Multi Core Performance of a Block Syndrome Decoder for Convolutional Codes

Klaus Hueske, Jan Geldmacher and Jürgen Götze
Information Processing Lab
TU Dortmund University
Dortmund, Germany
klaus.hueske@tu-dortmund.de

Abstract— **This paper investigates the implementation of forward error correction algorithms on symmetric multiprocessor (SMP) platforms. The focus lies on the Viterbi algorithm (VA) for decoding of convolutional codes. Two aspects will be discussed: How can the VA be implemented on the SMP and how can the decoding complexity be adapted regarding the number of transmission errors? These questions lead to an alternative decoding concept based on syndrome decoding, called block syndrome decoder (BSD). The BSD is compared to alternative parallel implementations of the VA in terms of bit error rate (BER) and achievable speedup on the SMP platform**

Index Terms— **Viterbi Algorithm, Syndrome Decoding, Adaptation, Multicore, SDR.**

I. Introduction

Due to limited battery capacity, the energy efficiency of a mobile radio receiver implementation is a crucial factor. With a focus only on energy efficiency, an application specific integrated circuit (ASIC) based implementation would be the best choice [1]. However, ASICs have several drawbacks: Their development is expensive and time consuming. It is not possible to apply bug fixes or updates once a product is finished. And they provide a low flexibility when dealing with multiple new transmission standards and cognitive radio approaches. All these aspects led to the idea of a software based receiver implementation, the Software Defined Radio (SDR) [2].

To apply the SDR approach to mobile devices, platforms with high energy efficiency are required. Several concepts were proposed, which are mainly based on parallel signal processing, e.g. SIMD (Single Instruction Multiple Data) architectures, vector processors or multi processor architectures. The latter can be divided into symmetric (SMP) and asymmetric (AMP) architectures. While the AMPs, like e.g. TI's OMAP [3], have specialized cores for specific computation tasks, the SMPs are usually composed of general purpose processors. For our investigations we focus on an SMP system based on ARM's MPCore embedded multi core processor [4].

This paper deals with the implementation of a forward error correction (FEC) algorithm on such an SMP platform. The focus on FEC has two reasons: First, FEC is one of the tasks with highest computational complexity in a radio receiver implementation [5] and second, it offers several degrees of freedom for parallel implementations. Common algorithms used in FEC are the Viterbi Algorithm (VA) for decoding convolutional codes and the Maximum-A-Posteriori algorithm (MAP) for Turbo decoding. Both algorithms are based on the trellis representation of the respective encoder. As their structure is related, we will focus on the VA for decoding convolutional codes. The results can be easily extended to MAP decoding.

Two aspects will be considered in this paper: How can the VA be implemented on the SMP in an efficient way and how can the high flexibility of an SDR be exploited to adaptively reduce the computational effort of the algorithms.

Parallel implementations of the VA are well known in VLSI design. Several Viterbi processor designs were proposed, e.g. the canonic cascade Viterbi Decoder [6], the fully parallel bit-serial decoder [7] or a reconfigurable multi processor design [8]. These processors were specifically designed for the VA and are not suitable for general purpose computations. In contrast, with the SMP given we have to adapt the VA to fit onto this particular platform.

An implementation of the VA on an SMP platform requires a sufficiently high granularity, as permanent communication and synchronization between the processors will degrade the speedup. In [5] it was shown

that partitioning the decoding process in state direction, i.e. each core processes different trellis states, yields a significantly lower speedup than partitioning in time direction, i.e. each core processes a different part of the received sequence.

One option to achieve a partitioning in time direction is to modify the encoder and to periodically force it into a certain state, by either inserting zero sequences into the stream of information bits or just resetting the encoder to a certain state (zero-state)[9]. If the intervals are known to the decoder, the received sequence can be separated into blocks accordingly.

However, these approaches lead to a reduction of the code rate or a degradation of the decoding performance, respectively. Another approach for block decoding is to separate the received sequence into overlapping blocks and to decode these blocks independently. This general principle has been called sliding block decoding [10], overlap-add Viterbi algorithm [11] or sliding block Viterbi decoder [12]. It is also applied in the minimized method presented in [13].

Two drawbacks can be identified regarding the overlapping VA: First, an overhead is introduced by decoding overlapping parts of the received sequence. This overhead will limit the achievable speedup. Second, the VA has to process the whole received sequence, even if few or no errors occurred during transmission. The decoding complexity remains constant.

These drawbacks are addressed in an alternative decoder concept based on syndrome decoding (SD) [14]: With SD error-free parts of the received sequence can be identified and the VA is only applied to erroneous parts of the sequence. This significantly reduces the computational demand of the decoder for good transmission conditions, i.e. high SNR. While this adaptive behavior regarding the SNR is difficult to exploit in ASIC implementations, it is perfectly suitable for SDR implementations. Parallel decoding benefits from SD as well, because it allows to partition the received sequence without any overlapping overhead. This property leads to the so-called Block Syndrome Decoder (BSD) [15]. The total achievable speedup S in decoding time of the BSD compared to the conventional Viterbi decoder has two independent origins: The speedup S_{Par} that can be achieved by a parallel implementation and the speedup S_{SNR} that arises from the saved decoding operations due to error-free parts in the received sequence.

The paper is organized as follows: In Section II, the main principles of overlapping Viterbi decoding are reviewed. The Block Syndrome Decoder (BSD) will be presented in Section III. In section IV we will focus on the implementation details and the MPCore processor platform, which is used for performance evaluation in terms of speedup. Final conclusions are drawn in Section V.

II. Viterbi Decoding

In this section, the conventional Viterbi decoder and the overlapping Viterbi decoder are briefly described.

A. Basic Principles

Assume an information sequence $\mathbf{u}$ is encoded with a generator matrix $\mathbf{G}$ and transmitted over a channel. With $\mathbf{e}$ representing the channel error, the received sequence can be expressed as

$$\mathbf{r} = \mathbf{uG} \oplus \mathbf{e} = \mathbf{v} \oplus \mathbf{e}, \tag{1}$$

where $\mathbf{v}$ is the encoded sequence. The decoding problem can then be written in terms of an optimization problem:

$$\min\|\hat{\mathbf{e}}\|, \tag{2}$$

i.e. to find an estimate of the information sequence $\hat{\mathbf{u}}$ with minimum estimation error $\hat{\mathbf{e}}$, where

$$\hat{\mathbf{e}} = \mathbf{r} \oplus \hat{\mathbf{u}}\mathbf{G} = \mathbf{r} \oplus \hat{\mathbf{v}}. \tag{3}$$

In Viterbi decoding [16] the problem (2) may be formulated as

$$\min_{\hat{\mathbf{v}}}\|\mathbf{r} \oplus \hat{\mathbf{v}}\| = \min_{\hat{\mathbf{u}}}\|\mathbf{r} \oplus \hat{\mathbf{u}}\mathbf{G}\|, \tag{4}$$

where $\hat{\mathbf{e}}$ has been substituted by (3). This optimization problem is solved by applying the VA to search the trellis of the encoder $\mathbf{G}$ for the code sequence $\mathbf{v}^* = \mathbf{u}^*\mathbf{G}$ with minimum distance to the received sequence $\mathbf{r}$.

B. Overlapping Viterbi Decoder

If parallel decoding is desired, a possible approach is to separate the received sequence $\mathbf{r}$ into blocks and distribute these block to the parallel processors. However, the encoder introduces continuous dependencies between successive code symbols, i.e. realizing a partitioning is not straight-forward.

A basic idea to tackle this problem and to achieve parallel, high-speed implementations has been presented in [10] with the so called sliding block decoder. The idea is, to "cut" blocks out of the received sequence and determine a decoded symbol for every block. While [10] applies look up tables for decoding,

other approaches employ the VA for decoding the extracted parts (cf. [11], [12], [13]).

However, it is known that the blocks have to overlap to a certain extend to make them independent from each other and to achieve optimal decoding performance in maximum likelihood sense. The required minimum overlapping length is governed by two parts: The acquisition depth and the truncation depth. Fig. 1 illustrates the composition of a block

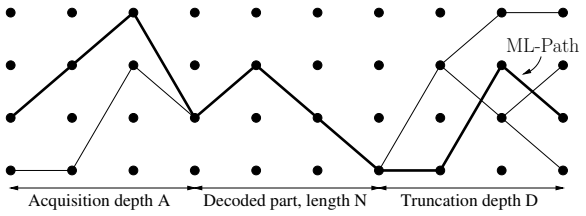

Fig. 1. Composition of a trellis block.

with overlapping parts:

- *Acquisition part.* Because of the unknown initial state and metric, the first part of the block consists of an acquisition part of length A where the state metrics are dependent of the unknown initial metrics. After A steps the metric can be assumed to be independent from the initial metrics. This is likely the case for $A = 5\nu$ [17], where ν is the constraint length of the code.
- *Decoded part.* The middle part of length N is the part where symbol estimates are delivered.
- *Truncation part.* When starting the traceback from the end of the block with unknown final state, the traceback length $D = 5\nu$ is the number of steps after which all paths have most likely merged [18].

The additional minimum number of received symbols required to decode N symbols from an encoded sequence is $A + D = 10\nu$. Depending on the length of the decoded block N this means a considerable computational overhead, which can significantly reduce the achievable speedup. For a frame of length B that is decoded as P overlapping blocks with size $N = B/P$ on P parallel processors the maximum achievable speedup is limited by

$$S_{Par} = \frac{t_1}{t_P} = \frac{B}{\frac{B}{P} + \frac{(P-1)(A+D)}{P}} = \frac{P}{1 + \frac{(P-1)10\nu}{B}}. \tag{5}$$

Here t_1 denotes the execution time on one core and t_P the execution time on P cores. Furthermore, it is assumed that the frames are terminated, i.e. $A = 0$ for the first block and $D = 0$ for the last block. Two drawbacks can be identified regarding the overlapping VA: First, the overhead will limit the achievable speedup. Second, the VA has to process the whole received sequence, even if few or no errors occurred during transmission. It is therefore interesting to investigate how the necessity of overlapping could be avoided without modifying the encoder and how the decoding complexity can be adjusted for different transmission conditions. One approach to address these drawbacks is based on syndrome decoding.

III. Syndrome Decoding

A. Basic Principles

The proposed syndrome decoder is based on Schalkwijk's syndrome decoder presented in [14]. Instead of searching for $\hat{\mathbf{v}}$ directly, the syndrome decoder searches a sequence $\mathbf{e}^*$, which corrects the errors in $\mathbf{r}$. The according constrained optimization problem can be stated as

$$\mathbf{e}^* = \arg\min_{\hat{\mathbf{e}}} \left\{ \|\hat{\mathbf{e}}\| \mid \mathbf{r}\mathbf{H}^T = \hat{\mathbf{e}}\mathbf{H}^T \right\}, \tag{6}$$

where $\mathbf{H}^T$ is the syndrome former matrix and $\mathbf{r}\mathbf{H}^T = \mathbf{b}$ the syndrome of $\mathbf{r}$. The equivalence to (4) can easily be verified by inserting (3) into (6). The syndrome former $\mathbf{H}^T$ is defined to be orthogonal to the generator matrix of the used code and thus to all code sequences. So it holds that

$$\mathbf{b} = \mathbf{r}\mathbf{H}^T = (\mathbf{v} \oplus \mathbf{e})\mathbf{H}^T = \mathbf{u}\mathbf{G}\mathbf{H}^T \oplus \mathbf{e}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$$

and it is obvious that the syndrome only depends on the error sequence.

The optimization problem (6) is solved by applying the VA to search the trellis of the syndrome former $\mathbf{H}^T$ for the error sequence $\mathbf{e}^*$, which satisfies $\mathbf{e}^*\mathbf{H}^T = \mathbf{b}$ and has minimum weight[1]. The estimated error sequence $\mathbf{e}^*$ is then used to correct the transmission errors in $\mathbf{r}$. Finally the estimation of the information sequence is obtained by multiplication with the right inverse of the generator matrix $\mathbf{G}^{-1}$, $\mathbf{u}^* = (\mathbf{r} \oplus \mathbf{e}^*)\mathbf{G}^{-1}$.

We summarize the following important properties of the syndrome decoder:

First, as mentioned above the syndrome only depends on the transmission errors. Error-free parts in the received sequence will lead to zero sequences in

[1]While the original paper [14] only considered hard-decision decoding, the modification of the decoder metric allows also soft-decision decoding [19]. Furthermore, the syndrome decoding approach can be extended to enable MAP decoding.

the syndrome. Thus error-free periods in the received sequence can be identified by detecting parts of consecutive zeros with sufficient length in the syndrome sequence. This allows the detection of error-free periods before the actual decoding happens, i.e. only erroneous parts of the received sequence have to be processed by the VA. This significantly reduces the decoding complexity for good transmission conditions, i.e. high SNR [20].

Second, the additional operations for syndrome computing and the application of the inverse generator matrix are simple XOR-operations and thus of negligible complexity compared to the VA. Consequently, the worst case complexity of the conventional Viterbi decoder and the SD are basically identical.

Third, if we assume that all transmitted information sequences have equal probability, then all code sequences, all paths through the encoder trellis and consequently all trellis states will have equal probability. The SD is based on the idea of estimating error sequences instead of the code sequences and using this estimate to correct the transmission errors. This leads to decoding with unbalanced state probabilities because the state probabilities now depend on the transmission errors and no longer on the information sequence. In error-free parts the decoding path traverses a certain trellis state with high probability. This state with all zero register contents and a leaving edge with input/output bits all zero is referred to as the "zero-state" in the following. In the Block Syndrome Decoding (BSD) approach proposed in [15], error-free parts are identified before decoding and the sequence is separated into bocks with the known initial and final state being the zero-state.

B. Block Syndrome Decoder

The principle of the BSD for parallel decoding is illustrated in Fig. 2. First the syndrome sequence is analyzed and parts with a sufficient number of consecutive zeros are identified ("all-zero parts"). Using this information, the received sequence is separated into blocks with initial and final state being the zero-state. The resulting blocks can then be distributed to the decoders. In this example the number of blocks equals the number of decoders, but one could clearly use an arbitrary (smaller) number of decoders along with a simple scheduling scheme to distribute the resulting blocks to the decoders.

A critical part of the BSD is how to decide from the syndrome sequence at which points a separation of

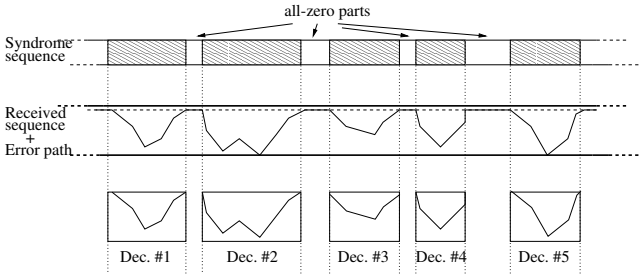

Fig. 2. Overview of the proposed block decoding approach.

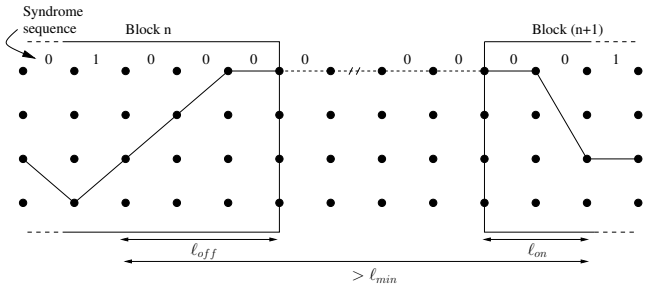

Fig. 3. Design parameters of the block processing scheme.

the received sequence is possible. Therefore, the following design parameters are defined: The parameter ℓ_{min} is defined to be the minimum number of consecutive *0*s in the syndrome sequence required for a separation. That is, whenever a sequence of consecutive *0*s with length greater or equal to ℓ_{min} is detected, the received sequence is separated at this point. The parameters ℓ_{off} and ℓ_{on} define where exactly the previous block ends and where the next block starts: ℓ_{off} denotes the number of *0*s in the syndrome sequence at the end of a block, i.e. the number of stages until it can be assumed with sufficient probability that the ML path has returned to the zero-state. The third parameter ℓ_{on} is defined to be the number of *0*s in the syndrome sequence from the beginning of a block to the first *1*. Fig. 3 illustrates the meaning of these parameters for a four state trellis, where values are chosen as $\ell_{off} = 3$ and $\ell_{on} = 2$.

The selection of the parameter set has an impact on the performance in terms of bit-error-rate (BER) and parallelization speedup: Choosing too small values for ℓ_{on}, ℓ_{off} or ℓ_{min} will result in a degradation of the BER. On the other hand, large values for ℓ_{min} reduce the amount of possible separation points, resulting in longer blocks. Large values for ℓ_{on} and ℓ_{off} reduce the savings, that result from avoiding the decoding of error-free parts between two consecutive blocks. Suitable parameter sets can be determined from simulations results.

Compared to the overlapping schemes described in Section II-B, in the proposed method the block length

is not fixed and cannot be determined before decoding. The block length in fact depends on the channel error, i.e. the transmissions conditions: Good transmission conditions will allow separations more frequently and thus result in shorter block lengths, while bad transmission conditions (more errors) will result in longer blocks.

Hence, in a practical realization of the proposed scheme the block length has to be limited by a suitable upper bound. This can easily be achieved by implementing overlapping as a fallback. This extension of the BSD is refered to as BSD/OL in the following. For the BSD/OL the block length is limited by partitioning blocks, whose lengths exceed a certain upper bound, into overlapping subblocks. To avoid degradation of bit-error-rate in this scheme the overlapping length between consecutive subblocks is set to 5ν, as discussed in Section II-B for the overlapping Viterbi decoder. Hence, the worst-case complexity of the BSD and the conventional Viterbi decoder are basically identical.

As error-free and erroneous sequences can be identified it is reasonable to feed only those parts of the received sequence into the decoders, which are actually erroneous. For the error-free parts no error sequence has to be determined. Compared to the conventional Viterbi Decoder, whose complexity is independent from the number of transmission errors, this will result in an additional decoding speedup S_{SNR}, which however depends on the transmission conditions. A good SNR will result in longer error-free sequences and thus allows higher speedups, while lower SNR will result in shorter error-free sequences and thus less speedup.

IV. Multi Core Implementation and Performance Evaluation

For evaluation the presented approaches were implemented as "C" program and the resulting speedup was determined on an ARM MPCore and, for comparison, on an Intel iCore7 SMP platform.

A. MPCore Architecture

The ARM MPCore architecture is based on 32-bit integer RISC ARM 11 cores with a clock frequency of 200 MHz. The evaluation platform consists of 4 identical cores. Each core can use 32 kB of dedicated level-1 cache for data and 32 kB for instructions. A shared second level cache of 1 MB, which runs at core frequency, allows fast data exchange between the processors. A schematic of the MPCore platform is given in Fig. 4 [4]. A linux based operating system with modified SMP kernel is used for resource management and scheduling. A tool-chain with cross-compiler is used to translate the "C" based source code.

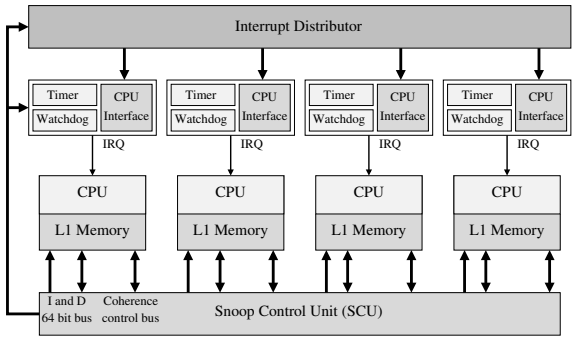

Fig. 4. ARM MPCore Architecture.

B. Implementation

The parallel implementation is based on the *pthreads* library [21]. Multiple decoders are created as P different threads. In case of overlapping Viterbi decoding overlapping parts of the received sequence are assigned to each decoding thread. Each time a thread finishes decoding, a new overlapping part is assigned. After decoding, the inner parts of each decoded block are extracted and reassembled to obtain the final decoded output sequence. The received sequence of length L is divided into P parts and each part is decoded as an independent thread. The traceback is realized within the thread.

In case of the BSD the syndrome analysis has to be performed before starting the actual decoding process. Starting from the beginning of the syndrome sequence, a counter is incremented for each detected zero in the syndrome sequence and reset for each detected one. If ℓ_{min} consecutive zeros are detected, the thread will start decoding the detected block. If the maximum allowed block length is reached without finding ℓ_{min} consecutive zeros, the thread will decode the block using the overlapping method (BSD/OL fallback). In this case a flag is set to prevent the next active thread from starting in zero state. A pointer is used to identify the last decoded element of the received sequence. To avoid indeterministic behavior, the access to the pointers and flags are protected by *mutexes*, i.e. only one thread can access these resources at a given time. It should be noted here, that only the syndrome analysis has to be performed sequentially, while the decoding process itself can be performed concurrently.

C. Performance

The multicore performance of the presented approaches in terms of speedup is evaluated using the implementation described in the previous section. The speedup of a parallel implementation is measured as the quotient of the execution time for one thread t_1 and for P threads t_P:

$$S_{Par}(P) = \frac{t_1}{t_P} \tag{7}$$

To minimize the influence of operating system activity, all measurements were averaged over 2000 decoded blocks. The speedup is given for the described MPCore platform and, for comparison, for Intel's iCore7 processor.

The additional speedup for the BSD compared to the conventional Viterbi decoder depends on the SNR and is defined as the quotient of the execution time t_{Vit} of the conventional Viterbi decoder and the execution time $t_{M\,dB}$ of the BSD at an SNR of MdB on a single core:

$$S_{SNR}(M) = \frac{t_{Vit}}{t_{M\,dB}} \tag{8}$$

The simulation results were generated with the following parameter settings: The convolutional code has a rate of $R = 1/2$ and $\nu = 6$ registers. This corresponds to 64 states. The frame length is set to $B = 1632$ received symbols. These parameters were chosen according to the DVB-T standard [22]. The BSD parameters were set to $\ell_{min} = 18$, $\ell_{on} = 6$ and $\ell_{off} = 6$. The received sequence was partitioned into P blocks, where P is the number of threads.

As stated in Section III-B the parameters ℓ_{min}, ℓ_{on} and ℓ_{off} have to be chosen as a trade-off between speedup and reduction of decoding operation on the one hand and BER performance on the other hand. Hence, the applicability of the chosen parameter set is validated by BER simulations shown in Figure 5.

The results show that the BER performance is basically identical. A small deviation of about $0.1dB$ is visible for BERs $< 10^{-5}$. This, however, is insignificant for most practical systems. The given parameter set can be termed as quite conservative, smaller parameters can further increase the savings in decoding operations at the price of slightly reduced BER performance.

As discussed in Section III the additional computational effort for syndrome calculation and application of the inverse generator matrix can be considered as insignificant compared to the VA operations. This assumption can be verified by the measurement results:

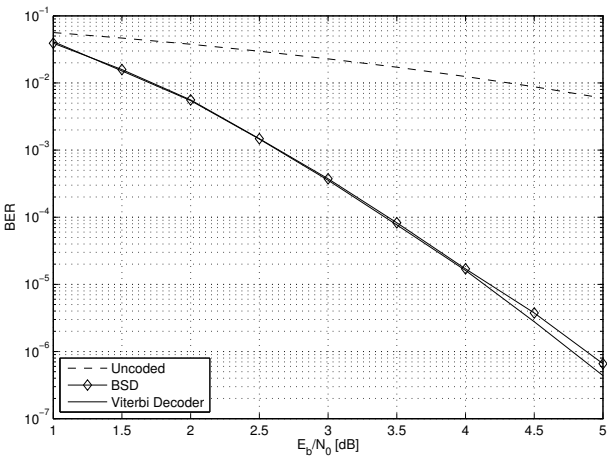

Fig. 5. BER over SNR for conventional Viterbi decoder and BSD.

From the total runtime of the decoder, only 1.48% are consumed for syndrome calculation and another 1.30% for the application of the inverse generator matrix.

In Fig. 6 the parallel speedups S_{Par} for the two decoder concepts running on the ARM MPCore and iCore7 platforms are depicted. Note that only the speedup S_{Par} is considered here, the influence of the additional speedup S_{SNR} is given in Figure 8.

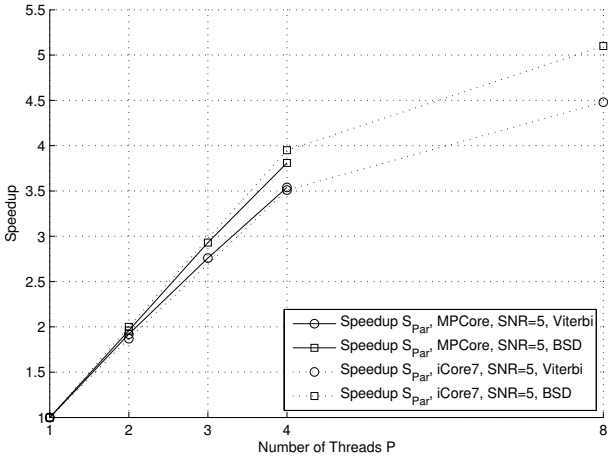

Fig. 6. Speedup by parallel implementation of BSD and Viterbi over P.

With two threads both algorithms perform almost perfectly and reach a speedup close to two. While the BSD scales almost linearly for an increasing number of threads, the gap between overlapping VA and BSD increases. This is caused by the additional overhead due to overlapping. From (5) the maximum speedup for the VA running on 4 threads can be computed as 3.60, which conforms to the measurement results.

The results for 8 threads running on the iCore7 are also given. Although the iCore7 has 8 logical cores,

these cores are based on 4 physical cores using hyperthreading technology. Consequently the increase of speedup is not linear anymore.

The ability of the BSD to separate the received sequence into blocks with known initial and final states strongly depends on the number of errors. This also has an impact on the multicore speedup as shown in Figure 7.

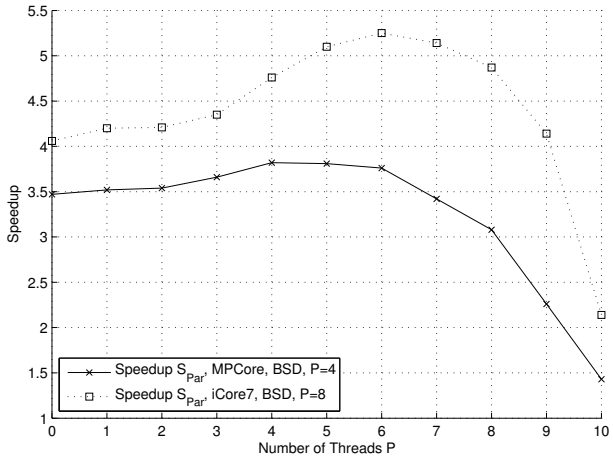

Fig. 7. Speedup by parallel implementation of BSD over SNR.

For low SNR only few and long blocks can be detected, which forces the BSD into the overlapping fallback mode (BSD/OL). In this case the speedup is identical to that of the overlapping Viterbi decoder. For higher SNR more suitable blocks can be found and the speedup approaches the theoretical limit. Interestingly, for very high SNR the speedup by parallel implementation decreases. For very high SNR only few very short blocks have to be decoded, while the bigger part of the received sequence is error-free. In this case more time is spent for finding suitable blocks in the syndrome than for the actually decoding process. This preprocessing, however, has to be performed sequentially, as mentioned in Section IV-B. Hence, the benefit from parallel implementation decreases for very high SNR. This behavior, however, is irrelevant for a practical implementation, because the additional speedup generated by saved decoding operations clearly exceeds the speedup by parallel implementation. This additional speedup is shown in Figure 8 (lower curves).

For increasing SNR the savings in decoding operations result in a significant decoding speedup compared to the conventional VA, whose complexity is constant for all SNR. The savings in decoding complexity are unpredictable and depend on the transmission conditions. Hence, the receiver needs the flexibility to adapt to different conditions in order to exploit this property of the BSD. In a mobile terminal SDR implementation this can result in significant energy savings, when the transmission conditions are good enough.

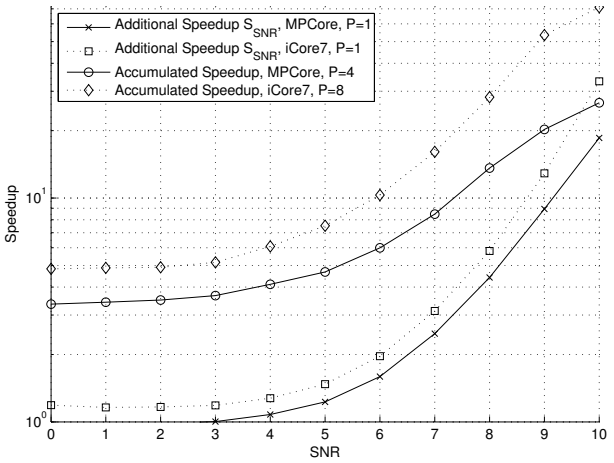

Fig. 8. Additional Speedup and Total Speedup for BSD.

In Figure 8 (upper curves) the total speedup $S = \frac{t_{Vit}}{t_{BSD}} = S_{Par} S_{SNR}$ is shown. As mentioned before, the contribution from parallel implementation decreases for very high SNR. However, the contribution from speedup by reduced decoding operations is significantly increased. Using all 4 cores of the MPCore platform a speedup of almost 30 compared to the conventional VA can be reached for very high SNR.

V. Conclusions

The implementation of decoding algorithms for convolutional codes on symmetric multiprocessor platforms, namely ARM's MPCore and Intel's iCore7, has been investigated. While both algorithms show a significant speedup in a parallel implementation, the block syndrome decoder outperforms the overlapping Viterbi decoder due to its avoidance of overlapping overhead. This leads to an almost linear increase in speedup regarding the number of available cores. The further reduction of decoding operations for received sequences with only few errors lead to a significant accumulated speedup compared to the conventional Viterbi decoder. This adaptive behavior can be particularly exploited in SDR approaches. It should be mentioned here that the speedup by reduction of decoding operations can be further increased by choosing smaller values for ℓ_{min}. This parameter can be reduced if only a specific BER is required as in [23] or a code with shorter constraint length is chosen.

ACKNOWLEDGMENTS

The authors would like to thank the Task9 GmbH in Bochum and Prof. Dr.-Ing. E. Coersmeier, FH Bochum, for providing the ARM MPCore evaluation platform.

REFERENCES

[1] H. Blume, H. T. Feldkaemper, and T. G. Noll. Model-based exploration of the design space for heterogeneous systems on chip. *The Journal of VLSI Signal Processing (Springer)*, Volume 40(1):19–34, May 2005.

[2] J. Mitola. The software radio architecture. *IEEE Communications Magazine*, 33:26 – 38, 1995.

[3] Texas Instruments. *Technical Docmumentation - www.ti.com*.

[4] ARM. *Technical Docmumentation - www.arm.com*.

[5] K. Hueske, J. Geldmacher, J. Götze, and E. Coersmeier. Multi core processing for software radio channel decoder. In *5th Karlsruhe Workshop on Software Radios (WSR 2008)*, Karlsruhe, Germany, March 2008.

[6] Gennady Feygin and Patrick G. Gulak. A multiprocessor architecture for viterbi decoders with linear speedup. *IEEE Transactions on Sginal Processing*, 41:2907–2917, 1993.

[7] Y.-N. Chang, H. Suzuki, and K. K. Parhi. A 2mb/s 256-state 10-mw rate-1/3 viterbi decoder. *IEEE Journal of Solid-State Circuits*, 35:826–834, June 2000.

[8] Guichang Zhong and A.N. Willson. An energy-efficient reconfigurable viterbi decoder on a programmable multiprocessor. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007.

[9] H.-D. Lin and D.G. Messerschmitt. Algorithms and architectures for concurrent viterbi decoding. In *IEEE International Conference on World Prosperity Through Communications (ICC89)*, volume 2, pages 836–840, June 1989.

[10] Kou-Hu Tzou and J. Dunham. Sliding block decoding of convolutional codes. *IEEE Transactions on Communications*, 29(9):1401–1403, Sept. 1981.

[11] P.J. Black and T.H.-Y. Meng. A hardware efficient parallel viterbi algorithm. In *International Conference on Acoustics, Speech, and Signal Processing, 1990. ICASSP-90.*, volume 2, pages 893–896, April 1990.

[12] P.J. Black and T.H.-Y. Meng. A 1-Gb/s, four-state, sliding block viterbi decoder. *IEEE Journal of Solid-State Circuits*, 32(6):797–805, June 1997.

[13] G. Fettweis, H. Dawid, and H. Meyr. Minimized method viterbi decoding: 600 Mbit/s per chip. In *IEEE Global Telecommunications Conference.*, volume 3, pages 1712–1716, Dec. 1990.

[14] J. Schalkwijk and A. Vinck. Syndrome decoding of binary rate-1/2 convolutional codes. *IEEE Transactions on Communications*, 24(9):977–985, Sept. 1976.

[15] J. Geldmacher, K. Hueske, and J. Goetze. Syndrome based block decoding of convolutional codes. In *Proceedings of the IEEE International Symposium on Wireless Communication Systems (ISWCS '08)*, pages 542–546, Reykjavik, Iceland, Oct. 2008.

[16] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[17] G. Fettweis and H. Meyr. Feedforward architectures for parallel viterbi decoding. *The Journal of VLSI Signal Processing*, 3(1-2):105–119, June 1991.

[18] Shu Lin and Daniel J. Costello Jr. *Error Control Coding*. Prentice Hall, 2nd edition, 2004.

[19] M. Tajima, K. Shibata, and Z. Kawasaki. Soft-in syndrome decoding of convolutional codes. *IEICE Trans. Fundamentals*, E85-A(8):1979–1983, 2002.

[20] K. Hueske, J. Geldmacher, and J. Götze. Adaptive decoding of convolutional codes. *Advances in Radio Science*, 5:209214, 2007.

[21] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farell. *Pthreads Programming. A POSIX Standard for Better Multiprocessing*. O'Reilly Media, 1996.

[22] DVB; Framing structure, channel coding and modulation for digital terrestrial television (ETSI EN 300 744), 2004.

[23] J. Geldmacher, K. Hueske, and J. Goetze. An adaptive and complexity reduced decoding algorithm for convolutional codes and its application to digital broadcasting systems. In *IEEE International Conference on Ultra Modern Communications (ICUMT2009)*, 2009.