A Design Approach Dedicated to Pattern-Based and Conflict-Free Parallel Memory System

A. Briki, C. Chavet, P. Coussy and E. Martin

Abstract – Hardware design of parallel interleaver architectures for Turbo-Codes (TB) and Low Density Parity Check (LDPC) is one of the hottest topics in channel coding domain. The throughput constraints that have to be supported by these architectures require avoiding memory collision (i.e. simultaneous read/write accesses should not target the same memory bank). Parallel TB or LDPC decoder typically includes several processing elements (PE) and memory banks (MB), an interconnection network allowing PEs to concurrently access MBs and a controller. In this paper we propose a methodology that finds collision-free mapping in the parallel memory banks and also respects user defined interconnection network. This allows to generate regular architectures where the steering logic and the controller complexities have been optimized.

1. Introduction

LDPC and Turbo Codes are well-known, near Shannon limit, coding/decoding approaches, and are able to achieve very low bit error rates for low Signal-to-Noise Ratio (SNR) applications. In order to design high data rate applications such as wideband wireless multimedia communications, classical solutions are based either on time or space parallelism between memory and computational resources that compose the system [1]. The memory bandwidth is increased by allowing several concurrent memory accesses to different memory modules (cf. Figure 1). The main problem in such parallel memory systems is to find a method and an associated interconnection network to distribute data over the memories in such a way that access conflicts are avoided. Collision arises when different parallel processing elements try to simultaneously read/write data from/to the same memory bank [2]. Designers try to target well-known and optimized interconnection network (referred as pattern in this paper) when they design parallel architectures. They also try to maximize the regularity (reducing control overhead, trying to optimize power consumption...) and the potential reusability (time-to-market) of the resulting architecture [3]. To tackle these problems different solutions have been proposed. A first solution to get rid of collisions with non-prunable interleavers consists in designing a specific interleaving rule. In [4] and [6] the authors consider spatial and temporal combinations as a permutation. This solution is reliable if and only if the designer is free to define his own permutation law.

A second approach consists in adding extra memory elements in the communication network. In [7] the authors propose, in case of conflict during interleaved order access, to use additional buffers within the network in order to store conflicting data, until the targeted processor can process it. This is a suboptimal strategy, in terms of latency and thus throughput, which avoids collisions at the expense of area and memory.

The third approach consist in statically compute a conflict-free memory mapping. In [8] it has been proven that there is a suitable mapping of the variables in the memory that grants a collision free access. But the proposed approach is based on a simulated-anneling algorithm (i.e. the user cannot predict when the algorithm will end). In addition, the proposed approach neither targets the optimization of the storage elements, nor the optimization of the network. In [5], the author propose a methodology called SAGE which generates a collision free mapping of the variables in the memory banks and optimizes the resulting interleaving architectures. The main interest of the proposed algorithm is its lower complexity and the possibility to target user defined interconnection network. However, the targeted architecture can be achieved only if the interleaved scheme allows it. Furthermore, this approach is strictly dedicated to Turbo-Code applications [1].

2. Problem formulation

To explain the problem, we consider a set of L elements E={e₀,...,e_L} each of them will be processed N times by the processing elements. The parallelism P represents the number of processing elements required in order to achieve throughput constraints. The number of parallel memory banks B needed to store these data equals to P, i.e. B={b₀,...,b_P}. For the selected column, our algorithm search for a mapping for 3 memory banks. The pattern constraint supposes that this proposed conflict-free mapping can be used with the required interconnection architecture.

3. Proposed approach

A- Formal model

In order to solve the mapping problem, a first solution has been proposed in [9]: considering separately the read and the write access of each data at each cycle (see Figure 3).

B- Mapping constraints

In order to guarantee a valid memory mapping, some structural and architectural constraints have to be respected. Structural constraints:

These constraints have to target a user defined steering architecture (barrel shifter, butterfly, De Bruijn network...).

Algorithm:

Our algorithm can be divided into two successive steps: a first greedy conflict-free mapping algorithm is used to generate the memory mapping matrix. During this first step, if any data access may cause the non-respect of any of the previously defined constraints, the conflicting data will not be mapped, resulting in a partially filled matrix. The second step of our algorithm corrects these conflicts as much as possible, but without modifying the previously mapped data accesses.

Finally, if no correct mapping is possible (with respect to the predefined constraints), then these data accesses will be mapped in a dedicated register.

Initial mapping:

This first step aims to generate a first conflict free memory mapping. Each time a data access does not respect the architectural or structural constraints then it is not mapped. If the entire matrix has been filled by the initial mapping step, then a valid memory mapping has been found. Initial mapping is performed as follows:

- Column selection: This step selects the most constrained although not yet explored column in the mapping matrix. The most constrained column (read or write access column for a given memory bank) is the column in which there remains the less data accesses to be mapped.

Assigning and reporting: For the selected column, our algorithm search for a valid memory mapping with respect to architectural and structural constraints. Then the algorithm updates the corresponding cells with respect to structural constraints.

Verification: This algorithm checks if the updated memory mapping in the matrix respects our exploration constraints, otherwise these elements are not assigned yet, and the related mapping (with respect to the structural constraint 3) are removed from the mapping matrix. Steps 1 to 3 are repeated until all the columns of the mapping matrix have been explored. Then the conflict solving step is applied.

Conflict solving: This step inputs the resulting mapping matrix provided by the initial mapping step and tries to map the unassigned memory accesses. The idea is still to map the data in the memory banks as much as possible, with respect to structural and architectural constraints. Each time data access cannot be assigned, the data is definitely removed from the memory bank and mapped to an additional register. This step is referred as constraint relaxation.

The algorithm aims to minimize the number of required registers, and the number of additional multiplexers, and to reduce the complexity of the associated controller. Binding of additional registers is performed thought a dedicated algorithm based on the STAR design flow [1].

4. Practical Implementation

Let’s take as example the matrix proposed in Figure 1. The targeted architecture should be composed of three memory banks (A, B, C) and three processors (P₁, P₂, P₃).
P2, P3). Let’s consider the architectural constraint to be a barrel-shifter for designing the interconnection network. Our algorithm begins by assigning a memory mapping for a first set of data in column 1, the first read accesses of the data in the first column (Figure 4); data 1 in the first column of the mapping matrix is read in the memory bank A (resp. bank B for data 2 and bank C for data 3).

Then the last write access to this data is assigned to the same memory bank as can be seen in the Figure 5. Once this update has been done, the algorithm selects the most constrained column (i.e. the most constrained cycle for read or write access). After that, the algorithm tries to assign a memory mapping with respect to both the structural and the architectural constraints. In Figure 5, the most constrained column is the last one (write access of cycle 6), so the data 4 is assigned to the memory bank C, and as a consequence, the first read access to this data is also assigned to the same memory bank C (in order to respect the structural constraint 2), see Figure 6.a.

Now, the most constrained column is either the read access column of cycle 3 or the write access column of cycle 5. In such a case, our algorithm select the column which is the most constrained in the past. In our example, the selected column will be the read access column of cycle 3. So the read access to data 6 is assigned to the memory bank B and the read access to data 1 is assigned to memory bank A (in order to respect the architectural constraint). With respect to the structural constraints, the previous write access of these data is assigned to the same memory bank (see Figure 6.a). Then our algorithm is performed on the rest of the matrix until it reaches a mapping conflict due to the structural or architectural constraints (a barrel shifter in this example), see Figure 7.a. In this case the write access to data 5 (cycle 2) has been mapped in the memory bank A, so the next read access to data 5 (cycle 3) has to be done in the same memory bank. Data 2 has also been written in the memory bank B so it has to be read in the same memory bank, therefore the memory mapping of the read access column in cycle 4 does not respect the architectural constraint. So in this first step of our algorithm, the read access in cycle 4 and the corresponding write access of data 5 and 2 will not be mapped: these cancelled mapping are represented by colored grey cells in Figure 7.b.

Next, the algorithm is applied on the rest of the matrix. Figure 8 shows the resulting partially mapped matrix obtained by the initial step of our algorithm.

The Second step consists in filling the blanks in the mapping matrix, if such an unfilled access exists. For each blank we generate the possible solutions, with an obvious no overlapping lifetime, only one register is required in our example (see Figure 9).

5. Experimental result

Our approach has been applied to the non-binary LDPC applications and it has been compared to the approach presented in [9], which is not able to target a dedicated interconnection network architecture. Because of non-disclosure agreement, the synthesis results area, which have been obtained using DC compiler, are given in NAND-gate equivalent. Non-binary LDPC codes are now recognized as a potential competitor to binary coded solutions, especially when the codeword length is small or moderate.

<table>
<thead>
<tr>
<th>Area</th>
<th>Non-binary LDPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>42/64</td>
</tr>
<tr>
<td>Interleaving</td>
<td>42/64</td>
</tr>
<tr>
<td>Custom FSM</td>
<td>718</td>
</tr>
<tr>
<td>Total Area</td>
<td>44/74</td>
</tr>
</tbody>
</table>

Table 1. Synthesis results (area in NAND gate equivalent)

These experiments have been performed on the non-binary LDPC for 192 data accesses twice and a parallelism of 6 processing units. The architectural objective was to target a barrel-shifter based interconnection network. Thanks to the approach proposed in this paper, the architecture we generated by using our approach is capable of mapping 65% bigger than the architecture generated with the approach proposed in [9]. However, the architecture provided by [9] does not respect the user defined architectural objective, contrary to the architecture generated with our algorithm. The limited over-cost observed with our approach comes from the additional multiplexers and registers that are associated with the inputs. Moreover, the total memory cost can be reduced in the memory banks, so the silicon of these memory banks can be reduced in this case. Moreover the number of registers (and associated multiplexers) can also be optimized during the binding step performed by [1].

6. Conclusion

In this paper, we propose a parallel memory design space exploration methodology. This approach relies on a formal modeling describing data communication constraints. This methodology allows the generation of a conflict-free memory mapping and the resulting architecture respects the designer architectural constraints in any case. Our approach has been compared through industrial test-cases to the state-of-the-art techniques and its interest has been shown. Future works will enhance the conflict solving algorithm performances. Moreover, if our first objective was to strictly respect the designer architectural constraints, now, in order to reduce the cost of the added registers and multiplexers, we explore the decomposition of the data exchanges to generate smaller and complementary interconnection networks instead of one.

References