TURBO PRODUCT CODE IMPLEMENTATION ON ALTIVEC EXTENSION TO POWERPC

Sonali Agarwal (DEAL, Dehradun, Uttarakhand, India; sonaliagr@gmail.com);
Jasvinder Singh (DEAL, Dehradun, Uttarakhand, India; d.jasvinder@gmail.com); and
M.H.Rahaman (DEAL, Dehradun, Uttarakhand, India; dealdrdo@del2.vsnl.net.in)

ABSTRACT

Real time implementation of coding schemes suitable for radio application with high throughput requirement in low SNR conditions poses a serious challenge to the overall radio design. Amongst various channel coding schemes, Turbo-Code is the most promising choice in many radio systems for high data rate services with superior BER performance. However, turbo decoding is highly computationally intensive, and its real time implementation requires considerable processing power. To address this issue, we propose a new Vectorized approach for implementing Turbo Product Code (TPC) Decoder on PowerPC based COTS hardware used for developing SDR based communication systems. In this paper, we have discussed the implementation and performance of Scalar as well as Vectorized version of TPC decoder based on extended block code BCH (16,11,4) on G4 generation of PowerPC MPC7447A. Both the versions of TPC decoder when compiled using the WindRiver diab Compiler (version 5.4.0) with the same compiling, linking and optimization options, the Vectorized approach offers both significant speedup and complexity reduction over the conventional Scalar implementation.

1. INTRODUCTION

Forward Error Correcting (FEC) block is one of the important components of an SDR system. It allows the SDR system to preserve spectral efficiency and improve the quality of the transmission, which is an important requirement in today's digital world. The demand for greater data throughput is increasing for all communication systems, and will continue to grow. At the same time, tolerance for data errors and data latency is diminishing, and that trend will like-wise persist. These demands play directly to the strength of robust FEC schemes. Turbo Product Codes bring the most value for communication systems that have high data rates, require low coding overhead, and exhibit the need for the utmost in data integrity [1].

Since TPC decoder involves exhaustive mathematical operations, many time-consuming functions of this algorithm can be optimized if the underlying parallelism is investigated and then applying parallel processing to speedup the execution [1].

General Purpose Processors (GPPs) are attractive for the digital signal processing requirements of software defined radio as most of the key communication kernels can be executed on these general purpose programmable platforms, instead of dedicated hardware. For platforms where the power budget allows the use of GPPs, development cost and time to market can be significantly reduced compared to dedicated DSPs. Major microprocessor vendors have announced high performance parallel processing extensions to their general purpose processors in an effort to concurrently address high bandwidth data-processing and algorithmically intensive applications [2]. This paper jointly discusses the processor architecture and characteristics of the decoding algorithm and proposes a SIMD-friendly implementation of TPC decoder. Although described here in the AltiVec context, this implementation is applicable to any SIMD (Single Instruction Multiple Data) architecture.

The paper is organized as follows. Section 2 is dedicated to brief description of Turbo Product Codes and the decoding algorithm. Section 3 deals with the real-time Scalar PowerPC implementation of the decoder. A brief overview of SIMD architecture is portrayed in Section 4. The Vectorized implementation of TPC decoder using AltiVec technology along with code optimizations is described in Section 5. Section 6 presents the promising results of throughput improvement of our Vectorized approach over conventional Scalar PowerPC implementation. Finally, concluding remarks are given in Section 7.

2. TURBO PRODUCT CODES

In 1993, Berrou presented turbo code that consists of two recursive systematic convolutional codes concatenated in parallel. Its outstanding performance is possible due to the employment of random interleavers and iterative soft-input

soft-output (SISO) decoding algorithms. An alternative to convolutional turbo codes is Turbo Product Codes. TPC is block product code with turbo decoding algorithm. Pyndiah has devised a low-complexity block turbo decoding algorithm which has three appealing features for high bit rate applications; First, it has inherited from the low complexity of algebraic decoders; Second, the turbo process converges in only four iterations [3]. Third, TPC has a structure not requiring additional interleaving. That's why the product code is a simple and efficient method to construct powerful codes using a classic linear block code that combines high coding gain and acceptable delay.

Let us consider two systematic linear block codes C_1 having parameters (n_1, k_1, δ_1) and C_2 having parameters (n_2, k_2, δ_2) where n_i, k_i and δ_i (i=1, 2) stand for code length, number of information symbols and minimum Hamming distance respectively. The product code $P = C_1 \otimes C_2$ is obtained by:

- placing (k_1 x k_2) information bits in an array of k_1 rows and k_2 columns,
- coding the k_1 rows using code C_2,
- coding the n_2 columns using code C_1.

The parameters of the resulting product code P is given by $n=n_1xn_2$, $k=k_1xk_2$ and $\delta=\delta_1x\delta_2$ and the code rate R by $R=R_1xR_2$ where R_i is the code rate of code C_i.

For this implementation, we have used a product code P with $C_1 = C_2$ = BCH (16, 11, 4) which is an extended BCH code. The minimum distance δ is then equal to 16 and the code rate 0.473.

2.1. TPC Decoder

The main idea behind iterative decoding is to break up the decoding problem into a sequence of stages or iterations, such that each stage utilizes the output from the previous stages to formulate its own result.

Amongst various decoding algorithms, Chase-II Algorithm for iterative decoding of TPCs presents a possibility for optimal decoding, has lesser computational complexity for the same performance and offers ample flexibility in terms of performance-complexity tradeoff [4]. Some key points involved in the decoding algorithm are as follows [5]:

- The component block codes are decoded using soft decision decoding,
- Iterative decoding with the Hamming threshold is used, that not only results in a significant coding gain but also a faster iterative process [6],
- Small values (<0.5) are used for constants α (weight factor) and β (reliability factor) in the first iteration where the BER is high and gradually increased at each additional iteration.

The decoding procedure has been fully described in [7]. Thus, we shall here give only a brief description of the algorithm for the sake of clarity. The basic component of TPC Decoder is the SISO decoder used for decoding the rows and columns of product code. It is a modified Chase algorithm which starts by computing the maximum-likelihood (ML) codeword using the log-likelihood ratio (LLR) of the bits at the input of the SISO decoder. For each bit of the ML codeword, it then computes the log-likelihood ratio which is the soft output of the decoder. The reliability of the decision d_j is the LLR of transmitted symbol e_j which is given by the following relation [8]:

$$W_j = \begin{cases} \sum_{i=1, i\neq j}^{n} r_i c_i^d p_i & if\ c_j^d \neq c_j^c \\ \beta \times c_i^d & else \end{cases}$$

$$where \quad p_i = \begin{cases} 0 & if\ c_i^d = c_i^c \\ 1 & if\ c_i^d \neq c_i^c \end{cases}$$

where W_j is the extrinsic information of the j^{th} bit, r_i is the received codeword bit, c_i^d is the detected codeword bit(i^{th}) and c_i^c is the concurrent codeword bit(i^{th}). The basic flow of the algorithm is illustrated in Fig. 1. [8].

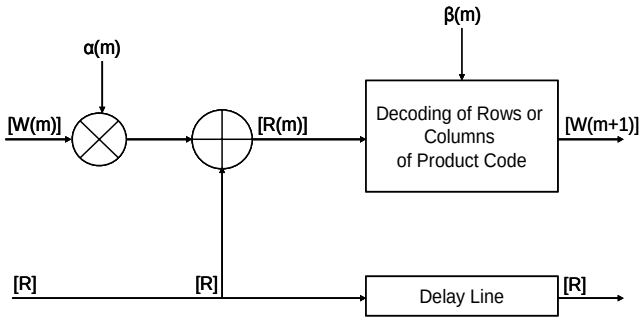

Fig.1. Block diagram of elementary TPC decoder (m^{th} half-iteration)

- [R] is the received codeword (row/column)
- [W(m+1)] is the additional information given by the previous decoder concerning the reliability of the decoded bit (extrinsic information),
- [R(m)] = [R] + α(m).[W(m)],
- α(m) and β(m) are weight and reliability constants respectively, which depend on the current half-iteration.

3. SCALER POWERPC IMPLEMENTATION

Conventional PowerPC implementation involves sequential processing i.e. computes one value at a time that introduces an extra overhead to call a routine number of times to perform the same operation on multiple pieces of data.

The major building blocks of TPC decoding algorithm are Hard Decision Computation, Forming Subset of Candidate Codewords, Metric Calculation and Extrinsic

Information Calculation [9]. These tasks are considered to be performance bottlenecks because of several mathematical operations like multiplication, accumulation, logical exclusive-OR etc. Their implementation involves excessive loops and branching that severely limits the execution speed of the decoding algorithm.

WindRiver's Workbench 2.5 IDE is used to edit/debug the TPC decoder source code and the object file is downloaded on Champ AV-IV board having PowerPC MPC7447A. The complete Scalar C code of decoding algorithm was compiled using diab compiler version 5.4.0 (WindRiver) with –XO–Xsize-opt optimization options to produce the fastest code possible. The 16Kbytes of object file when downloaded on the target platform yields 30 Kbps throughput (approx.). This low throughput restricts the Scalar implementation of TPC in high data rate applications developed over general-purpose platforms.

4. SIMD ARCHITECTURE

The primary requirements of hardware intended for Software Defined Radio systems are flexibility and programmability. The flexibility of the original SDR concept comes at the price of excessive demand for computational power and its associated problems. Selecting the best choice of processing resources is arguably the most critical problem faced in the course of developing any software radio or multi-waveform platform. GPPs have been a preferable choice for SDR systems due to their large amounts of program and user memory, floating-point operation and better high-level language development environments.

Various SIMD extensions to GPP have been classified on the basis whether the separate execution unit is employed for vector processing or the existing floating point unit is used. Some are constrained by backward compatibility and a desire to limit silicon investment to a small fraction of the processor die area by overloading their floating-point (FP) registers to accommodate vector data. These extensions eliminate the need to modify the operating system by significantly compromising performance. Few extensions use new register files exclusively to FP registers to accelerate the next level of performance driven, high-bandwidth communications and computing applications [2].

AltiVec Technology added to the industry standard PowerPC can be most accurately thought of as a set of registers and SIMD execution units [10]. It offers a simple software-based high-performance computing model to the system engineer. Software based multiprocessing can be used to further increase the capabilities of a system, not to mention providing the ability to revise and refine algorithms in software with no changes required to the underlying hardware platform. This new computational engine operates concurrently with the existing integer and floating-point units, allowing highly parallel operations with simultaneous execution of up to 16 arithmetic operations in a single clock cycle. One can freely intermix Scalar integer, floating-point and vector instructions in the source code without impacting a program's performance. Fig. 2 shows the architectural view of the AltiVec unit [10].

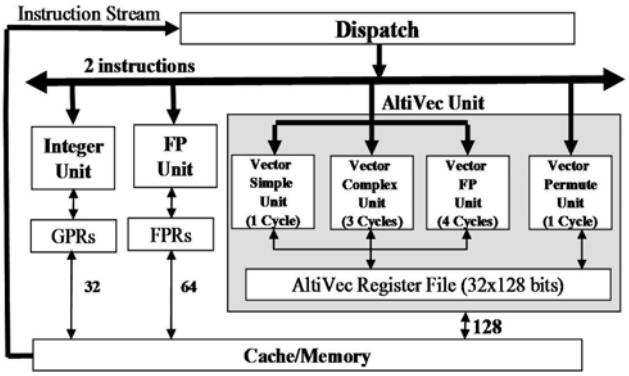

Fig.2. AltiVec architectural diagram with three Vector ALUs and one Permute unit

5. VECTORIZED IMPLEMENTATION

In order for the iterative decoding algorithms to be practically feasible, the complexity in each stage, in terms of number of operations and hardware complexity should be comparable to that of the original non-iterative decoding algorithms. At the same time, the performance should approach the optimum, maximum likelihood decoding performance in terms of bit error rate.

Turbo Product Code decoding algorithm provides ample opportunities for parallel execution and thus performance increases. Vectorization performs computation on multiple elements at a time, i.e. vector processing reduces the fetch and decode bandwidth as the number of instructions fetched are less. Since a block of 256 independent elements is required to be processed with same series of operations by the decoder in minimum possible time, AltiVec unit meets this requirement. The AltiVec version for TPC decoding takes advantage of this data parallelism.

The manner in which the 128-bit wide AltiVec registers are split is important [11]. In TPC decoder, the data is represented as 32-bit, hence 4 samples in a vector are processed simultaneously. In our parallel approach, each row/column of 16 floating point elements is represented by four floating-point vectors that are processed using AltiVec instruction set in the smallest possible time. However, for a 16-bit or 8-bit quantized data, more parallelism can be obtained.

5.1. Programming with AltiVec

Writing AltiVec code for DSP algorithms at assembly language level is very time-consuming in general, but C Programming Model for AltiVec enables facile use of AltiVec within any C, C++ program with less development time [11]. We have enabled the implementation using C intrinsics that has the following characteristics:

- Dynamic Flexibility
- Portability
- Software Reuse
- Improved code readability

The AltiVec C/C++ programming model provides a way to match the parallelism in TPC decoding algorithm to the functional units provided by AltiVec technology.

5.2. Code Optimizations

Our goal is to optimize the TPC decoder for computation time. It is critical to have all code and critical data segments reside on-chip. Since large blocks of data are needed to be stored efficiently in the internal memory for every subsequent iteration, limited on-chip data memory size of 64KB imposes a challenging task on the optimization of highly complex iterative TPC Decoder. The following code optimizations were done during the implementation of iterative soft decoding algorithm for product codes. The code is optimized with respect to execution time and memory usage.

5.2.1. High Throughput Design

High Throughput Designs focus on moving as much data through the calculation as possible in the shortest amount of time [12]. By implementing high-throughput functions of TPC decoder, we have gracefully eliminated the overheads like cost of saving and restoring vector registers, organizing data within register, and loading/assembling constants associated with AltiVec accelerated functions.

5.2.2. Globals and Constants

Globals and Constants are cycle and bandwidth wasters. This is especially true of globals [12]. When we use a global in a tight loop, the compiler can't be sure that some other thread isn't changing that global. As a result, the global is freshly loaded every time through the loop. To avoid this, a stack variable is declared and global is copied into it. Hence in our implementation, loading overhead is reduced by generating constants and globals on the fly.

5.2.3. Data Organization and Alignment

In order to take advantage of high throughput code architecture, all data is kept in one place and accessible as an array [12]. If one is jumping around in memory, especially if one can't even load data as whole vectors, performance will be poor. Since AltiVec Unit does not automatically or transparently handles unaligned Loads and Stores, this problem is addressed by proper data alignment in software although it had substantial effects on the speed.

5.2.4. Uniform vs. Non-uniform Vectors

The functions using non-uniform vectors become more complicated with a lot of permute operations, data shuffling on stack, redundant calculations and lost opportunity for parallelism [12]. We have used uniform vectors that are typically easier to read and write and they rarely require the use of the permute unit at all.

5.2.5. Eliminate Branching

We have avoided the use of conditional jumps as it is difficult for the processor to predict whether to jump or not. Since mispredictions cause undesirable stalling, we have attempted the limited use of conditional jumps as much as possible in the core of the algorithm [12].

Here are few of the optimized Vectorized routines used in TPC decoding algorithm that are frequently called and are the performance bottlenecks in conventional Scalar PowerPC implementation.

- **Hard Decision**

```
        for (i = 0; i<16; i++) {
        vecHardCode[i][0] =
vec_abs(vec_cmplt(vecRecUpCode[i][0], zeroF));
        vecHardCode[i][1]=
vec_abs(vec_cmplt(vecRecUpCode [i][1], zeroF));
        vecHardCode[i][2]=
vec_abs(vec_cmplt(vecRecUpCode [i][2], zeroF));
        vecHardCode[i][3]=
vec_abs(vec_cmplt(vecRecUpCode[i][3], zeroF));
        }
```

// When processor encounters a branch instruction and conditional data is not available, the processor guesses…If guessed wrong, it will back trace the decision point and start over. To reduce computational latency, branching (use of if-then-else) is eliminated by using vec_cmplt () instruction [12].

- **Candidate Codewords Formation**

```
        for (i=0; i<8; i++) {
        vecTestPattern[i][0]=vec_xor(vecTestPattern[i][0],
vecHardCode[row][0]);
        vecTestPattern[i][1]=vec_xor(vecTestPattern[i][1],
vecHardCode[row][1]);
        vecTestPattern[i][2]=vec_xor(vecTestPattern[i][2],
vecHardCode[row][2]);
        vecTestPattern[i][3]=vec_xor(vecTestPattern[i][3],
vecHardCode[row][3]);
```

```
        }
```
// loop unrolling eliminates data dependency, Single vec_xor instruction performs logical exclusive-OR operation between two vectors each consisting of four floating-point elements.

- **Metric Calculation**

```
        for (i=0; i<rSet; i++) {
        Vec1 = vec_madd(vecZdash[i][0],
vecRecUpCode[row][0], zeroF);
        Vec2 = vec_madd(vecZdash[i][1], vecRecUpCode
[row][1], zeroF);
        Vec3 = vec_madd(vecZdash[i][2], vecRecUpCode
[row][2], zeroF);
        Vec4 = vec_madd(vecZdash[i][3], vecRecUpCode
[row][3], zeroF);
        }
```
// Single vec_madd instruction for multiply-accumulate operation, Loop over the length of the vectors, this time doing 4 vectors in parallel to stuff the pipeline. We have unrolled the loop to allow multiple vec_madd to pipeline with one another. This is now possible because the result from one vec_madd is not used in the next one.

- **Search for Concurrent Codeword**

```
typedef union
        {
        vector float vecDetectedCode[4];
        float detectedCode[16];
        } vec_detectedCode;
for (j=0; j<16; j++)
 {
        temp = 0;
        for (i=0; i<rSet; i++)
         {
        if (vDetectedCode.detectedCode[j] !=
        vZdash.zDash[i][j])
                {
                ii[temp] = i;
                temp++;
                }
        }
 }
```
// A union allows member wise access to vector float. It provides 16 byte alignment to non-vector types independent of 8 byte stack frame alignment and prevents unaligned access.

6. RESULTS AND DISCUSSIONS

We have simulated the complete Scalar C code of TPC for gaussian channel in Microsoft Visual C++6.0 IDE. Fig. 3 shows the performance characteristics for $(16,11,4)^2$ TPC, iteration by iteration, in terms of bit error rate (BER) vs. signal to noise ratio (SNR) curves. At BER of 10^{-5}, 6 dB of coding gain is achieved after four iterations as compared to uncoded system. As can be seen from the figure, there is no significant improvement in performance after four iterations.

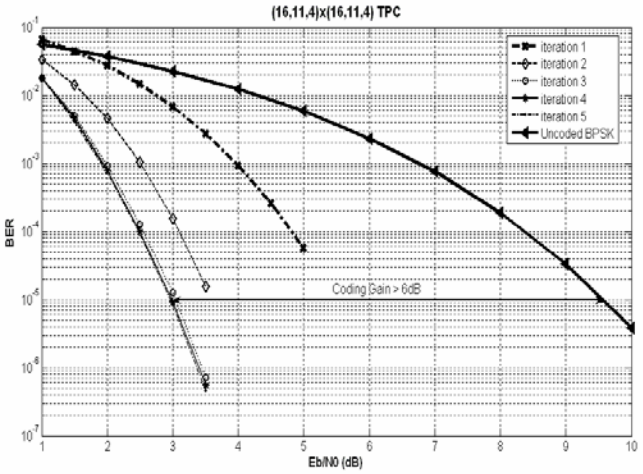

Fig. 3. BER vs. E_b/N_0 of product code [BCH $(16,11,4)^2$] on a Gaussian Channel

The Vectorized C code implementation of decoder using 32-bit floating point representation (without quantization) was compiled using AltiVec enhanced diab compiler (WindRiver) version 5.4.0 and –XO–Xsize-opt optimization options (same as for Scalar code). The resultant object file of size 7Kbytes when downloaded on the target platform only takes 0.48ms of execution time giving a throughput of 252 kbps, while the Scalar PowerPC implementation over the same platform generated 16 Kbytes of object file with throughput of 30 Kbps as shown in Fig. 4. Hence, we have achieved 8x speedup with considerable reduction in object file size as compared to the conventional Scalar version. This is one of the fastest implementation of TPC on a single GPP described to date.

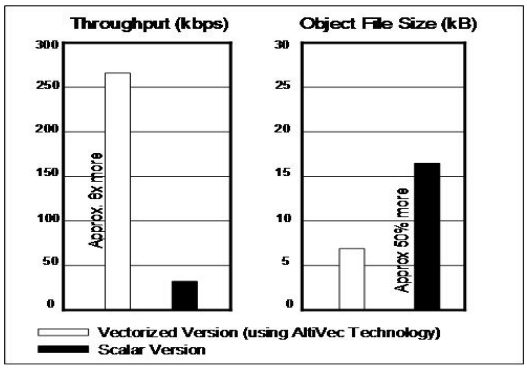

Fig. 4. TPC: Vector vs. Scalar Implementation Results (Both Implementations were carried out on MPC7447A@1.0 GHz)

The main reasons for the speedup are as follows: First, we have efficiently exploited the inherent data-independency of TPC to implement a SIMD-friendly algorithm. Second, the load/store instructions in the iterations are saved as all the variables can be stored in AltiVec registers. Thirdly, the AltiVec version does 4 multiplications and 4 additions in one cycle as opposed to 8 to do the same in Scalar PowerPC implementation. Therefore, the optimized, vector based TPC decoder implementation on AltiVec extension to PowerPC is better suited for real-time applications running on COTS based SDR platforms.

7. CONCLUSIONS

In this paper, we presented the implementation of algorithmically intensive turbo product code on the AltiVec architecture and compared the execution times with those for the Scalar PowerPC processor. It was found that this SIMD architecture significantly increases the speed of execution with appreciable reduction in code footprint. Traditionally, the decoding algorithm was developed by keeping Scalar processor in mind where the goal was to minimize the number of computations. With the SIMD architecture, if proper alignment of data is done to exploit parallelism, significant performance speedup is obtained. However, the hand coding and manual optimization for Vectorization of TPC algorithm is substantially time consuming with significant programming effort and there is still a bit scope of further optimization to achieve more improvement in throughput.

The Turbo Product Code is widely used in SDR systems for high data rate services and its software development using high level language on COTS hardware offers enhanced support of rapid prototyping solutions on SDR platforms with ease of portability and reconfigurability. The technique discussed in Vectorized version can be easily extended to 3D TPC decoder for increase in system performance without upgrading hardware.

8. REFERENCES

[1] Patrick ADDE and Ramesh Pyndiah, "Recent simplifications and improvements in Block Turbo Codes," *2nd International symposium on Turbo codes and related topics*, Brest, France-2000.

[2] Keith diefendorff , Pradeep K. Dubey, Ron Hochsprung, Hunter Scales, "AltiVec Extensions To PowerPC Accelerates Media Processing," *IEEE Micro*, March/April 2000,pp.85-96.

[3] R. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Trans. Commun.*, vol. 46, pp. 1003–1010, Aug. 1998.

[4] Cenk Argon and Steven W. McLaughlin, "An Efficient Chase Decoder for Turbo Product codes," *IEEE Trans. on Communications*, vol. 52, No. 6, June. 2004.

[5] R.Pyndiah, A.Glavieux, A.Picart, and S.Jacq, "Near optimum decoding of product codes," in Proc. *IEEE GLOBECOM'94 Conf.,*vol.1/3, San Francisco, CA, Nov.-Dec. 1994, pp. 339-343.

[6] A. Mahran and M. Benaissa, "Iterative Decoding with a Hamming Threshold for Block Turbo Codes," *IEEE Communication Letters,* Vol.8, No. 9, September 2004.

[7] Patrick ADDE, Ramesh PYNDIAH, Olivier RAOUL and Jean-Rene INISAN , "Block Turbo Decoder Design," *International Symposium on Turbo Codes,* Brest, France-1997.

[8] Ramesh Pyndiah, "Iterative Decoding of Product codes: Block Turbo Codes," *International Symposium on Turbo Codes*, Brest, France-1997.

[9] Andre GOALIC and Ramesh PYNDIAH, "Real-Time Turbo Decoding of Product Codes on a Digital Signal Processor," *International Symposium on Turbo Codes,* Brest, France-1997.

[10] www.freescale.com/files/32bit/factsheet/ALTIVECFACT.pdf

[11] Motorola Inc., "*AltiVec Technology Programming Interface Manual*," 1999.

[12] Ian Ollmann, **"**Practical AltiVec Strategies," 2001.