THE LD-RLS ALGORITHM WITH DIRECTIONAL FORGETTING IMPLEMENTED ON A VECTOR-LIKE HARDWARE ACCELERATOR

Roman Bartosinski

Department of Signal Processing
Institute of Information Theory and Automation, UTIA AV CR
Pod Vodárenskou věží 4, Praha 8, Czech Republic

ABSTRACT

The paper discusses an RLS algorithm based on the LDU decomposition (LD-RLS) with directional forgetting implemented on an embedded system with a vector-oriented hardware accelerator. The LD-RLS algorithm can be attractive for control applications to identify an unknown system or to track time-varying parameters. A solution of the LD-RLS algorithm directly contains the estimated parameters. It also offers a possibility to use a priori information about the identified system and its parameters. The implementation of the LD-RLS algorithm is done on an FPGA-based accelerator from a high-level abstraction. It is compared with an implementation of the same algorithm in software on the same platform.

***Index Terms*—** FPGA, RLS, LDU decomposition, directional forgetting, hardware accelerator

1. INTRODUCTION

Today's implementations of adaptive algorithms for embedded systems use mainly the least mean square (LMS) algorithms for their simplicity and low computational complexity which result in high speed and throughput. But a better algorithm based on method of least squares is necessary in many applications. Mainly the recursive least squares (RLS) method with exponential forgetting (EF) is used in such cases.
The RLS algorithm can be implemented in many ways. In cases when the extended information matrix $\mathbf{V}_k$ is numerically ill-conditioned (nearly singular), we must use a computation which numerically ensures positive semi-definiteness of $\mathbf{V}(k)$ for all k. Otherwise the entire identification can numerically collapse. This problem has been solved with radical algorithms which use suitable decompositions of the inversion of the matrix $\mathbf{V}(k)$. Algorithms that use QR, Cholesky, LDU and UDL decompositions are examples of such algorithms.
The main drawback of the EF method is called *wind-up*, and it comes when a data vector is not persistently exciting, i.e. when it does not carry sufficient information. The old data is discounted continuously, but only a part of the old data can be replaced by new data. As a consequence, some eigenvalues of the covariance matrix will tend to be zero, and the Kalman gain will tend to be unbounded. In that case the algorithm is very sensitive to noise, and thus the estimation may be completely unreliable. One of the methods to avoid the EF windup is *directional forgetting* DF which is attractive for its potential performance and algorithmic simplicity. The disadvantage of the DF technique is its higher data dependence compared to EF. In DF algorithms, the data is considered to have directions, and the old data is exponentially forgotten only in specific directions. The mathematical background of the DF LD-RLS is in Section 2.1
The algorithm has been implemented on a floating point vector processing accelerator [1], [2]. A concept of the platform used is based on floating point computing units and a data flow unit which is controlled by a microcontroller configured by a host CPU. The platform is described in more detail in Section 2.2.

This work was partially supported by the ARTEMIS JU and Ministry of Education, Youth and Sports of the Czech Republic under the project SMECY Artemis JU 100230, MSMT 7H10001.

Related work. The method of directional forgetting is also called *restricted exponential forgetting*, and it is described in [3]. [4] and [3] propose a DF algorithm based on the Bayesian estimation approach. The RLS with UDL factorization of the covariance matrix (UD-RLS) were used besides the QR factorization for example in [5], where the authors proposed two architectures for UD-RLS with DF based on systolic architectures with $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ processing elements. Many works on FPGA-based implementations of LMS and QR RLS algorithms have been published, but to my best knowledge there hasn't been any work about FPGA-based implementation of the LD-RLS algorithm or directional forgetting.
A similar accelerator with a different structure of data paths and accesses to data memories is described in [6].

2. BACKGROUND

This section contains relevant background on the RLS algorithm based on the LDU decomposition (LD-RLS) with directional forgetting (DF) and overview of the hardware platform used for implementation.

2.1. The LD-RLS algorithm with directional forgetting

In the following text we will suppose that a system is modeled by the ARMA process which has defined output as a function of its inputs, past outputs and unmeasurable white error.

$$y(k) = -\sum_{i=1}^{na} a_i y(k-i) + \sum_{i=0}^{nb} b_i u(k-i) + e_s(k), \qquad (1)$$

which can be written as a product of vectors of model parameters $\mathbf{\Theta}(k)$ and a data regressor $\mathbf{z}(k)$

$$\begin{aligned} &y(k) = \mathbf{\Theta}^T(k)\mathbf{z}(k) + e_s(k), \\ &\mathbf{\Theta}^T(k) = [a_1, a_2, ..., a_{na}, b_0, b_1, ..., b_{nb}], \\ &\mathbf{z}^T(k) = [-y(k-1), -y(k-2), ..., -y(k-na), \\ &\qquad u(k), u(k-1), ..., u(k-nb)]. \end{aligned} \qquad (2)$$

In this work, we use square-root factorization of the correlation matrix $C = V^{-1}$ based on the LDU decomposition of the form $C = LDU$, where D is a diagonal matrix, L is a lower triangular matrix with unit diagonal elements, and $U = L'$ ([7], [8]). The computation of roots is not necessary in this method. Another advantage of the LD-RLS algorithm is that the estimated parameters $\hat{\mathbf{\Theta}}$ are directly included in the solution of the decomposition. It allows to use a priori information about the identified system and its parameters.
In this algorithm, the forgetting factor is set according to the information in the input data. DF is more robust than EF for systems with insufficiently excited inputs, but it brings high data dependence, and therefore the method is not used so often. The information matrix $V_z(k)$, covariance matrix $C(k) = V_z^{-1}(k)$ and parameter estimation $\hat{\Theta}(k)$ evolve according to the incoming information as shown in the following equations (3).

$$\begin{aligned} &\|\zeta(k-1)\| > 0 \\ &\quad \mathbf{V_z}(k) = \mathbf{V_z}(k-1) + \left(\lambda(k) - \frac{1-\lambda(k)}{\zeta(k-1)}\right)\mathbf{z}(k)\mathbf{z}^T(k), \\ &\quad \mathbf{C}(k) = \mathbf{C}(k-1) - \frac{\mathbf{C}(k-1)\mathbf{z}(k)\mathbf{z}^T(k)\mathbf{C}(k-1)}{\varepsilon^{-1}(k-1) + \zeta(k-1)}, \\ &\quad \hat{\mathbf{\Theta}}(k) = \hat{\mathbf{\Theta}}(k-1) + \frac{\mathbf{C}(k-1)\mathbf{z}(k)}{1+\zeta(k-1)}\hat{e}(k), \\ &\|\zeta(k-1)\| = 0 \\ &\quad \mathbf{V_z}(k) = \mathbf{V_z}(k-1). \\ &\quad \mathbf{C}(k) = \mathbf{C}(k-1), \\ &\quad \hat{\mathbf{\Theta}}(k) = \hat{\mathbf{\Theta}}(k-1), \end{aligned} \qquad (3)$$

where

$$\begin{aligned} &\zeta(k-1) = \mathbf{z}^T(k)\mathbf{C}(k-1)\mathbf{z}(k), \\ &\varepsilon(k-1) = \lambda(k) - \frac{1-\lambda(k)}{\zeta(k-1)}, \\ &\hat{e}(k) = y - \hat{\Theta}^T(k-1)\mathbf{z}(k) \end{aligned}$$

For recursive algorithms direct updating of variables is important. In this part we present equations for updating the DF LD-RLS algorithms which are used in implementations in the next chapter. A vectorized form of the directly updated DF LD-RLS algorithm used in the implementation is shown in equations (4) and (5). The algorithm has two updating ways according to the excitation of input data ($h_2(k) = \zeta(k)$). If the input data are sufficiently excited ($h_2(k) > 0$), the matrices L and D are updated

$$\begin{aligned} &L_{i,j}(k) = L_{i,j}(k-1) - \frac{f_j(k) g_i^{(j+1)}(k)}{\alpha_j(k) + h_{j+1}(k)} \\ &\bar{D}_i(k) = D_i(k-1)\frac{\alpha_i(k) + h_{i+1}(k)}{\alpha_i(k) + h_i(k)} \\ &D_1(k) = \frac{\bar{D}_1(k)}{\lambda}, \quad D_i(k) = \bar{D}_i(k) \quad \forall i \in (2..n) \end{aligned} \qquad (4)$$

where

$$\begin{aligned} &\mathbf{f}(k) = L(k-1)\mathbf{d}(k) \\ &\mathbf{g}_i(k) = D_i(k-1) f_i(k) \\ &h_i(k) = \sum_{l=i}^{n} f_l(k) g_l(k), \quad h_{n+1}(k) = 0; \\ &g_i^{(m)}(k) = g_i^{m+1}(k) + D_m(k-1) L_{i,m}(k-1) f_m(k) \\ &\alpha_1(k) = 1, \quad \alpha_i(k) = \psi(k) \quad \forall i \in (2..n) \\ &\psi(k) = \frac{h_2(k)}{h_2(k)(\lambda+1) - 1} \end{aligned}$$

If the input data aren't sufficiently excited ($h_2(k) \to 0$), only the first element of the diagonal matrix D is updated.

$$D_1(k) = \frac{D_1(k-1)}{\lambda(1 + h_1(k) - h_2(k))} \qquad (5)$$

2.2. UTIA EdkDSP platform

The *UTIA EdkDSP Platform* [1], [2] has been used to implement the DF LD-RLS algorithm. The UTIA EdkDSP platform is a generic concept of a flexible, reprogrammable and reconfigurable hardware accelerator. The domain of use depends on the configuration of the platform.The platform is intended as a hardware accelerator for a general-purpose processor in a system-on-chip.
The basic implementation of the UTIA EdkDSP platform with pipelined floating-point (FP) operations such as addition, multiplication and division is denoted as the *Basic Computing Element* (BCE).
Figure 1 depicts the BCE which consists of basic pipelined floating-point operations, a *data-flow unit* (DFU), dual-ported

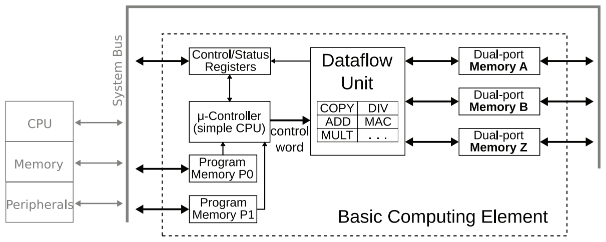

Fig. 1. BCE from the point of view of the host CPU.

data memories which can be connected to a host system for direct data exchange, a simple micro-controller (SCPU) and memories with firmware for the micro-controller. The DFU provides data paths between the data memories and the basic FP operations. The platform can be configured with more DFU and FP units which can operate in a *single instruction multiple data* (SIMD) computing mode. The current DFU configuration is controlled by a control word generated in the SCPU according to its firmware. The firmware for the SCPU can be prepared in the host CPU, and it allows to perform the entire required algorithm as one sequence of FP operations. The concept of the platform is also suitable for building an array of hardware accelerators running in parallel where the dual-ported data memories can be used for direct interconnection between the accelerators.
Algorithms can be implemented on several levels: an algorithm implemented purely in software on a host CPU, algorithm implemented in the host CPU software with FP operations performed by BCE, the entire algorithm in the SCPU firmware, and the entire algorithm in hardware as a hard-wired data path in the DFU or a new built-in FP operation.

3. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The DF LD-RLS algorithm has been implemented as a SCPU firmware with basic FP functions in hardware. The firmware with the algorithm has been automatically generated from a Matlab code which describes the algorithm, the set of basic operations and the configuration of data memories in the accelerator. Then our tool maps all variables to data memories and generates the firmware code. The concept of implementing the algorithm in firmware, i.e. on a higher level of abstraction, makes the implementation simpler, and it increases the productivity of development against an implementation hard-wired in hardware. Experiments with the BCE platform show that implementations in firmware can reach the same performance as implementations in hardware.

3.1. Used hardware

All the results presented in this section have been obtained from an implementation in the Xilinx 'Embedded Development HW/SW Kit - Spartan-3A DSP S3D1800A MicroBlaze Processor Edition' with Xilinx FPGA XC3SD1800A.
The BCE accelerators have been implemented with the single precision FP computing units ADD, MULT, DIV and a FP CMP unit for comparing data and branching the algorithm in the SCPU firmware. The accelerator in this configuration provides vector FP operations shown in Table 1.

Operation	**latency**[ClC]	**Description**
VCOPY	0	$Z_i = A_i$
VADD	3	$Z_i = A_i + B_i$
VSUB	3	$Z_i = A_i - B_i$
VMULT	4	$Z_i = A_i * B_i$
DPROD	≥ 3	$Z_0 = \sum_i (A_i * B_i)$
VMAC	8	$Z_i = Z_i + A_i * B_i$
DIV	16	$Z_i = A_i / B_i$
CMP	2	$Z_i = 1\, if (A_i = B_i)$
		$Z_i = 0\, otherwise$

Table 1. Used BCE FP operations (ClC=Clock Cycles)

The basic FP operations have been generated in the Xilinx CORE Generator. Their latencies and numbers of used DSP blocks have been selected to maximize the estimated maximal clock frequency of the accelerator, hence the implementation doesn't use DSP blocks as shown in Table 2.

Resource	**BCE accelerator**	**MicroBlaze SoC**
Occupied Slices	2848	8442
Slice Flip Flops	2483	7119
4 input LUTs	4837	11115
Block RAMs	10	34
DSP48	0	8

Table 2. The occupied FPGA resources for the used accelerator and remaining SoC(with HW FPU)

The accelerator also contains three data memories, each for 1024 single precision FP values. The Xilinx KCPSM3 has been used as the SCPU. The maximal clock frequency of the BCE accelerator in this implementation is 69.7 MHz. The entire system was run on 62.5 MHz for all experiments. The amount of resources required by the accelerator is shown in Table 2. The values are for a standalone BCE accelerator with single-precision FP operations as a peripheral core directly connected to a SoC with MicroBlaze as the host CPU.

3.2. Results

Figure 2 shows the performance of the implementation in the BCE accelerator and in software on the host CPU with the hardware FP co-processor. The times are measured include data transfers from the CPU to the accelerator before the computation and reading the data from the accelerator by the host

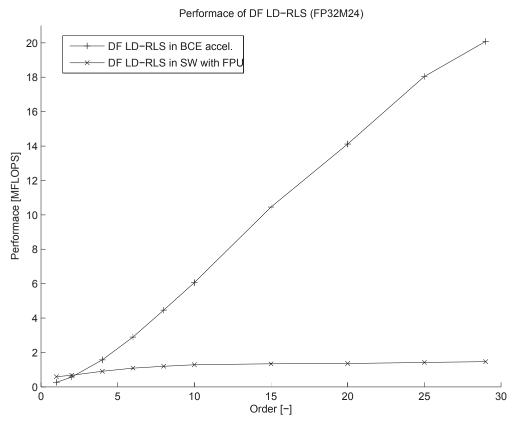

Fig. 2. Accelerator performance for the DF LD-RLS algorithm

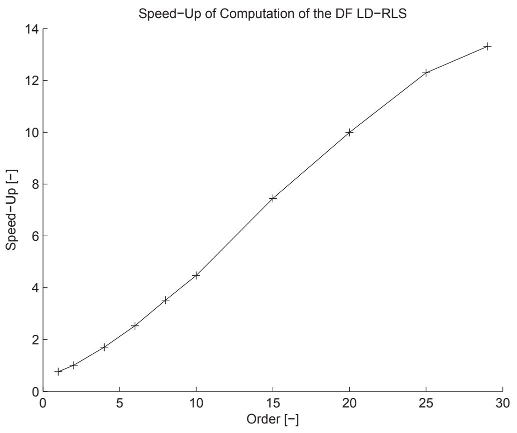

Fig. 3. Accelerator speed-up for the DF LD-RLS algorithm

CPU after the computation. The figure also shows that the algorithm with an order (the number of estimated parameters) less than 2 is inefficient, because the computed vectors are too short. Figure 3 shows the speed-up of the computation performed by the hardware accelerator against the computation in software with the FP co-processor (FPU).
The figures show that the speed-up grows with a higher order of the algorithm; this is because the accelerator uses longer vectors in operations. For the computation in software without FPU the speed-up is about hundred times better than with FPU. The speed-up isn't higher because the directional forgetting adds additional data dependences to the algorithm, and therefore it cannot be parallelized in a better way.

4. CONCLUSION

In this paper, we have explored and described a hardware implementation of the RLS algorithm based on the LDU decomposition with directional forgetting. The implementation has been based on a hardware accelerator for vector processing.
The LD-RLS algorithm is interesting for control applications to identify an unknown system or to track time-varying parameters. The solution directly contains the estimated parameters, and their uncertainties can be evaluated in a simple way. The next advantage is the possibility to use a priori information about the identified system and its parameters. From the point of view of the implementation this algorithm has the advantage that it doesn't need to compute the roots. The disadvantage of the LD-RLS algorithm is its higher computational complexity compared to LMS and more complicated data dependences in the algorithm. The complexity of the LD-RLS with directional forgetting is $\mathcal{O}(n) = 3n^2 + 6n + 2$ FLOPs, where n is the number of parameters.

5. REFERENCES

[1] J. Kadlec, R. Bartosinski, and M. Daněk, "Accelerating Microblaze Floating Point Operations," in *Proceedings 2007 International Conference on Field Programmable Logic and Applications (FPL)*, 2007.

[2] Martin Daněk, Jiří Kadlec, Roman Bartosinski, and Lukáš Kohout, "Increasing the Level of Abstraction in FPGA-based Designs," in *Proceedings 2008 International Conference on Field Programmable Logic and Applications (FPL)*, 2008.

[3] R. Kulhavý, "Restricted exponential forgetting in real-time identification," *Automatica*, vol. 23, no. 5, pp. 589 – 600, 1987.

[4] R. Kulhavý and M. Kárný, "Tracking of slowly varying parametrs by directional forgetting," in *Proceedings 9th IFAC World Congress*, 1984, vol. 10, pp. 78–83.

[5] L. Chisci and E. Mosca, "Parallel architectures for RLS with directional forgetting," *International Journal of Adaptive Control and Signal Processing*, vol. 1, no. 1, pp. 69 – 88, 1987.

[6] Jason Yu, Christopher Eagleston, Christopher Han-Yu Chou, Maxime Perreault, and Guy Lemieux, "Vector Processing as a Soft Processor Accelerator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 12:1–12:34, June 2009.

[7] V. Peterka et al., "Algorithms for adaptive microprocessor regulation of technological processes (in czech)," Tech. Rep., UTIA CSAV, Praha, CZ, 1982.

[8] John G. Proakis, Chrysostomos L. Nikias, Charles M. Rader, Fuyun Ling, Marc Moonen, and Ian K. Proudler, *Algorithms for Statistical Signal Processing*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.