High Performance Parallelised 3GPP Turbo Decoder

K. K. Loo, T. Alukaidey, S. A. Jimaa
Department of ECEE
University of Hertfordshire
College Lane, AL10 9AB, UK

Abstract **– The maximum *a-posteriori* based turbo decoder is extremely complex due to the large number of identical operations repeated in sequence for the processing of huge data volume. This also introduces a significant decoding delay which is undesirable for real-time applications as required in the 3G communication systems. However, this scenario shows that the algorithm is highly parallelisable. In this paper, we propose a parallelised max-Log-MAP (P-max-Log-MAP) model which exploits the sub-word parallelism (SWP) and very long instruction word (VLIW) architecture of a microprocessor or a DSP. The proposed model reduces considerably the complexity of max-Log-MAP algorithm wherein it maximises the decoding performance of 3GPP turbo codes; therefore facilitates easy implementation. The proposed model is implemented on the Analog Devices s TigerSHARC dual-core DSP. Based on this dual-cores DSP, we also propose a parallel sliding window (P-SW) scheme where two P-max-Log-MAP component decoders are able to work in parallel for the processing of two sliding windows. We show that the turbo decoder uses the P-max-Log-MAP combined with the P-SW to achieve decoding throughput exceeding 2 Mbps on a single chip DSP implementation.**

Keyword **– Parallelised max-Log-MAP, Parallel sliding window, Turbo decoder, SIMD, SWP, VLIW, DSP**

1 Introduction

In the context of recursive algorithms for decoding of turbo codes, the maximum *a-posteriori* (MAP) algorithm [1] is a probabilistic orientated algorithm used to minimise bit errors. The algorithm computes probability information recursively through decoding of the trellis in forward and backward manners. The decoding bits are estimated based on the pre-computed probabilities by using logarithm likelihood ratio (LLR) to evaluate the weight between information "1" and "0". In fact, the optimal MAP algorithm inherently involves multiplication and exponential operators which requires intense computations. It is also highly sensitive to numerical precision due to its linear operations. The algorithm is therefore not attractive for practical implementation even though it has a merit of outstanding performance. Instead, the approximate Log domain of the MAP algorithm such as Log-MAP and max-Log-MAP, which solely involve addition, subtraction and max operators, are well received for practical implementation.

Figure 1 illustrates the 3rd Generation Partnership Project (3GPP) standard turbo encoder [2] which has two constituent Recursive Systematic Convolutional (RSC) encoder with constraint length, $K = 4$. For decoding of such component encoder, we consider the max-Log-MAP algorithm. This algorithm works on large number of Add-Compare-Select (ACS) operation. The ACS is the basic and most intensive operation which repeated in sequence to compute metric recursively for each trellis state in the forward and backward manner over huge data volume underlying the trellis. This operation introduces a great deal of computational complexity to the max-Log-MAP algorithm. Due to its excessive decoding latency and memory used, the max-Log-MAP algorithm for decoding of turbo codes may not be attractive to meet the high performance and time sensitive applications such as wireless multimedia applications as demanded in the 3G communication systems. It can be shown that max-Log-MAP algorithm lends itself to parallelism.

In this paper, the P-max-Log-MAP model [3] which attempts to provide solution to the above difficulties is presented. The proposed model borrows the single-instruction multiple-data (SIMD) concept to reduce the computational complexity of the max-Log-MAP s forward and backward recursions. The proposed model fully exploits the SWP and VLIW architecture of a microprocessor or a DSP to achieve high level of parallelism. The SWP allows single instruction for processing of several different data in parallel within a defined data width. With the combination of VLIW architecture, at least two SWP instructions can be executed in parallel. In other words, a greater data parallelism can be achieved. The proposed model is implemented on the Analog Devices TigerSHARC DSP where the processor has two cores and each core supports both the SWP and VLIW instructions operating in lock-step. The P-SW scheme is also proposed to fully exploit this architecture where two P-max-Log-MAP component decoders are able to work in parallel for the processing of two sliding windows. In this process, two *a-posteriori* LLR soft values are computed simultaneously. As a result, the turbo decoder uses the P-max-Log-MAP in combination with the P-SW scheme has achieved decoding throughput exceeding 2 Mbps on a single chip DSP implementation.

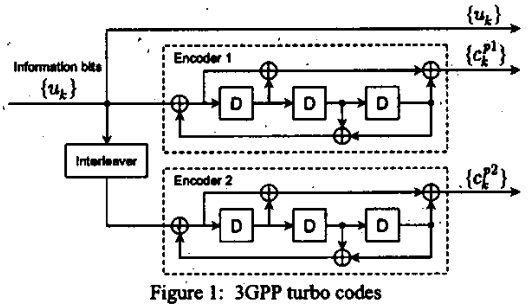

Figure 1: 3GPP turbo codes

The paper is organised as follows: in Section 2, a simplified iterative decoder is introduced where the simplification eliminates the need for an extra interleaver/deinterleaver. Sections 3 and 4 describe the max-Log-MAP algorithm and the simplification for the 3GPP turbo codes. In sections 5 and 6, the proposed P-

max-Log-MAP and P-SW are presented. Finally, in sections 7 and 8, the performance of the proposed turbo decoder is shown and the conclusion is drawn.

2 Iterative Decoder

In a general communication system, information bits u_k are grouped into frames of N bits and encoded with the turbo encoder consisting of RSC codes, whose encoder structure is shown in Figure 1. The turbo encoder generates one parity bit c_k^{p1} associated with each input information bit u_k and another parity bit c_k^{p2} associated with each interleaved input information bit $\tilde{u}_k$. At the receiver, the matched filter outputs $y_k = \{y_k^s, y_k^{p1}, y_k^{p2}\}$ are presented to the turbo decoder. A simplified iterative decoder structure is shown in Figure 2.

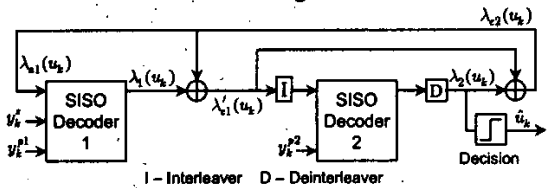

Figure 2: Simplified iterative decoder

The iterative max-Log-MAP soft-input soft-output (SISO) decoder calculates the *a-posteriori* probability LLR for the bit u_k using the following equations:

$$\lambda(u_k) = \max_{u_k=1}\left[\alpha_{k-1}(s_{k-1}) + \gamma_1(y_k, s_{k-1} \leftarrow s_k) + \beta_k(s_k)\right] - \max_{u_k=0}\left[\alpha_{k-1}(s_{k-1}) + \gamma_0(y_k, s_{k-1} \leftarrow s_k) + \beta_k(s_k)\right]. \quad (1)$$

The purpose of the first SISO decoder lies in yielding the extrinsic information $\lambda_{e1}(u_k)$ which would be interleaved and applied as the *a-priori* information to the second SISO decoder, $\lambda_{a2}(u_k)$. The extrinsic information of the first SISO decoder is obtained as follows:

$$\lambda_{e1}(u_k) = \lambda_1(\hat{u}_k) - y_k^s - \lambda_{a1}(u_k), \quad (2)$$

where $\lambda_{a1}(u_k)$ is the *a-priori* information provided by the second SISO decoder after first iteration, and $\lambda_1(u_k)$ is the *a-posteriori* LLR for the bit u_k calculated in the first half iteration. The systematic bit y_k^s in (2) does not necessarily need to be scaled by the channel reliability value because the turbo decoding with max-Log-MAP decoder is SNR independent [4]. Inherently, the systematic bit y_k^s is not directly accessible by the second SISO decoder due to fact that the systematic bit was interleaved at the second encoder. Beside the interleaver for the extrinsic information $\lambda_{e1}(u_k)$, in the classic iterative decoder, an extra interleaver for the systematic bit is required y_k^s, so that the interleaved version of the systematic bit $\tilde{y}_k^s$ will match with the second set of parity bits y_k^{p2} to be processed in the second half iteration. In this simplified iterative decoder, the extrinsic information $\lambda_{e1}(u_k)$ and the systematic bit y_k^s of the first SISO decoder are combined as follows:

$$\lambda_{e1}(u_k)' = \lambda_{e1}(u_k) + y_k^s, \quad (3)$$

which may be obtained by $\lambda_{e1}(u_k)' = \lambda_1(\hat{u}_k) - \lambda_{a1}(u_k)$, and interleaved at once before delivered to the second SISO decoder. Since the first SISO decoder has direct access to the systematic bit y_k^s, thus the second SISO decoder requires to compute only the extrinsic information $\lambda_{e2}(u_k)$, which will be used directly as the *a-priori* information to the first SISO decoder, $\lambda_{a1}(u_k)$, as visualised in Figure 2. The relationship between $\lambda_{e2}(u_k)$ and $\lambda_{a1}(u_k)$ is given by

$$\begin{aligned}\lambda_{e2}(u_k) &= \lambda_2(\hat{u}_k) - \lambda_{e1}(u_k)' \\ &= \lambda_2(\hat{u}_k) - \lambda_{e1}(u_k) - y_k^s \triangleq \lambda_{a1}(u_k).\end{aligned} \quad (4)$$

By doing so, the extra interleaver and deinterleaver in the turbo decoder can be eliminated.

3 Max-Log-MAP Algorithm

The sub-optimal max-Log-MAP algorithm is chosen in this paper rather than the optimal Log-MAP algorithm because of its reduced complexity without significant performance degradation. The algorithm is divided into three groups of computations that can be summarised as follows:

1) Branch metric computation for branching between states s_{k-1} to s_k:

$$\gamma_k(y_k, s_{k-1}, s_k) = 0.5\left[\lambda_a(u_k)x_k^s + y_k^s x_k^s + y_k^p x_k^p\right], \quad (5)$$

where $x_k = (x_k^s, x_k^p)$ is the input/output symbol of the encoder for each branch between state s_{k-1} to s_k, $y_k = (y_k^s, y_k^p)$ is the received channel symbol and $\lambda_a(u_k)$ is the *a-priori* information.

2) In the forward and backward recursions:

- Forward metric computation for valid transitions $(s_{k-1} \rightarrow s_k)$ and $(s'_{k-1} \rightarrow s_k)$ are:

$$\alpha_k(s_k) = \max_{(s_{k-1}, i) \in S_f}\left[\alpha_{k-1}(s_{k-1}) + \gamma_i(y_{k-1}, s_{k-1} \rightarrow s_k)\right], \quad (6)$$

where S_f is a set of states connected to s_k.

- Backward metric computation for valid transitions $(s_{k-1} \leftarrow s_k)$ and $(s_{k-1} \leftarrow s'_k)$ are:

$$\beta_{k-1}(s_{k-1}) = \max_{(s_k, i) \in S_b}\left[\beta_k(s_k) + \gamma_i(y_k, s_{k-1} \leftarrow s_k)\right], \quad (7)$$

where S_b is a set of states connected to s_{k-1}.

3) The LLR computation is as described in (1) above.

4 Simplified Max-Log-MAP Algorithm

In this section, we simplify the max-Log-MAP algorithm especially for the decoding of the 3GPP turbo codes, whose trellis representation is shown in Figure 3. Given the encoder input and output, $u_b = u_k$ and $c_b = c_k^p$ respectively, a possible bit value "1" or "0" feeding to the input u_b, the output c_b is generated depending on the current state, memory, and polynomial generator of the encoder [5]. The trellis diagram in Figure 3 also highlights all the possible transitions corresponding to the input/output symbol (u_b, c_b). Since the symbol (u_b, c_b)

is directly corresponding to $(x_k^s, x_k^p) \in \{+1, -1\}$, the branch metric of (5) can be expressed as follows:

$$\gamma_{u_k c_k} = 0.5\left[\lambda_a(u_k)x_k^s + y_k^s x_k^s + y_k^p x_k^p\right], \quad (8)$$

where $u_k c_k \equiv x_k^s x_k^p$ has four possibilities that are symmetrical, and thus only the customised calculations of the branch metric are required as given below:

$$\begin{aligned}\gamma_{11} &= 0.5\left[\lambda_k(u_k) + y_k^s + y_k^p\right],\\ \gamma_{10} &= 0.5\left[\lambda_k(u_k) + y_k^s - y_k^p\right].\end{aligned} \quad (9)$$

Therefore, referring to (9), the forward and backward metrics calculations can be simplified as follows:

$$\begin{aligned}\alpha_k(0) &= \max\left[\alpha_{k-1}(0) - \gamma_{11}, \alpha_{k-1}(1) + \gamma_{11}\right]\\ \alpha_k(1) &= \max\left[\alpha_{k-1}(2) + \gamma_{10}, \alpha_{k-1}(3) - \gamma_{10}\right]\\ &\vdots\\ \alpha_k(4) &= \max\left[\alpha_{k-1}(0) + \gamma_{11}, \alpha_{k-1}(1) - \gamma_{11}\right]\\ \alpha_k(5) &= \max\left[\alpha_{k-1}(2) - \gamma_{10}, \alpha_{k-1}(3) + \gamma_{10}\right],\\ &\cdots\end{aligned} \quad (10)$$

$$\begin{aligned}\beta_{k-1}(0) &= \max\left[\beta_k(0) - \gamma_{11}, \beta_k(4) + \gamma_{11}\right]\\ \beta_{k-1}(1) &= \max\left[\beta_k(0) + \gamma_{10}, \beta_k(4) - \gamma_{10}\right]\\ &\vdots\\ \beta_{k-1}(4) &= \max\left[\beta_k(2) + \gamma_{11}, \beta_k(6) - \gamma_{11}\right]\\ \beta_{k-1}(5) &= \max\left[\beta_k(2) - \gamma_{10}, \beta_k(6) + \gamma_{10}\right].\\ &\cdots\end{aligned} \quad (11)$$

Considering the *a-posteriori* LLR to be computed in the backward recursion, and the backward state transition $(s_{k-1} \leftarrow s_k)$ is valid, parallel operation of (1) can be written as follows:

$$\lambda(u_k) = \max_{u_k=1}\left\{\begin{matrix}\max\left[\varphi(0,4),\varphi(1,0)\right], \max\left[\varphi(2,1),\varphi(3,5)\right],\\ \max\left[\varphi(4,6),\varphi(5,2)\right], \max\left[\varphi(6,3),\varphi(7,7)\right]\end{matrix}\right\}$$
$$-\max_{u_k=0}\left\{\begin{matrix}\max\left[\varphi(0,0),\varphi(1,4)\right], \max\left[\varphi(2,5),\varphi(3,1)\right],\\ \max\left[\varphi(4,2),\varphi(5,6)\right], \max\left[\varphi(6,7),\varphi(7,3)\right]\end{matrix}\right\}, \quad (12)$$

where $\varphi(s_{k-1}, s_k) = \alpha_{k-1}(s_{k-1}) + \gamma_{u_k c_k} + \beta_k(s_k)$.

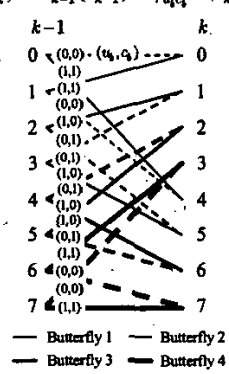

Figure 3: Trellis of the 3GPP turbo codes

5 Parallelised Max-Log-MAP Model

Consider the simplified max-Log-MAP algorithm for the decoding of 3GPP turbo codes with constraint length $K = 4$ and polynomial generator $g(D) = 15/13$. We show that the forward recursion, backward recursion, and *a-posteriori* LLR calculation of the max-Log-MAP algorithm can be highly parallelised at both data and instruction parallelism. Another technique of parallelising the max-Log-MAP algorithm is presented in [5], where data parallelism is applied. Here, assumptions are made where the forward recursion is first executed, and then followed by the backward recursion wherein the backward metrics are calculated in the same loop with the *a-posteriori* LLR. The branch metrics, γ_{10} and γ_{11}, are computed and stored in memory which will be retrieved when computing the forward and backward metrics.

In addition we assumed that a typical SWP architecture of a processing unit is either a microprocessor or a DSP that supports 8-bit ALU operations as minimum. With data widths of 64-bit, the SWP instruction is capable of computing eight ACS operations over eight 8-bit different data individually in parallel as illustrated in Figure 4. The computations of eight forward or backward metrics usually require 16 add/sub operations and 8 max operations. However, by using a single VLIW s SWP instruction, 16 add/sub operations can be performed in 1 cycle, 0.5 cycles for each add/sub operation, and 8 max operations in 1 cycle. Our model also requires that the data positions within a defined field to be arranged for a proper match of computation. For example, an arbitrary vector $z_s = \{z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$ may be arranged $\tilde{z}_s = \{z_4, z_7, z_3, z_6, z_2, z_0, z_5, z_1\}$ which matches $v_s = \{v_4, v_7, v_3, v_6, v_2, v_0, v_5, v_1\}$ for possible computations. The arrangement can be done by simple mapping of $z_s \mapsto \tilde{z}_{p(s)}$ where the permutation index p is related to v_s.

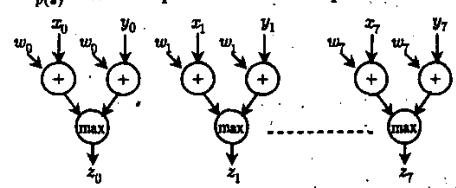

Figure 4: Parallel ACS operations

In the following sub-sections, we show the complete computational flow of our P-max-Log-MAP model that consists of the forward recursion, the backward recursion and the *a-posteriori LLR* calculation in which the detail of each computation is discussed.

Forward Recursion:

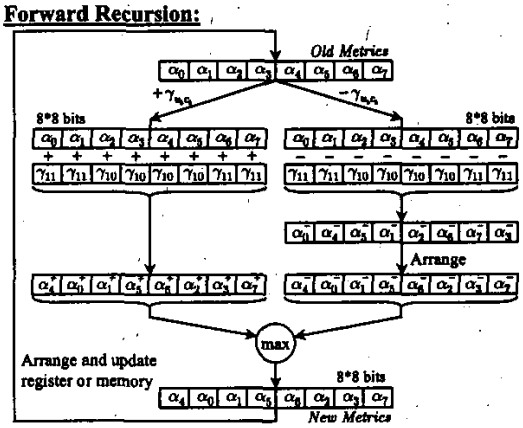

Figure 5: Computational flow of forward recursion

The forward metrics $\alpha_s = \alpha_{0,\ldots,7}$ are initialised and stored in a register as shown in Figure 5. The branch metrics, $\gamma_{11}\gamma_{10}$, are retrieved from memory and arranged according to the trellis branch transitions. The branch metrics of other polynomial generators within $K=4$ can be arranged according to new branch transitions with a corresponding permutation index p. The trellis metrics $\alpha_s = \alpha_{0,\ldots,7}$ at time index $k-1$ will add/sub simultaneously with the corresponding branch metrics individually to yield, α_s^+ and α_s^- that represent the metrics related to bit value "1" and "0" respectively. After the operation, the position of α_s^- is arranged to match α_s^+, then a comparison is applied to select the survival metrics as new metrics, $\hat{\alpha}_s$, at index k. The $\hat{\alpha}_s$ is arranged and updated to register α_s and a copy is stored in memory where they will be retrieved when computing the *a-posteriori* LLR.

The proposed operations may be summarised as follows:

1) At $k=0$, initialise $\alpha_0 = 0$ and $\alpha_{1,\ldots,7} = -128$.
2) Retrieve branch metrics $\gamma_{u_k c_k}$ and arrange according to the forward branch transitions.
3) Perform add/sub operation, $\alpha_s^{+/-} = \alpha_s \pm \gamma_{u_k c_k}$.
4) Arrange $\alpha_{p(s)}^- = \alpha_s^-$ where the permutation index p is related to α_s^+.
5) Select survival metrics, $\hat{\alpha}_s = \max\left[\alpha_s^+, \alpha_{p(s)}^-\right]$.
6) Arrange and update $\hat{\alpha}_s$ to α_s by $\alpha_s = \hat{\alpha}_{p(s)}$.
7) If $k > N$ where N is the frame size, then forward recursion is completed, otherwise go back to step2.

Backward Recursion:

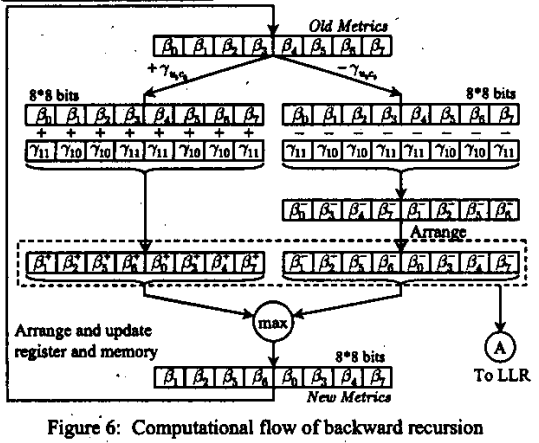

Figure 6: Computational flow of backward recursion

Figure 6 depicts the computation of the backward recursion which is identical to the forward recursion. Therefore, the backward metrics can be computed using the forward recursion method but with data collected from the backward trellis trace. The backward metrics are computed in the same loop with the *a-posteriori* LLR. As the new backward metrics are being computed, the metrics β_s^+ and β_s^- that are related to bit value "1" and "0" are kept in the register for the *a-posteriori* LLR calculation. The reference "A" in Figure 6 marks this situation. The new metrics are updated to β_s and are used for computing next set of metrics while executing the *a-posteriori* LLR. In order to eliminate excessive memory used, the metrics are not required to be stored in the memory. Next, we will discuss the *a-posteriori* LLR calculation which completes the backward recursion operations.

LLR Computation:

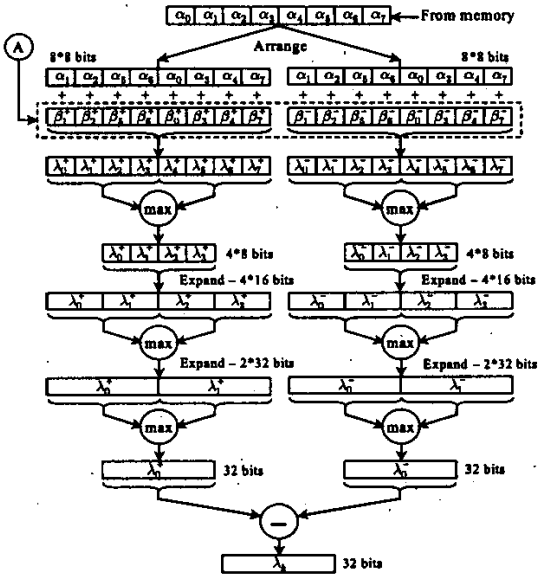

Figure 7: Computational flow of LLR computation

Figure 7 shows the computational flow of the *a-posteriori* LLR calculation which is a continuation from the reference "A" in Figure 6. Both the backward metrics, β_s^+ and β_s^- will be added with the forward metrics α_s in a matched position, yielding the *a-posteriori* probability soft values for bit value "1" and "0" represented as $\lambda_{s=0,\ldots,7}^+$ and $\lambda_{s=0,\ldots,7}^-$, respectively. The rest of the *a-posteriori* LLR calculation is to find maximum value of $\lambda_{s=0,\ldots,7}^+$ and $\lambda_{s=0,\ldots,7}^-$ by using a unique search procedure as shown in Figure 7. Finally, the *a-posteriori* LLR for the bit u_k is calculated by finding the ratio between the two maximum values as in (13):

$$\lambda_k = \max\left[\lambda_{s=0,\ldots,7}^+\right] - \max\left[\lambda_{s=0,\ldots,7}^-\right]. \qquad (13)$$

The proposed operations may be summarised as follows:

1) At $k=N$, initialise $\beta_0 = 0$ and $\beta_{1,\ldots,7} = -128$.
2) Retrieve branch metrics $\gamma_{u_k c_k}$ and arrange according to the backward branch transitions.
3) Perform add/sub operation, $\beta_s^+ = \beta_s \pm \gamma_{u_k c_k}$.

4) Arrange $\beta^-_{p(s)} = \beta^-_s$ where the permutation index p is related to β^+_s.
5) Select survival metrics, $\hat{\beta}_s = \max\left[\beta^+_s, \beta^-_{p(s)}\right]$.
6) Arrange and update $\hat{\beta}_s$ to β_s by $\beta_s = \hat{\beta}_{p(s)}$.
7) Retrieve forward metrics α_s from the memory and arrange to $\alpha_{p(s)} = \beta^{+/-}_s$.
8) Compute the *a-posteriori* probability for information "1" and "0" $\lambda^{+/-}_s = \beta^{+/-}_s + \alpha_s$.
9) Find maximum soft value, $\lambda^{+/-}_0 = \max\left[\lambda^{+/-}_{s=0,\ldots,7}\right]$
10) Finally, compute *a-posteriori* LLR by $\lambda_k = \lambda^+_0 - \lambda^-_0$.
11) If $k < 0$, then the backward recursion is completed, otherwise go back to step 2.

6 Parallel Sliding Window

The P-max-Log-MAP component decoder can be executed on a single core processor that supports SWP and VLIW operations. However, with a processor that has two cores in SIMD orientation, therefore two P-max-Log-MAP component decoders can be executed on a single set of operation. Based on this proposition, the received sequence may be segmented into two blocks and processed in parallel [5, 6]. In this section, the proposed P-SW is described in which two sliding windows are processed in parallel as shown in Figure 8. The train of thought and the sliding orientation given in this paper are different from the algorithms used in [6, 7].

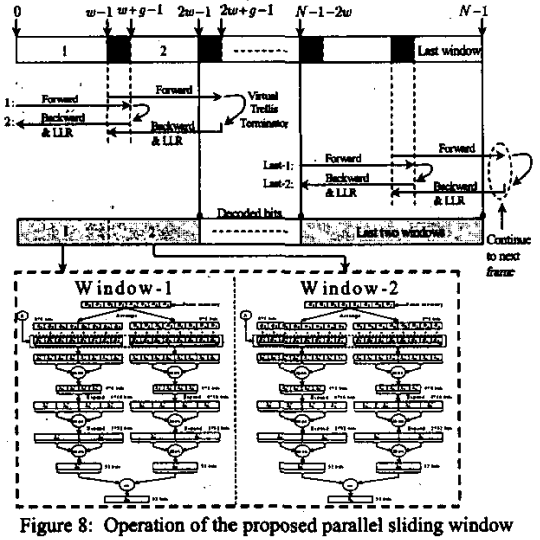

Figure 8: Operation of the proposed parallel sliding window

First of all, the P-SW allows continuous decoding of 3GPP turbo codes without trellis termination. The turbo decoder will operate after receiving two windows, $W = 2w$, where w is the window length, of information sequence. Since both the RSC encoders are not terminated, trellis of each window stop at an unknown state. Thus, the decoding performance may be affected. However, we employ a technique which virtually terminates the trellis by first executing the forward recursion, and then followed by the backward recursion. In our P-SW model, two forward recursions are executed in parallel where each recursion only works up to length w. Once the first two windows of information sequence are received, the forward recursion of window-1 starts from beginning of the trellis at time index $k = 0$, where the trellis metrics are properly initialised. As for window-2, the recursion starts at $k + w$, where the trellis metrics are initialised at equal probabilities. Since, the forward recursion of window-2 always starts with the metrics at equal probabilities; there will be a region where the trellis metrics may not be reliable. This region may spread up to $5K$ length of trellis [8], where K is the constraint length. Therefore, in order to compensate for this unreliable region, an overlapping window of length g satisfying the condition $g > 5K$, is applied as shown in Figure 8. In this case, forward and backward recursions should then work up to window length $w + g$ instead of w. After a proper learning period of length $w + g$, the trellis metrics of two windows should be mature and reliable. These pre-computed forward metrics are then used to initialise the backward recursion of window-1 and window-2 respectively, and each recursion again works up to length $w + g$. By doing so, the trellis decoding is virtually terminated. During this backward trace, two *a-posteriori* LLR soft values are computed simultaneously. Therefore, the decoding speed of the conventional sliding window [8] is doubled.

The proposed operations may be summarised as follows:

Forward recursion

1) Initialise forward metrics:
Window-1: If this is the first window from the beginning of the trellis, then

$$\alpha_0(s) = \begin{cases} 0 & s = 0 \\ -128 & s \neq 0 \end{cases}$$

Otherwise, $\alpha_0(s) = \alpha_{2w}(s)\ \forall s$

Window-2: $\alpha_w(s) = 0\ \forall s$

2) For $k = 1$ to $w + g - 1$, compute forward metrics:
Window-1: $\alpha_k(s) = \max_{\forall s}\left[\alpha_{k-1} + \gamma\right]$

Window-2: $\alpha_{k+w}(s) = \max_{\forall s}\left[\alpha_{k+w-1} + \gamma\right]$

Backward recursion

1) After the forward recursion is finished, initialise backward metrics as:
Window-1: $\beta_{w+g-1}(s) = \alpha_{w+g-1}(s)\ \forall s$

Window-2: $\beta_{2w+g-1}(s) = \alpha_{2w+g-1}(s)\ \forall s$

2) For $k = w + g - 1$ to 1, compute backward metrics:
Window-1: $\beta_{k-1}(s) = \max_{\forall s}\left[\beta_k + \gamma\right]$

Window-2: $\beta_{k+w-1}(s) = \max_{\forall s}\left[\beta_{k+w} + \gamma\right]$

3) Compute *a-posteriori* LLR:
Window-1: Valid from $k = w - 1$ to 0
Window-2: Valid from $k = 2w - 1$ to w

4) Go back to step 1, after receiving next two windows, $W = 2w$.

7 Implementation Results

Analog Devices TigerSHARC dual-core DSP is currently a high profile DSP in the market which aims to providing computational power to the implementation of advanced communication systems. It provides a comprehensive set of SWP and VLIW fixed-point instructions that are required for the implementation of our P-max-Log-MAP model. The TigerSHARC has two cores and each core supports both the SWP and VLIW instructions operating in lock-step which allow two P-max-Log-MAP component decoders to be executed on a single set of operation.

In this implementation, the input data is quantised into a minimum of 8-bits signed integer format. Besides saving a great amount of memory resources, multiple 8-bits operations can also be used as intended. For a single-core operation of the TigerSHARC DSP, sixteen 8-bits data can be processed in parallel on a single VLIW s SWP instruction. In the dual-cores SIMD operation, a total of thirty two 8-bit data can be processed at a time. Therefore, in terms of ACS operation, a total of 32 add/sub operations and 16 max operations can be performed in 1 cycle respectively. This shows that we have a great opportunity to achieve high level of parallelism in the implementation of the 3GPP turbo decoder using the proposed model. The cycle count and decoding throughput of P-max-Log-MAP and P-SW turbo decoder are presented in Tables 1 to 4.

8 Conclusions

In this paper, the P-max-Log-MAP model has been presented. The proposed model exploits the SWP and VLIW architecture of a microprocessor or a DSP for reducing the computational complexity of the max-Log-MAP algorithm. In particular, the forward and backward recursions of the max-Log-MAP algorithm have been highly parallelised by employing both data and instruction parallelism. The proposed model has been implemented on the Analog Devices TigerSHARC DSP where the processor has two cores and is capable of executing two P-max-Log-MAP component decoders in parallel using the proposed P-SW scheme. The implementation result shows that the P-SW turbo decoder can decode, with 4 iterations, 3G traffic channels at a rate beyond 2 Mbps. Therefore, the decoding of large frame size ranging from 40 to 5114 bits with a demanding speed and performance, on a single DSP chip has been made possible. Consider the traffic channel at a rate 384 Kbps, the same turbo decoder can execute up to 25 iterations for $W = 1024$ at clock speed of 250 MHz. Therefore, we are highly confident that BER lower than 10^{-6} for real-time high quality wireless multimedia applications can be achieved easily. Instead, we may execute the decoder for 4 iterations and reduce the clock speed to 40 MHz, thus reduces power consumption.

P-max-Log-MAP	Latency (cycle)	Memory (bit)
Branch Metrics	7+(N/16)*8	2N*8
Forward Metrics	8+(N*5)	8N*8
Backward Metrics	10+(N*20)	-
LLR		N*16
Total:	25 + 25.5N	96N
Turbo Decoder	**Latency (cycle)**	**Memory (bit)**
2* P-max-Log-MAP	50+51N	96N

Table 1: Cycle count of P-max-Log-MAP turbo decoder

P-SW	Latency (cycle)	Memory (bit)
Branch Metrics	$7+[(W+g)/16]*8$	$2(W+g)*8$
Forward Metrics	$8+[(W+g)*5]/2$	$8(W+g)*8$
Backward Metrics	$10+[(W+g)*20]/2$	-
LLR		$W*16$
Total:	$25+13W+13g$	$96W+80g$
Turbo Decoder	**Latency (cycle)**	**Memory (bit)**
2* P-SW	$50+26W+26g$	$96W+80g$

Table 2: Cycle count of P-SW turbo decoder

N	4 Iterations (I)		Kbps	
	Cycles	(c/b)*I	200 MHz	250 MHz
256	13106	204.78	976.65	1220.8
4096	208948	204.05	980.16	1225.2

Table 3: Decoding throughput of P-max-Log-MAP turbo decoder

W	g	4 Iterations (I)		Mbps	
		Cycles	(c/b)*I	200 MHz	250 MHz
256	32	7538	104.69	1.910	2.388
1024	32	27506	104.19	1.919	2.399

Table 4: Decoding throughput of P-SW turbo decoder

9 Acknowledgement

The authors gratefully acknowledge the financial support from Analog Devices Inc., Boston, USA. This paper had appeared in part in the Electronics Letters [3].

10 References

[1] L.R. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal decoding of linear block codes for minimizing symbol error rate", *IEEE Trans. Inf. Theory*, 1974, IT-20, pp. 284–287.

[2] 3GPP TS 25.212 v4.3.0, "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD) (Release 4)". December 2001.

[3] K.K. Loo, K. Salman, T. Alukaidey, S.A. Jimaa, "Parallelised Max-Log-MAP Model", *IEE Electron. Lett.*, August 2002, Vol. 38, Issue. 17

[4] A. Worm, P. Hoeher, N. When, "Turbo-Decoding without SNR Estimation", *IEEE Commun. Lett.*, 2000, Vol. 4, Issue 6, pp. 193–195.

[5] K.K. Loo, P.Y. Yip, T. Alukaidey, S.A. Jimaa, "Novel implementation of max-log-MAP algorithm on SIMD SHARC DSP", Proceedings of the 3rd SHARC®2001 International DSP Conference, Northeastern University, Boston, MA, USA, September 2001, pp. 150–159.

[6] J.M. Hsu, C.L. Wang, "A Parallel decoding scheme for Turbo Codes", Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, ISCAS'98, 1998, Vol. 4, pp. 445–448.

[7] R. Akella, J.K. Wolf, "On the parallel MAP algorithm", IEEE Fourth Workshop on Multimedia Signal Processing, 2001, pp. 371–376.

[8] X. Li, W.T. Song, H.W. Luo, "Design and analysis of Turbo decoder for Chinese third generation mobile communication system", The 7th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2000, 2000, Vol. 2, pp. 680–683.

[9] A.J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes", *IEEE J. Sel. Areas Commun.*, 1998, Vol. 16, Issue 2, pp. 260–264.