Kybernetika

Jiří Kadlec

Transputer implementation of block regularized filtering

Kybernetika, Vol. 32 (1996), No. 3, 235--250

Persistent URL: http://dml.cz/dmlcz/125518

KYBERNETIKA — VOLUME 32 (1996), NUMBER 3, PAGES 235 - 250

TRANSPUTER IMPLEMENTATION OF BLOCK REGULARIZED FILTERING

JIŘÍ KADLEC

A novel approach to parallel transputer implementation of a regularized exponential parameter tracking is described. The proposed block regularized exponential algorithm allows to define and automatically adjust mean value of an alternative covariance of the estimated parameters in the boundaries of blocks of processed data. The alternative mean tracks in blocks the current parameter estimates. That is why the influence of the regularization on the parameter estimates is reduced, and the algorithm remains compatible with the completely pipelined parallel transputer implementation.

1. INTRODUCTION

A brief introduction to the existing parallel technology for implementation of identification and control algorithms is presented in this section. The objective is to provide an overview of the transputer technology.

Hardware used in signal processing and control applications

A wide variety of hardware has been employed for embedded real-time systems, ranging from conventional microprocessors to digital signal processors (DSPs). The main advantage of DSPs over general purpose microprocessors is that they contain dedicated circuitry which provides high resolution and high speed arithmetic operations. Furthermore, these devices contain A/D and D/A converters on-chip and require little external circuitry. The increasing trend in this area is to provide low-cost application specific integrated circuits (ASICs) which, although restrict flexibility, increase the computational rates and simplify the support hardware requirements. One of the most commonly used digital processing devices is the TI TMS 320.

Despite the success of embedded digital controllers in industrial control applications, an increasing number of parallel processors, such as the Inmos transputer, are appearing on the market. Increased computational speed is of course the primary benefit of parallel processing. This allows faster systems to be controlled and gives the designer the choice of added complexity in the algorithms. Easy expansion, within a uniform hardware and software base is another feature of concurrent systems,

since it is possible to add more processors as required which has important implications for reduced development and maintenance costs. Parallel processing also offers a closer relationship between the inherent parallelism expressed at the design stage, in the form of block diagram descriptions of the system, and the hardware implementation.

Parallel architectures

The most commonly accepted classification of digital computers was introduced by Flynn in 1966 [2], based on the multiplicity of instruction and data streams. This results in four categories:

1. Single instruction stream – Single data stream (SISD)
2. Single instruction stream – Multiple data stream (SIMD)
3. Multiple instruction stream – Single data stream (MISD)
4. Multiple instruction stream – Multiple data stream (MIMD)

where an instruction stream is a sequence of instructions executed by the machine and a data stream is a sequence of data including input, partial or temporary results, called for by the machine [8]. The most useful parallel processing architectures can be classified as SIMD, for example a systolic array, or as MIMD, for example an array of transputers.

Systolic arrays. The systolic array was first proposed by H. T. Kung and C. E. Leiserson in 1980 [19]. This innovative array was originally proposed to exploit VLSI technology as it consisted of an array of simple, mostly identical, processing elements arranged in a regular, repeated structure. Furthermore, each processing element is connected only to its nearest neighbors and communication is achieved by pumping data through the array on each cycle of a clock. As a result data is passed through the array in a specific direction and the term 'systolic' is used to describe this rhythmical movement which is analogous to the pumping action of the heart.

During the last ten years, the concept of a systolic array has evolved to provide a graphical representation of the computational and data flows for a concurrent description of an algorithm. Indeed, considerable effort has been devoted to the design of systolic systems ranging from convolution to Kalman filtering [20]. The general consensus emerging from this work seems to be that the structures should be referred to as systolic architectures as their actual hardware implementation could be either a dedicated integrated circuit (SIMD) or an array of transputers (MIMD). Furthermore, the advances in systolic arrays have provided a major stimulus to the development of other array processors such as the wavefront array. This latter type, which was proposed by S. Y. Kung [20], operates in a similar fashion as the systolic array except that the communication is asynchronous, as the global clock pulse is replaced by a local handshaking type procedure.

Transputers. The Inmos transputer, which was introduced in the mid 1980's, is a 32-bit microprocessor specifically designed for implementing parallel systems. Essentially, the device is a microprocessor with on-chip RAM and with the novel feature of four bi-directional serial links which allow point-to-point communication between one transputer and another. This allows arrays of transputers to be connected together to form MIMD systems.

There are various generations of the transputer commercially available, such as the T800. It contains an on-chip 64 bit floating point unit and the hardware floating point unit sustains 2.5 million floating point operations per second (FLOPS). In addition, the T800 transputers employ four links with speeds 20 Mbits/sec.

The next generation of the transputer, the T9000, provides further improvements over the T800 processor as it contains a pipelined superscalar micro-architecture and 16 Kbytes of on-chip instruction and data cache. Furthermore, this processor is capable of a peak performance of 25 million FLOPS running at a clock speed of 50 MHz. In addition, the four communication links provide a total of 80 Mbytes/sec bi-directional bandwidth.

The development of the the T9000 transputer has been delayed for several years in early 90's. The vendors of parallel systems developed multiple alternative solutions. The TI C40 signal processors operate with 6 eight-bit wide links and 40 million FLOPS CPU. The drawback of this processor is the cost. The true Parallel C support emerged only in 1995.

Another example can be the combination of a transputer with a powerful CPU like the DEC AXP Alpha. The DEC 21066 Alpha processor is fully pipelined, dual-issue 64-bit advanced RISC processor. Running at 233 MHz, it is capable of a maximum 233 million FLOPS in IEEE double precision format and performs 466 million instructions per second. There are 300 MHz versions of this processor but not available for commercial market (spring 1995). The four Inmos compatible bi-directional transputer links are used to connect multiple Alpha processors into network. These "twins" are programmed in Parallel C as standard nodes of the transputer array.

Programming languages. The transputer was originally developed to utilize the occam programming language, although over the last few years various other compilers have been developed. Occam is a unique parallel programming language which has roots in the mathematics of communicating sequential processes [7]. The language encompasses a design formalism which has strong modularity and yet is simple to use. An occam program consists of a hierarchy of parallel processes which can be run on one or more transputers, with the communication synchronized by data paths known as channels. These features provide a powerful tool for the development of concurrent programs and all of the real-time implementations, presented in the following sections, were programmed using the occam language.

Parallel Implementation Issues

The innovation of parallel processing has added a new dimension to the design of algorithms and programs. Parallel programming is not a simple extension of its serial

counterpart but it requires a complete reconsideration of the implementation process in order to fully exploit the possibilities offered by concurrency. Furthermore, the widespread availability of parallel processing hardware has led to a diverse range of applications from database management systems to real-time embedded control systems.

An efficient parallel implementation must exploit both the capabilities of the target architecture and the natural parallelism inherent in the algorithm. There are three important considerations involved in the process of partitioning an algorithm to obtain a concurrent realization:

- Subdivision of the algorithm into parallel processes
- Inter-process communication
- The efficient execution of each process.

A crucial requirement, when subdividing an algorithm, is to obtain a balanced computational load across each of the processes. This ensures that no one process dominates over the others which provides the most effective concurrent realization. The procedure involves decomposing the problem into a set of tasks, or processes, to extract the natural parallelism inherent in the algorithm.

Inter-process communication is the major overhead in parallel processing systems. Indeed, communication delays can considerably degrade the performance of a concurrent system and render its application unjustifiable. Thus, it is a crucial requirement to obtain an effective trade-off between the communication and computation demands.

The transputer application reported in this paper was programmed in Parallel C for a network of T800 transputers and a network of DEC 21066 233 MHz Alpha processors. The implementation runs under MS Windows 3.1 version of Matlab 4.2. The transputer/Alpha network is interfaced to Matlab via the Alpha-Bridge software development support utility *Alpha-Bridge* [14].

A parallel transputer implementation of a regularized exponential parameter tracking is described in next sections of the paper. The formulation of the regularized parameter tracking takes in to account the transputer implementation requirements outlined above. This results in the final parallel algorithm suitable for the transputer implementation.

2. RECURSIVE IDENTIFICATION

Recursive parameter estimation and filtering is a key tool in modern application areas such as the adaptive signal processing or control.

When there is little information available about the detailed structure of the system, the standard Kalman filter model [3, 4] is reduced to a "random-walk of parameters" regression type model

$$\begin{aligned} \theta_{k+1} &= \theta_k + w_k \\ y_k &= \varphi_k' \theta_k + v_k \,. \end{aligned} \tag{1}$$

Here θ_k is the vector of n unknown system parameters, y_k is the sampled system output, φ_k is the vector of n past input and output data, v_k and w_k are assumed to be the zero mean, Gaussian, white noise sequences characterized by the covariance matrix W_k and V_k respectively.

The related estimator takes the form of the standard Kalman-filter data update step (2) combined with different specifications of the time update part of the recursive parameter estimation.

Data update.

$$\begin{aligned} \hat{e}_{k|k-1} &= y_k - \varphi_k' \hat{\theta}_{k|k-1} \\ P_{k|k} &= P_{k|k-1} - \frac{P_{k|k-1}\varphi_k\varphi_k' P_{k|k-1}}{1 + \varphi_k' P_{k|k-1}\varphi_k} \\ \hat{\theta}_{k|k} &= \hat{\theta}_{k|k-1} + P_{k|k}\,\varphi_k\,\hat{e}_{k|k-1}. \end{aligned} \tag{2}$$

Kalman filter-like time update. Specified by $0 \leq W_k$.

$$\begin{aligned} \hat{\theta}_{k+1|k} &= \hat{\theta}_{k|k} \\ P_{k+1|k} &= P_{k|k} + W_k. \end{aligned} \tag{3}$$

The alternative concept of exponential age–weighting [23] by the scalar exponential weighting factor $0 < \lambda_k \leq 1$ is frequently used when an effective implementation is required [25].

Exponential forgetting time update. Specified by $0 < \lambda_k \leq 1$.

$$\begin{aligned} \hat{\theta}_{k+1|k} &= \hat{\theta}_{k|k} \\ P_{k+1|k} &= \lambda_k^{-1} P_{k|k}. \end{aligned} \tag{4}$$

The adding of W_k to $P_{k|k}$ (3) is avoided in (4) and the algorithm can be based on the effective QR update of the square root factors of the information matrix $P_{k|k}^{-1}$ and implemented on the triangular systolic array of Gentleman–Kung [24], [21], [28].

Limits of the standard time update formulation

When the measured data contains little information about some of the parameters, the parameter estimates become increasingly dependent on the parameters specifying the time update. Standard time updates (3) or (4) take into account neither the information contained in previous data, nor prior information about the possible values of the estimated parameters. Thus, numerical problems could eventually occur, even in the case of numerically robust factorised implementations.

Several techniques to cope with poor informative data have been proposed in the current literature. Directional forgetting [6, 18], the constant trace approach [22] and the shift invariant directional forgetting are only a few examples. Their original formulation has been designed for efficient sequential implementation. If these

methods are directly mapped onto a systolic implementation, the global feedback loops in the signal flow degrade considerably the data throughput of the array. Their formulation must therefore be revised in the pipelined transputer implementation to avoid low processor utilization. This paper addresses this issue.

The concept of regularized estimator

To overcome these limitations, we shall design a systolic regularized estimator using the general concept of parameter tracking introduced in the Bayesian formulation by Kulhavý in [16] and analyzed in [17].

Generalized exponential forgetting time update. Specified by $\theta_k^*\,;\,(P_k^*)^{-1} \geq 0\,;\,0 < \lambda_k \leq 1$.

$$\begin{aligned} A_k &\hat{=} \left\{ I + \frac{1-\lambda_k}{\lambda_k} P_{k|k} (P_k^*)^{-1} \right\}^{-1} \\ \hat{\theta}_{k+1|k} &= A_k \hat{\theta}_{k|k} + (I - A_k)\,\theta_k^* \\ P_{k+1|k} &= \left\{ \lambda_k P_{k|k}^{-1} + (1-\lambda_k)\,(P_k^*)^{-1} \right\}^{-1}. \end{aligned} \tag{5}$$

The time update step (5) could be understood as the introduction of an additional, alternative reference matrix $(P_k^*)^{-1} > 0$ and parameter estimate θ_k^* exhibiting the role of an "alternative value" or "regularizing point" in the specification of the time update step.

Regularized information filtering

The vectors $\hat{\theta}_{k|k}\,;\,\theta_k^*$ and (n,n) matrices $P_{k|k}\,;\,P_k^*$ can be propagated in transformed, inverse or factorised forms. The information filter works with the transformed regression parameter estimates denoted as

$$d_{k|k} \hat{=} P_{k|k}^{-1} \hat{\theta}_{k|k}\;;\; d_k^* \hat{=} (P_k^*)^{-1} \theta_k^*. \tag{6}$$

Information data update.

$$\begin{aligned} P_{k|k}^{-1} &= P_{k|k-1}^{-1} + \varphi_k \varphi_k' \\ d_{k|k} &= d_{k|k-1} + \varphi_k y_k \\ \hat{\theta}_{k|k} &= P_{k|k} d_{k|k}. \end{aligned} \tag{7}$$

Regularized information time update. Specified by $d_k^*\,;\,(P_k^*)^{-1} \geq 0\,;\,0 < \lambda_k \leq 1$.

$$\begin{aligned} d_{k+1|k} &= \lambda_k d_{k|k} + (1-\lambda_k)\,d_k^* \\ P_{k+1|k}^{-1} &= \lambda_k P_{k|k}^{-1} + (1-\lambda_k)\,(P_k^*)^{-1}. \end{aligned} \tag{8}$$

The information data update (7) is an inverse equivalent of (2) and the regularized information time update (8) is identical to (5) as follows directly from (6) substituted into (5).

One of possible square root systolic implementations of (7), (8) is proposed in next section.

3. SQUARE ROOT SYSTOLIC IMPLEMENTATION

We propose implementation of the regularized information filter (7), (8) as an estimator implementing recursive QR decomposition technique [25], based on Gentleman–Kung array [24], [21], [28].

The QR algorithm [5, 24] implements the data update of the factorised information matrix $P_{k|k}^{-1}$ (7) and the data update of vector $r_{k|k}$ defined

$$\begin{aligned} P_{k|k}^{-1} &\hat{=} R_{k|k}^T R_{k|k} \\ r_{k|k} &\hat{=} R_{k|k}\,\hat{\theta}_{k|k} = R_{k|k}^{-T} d_{k|k} \end{aligned} \tag{9}$$

where $R_{k|k} > 0$ is upper triangular matrix.

Regularized QR factorization

Let us define analogically to (9) the factorization

$$\begin{aligned} (P_k^*)^{-1} &\hat{=} R_k^{*T} R_k^* \\ r_k^* &\hat{=} R_k^*\,\theta_k^* = (R_k^*)^{-T} d_k^*. \end{aligned} \tag{10}$$

$$\begin{bmatrix} P_{k|k}^{-1} & d_{k|k} \\ d'_{k|k} & * \end{bmatrix} \hat{=} \tilde{R}_{k|k}^T \tilde{R}_{k|k} = \begin{bmatrix} R_{k|k} & r_{k|k} \\ 0 & * \end{bmatrix}^T \begin{bmatrix} R_{k|k} & r_{k|k} \\ 0 & * \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} (P_k^*)^{-1} & d_k^* \\ d_k^{*T} & * \end{bmatrix} \hat{=} \tilde{R}_k^{*T} \tilde{R}_k^* = \begin{bmatrix} R_k^* & r_k^* \\ 0 & * \end{bmatrix}^T \begin{bmatrix} R_k^* & r_k^* \\ 0 & * \end{bmatrix} \tag{12}$$

where R_k^* ; $\tilde{R}_{k|k}$; $\tilde{R}_k^*$ are upper triangular and $*$ is a "don't care" scalar term.

Then the regularized information filter (7) – (8) can be implemented in the square root factorised form (13), (14).

Square root regularized data update.

$$\tilde{R}_{k|k}^T \tilde{R}_{k|k} = \tilde{R}_{k|k-1}^T \tilde{R}_{k|k-1} + \begin{bmatrix} \varphi_k \\ y_k \end{bmatrix} \begin{bmatrix} \varphi_k \\ y_k \end{bmatrix}'. \tag{13}$$

Square root regularized time update. Specified by $\tilde{R}_k^* \geq 0$; $0 < \lambda_k \leq 1$.

$$\tilde{R}_{k+1|k}^T \tilde{R}_{k+1|k} = \lambda_k \tilde{R}_{k|k}^T \tilde{R}_{k|k} + (1 - \lambda_k) \tilde{R}_k^{*T} \tilde{R}_k^*. \tag{14}$$

It follows directly from compare with (7), (8) and the definition of factorization (9) and (10) – (12).

Systolic implementation of the regularized estimator

The proposed regularized parallel estimator implements (13) and (14). This is in fact the standard systolic array for the QR filtering with $(n) * (n+1)/2$ cells. The cells store and update the triangular matrix $R_{k|k}$ and regression parameter estimate is updated in the transformed form as $r_{k|k}$ on the extended row of n cells.

The array requires $n+1$ time steps between measurements to complete the recursion and can be formally described as follows.

Systolic regularized estimator array.

$$\left|\begin{array}{c} \varphi_k \\ y_k \\ \Leftarrow 1 \Rightarrow \end{array}\right| \quad \begin{array}{c} \mathrm{Q} \\ \Longrightarrow \end{array} \quad \left[\begin{array}{c} R^T_{k|k-1} \rightarrow R^T_{k|k} \\ r^T_{k|k-1} \rightarrow r^T_{k|k} \\ \text{QR array} \end{array}\right] \Longrightarrow \hat{e}_{k|k} \tag{15}$$

$$\left|\begin{array}{c} (1-\lambda_k)^{1/2} R^{*T}_k \\ (1-\lambda_k)^{1/2} r^{*T}_k \\ \Leftarrow n \Rightarrow \end{array}\right| \quad \begin{array}{c} \mathrm{Q}\ldots\mathrm{Q} \\ \Longrightarrow \\ \leftarrow n \rightarrow \end{array} \quad \left[\begin{array}{c} \lambda_k^{1/2} R^T_{k|k} \rightarrow R^T_{k+1|k} \\ \lambda_k^{1/2} r^T_{k|k} \rightarrow r^T_{k+1|k} \\ \text{QR array} \end{array}\right]. \tag{16}$$

The array implements one QR update (15) of the data measurement (13) and n additional QR updates (16) representing the "addition" of $(1-\lambda_k)^{1/2} R^{*T}_k$ and $(1-\lambda_k)^{1/2} r^{*T}_k$ as n columns of "artificial" data to implement the time update (14).

If additional information about the changes of the parameter estimates is available, it can be built into the regularizing terms r^*_k ; $R^*_k \geq 0$ related to θ^*_k and $(P^*_k)^{-1}$ through the definitions (6), (12).

The regularized array (15) – (16) provides as a by-product of the data update (15) the output filtration error $\hat{e}_{k|k}$ equally to the Gentleman–Kung array [21], [24], [28].

Regression parameter estimates extraction

The function of array (16) can be combined with the n data updates necessary for the parameter estimate extraction technique known as " weight flushing " [25, 28].

Systolic implementation of parameter extraction.

$$\left|\begin{array}{c} I_n \\ 0 \\ \Leftarrow n \Rightarrow \end{array}\right| \quad \Longrightarrow \quad \left[\begin{array}{c} R^T_{k|k}(\text{frozen}) \\ r^T_{k|k}(\text{frozen}) \\ \text{QR array} \end{array}\right] \Longrightarrow \hat{\theta}^T_{k|k}. \tag{17}$$

The regression parameter estimates can be captured as a serial stream of n items from the bottom of the array.

4. BLOCK REGULARIZED ALGORITHM

In [12], [13] a novel block regularized algorithm has been described. It is in fact a modification of the regularized exponential forgetting algorithm which allows parallel implementation.

The Block Regularized Algorithm [12] overcomes both the *Pipelining* and *Complexity* problems arising in the implementation of the regularized exponential algorithm (8)–(7). It is achieved by the additional assumption about the alternative covariance P^* and the alternative parameter vector θ^*. If they both remain constant within an block i. e.

$$P^* = P^*_{k-N} = P^*_{k-N+1} = \cdots = P^*_{k-1} \tag{18}$$

$$\theta^* = \theta^*_{k-N} = \theta^*_{k-N+1} = \cdots = \theta^*_{k-1} \tag{19}$$

the computation will be simplified as the $(P^*)^{-1}$ and $(P^*)^{-1}\theta^*$ terms can be accumulated and added in every Nth iteration as shown in (20)–(22)

$$\begin{aligned} P^{-1}_{k|k} &= \lambda_{k-1}\cdots\lambda_{k-N}P^{-1}_{k-N|k-N} + \\ &+ \lambda_{k-1}\cdots\lambda_{k-N+1}\varphi_{k-N+1}\varphi'_{k-N+1} + \lambda_{k-1}\cdots\lambda_{k-N+2}\varphi_{k-N+2}\varphi'_{k-N+2} \\ &+ \cdots + \lambda_{k-1}\varphi_{k-1}\varphi'_{k-1} + \varphi_k\varphi'_k + \lambda_{[k-N,k]}(P^*)^{-1} \end{aligned} \tag{20}$$

$$\begin{aligned} d_{k|k} &= \lambda_{k-1}\cdots\lambda_{k-N}d_{k-N|k-N} \\ &+ \lambda_{k-1}\cdots\lambda_{k-N+1}\varphi_{k-N+1}y_{k-N+1} + \lambda_{k-1}\cdots\lambda_{k-N+2}\varphi_{k-N+2}y_{k-N+2} \\ &+ \cdots + \lambda_{k-1}\varphi_{k-1}y_{k-1} + \varphi_k y_k + \lambda_{[k-N,k]}(P^*)^{-1}\theta^* \end{aligned} \tag{21}$$

$$\hat{\theta}_{k|k} = P_{k|k}d_{k|k} \tag{22}$$

where $0 < \lambda_{[k-N,k]}$ is a scalar accumulated weight given by (23)

$$\begin{aligned} \lambda_{[k-N,k]} &= \lambda_{k-1}\cdots\lambda_{k-N+1}(1-\lambda_{k-N}) + \lambda_{k-1}\cdots\lambda_{k-N+2}(1-\lambda_{k-N+1}) \\ &+ \cdots + \lambda_{k-1}(1-\lambda_{k-2}) + (1-\lambda_{k-1}). \end{aligned} \tag{23}$$

The regularized exponential algorithm is thus modified by these assumptions (18), (19) into The Block Regularized Algorithm [12].

In the block regularized algorithm the alternative covariance matrix is added only every Nth iteration but in such a way as to achieve *exactly* the same results as the algorithm which adds it every step. Exactly the same parameter estimates will be provided, but they will be only available at every Nth iteration.

Figure 1 demonstrates the block regularized algorithm. For a clearer comparison, there is a trace of the parameter estimates of the original regularized exponential algorithm. Notice, that both algorithms start and end at precisely the same conditions. The block regularized algorithm reduces considerably the number of computations necessary to get to the final point.

Note that θ^*, P^* can change in the time instants $k, k+N, \ldots$ but must remain constant in the intermediate N steps. Usually $N = n$, where n is the number of estimated parameters.

Using the QR orthogonal rotation concept (15), the factorised block restricted algorithm can be rewritten into the final QR form, updating the upper triangular matrix $R_{k|k}$ and vector $r_{k|k}$.

The final QR block restricted algorithm (24) is a sequence of QR orthogonal rotations of the type (15), applied for the real data measurements. It is followed by the accumulated regularization of the type (16) performed on the boundaries of blocks, combined with the weight flushing (17) of the current parameter estimates.

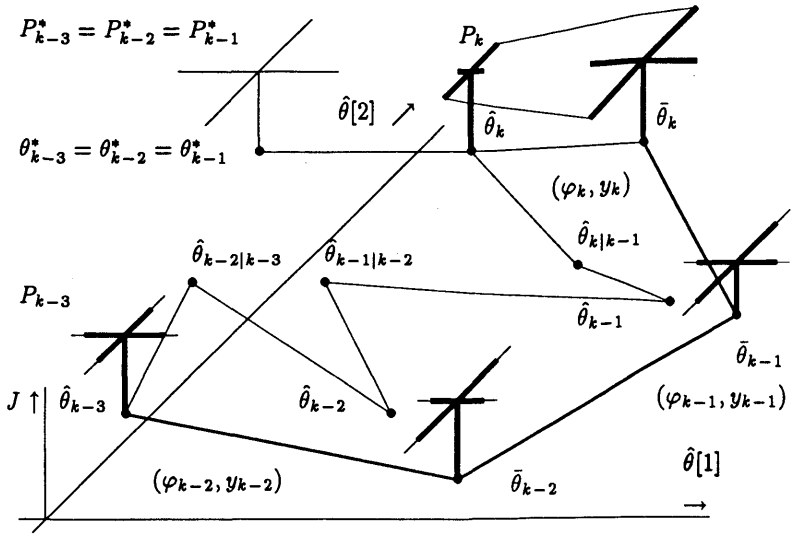

Fig. 1. Block Regularized Algorithm for time $k-3$ to k.

Block regularized algorithm. The algorithm is specified by the parameters: $R^* \geq 0; \lambda_{k-N}, \ldots, \lambda_{k-1}; 0 < \lambda_j \leq 1; N \geq n$.

$$\bar{R} = R_{k-N|k-N}\,;\ \bar{r} = r_{k-N|k-N} \qquad \text{(a)}$$

cycle for $j = N-1, N-2, \ldots, 0$:

$$\begin{bmatrix} \bar{R} & \bar{r} \\ 0 & * \end{bmatrix} := Q_{k-j-1} \begin{bmatrix} \sqrt{\lambda_{k-j-1}}\,\bar{R} & \sqrt{\lambda_{k-j-1}}\,\bar{r} \\ \varphi'_{k-j} & y_{k-j} \end{bmatrix} \qquad \text{(b)}$$

end of cycle for j

$$\bar{R}_k = \bar{R}\,;\ \bar{r}_k = \bar{r} \qquad \text{(c)}$$

$$\bar{\theta}_{k-N} = \bar{R}^{-1}_{k-N} \bar{r}_{k-N} \qquad \text{(d)}$$

$$\theta^*_{k-N} \equiv \bar{\theta}_{k-N} \qquad \text{(e)}$$

$$r^*_{k-N} = R^*_{k-N} \theta^*_{k-N} \qquad \text{(f)}$$

cycle for $i = n, n-1, \ldots, 1$:

$$\begin{bmatrix} \bar{R} & \bar{r} \\ 0 & * \end{bmatrix} := Q_{i,k} \begin{bmatrix} \bar{R} & \bar{r} \\ \sqrt{\lambda_{[k-N,k]}}\,R^*_{k-N}[i] & \sqrt{\lambda_{[k-N,k]}}\,r^*_{k-N}[i] \end{bmatrix} \qquad \text{(g)}$$

end of cycle for i

$$R_{k|k} = \bar{R}\ ;\ r_{k|k} = \bar{r}\ ;\ (\hat{\theta}_{k|k} = R^{-1}_{k|k} r_{k|k}). \qquad \text{(h)}$$

(24)

5. TRANSPUTER ARRAY DESCRIPTION

The transputer array implementing the algorithm (24) is plotted in Figure 2. The detailed cell description depends on the specification of the basic "rank-one" update at the cell level. See [21], [24] [25], or [28] for different options. In this paper the QR type of cells [24] is worked out in detail.

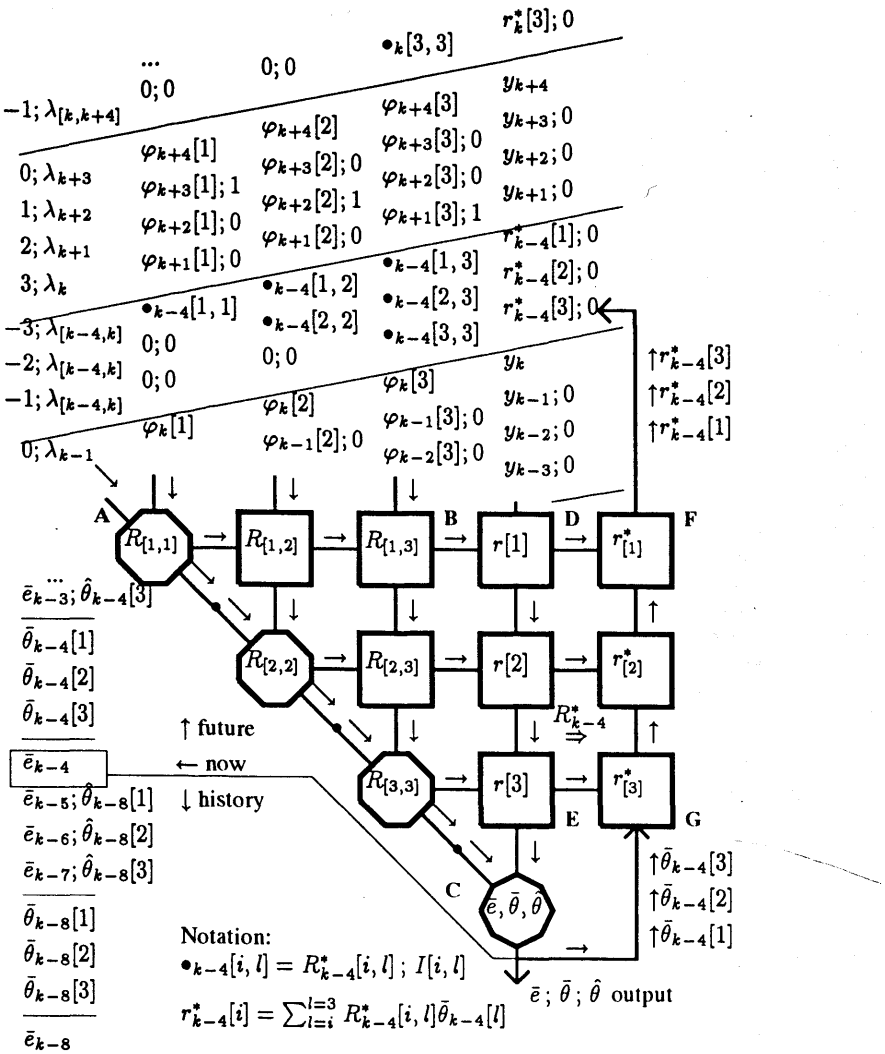

Fig. 2. Block Regularized QR Array Concept.

Let us discuss first, how the array selects the new position of the alternative estimate θ^* for each block of N data measurements in (24).

The basic areas marked in Figure 2 by ABC and BCDE store and update matrix R_k and r_k (respective $\bar{R}$ and $\bar{r}$) as in (24b) and (24g).

The automatic selection of θ^*_{k-N} for each block requires a sequence of additional computations (24d)–(24f). The equation (24d) computes the parameter estimate $\bar{\theta}_{k-N}$. In (24e) the selection of θ^*_{k-N} is made. Consequently the alternative quadratic form $J^*(\theta_k)$ for the block $[k-N, k]$ will have its minimum positioned in the point $\theta^*_{k-N} \equiv \bar{\theta}_{k-N} = \bar{R}^{-1}_{k-N}\bar{r}_{k-N}$.

Notice, that by using "old" variables $\bar{R}^{-1}_{k-N}$; $\bar{r}_{k-N}$ which have been available N time steps before, $\bar{\theta}_{k-N}$ can be computed off-line at this time.

Finally, once the alternative parameter θ^*_{k-N} and R^*_{k-N} for the block $[k-N, k]$ are chosen, the vector r^*_{k-N} as a matrix by vector product $R^*_{k-N}\theta^*_{k-N}$ is computed in (24f). The vector r^*_{k-N}, weighted by $\lambda_{[k-N,k]}$ (23), is finally "added" to $\bar{r}_k$ in (24g) to realize the accumulated regularization.

The block regularized algorithm (24) produces as a side effect the parameter estimates $\bar{\theta}_{k-N}$; $\bar{\theta}_k$ ... used for the automatic positioning of the alternative quadratic form $J^*(\theta_k)$. These parameters have to be distinguished from the parameter estimate $\hat{\theta}_{k-N|k-N}$; $\hat{\theta}_{k|k}$ ... which can be computed using the formulae (22h). In Figure 3 the difference between $\bar{\theta}_k$ and $\hat{\theta}_k$ is illustrated.

The advantage of selecting the alternative quadratic form $J^*(\theta_k)$ in the semi constant points $\bar{\theta}_{k-N}$; $\bar{\theta}_k$ .... is an efficient parallel implementation. The computation of $\bar{\theta}_{k-N}$ (24d) can be started in parallel with the accumulated regularization (24g) computation in the previous $[k-2N, k-N]$ block. In contrast, computation of $\hat{\theta}_{k-N|k-N}$ (24h) can start only after finishing of the accumulated regularization (24g) of the transformed variables in the previous block ie. after all the computations of the block $[k-2N, k-N]$ and therefore it would cause a decrease of the throughput of the final pipelined implementation.

The extended row of n cells, DEFG, performs the matrix by vector multiplication (24f). The matrix R^*_{k-N} is loaded from the north to ABC area. When loaded, R^* starts to move (with the element $R^*_{k-N}[n, n]$ first) through BCDE in the easterly direction, to realize (24f) in DEFG.

The vector θ^*_{k-N} moves (the item $\theta^*_{k-N}[n]$ first) to the array marked DEFG from the south to realize (24f). The resulting product $r^*_{k-N}[i]$ emerges from the northernly output of the DEFG array. It is directly loaded back to the north input of the array BCDE (synchronously with the rows $R^*_{k-N}[i]$ loaded to the array ABC) to realize the accumulated "addition" (24g).

Simultaneously, both arrays ABC and BCDE provide the parameter estimate extraction known as "weight flushing" [25, 28] (24d) by inserting (n, n) unit matrix and a column of zeros on the array data inputs (synchronously with the rows $[R^*_{k-N}[i], r^*_{k-N}[i]]$). The parameter extraction "filter" works with the "frozen" matrix $\bar{R}_{k-N}$ and vector $\bar{r}_{k-N}$, stored in the areas ABC and BCDE.

Systolic array characteristics

The concept of the systolic implementation of the regression parameter estimator is presented in Figure 2. Figure 3 presents the use of the input data buffering

and multiplexing, to provide approximately 50 %th throughput in compare with the systolic triangular RLS filter array.

The cells of the array work in two phases. These phases are distinguished in the cells by the sign of an index j propagated through the array. The value of the index gives to the cells the timing information within both phases.

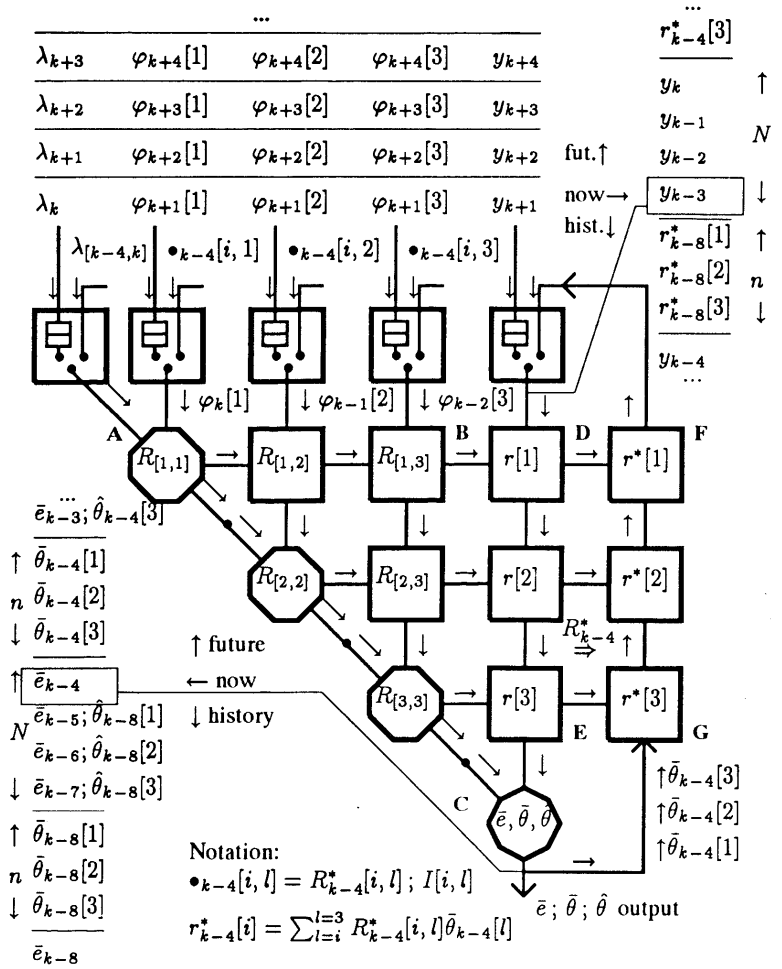

Fig. 3. Transputer implementation of Block Regularized QR Array.

In the first phase a block of $N \geq n$ data is processed (n is the number of estimated regression parameters) using the standard function of the RLS systolic information filter (N rank-one data updates).

In the second phase the processing of the input data temporary stops and the

array computes (in n steps, each with complexity comparable to a rank-one data update) these combined functions:

(1) The computation of the regression parameter estimate by "weight flushing". The parameters emerge as a serial stream of n items.

(2) The positioning of the minimum of the regularizing semi-constant quadratic form (the point θ^*) to the point of the recently flushed-out regression parameter estimate.

(3) The regularization, accumulated over last N data measurements.

The resulting parallel implementation of the estimator has several remarkable features:

- The transputer implementation is completely pipelined with processor utilization close to 100 %.
- The processes perform only the neighbor–to–neighbor one–directional communication.

6. CONCLUSIONS

The block regularized algorithm is described and its square root form derived, which allows parallel transputer implementation under the assumption, that the alternative regularizing cost function is constant within data blocks and adjusted to the new value on the boundaries of blocks only. This restriction (see Fig. 1) does not reduce significantly the effectiveness of the general regularized concept because the array implementation adjusts the minimum of the semi-constant alternative cost to follow the current regression parameter estimate.

It is then shown how it could be implemented on a systolic architecture. Figure 2 presents the principle, Figure 3 describes the final structure of a compacted transputer array, with the input data buffering and multiplexors.

The final block regularized transputer architecture (Fig. 3) is half as fast as the standard architecture of McWhirter [24] for the exponentially weighted recursive least squares without regularization. It should be noted again, that the processes perform nearest-neighbor, one-directional communication. There are no long global connections in Figure 3. Data only passes in one direction on any given link. The feedback associated with adjustment of the new position of the alternative cost function is completely compatible with the pipelined implementation.

The side effect of the parallel square root information filter implementation is that the regression parameter estimates are provided by the accumulated regularization process on the boundaries of the computational blocks. The direct use of these parameters can be found for example in the design of adaptive optimal controllers, or in the high resolution adaptive spectral analysis.

(Received November 10, 1993.)

REFERENCES

[1] G. J. Bierman: Factorisation Methods for Discrete Sequential Estimation. Academic Press, New York 1977.

[2] M. J. Flynn: Very high speed computing systems. IEEE Proc. *54* (1966), 1901–1909.

[3] F. M. F. Gaston and G. W. Irwin: The systolic approach to information Kalman filtering. Internat. J. Control *15* (1989), 1, 225–228.

[4] F. M. F. Gaston, G. W. Irwin and J. G. Mc Whirter: Systolic square root covariance Kalman filtering. J. VLSI Signal Processing *2* (1990), 37–49.

[5] W. M. Gentleman and H. T. Kung: Matrix triangularisation by systolic arrays. Proc. SPIE, Vol. 298, Real Time Signal Processing IV, 1981, pp. 19–26.

[6] T. Hägglund: The problem of forgetting old data in recursive estimation. In: Proceedings of the IFAC Workshop on Adaptive Systems in Control and Signal Processing, San Francisco, California 1983.

[7] C. A. R. Hoare: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs, N. J. 1985.

[8] K. Hwang and F. A. Briggs: Computer Architecture and Parallel Processing. McGraw Hill, New York 1985.

[9] J. Kadlec: A joint criterion for exponential directional and mixed parameter tracking. In: Preprints of the International Symposium on Adaptive Systems in Control and Signal Processing (ACASP'92) Conference, Grenoble 1992, pp. 687–692.

[10] J. Kadlec: A recursive Gramm–Schmidt identification with directional tracking of parameters. In: Preprints of the the 9-th IFAC/IFORS Symposium on Identification and System Parameter Estimation, Budapest 1991, pp. 1707–1712.

[11] J. Kadlec, F. M. F. Gaston and G. W. Irwin: Implementation of the regularized parameter estimator. In: Proceedings of the 1992 IEEE Workshop on VLSI and Signal Processing, Napa 1992, pp. 520–529.

[12] J. Kadlec, F. M. F. Gaston and G. W. Irwin: Parallel implementation of restricted parameter tracking. In: Mathematics in Signal Processing (J. McWhirter, ed.), Oxford Publishers Press., Oxford 1992, pp. 315–326.

[13] J. Kadlec, F. M. F. Gaston and G. W. Irwin: The block regularised parameter estimator and its parallel implementation. Automatica *31* (1995), 8, 1125–1136.

[14] J. Kadlec and N. N. Nakhaee: Alpha Bridge for MATLAB 4. In: World Transputer Congress WTC 95, UK 1995 (submitted).

[15] R. Kulhavý: Directional tracking of regression–type model parameters. In: Preprints of the 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing, Lund 1986, pp. 97–102.

[16] R. Kulhavý: Restricted exponential forgetting in real–time identification. Automatica *23* (1987), 589–600.

[17] R. Kulhavý and B. Zarrop: On a general concept of forgetting. Internat. J. Control *58* (19993), 4, 905–924.

[18] R. Kulhavý and M. Kárný: Tracking of slowly varying parameters by directional forgetting. In: Preprints 9th IFAC World Congress, Budapest 1984, pp. 178–183, 1984.

[19] H. T. Kung, C. E. Leiserson: Algorithms for VLSI processor arrays. In: Introduction to VLSI systems (C. Mead and L. Conway, eds.), Addison–Wesley, Reading, Mass. 1980.

[20] S. Y. Kung: VLSI Array Processors. Prentice Hall, Englewood Cliffs, N. J. 1988.

[21] F. D. Ling, D. Manolakis and J. G. Proakis: A recursive modified Gram-Schmidt algorithm for least squares estimation. IEEE Trans. Acoust. Speech Signal Process. *34* (1986), 829–835.

[22] L. Ljung and S. Gunnarsson: Adaptation and tracking in system identification – a survey. Automatica *26* (1990), 7–21.
[23] L. Ljung and T. Söderström: Theory and Practice of Recursive Identification. MIT Press, Cambridge, MA 1983.
[24] J. G. McWhirter: Recursive least squares minimisation using a systolic array. In: Proc. SPIE *431*, Real Time Signal Processing VI, pp. 105, 1983.
[25] J. G. McWhirter: Algorithmic engineering – an emerging discipline. SPIE Vol. 1152, Advanced Algorithms and Architectures for Signal Processing IV, pp. 2–15, 1989.
[26] J. E. Parkum, N. K. Poulsen and J. Holst: Recursive forgetting algorithms. Internat. J. Control *55* (1992), 109–128.
[27] V. Peterka: Bayesian approach to system identification. In: Trends and Progress in System Identification (P. Eykhoff, ed.), Pergamon Press, Eindhoven 1981, Chap. 8, pp. 239–304.
[28] T. J. Shepherd, J. G. McWhirter and J. E. Hudson: Parallel weight extraction from a systolic adaptive beamformer. In: Proc. IMA Internat. Conf. on Mathematics in Signal Processing, Warwick 1988, pp. 775–790.

Ing. Jiří Kadlec, CSc., Ústav teorie informace a automatizace AV ČR (Institute of Information Theory and Automation – Academy of Sciences of the Czech Republic), Pod vodárenskou věží 4, 182 08 Praha 8. Czech Republic.