Efficient SIMD technique with parallel Max-Log-MAP Algorithm for Turbo Decoders

David Gnaedig, Mathias Lapeyre, Florent Mouchou, Emmanuel Boutillon

Efficient SIMD technique with parallel Max-Log-MAP Algorithm for Turbo Decoders

David Gnaedig
TurboConcept / ENST Bretagne / LESTER - UBS
Technopôle Brest-Iroise
115, rue Claude Chappe
29280 Plouzané, France
david.gnaedig@turboconcept.com

Mathias Lapeyre
Université de Bretagne Sud
Centre de recherche
BP 92116
56321 Lorient Cedex, France
mathias.lapeyre1@etud.univ-ubs.fr

Florent Mouchoux
Université de Bretagne Sud
Centre de recherche
BP 92116
56321 Lorient Cedex, France
florent.mouchoux1@etud.univ-ubs.fr

Emmanuel Boutillon
Université de Bretagne Sud - Centre de recherche
BP 92116 - 56321 Lorient Cedex, France
emmanuel.boutillon@univ-ubs.fr

Abstract : This paper presents a new SIMD technique to implement efficiently on a DSP a parallel Max-Log-MAP algorithm for turbo decoders. It consists in using SIMD instructions to perform several independent trellises in parallel. This trellis parallelization is made possible by the use of an adapted two-dimensional turbo code and its parallel interleaver structure. After a brief description of the Max-Log-MAP algorithm, the implementation of a fixed point algorithm for 8-bit SIMD operations is discussed. The parallel Max-Log-MAP algorithm and the associated memory organization are described. Performance results show the effectiveness of this technique that achieves 4 times the throughput of a classical sequential implementation.

***Keywords*:** turbo codes, slice turbo code, SIMD, parallelization, Max-Log-MAP algorithm

1. INTRODUCTION

Since the introduction of turbo codes [1], there has been considerable interest in those error correcting codes. A turbo decoder consists of several concatenated soft-output decoders, each of which decodes part of the overall code and then passes "soft" reliability information in an iterative scheme. The component soft-output algorithm described in the original turbo code paper [1] is usually known as the maximum *a posteriori* (MAP), forward–backward (FB), or Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm [2]. Usually, for implementation of a fast turbo-decoder on a Digital Signal Processor (DSP), its simplified version called the Max-Log-MAP algorithm working in the logarithmic domain is implemented [3].

The Max-Log-MAP algorithm is the key component of a turbo-decoder. It performs the soft decoding of a convolutional code using two finite Viterbi recursions on a trellis: the forward and the backward recursions. The basic computations involved in a trellis stage computation are Addition-Comparison-Selection (ACS) operations. The throughput of a turbo-decoder is limited by the recursions involved in the Max-Log-MAP processing of each dimension, which cannot be parallelized easily.

Loo *et al.* proposed to speed up the decoding throughput of the Max-Log-MAP algorithm on a DSP by using its Single Instruction Multiple Data (SIMD) functionality [4]. For DSP offering SIMD operations, parallel (vectorized) computations can be achieved simultaneously, on a set of data called a vector. Their SIMD technique performs, for a given trellis stage, several ACS operations. This method improves the decoding throughput but is not optimal. In fact, because of the structure of the trellis, the data need to be reordered into a single vector at the end of each trellis stage. This operation requires 2 instructions and reduces the efficiency of the SIMD technique. The use of this technique is optimal when all operations can be vectorized, *i.e.* when there is no need to pack/unpack data into a single vector to execute a SIMD operation.

To overcome this problem, this paper proposes a new technique to implement efficiently the Max-Log-MAP algorithm on a DSP. The proposed method exploits the SIMD architecture of a processor to achieve parallel processing of several Max-Log-MAP algorithms by vectorizing all operations involved in the Max-Log-MAP decoding for turbo decoders. Hence, each variable in the Max-Log-MAP algorithm is vectorized as a vector of P components: each component of the vector is associated to one of the P trellises decoded. This trellis parallelization is made possible by the use of an adapted two-dimensional turbo code proposed in [5] and called Multiple Slice Turbo Codes.

In the present paper, the adapted turbo-code is described with its interleaver properties. After a brief

description of the Max-Log-MAP algorithm, the key architectural features for implementing the algorithm with SIMD functionality are given (memory organization and SIMD operations).

1. MULTIPLE SLICE TURBO CODES

The idea of Multiple Slice Turbo Codes (MSTC) was proposed by Gnaedig et al. [5]. The motivation of MSTCs construction was to design an adapted turbo code suitable for an efficient parallel implementation. It is made possible by an increase by a factor *P* (the number of slices) of the decoding parallelism of the turbo-decoder without memory duplication. This technique leads to efficient parallel turbo decoder FPGA or ASIC implementations.

An adapted turbo code can also be designed to decode efficiently a turbo decoder on a DSP by using SIMD techniques. It has been noticed that the properties of Multiple Slice Turbo Codes are suitable for efficient vector operations on a DSP. Indeed, this adapted turbo-code is constructed, in each dimension, as the parallel concatenation of *P* independent Circular Recursive Systematic Convolutional (CRSC) codes, called slices. These *P* slices can be decoded in parallel with a Max-Log-MAP using vectorized operations.

Moreover, an appropriate parallel interleaver is also proposed in [5]. This parallel property avoids conflicts in parallel memory accesses. Therefore, no memory duplication is required and a single memory can be used. In addition, thanks to its structure, the interleaver maintains the vector organization between the natural and the interleaved order. This vectorization property leads in an efficient implementation, because there is no need to shuffle data by unpacking / repacking into vectors between half-iterations.

In this section, the construction of Multiple Slice Turbo Codes is described. Then, it will be shown how the structure and the design of the parallel interleaver enables a vectorization and a parallellization of the decoding.

1.1. Code construction

Multiple Slice Turbo Codes are constructed as follows. An information frame of *N m*-binary symbols is divided into *P* blocks (called "slices") of *M* symbols, where $N = M \cdot P$. The resulting turbo code is denoted (*N*,*M*,*P*). As with a classical convolutional turbo code, the coding process is first performed in the natural order to produce the coded symbols of the first dimension. Each slice is encoded independently with a Circular Recursive Systematic Convolutional (CRSC) code. The information frame is then permuted by an *N* symbol interleaver. The permuted frame is again divided into *P* slices of size *M* and each of them is encoded independently with a CRSC code to produce the coded symbols of the second dimension. Puncturing is applied to generate the desired code rate.

The interleaver is constructed jointly with the memory organization to allow parallel decoding of the *P* slices. In other words, at each symbol cycle *k*, the interleaver structure allows the *P* decoders to read and write the *P* necessary data symbols from the *P* Memory Banks MB_0, MB_1,…, MB_{P-1} without conflict. Indeed, only one read can be made at any given time from a single port memory: in order to access *P* data symbols in parallel, *P* memory banks are necessary. With the solution described in the present paper, the degree of parallelism can be chosen according to the requirements of the application.

The next section presents the parallel interleaver construction, ensuring the parallelism constraint while maintaining good performance.

1.2. Parallel interleaver design

The interleaver structure is mapped onto a hard-ware architecture allowing a parallel decoding process. Figure 1 presents the interleaver structure used to construct a multiple slice turbo code constrained by decoding parallelism.

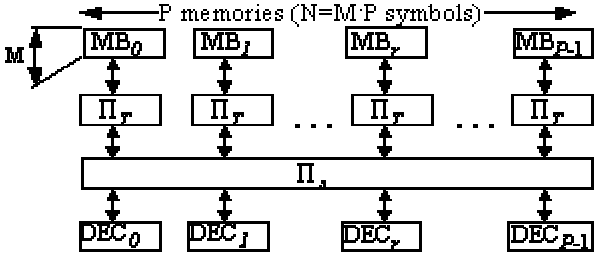

Figure 1: Interleaver structure for the (*N*,*M*,*P*) code

Let *l* and *k* denote the indices of the symbols in the natural and interleaved order, respectively. The coding process is performed in the natural order on independent consecutive blocks of *M* symbols. The symbol with index *l* is used in slice $\lfloor l/M \rfloor$ at temporal index time *l* mod *M,* where $\lfloor . \rfloor$ denotes the integral part function. Likewise, in the interleaved order, the symbol with index *k* is used in slice $r = \lfloor k/M \rfloor$ at temporal index $t = k \bmod M$. Note that $k = M \cdot r + t$, where $r \in \{0..P-1\}$ and $t \in \{0..M-1\}$. For each symbol with index *k* in the interleaved order, the permutation Π associates a corresponding symbol in the natural order with index $l = \Pi(k) = \Pi(t,r)$. The interleaver function can be split into two levels: a spatial permutation $\Pi_S(t,r)$ (ranging from 0 to *P*-1) and a temporal permutation $\Pi_T(t,r)$ (ranging from 0 to *M*-1), as defined in (1) and described in Figure 1.

$$\Pi(k) = \Pi(t,r) = \Pi_S(t,r) \cdot M + \Pi_T(t,r) \qquad (1)$$

The symbol at index *k* in the interleaved order is read from the memory bank $\Pi_S(t,r)$ at address $\Pi_T(t,r)$. While decoding the first dimension of the code, the frame is processed in the natural order . The spatial and temporal permutations are then simply replaced by *Identity* functions. The spatial permutation allows the *P* data read out to be transferred to the *P* SISO units

(denoted SISO in Figure 1). Decoder *r* receives the data from memory bank $\Pi_S(t,r)$ at instant *t*. This spatial permutation guarantees the parallel property of the interleaver.

1.3. Design of the permutations

The temporal $\Pi_T(t,r)$ and $\Pi_S(t,r)$ spatial permutations are chosen to simplify the implementation on a DSP (address computation, memory organization, simple DSP operations, ...), while maintaining good performance. To guarantee the vectorization property of the interleaver, the same temporal permutation is chosen for all memory banks. Hence, $\Pi_T(t,r) = \Pi_T(t)$ depends only on the temporal index *t*. The temporal permutation is then given by:

$$\Pi_T(t) = \alpha \cdot t + \beta(t \bmod 4) \bmod M \,, \qquad (2)$$

where α is relatively prime with *M*, and $(\beta(i))_{0\le i<4}$ are four coefficients inferior or equal to *M,* which verify that their values modulo 4 are all different.

The spatial permutation is defined as a circular rotation:

$$\Pi_S(t,r) = \left(A(t \bmod P) + r\right) \bmod P \,, \qquad (3)$$

where *A* is a bijection of variable $t \in \{0,...,P-1\}$ to $\{0,...,P-1\}$. The choice of the temporal and spatial permutation (2) and (3) can be easily implemented by additions and circular shift operations, respectively. Their optimization described in [6] in more details leads to very good performance, where it has also been shown that this adapted turbo code introduces no performance degradation compared to a conventional turbo code.

1.4. A simple example of an interleaver

Let us construct a simple (18,6,3) code to clarify the interleaver construction. Let the temporal permutation be $\Pi_T(t) = \{1,4,3,2,5,0\}$ and the spatial permutation a circular shift of amplitude $A(t \bmod 3)$, i.e. the slice of index *r* is associated to the memory bank of index $\Pi_S(t,r) = \left(A(t \bmod 3) + r\right) \bmod 3$, with $A(t \bmod 3) = \{2,0,1\}$. The spatial permutation is then bijective and 3-periodic.

The interleaver is illustrated on Figure 2, which shows the permutations for the 3 temporal indexes *t*=0 (2.a), *t*=1 (2.b) and *t*=5 (2.c). The 18 symbols in the natural order are separated into 3 slices of 6 symbols corresponding to the first dimension. In the second dimension, at temporal index *t*, symbols $\Pi_T(t)$ are selected from the 3 slices of the first dimension, and then permuted by the spatial permutation $\Pi_S(t,r)$. For example, at temporal index $t = 0$ (a), symbols at index $\Pi_T(0) = 1$ are selected. Then, they are shifted with an amplitude $A(0 \bmod 3) = 2$. Thus, symbols 1 from slices 0,1 and 2 of the first dimension go to slices 2, 0 and 1 of the second dimension respectively.

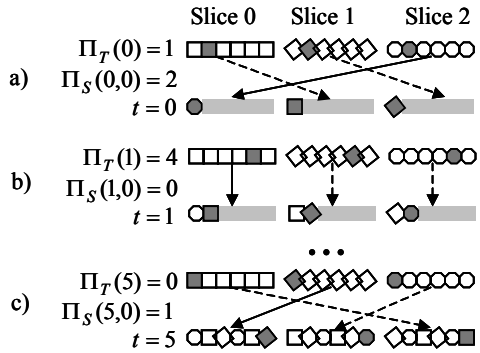

Figure 2 : A basic example of a (18,6,3) code.

2. THE MAX-LOG-MAP ALGORITHM

In this section the Max-Log-MAP algorithm for a binary MSTC used for decoding each slice of *M* bits is described by using the same notations than in [11].

The CRSC code is represented by a circular trellis of *M* stages. For each information bit u_k, the convolutional encoder produces the coded bits $X_k = (x_k^i)_{0\le i<r}$, where *r* is the number of bits produced by the convolutional code. These coded bits are transmitted over an additive white gaussian noise channel (AWGN). The decoder receives the noisy bits $Y_k = (y_k^i)_{0\le i<r}$ corresponding to the transmitted bits.

The Max-Log-MAP algorithm [3] provides for each information bit u_k, $k = 0,..., M-1$ a soft output $L(u_k)$, which represents the *a posteriori* probability for the bit u_k. For iterative decoding, another soft output $L_e(u_k)$ called the extrinsic information is computed from $L(u_k)$. This latter is computed by using the all received values of the channel $Y_k = (y_k^i)_{0\le i<r}$, $k = 0,..., M$-1 and the *a priori* information $L_a(u_k)$ provided by the other decoder, which is its extrinsic soft output.

The output of the channel and the *a priori* decoder inputs are used to compute the branch metrics (are used to process the forward and backward recursions.) between state *s*' at stage *k* and *s* at stage *k*+1 of the trellis by a scalar product between the received vector Y_k at time *k*:

$$c_k(s',s) = \sum_{i=1}^{r} x_k^i(s',s) \cdot y_k^i + L_a(u_k) \qquad (4)$$

The soft estimate $L(u_k)$ is computed by exhaustively exploring all possible paths in the trellis using a forward recursion and a backward recursion. This algorithm consists of three steps (Figure 3):

- Forward recursion: The forward state metrics a_k are recursively calculated using the bits in an increasing order from 0 to *M*-1:

$$a_k(s) = \max_{(s,s')} \left(a_{k-1}(s') + c_k(s',s)\right), \qquad (5)$$

where s' and s are the generic states at the (k-1)th and kth nodes respectively.
The forward state metrics are stored in an internal memory in order to be used to compute the soft output.

- Backward recursion: The backward state metrics b_k are recursively computed using the bits in a decreasing order from *M-1* to *0*.

$$b_k(s) = \max_{(s,s')}\left(b_{k+1}(s') + c_{k+1}(s,s')\right) \quad (6)$$

- Soft-Output Computation. The soft output for each bit at time k is computed by using the backward state metric b_k and the corresponding forward state metric a_{k-1} read from the memory.

$$L(u_k) = \max_{\substack{(s',s)\\ u_k=1}}\left(a_{k-1}(s') + c_k(s',s) + b_k(s)\right) - \max_{\substack{(s',s)\\ u_k=0}}\left(a_{k-1}(s') + c_k(s',s) + b_k(s)\right) \quad (7)$$

The extrinsic soft output is computed from the soft ouput $L(u_k)$ by subtracting the input of the decoder:

$$L_e(u_k) = L(u_k) - L_a(u_k) - 2 \cdot y_k^1, \quad (8)$$

where y_k^1 is the received bit corresponding to the transmitted bit $x_k^1 = u_k$.

Since the trellis of the code is circular, the initial state metrics of the forward and backward recursions a_0 and b_{M-1} are provided by the final state metrics of the previous iterations (see Figure 3). At the first iteration, these initial state metrics are set to the all zero vector.

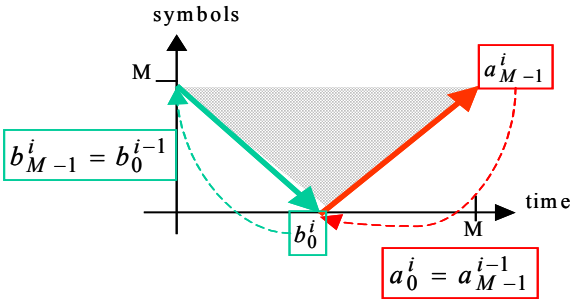

Figure 3: Graphical representation of the Forward-Backward Max-Log-MAP algorithm

3. TURBO DECODER IMPLEMENTATION

In this section, the algorithm described in the previous section is processed in parallel on several trellises by using the SIMD technique of the DSP. After a brief description of DSP architectures and SIMD techniques, the implementation of the turbo decoder is described.

3.1. VLIW and SIMD techniques for DSP

Current DSPs can achieve several thousands of million of operations per seconds. This high number of operations is achieved thanks to two architectural concepts. The first concept consists in Very Long Instruction Word (VLIW). It consists in processing several operations simultaneously by several processing units. Usually, high performance fixed point DSPs are processing operations on 32 bits. The second architectural concept to increase the parallelism in the DSP is to use an Single Instruction Multiple Data (SIMD) technique. This technique enables to process two 16-bit or four 8-bit identical operations with a single instruction on a single 32-bit processing unit. The set of two or four data that are processed simultaneously is called a vector. The operations processing a vector of data is said to be "vectorized".

It is not an easy task to achieve an efficient utilization of the VLIW technique, *i.e.* to write a code that uses all processing units continuously. Hence, this task is left to the compiler. The turbo decoder is thus described with a fixed point C code. This C code uses explicitly vector operations and has an adapted memory organization suitable for efficient use of the SIMD capability.

3.2. Fixed point Max-Log-MAP algorithm

When using vectorized operations, the number of bits used to code internal variable has to be lower or equal to the number of bit of a SIMD operation. For example, for a 32-bit processor processing 4 operations of 8-bit on one SIMD processing unit, the maximum number of bits for the internal variables is 8. If one internal variable is higher than 8, the corresponding operations can be done in parallel only by two 16-bit operations and additional cycles are needed to unpack and repack the data after and before this operation. This solution decreases the efficiency of the parallel implementation. The other solution consists in using 16-bits operations for the whole algorithm, but the parellelism and hence the throughput are divided by two. This motivates the need for techniques that reduce to 8 bits the internal precision of the Max-Log-MAP algorithm.

The conversion of a floating point Max-Log-MAP algorithm to a fixed point algorithm has been widely described in the literature [8]. The key parameter is the precision of the inputs of the decoder, *i.e.*. the number of bit *wd* used to code the decoder inputs. This parameter incluences the performances and also determines the internal size w_n of the state metrics.

The problem that arises when implementing a Viterbi recursion, is the increase of the state metrics. This problem has been intensively studied for the Viterbi algorithm [9] and solutions using rescaling or modulo 2^{wn} arithmetic are widely used [10]. These techniques are based on the fact that, at every instant, the dynamic range of a state metric (i.e., the difference between the state metrics with the highest and lowest values), is bounded. This upper bound is derived from the constraint length of the code and the maximum value of

the branch metrics. It enables us to use a simple rescaling technique. For our fixed point implementation, the max operations in equations (5-7) are replaced by min operations, by changing the sign of the branch metrics (4). Thus, at each trellis state, the state metrics are normalized by subtracting the minimal state metric. This solution leads to the least number of bits w_n to store the state metrics.

For our application, the constituent code used is a duo-binary code of rate 2/3 [7]. The equations of the Max-Log-MAP algorithm for a duo-binary turbo code are similar to (4)-(7). It produces 4 reliabilities for the 4 possible transmitted symbols and the number of branches leaving each trellis state is equal to 4.

After demodulation, the ouput of the channel are provided to the decoder as quantized unsigned values of w_d bits. The extrinsic information are also coded on $w_e=w_d$ bits. Hence, from (4), the branch metric can be coded on w_d+2 bits. In addition, the dynamic range of a state metrics is bounded by twice the maximum value of the branch metrics. Thus, the state metrics are coded on w_d+3 bits for equations for (5) and (6). In equation (7), the addition of the forward state metric and the backward state metrics requires w_d+4 bits to code the soft output. After computation of the extrinsic information using (8), the 4 values of extrinsic information for each symbol are normalized by subtracting the minimum value. Then these unsigned values are thresholded to be stored on $w_e=w_d$ bits. From this study, the maximum number of bits for an internal value of the decoder is w_d+4. This number must be inferior or equal to the maximum number of bits for a single SIMD operation, *i.e.* 8. This leads to $w_d = 4$ bits to code the inputs of the decoder. In section 4, the performance of the decoder with $w_d = 4$ bits is compared to those of an infinite precision decoder.

3.3. Parallel Max-Log-MAP algorithm and memory organization

The fixed point Max-Log-MAP algorithm described in the previous section is implemented with the SIMD technique. For a DSP with a SIMD parallelism enabling *P*-component vector operations, the adapted turbo code is designed with a parallelism of *P*. Hence, *P* different trellises corresponding to *P* slices can be decoded in parallel. Each component of an internal vector corresponds to an internal variable of the Max-Log-MAP algorithm for one trellis. Hence, all operations of the Max-Log-MAP algorithm are vectorized. Therefore, a memory organization is needed for the input and outputs of the Max-Log-MAP algorithm. This memory organization allows us to simply use vectorized operations for the parallel Max-Log-MAP algorithms, without extra cycles for packing / unpacking of data. In addition, this memory organization verifies the vectorization property of the interleaver. The *P* inputs of the Max-Log-MAP algorithm are regrouped in vectors of *P* component, with one vector for *P* data that are used at the same time, in the same vector operation.

Because of the vectorization property of the interleaver, only one address computation and one memory access in needed to read one vector from the memory. Only a circular shift is needed in the interleaved order to implement the spatial rotation of the interleaver. This operation can be done in a single cycle.

4. RESULTS

The trellis parallelization technique has been implemented on a DSP TMS320C6416: four parallel Max-Log-MAP algorithms are performed on $P = 4$ independent trellises by using 8-bit SIMD operations.

4.1. Implementation

The fixed point Max-Log-MAP algorithm with $w_d = 4$ bits is implemented in C language. The SIMD instructions are specified by using intrinsic functions. Program 1 gives the corresponding C code implementing the forward recursion. To reduce the complexity, the rescaling of the state metrics is not performed at each trellis stage. They are only rescaled when at least one of the state metrics exceed 2^{w_d-1}.

The SIMD implementation of the backward recursion and the extrinsic information computation are similar to the one of the forward recursion. They are therefore not described here.

```
test = 0x00000000;
/* ACS */
for(s = 0; s < number_state; s++) {
   /* Addition */
   for (b = 0; b < 4; b++) {
      met_br = _add4(br_met[s][b],ap_in[b]);
      fwd_br[s][i] = _add4(fwd_in[s],met_br);
   }
   /* Comparison and Selection */
   tmp0 = _minu4(
      fwd_br[previous_state[s][0]][0],
      fwd_br[previous_state[s][1]][1]);
   tmp1 = _minu4(
      fwd_br[previous_state[s][2]][2],
      fwd_br[previous_state[s][3]][3]);
   fwd_out[s] = _minu4(tmp0,tmp1);
   /* Test */
   if (!test)
      test |= fwd_out[s] & 0x80808080;
}

/* Rescaling */
if (test) {
  fwd_min = fwd_out[0];
   for(s = 1; s < number_state; s++) {
      fwd_min = _min4(fwd_min, fwd_out[s]);
   }
   for(s = 0; s < number_state; s++) {
      fwd_out[s] = _sub4(fwd_out[s], fwd_min);
   }
}
```

Program 1: SIMD description of the forward algorithm.

4.2. Performance

The performance of the fixed point algorithm with $wd = 4$ is compared to the performance of an infinite precision floating point algorithm. The simulated performance of turbo decoders using these algorithms are shown in Figure 4 for the same duo-binary turbo code of 512 bits constructed with $P = 4$ slices of 64 symbols. Eight iterations have been performed for the two algorithms.

The performance degradation of the algorithm with $wd = 4$ bits is only 0.1 dB compared to the infinite precision algorithm, which is the same than for a fixed point algorithm with $wd = 8$. This result shows that increasing the number of input bits of the algorithm does not provide a huge improvement. It has been shown in section 0 that with $wd > 4$ bits, only 16-bit SIMD operations can be used. This reduces the thoughput of the Max-Log-MAP algorithm by a factor of two.

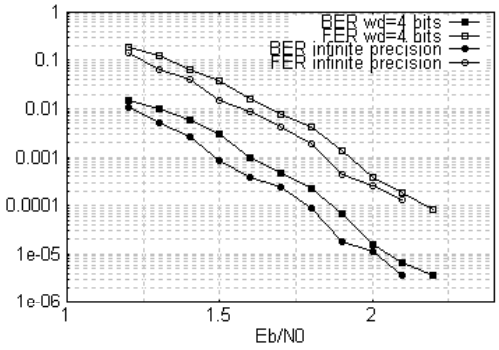

Figure 4: Performance comparison of the Max-Log-MAP algorithm for a (256,64,4) duo-binary turbo-code: the algorithm with wd = 4 inputs bits is compared to the infinite precision algorithm.

4.3. Throughput

The Max-Log-MAP algorithm has been simulated on the DSP for the duo-binary turbo code (256,64,4). The number of cycles for performing the Max-Log-MAP algorithm for one dimension and 8 iterations respectively are given in Table 1. The number of cycles does not increase linearly with the number of iterations, because the branch metrics are computed only once at the first iteration and stored into the memory.

	Number of cycles	Throughput @ 600 MHz
One dimension	292 554	1050 kbits/s
Turbo decoder @ 8 it.	1 927 344	160 kbits/s

Table 1: Number of cycles and throughput for the Parallel Max-Log-MAP algorithm and the turbo-decoder with 8 decoding iterations.

The implementation results shows that for a 600 MHz clock frequency, the parallel Max-Log-MAP algorithm used in a turbo decoder with 8 decoding iterations achieves a throughput of 160 kbits/s. It is exactly 4 times the throughput of a sequential implementation without using the SIMD technique.

5. CONCLUSION

A new SIMD technique for the Max-Log-MAP algorithm has been presented in this paper. This technique uses an adapted turbo, whose interleaver structure enables us to use SIMD operations for all computations and memory accesses. The degradation in performance with 8-bit SIMD operations is only 0.1 dB from an infinite precision algorithm with a throughput of 160 kbits/s, 4 times that of a sequential implementation. With 16-bit SIMD operations there is no performance degradation, but the throughput is only doubled. Futher work includes designing a complete turbo decoder and optimizing its implementation to further increase the thoughput.

6. REFERENCES

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes", *Proc. ICC'93*, Geneva, Switzerland, pp. 1064-1070, May 1993.

[2] L.R Bahl, J. Cocke, F. Jelinek, J. Raviv, " Optimal decoding of linear codes for minimizing symbol error rate ", IEEE Transactions on Information Theory, pp. 284-287, mars 1974.

[3] L. Papke and P. Robertson, "A Viterbi algorithm with soft-decision outputs and its applications", *Proc. IEEE Int. Conf. Commun.*, Dallas, pp. 102-106, June 1996.

[4] K.K. Loo, T. Alukaidey, S.A. Jimaa, K. Salman, *"SIMD Technique for Implementing the Max-Log-MAP Algorithm",* GSPx 2003.

[5] D. Gnaedig, E. Boutillon, V. C. Gaudet, M. Jézéquel, P. G. Gulak, "On Multiple Slice Turbo Codes", in *Proc. International Symposium on Turbo Codes and Related Topics*, Brest, pp. 343-346, Sept. 2003.

[6] ____ , "On Multiple Slice Turbo Codes", *submitted to Annales des Télécommunications*

[7] DVB-RCS Standard, "Interaction channel for satellite distribution systems", *ETSI EN 301 790*, V1.2.2, pp. 21-24, Dec 2000.

[8] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 871–882, May 2001.

[9] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in Proc. *IEEE Int. Conf. Communications* (ICC '90), vol. 4, Atlanta, GA, Apr. 16–19, 1990, pp. 1723–1728.

[10] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," IEEE Trans. VLSI Syst., vol. 7, pp. 369–379, Sept. 1999.

[11] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.