kirkpatrick, CD Gelatt, MP Vecchi, Optimization by simulated annealing. Science, New Series. **220**(4598), 671–680 (1983)
19. IEEE Standard for Local and Metropolitan Area Networks Part 16. IEEE Standard 802.16e (2008)
20. M Rovini, G Gentile, F Rossi, Multi-size circular shifting networking for decoders of structured LDPC codes. Electron Lett. **43**(17), 938–940 (2007). doi:10.1049/el:20071157
21. J Tang, T Bhatt, V Sundaramurthy, Reconfigurable shuffle network design in LDPC decoders, *IEEE Intern Conf ASAP*, (Steamboat Springs, CO, 2006), pp. 81–86
22. D Oh, K Parhi, Area efficient controller design of barrel shifters for reconfigurable LDPC decoders, in *IEEE Intern Symp on Circuits and Systems*, (Seattle, 2008), pp. 240–243
23. L Jin, Y Xiao-hu, L Jing, Early stopping for LDPC decoding: convergence of mean magnitude (CMM). IEEE Commun Lett. **10**(9), 667–669 (2006). doi:10.1109/LCOMM.2006.1714539
24. S Donghyuk, H Kyoungwoo, O Sangbong, A Jeongseok Ha, A stopping criterion for low-density parity-check codes, in *Vehicular Technology Conference*, (Dublin, 2007), pp. 1529–1533
25. F Kienle, N Wehn, Low complexity stopping criterion for LDPC code decoders, in *Vehicular Technology Conference*. **1**, 606–609 (2005)

doi:10.1186/1687-1499-2012-84
Cite this article as: Yu *et al.*: Systematic construction, verification and implementation methodology for LDPC codes. *EURASIP Journal on Wireless Communications and Networking* 2012 **2012**:84.