A PARALLEL ARCHITECTURE FOR 3GPP2/UMB TURBO INTERLEAVERS

Mohammad M. Mansour

American University of Beirut
ECE Department
Beirut, Lebanon
Email: mmansour@aub.edu.lb

ABSTRACT

In this paper, an efficient architecture for a parallel pruned turbo interleaver for 3GPP2/UMB physical layer standard [1] is presented. Turbo interleaving in UMB turbo codes is based on filling a 2D array row by row, interleaving each row using a linear congruential sequence, bit-reversing the order of the rows, and then reading the interleaved addresses column by column. Pruning creates a serial bottleneck since the interleaved address of a linear address x is a function of the number of pruned addresses up to x. An architecture based on the parallel lookahead pruned interleaving algorithm proposed in [2] is presented. The algorithm breaks this dependency and interleaves any address in $\mathcal{O}(\log^2 x)$ steps by enabling a parallel turbo interleaver design with a desired degree of parallelism. The architecture can be implemented efficiently in hardware using basic arithmetic building blocks.

Index Terms— Turbo codes, turbo interleavers

1. INTRODUCTION

After the introduction of turbo codes [3], the interest in designing turbo interleavers and their corresponding efficient architectures grew. An integral part of a turbo code is the pseudo-random interleaver which produces a random-like weight spectrum of codewords when used with systematic feedback constituent convolutional encoders. Computationally efficient turbo interleaving algorithms that tend to approximate the behavior of pseudo-random interleavers are typically based on block interleavers that write a set of linear addresses into a 2D array in one direction, apply independent pseudo-random permutations to the row and column entries, and then read the resulting reshuffled entries in the other direction. To accommodate for flexible codeword lengths, interleavers usually employ pruning to have a programmable length. In pruned interleaving, a packet of length L is interleaved by mapping n-bit addresses from linear order into permuted order, where n is the smallest integer such that $L \leq 2^n$. Linear addresses that map to addresses greater than L are pruned away. Other pruning techniques are discussed in [4–7].

This work was supported by funds from the University Research Board at the American University of Beirut.

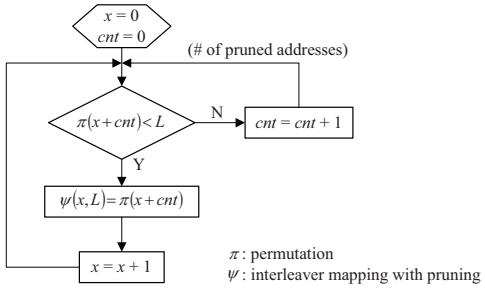


Figure 1: Flowchart of a sequential pruned interleaver.

A major disadvantage of pruned interleavers is that, despite their simplicity, interleaved addresses must be generated sequentially. That is, in order to generate the interleaved address corresponding to a linear address x, the interleaved addresses of all linear addresses less than x must first be generated. This follows from the fact that the number of pruned addresses that have occurred before x must be known in order to know where x gets mapped to. This requirement introduces a latency bottleneck, especially when (de-)interleaving and turbo encoding/decoding long packets. Fig. 1 shows the flowchart of a generic pruned interleaver ψ that interleaves a sequence of linear addresses from 0 to $L-1$ according to some permutation π with pruning.

In [8], the concept of parallel lookahead pruned interleaving (PLPI) was introduced to address the disadvantage of pruning latency on interleaver address generation, in particular, the feedback loop on the right in Fig. 1 that keeps track of the number of pruned addresses. The scheme, as illustrated in Fig. 2, interleaves a packet of length L in logarithmic time complexity by dividing it into P blocks each of size N, interleaving the P blocks in parallel, and then concatenating the results. Each lookahead block in Fig. 2 (designated by ϕ) computes the number of pruned addresses in the blocks to its left, thus enabling each block to be interleaved independently from the blocks preceding it. This results in a speedup by a factor of P over the sequential scheme in Fig. 1. The compu-

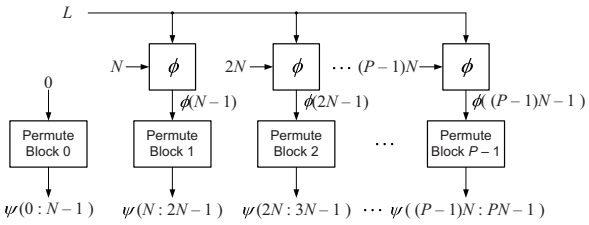


Figure 2: Parallel lookahead pruned interleaving scheme.

tations done by the lookahead ϕ-blocks depend on the underlying interleaving permutation; however, the overall scheme applies to any sequentially pruned interleaver.

In this paper, we develop an efficient architecture based on the PLPI scheme for turbo interleavers employed in the emerging Ultra Mobile Broadband (UMB) standard part of the 3rd Generation Partnership Project 2 (3GPP2) [1].

2. TURBO INTERLEAVING FOR UMB

The turbo interleaver adopted in UMB is based on linear congruential sequences (LCS) [9]. The sequence of interleaver output addresses generated by an LCS turbo interleaver is equivalent to the sequence obtained by the following process. A 2D-array is filled with a sequence of linear addresses row by row from top to bottom, the entries of the array are shuffled according to a procedure to be described next, and the resulting shuffled entries are read column by column from left to right. The shuffling of the array entries is based on applying an independent permutation to the entries in every row, and then permuting the order of the rows. In UMB, the number of rows is $2^r = 32$. Next, the smallest positive integer n such that $L \leq 2^{r+n}$ is determined. This is equivalent to finding the smallest sized $2^r \times 2^n$ array that can hold L entries. The 2^n entries of each row are interleaved independently using a LCS recursion whose parameters are determined using a 2D look-up table (LUT) based on the row index and n. This results in a set of new interleaved column indices. Next, the 2^r rows are shuffled in bit-reversed order. The result of this operation is a set of new interleaved row indices. Finally, the interleaved addresses are formed by concatenating the corresponding interleaved column and row indices in opposite order with respect to their order in the linear address. If the resulting interleaved address is greater than or equal to L, then it is pruned away and the same operations are repeated on the next consecutive address in linear order.

Let x be an $(r+n)$-bit linear address, and $\rho_{r,n}(x)$ be the corresponding $(r+n)$-bit turbo-interleaved address [9]:

$$\rho_{r,n}(x) = 2^n \cdot \pi_r(x \bmod 2^r) + \left[\left[\left(\left\lfloor \frac{x}{2^r} \right\rfloor + 1\right) \bmod 2^n\right] \times \mathrm{LUT}\left(x \bmod 2^r, n\right)\right] \bmod 2^n \quad (1)$$

where π_r is the r-bit-reversal function [2] and LUT is a 2D look-up table that stores the moduli of the 2^r LCS recursions for every n. Due to pruning of addresses, the address generated by $\rho_{r,n}(x)$ in (1) is not always valid, and hence not all integers in the interval from 0 to x have valid mappings. Denote by $\psi_{r,n}(x, L)$ the function that generates valid addresses under the mapping $\rho_{r,n}(x)$ for all integers between 0 and x. Obviously, if $L = 2^{r+n}$, then there are no pruned addresses and $\psi_{r,n}(x, L)$ coincides with $\rho_{r,n}(x)$. However, if $L < 2^{r+n}$, this equality no longer holds, but rather

$$\psi_{r,n}(x, L) = \rho_{r,n}\left(x + \phi_{r,n}\left(x, L\right)\right), \quad (2)$$

where $\phi_{r,n}(x, L)$ is the minimum number to be added to x such that the interval from 0 to $x+\phi_{r,n}(x, L)$ contains exactly $x + 1$ valid addresses when mapped by (1). The problem reduces to determining $\phi_{r,n}(x, L)$ [see Fig. 3].

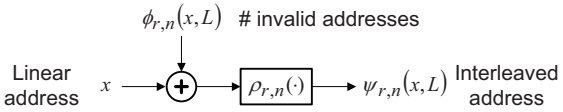


Figure 3: Relationship between $\phi_{r,n}$, $\rho_{r,n}$, and $\psi_{r,n}$.

From (1), $\rho_{r,n}(x)$ involves both a bit-reversal mapping and a linear congruential sequence mapping. In [10], the problem was solved for the bit-reversal mapping and corresponding architectures were developed. In the following, we consider the linear congruential sequence mappings, and then combine the results to determine $\phi_{r,n}(x, L)$.

Let $s(\cdot)$ be a linear congruential sequence defined as

$$s(c, m, x) = c \cdot (x + 1) \bmod m,\ 0 \leq x < m, \quad (3)$$

and let $T(c, m, \alpha, \beta)$ be the number of integers between 0 and some α whose image under (3) is between 0 and some β, where $\alpha \geq 0$, $\beta \geq 0$. Using the direct approach of counting the number of such integers as we step through the sequence values in (3) by comparing $s(c, m, x)$ to α for all $0 \leq x \leq \alpha$ has time complexity proportional to x. In [8], it was shown that the number of such integers is given by [8]

$$T(c, m, \alpha, \beta) = \begin{cases} \left\lfloor \frac{\alpha}{m} \right\rfloor \beta + S(c, m, \alpha \bmod m, \beta), & \text{if } \beta < m; \\ \alpha, & \text{otherwise.} \end{cases} \quad (4)$$

where

$$S(c, m, \alpha, \beta) = \frac{\alpha\beta}{m^2} + D(c, m, c + \alpha c - \beta, c + \alpha c, c, c - \beta) + E. \quad (5)$$

and E is a constant. The function $D(\cdot)$ can be evaluated iteratively in t iterations using integer arithmetic as:

$$D(h, k, c_1, c_2, c_3, c_4) = \sum_{j=1}^{t} \sum_{i=1}^{4} (-1)^{i+j+2} \times \left(\frac{b_i[j](c_i[j] + c_i[j+1])p[j-1]}{2h[1]} - \frac{b_i[j]}{2} - \frac{e(h[j+1], c_i[j])}{4} \right). \quad (6)$$

where $c_1 = c + \alpha c - \beta$, $c_2 = c + \alpha c$, $c_3 = c$, $c_4 = c - \beta$, and

$$a[j] = \left\lfloor \frac{h[j]}{h[j+1]} \right\rfloor, \quad h[j+2] = h[j] \bmod h[j+1],$$
$$b_i[j] = \left\lfloor \frac{c_i[j]}{h[j+1]} \right\rfloor, \quad c_i[j+1] = c_i[j] \bmod h[j+1],$$
$$p[j] = a[j]p[j-1] + p[j-2], \tag{7}$$

for $i = 1, \cdots, 4$, $j = 1, \cdots, t$, with initial conditions $h[1] = k$, $h[2] = h$, $c_i[1] = c_i, i = 1, \cdots, 4$, $p[0] = 1$ and $p[1] = a[1]$. The function $e(h, c)$ is defined to be 1 if $c \neq 0$ or $c \bmod h = 0$, and 0 otherwise.

Next, let $\sigma_{r,n}(\alpha, \beta)$ be the number of integers x between 0 and $\alpha - 1$ (inclusive) such that $\rho_{r,n}(x) \geq \beta$, where $\alpha \geq 0$, $\beta \geq 0$. Then $\sigma_{r,n}(\alpha, \beta)$ is given by:

$$\sigma_{r,n}(\alpha, \beta) = \sigma'_{r,n}(\alpha, \beta) + \sigma''_{r,n}(\alpha, \beta) \tag{8}$$

where

$$\sigma'_{r,n}(\alpha, \beta) = \left\lfloor \frac{\alpha - 1}{2^r} \right\rfloor \times \left(2^r - \left\lfloor \frac{\beta}{2^n} \right\rfloor - 1 \right) + \phi_r \left((\alpha - 1) \bmod 2^r, \left\lfloor \frac{\beta}{2^n} \right\rfloor + 1 \right), \tag{9}$$

$$\sigma''_{r,n}(\alpha, \beta) = \alpha'' - T(c, 2^n, \alpha'', \beta''),$$
$$c = \mathrm{LUT}\left(\pi_r \left(\left\lfloor \frac{\beta}{2^n} \right\rfloor \right), n \right), \tag{10}$$
$$\alpha'' = \left\lfloor \frac{\alpha - 1}{2^r} \right\rfloor + \begin{cases} 1, & \text{if } \pi_r \left(\left\lfloor \frac{\beta}{2^n} \right\rfloor \right) \leq (\alpha - 1) \bmod 2^r; \\ 0, & \text{otherwise.} \end{cases}$$
$$\beta'' = \beta \bmod 2^n + 1.$$

In (9), ϕ_r is the number of addresses that are pruned by the bit-reversal map π_r [10].

3. PARALLEL INTERLEAVING ALGORITHM

To determine where an integer x gets mapped by $\rho_{r,n}(x)$ in the presence of pruning for an interleaver of length L, we first determine the number of pruned addresses in the interval from 0 to x using (8) as $\sigma_{r,n}(x + 1, L)$. Then, the interval is expanded to $x + \sigma_{r,n}(x+1, L)$ in order to include these pruned addresses, and the number of pruned addresses in the interval from 0 to $x + \sigma_{r,n}(x + 1, L)$ is determined. The process is repeated until a minimum sized interval that includes exactly x *valid* addresses, is reached.

Let $\sigma^{(k)}_{r,n}(x + 1, L)$ be the number of addresses pruned at the kth iteration. Then the number of pruned addresses at iteration $(k + 1)$ is given by

$$\sigma^{(k+1)}_{r,n}(x + 1, L) \leftarrow \sigma_{r,n} \left(x + \sigma^{(k)}_{r,n}(x + 1, L) + 1, L \right). \tag{11}$$

Algorithm 1 ϕ-Algorithm

procedure ϕ-ALGORITHM(n, L, x)
  $k \leftarrow 0$
  $\sigma^0_{r,n}(x + 1, L) \leftarrow 0$
  repeat
    $\sigma^{k+1}_{r,n}(x + 1, L) \leftarrow \sigma_{r,n} \left(x + \sigma^k_{r,n}(x + 1, L) + 1, L \right)$
    $k \leftarrow k + 1$
  until $\left(\sigma^{k+1}_{r,n}(x + 1, L) = \sigma^k_{r,n}(x + 1, L) \right)$
  $\phi_{r,n}(x, L) \leftarrow \sigma^k_{r,n}(x + 1, L)$
  $\psi_{r,n}(x, L) \leftarrow \pi_n \left(x + \phi_{r,n}(x + 1, L) \right)$
end procedure

The process is repeated until $\sigma^{(k+1)}_{r,n}(x + 1, L) = \sigma^{(k)}_{r,n}(x + 1, L)$. The pseudo-code of the ϕ-algorithm for computing $\phi_{r,n}(x, L)$ as defined in (2) is outlined in Algorithm 1.

The ϕ-algorithm for the map $\rho_{r,n}(x, L)$ converges in at most $\log(L) - 1$ iterations based on the observation that two consecutive addresses generated by $\rho_{r,n}(x)$ in (1) can not both be invalid due to the bit-reversal map on the r LSBs. Moreover, each iteration of the ϕ-algorithm involves computing $\sigma_{r,n}(x, L)$ according to (8), which requires at most $\log(L) - \min(r, n)$ steps to converge [8]. Hence the time complexity of the ϕ-algorithm is $\mathcal{O}(\log^2(L))$. For the choice of L between 2^7 and 2^{14} as defined in UMB [1], it can be shown through simulations that the algorithm converges in at most 45 iterations.

4. HARDWARE ARCHITECTURES

Fig. 4 shows the architecture for computing $T(c, m, \alpha, \beta)$ in (5) iteratively. The architecture uses basic integer adders, subtractors, unsigned multipliers, and unsigned integer dividers building blocks. The dividers generate the quotient as well as the remainder (modulo), and have comparable complexity to an unsigned array multiplier. The look-up table stores the values of the constant E in (5) [8]. Intermediate registers that store results between iterations are shown highlighted in the figure.

Fig. 5a shows the architecture for computing $\sigma'_{r,n}(\alpha, \beta)$ in (9), where the ϕ-block implements the ϕ-algorithm for the bit-reversal map π_r [10]. The blocks in the figure designated by $\gg$ and $\ll$ perform logical right and left shifting respectively, while the "LSB" block in the figure passes the least-significant bits as indicated by the top input to these blocks.

Fig. 5b shows the architecture for computing $\sigma''_{r,n}(\alpha, \beta)$ in (10) using the T-block shown in Fig. 4 to compute $T(c, 2^n, \alpha'', \beta'')$. The "BREV" block in the figure reverses the order of the bits of its input (physically just using wires without any logic). The logic to the left of the T-block in Fig. 5b generates the appropriate inputs to the T-block according to (10).

Both the σ' and the σ'' blocks are combined in Fig. 6 according to (8) to compute $\phi_{r,n}(\alpha, \beta)$ iteratively using the ϕ-algorithm. The iterations performed by (11) over $\sigma_{r,n}(\alpha, \beta)$

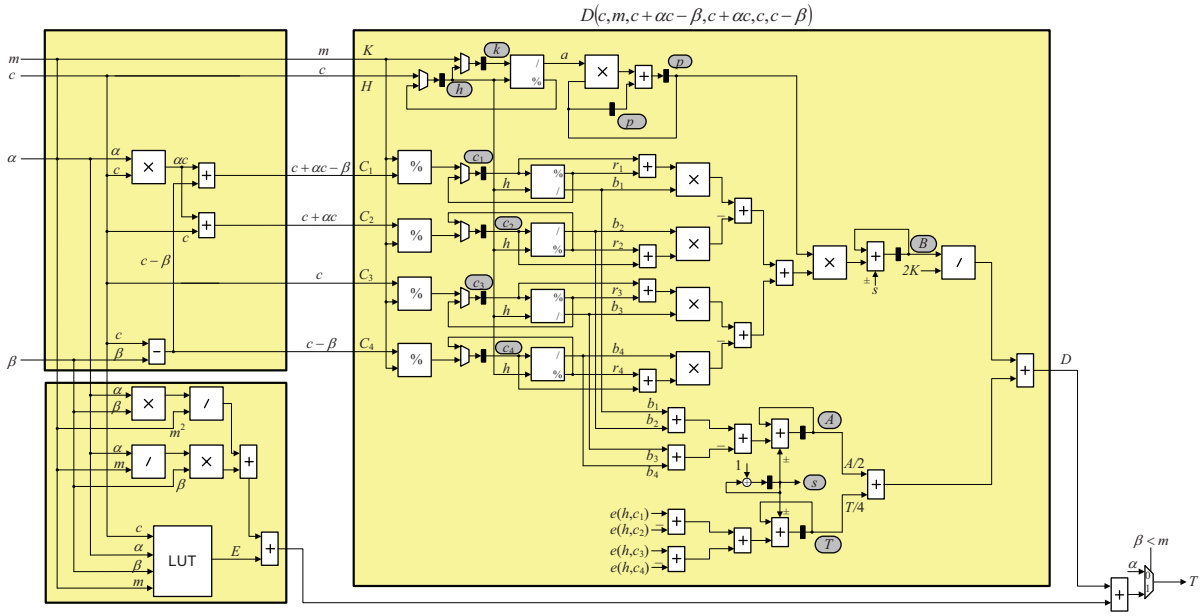
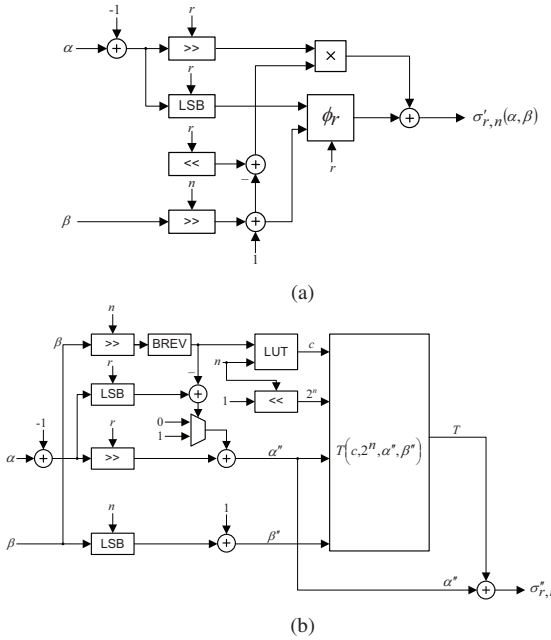


Figure 4: Architecture for computing $T(c, m, \alpha, \beta)$ in (4).



(a)

(b)

Figure 5: Architectures for computing (a) $\sigma'_{r,n}(\alpha, \beta)$, and (b) $\sigma''_{r,n}(\alpha, \beta)$.

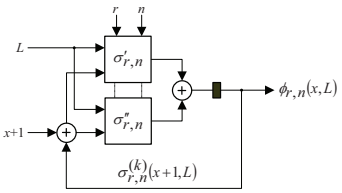


Figure 6: Architecture for computing $\phi_{r,n}(\alpha, \beta)$ iteratively.

to generate $\phi_{r,n}(x, L)$ are done using the feedback connection shown in Fig. 6. As shown in the figures, the architectures employ basic arithmetic circuits that can be implemented efficiently in hardware.

Referring to the parallel lookahead interleaving scheme in Fig. 2, if the ϕ-blocks implement the ϕ-algorithm shown in Fig. 6 for the interleaver map $\rho_{r,n}$, then the lookahead interleaver would correspond to a parallel UMB turbo interleaver.

5. REFERENCES

[1] 3rd Generation Partnership Project 2 (3GPP2), *Physical layer for Ultra Mobile Broadband (UMB) air interface specification*, 2007.

[2] M. M. Mansour, "Parallel channel interleavers for 3GPP2/UMB," in *IEEE Workshop on Sig. Proc. Sys. (submitted for review)*, Oct 2008.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *IEEE Int. Conf. on Communications, Geneva, Switzerland*, 1993, pp. 1064–1070.

[4] F. Daneshgaran and P. Mulassano, "Interleaver pruning for construction of variable-length turbo codes," *IEEE Trans. on Inf. Theory*, vol. 50, pp. 455–467, Mar 2004.

[5] M. Eroz and A. R. Hammongs Jr., "On the design of prunable interleavers for turbo codes," in *IEEE Vehicular Technology Conference*, May 1999, vol. 2, pp. 1669–1673.

[6] L. Dinoi and S. Benedetto, "Design of fast-prunable s-random interleavers," *IEEE Trans. on Wireless Comm.*, pp. 2540–2548, Sep 2005.

[7] L. Dinoi and S. Benedetto, "Variable-size interleaver design for parallel turbo decoder architectures," *IEEE Trans. on Comm.*, vol. 53, no. 11, pp. 1833–1840, Nov 2005.

[8] M. M. Mansour, "Parallel lookahead algorithms for pruned interleavers," *submitted to IEEE Trans. on Communications*, Dec 2007.

[9] F. Ling and D. N. Rowitch, "Turbo code interleaver using linear congruential sequence," Oct 2001, U.S. Patent 6304991.

[10] M. M. Mansour, "A parallel pruned bit-reversal interleaver," *To appear in IEEE Trans. on VLSI Systems*.