Design of Parallel LDPC Interleaver Architecture: A Bipartite Edge Coloring Approach
Awais Hussein Sani, Philippe Coussy, Cyrille Chavet, Eric Martin

To cite this version:

HAL Id: hal-00551432
https://hal.archives-ouvertes.fr/hal-00551432
Submitted on 7 Jan 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract- Parallel hardware architecture proves to be an excellent compromise between area, cost, flexibility and high throughput in the hardware design of LDPC decoder. However, this type of architecture suffers from memory mapping problem: concurrent read and write accesses to data have to be performed at each time instance without any conflict. In this paper, we present an original approach based on the Tanner graph modeling and a modified bipartite edge coloring algorithm to design parallel LDPC interleaver architecture.

1. INTRODUCTION
Near Shannon limit error correcting capabilities of Low Density Parity Check (LDPC) codes [1] has gained a lot of attention in information theory community. Due to very high decoding throughput and communication performance, LDPC codes are increasingly included in the standards such as DVB-S2 and DVB-T2 [3], WiFi (IEEE 802.11n) [4] or WiMAX (IEEE 802.16e) [5]. LDPC codes are linear block codes and are represented either by parity check matrix H or by Tanner graph [2], which is a bipartite graph. In its Tanner graph representation two types of vertices, variable nodes (VNs) and check nodes (CNs), construct the two vertex sets of bipartite graph (cf. Figure 1). VNs represent the codewords (i.e. data to be processed) and CNs corresponds to the parity-check sums (i.e. operations to be done on the data). A VN is connected to a CN by an edge if and only if it is checked by that check node.

The decoding process is carried out by an iterative message-passing algorithm called “Belief Propagation Algorithm”. In this algorithm, VN and CN iteratively exchange their soft-information to qualify the likelihood of the variable in accordance with the associated parity-check equation [1].

The different methodologies for solving collision problem are proposed to map the data in different memory banks for conflict free concurrent read/write accesses. In [6], [7], the authors propose to use a mapping algorithm to remove memory conflicts in flexible LDPC decoders. However, the proposed approach is based on a simulated-annealing algorithm, so the user cannot predict when the algorithm will end. Moreover, it fails to optimize either the storage elements or the interconnection network. Finally, different heuristics [15], [16] have been proposed to solve the mapping problem in turbo and LDPC decoders. However, they consider in-place memory access in which data have to be read from and write to the same memory location. Finally, conflict graph can be used. In this model, a node represents a data and two nodes are connected if and only if the associated data are accessed at the same time. Node coloring algorithm can then be used to solve the mapping problem: each color corresponds to one memory bank. Unfortunately only one color can be assigned to one node i.e. a data can be stored in only one memory bank. This constraint may require more memory banks than needed (see [17] for more details). Similarly, number of algorithms have been proposed for coloring the edges of a biparite graph by constructing partitions ([13] and [14]) for
example). Unfortunately, like node coloring approaches they can not be used to solve the mapping problem because each data is supposed to be stored in one memory bank only i.e. only one color can be assigned to one edge.

### 3. PROBLEM FORMULATION

To explain the problem, we consider a set of $K$ data elements $\{d_0, d_1, \ldots, d_{K-1}\}$ and a set of $P$ processing elements $\{PE_0, PE_1, \ldots, PE_{P-1}\}$ which iteratively process these $K$ data elements in $N$ time instances $\{t_0, t_1, \ldots, t_{N-1}\}$.

In order to store these $K$ data elements and to achieve parallel iterative processing of data for high throughput a set of $B$ memory banks $\{b_0, b_1, \ldots, b_{B-1}\}$, where $B = P$, is used. All the memory banks have the same size $M$ which is equal to $M = K/P$.

#### Mapping problem

Store $K$ data elements in $B$ memory banks in such a manner that $P$ processing elements can access $B$ memory banks in parallel at each time instance for first reading and then writing $B$ data elements without any conflict.

To highlight this problem, we introduce a data access matrix in which we have $P$ rows, related to the processing elements, and $N$ columns, related to the time instances. Data elements in each row are processed by the processing element connected with this row. Similarly data elements in each column need to be accessed in parallel by $P$ processing elements for partially parallel decoding architecture. Figure 2 represents the data access matrix in which we have $K = 6$, $P = B = 3$, $M = 2$ and $N = 6$. Each data is processed by 3 times which shows the iterative nature of the data access. However, data accesses are interleaved in time and there is no regularity in processing the data elements; e.g., data 3 is successively processed in time instances $t_1$ and $t_2$ whereas the first access to the data element 4 occurs at time instance $t_4$.

![Figure 2: Data Access Matrix](image)

#### Memory Mapping Constraints

To successfully map the data (i.e. to allow conflict free parallel memory access) in (1) a given number of memory banks and (2) to tackle the iterative nature of data access in error correction coding, the mapping matrix must fulfill the following constraints:

1. At each time instance, all the memory banks have to be used one and only one time.
2. The bank of the last write access to a data must be the same as the bank of its first read access.

#### Formal modeling of mapping problem

To tackle the mapping problem, we introduce the concept of multiple read and multiple write access in the formal modeling of mapping problem in which we can not only access the data with in-place strategy (if it is possible) but we can also read a data element from one memory bank and then write it in a different one in order to map the data in minimum required memory banks. This approach is based on the edge coloring of the bipartite graph and presented in section 5.

### 4. DEFINITIONS

A graph $G = (V, E)$ is a collection of node, set $V$, and edge, set $E$. If $v, w \in V$ then an edge $(v, w) \in E$ is incident to $v$ and to $w$, and vertices $v$ and $w$ are said adjacent. A subgraph of $G$ is a graph whose vertices and edges are in $G$.

To delete edge $(v, w)$ from $G$ means to form the subgraph $G - (v, w)$, consisting of all vertices of $G$ and all edges of $G$ except $(v, w)$. A graph $G = (S_1 \cup S_2, E)$ is bipartite, if $S_1$ and $S_2$ divide the vertices set so that each edge is incident to a vertex in $S_1$ and a vertex in $S_2$ i.e. $S_1 \cap S_2 = \emptyset$.

The degree of vertex $v$ is the number of edges incident to $v$. A graph is regular if all vertices have the same degree. A graph is semi regular, if either all the vertices in $S_1$ or all the vertices in $S_2$ have the same degree. A path $P$ is a sequence of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$. The ends of $P$ are vertices $v_1$ and $v_n$. If $v_i \neq v_j$, $P$ is open; otherwise $P$ is closed. A graph is connected if there is a path between any two distinct vertices.

We define a partition in semi regular bipartite graph as a subgraph including all time vertices.

**Lemma 1:** When the degree $d$ of the time vertex in a semi regular graph is even then we have $d/2$ partitions in which time vertex’s degree $d_i$ is 2. **Lemma 2:** When the degree $d$ of the time vertex in a semi regular graph is odd then we have $\lfloor d/2 \rfloor$ partitions in which time vertex’s degree $d_i$ is 2 and one subgraph in which $d_i$ is 1.

We finally define a proper partition in semi regular bipartite graph as a partition that respects either Lemma 1 or Lemma 2.

An edge coloring of $G$ is an assignment of a color to each edge in $G$. An edge chromatic number, $\chi(G)$, is the fewest number of colors necessary to color each edge of a graph so that no two edges incident to the same vertex have the same color.

In [12], König proved that if the maximum vertex degree of a bipartite graph is $d$ then, $\chi(G) = d$.

### 5. PROPOSED APPROACH

The proposed algorithm is divided into three steps. In the first step we construct a bipartite graph based on data access matrix. In the second step, we divide our graph into different proper partitions. In the third step, the edges of each partition are colored.

**Step I:** A bipartite graph $G = (T \cup L, E)$ is constructed based on data access matrix (e.g. figure 3) in which vertex set $T$ represents all the time instances and vertex set $L$ represents all the data elements used in the computation. An edge $(t, l)$ is incident to the vertex element $l$ and to the time instance vertex $t$ if $l$ needs to be processed at $t$ (i.e. data $l$ will be read and next written at time $t$). Moreover, different data accesses are represented based on the relative position $i$ of edges at the data vertex i.e. first edge at $t$ represents the first read and write accesses and so on. However, the read access that follows the $i^{th}$ write access is the $(i+1)^{th}$ edge at the data node $l$. An edge that represents the $j^{th}$ read access will be next referred in this paper as a direct edge and the edge corresponding to the associated write access as the induced edge. This placement property will be used during steps II and III. One interesting property of LDPC decoding is that the number of accesses to data or processing elements at any time instance is always equal which implies that corresponding bipartite graph is always semi regular. This implies that all the time nodes in the bipartite graph have the same degree $d_i$.

![Figure 3: Bipartite representation](image)

**Step II:** In this step, bipartite graph $G$ is divided into proper partitions. In order to simplify the coloring algorithm next used in step III, one constraint named partitioning constraint is introduced: no more than 2 read or write accesses have to be done at each time instance in a proper partition. Following this constraint always allows to construct proper partitions. Each proper partition is constructed using the partitioning algorithm which is shown in Figure 4.a. In this algorithm, two processes working side by side apply at each time and data vertex: Process of traversal and Process of elimination. Process of traversal randomly selects one edge available at the current data or time node and records its induced edge. Process of elimination removes all the edges from the current partition which contradict the partitioning constraint. Hence if $d_i$ number of selected direct edges (i.e. read accesses) appear in a time node then the remaining (i.e. non-selected) available edges at that time...
instance are eliminated. Also, if $d_i'$ number of recorded induced edges (i.e. write accesses) appear in a time node then the direct edges associated to the remaining (i.e. non-recorded) induced edges of that time node are eliminated.

Hence, the algorithm starts constructing a path $p_{cur}$ by choosing any data vertex $t_{cur}$ and then by applying process of traversal which selects randomly an edge $(t_{cur}, t_{new})$ to reach at the time vertex $t_{cur}$. Process of elimination is then applied to remove all the edges which contradict the partitioning constraint. At $t_{cur}$, the process of traversal is again applied to choose another edge $(t_{cur}, t_{new})$ to reach at the data vertex $t_{new}$. Again the process of elimination is applied to remove all the edges which contradict the partitioning constraint. At that time $p_{cur} = \{ (t_{cur}, t_{new}), (t_{new}, t_{next}) \}$. The algorithm continues until $p_{cur}$ is completed, i.e. the process of traversal does not find any valid edge to be included in $p_{cur}$. The path is added in the current subgraph $sg_{cur}$. The algorithm tests if the $sg_{cur}$ is a partition (i.e. all the time node has been traversed).

Once a partition has been extracted the algorithm stops. Otherwise, the algorithm starts constructing another partition on the remaining graph $G = G - sg_{cur}$. Step II is explained through a pedagogical example in the next section.

Step III: Thanks to the construction of proper partitions respecting the partitioning constraint, our coloring algorithm, which flow chart is shown in Figure 4.b, colors each partition with at most two colors. For this it uses a strategy to color each edge in each partition so that there is no conflict in the read and write access at each time node.

For each uncolored partition $sg_{cur}$, the algorithm starts by removing the read conflict accesses by assigning different color to each edge $(l_i, t_{cur})$ of $t_{cur}$. After that, following the placement property (see step I description) the algorithm searches in $G$ for each edge $(l_i, t_{cur})$ of $t_{cur}$ for the induced edge $(l_{induced}, l_k)$. Since only two write accesses are possible at each time node (by partitioning constraint), the algorithm searches in $G$ for the direct edge $(l_i, t_j)$ of the induced edge $(l_{induced}, l_k)$ that belongs to $sg_{cur}$. The algorithm then colors $(l_i, t_j)$ differently from $(l_i, t_{cur})$ and continues until it reaches the starting node whose both direct edges are already colored.

While the partition is not completely colored the algorithm selects another time node $t_{cur}$ and repeats. It should be noticed that simply giving different colors to both the direct edges at each time node in each partition without taking into account the write access memory conflicts makes the algorithm recursive.

![Figure 4: Partitioning and Coloring Algorithms](image)

6. PRACTICAL IMPLEMENTATION

Let us present an example based on the data access matrix introduced in section 3. The first column shows the memory banks which are used for read access and second column shows the memory banks which are used for write access at this time instance. The mapping matrix of Figure 2 is shown in Figure 5.

![Figure 5: Mapping matrix for data access matrix of Figure 2](image)

The algorithm starts constructing the path $p_1$, by using the first available edge of data $l_1$ which is $(1, l_1)$, leading to $p_1 = \{(1, l_1)\}$. The selected edge $(1, l_1)$ and its corresponding recorded induced edge $(l_1, t_1)$ appears respectively as bold and dotted line in Figure 6.a. Using the placement property the write access of the edge $(1, l_1)$ indeed appears on the edge $(1, t_1)$. The process of elimination is applied and no edge is removed. Since this edge is both a recorded induced edge and a direct selected edge, it thus appears in bold and dotted line in Figure 6.c.

The process continues until we traverse the path $p_2 = \{(1, l_1), (2, l_2), (3, l_3), (4, l_4), (4, l_5), (6, t_6)\}$ and reach at the time node $t_6$. At this point, recorded induced edges at $t_2$ increase to two and the process of elimination deletes all the direct edges associated to the remaining (i.e. non-recorded) induced edges at $t_2$. All this process is shown in Figure 6.d.

![Figure 6: Path construction through Partitioning Algorithm](image)
After the generation of the partitions, each partition is colored depending on the degree $d'_t$ of its node time. For example, the $sG_1$ is colored with, $d'_1 = 2$, colors and the $sG_2$ is colored with, $d'_1 = 1$, color. To color the partition $sG_t$, we apply the already presented coloring algorithm. We start by coloring the edges connected with $t_i$ with different colors $b_0$ and $b_1$ to avoid a conflict access. Edge $(t_1, 1) = b_0$ and edge $(t_3, 3) = b_1$ as shown in Figure 8.a. In this figure, bold grey straight line represents color $b_0$ and thin bold grey dotted line represents color $b_1$. The corresponding mapping matrix is shown in Figure 8.b.

The coloring of $sG_t$ is easier: all the edges are colored with one single color $b_2$. The complete coloring of $G$ is shown in Figure 10.a. The corresponding mapping matrix is presented in Figure 10.b.

**Reference**