Int'l. Journal of Adaptive Control and Signal Processing to appear 1/1995, in press
Special issue on Adaptive Algorithms for Higher Order Statistics

Adaptive Computation of Higher Order Moments and its Systolic Realization

Haris M. Stellakis Elias S. Manolakos*

Communications and Digital Signal Processing (CDSP)
Center for Research and Graduate Studies

Electrical and Computer Engineering Department
409 Dana Research Building
Northeastern University, Boston, MA 02115

*To whom all correspondence should be addressed, e-mail:elias@athina.cdsp.neu.edu, tel:(617)-373-3021, fax:(617)-373-4189. This work is partially supported by a Research Initiation Award from the National Science Foundation, number MIP-9309319

Keywords:

Higher Order Moments, time- and order-recursive estimation, parallel processing and pipelining, systolic arrays, algorithm-to-architectures unified synthesis methodology.

Abstract

In signal processing applications that require new estimates of the fourth and lower order moments every time a new data sample is received, it is necessary to design algorithms that adaptively update these terms. In addition, if real-time performance is necessary we should transform these algorithms so that their parallel processing and pipelining potential is exploited by a suitable multiprocessor architecture. In this paper we present a time- and order-recursive estimation procedure for updating all moment lag estimates (up to the fourth order) in one of their primary region of support, using the previous estimates and the newly arrived data sample in real-time. Then we systematically transform the moments updating algorithm onto an architecture that is suitable for VLSI implementation. As a special case a linear array computing the diagonal 1-D slice of the higher order moments is also synthesized. Under the algorithm-to-architectures transformation the time- and order- recursive characteristics of the adaptive procedure translate to a scalable architecture whose processing elements consist of pipelined stages of simple multiply-accumulate units. The unified top-down synthesis of the architectures facilitates formal verification of correctness at the behavioral level, identification of trade-offs, and the easy introduction of modifications, should the design objectives change during the design phase.

1 Introduction

Higher Order Statistics (HOS), known also as higher order *moments*, *cumulants*, along with their frequency domain counterparts, the *polyspectra*, have been added during the recent years in the arsenal of commonly used signal processing tools. There are many situations arising in time series processing that cannot be handled adequately by second order methods (power spectrum or autocorrelation based). For example, HOS techniques are often employed in order to to extract information due to deviation from Gaussianity, to estimate the phase of non-Gaussian signals, or detect and characterize the nonlinear properties of mechanisms generating time series via phase relations of their harmonic components [1, 2]. Due to these and other important capabilities, there has been an increasing interest in introducing HOS based techniques in diverse application domains including - but not limited to - digital communications, biomedical, sonar, radar, geophysical, speech and image processing [3].

The computation of Higher Order Moments (HOMs) in particular, is a necessary step towards the estimation of other statistics such as the cumulants [3] and the Wigner spectra [4]. In applications where it is crucial to update the HOS estimates frequently using previous estimates and the newly available data samples, adaptive computation procedures should be employed. However the consistent estimation of HOS usually requires large data blocks and algorithms with high computational complexity. To support real-time applications using HOS it becomes necessary to exploit the parallel processing and pipelining potential of the algorithms by transforming them to regular and modular multiprocessor structures that can be implemented using VLSI technology.

During the last few years considerable effort has been placed in designing application specific systems to speed-up HOS computations using VLSI architectures [5] or other emerging technologies [6, 7]. Our research group has also contributed significantly to this effort. In [8] we introduced a two-dimensional array architecture that can compute the higher order moments and the Bispectrum based on the conventional "indirect" estimation algorithm [9]. This batch-type approach segments the incoming data in non-overlapping blocks of M samples; HOS are estimated for each block by the array and the average of block estimates is computed by the host. The approach has been used for the computation of all moment lags up to the fourth order in [10], as well as for the computation of the fourth order cumulants (via the moments) in [11]. An integrated architecture providing in real-time both the moments and cumulants, up to the fourth order, has been described in [12]. For all the details on the systematic VLSI array synthesis methodology used to derive these designs and on the particular characteristics of the resulting architectures the interested reader is referred to [13].

In this paper we propose a *time-and-order recursive* computation procedure using a sliding window of size M, that allows for the continuous updating of moment estimates at a minimum computational cost as soon as a new data sample becomes available. Its data dependences structure is analyzed and the adaptive algorithm is transformed *systematically* to an equivalent one with localized dependences only. The resulting *Locally Recursive Algorithm* (LRA) is then mapped to a

systolic array realization providing estimates for all moment lags, up to the fourth order, in one of their primary region of support. The design is not optimized for a particular order, but is rather meant to serve as a general purpose moments updating engine.

The systematic architecture synthesis allows the formal verification of the candidate array solutions and their relative evaluation. Moreover it provides a mechanism for deriving architectures for the given problem *under constraints* imposed either by the nature of the problem itself or by the implementation technology. So the linear array synthesized in this paper is not a unique solution and can be easily modified, should the design objectives change. At the level of granularity presented the array is more suitable for a VLSI implementation; however larger granularity linear arrays using off-the-self microprocessors (DSP, RISC, Transputers) can be obtained with minor effort following the same design methodology by trading time (reducing the data rates) for less hardware per processor.

The rest of the paper is organized as follows: In section 2 we develop the time- and order-recursive procedure for the estimation of HOMs. In section 3 we provide the necessary background on the theory of algorithm-to-architectures transformation methodologies, highlight the limitations of the traditional space-time mapping approaches and demonstrate how the *unified* synthesis methodology can be applied to derive a VLSI array implementing the adaptive algorithm in real-time. The operation of the synthesized array is described in section 4. As a special case the design of a VLSI array computing the slice of the higher order moments when all lags are equal is discussed in section 5. Finally the paper is summarized and directions of current investigation are highlighted in section 6.

Before we proceed, let us clarify the notation to be used. We denote the $n-$ dimensional set of integers by $\mathcal{Z}^n$, the matrices/column vectors by bold and capital/lower case letters respectively, the matrix transposition sign by "^{T}", index points in $\mathcal{Z}^n$ by vectors "$\vec{\ }$", and scalar quantities by lower case letters. In addition let the algorithm and processor index space have dimensionality n_a and n_p respectively, and $\mathbf{e}_i^{(n)} \in \mathcal{Z}^{n\times 1}$, be the i-th canonical basis vectors (i-th column of identity matrix $\mathbf{I}_n$). Finally, for simplicity let $\mathbf{e}_i$ and $\mathbf{u}_i$ (without a superscript) denote the canonical basis vectors in the algorithm and processor space respectively.

2 Time and order recursive computation of Higher Order Moments

We recall [3] that the kth order moment estimates $\{\widehat{m}_k\}$, based on M samples of a real, one-dimensional process, can be obtained at one of the primary domains of support $\mathcal{R}_0$, as follows:

$$\widehat{m}_k(i_1, i_2, ..., i_{k-1}) = \frac{1}{M} \sum_{i=0}^{M-1-i_1} x_i \; x_{i+i_1} \; \cdots \; x_{i+i_{k-1}} \tag{1}$$

where $\mathcal{R}_0 = \{(i_1, ..., i_{k-1}) : 0 \leq i_{k-1} \leq ... \leq i_2 \leq i_1 \leq M-1\}$.

If we want to compute the $\{\widehat{m}_k\}$ every time a new data sample becomes available we should develop a time-recursive algorithm. If in addition we want to estimate the cumulants as well, via the moments, then order-recursive algorithms will be preferable in order to reduce the computational complexity. The following proposition addresses the recursive updating of *all* higher order moments based on their estimates at the previous time instant as well as the current and the past M data samples.

Proposition 1 *Given initial estimates* $\{\widehat{m}_k\}$ *as defined by (1), new estimates of* all *the moments up to desired order* k *can be computed in the non-redundant region* $\mathcal{R}_0$ *using time- and order- recursive expressions. If the data window is of fixed size* M*, [growing], the moment updating formulas are given by equations (2), [(3)] respectively, at any time instant* $n \geq M$*.*

$$\widehat{m}_l(i_1, ..., i_{l-1}; n) = \widehat{m}_l(i_1, ..., i_{l-1}; n-1) + \tag{2}$$
$$\frac{1}{M}\left(p_{l-1}(i_1, ..., i_{l-2}; n) \cdot x_{n-i_1+i_{l-1}} - q_{l-1}(i_1, ..., i_{l-2}; n) \cdot x_{n-M+i_{l-1}}\right)$$

$$\widehat{m}_l(i_1, ..., i_{l-1}; n) = \tag{3}$$
$$\frac{1}{n+1}\left(n \cdot \widehat{m}_l(i_1, ..., i_{l-1}; n-1) + \ p_{l-1}(i_1, ..., i_{l-2}; n) \cdot x_{n-i_1+i_{l-1}}\right)$$

where by definition $i_0 \equiv 0$ *and the product variables are* $p_l(i_1, ..., i_{l-1}; \tau) \equiv \prod_{j=0}^{l-1} x_{\tau-i_1+i_j}$ *and* $q_l(i_1, ..., i_{l-1}; \tau) \equiv \prod_{j=0}^{l-1} x_{\tau-M+i_j}$ *with* $p_0(\tau) = q_0(\tau) = 1$.

Proof:

At every time instant $n \geq M$ the data window of fixed size M will contain the samples $\{x_{n-M+1}, ..., x_{n-1}, x_n\}$. Then from (1) and for every $(i_1, ..., i_{k-1}) \in \mathcal{R}_0$ we get:

$$\widehat{m}_k(i_1, ..., i_{k-1}; n) = \frac{1}{M} \sum_{i=n-M+1}^{n-i_1} \prod_{l=1}^{k} x_{i+i_{l-1}} \tag{4}$$
$$= \frac{1}{M}\left(\sum_{i=n-M}^{n-i_1-1} \prod_{l=1}^{k} x_{i+i_{l-1}} + \prod_{l=1}^{k} x_{n-i_1+i_{l-1}} - \prod_{l=1}^{k} x_{n-M+i_{l-1}}\right)$$
$$= \widehat{m}_k(i_1, ..., i_{k-1}; n-1) + \frac{1}{M}\left(\prod_{l=1}^{k} x_{n-i_1+i_{l-1}} - \prod_{l=1}^{k} x_{n-M+i_{l-1}}\right)$$

where $i_0 \equiv 0$. Let us now define two k-th order product terms at time instant n as:

$$p_k(i_1, ..., i_{k-1}; n) \equiv \prod_{l=1}^{k} x_{n-i_1+i_{l-1}} = p_{k-1}(i_1, ..., i_{k-2}; n) \cdot x_{n-i_1+i_{k-1}} \tag{5}$$

$$q_k(i_1, ..., i_{k-1}; n) \equiv \prod_{l=1}^{k} x_{n-M+i_{l-1}} = q_{k-1}(i_1, ..., i_{k-2}; n) \cdot x_{n-M+i_{k-1}} \tag{6}$$

Then (4) becomes:

$$\begin{aligned}\widehat{m}_k(i_1,...,i_{k-1};n) &= \widehat{m}_k(i_1,...,i_{k-1};n-1)+\frac{1}{M}\left(p_k(i_1,...,i_{k-1};n)-q_k(i_1,...,i_{k-1};n)\right)\\ &= \widehat{m}_k(i_1,...,i_{k-1};n-1)+\\ &\quad +\frac{1}{M}\left(p_{k-1}(i_1,...,i_{k-2};n)\;x_{n-i_1+i_{k-1}}-q_{k-1}(i_1,...,i_{k-2};n)\;x_{n-M+i_{k-1}}\right)\end{aligned}$$

The proof of (3) is easier and can be reached by following similar steps. In this case since the data window is growing we do not need to subtract the second product term. □

In Figure 1 we provide a nested loop algorithm that provides *all* moments, up to a desired order k in a forward order- and time- recursive fashion based on (2). Notice that we made all variable instances p_l and q_l, for different values of l, to have common region of support in order to avoid introducing conditional statements. An algorithm of similar complexity can be formulated for the recursive expression (3), with the difference that only one product term should be generated at every time iteration. On the other hand, a multiplication by n and a division by $n+1$ respectively is needed, whereas in (2) we may pre-scale by M (fixed and independent of time n) all the moment terms and avoid division.

In the next section we derive a linear array of processors implementable in VLSI for the efficient estimation of *all* the higher order moments, up to the 4th order, since these are most commonly needed in practice. However our approach is order-independent and can be extended to any $k > 4$, should that become necessary.

3 Unified Array synthesis methodology

In general, array synthesis can be achieved via several behavior preserving transformations applied at different levels of abstraction. First the nested loop algorithm is transformed into an equivalent *Locally Recursive Algorithm* (LRA). Then the array architecture is generated as a result of appropriate linear *space-time transformations* $(\mathbf{S},\mathbf{T})$ directly applied to the LRA [14, 15]. The synthesis has been traditionally treated as a two-phase process, namely the *localization* of the algorithm (selection of the LRA) followed by the *mapping* of the LRA to an array structure. We emphasize here that in both phases, a specific starting point (nested-loop algorithm/LRA) may lead to *numerous* solutions (equivalent LRA's/ASAP's) respectively. If the two phases are dealt with independently, it is possible that either *no* architecture can be found that meets all the design specifications, or a "good" choice for $\mathbf{S}$ and $\mathbf{T}$ has led to a suboptimal array, because of the inappropriate choice of the LRA in the first place.

In this work we eliminate the aforementioned drawback by applying the *unified array synthesis* methodology we introduced in [13]. The key idea behind this methodology is that the localization

and mapping phases are *interleaved*. In particular, instead of *choosing* one LRA out of the many possible, we rather gradually *construct* an appropriate LRA that meets architectural constraints and design specifications arising from the mapping phase. In the sequel, we explain the steps we follow towards the systematic synthesis for the architecture computing adaptively the higher order moments.

1. DATA DEPENDENCE ANALYSIS: The index space of the nested-loop algorithm is known at compile-time to be $\mathcal{I} = \{ (i_1, i_2, i_3, n, l) : \ 0 \le i_3 \le i_2 \le i_1 \le M-1, \ n \ge M, \ 1 \le l \le 4 \} \subset \mathcal{Z}^5$. The first three indices correspond to the lags i_1, i_2, and i_3 of the fourth order moments restricted in the primary domain of support $\mathcal{R}_0$. The fourth index n corresponds to time iteration, and the last index l to the order of the moments under investigation. Clearly, the range of n makes $\mathcal{I}$ unbounded. However, every hyperplane (for a given n), is a convex polyhedron in $\mathcal{Z}^4$. The variables involved are m, p, q and x. Their instances (encountered at the right hand side of the assignment statements) are $m_l(i_1, i_2, i_3; n-1)$, $p_{l-1}(i_1, i_2, i_3; n)$, $q_{l-1}(i_1, i_2, i_3; n)$ and $x_{n-i_1+i_{l-1}}$, $x_{n-M+i_{l-1}}$, for $l = 1, 2, 3, 4$ respectively.

To perform a systematic localization we need to identify *all* possible dependences associated with every variable involved. This can be achieved by finding the *indexing matrix* of every variable instance, as explained below.

Let us assume that a variable w has an instance $w(f[\vec{v}])$ defined by an *affine* transformation $f[\vec{v}] \equiv \mathbf{A}_w \cdot \vec{v} + \mathbf{b}_w$, where $\vec{v} \in \mathcal{I}$ is a point in the index space of the algorithm. Then $\mathbf{A}_w$ is called the indexing matrix of $w(f[\vec{v}])$, and if $\mathbf{A}_w$ is row rank deficient, variable w is called a *broadcast*, meaning that its value is needed to more than one indexed computations of the algorithm. Moreover, the valid contours over which the value of w can be propagated are restricted within a subspace of the right null space of $\mathbf{A}_w$, namely $\mathcal{N}(\mathbf{A}_w)$ [16, 17]. Clearly, there may exist many possible contours of propagation to be considered for the localization of a broadcast variable; the number is directly related to the row rank of the indexing matrix.

In our case, we identify the instances $x_{n-i_1+i_{l-1}}$ and $x_{n-M+i_{l-1}}$ with w_l and v_l, $l = 1, 2, 3, 4$ respectively and consider the index space $\mathcal{I}$ as their common domain of support. Then the indexing matrices for all instances are: $\mathbf{A}_p = [\mathbf{e}_1 \mid \mathbf{e}_2 \mid \mathbf{e}_3 \mid \mathbf{e}_4 \mid \mathbf{e}_5]^T$; $\mathbf{A}_q = \mathbf{I}_5$, $\mathbf{A}_{w_1} = [-\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{w_2} = [\mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{w_3} = [-\mathbf{e}_1^{(4)} + \mathbf{e}_2^{(4)} + \mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{w_4} = [-\mathbf{e}_1^{(4)} + \mathbf{e}_3^{(4)} + \mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{v_1} = [\mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{v_2} = [\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}]^T$, $\mathbf{A}_{v_3} = [\mathbf{e}_2^{(4)} + \mathbf{e}_4^{(4)}]^T$ and $\mathbf{A}_{v_4} = [\mathbf{e}_3^{(4)} + \mathbf{e}_4^{(4)}]^T$. In addition the indexing matrix for the moment estimates $m_l(i_1, i_2, i_3; n-1)$ is $\mathbf{A}_m = \mathbf{I}_5$.

Since $\mathbf{A}_m$, $\mathbf{A}_p$ and $\mathbf{A}_q$ are full rank, variables m, p, q are not broadcasts. So there is a *true* data dependence associated with each one of them, between their instances on the right and left hand side of the assignment statements. These true data dependences are $\mathbf{d}_m = \mathbf{e}_4$, $\mathbf{d}_p = \mathbf{d}_q = \mathbf{e_5}$. On the other hand, w_l and v_l are broadcasts that can be localized by introducing dependences with directions confined within the null space of their indexing matrices, i.e.,

$$\mathbf{d}_{w_1} \quad = \quad a_{1,1}(\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}) + a_{1,2}\mathbf{e}_2^{(4)} + a_{1,3}\mathbf{e}_3^{(4)}$$

$$
\begin{aligned}
\mathbf{d}_{w_2} &= a_{2,1}\mathbf{e}_1^{(4)} + a_{2,2}\mathbf{e}_2^{(4)} + a_{2,3}\mathbf{e}_3^{(4)} \\
\mathbf{d}_{w_3} &= a_{3,1}(\mathbf{e}_1^{(4)} + \mathbf{e}_2^{(4)}) + a_{3,2}(\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}) + a_{3,3}\mathbf{e}_3^{(4)} \\
\mathbf{d}_{w_4} &= a_{4,1}(\mathbf{e}_1^{(4)} + \mathbf{e}_3^{(4)}) + a_{4,2}(\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}) + a_{4,3}\mathbf{e}_2^{(4)} \\
\mathbf{d}_{v_1} &= b_{1,1}\mathbf{e}_1^{(4)} + b_{1,2}\mathbf{e}_2^{(4)} + b_{1,3}\mathbf{e}_3^{(4)} \\
\mathbf{d}_{v_2} &= b_{2,1}(-\mathbf{e}_1^{(4)} + \mathbf{e}_4^{(4)}) + b_{2,2}\mathbf{e}_2^{(4)} + b_{2,3}\mathbf{e}_3^{(4)} \\
\mathbf{d}_{v_3} &= b_{3,1}\mathbf{e}_1^{(4)} + b_{3,2}(-\mathbf{e}_2^{(4)} + \mathbf{e}_4^{(4)}) + b_{3,3}\mathbf{e}_3^{(4)} \\
\mathbf{d}_{v_4} &= b_{4,1}\mathbf{e}_1^{(4)} + b_{4,2}\mathbf{e}_2^{(4)} + b_{4,3}(-\mathbf{e}_3^{(4)} + \mathbf{e}_4^{(4)})
\end{aligned} \tag{7}
$$

where $a_{l,j}$ and $b_{l,j}$, $l = 1,2,3,4$, $j = 1,2,3$ are integers to be determined.

2. Choosing The Space Transformation $\mathbf{S}$: The linear space transformation allocates (places) the algorithm's index points to Processing Elements (PEs). It may be represented by a matrix $\mathbf{S} \in \mathcal{Z}^{n_p \times n_a}$ with full row rank [15] that maps the n_a- dimensional algorithm index space onto the n_p- dimensional processor space, i.e., $\mathbf{S}: \ \mathcal{Z}^{n_a} \supset \mathcal{I} \longmapsto \mathcal{P} \subset \mathcal{Z}^{n_p}$. Under $\mathbf{S}$ an index point (Dependence Graph node [14]) $\vec{v} \in \mathcal{I}$ will be transformed to the PE (or Signal Flow Graph node [14]) $\mathbf{S} \cdot \vec{v} \in \mathcal{P}$ and a dependence vector $\mathbf{d}$ onto the communication link $\mathbf{c} = \mathbf{S} \cdot \mathbf{d}$.

The choice of permissible and optimum $\mathbf{S}$ is based on architecture design considerations and application imposed constraints. Current VLSI technology mandates that the target architecture should be at most two-dimensional with input/output operations occurring at the architecture boundaries and obviously should have some finite size.

The last constraint imposes that we "project" along the time axis n in order to obtain a *bounded* processor space (finite number of processors). Therefore, since $\mathbf{e}_4$ will not in the bases of $\mathcal{P}$ and $\mathbf{d}_m = \mathbf{e}_4$ is a true dependence, the updated moment estimates will be distributed among all the processors of the architecture. The VLSI constraints impose that only one of the canonical row vectors $\mathbf{e}_1^T$, $\mathbf{e}_2^T$, $\mathbf{e}_3^T$ (or their linear combination) constitute a row of matrix $\mathbf{S}$ (basis vector for $\mathcal{P}$). Among different choices, we selected $\mathbf{e}_1^T$. Finally, we want to fully exploit all the available parallelism and pipelining potential of the algorithm. Therefore we should not perform unnecessary projections since these introduce sequentiality. So the selected space transformation is:

$$
\mathbf{S} = [\mathbf{e}_1, \mathbf{e}_5]^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}
$$

This choice of $\mathbf{S}$ results to a two-dimensional processor space $\mathcal{P} \equiv \{(i_1, l): \ 0 \le i_1 \le M-1, \ 1 \le l \le 4\}$. Since the second dimension l of $\mathcal{P}$ is independent of the problem size M (pseudo-dimension), the architecture can be thought of as a linear array of M PE's communicating along i_1. Since l corresponds to the order index, by including $\mathbf{e}_5^T$ in the rows of $\mathbf{S}$ (i.e. by not projecting along l), we translate the order recursive generation of products to a pipelined internal structure for the processors of the linear array, where each pipeline stage l computes the corresponding l-th order moment terms, $l = 1,2,3,4$.

3. Constructing an Appropriate Locally Recursive Algorithm: This step corresponds to fixing the dependence directions associated with every broadcast variable, or equivalently choosing the values of all integer coefficients $\{a\}$ and $\{b\}$ of (7). If we want to have near-neighbor interconnections only it is sufficient to limit to unity the lengths of every communication link, or equivalently impose $a_{l,j} = \pm 1,\ b_{l,j} = \pm 1,\ \forall\, l = 1,2,3,4,\ \forall\, j = 1,2,3$.

It is meritorious to find a minimum number of linearly independent vectors whose linear combinations with non-negative coefficients correspond to valid directions of propagation for all the broadcast variables (see equations (7)). In this way we reduce all critical paths of the algorithm, or equivalently minimize the number of precedence constraints on the choice of a permissible schedule $\mathbf{T}$, thus increasing the probability of finding a minimum latency schedule [13]. In addition we also minimize the number of distinct communication links in the array.

Based on this criterion and using equations (7) we fix the direction of dependences for all broadcast variables. In particular, we choose $\mathbf{d}_{w_2} = \mathbf{d}_{v_1} = \mathbf{d}_{v_3} = -\mathbf{e}_1^{(4)}$. This direction (along "decreasing i_1") yields an input hyperplane at all index points with $i_1 = M-1$, which is mapped under $\mathbf{S}$ onto a single Processing Element, namely PE_{M-1}. Notice that the opposite direction $\mathbf{e}_1^{(4)}$ (i.e., along "increasing i_1") yields the input hyperplane $i_1 = i_2$, which is mapped under $\mathbf{S}$ onto M different PEs of the linear array.

Then, we observe that the direction of propagation for the variable w_3 can be $\pm(\mathbf{e}_1^{(4)} + \mathbf{e}_2^{(4)})$. Given that $-\mathbf{e}_1^{(4)}$ is already selected, we choose the dependences $\mathbf{d}_{w_1} = \mathbf{d}_{w_4} = \mathbf{d}_{v_2} = \mathbf{d}_{v_4} = -\mathbf{e}_2^{(4)}$ for the rest of the broadcast variables. In this way, we can make $\mathbf{d}_{w_3} = -\mathbf{e}_1^{(4)} - \mathbf{e}_2^{(4)}$ to be a linear combination of two other dependences with non-negative integer coefficients. The resulting LRA is shown in Figure 2, where the "transmittal" [14] variables w, v embody all the variables $\{w_l,\ v_l,$ for $l = 1,2,3,4\}$ respectively.

4. Choosing the Time Transformation $\mathbf{T}$: The linear time transformation (schedule) $\mathbf{T} \in \mathcal{Z}^{1 \times n_a}$ assigns to every computation, $\vec{v}$, an integer $\mathbf{T} \cdot \vec{v}$ corresponding to the schedule time instant of its execution, i.e. $\mathbf{T} : \mathcal{Z}^{n_a} \longmapsto \mathcal{Z}$. The chosen schedule has to be *permissible* and satisfy certain optimality criteria, such as achieving *minimum latency* or *maximum PE utilization*. To be permissible, a systolic schedule must satisfy both the *precedence* and *space-time compatibility* constraints [18, 15]. The former ensure that if there is a dependence from a Dependence Graph (DG) node $\vec{v}_1$ to node $\vec{v}_2$, then $\vec{v}_2$ will be executed at least one schedule time instant after $\vec{v}_1$. The latter constraints guarantee that there will be no data *collisions* in the array, i.e., two distinct points in the algorithm's index space allocated to the same PE will not be scheduled for execution at the same time. For linear transformations $\mathbf{S}$ and $\mathbf{T}$, both sets of constraints can be expressed analytically as:

Precedence Constraints (PC's): For every data dependence $\mathbf{d}$ it should be true that $\mathbf{T} \cdot \mathbf{d} \geq 1$

Space-Time Compatibility Constraints (STC's): Let $\vec{v}_1$, $\vec{v}_2$ be two arbitrary and distinct

index points allocated to processors $\mathbf{S}\cdot\vec{v}_1$, $\mathbf{S}\cdot\vec{v}_2$ at time instants $\mathbf{T}\cdot\vec{v}_1$, $\mathbf{T}\cdot\vec{v}_2$ respectively and $\vec{\delta}\equiv\vec{v}_2-\vec{v}_1$; it should be true that $\begin{bmatrix}\mathbf{S}\\ \mathbf{T}\end{bmatrix}\cdot\vec{\delta}\neq\mathbf{0}_{n_p+1},\ \forall\vec{\delta}$

The former constraints translate to:

$$\mathbf{T}\cdot[-\mathbf{e}_1,-\mathbf{e}_2,\mathbf{e}_4,\mathbf{e}_5]\geq[1,1,1,1]\Rightarrow t_1,t_2\leq-1 \text{ and } t_4,t_5\geq 1 \tag{9}$$

The latter translate to $\mathbf{S}\cdot\vec{\delta}=0\Rightarrow\mathbf{T}\cdot\delta\neq 0$ or equivalently

$$t_2\delta_2+t_3\delta_3+t_4\delta_4\geq 1,\quad\forall\delta \tag{10}$$

where $\vec{\delta}\equiv[\delta_1,\delta_2,\delta_3,\delta_4,\delta_5]^T$ represents the difference between *any* two distinct points within the index space $\mathcal{I}$. We find that $\mathbf{T}=[-1,-1,M,M^2,1]$ is indeed permissible. We can easily verify that this $\mathbf{T}$ is indeed conflict free since for $\mathbf{S}\cdot\vec{\delta}=0\Rightarrow\delta_1=\delta_5=0$, the Diofantine equation $\mathbf{T}\cdot\vec{\delta}=0\Rightarrow M\delta_3+M^2\delta_4=-\delta_2$ does not have an integer solution, because $gcd(M,M^2)=M$ cannot divide δ_2, since $|\delta_2|\leq M-1$ for the given index space $\mathcal{I}$. From this analysis, we conclude that any choice of $\{t_2,t_3,t_4\}=\{\pm1,\ \pm M,\ \pm M^2\}$ would yield a conflict free schedule. We select $t_2=-1$ since (a) it satisfies the precedence constraints on the selected dependence vectors, and (b) the resulting delays on the corresponding communication links are problem size independent. By selecting $t_3=M$ the results are generated in increasing order with respect to lag i_3. Notice that we could not achieve the same with respect to the other two lags i_1 and i_2 because this would require $t_1,t_2\geq 1$ that contradicts the precedence constraints (9). The interpretation of $t_2=-1$, $t_3=M$ is that index i_2, decreases by one every schedule time instant whereas index i_3 increases by one every M time instants when we keep fixed the rest of the indexes.

Regarding the fourth element of the schedule vector, we let $t_4=M^2$ take the largest among all other permissible values because we want to have all moments terms updated before a new sample becomes available. Finally we let $t_5=1$ since this value is permissible and yields the smallest number of delays for the dependences $\mathbf{d}_p$ and $\mathbf{d}_q$ as it will become more clear in the next section.

4 Array architecture

The resulting architecture is a linear array processor consisting of M PEs each having a four-stage pipelined structure, as shown in Figure 3 for the case $M=4$. Application of the space-time transformations to the dependences of the LRA yields the communication links of the array and the associated delays. For example, when $l=1$ the communication links become $\mathbf{c}_{w_1}\equiv\mathbf{S}\cdot\mathbf{d}_{w_1}=\mathbf{0}_2$ (self-loop) and $\mathbf{c}_{v_1}\equiv\mathbf{S}\cdot\mathbf{d}_{v_1}=-\mathbf{u}_1$ (along "decreasing i_1") with delays $\mathbf{T}\cdot\mathbf{d}_{w_1}=1$ and $\mathbf{T}\cdot\mathbf{d}_{v_1}=1$ respectively. Similarly we find: $\mathbf{c}_{w_2}=\mathbf{c}_{w_3}=\mathbf{c}_{v_3}=-\mathbf{u}_1$ and $\mathbf{c}_{v_2}=\mathbf{c}_{w_4}=\mathbf{c}_{v_4}=\mathbf{0}_2$ whereas $\mathbf{c}_p=\mathbf{c}_q=\mathbf{u}_2$. The associated delays are:

$$\mathbf{T}\cdot[\mathbf{d}_{w_2},\ \mathbf{d}_{v_2},\ \mathbf{d}_{w_3},\ \mathbf{d}_{v_3},\ \mathbf{d}_{w_4},\ \mathbf{d}_{v_4},\ \mathbf{d}_p\ \mathbf{d}_q]=[1,\ 1,\ 2,\ 1,\ 1,\ 1,\ 1,\ 1]$$

Notice that the systematic construction of the LRA yielded an array with *local memory per PE independent of the problem size* since all dependence vectors depend only on $\mathbf{e}_1$ and $\mathbf{e}_2$ whereas the schedule elements t_1 and t_2 are independent of M.

Every PE consists of four pipelined Multiply-ACcumulate (MAC) stages. Moment terms $m_l(i_1, i_2, i_3)$ become available from PE_{i_1}, MAC stage l. Clearly the average (m_1) becomes available from the first stage of the left most PE_0. At iteration (sampling period) $t_s = n$, the current sample x_n and only M previous samples (i.e., $x_{n-1}, ..., x_{n-M}$) are necessary as input data for the correct operation of the architecture. Input occurs either at specific stages of all PEs, or at the rightmost PE_{M-1} depending on the input hyperplane of the corresponding variable. Simple M- size First-In-First-Out (FIFO) queues or buffers are sufficient to achieve proper input distribution, as it can be seen in Figures 4 and 5. At every MAC(i_1, l) stage the product terms p_l, q_l are produced and then added/subtracted respectively from estimates of m_l at iteration $n-1$ to generate the new estimates m_l at sampling time $t_s = n$. Then, the products p_l, q_l are fed to the next stage to produce the $(l+1)$- order terms. Figures 4 and 5 show the internal pipelined structure of PE_{M-1} and PE_{M-2} respectively and the input/output skews when x_n becomes available ($t_s = n$). All other PEs have the same structure.

In the sequel we elaborate on the array operation by presenting the recursive computation of some product and moment terms from the two rightmost PEs. We make use of two timing variables the *schedule* time t, and *sampling* time t_s. The former denotes when a new data token may become available to one of the input lines. The period of the schedule clock corresponds roughly to the time needed for a MAC computation. The sampling instant denotes the time when every new sample becomes available from the host. The sampling period is equal to M^2 schedule periods; this is so because in the worst case, M^2 moment terms (of fourth order) have to be updated on every PE within a sampling period and before the new data sample becomes available.

When sample x_n becomes available (at sampling time instant $t_s = n$) tokens x_{n-M+1} and x_{n-M} are fed as input into stage $l = 1$ of PE_{M-1} via input lines $\mathbf{c}_{w_1}$ and $\mathbf{c}_{v_1}$ and loaded to communication links p_1 and q_1 respectively (see Fig. 4). This schedule time instant corresponds to the execution of the index point $\vec{v}_1 \equiv [M-1, M-1, 0, n, 1]^T$, that happens at $t_1 \equiv \mathbf{T} \cdot \vec{v}_1 = nM^2 - 2M + 3$. At subsequent schedule time instants, the same token x_{n-M+1} is either reused (dash lines in the input line $\mathbf{c}_{w_1}$) or provided as input again. This is a direct consequence of the space-time mapping of the input hyperplane for variable w_1 ($l = 1$). Recall that for w_1 input occurs when $i_1 = i_2$ (see Fig. 2), or every other M schedule time instants under $\mathbf{T} = [-1, -1, M, M^2, 1]$. On the other hand token x_{n-M} should be provided into input line $\mathbf{c}_{v_1}$ at every schedule time instant. Again this is due to the fact that input for v_1 happens when $i_1 = M-1$ (see Fig. 2). Dashes at line $\mathbf{c}_{v_1}$ denote schedule periods where no useful operation is performed and this is so because of the hyper-pyramid like shape of the index space for the lags in the non-redundant region $\mathcal{R}_0$ ($i_3 \leq i_2 \leq i_1$). For instance, the eighth input token ($2M$-th token in general) in line $\mathbf{c}_{v_1}$ is shown as a dash since it corresponds to the execution of index point $\vec{v} \equiv [M-1, M-2, M-1, n, 1]$ which does not belong in the index space ($i_3 > i_2$).

At the next schedule time, t_1+1, the two most recent samples x_n, x_{n-1} are fed to stage $l=2$ of PE_{M-1} via input lines $\mathbf{c}_{w_2}$ and $\mathbf{c}_{v_2}$ respectively. They are multiplied by p_1, q_1 in order to update the autocorrelation term $m_2(M-1;n)$. Then at t_1+2 the second token x_{n-M} at input line $\mathbf{c}_{v_1}$ of PE_{M-1} becomes available at stage $l=1$ of PE_{M-2}; at the same time and PE token x_{n-M+2} is also fed via line $\mathbf{c}_{w_1}$ and participate at the first useful computation in MAC stage $l=1$ of PE_{M-2}. Notice that even though a token x_{n-M} (the first in $\mathbf{c}_{v_1}$ of PE_{M-1}) arrives at stage $l=1$ of PE_{M-2} at t_1+1, no useful computation is performed at that time because it corresponds to the execution of index point $[M-2,M-1,0,n,1]$ which does not belong to the index space ($i_2>i_1$). In a similar fashion all tokens arrive at every stage of all PEs at the proper schedule time instant so that synchronization is achieved thus guaranteeing correct operation. It is important to mention that no additional control complexity is introduced in order to deal with the non-useful computations. They are indeed allowed to happen and produce results that are discarded at the end of the processing. Furthermore for the given shape of the index space these computations cannot be avoided without introducing complicated non-linear scheduling, that will of-course induce considerable hardware/software cost and compromise the capability of formally proving the correctness of the design.

Let us now describe how some moment terms at PE_{M-2} are updated at sampling time $t_s=n$. During that time data tokens x_{n-M+2} and x_{n-M} are fed into stage $l=1$ of PE_{M-2}. The former is fed via the input line $\mathbf{c}_{w_1}$ and the latter via the communication link $\mathbf{c}_{v_1}$ coming from the right neighbor PE_{M-1}. Both tokens are loaded to internal product links p_1 and q_1 respectively. Next, the tokens x_n, x_{n-2} are fed into the stage $l=2$ originated from PE_{M-1} via link $\mathbf{c}_{w_2}$ and input line $\mathbf{c}_{v_2}$ respectively. They are multiplied with the products of the previous stage p_1 and q_1 respectively to form products $p_2=x_{n-M+2}x_n$ and $q_2=x_{n-M}x_{n-2}$. These products are added/subtracted respectively from autocorrelation term $m_2(M-2;n-1)\equiv x_{n-M}x_{n-2}+x_{n-M+1}x_{n-1}$ to produce the new estimate $m_2(M-2;n)=x_{n-(M-1)}x_{n-1}+x_{n-M+2}x_n$. Similarly, tokens x_n and x_{n-2} are fed into the third stage, both coming from PE_{M-1} via links $\mathbf{c}_{w_3}$ and $\mathbf{c}_{v_3}$ respectively. They are multiplied with p_2 and q_2, added and subtracted from the $m_3(M-2,M-2;n-1)\equiv x_{n-M}x_{n-2}^2+x_{n-M+1}x_{n-1}^2$ to yield $m_3(M-2,M-2;n)=x_{n-M+1}x_{n-1}^2+x_{n-M+2}x_n^2$. Similarly, $m_4(M-2,M-2,0;n)=x_{n-M+1}^2x_{n-1}^2+x_{n-M+2}^2x_n^2$ is produced at stage MAC($i_1=M-2,l=4$). Clearly the resulting moment terms correspond to those defined by (1) based on the M- size sliding data window at time n.

At any time iteration, the array computes all the lags in $\mathcal{R}_0$ of all the moments up to the 4th order within $M(M-1)+3$ schedule periods using $4M$ pairs of MAC units (a schedule period is the time required for a MAC-stage operation). We can easily verify that claim as follows: Due to the non-redundant region of estimation, PE_{M-1} performs most of the computation. Index points assigned to that PE vary from $[M-1,M-1,0,n,1]^T$ to $[M-1,M-1,M-1,n,4]^T$. The difference $[0,0,M-1,0,3]^T$ between these two points translates to a $M(M-1)+3$ time difference under the selected schedule $\mathbf{T}=[-1,-1,M,M^2,1]$. The performance figures attained by the array are summarized in Table 1. For every sampling time iteration n, the sequential algorithm

based on (2) requires $\begin{pmatrix} M+k-2 \\ k-1 \end{pmatrix} = \frac{M(M+1)...(M+k-2)}{(k-1)!}$ multiplications to produce every product $p_k(i_1, ..., i_{k-1}; n)$ within $\mathcal{R}_0$ [13]. Thus, the total number of multiplications required would be twice as much to account for the product q_k as well. Thus, the total sequential time after N sampling periods would be $L_s = \frac{M(M+1)(M+2)N}{3}$ for orders up to the fourth ($k = 4$). On the other hand, the total parallel execution time in the array will be the time difference between the execution of the first ($[M-1, M-1, 0, M, 1]^T$) and the last ($[M-1, M-1, M-1, N+M, 4]^T$) index points according to $\mathbf{T}$. This gives a latency $L_p = M^2N + M(M-1) + 3$ and consequently theoretical speedup (as $N \to \infty$) of $S = \frac{(M+1)(M+2)}{3M}$. The PE locations and time instants of results generation are provided in Table 2.

5 Moments estimation over an 1-D slice

In many practical situations only one or more 1-D slices of the higher order moments is needed [19, 1]. The formula for such a *diagonal* 1-D slice can be easily deduced from (1) if we let $i_1 = i_2 = \cdots = i_{k-1}$

$$\widehat{m}_k(i_1, i_1, ..., i_1; n) = \frac{1}{M} \sum_{i=n-M+1}^{n-i_1} x_i \; x_{i+i_1}^{k-1} \tag{11}$$

Following similar steps as with the general case, time-recursive computation of the diagonal 1-D slice based on a fixed-size M sliding data window is possible according to:

$$\widehat{m}_l(i_1, i_1, ..., i_1; n) = \widehat{m}_l(i_1, i_1, ..., i_1; n-1) + \frac{1}{M}\left(p_{l-1}(i_1; n)\; x_n - q_{l-1}(i_1; n)\; x_{n-M+i_1}\right), \tag{12}$$

where the product terms are defined as $p_l(i_1; n) \equiv x_{n-i_1} \; x_n^{(l-1)}$ and $q_l(i_1; n) \equiv x_{n-M} \; x_{n-M+i_1}^{(l-1)}$, $l = 1, 2, \cdots k$. A nested-loop, time- and order-recursive algorithm based on (12) is shown is Fig. 6. Next we derive an array processor for the computation of the diagonal 1-D slice of all moments up to the fourth order based on this algorithm.

The algorithm index space is now $\mathcal{I} \equiv \{(i_1, n, l) : \; 0 \le i_1 \le M-1, \; n \ge M, \; 1 \le l \le 4\}$. We define the variables $w_1(i_1, n) \equiv x_{n-i_1}$, $w_l(i_1, n) \equiv x_n$, $v_1(i_1, n) \equiv x_{n-M}$ and $v_l(i_1, n) \equiv x_{n-M+i_1}$ for $l = 2, 3, 4$. Based on their indexing matrices we find that variables w and v are broadcasts whereas p, q and m are not. The corresponding data dependences are:

$$\begin{aligned}
\mathbf{d}_p &= \mathbf{d}_q = \mathbf{e}_3 \\
\mathbf{d}_m &= \mathbf{e}_2 \\
\mathbf{d}_{w_1} &= a_1(\mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)}) \\
\mathbf{d}_{w_2} &= \mathbf{d}_{w_3} = \mathbf{d}_{w_4} = a\mathbf{e}_1^{(2)} \\
\mathbf{d}_{v_1} &= b_1\mathbf{e}_1^{(2)} \\
\mathbf{d}_{v_2} &= \mathbf{d}_{v_3} = \mathbf{d}_{v_4} = b(-\mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)})
\end{aligned} \tag{13}$$

We project "along n and l" to get a bounded processor space and reduce the granularity of every PE. These projections yield the space transformation $\mathbf{S} = \mathbf{e}_1^T$. Next we need to select a permissible linear schedule $\mathbf{T} = [t_1, t_2, t_3]$. The precedence constraints with respect to the true dependences require that $t_2 \geq 1, \quad t_3 \geq 1$. In addition the space-time compatibility constraints require that $t_2\delta_2 + t_3\delta_3 \geq 1$ for every pair of arbitrary and distinct points of the index space. But for the given $\mathcal{I}$ we have $|\delta_3| \leq 3$. So to avoid collisions it should be true that $t_2 \geq 1 + 3t_3$. Therefore if we let $a_1 = b_1 = a = b = 1$ then the schedule $\mathbf{T} = [1, 4, 1]$, whose elements do not depend on the problem size M, is permissible. The dependence matrix $\mathbf{D}$ becomes:

$$\mathbf{D} \equiv [\mathbf{d}_{w_1}, \mathbf{d}_{w_2}, \mathbf{d}_{w_3}, \mathbf{d}_{w_4}, \mathbf{d}_{v_1}, \mathbf{d}_{v_2}, \mathbf{d}_{v_3}, \mathbf{d}_{v_4}, \mathbf{d}_p, \mathbf{d}_q, \mathbf{d}_m] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{14}$$

Due to the dependences $\mathbf{d}_{w_1}$ and $\mathbf{d}_{v_l}$, $l = 2, 3, 4$ input is required to all the PEs when $n = M$. However this can be avoided if we *extend* the algorithm index space appropriately. Indeed let us consider the index space where $n \geq 1$. This extension reduces the input hyperplanes to $i_1 = 0$ and $i_1 = M - 1$ that are mapped under $\mathbf{S}$ to the border processors PE_0 and PE_{M-1} respectively. The corresponding LRA is given in Figure 7.

Applying the aforementioned space-time transformations to the LRA we get the SFG shown in Fig. 8. The resulting architecture is a linear array of M bidirectionally interconnected PEs along the i_1 direction. Every SFG node accepts data from four communication lines $\mathbf{c}_{w_1}$, $\mathbf{c}_w$, $\mathbf{c}_{v_1}$ and $\mathbf{c}_v$ with delays $5, 1, 1, 3$ respectively. The sampling time period is now equivalent to four schedule periods, i.e., a new data sample may become available and consumed by the array every four schedule time instants. During the first $M(M-1)$ schedule time instants of operation, preloading takes place. This phase corresponds to the execution of all points of the index space extension (part with $1 \leq n \leq M - 1$). During preloading data tokens $\{x_1, ..., x_{M-1}\}$ and $\{x_0, ..., x_{M-1}\}$ are fed to PE_0 and PE_{M-1} via lines $\mathbf{c}_{w_1}$ and $\mathbf{c}_v$ respectively and from those PEs they become available wherever they might be needed during the computation phase. Preloading takes a total of $M(M-1)$ schedule time instants to be completed and there are no results generated during this phase.

During computation phase, every PE operates on a periodic fashion with period equal to 4 schedule time instants. At the first schedule time instant, it receives data from lines $\mathbf{c}_{w_1}$ and $\mathbf{c}_{v_1}$ whereas at the rest three instants data are fed from lines $\mathbf{c}_w$ and $\mathbf{c}_v$. Computation starts when sample x_M becomes available (i.e. sampling instant $t_s = M$) or equivalently $M(M-1)$ schedule time instants after the initiation of array operation. In general, when x_n becomes available ($t_s = n$) lines $\mathbf{c}_{w_1}$ and $\mathbf{c}_{v_1}$ should receive tokens x_n and x_{n-M} respectively. At the next three consecutive schedule time instants (within the same $t_s = n$), every PE receives data from lines $\mathbf{c}_w$ and $\mathbf{c}_v$ which carry data tokens x_n and x_{n-1} respectively. Apparently, the interface with the host is considerably

simpler. Moment terms at sampling instant $t_s = n$ need only three data samples, namely the current x_n, the previous one x_{n-1} and that one before M sampling time instants x_{n-M}. Update of the moment terms is performed in the same fashion as with the array in the previous section. The only difference is that every PE updates the moment terms of different order in a sequential rather than pipelined fashion.

We give description of the architecture by explaining the update of moment terms with lag $i_1 = 1$ at PE_1 during the sampling time instant $t_s = n$. At that time the moment terms are updated according to (12) as follows:

$$\begin{aligned} m_2(1;n) &\leftarrow m_2(1;n-1) + x_{n-1}x_n - x_{n-M}x_{n-M+1} \\ m_3(1,1;n) &\leftarrow m_3(1,1;n-1) + x_{n-1}x_n^2 - x_{n-M}x_{n-M+1}^2 \\ m_4(1,1,1;n) &\leftarrow m_4(1,1,1;n-1) + x_{n-1}x_n^3 - x_{n-M}x_{n-M+1}^3 \end{aligned} \tag{15}$$

PE_1 starts updating the moment terms with lags $i_1 = 1$ at sampling instant $t_s = n$ when the index point $[1, n, 1]^T$ is executed under the selected schedule. This is also schedule time instant $t_1 \equiv 4n + 2$. At that time the register corresponding to $\mathbf{c}_p$ is loaded with the value carried by link $\mathbf{c}_{w_1}$. This line has a 5-size buffer associated with it, meaning that the token residing in the fifth position of the buffer that is consumed by $\mathbf{c}_p$ at PE_1 has been fed as input to the architecture via $\mathbf{c}_{w_1}$ five schedule time instants before i.e., at time $t = 4n - 3 = 4(n-1) + 1$. But at that time token x_{n-1} is fed at input line $\mathbf{c}_{w_1}$ at PE_0. On the other hand the contents of register $\mathbf{c}_q$ at PE_1 are loaded with token x_{n-M} because this is the token that is fed to the architecture at PE_0 at time $t_1 - 1$, where the one delay we subtract corresponds to the delay associated with link $\mathbf{c}_{v_1}$. At the next schedule time instant, the content of $\mathbf{c}_p$ is multiplied with the token available at PE_1 via link $\mathbf{c}_w$, namely x_n. Indeed, this token is fed as input via line $\mathbf{c}_w$ at PE_0 at time t_1. On the other hand the content of $\mathbf{c}_q$ is multiplied by the token currently available at link $\mathbf{c}_v$, namely x_{n-M+1}. This token is originated from PE_{M-1} at sampling time instant $t_s = n - M + 2$, or equivalently when index point $[M-1, n-M+2, 2]^T$ is executed, i.e., at schedule time instant $t = 4n-3M+9$. Indeed, at that time, token x_{n-M+1} is fed as input via $\mathbf{c}_v$ at PE_{M-1}. Then, after $3(M-2)$ schedule time instants, it becomes available at PE_1 at time $4n - 3M + 9 - 3(M-2) = 4n + 3$. Thus, contents of registers $\mathbf{c}_p$ and $\mathbf{c}_q$ become $x_n x_{n-1}$ and $x_{n-M}x_{n-M+1}$ respectively. Clearly, these are the necessary terms to be added subtracted respectively from $m_2(1;n-1)$ in order to yield the new estimates $m_2(1;n)$. In a similar fashion the rest of the moment terms are updated and assume their new values according to (15).

6 Conclusions– Further research directions

We have investigated the adaptive computation of higher order moments based on real-time data. We first proposed two time- and order- recursive algorithms for the computation of all moments up to an arbitrary order, based on a fixed size sliding, or a growing data window. Then we

systematically transformed the fixed size sliding window algorithm onto a linear array realization suitable for VLSI implementation. At any time instant, the last received sample along with the M previous ones are used to update all moment lags up to the fourth order within the primary domain of support. As a special case a linear array that adaptively computes the diagonal 1-D slice of moments has been synthesized.

The *unified* systematic array synthesis methodology employed [13], facilitated (a) the design of processor arrays with elements having regular, pipelined structure and problem size independent memory requirements, and (b) the formal verification of the architecture's correctness at the behavioral level. The evaluation of trade-offs between computational complexity and local memory, the partitioning and matching of the algorithm to arrays with a fixed number of processors, and the integration of the moment generating array to an architecture can also provide the cumulants via the moments, are currently under investigation.

Initially: $p_0(i_1, ..., i_{k-1}; n) = q_0(i_1, ..., i_{k-1}; n) = 1, \ \forall \ (i_1, ..., i_{k-1}) \in \mathcal{R}_0$ and $n \geq M. \ i_0 \equiv 0.$

for $n = M, M+1, M+2, ...$
  for $i_1 = 0$ to $M-1$
    for $i_2 = 0$ to i_1
      $\vdots$
      for $i_{k-1} = 0$ to i_{k-2}
        for $l = 1$ to k

$$p_l(i_1, ..., i_{k-1}; n) \leftarrow p_{l-1}(i_1, ..., i_{k-1}; n) \ x_{n-i_1+i_{l-1}}$$

$$q_l(i_1, ..., i_{k-1}; n) \leftarrow q_{l-1}(i_1, ..., i_{k-1}; n) \ x_{n-M+i_{l-1}}$$

$$m_l(i_1, ..., i_{k-1}; n) \leftarrow m_l(i_1, ..., i_{k-1}; n-1) + p_l(i_1, ..., i_{k-1}; n) - q_l(i_1, ..., i_{k-1}; n)$$

        endfor $\{l\}$
      endfor $\{i_{k-1}\}$
      $\vdots$
    endfor $\{i_2\}$
  endfor $\{i_1\}$
endfor $\{n\}$

Moment Estimates: $\widehat{m}_l(i_1, ..., i_{k-1}; n) \leftarrow \frac{1}{M} m_l(i_1, ..., i_{k-1}; n)$

Figure 1: A nested-loop, time- and order- recursive algorithm for the adaptive updating of all the moments, up to a desired order k, at the primary domain of support $\mathcal{R}_0 = \{(i_1, ..., i_{k-1}) : \ 0 \leq i_{k-1} \leq ... \leq i_2 \leq i_1 \leq M-1\}$.

Initially: $p(i_1,i_2,i_3,n,0) = q(i_1,i_2,i_3,n,0) = 1, \ \forall\ (i_1,i_2,i_3) \in \mathcal{R}_0$ and $n \geq 1$.

for $n = M, M+1, M+2, ...$

for $i_1 = 0$ to $M-1$

for $i_2 = 0$ to i_1

for $i_3 = 0$ to i_2

for $l = 1$ to 4

$$w(i_1,i_2,i_3,n,l) \leftarrow \begin{cases} w(i_1,i_2+1,i_3,n,l), & \text{if } l=1,\ i_2 < i_1 \quad (\mathbf{d}_{w_1} = -\mathbf{e}_2) \\ x_{n-i_1}, & \text{if } l=1,\ i_1 = i_2 \\ \\ w(i_1+1,i_2,i_3,n,l), & \text{if } l=2,\ i_1 < M\text{-}1 \quad (\mathbf{d}_{w_2} = -\mathbf{e}_1) \\ x_n, & \text{if } l=2,\ i_1 = M\text{-}1 \\ \\ w(i_1+1,i_2+1,i_3,n,l), & \text{if } l=3,\ i_1 < M\text{-}1 \quad (\mathbf{d}_{w_3} = -\mathbf{e}_1 - \mathbf{e}_2) \\ x_{n-i_1+i_2}, & \text{if } l=3,\ i_1 = M\text{-}1 \\ \\ w(i_1,i_2+1,i_3,n,l), & \text{if } l=4,\ i_2 < i_1 \quad (\mathbf{d}_{w_4} = -\mathbf{e}_2) \\ x_{n-i_1+i_3}, & \text{if } l=4,\ i_1 = i_2 \end{cases}$$

$$v(i_1,i_2,i_3,n,l) \leftarrow \begin{cases} v(i_1+1,i_2,i_3,n,l), & \text{if } l=1,\ i_1 < M\text{-}1 \quad (\mathbf{d}_{v_1} = -\mathbf{e}_1) \\ x_{n-M}, & \text{if } l=1,\ i_1 = M\text{-}1 \\ \\ v(i_1,i_2+1,i_3,n,l), & \text{if } l=2,\ i_2 < i_1 \quad (\mathbf{d}_{v_2} = -\mathbf{e}_2) \\ x_{n-M+i_1}, & \text{if } l=2,\ i_1 = i_2 \\ \\ v(i_1+1,i_2,i_3,n,l), & \text{if } l=3,\ i_1 < M\text{-}1 \quad (\mathbf{d}_{v_3} = -\mathbf{e}_1) \\ x_{n-M+i_2}, & \text{if } l=3,\ i_1 = M\text{-}1 \\ \\ v(i_1,i_2+1,i_3,n,l), & \text{if } l=4,\ i_2 < i_1 \quad (\mathbf{d}_{v_4} = -\mathbf{e}_2) \\ x_{n-M+i_3}, & \text{if } l=4,\ i_1 = i_2 \end{cases}$$

$$p(i_1,i_2,i_3,n,l) \leftarrow p(i_1,i_2,i_3,n,l-1) \cdot w(i_1,i_2,i_3,n,l) \quad (\mathbf{d}_p = \mathbf{e}_5)$$

$$q(i_1,i_2,i_3,n,l) \leftarrow q(i_1,i_2,i_3,n,l-1) \cdot v(i_1,i_2,i_3,n,l) \quad (\mathbf{d}_q = \mathbf{e}_5)$$

$$m(i_1,i_2,i_3,n,l) \leftarrow m(i_1,i_2,i_3,n-1,l) + p(i_1,i_2,i_3,n,l) - q(i_1,...,i_{k-1},n,l) \quad (\mathbf{d}_m = \mathbf{e}_4)$$

endfor $\{l\}$

endfor $\{i_3\}$

endfor $\{i_2\}$

endfor $\{i_1\}$

endfor $\{n\}$

Moment Estimates: $\widehat{m}(i_1,i_2,i_3,n,l) \leftarrow \frac{1}{M} m(i_1,i_2,i_3,n,l)$

Figure 2: The Locally Recursive Algorithm in Single Assignment Form constructed systematically for the time- and order- recursive updating of all moments up to the 4th order, $k = 4$.

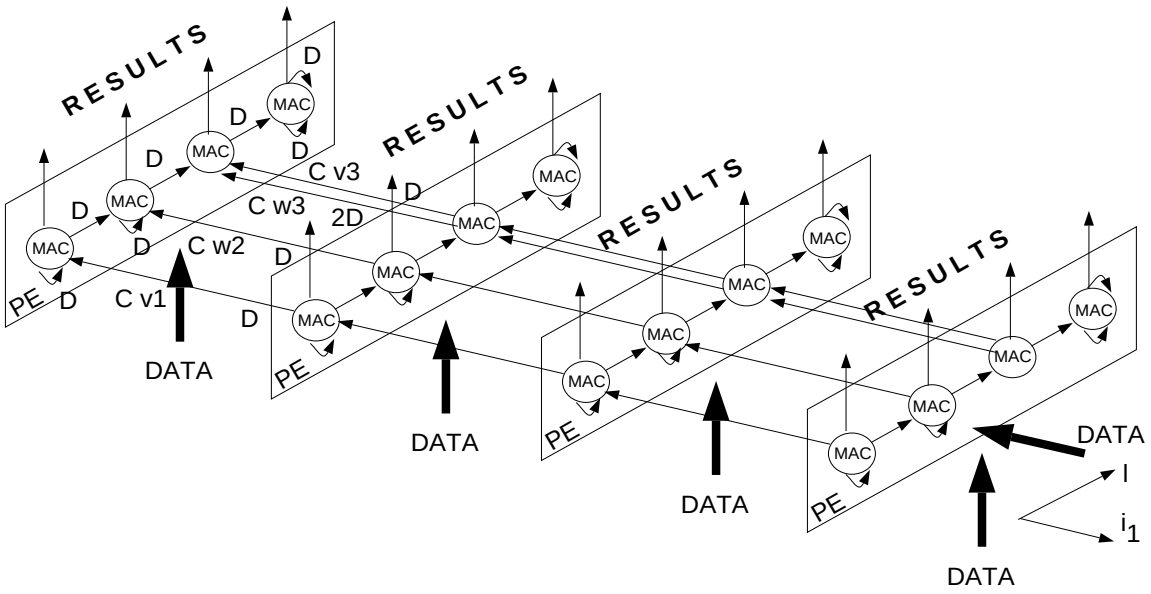

Figure 3: The structure of the linear array. Every PE consists of 4 pairs of MAC units. Product terms of different orders are generated along the direction l. The communication delays are shown in the left most PE ($i_1 = 0$) only. The data window size is $M = 4$.

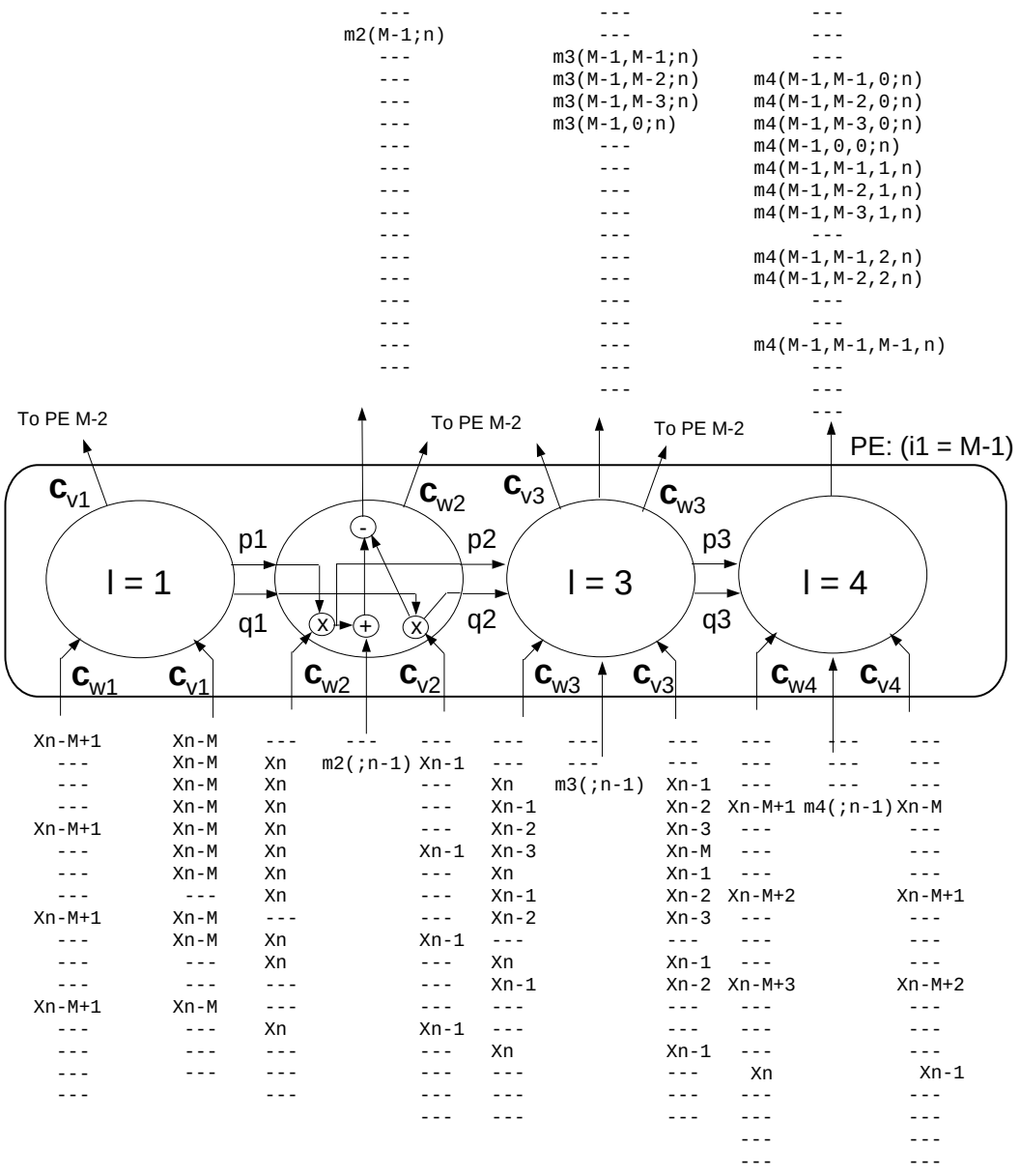

Figure 4: The structure of PE_{M-1} and the input/output data skews at time when sample x_n becomes available (i.e. $t_s = n$).

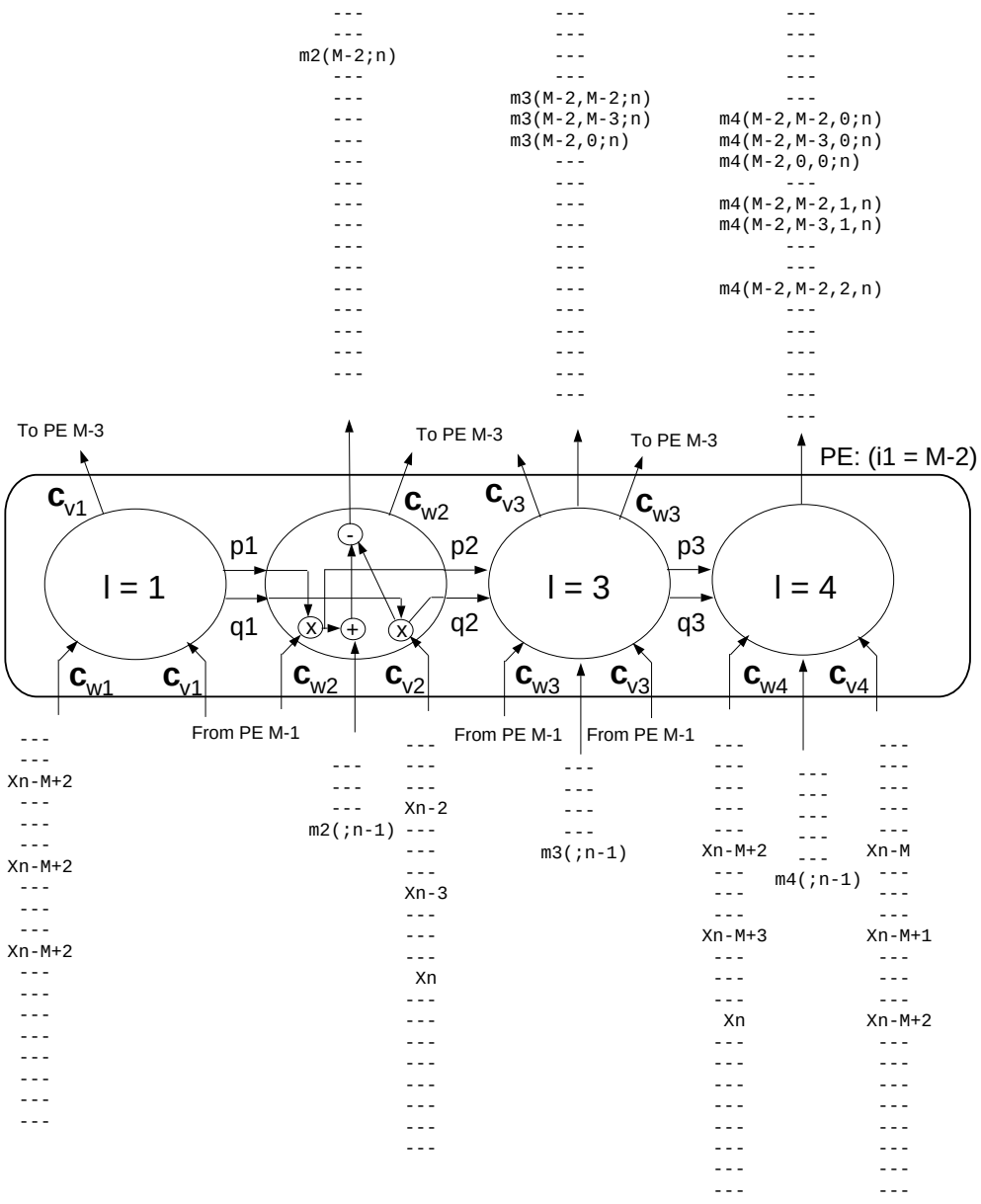

Figure 5: The structure of PE_{M-2} and the input/output data skews at time when sample x_n becomes available (i.e. $t_s = n$).

Initially: $p_0(i_1;n) = q_0(i_1;n) = 1, \ \forall \ i_1 \in [0, M-1]$ and $n \geq M$.

for $n = M, M+1, M+2, ...$
  for $i_1 = 0$ to $M-1$
    for $l = 1$ to k

$$p_l(i_1;n) \leftarrow \begin{cases} p_{l-1}(i_1;n)\, x_n, & \text{if } l > 1 \\ x_{n-i_1}, & \text{if } l = 1 \end{cases}$$

$$q_l(i_1;n) \leftarrow \begin{cases} q_{l-1}(i_1;n)\, x_{n-M+i_1}, & \text{if } l > 1 \\ x_{n-M}, & \text{if } l = 1 \end{cases}$$

$$m_l(i_1, ..., i_1; n) \leftarrow m_l(i_1, ..., i_1; n-1) + p_l(i_1;n) - q_l(i_1;n)$$

    endfor $\{l\}$
  endfor $\{i_1\}$
endfor $\{n\}$

Moment Estimates: $\widehat{m}_l(i_1, ..., i_1; n) \leftarrow \frac{1}{M} m_l(i_1, ..., i_1; n)$

Figure 6: A nested-loop, time- and order- recursive algorithm for the adaptive computation of the diagonal 1-D slice of all moments, up to a desired order k.

Initially: $p(i_1,,n,0) = q(i_1,n,0) = 1,\ \forall\ 0 \le i_1 \le M-1,\ n \ge 1$

for $n = 1, 2, ..., M, M+1, M+1, ...$
  for $i_1 = 0$ to $M-1$
    for $l = 1$ to k

$$w(i_1,n,l) \leftarrow \begin{cases} w(i_1-1,n-1,l), & \text{if } l=1,\ i_1>0 \quad (\mathbf{d}_{w_1} = \mathbf{e}_1 + \mathbf{e}_2) \\ w(i_1-1,n,l), & \text{if } l>1,\ i_1>0 \quad (\mathbf{d}_{w_l} = \mathbf{e}_1) \\ x_n, & \text{if } i_1 = 0, \forall l \end{cases}$$

$$v(i_1,n,l) \leftarrow \begin{cases} v(i_1-1,n,l), & \text{if } l=1,\ i_1>0 \quad (\mathbf{d}_{v_1} = \mathbf{e}_1) \\ x_{n-M}, & \text{if } l=1,\ i_1=0 \\ v(i_1+1,n-1,l), & \text{if } l>1,\ i_1<M-1 \quad (\mathbf{d}_{v_l} = -\mathbf{e}_1 + \mathbf{e}_2) \\ x_{n-1}, & \text{if } l>1,\ i_1=M-1 \end{cases}$$

$$p(i_1,n,l) \leftarrow p(i_1,n,l-1)\ w(i_1,n,l) \quad \text{if } n \ge M \qquad (\mathbf{d}_p = \mathbf{e}_3)$$

$$q(i_1,n,l) \leftarrow q(i_1,n,l-1)\ v(i_1,n,l) \quad \text{if } n \ge M \qquad (\mathbf{d}_q = \mathbf{e}_3)$$

$$m(i_1,n,l) \leftarrow m(i_1,n-1,l) + p(i_1,n,l) - q(i_1,n,l) \quad \text{if } n \ge M \qquad (\mathbf{d}_m = \mathbf{e}_2)$$

    endfor $\{l\}$
  endfor $\{i_1\}$
endfor $\{n\}$

Moment Estimates: $\widehat{m}_1(;n) \leftarrow \frac{1}{M} m(i_1,n,1)$, for $n \ge M$

$$\widehat{m}_2(i_1;n) \leftarrow \frac{1}{M} m(i_1,n,2)$$
$$\widehat{m}_3(i_1,i_1;n) \leftarrow \frac{1}{M} m(i_1,n,3)$$
$$\widehat{m}_4(i_1,i_1,i_1;n) \leftarrow \frac{1}{M} m(i_1,n,4)$$

Figure 7: The constructed LRA for the time- and order- recursive updating of all the moments up to the fourth order on the diagonal 1-D slice

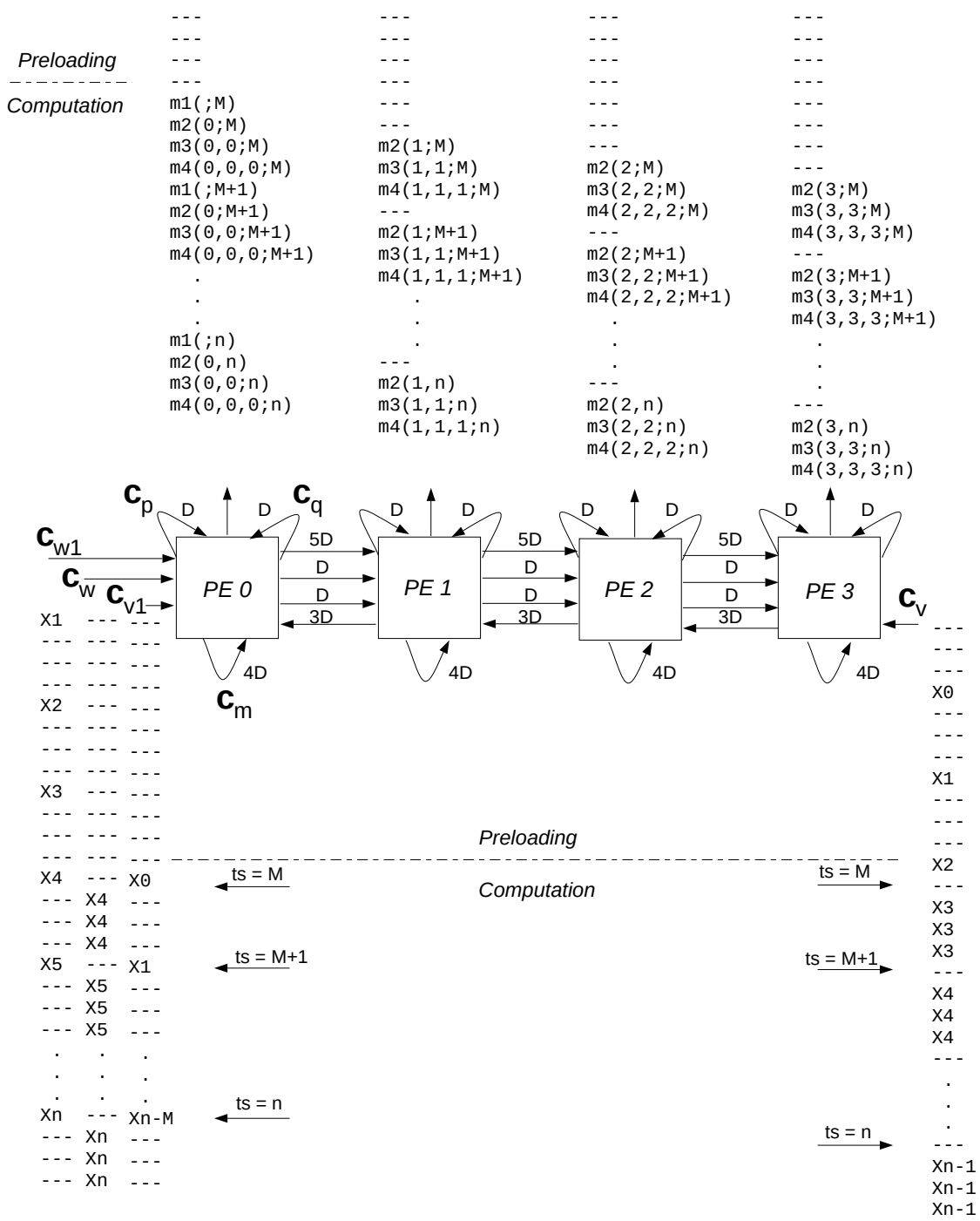

Figure 8: The SFG with a typical data skew for the time-and order- recursive updating of 1-D slices of all moments up to the fourth order, $M = 4$.

Algorithm Execution	**Latency (schedule time)**	**Speedup**, $(N \rightarrow \infty)$
Sequential	$L_s = \frac{M(M+1)(M+2)N}{3}$	$S \equiv L_s/L_p = \frac{(M+1)(M+2)}{3M}$
Parallel (Linear Array)	$L_p = M^2N + M(M-1) + 3$	

Table 1: Comparison of the sequential and parallel running times achieved using the proposed time- and order- recursive algorithm for the computation of all the moments up to the 4th order. We assume that the data sequence has length $N >> M$ and the sliding window has length M.

Sequence	**PE Index**,	**MAC Index**	**Schedule Time**
$m^{(1)}(n)$	$i_1 = 0,$	$l = 1$	$M^2(n-M) + 2(M-1)$
$m^{(2)}(i_1, n)$	$0 \leq i_1 \leq M-1,$	$l = 2$	$-i_1 + M^2(n-M) + 2M - 1$
$m^{(3)}(i_1, i_2, n)$	$0 \leq i_1 \leq M-1,$	$l = 3$	$-i_1 - i_2 + M^2(n-M) + 2M$
$m^{(4)}(i_1, i_2, i_3, n)$	$0 \leq i_1 \leq M-1,$	$l = 4$	$-i_1 - i_2 + Mi_3 + M^2(n-M) + 2M + 1$

Table 2: The PE locations and time instants of results generation on the linear array. The permissible schedule used is $\mathbf{T} = [-1, -1, M, M^2, 1]$. It is assumed that the array starts operating at $t = 0$.

References

[1] J.M. Mendel. "Tutorial on Higher-Order Statistics (spectra) in Signal Processing and System Theory: Theoretical Results and Some Applications". *Proceedings IEEE*, 79(3):278–305, March 1991.

[2] C.L. Nikias and A.P. Petropulu. *Higher-Order Spectral Analysis: a Nonlinear Signal Processing Framework*. Prentice Hall, in press, 1993.

[3] C.L. Nikias and J.M. Mendel. Signal Processing with Higher-Order Spectra. *IEEE Signal Processing Magazine*, 10(3):10–37, July 1993.

[4] J.R. Fonollosa and C.L. Nikias. "Wigner Higher-Order Moment Spectra: Definition, Properties, Computation and Application to Transient Signal Analysis". *IEEE Transactions on Signal Processing*, 41(1):245–266, January 1993.

[5] R. Perry and M. Amin. "Systolic Array Implementation of Recursive Bispectrum Estimation". In *Proc. of IEEE Signal Processing Workshop on Higher-Order Statistics*, pages 91–95, June 1993.

[6] K. Kameyama, M. Iwata, T. Sakai, K-Y Jhang, and T. Sato. "Acousto-Optical third order corellator". In *Proc. of IEEE Signal Processing Workshop on Higher-Order Statistics*, pages 81–85, June 1993.

[7] K. Kameyama, M. Sakamoto, H. Akagi, K-Y Jhang, and T. Sato. "Robot Vision System using High Order Correlation Analysis". In *Proc. of IEEE Signal Processing Workshop on Higher-Order Statistics*, pages 86–90, June 1993.

[8] E.S. Manolakos, H.M. Stellakis, and D.H. Brooks. "Parallel Processing for Biomedical Signal Processing: Higher Order Spectral Analysis - An Application". *IEEE Computer*, 24(3):33–43, March 1991.

[9] C. L. Nikias and M. R. Raghuveer. "Bispectrum Estimation: A Digital Signal Processing Framework". *IEEE Proceedings*, 75(7):869–891, July 1987.

[10] H.M. Stellakis and E.S. Manolakos. "An Array of Processors for the Real-Time Estimation of the Fourth and lower Order Moments". *Signal Processing, Eurasip Journal (*special issue on Higher Order Statistics*)*, 36(3):341–354, 1994.

[11] H.M. Stellakis and E.S. Manolakos. "An Architecture for the Real-Time Estimation of Cumulants". In *Proceedings of IEEE Int'l Conference on Acoustics Speech and Signal Processing, (ICASSP'93)*, pages IV·220–IV·223, April 1993.

[12] H.M. Stellakis and E.S. Manolakos. "A VLSI Architecture for the Order Recursive Estimation of Higher Order Statistics". In *Proceedings of IEEE Signal Processing Workshop on Higher Order Statistics*, pages 96–100, June 1993.

[13] H.M. Stellakis. *"Systematic Synthesis of Parallel Architectures for Estimation of Higher Order Statistics"*. PhD thesis, Northeastern University, August 1993.

[14] S.Y. Kung. *VLSI Array Processors*. Prentice Hall, 1989.

[15] W. Shang and J. A. B. Fortes. "On Time Mapping of Uniform Dependence Algorithms into Lower Dimensional Processor Arrays". *IEEE Trans. on Parallel and Distributed Processing*, 3(3):350–363, May 1992.

[16] J. Bu, E.F. Deprettere, and L. Thiele. "Systolic Array Implemenation of Nested Loop Programs". In *Proc. of IEEE Int'l Conference on ASAP*, pages 31–42, 1990.

[17] J.A.B. Fortes and D.I. Moldovan. "Data Broadcasting in Linearly Scheduled Array Processors". In *Proc. of IEEE Annual International Symposium on Computer Architecture*, 1984.

[18] P. Lee and Z.M. Kedem. "Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays". *IEEE Transactions on Parallel and Distributed Systems*, 1(1):64–76, January 1990.

[19] G.B. Giannakis. "On the Identifiability of Non-Gaussian ARMA Models Using Cumulants. *IEEE Transactions on Automatic Control*, 35(1):18–26, January 1990.

List of Figures

List of Tables