Multi Core Processing for Software Radio Channel Decoder

Klaus Hueske, Jan Geldmacher, Jürgen Götze
AG DT, ET/IT, TU Dortmund
Email: klaus.hueske@tu-dortmund.de

Edmund Coersmeier
Nokia Research Center, Bochum
Email: edmund.coersmeier@nokia.com

***Abstract*— The growing number of transmission standards and technologies leads to an increased interest in software defined radios (SDRs). To provide the required computing power for these SDRs, multi processor architectures are considered as a realistic approach. To achieve an optimum speedup, the algorithmic descriptions of the digital baseband processing tasks have to be adequate for these architectures. In this work two approaches for parallel Viterbi decoding on a symmetric multi processor platform are presented, which are based on segmentation of the code trellis in state and time direction. The resulting speedup of these approaches is evaluated on an ARM MPCore platform.**

***Index Terms*—Viterbi decoder, symmetric multi processor, MPCore, overlapping, software defined radio**

I. Introduction

THE Viterbi Algorithm (VA) [1] is widely used for decoding Forward Error Correction (FEC) codes in digital transmission systems. Mainly used as standard method for maximum-likelihood decoding of convolutional channel codes, the algorithm is also applicable to block codes or can be used as component decoder for Turbo Codes [2].
Convolutional codes are utilized in nearly all current transmission standards, like e.g. WLAN, 3G or DVB. Digital radio baseband signal processing is mainly based on ASIC (Application Specific Integrated Circuit) implementations, which enable fast and energy efficient mobile devices. However, due to the growing number of transmission standards, ASIC based devices tend to be to inflexible to serve all technologies. Driven by the advantages in processor technology, pure software implementations of digital baseband processing tasks became feasible. Besides the increased flexibility regarding the transmission technologies and standards, the development and maintenance of these Software Defined Radios (SDR) are less time consuming and less expensive compared to hardware solutions [3].

A still challenging problem is to provide the high processing power required for the SDR in an energy efficient way. Increasing the clock rate of the processors will lead to an intolerable power dissipation and energy consumption. Alternative approaches to increase the processing power while keeping the energy consumption low are mainly based on parallel signal processing. Known concepts are 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 [4], have specialized cores for specific computation tasks, the SMPs are usually composed of general purpose processors.
Due to its high computational complexity, the Viterbi decoder causes a significant processor load in an SDR system (see Fig. 1) [5].

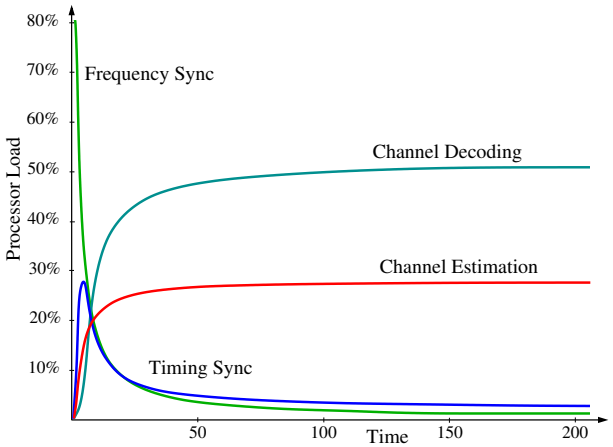

Fig. 1. Processor load of a Digital Radio Mondiale (DRM) receiver on a TI C6713 DSP.

Hence, the Viterbi decoder is often implemented as an additional co-processor (e.g. TI C6416) or hardware accelerator, while other base band processing tasks (e.g. channel estimation) are completely modeled in software.
Parallel implementations of the VA are well known

in VLSI design. Several specialized Viterbi processor designs were presented, e.g. the canonic cascade Viterbi Decoder [6], the fully parallel bit-serial decoder [7] or a reconfigurable multi processor design [8]. However, these processor designs are tailored to the VA. Instead, when implementing the VA on a general purpose SMP, the algorithmic description has to be tailored to the used architecture. Especially the granularity has to be considered, as permanent communication and synchronization between the processors will degrade the speedup.

In previous work an alternative, optimization network based channel decoder concept was presented [9]. Due to the algorithmic structure, no communication or synchronization between the processors is required during the decoding process. This results in an almost linear speedup on an SMP platform. However, for larger constraint lengths of the convolutional code, the decoding performance degraded compared to the maximum likelihood Viterbi decoder. Therefore, concepts for parallel implementations of the VA on an SMP architecture are presented in this work. Two different approaches are considered: The segmentation in state direction and in time direction. In the first approach the trellis state metrics are computed group-wise on the parallel processors in each decoding step, while the latter will compute independent overlapping trellis segments [10], [11]. The speedup of the approaches is evaluated on the ARM MPCore platform using four processors [12].

The paper is organized as follows: In Section II, the main principles of Viterbi decoding will be presented. In Section III, the two parallelization approaches, time direction segmentation and state direction segmentation, will be introduced. In section IV we will focus on the implementation details and the used processor platform, which is used for performance evaluation in terms of speedup. Final conclusions are drawn in Section V.

II. The Viterbi Decoder

The Viterbi decoder [1] is an implementation of the VA and features maximum likelihood decoding with feasible computational complexity and easy processing of reliability informations ("soft-decision decoding").

The Viterbi decoder operates on the trellis of the convolutional encoder. In the following we denote the number of inputs of the encoder by k, the number of outputs by n and the number of encoder memory elements by ν. The *trellis* represents the state transitions of the encoder over time. It is a directed graph with $N = 2^\nu$ nodes ("states") at every time step and 2^k edges ("state transitions") leaving each node, where each edge is labeled with an n-bit output symbol. Fig. 2 shows an example of a trellis for an encoder with $N = 2^2 = 4$ states and code rate $R = k/n = 1/2$. A single trellis segment is depicted on the left and a trellis with five time steps is illustrated on the right hand side. The encoder input at time t is denoted by u_t and the path according to an input sequence 10011 is plotted bold.

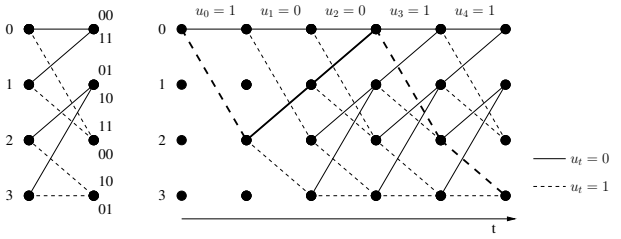

Fig. 2. Example of a trellis segment (left) and a trellis path (right) for a 4-state trellis and five time steps.

As each trellis path represents a valid code sequence, a received sequence can be soft-decision decoded by searching for the path with minimum euclidean distance to the received sequence. For an AWGN-channel this can be shown to be equal to maximum likelihood decoding, i.e. maximizing the probability $P(\hat{\mathbf{v}}|\mathbf{r})$, where $\hat{\mathbf{v}}$ and $\mathbf{r}$ denote decoded and received sequence, respectively. The probability $P(\hat{\mathbf{v}}|\mathbf{r})$ can be calculated as the sum of all individual logarithmic symbol probabilities, which occur on the according path:

$$P(\hat{\mathbf{v}}|\mathbf{r}) = \sum -\log P(\hat{\mathbf{v}}_t|\mathbf{r}_t), \tag{1}$$

where $\hat{\mathbf{v}}_\mathbf{t}$ and $\mathbf{r}_\mathbf{t}$ are decoded and received symbols at time step t, respectively.

In terms of the VA the probability of a trellis path, which ends at time t in a state p, is usually qualified by its *metric* $M_t^{(p)}$. According to Eq. (1) the metric $M_t^{(p)}$ can be expressed as the sum of the *branch metrics* of the path. We denote the branch metric at time t for a transition from state p to state q by $\mu_t^{(p,q)}$. Thus the path metric at time $(t+1)$ can be calculated as

$$M_{t+1}^{(q)} = M_t^{(p)} + \mu_t^{(p,q)}, \tag{2}$$

where we assume a transition from state p at time t into state q at time $(t+1)$. The branch metric for a transition from state p to state q, with an associated BPSK-modulated code symbol $\mathbf{v}_t$ and a received

symbol $\mathbf{r}_t$ can be calculated from the inner product as

$$\mu_t^{(p,q)} =< \mathbf{r}_t, \mathbf{v}_t > .$$

However, normally multiple paths merge into a single

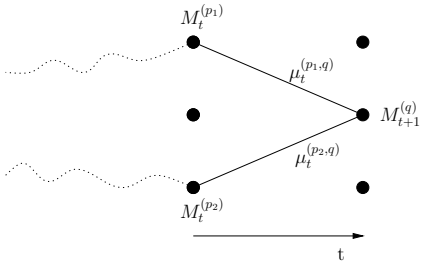

Fig. 3. Configuration of two paths merging into a single state.

state. Fig. 3 illustrates this situation with two paths merging into the state q having the path metrics

$$M_t^{(p_1)} + \mu_t^{(p_1,q)} \text{ and } M_t^{(p_2)} + \mu_t^{(p_2,q)}.$$

The key idea of the VA is that it is only necessary to retain the path with the highest metric, which is called the *survivor*. All other paths can be neglected. Thus for every state and every received symbol the VA has to calculate

$$M_{t+1}^{(q)} = \max_i \left\{ M_t^{(p_i)} + \mu_t^{(p_i,q)} \right\}, \qquad (3)$$

where the preceeding states of q are denoted by p_i. The maximum metric along with the associated survivor-path are stored for each state q at time $(t+1)$. This metric accumulation and the selection of the survivor is known as the Add-Compare-Select (ACS) operation. From Eq. (3) it is easy to see, that the VA has a recursive structure with at first sight limited parallelization options, because the metrics at time step $(t+1)$ are dependent on the metrics at time step t.

The actual decoding operation is realized by tracing back the trellis path from the state with best metric (best-state decoding) or from an arbitrary state (fixed-state decoding) and returning the information symbol at a certain depth. For best-state decoding this decoding delay (depth) is usually chosen as $D = 5\nu$ steps back from the current symbol, which is the number of stages after which the maximum likelihood path has merged with the traceback path with sufficient probability [2].

A decoding delay can be avoided if a coding scheme with initialization and termination is employed. In this case a certain bit pattern is periodically appended to the encoded sequence to force the trellis path into a known state. Thus a periodic traceback from a fixed state can be realized.

The complexity of the Viterbi Decoder is mainly governed by two operations: The ACS operation required for each state is computational intensive. The number of states and with it the number of required ACS operations in turn depend exponentially on ν. The second factor is the traceback operation, which makes heavy use of memory access.

III. Parallelization Approaches

A. State Direction Segmentation

As shown in the previous section, the Viterbi decoding process is time recursive, i.e. all state metrics in a decoding step $t+1$ have to be computed from the state metrics at step t. Thinking about a parallel implementation, a straightforward approach is to distribute the computation of the state metrics M_{t+1} at time step $t+1$ over all available processors, i.e. segment the computation in state direction. With P the number of available processors, N/P states have to be computed on each processor. Due to the recursive structure of the VA, this approach requires synchronization and an exchange of the computed metrics between the processors after each decoding step.

To reduce the synchronization effort, each processor can compute b decoding steps before sharing the results with the other processors. This is realized by exploiting the dependencies in the trellis graph. The resulting decoding scheme for two processors and $N = 16$ states is given in Fig. 4. In the first step the metrics $M_{t+1}^{(q)}$ for $q = 0, 1, 2, 3, 8, 9, 10, 11$ are computed from the preceding metrics $M_t^{(q)}$ with $q = 0, ..., 7$. It can be seen from the trellis structure that the new computed $M_{t+1}^{(q)}$ enable the computation of $M_{t+2}^{(q)}$ with

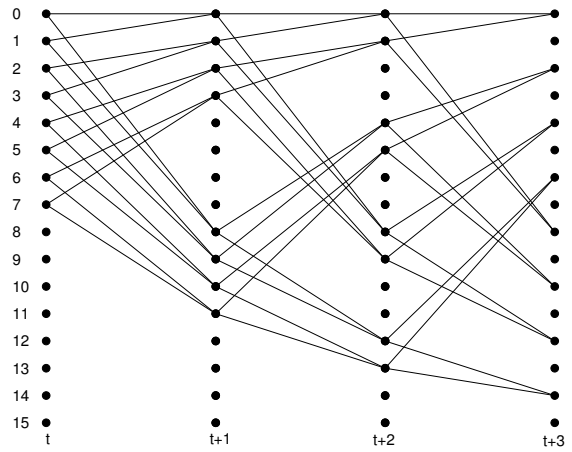

Fig. 4. Parallel decoding using two processors for a trellis with N=16 states. Workload of the first processor.

$q = 0, 1, 4, 5, 8, 9, 12, 13$. In the third step, the metrics $M_{t+3}^{(q)}$ with $q = 0, 2, 4, 6, 8, 10, 12, 14$ can be obtained from the previous computed states. After this step no more couples of preceding states are available for further computations.

The example shows that the number of decoding steps that can be performed independently on the parallel processors is limited. The maximum number of decoding steps depends on the number of states, i.e. the length of the encoder memory, and the number of used processors. The limit is given as

$$b \leq \nu - \log_2 P. \tag{4}$$

Furthermore, to achieve a regular segmentation of the trellis states, b should be a power of 2.

B. Time Direction Segmentation

The state direction segmentation approach allows an independent computation of the state metrics for b decoding steps. However, in a block with l data bits, still l/b synchronization points have to be inserted. The resulting synchronization overhead can significantly reduce the achievable speedup [13].

To reduce the synchronization effort, an algorithmic description with increased granularity is desirable. Here we suggest a segmentation of the code trellis in time direction, i.e. a data block of length l is divided into P segments and all segments are concurrently decoded on the available processors. With this approach only one synchronization point per data block is required, i.e. the synchronization and communication overhead is minimized.

However, the segmentation of a data block into several parts will lead to an additional decoding error, which is caused by the missing initialization and termination of the segments (see Section II). Without this information, the initial and final state of the path are unknown. As mentioned before, after several decoding steps all possible paths will merge into one unique path, i.e. the additional error resulting from the missing termination/initialization is significant only in the beginning and end of each segment. To eliminate the additional decoding error, the block is divided into overlapping segments, where the beginning and end of each segment are discarded after decoding and only the middle, unique part is kept for further processing [10].

Corresponding to the decoding delay during traceback, all path have most likely merged after $D = 5\nu$ decoding steps [11]. The relations are clarified in Fig. 5. The use of overlapping segments will introduce

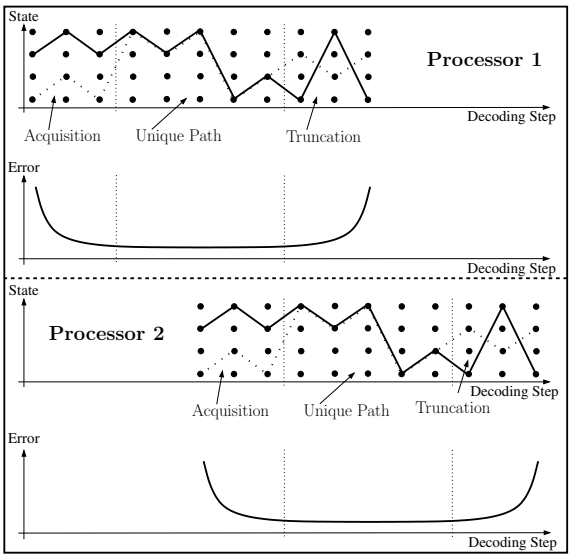

Fig. 5. Channel decoding using overlapping decoders.

computational overhead, as parts of the decoded segment are discarded. With a truncation and acquisition length of $D = 5\nu$ each, the number of required decoding steps is increased by a factor

$$F = \frac{P(\frac{l}{P} + 2D)}{l} = 1 + 10\nu\frac{P}{l}. \tag{5}$$

IV. Implementation and Performance Evaluation

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

A. MPCore architecture

The ARM MPCore architecture is based on 32-bit integer RISC ARM 11 cores. The evaluation platform provides 4 cores, which are clocked at 200 MHz. 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. 6 [12].

A linux based operating system with modified SMP kernel is used for resource management and scheduling.

B. Implementation

To enable parallel execution on the four available processors, the parallel parts of the algorithms are modeled as threads using the *pthreads* library

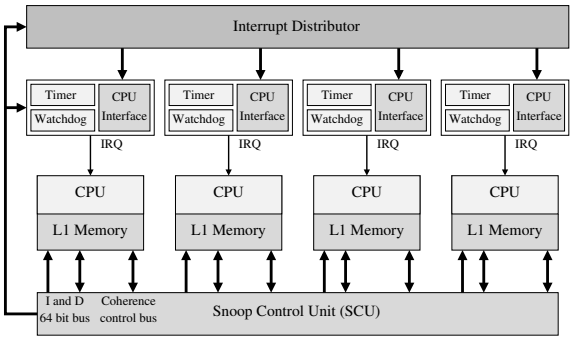

Fig. 6. ARM MPCore Architecture.

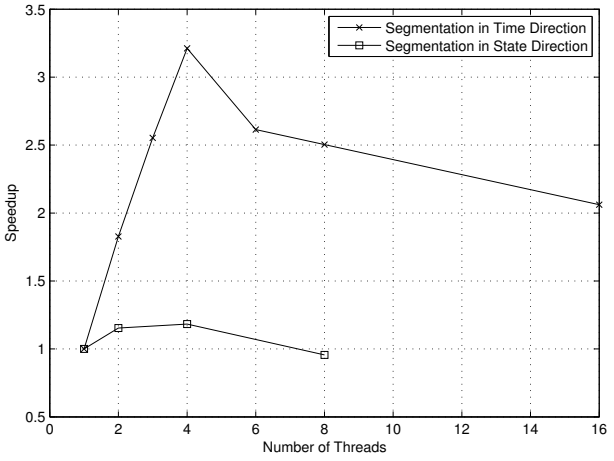

Fig. 7. Speedup for the segmentation approaches using 1 to 16 threads.

[14]. The management of the generated threads and their assignment to the processors is organized by the scheduler.

In case of state direction segmentation, a synchronization point has to be inserted after b decoding steps, which is modeled using the functions *pthread_cond_wait* and *pthread_cond_signal*. These functions allow to suspend the already finished threads until the last thread completes its computations. The traceback routine is started after all states of the data block have been computed.

For time direction segmentation only one synchronization point is necessary. The incoming data block is divided into P parts and each part is decoded as an independent thread. The traceback is part of each thread. After parallel decoding of all segments, the inner parts of each decoded block are extracted and reassembled to obtain the final decoded output sequence.

C. Performance

The performance of the presented approaches in terms of speedup is evaluated for the following parameter set:

- Code Rate $R = 1/2$
- Block length $l = 2500$
- Number of states $N = 2^9 = 512$
- Steps before synchronization $b = 4$

The resulting speedup for the two decoder approaches running on the ARM MPCore platform is depicted in Fig. 7. For time direction segmentation we can find a speedup with nearly linear shape if the number of threads is $\leq$ 4. Increasing the number of threads will reduce the achievable speedup, which can be explained by two effects: On the one hand the management overhead grows with increasing number of threads while the available number of processors is limited to 4. This confirms the common assumption that the number of threads should correspond to the number of processors to achieve maximum performance [13]. On the other hand, the computational overhead grows when assigning more threads, due to the use of overlapping segments (see Eq. 5). With the used parameter set and a number of 16 threads the overhead factor yields $F = 1.576$. The resulting speedup for 16 threads is about 2.

For state direction segmentation the speedup is close to one for a number of threads ≤ 4 or even below for a larger number of threads. This can be explained by the high number of required synchronization points. The forced synchronization results in idle times of the processors, if the computations finish at different times. Furthermore, threads have to be suspended and started again, which results in a huge operating system overhead. The required data exchange will further reduce the performance, as access to the L2 cache is generally slower than access to the L1 cache. With growing number of threads the number of computations performed by each thread is reduced while the amount of management overhead increases. This results in a speedup less than one.

Comparing the two approaches, we see that time direction segmentation scales quite well, while the speedup of state direction segmentation is limited by the high amount of synchronization points. On the other hand the memory requirements of the overlapping based approach are significantly higher compared to the state direction segmentation, as one data block is decoded by P complete Viterbi decoders, which requires P times more memory for saving the state metrics and trellis paths.

V. Conclusions

In this work we described two approaches for mapping the Viterbi algorithm on a symmetric multi processor platform. The state direction segmentation approach is based on computing groups of states on the parallel processors, while the time direction segmentation approach decodes overlapping parts of the data block. The achievable speedup was evaluated on the ARM MPCore SMP architecture. While the time direction segmentation approach offers a good scalability, the speedup of the state direction segmentation approach is limited by the large synchronization and communication effort.

References

[1] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

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

[3] J. Mitola, "The software radio architecture," *IEEE Communications Magazine*, vol. 33, pp. 26 – 38, 1995.

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

[5] A. Kurpiers and V. Fischer, "Open-source implementation of a digital radio mondiale (drm) receiver," in *9th International IEE Conference on HF Radio Systems and Techniques*, Bath, UK, 2003.

[6] Gennady Feygin and Patrick G. Gulak, "A multiprocessor architecture for viterbi decoders with linear speedup," *IEEE Transactions on Sginal Processing*, vol. 41, pp. 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*, vol. 35, pp. 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] K. Hueske, J. Götze, and E. Coersmeier, "Improving the performance of a recurrent neural network convolutional decoder," in *7th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2007)*, Cairo, Egypt, December 2007.

[10] K. Hueske, C. V. Sinn, and J. Götze, "Parallel block signal processing in high speed wireless communication systems," in *Proc. IEEE Int. Symp. on Wireless Communication Systems*, Trondheim, Normway, October 2007.

[11] G. Fettweis and H. Meyr, "Feedforward architectures for parallel viterbi decoding," *The Journal of VLSI Signal Processing*, vol. 3, no. 1-2, pp. 105–119, June 1991.

[12] ARM, *Technical Docmumentation - www.arm.com*.

[13] M.O. Tokhi, M. A. Hossain, and M. H. Shaheed, *Parallel Computing for Real-time Signal Processing and Control*, Springer Verlag, 2003.

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