High-Speed Turbo Equalization for GPP-based Software Defined Radios

Michael Schwall and Friedrich K. Jondral
Karlsruhe Institute of Technology, Germany
Email: {michael.schwall, friedrich.jondral}@kit.edu

***Abstract*—High data rate waveforms for software defined radios (SDR) have to cope with frequency selective fading due to the mobile use in different harsh transmission environments. The received signal needs to be equalized in order to restore the transmitted information. Turbo equalization is a promising approach to deal with the inter-symbol interference occurring at the receiver. The iterative exchange of soft information between the equalizer and the decoder improves the decision reliability and hence reduces the bit error probability compared to conventional receivers. However, the necessary resource-demanding soft-input soft-output algorithms require a high processing performance to ensure real-time capability. In this paper, we will present a high-speed implementation of a turbo equalizer for SDRs with digital signal processing being performed on general purpose processors (GPP). The implementation will utilize linear MMSE filtering and suboptimal algorithms like overlapping sub-trellis MAX-Log-MAP decoding, approximations of mathematical operations, parallelization methods such as threading-based pipelining, and processor specific optimizations like single instruction multiple data (SIMD) commands. We will present the processing gains for each optimization level, highlight the performance loss for the suboptimal modifications and analyze the latency introduced by the pipelined processing. So far, transmissions with data rates up to 5.4 Mbit/s can be decoded in real-time with negligible performance loss and tolerable delay.**

I. Introduction

In the past few years, software defined radios (SDR) have been introduced into various military, public safety and industrial domains. Although the SDRs differ in their architecture, performance characteristics and power consumption, the platforms are mostly equipped with a general purpose processor (GPP) which handles the digital signal processing of the waveforms' physical layer. Besides the implementation of existing narrow-band legacy waveforms, especially the development of innovative wide-band networking waveforms with support for high data rates is fostered currently. Mainly due to the mobile use of SDRs in different environments like mountainous or urban areas, the physical layer of a wide-band waveform has to be designed for frequency selective fading channels. This comprises a powerful forward error correction as well as an efficient channel equalization in the receiver.

Turbo equalization combines the equalization and the decoding in an iterative manner. It has been shown to enable high performance gains, in terms of bit error rates (BER), for the described channel. The approach is based on turbo decoding and was modified for equalization purposes by Douillard *et al.* in 1995 [1] and extended to less complex linear filter-based techniques by Tüchler *et al.* in 2002 [2]. The joint equalization and decoding is performed by a soft-input soft-output (SISO) equalizer and SISO decoder, which are separated by an interleaver and deinterleaver respectively, and carried out for a certain number of iterations. Given a received signal of finite length with inter-symbol interference, each iteration improves the decision reliability and hence reduces the bit error probability compared to conventional receivers. Nevertheless, this iterative procedure and the resource-demanding SISO algorithms put high requirements on digital signal processors to ensure real-time capability.

In this paper, we will implement a soft interference cancellation and linear minimum mean square error (SC-MMSE) based turbo equalizer on a GPP concerning maximum data rate. After introducing the system model in section II, we will present different optimizations in section III, which focus on suboptimal but less complex algorithms or exploit concurrencies within algorithms. Section IV describes the high-speed implementation on an Intel© Core™ i7 GPP and highlights the performance gains.

II. System Model

A. The Received Signal

Based on a single carrier block-transmission, a sequence of N_b independent and uniformly distributed information bits $\mathbf{b}(n) = [b_0(n), b_1(n), \ldots, b_{N_b-1}(n)]^T$ is encoded using a forward error correction. The encoder produces N_c code bits $\mathbf{c}(n)$ using N_p generator polynomials $\mathbf{g} = [\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{N_p-1}]$. The code bits are randomly interleaved and mapped to complex-valued symbols $\mathbf{x}(n)$ using a QPSK modulation scheme with a power of $\mathbb{E}\{|\mathbf{x}(n)|^2\} = 1$. Each transmission block of N_x modulated symbols is prefixed by its last N_{cp} symbols in order to perform frequency domain equalization at the receiver.

We assume a frequency selective block fading channel with an impulse response $\mathbf{h}(n) = [h_0(n), h_1(n), \ldots, h_{N_h-1}]^T$ and additive white Gaussian noise. Hence, after removing the cyclic prefix, the received baseband signal is given by

$$\mathbf{y}(n) = \mathbf{H}(n)\mathbf{x}(n) + \mathbf{n}(n), \tag{1}$$

where $\mathbf{H}(n)$ is the circulant channel matrix of size $N_x \times N_x$ and $\mathbf{n}(n)$ is the additive noise vector whose elements are $\mathcal{CN}(0, N_0)$ distributed. In the following, the elements (l, j) of the $N_x \times N_x$ Fourier matrix $\mathbf{F}$ are given by

$$\mathbf{F}(l,j) = \tfrac{1}{\sqrt{N_x}} e^{-i\frac{2\pi}{N_x}lj}, \quad 0 \le l, j \le N_x - 1,\ i = \sqrt{-1} \tag{2}$$

and the channel transfer function in matrix notation as $\mathbf{\Xi}(n) = \mathrm{diag}(\boldsymbol{\tau}(n))$ where

$$\boldsymbol{\tau}(n) = \sqrt{N_x}\,\mathbf{F}[\mathbf{h}^T(n), \mathbf{0}_{1\times(N_x-N_h)}]. \tag{3}$$

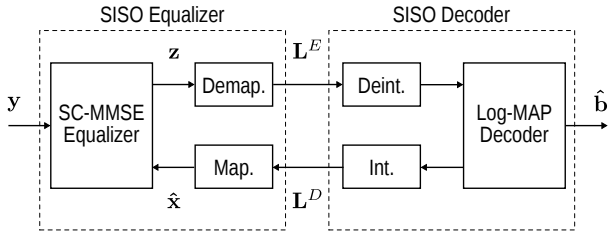

Fig. 1. Block diagram of the SC-MMSE turbo equalizer

B. SC-MMSE Turbo Equalization

In order to recover the transmitted information sequence $\mathbf{b}(n)$, equalization and decoding of the received signal $\mathbf{y}(n)$ are performed iteratively. In theory, turbo equalization applies two maximum *a posteriori* probability (MAP) based SISO components, which exchange reliability information in terms of log-likelihood ratios (LLR) for a given number of iterations N_{it}. Whereas the complexity of a MAP-based decoder is known and only depends on the channel code, the complexity of a MAP-based equalizer is determined by the dynamic and the length of the channel impulse response. Hence, MAP equalizer are not feasible for mobile, power-limited systems and need to be replaced by suboptimal methods. Soft cancellation (SC) minimum mean square error (MMSE) equalization in the frequency domain reduces the complexity using a linear approach and incorporates the available *a priori* information provided by the decoder. In the following, we will outline the basics of the SC-MMSE turbo equalizer based on [3] for a single iteration and drop the block index n for simplicity reasons. The structure of the receiver is depicted in figure 1.

After calculating the Fourier transform of the observation in (1), the SC-MMSE equalizer performs the soft cancellation by subtracting the estimated received signal $\mathbf{\Xi F}\hat{\mathbf{x}}$. Due to the fact that no *a priori* information from the decoder is provided in the 0^{th} iteration, this cancellation is omitted in the initial phase. Given the mean energy $\varphi = \hat{\mathbf{x}}^H\hat{\mathbf{x}}/N_x$ of the symbol estimates and the MMSE equalization matrix $\mathbf{\Psi}$ in the frequency domain

$$\mathbf{\Psi} = \frac{\mathbf{\Xi}^H}{(1-\varphi)\mathbf{\Xi}\mathbf{\Xi}^H + N_0\mathbf{I}}, \tag{4}$$

the equalized signal $\tilde{\mathbf{y}}$, back in the time domain, is calculated using

$$\tilde{\mathbf{y}} = \mathbf{F}^H\mathbf{\Psi}(\mathbf{F}\mathbf{y} - \mathbf{\Xi F}\hat{\mathbf{x}}). \tag{5}$$

Since only extrinsic information is exchanged between the SISO components, the *a priori* information that was incorporated in the equalization so far, has to be suppressed, finally resulting in the linear MMSE filtered output signal

$$\mathbf{z} = (1+\gamma\varphi)^{-1}(\tilde{\mathbf{y}} + \gamma\hat{\mathbf{x}}), \tag{6}$$

where $\gamma = 1/N\,\mathrm{Trace}\{\mathbf{\Psi\Xi}\}$ is the energy adjustment required due to the equalization. In order to map the complex-valued sequence $\mathbf{z} = [z_0, \ldots, z_k, \ldots, z_{N_x-1}]^T$ to LLRs, the probability distribution of a symbol z_k has to be known. According to [2] and a QPSK modulation scheme with symbols $x \in \{1/\sqrt{2}\,(\pm 1 \pm i1)\}$, this distribution can be approximated by

$$P(z_k|x_k = x) \sim \mathcal{CN}\left(\sqrt{2}\,\theta x, \sigma_z^2\right), \tag{7}$$

where $\theta = \gamma/(1+\gamma\varphi)$ and $\sigma_z^2 = \theta(1-\theta)$ are given by the previous calculated energy scaling factors. Finally, the mapping to extrinsic LLRs generated by the equalizer (superscript E) is given by

$$L_{2k}^E = \sqrt{2}\,L_c\,\mathrm{Re}\{z_k\} \tag{8}$$

$$L_{2k+1}^E = \sqrt{2}\,L_c\,\mathrm{Im}\{z_k\}, \quad k = 0, 1, \ldots, N_x - 1 \tag{9}$$

where $L_c = 2\theta/\sigma_z^2$ is the channel reliability. The following deinterleaver ensures independent requirements [2] and restores the natural code bit order.

Based on the *a priori* information $\mathbf{L}^E$ from the equalizer and the applied channel code $\mathbf{g}$, the decoder generates new extrinsic information $\mathbf{L}^D$ (soft decoding) or, if the last iteration is performed, provides an estimate of the transmitted information sequence $\hat{\mathbf{b}}$ (hard decoding). Since the complexity of a MAP-based decoder and hence the processing time is still a problem for a high-speed GPP implementation, it will be replaced by suboptimal overlapping sub-trellis MAX-Log-MAP algorithm which is described in section III-A1.

After interleaving, the LLRs from the decoder are again mapped to complex-valued symbols using

$$\hat{x}_k = \tfrac{1}{\sqrt{2}}\left(\tanh\left(\tfrac{1}{2}L_{2k}^D\right) + i\,\tanh\left(\tfrac{1}{2}L_{2k+1}^D\right)\right), \tag{10}$$

where $k = 0, \ldots, N_x - 1$. The estimated symbols $\hat{\mathbf{x}}$ serve as *a priori* information for the equalizer in (5). The turbo equalizer performs a fixed number of iterations N_{it}, as explained above, to reduce the number of bit errors in the estimated information sequence $\hat{\mathbf{b}}$.

III. Optimizations

In this section, we will describe the optimizations we applied to implement the SC-MMSE turbo equalizer described in section II. As already mentioned, applying a BER-optimal MAP-based equalizer is not feasible due to the complexity of this approach. The length of the channel impulse response and its dynamic lead to exponentially growing complexities not applicable for mobile and power-limited SDRs. The complexity of the linear MMSE approach with frequency domain equalization is primarily independent of the length of the channel impulse response and corresponds to classical equalizer techniques [2]. A slight difference follows from the incorporated *a priori* information that is provided by the decoder. In the following, the optimizations will cover the domains:

- suboptimal algorithms and approximations,
- pipelining, and
- processor specific optimization.

The implementation and the results of each optimization are outlined in section IV.

A. Suboptimal Algorithms and Approximations

1) Overlapping Sub-Trellis MAX-Log-MAP Decoder: The MAX-Log-MAP algorithm [4] performs a suboptimal maximum *a posteriori* probability (MAP) decoding based on the channel code and is derived from the BCJR algorithm [5]. Compared to the original approach, the calculations are carried

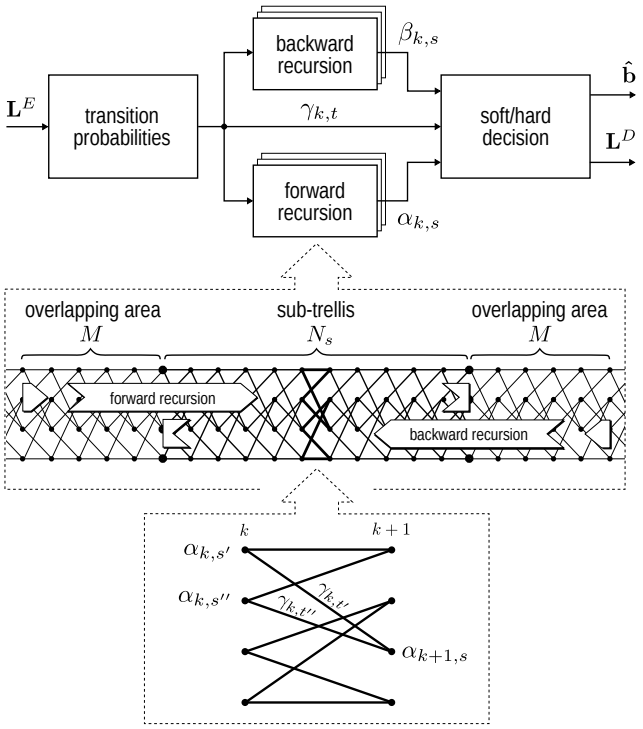

Fig. 2. Overlapping sub-trellis MAX-Log-MAP decoder

out in the logarithmic domain to avoid numerical instabilities. The second modification replaces the complex state metric updates by simple maximum (MAX) operations. Since the algorithm still mainly consists of recursive structures, which do not allow parallelization, the decoding is performed on data-independent sub-trellises. In the following, the overlapping sub-trellis MAX-Log-MAP decoder is described in more detail with respect to parallelization capabilities.

The decoder comprises four units (s. fig. 2) to yield the soft decoded extrinsic LLRs $\mathbf{L}^D$ and the hard decoded information bits $\hat{\mathbf{b}}$ on its output. Based on the input LLRs $\mathbf{L}^E$ of length N_c, which correspond to the reliability of each code bit decision, the logarithmic probabilities $\gamma_{k,t}$ for all valid trellis transitions t are given by

$$\gamma_{k,t} = \sum_{m=0}^{N_p-1} \log\left(\frac{1}{1+e^{(1-2u_{m,t})L^E_{kN_p+m}}}\right), \qquad (11)$$

where $u_{m,t} \in \{0,1\}$ is the code bit of the encoder at position m for a given transition and $k = 0, 1, \ldots, N_x - 1$. Since (11) shows no data dependencies, it can be calculated independently in both time (k) and transition (t) direction, allowing a high order of parallelization. The next step is the recursive computation of the trellis' state metrics. As already mentioned, the metrics $\alpha_{k,s}$ of the *forward* recursion are given by

$$\alpha_{k+1,s} = \mathrm{MAX}(\alpha_{k,s'} + \gamma_{k,t'}, \alpha_{k,s''} + \gamma_{k,t''}), \qquad (12)$$

where t' and t'' are the merging transitions in s, and s' and s'' are the corresponding origins as depicted in the lower part of figure 2. Similarly, the metrics $\beta_{k,s}$ are determined by traversing the trellis in the opposite direction (*backward* recursion). Due to the fact that each recursion requires a

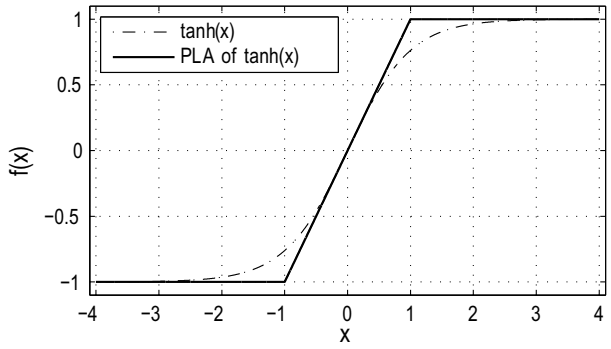

Fig. 3. Piece-wise linear approximation (PLA) of hyperbolic tangent

defined starting point, the encoders' states at the beginning and the end of a transmission frame of length N_c are known to the receiver. Given the metrics and the logarithmic transition probabilities for all transitions and states in the trellis, the soft and hard decision can again be performed independently for all time steps in the trellis.

However, the data-dependent calculation of the state metrics in conjunction with commonly long decoding lengths proves to be the bottleneck of the algorithm. To exploit parallel structures, the entire decoding of the input sequence is divided into L data-independent sub-trellises of equal length $N_s = N_x/L$. The calculation of $\alpha_{k,s}$ and $\beta_{k,s}$ remains unchanged. Only the state metrics at the beginning and at the end of a sub-trellis need to be determined. Thus, the actual computation is expanded by M transitions in both directions overlapping the previous and the following sub-trellis (s. fig. 2), where M corresponds to the Viterbi traceback depth of 5 times the encoders memory length. The performance loss due to the shortened decoding lengths is depicted in figure 5.

2) Approximations of Mathematical Functions: Complex mathematical operations like trigonometric or exponential functions are provided by software libraries and implemented by approximation procedures or lookup tables. Compared to basic operations like multiply or add, the processing complexity and hence the time clearly predominates. However, complex operations can be avoided and replaced by piece-wise linear approximations. The domain of a function definition is subdivided and the function linearly approximated within each subdomain yielding a reduced processing complexity.

Since the turbo equalizer described in II-B applies several complex mathematical operations like the hyperbolic function $\tanh(\cdot)$ in (10), the processing can be accelerated by piece-wise linear approximations. Figure 3 shows an exemplary approximation using three subdomains.

B. Pipelining

Due to the shared memory and multi-core architecture of modern GPPs, applications need to be parallelized in order to exploit the entire processing power. Although parallelization can be applied to a given problem in several ways, the overhead caused has to be considered. Since threads of an algorithm have to be started, managed and finally merged, the parallelized task has to predominate this overhead in order to achieve a speedup.

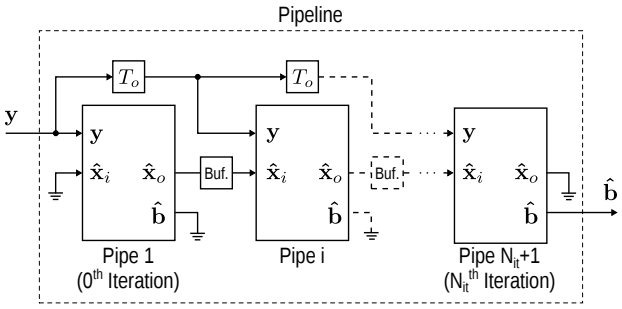

Fig. 4. Block diagram of a pipelined SC-MMSE turbo equalizer

The parallelization of the turbo equalization is performed by using a pipeline design as depicted in figure 4. Each iteration $i = 0, 1, \ldots, N_{it}$ of the turbo process is implemented as a single pipe which takes the observation $\mathbf{y}$ of duration T_f and the estimated symbols $\hat{\mathbf{x}}$ (*a priori* information, index *i*) as inputs. The outputs are again the estimated symbols (index *o*) after one iteration and the estimated transmitted information bits $\hat{\mathbf{b}}$. Due to the special status of the first and the last element in the pipeline, these pipes need to be connected differently. For the first pipe, *a priori* information is not available and has to be set to zero. Furthermore, the last pipe does not need to provide any further signals except the hard decoded information bits. Due to this step-by-step approach, the observation needs to be delayed one sequence duration before passing it to the subsequent pipe.

The advantage of the pipeline design follows from the created independence among the pipes. Assuming a single-core GPP with a CPU core that requires equal amount of time δ_1 for each turbo iteration, real time constraints can only be met when

$$\delta_1 \;\leq\; \frac{T_f}{1 + N_{it}}. \tag{13}$$

If the turbo equalizer is implemented as a pipeline on a multi-core GPP and the pipeline is filled (each buffer keeps data), pipes can be mapped individually to CPU cores and be performed in parallel during one pipeline step. Thus, the required processing time for one turbo iteration can be enhanced to

$$\delta_n \;\leq\; \frac{T_f}{\left\lceil \frac{1+N_{it}}{n} \right\rceil}, \tag{14}$$

where n is the number of CPU cores, $\lceil \cdot \rceil$ the ceiling-operator and any overheard due to scheduling is ignored. On the other hand, if no real time constraints are imposed and the processing time for one iteration on a CPU core is the same on both processors, an estimate for the theoretical speedup s_n on the multi-core GPP is given by

$$s_n \;=\; \frac{1 + N_{it}}{\left\lceil \frac{1+N_{it}}{n} \right\rceil}. \tag{15}$$

The major drawback of the pipeline design is the introduced latency. The delay is at least increased by the time $N_{it}T_f$, since the last pipe decodes an observation that was made N_{it} pipeline steps before. Furthermore, the pipeline has to be *flushed* (reinitialized) first, in order to reset the turbo equalizer. Pipeline related difficulties like data or control hazards need to be considered in the implementation.

C. Processor Specific Optimization

Although processor specific optimization, as the name implies, depend on the given processor, some optimizations are generally applicable to a large group of processors, resp. a *processor family* like *x86*-based GPPs.

Since modern GPPs come with multiple processing units within one CPU core, data level parallelism can be exploited. This single instruction multiple data (SIMD) architecture is intended for multimedia purposes and allows elementary bit or mathematical operations on a group of input values simultaneously. The number of parallel mathematical operations, which is the more relevant case for signal processing, depends on the register length of the processing unit and the word length of an input value. For example, if the values are quantized by a floating point representation of 32 bit, 8 operations can be performed on an 128 bit processing unit in parallel with, compared to a classical unit, a theoretical speedup of 8. Processor specific SIMD solutions like Intel's SSE or AMD's 3DNow! exist and can be used by software interfaces.

Due to the fact that most signal processing within the turbo equalizer is performed on vectors and comprised of elementary mathematical operations like add, multiply, etc., SIMD techniques can be applied to many processes. Computations like (5) need to be split into elementary operations (subtraction, multiply) and mapped to appropriate SIMD commands. However, the independence of the operations among each other has to be ensured in order to avoid data hazards.

IV. Implementation and Results

A. Setup

To evaluate the optimizations presented in section III, the SC-MMSE turbo equalizer is implemented in *C* on an Intel© Core™ i7-2600 CPU (3.40 GHz) GPP with four CPU cores. The GPP is supported by an NVIDIA GeForce 8400 GS graphics processing unit (GPU). We use a Linux-based operating system and the GNU compiler collection (GCC) in version 4.6.3. The length of the observation is set to $N_x = 2560$ complex-valued samples. The data bits are error-protected using an IEEE 802.11 compliant channel code that is designed by the polynomials $[133_8, 171_8]$. The number of turbo iterations is either set to 0 (non-turbo case), 3 or 10. Perfect time and frequency synchronization as well as perfect channel state information are assumed.

B. Programming

Besides standard *C* libraries, the following application programming interfaces (API) are used to implement the optimized version of the SC-MMSE turbo equalizer:

- Pthreads (POSIX threads [6])
  Framework for creating and manipulating threads in order to exploit the parallelism of a GPP.
- VOLK (Vector-Optimized Library of Kernels [7])
  Portable library to target the most suitable SIMD architecture of a GPP.
- OpenCL (Open Computing Language [8])
  Framework to implement applications across heterogeneous platforms consisting of different processing units like GPPs or GPUs.

TABLE I. OPTIMIZATIONS RESULTS

Optimization	0 iterations		3 iterations		10 iterations	
	R	S	R	S	R	S
None	632	1.00	152	1.00	55	1.00
Approximations	631	1.00	154	1.01	56	1.02
Pipelining	606	0.96	552	3.63	215	3.90
SIMD	672	1.06	163	1.07	59	1.07
GPU coprocessor	4963	7.85	735	4.84	205	3.73
All	**5448**	**8.62**	**3482**	**22.91**	**1877**	**34.13**

R=Data rate in kbps; S=Seepdup

- FFTW (Fastest Fourier Transform in the West [9]) Highly optimized software library for computing digital Fourier transforms.

Using Pthreads, the turbo iterations, resp. pipes, are implemented as threads and allocated to the four CPU cores during runtime. Hence, a maximum number of 4 iterations can be performed in parallel on the GPP. Due to the fact that the SISO decoder proves to be the bottleneck of the implementation, it is swapped to the GPU. The GPU acts like a coprocessor and performs the four computational units of the parallelized MAX-Log-MAP algorithm by invoking OpenCL kernels. The number of sub-trellises was set to $L = 4$ resulting in a partial decoding length of $N_s = 640$. During each turbo iteration, two Fourier transforms (frequency domain equalization and soft cancellation) of length 2560 are implemented using FFTW. The SIMD architecture that was detected by VOLK is *avx mmx*. Furthermore, we use the single-precision floating-point format (*float*) and define all system constants by preprocessor directives.

C. Results

Table I summarizes the measured data rates of the different optimizations. The speedup is determined with respect to the processing time of a non-optimized SC-MMSE turbo equalizer. The results show, that the highest processing gains are achieved by exploiting the parallelism of both processors. The scheduling overhead of the threading API amortizes with the number of turbo iterations and achieves a speedup of 3.90 for 10 iterations. The outsourcing of the MAX-Log-MAP decoder to the GPU further boosts the entire implementation by a maximum factor of 7.85. In contrast, the approximations of mathematical functions and the use of the SIMD architecture prove to be impractical and barely improve the data rate. Due to the fact that the presented optimizations are orthogonal and do not interfere, they can be applied at once resulting e.g. in data rates up to 3.4 Mbps for 3 iterations.

As depicted in figure 5, the performance loss due to the suboptimal overlapping sub-trellis MAX-Log-MAP algorithm and the approximations can be neglected for more than 3 iterations. For the non-turbo case, the loss is less than 0.5 dB for 16 sub-trellises and can again be disregarded for the implemented version ($L = 4$). The introduced latency of the pipeline design accounts 2.94 ms and 15 ms for 3 and 10 iterations respectively. In comparison to the requirements for a time critical service like voice over IP, this delay can be ignored.

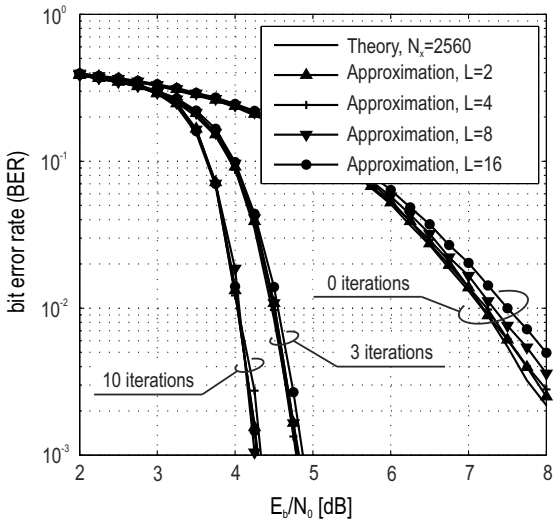

Fig. 5. Performance of the SC-MMSE turbo using approximations and the overlapping sub-trellis MAX-Log-MAP decoder

V. CONCLUSION

In this paper, we presented different optimizations to perform a high-speed SC-MMSE turbo equalization on a GPP-based SDR. The optimizations covered suboptimal algorithms and approximations, pipelining, and processor specific optimization. We used free APIs for the implementation and evaluated the optimizations on an Intel© Core™ i7 GPP with 4 CPU cores. The results showed, that the best performance gains are achieved by exploiting the parallelism of the GPP and using the GPU as coprocessor for the SISO decoding. The BER loss due to the suboptimal algorithms and approximations can be neglected.

REFERENCES

[1] C. Douillard, M. Jezequel, C. Berrou, P. Didier, and A. Picart, "Iterative correction of intersymbol interference: Turbo-equalization," *European Trans. Telecommun.*, vol. 6, no. 5, pp. 507–512, Sep. 1995.

[2] M. Tüchler, R. Koetter, and A. Singer, "Turbo equalization: Principles and new results," *IEEE Trans. Commun.*, vol. 50, no. 5, pp. 754 –767, May 2002.

[3] K. Kansanen and T. Matsumoto, "An analytical method for MMSE MIMO turbo equalizer EXIT chart computation," *IEEE Trans. Wireless Commun.*, vol. 6, no. 1, pp. 59–63, Jan. 2007.

[4] P. Robertson, E. Villebrun, and P. Höher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. Commun.*, Jun. 1995, pp. 1009–1013.

[5] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[6] "IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®) - System Application Program Interface (API) Amendment 2: Threads Extension (C Language)," *IEEE Std. 1003.1c-1995*, 1995.

[7] (2013, Sept.) Vector-optimized library of kernels (VOLK). [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk/

[8] (2013, Sept.) OpenCL - The open standard for parallel programming of heterogeneous systems. [Online]. Available: http://www.khronos.org/opencl/

[9] (2013, Sept.) Fastest Fourier transform in the west (FFTW). [Online]. Available: http://www.fftw.org/