

# Parallelization of Radio Algorithms for Multi-Processor Platforms

Edmund Coersmeier, Marc Hoffmann

Nokia Research Center Bochum, Germany, firstname.lastname@nokia.com

Klaus Hueske, Jürgen Götze

Information Processing Lab, University of Dortmund, Germany, firstname.lastname@udo.edu

Felix Leder, Peter Martini

Institute of Computer Science IV, University of Bonn, {peter.martini}{leder}@cs.uni-bonn.de

**Abstract**—This paper analyzes software radio processor load and algorithm speed for different OFDM physical layer tasks. Starting with a single core DSP implementation, the most time consuming tasks are evaluated. After that, the matrix inversion for Wiener filter channel estimation is used to evaluate the algorithm behaviour on a multi-processor platform. We will show that parallel algorithms used for hardware realizations (e.g. systolic arrays) are not tailored to multi-processor architectures, due to their fine granularity. This leads to the conclusion that different algorithmic descriptions for the physical layer tasks have to be developed to provide an optimum fit to multi-processor architectures. Exemplarily an alternative optimization network based channel decoding algorithm is presented.

**Index Terms**—software radio, channel estimation, channel coding, algorithm parallelization, multi-processor platform

## I. INTRODUCTION

Multi-processor platforms seem to be a realistic approach to realize physical layer software radio implementations. Theoretically, the processing power of the multi-core platform increases proportionally to the number of available cores. But in reality the performance gain is typically lower, due to communication and synchronization overhead. Especially for algorithms with a strongly sequential structure the benefits of multi-core architectures are close to zero (or even below), which leads to the requirement that the used algorithms have to be suitable for the used architecture.

A simple idea is to map different consecutive algorithms like e.g. FIR filters, FFT, channel estimation, interleaving or channel decoding onto different processors, such that each processing task allocates one processor. This would require a relatively large number of (heterogeneous) processors and the processor load depends on the task that was assigned to it.

A more regular approach is that different algorithms are scheduled optimally onto a limited number of processors. This task can be realized by a smart scheduler, which is not the topic of this paper. However, for a good processor load balancing it is important that each algorithm can be distributed onto more than one processor or thread at a time.

Before analyzing the behaviour of different algorithms on a multi-core architecture, the most time consuming tasks should be determined and chosen for further considerations. Figure 1 shows the processor load for an OFDM-based Digital Radio Mondiale receiver, which is running on a single floating point DSP [1]-[5].

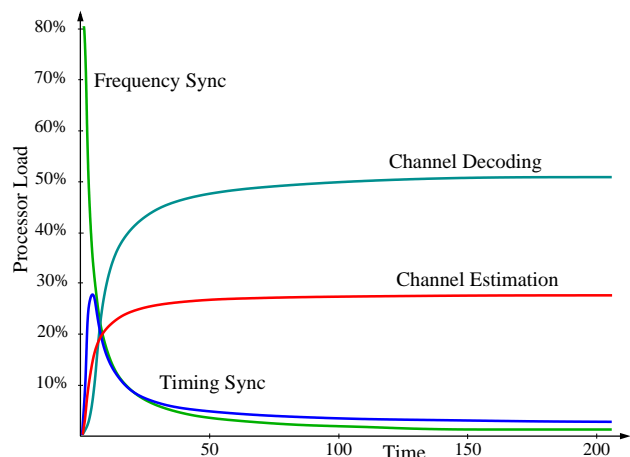


Fig. 1. Processor load for a DRM Radio.

The processor load is mainly caused by two processing tasks, the channel estimation and channel decoding. Synchronization in time and frequency direction are only significant during the beginning of the reception process. Thus, the first two receiver tasks will be investigated in more detail in this paper.

## II. CHANNEL ESTIMATION

To restore the distorted transmitted data, the receiver at first has to estimate the channel character-

istics, which can be described by a discrete channel transfer function  $\mathbf{H}$  [6] [7]. For an OFDM system the received signal in frequency domain can be described as

$$\mathbf{Y} = \mathbf{H}\mathbf{X} + \mathbf{N}, \quad (1)$$

where  $\mathbf{X}$  is a vector containing the transmitted signal and  $\mathbf{N}$  is a vector of uncorrelated identically distributed complex zero-mean Gaussian noise with variance  $\sigma$ . The estimation of  $\mathbf{H}$  is usually supported by so-called pilot symbols. These are training symbols that are known in the receiver and give an estimate of the channel transfer function at the pilot carrier locations. The discrete channel transfer function  $\mathbf{H}$  can then be calculated by interpolation between the pilot positions, which can be performed by the Wiener filter [8]. The MMSE (Minimum Mean Square Error) estimate using Wiener filtering is given as

$$\hat{\mathbf{H}}_{MMSE} = \mathbf{R}_{HH_p} \left( \mathbf{R}_{H_p H_p} + \frac{\beta}{\text{SNR}} \mathbf{I} \right)^{-1} \mathbf{H}_p \quad (2)$$

Table I gives a short description of the required parameters used by the Wiener filter.

$\mathbf{R}_{HH_p}$	cross correlation matrix between pilots and data carriers
$\mathbf{R}_{H_p H_p}$	auto correlation matrix of the channel at pilot positions
$\beta$	constant depending on the chosen modulation scheme
SNR	signal to noise ratio
$\hat{\mathbf{H}}_p$	noisy estimate of the transfer function at pilot positions

TABLE I  
PARAMETERS FOR WIENER FILTER CALCULATION.

In present-day receiver implementations the Wiener filter coefficients are typically precomputed and stored for different channel conditions. However, it is beneficial for the quality of the channel estimation to perform an on-line computation of the coefficients [9] [10]. The increased estimation quality will of course come along with an increased mathematical complexity, due to the matrix inversion that has to be performed for the coefficient calculation.

Wiener coefficient calculation, including matrix inversion, is one of the dominant tasks during the overall channel estimation process. In Figure 2, which shows an example how the processor load is distributed for different Wiener filter tasks, one can see that 49% of the processor load is caused by the coefficient calculations. In the next section different

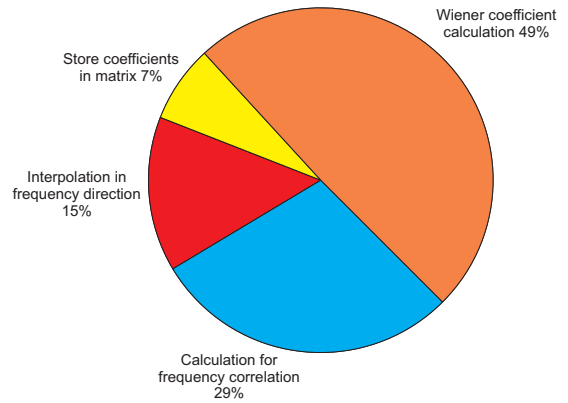


Fig. 2. Processor load for Wiener filtering in frequency direction.

matrix inversion algorithms for a software radio performing on-line Wiener filter coefficient calculations are investigated related to their suitability for multi-processor platforms.

### III. MULTI-PROCESSOR MATRIX INVERSION

There are several algorithms available to compute the inverse of a matrix. Well-known ones are Cholesky, Levinson Durbin Trench, QR as well as Schur Trench [11]-[13]. The key idea of this paper is to investigate the performance of these algorithms in a multi-core environment. The target is to compare the execution speed of each matrix inversion algorithm, running on the one hand on a single processor and on the other hand on several processors in parallel. For this task, an ARM four-processor platform has been used.

The ARM processors are based on a 32-bit integer RISC architecture. The used ARM 11 cores are clocked with 200 MHz and can use 32 kB of dedicated level-1-cache for data and instructions respectively. A second level cache with a size of 1 MB, which runs at the core frequency, is shared by the four processors and allows fast data exchange between the cores.

Because the focus was on evaluation of relative speed-up, the used floating point C++ based matrix inversions were not converted to fixed point implementations, even if no floating-point co-processor has been used. The floating-point operations are therefore emulated on the ARM architecture.

The input matrix is identical for all evaluated inversion algorithms. Based on the problem to be solved, the matrices have Toeplitz structure [14]. Different matrix sizes are evaluated in order to analyze scalability and performance improvements. Figure 3 shows the number of operations per algorithm as function of the matrix size. As we used a

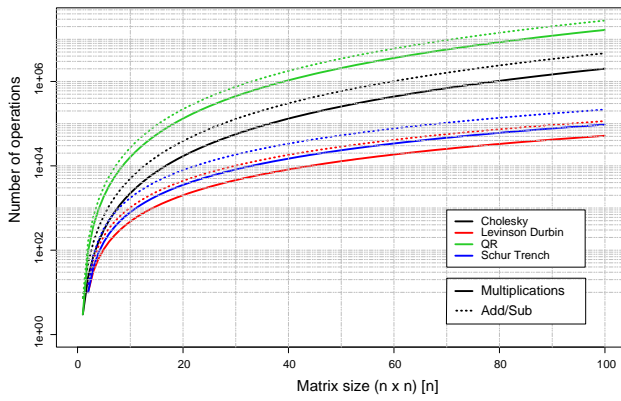


Fig. 3. Number of operations for different matrix sizes.

Toeplitz matrix, the Levinson Durbin Trench is the most efficient algorithm in terms of mathematical operations, followed by Schur, Cholesky and QR. However, the operation count is not necessarily a measure for the processing time of the algorithm. The execution time of the algorithms is shown in Figure 4. Each test was repeated 20 times to draw statistically significant conclusions.

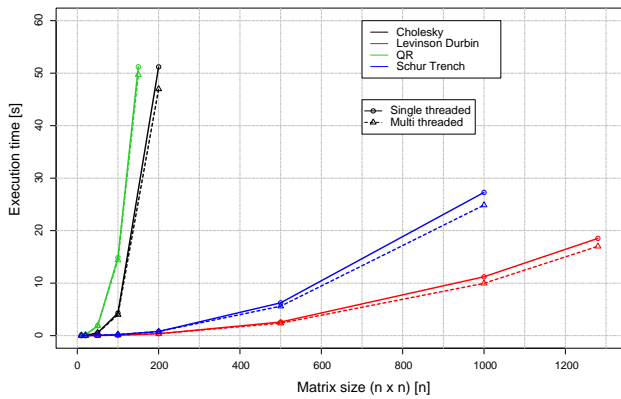


Fig. 4. Execution time of different matrix inversion algorithms.

Besides the single-threaded or sequential execution times, Figure 4 also shows the execution times of the multi-threaded algorithms. The algorithms are split into parts for parallel execution. Each part is executed in a separate thread, which is assigned to one of the four processors. The partitioning degree of the algorithms is matched to the number of processors for Schur-Trench, QR, and Cholesky. The Levinson-Durbin-Trench inversion only allows the use of two threads in some parts because of strong data interdependencies.

Figure 4 shows that a significant speed-up can only be determined for large matrices. The overhead caused by the multi-processor architecture will countervail the advantages of the increased processing power especially for small matrices. Hence, for signal processing algorithms, which often work

with small matrices, parallel execution is not always beneficial! This is clarified in Figure 5, which shows the relative speed-up of the parallel matrix inversions compared to sequential execution. Ideal parallel implementations on the considered ARM platform would reach speedups of close to +400% with four processing units. The results in Figure 5 show speedups from -360% to +10.7%.

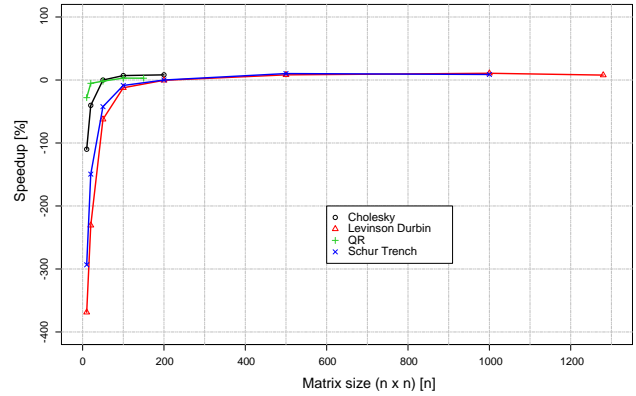


Fig. 5. Relative speed-up of parallel matrix inversion.

There are two interesting aspects about this result: **1)** The first aspect is how such a significant performance loss of 360% is possible to occur. The matrix inversion of a small sized matrix can be computed sequentially on a single processor in a very short time. For the execution on parallel processing units however, threads have to be started, assigned to the processors, and synchronized after their computations. Even if the time required for starting and stopping is relatively short, the contribution to the total processing time is enormous when considering small matrices. In this case the inversion will be already completed on a single processor before the threads are assigned to the multi-processors. The faster the inversion, the larger the relative management overhead. The computation time for matrices of size  $200 \times 200$  using Levinson Durbin Trench and Schur Trench is nearly identical (speed-up is zero) for both platforms (compare Figure 5). For smaller matrices we see a significant performance loss. The same applies to QR and Cholesky for matrix sizes below  $100 \times 100$ .

**2)** The second interesting aspect is the maximum performance improvement of less than 11% even for inversions that run 10, 20, and even more seconds. For those, the management overhead is relatively low. There are two reasons for this phenomenon, both related to the algorithmic structure of the inversions. The data interdependencies in the Levinson Durbin Trench algorithm are very strong and there-

fore, there are hardly any possibilities for parallel execution. QR, Cholesky and Schur Trench allow a better parallelization, however, their granularity is too fine for an efficient implementation on a four processor architecture. While the inter processor communication can be efficiently realized for hardware implementations based on systolic arrays, this communication overhead will significantly slow down the computations in the used configuration. The procedure of synchronization and communication results in idle processing units and limits the relative speedup to 10.7%, even for large matrices. Thus this multi-core investigation shows, that traditional mathematics are not generally applicable when switching from pure hardware implementations to parallel processor software radio architectures. Therefore, alternative mathematical descriptions for parallel processing of the physical layer tasks have to be investigated. As an example an alternative channel decoding algorithm is presented in the next section.

#### IV. PARALLELIZATION OF CHANNEL DECODING

Decoding of convolutional codes is normally realized using the Viterbi algorithm [15] [16]. The structure of this algorithm allows efficient and fast hardware implementations. But the fine granularity makes implementations on multi-processor platforms difficult. This is the reason for introducing an alternative approach. It is not the goal to provide a channel decoding algorithm which reaches the same performance as the Viterbi decoder, but to find an algorithm, which can be split into independent threads without much synchronization overhead from scheduler perspective.

The proposed method applies an optimization network, which can be described as Recurrent Neural Network (RNN) [17] [18]. The advantage of the RNN is the flexible adaptation of algorithm parameters to adjust processing speed, algorithm performance as well as parallelization on high abstraction level. Thus, the RNN channel decoder is a good candidate to fit on a multi-processor Software Radio platform.

While the encoding process can be described as a multiplication of the information vector  $\mathbf{a}$  with the generator matrix  $\mathbf{G}$ , following

$$\mathbf{c} = \mathbf{G}^T \mathbf{a}, \quad (3)$$

the decoding problem in the receiver can be described by the following least-squares expression

$$\min_{\hat{\mathbf{c}}} \|\mathbf{r} - \hat{\mathbf{c}}\|_2^2 = \min_{\hat{\mathbf{a}}} \|\mathbf{r} - \mathbf{G}^T \hat{\mathbf{a}}\|_2^2. \quad (4)$$

This means the decoder has to determine a valid code word  $\hat{\mathbf{c}}$  with minimum euclidean distance to the received vector  $\mathbf{r}$ . When using an antipodal transmission scheme, the alphabet  $\{0,1\}$  can be transferred to the alphabet  $\{-1,1\}$ , which leads to the modified encoding rule

$$c_k^{(q)} = - \prod_{p=1}^K (-a_{k+1-p})^{g_p^{(q)}}, \quad (5)$$

with  $c_k^{(q)} \in \{-1,1\}$  the  $k$ -th bit of the output sequence of the  $q$ -th encoder output and  $K$  the constraint length of the code. The exponent  $g_p^{(q)}$  is the  $p$ -th component of the impulse response of the  $q$ -th encoder output. The decoding problem can then be written as

$$\min_{\hat{\mathbf{a}}} E(\hat{\mathbf{a}}) = \sum_{k=0}^{l-1} \sum_{q=1}^n \left[ r_k^{(q)} + \prod_{p=1}^K (-\hat{a}_{k+1-p})^{g_p^{(q)}} \right]^2, \quad (6)$$

with  $l$  the length of the data vector  $\mathbf{a}$ . As the function  $E(\hat{\mathbf{a}})$  describes an error surface, whose shape is defined by the received vector  $\mathbf{r}$ , we can use the iterative gradient descent approach for minimization of  $E(\hat{\mathbf{a}})$  regarding to  $\hat{\mathbf{a}}$  [7] [17].

To reduce the number of required iterations and to reduce the risk of sub optimum solutions caused by local minima, several parallel decoders can be applied, each with another initial starting point for the gradient descent method. After all decoders have finished a fixed number of iterations, a final selection process evaluates the most probable transmitted code word. The setup is shown in Figure 6. Each of

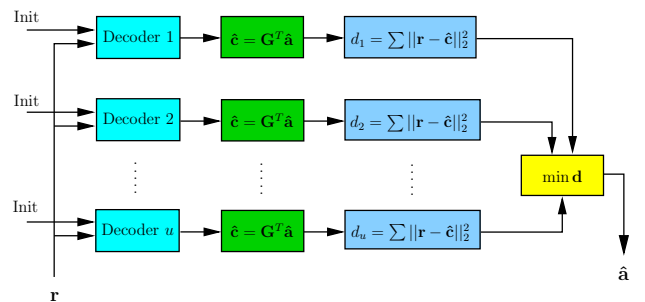


Fig. 6. Decoding using parallel networks.

the parallel channel decoders iterates on the same receiver data set. Thus no data transfer between the optimization networks is required during the iteration process and consequently each decoder can be executed on an own processor or as an own thread.

Final simulation results show that this approach does not outperform the Viterbi decoder but keeps

track with the Viterbi by providing only a marginal performance loss, especially when several RNNs are running in parallel. This is shown in Figure 7. At a potential working point with a bit error rate of  $10^{-4}$  there is only a small difference between the Viterbi decoder and different numbers of RNNs.

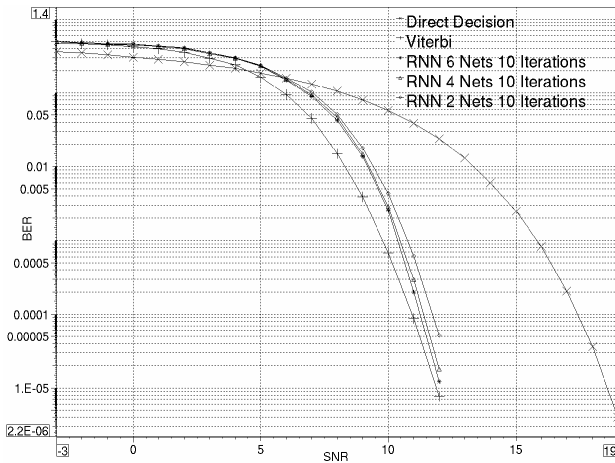


Fig. 7. Six, four and two parallel RNNs in comparison with the Viterbi decoder.

Thus, this section has shown that it is possible to develop physical radio algorithms, which can run as independent threads on a multi-processor platform.

## V. CONCLUSIONS

It was shown that the granularity of the presented matrix inversion algorithms is too fine for efficient implementations on a multi-processor platform when computing matrices with sizes smaller than  $100 \times 100$ . However, these matrix sizes are often used in state-of-the-art signal processing algorithms like Wiener filtering. The use of inversion algorithms for small matrices leads to a significant execution speed drop on multi-core platforms because of management overhead that is required in parallel execution. The evaluated matrix inversions are hardly useful for software implementation together with parallel execution because of the required synchronization during the inversion for QR, Cholesky, and Schur-Trench and the strong data dependencies in Levinson-Durbin-Trench. This shows that algorithms used in hardware implementation are not necessarily suitable for parallel processor software radio architectures.

Thus, new mathematical descriptions need to be developed, especially for parallel processing in physical layer radio applications. Therefore, this paper has proposed an optimization network as alternative channel decoding concept. This approach is a

promising candidate for parallel processing solutions, because each optimization network runs as own thread without generating data interdependencies between the different threads.

## REFERENCES

- [1] A. Kupiers, V. Fischer, "Open-Source Implementation of a Digital Radio Mondiale (DRM) Receiver", *9th International IEE Conference on HF Radio Systems and Techniques*, Bath, UK, 2003.
- [2] E. Coersmeier, M. Hoffmann et al., "Digital Radio Mondiale Architecture and Applications", *ISART 2005*, Boulder, USA.
- [3] E. Coersmeier, A. Bauer et al., "Evaluation Architecture for Digital Radio Mondiale Multimedia Applications", *CSA 2005*, Banff, Canada, July 2005.
- [4] F. Hofmann, C. Hansen and W. Schäfer, "Digital Radio Mondiale (DRM) Digital Sound Broadcasting in the AM Bands", *IEEE Transactions on Broadcasting*, vol. 49, no. 3, September 2003.
- [5] European Telecommunication Standards Institute, "Digital Radio Mondiale (DRM) System Specifications", *ETSI TS 201 980 V2.1.1*, June 2004.
- [6] O. Edfords, M. Sandell et al., "OFDM Channel Estimation by Singular Value Decomposition", *IEEE Transactions on Communications*, vol. 46, no. 7, July 1998.
- [7] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, third edition, 1996.
- [8] P. Hoeher, S. Kaiser and P. Robertson, "Two-Dimensional Pilot-Symbol-Aided Channel Estimation by Wiener Filtering", *IEEE ICASSP 1997*, München, Germany, April 1997.
- [9] E. Coersmeier, A. Bauer et al., "Dynamic Wiener Filter Coefficient Update for Software Radio", *Workshop on Software Radio*, Karlsruhe, Germany, 2006.
- [10] E. Coersmeier, A. Bauer et al., "Improving Channel Estimation for Software Radios", *CSA 2006*, Banff, Canada, July 2006.
- [11] G. H. Golub and C. F. van Loan, *Matrix Computations*, The John Hopkins University Press, third edition, 1996.
- [12] M. C. Pease, "Matrix inversion using parallel processing", *Journal of the ACM*, vol. 14, issue 4, pp. 757-764, October 1967.
- [13] M. R. Kant and T. Kimura, "Decentralized parallel algorithms for matrix computation", *Proceedings of the 5th annual symposium on computer architecture*, pp. 96-100, 1979.
- [14] R. P. Brent, "Parallel algorithms for Toeplitz systems", *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms*, Springer Verlag, pp. 75-92, 1991.
- [15] A. J. Viterbi, "Error Bounds for Convolutional Codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Technology*, 13:260-269, April 1967.
- [16] G. D. Forney, "The Viterbi Algorithm", *Proc. IEEE*, 61:268-278, March 1973.
- [17] A. Hämmäläinen and J. Henriksson, "Convolutional Decoding using Recurrent Neural Networks", *Proceedings of Inter. Joint Conf. on Neural Networks*, 5:3323-3327, San Diego, 1999.
- [18] E. Coersmeier, K. Hueske et al., "Combining Cognitive Radio and Software Radio approach for low complexity receiver architecture", *ISART 2007*, Boulder, USA.