Cost Measures in VLSI Array Design*

Peter Cappello † Sanjay Rajopadhye ‡

†Department of Computer Science, University of California, Santa Barbara, CA 93106
‡Department of Computer Science, University of Oregon, Eugene, OR 97403-1202

Abstract

Using a directed acyclic graph (dag) model of algorithms, several computational complexity measures are defined in terms of time, period, and processors. Research interest is explained for designs that are time-minimal, processor-time-minimal, and period-processor-time-minimal. Such designs are illustrated with the standard matrix product algorithm, modeled by the $n \times n \times n$ directed mesh.

1 Introduction

Although it is only one style of concurrent computation, systolic computation is both physically suited to VLSI technology [17], and natural for many algorithms [13, 20]. These properties suggest that it has an important, enduring role to play in concurrent computation. The systolic paradigm consequently has been applied to a wide variety of problems [13]. The 'goodness' of a systolic array is an issue in concrete computational complexity. Such issues can be investigated using a computational model that is either algorithm-dependent or algorithm-independent. With an

*This work supported by the National Science Foundation under grants MIP-89-20598 and MIP-88-02454, and the Lawrence Livermore National Laboratories.

algorithm-independent model, the computation typically is modeled by a set of binary inputs and outputs, where each output bit is defined as a boolean function of the input bits. When it is inconvenient or intractable to use an algorithm-independent model, we turn to an algorithm-dependent model. In such a model the computation often is viewed as a *directed acyclic graph* (dag). This is especially suitable for investigating systolic arrays—systolic algorithms have an iterative dag representation which can be extracted automatically from, for example, uniform recurrent equations [21], a system of uniform recurrence equations [12], regular iterative arrays [24, 10], a localization of a system of linear recurrence equations [6, 22, 23], as well as computations specified in the model of Moldovan and Fortes [18, 7, 19, 8].

We consider parameterized families of dags $G_n = (N, A)$ (n is the size parameter), whose nodes can be labelled with *integral index points* in a k-dimensional index space, and whose node set consists of all the integral points inside a convex polyhedron in k-space. The number of nodes in the dag is the volume of the polyhedron: $|N| = O(n^k)$. N can be partitioned into 3 sets N_{in}, N_{out}, and N_{int}, which respectively are the input nodes, the output nodes, and the internal nodes. N_{in} and N_{out} are typically the nodes on 1 or more faces of the polyhedron. $|N_{\text{in}}|$ and $|N_{\text{out}}|$ hence are $O(n^{k-1})$.

Given a dag, a multiprocessor schedule assigns node v for processing during step $\tau(v)$ on processor $\pi(v)$. The range of π also is a convex polyhedron (usually of $k-1$ dimensions). The schedule is subject to 2 constraints:

1. A node can be computed only when its children have been computed at previous time steps: $(u, v) \in A \Rightarrow \tau(u) < \tau(v)$.

2. No processor can compute 2 different nodes during the same time step: $\tau(v) = \tau(u) \Rightarrow \pi(v) \neq \pi(u)$.

In most of the work on systolic mapping, the schedules are restricted to be *linear transformations*:

$$\begin{bmatrix} \tau(v) \\ \pi(v) \end{bmatrix} = Mv$$

for some $k \times k$ matrix, M. In general, the designer is interested in determining the 'best' τ and π for a given dag, and a number

of different cost functions have been used. This paper attempts to develop a unified view of these measures. We first define these measures, study some relationships between them, and then discuss how they can be combined.

We are interested in these cost measures from 2 perspectives. By investigating the costs associated with linear mappings, we propose guidelines for practical mappings. From the complexity viewpoint, we are interested in the best that one can do for a given dag, regardless of the mapping chosen — we investigate some intrinsic properties of the dag.

2 Cost Measures for Mapping

Definition 1 [14] The *computation time* (T) is the time interval between the first computation and the last computation.

$$T = \max_{v \in N} \tau(v) - \min_{v \in N} \tau(v) = \max_{v \in N_{\text{out}}} \tau(v) - \min_{v \in N_{\text{in}}} \tau(v)$$

Definition 2 [14] The *processor pipelining period* (α) is the time interval between 2 successive computations in a processor (its reciprocal is called the *pipelining rate* or *processor utilization*.

Definition 3 [14] The *block pipelining period*, β, (also called the period—its reciprocal is throughput) is the time interval between the initiations of 2 successive computations by the processor array.

Definition 4 [14] The *array size* (P) is its number of processors.

Definition 5 The *input pipelining period* (γ) is the maximum time interval for any processor between 2 successive data inputs from the external world.

Definition 6 The *total computation time* (T_{tot}) is the time interval between the first data input (from the external world) to the array and the last result output from the array.

2.1 Relation between α, γ, and P

A common misconception in systolic array design is that a high value of α is *inherently* bad. Consider the hexagonal array for

band matrix product proposed by Kung and Leiserson which has $\alpha = 3$. A trivial modification (clustering 3 adjacent processors) not only reduces α to 1, but also reduces the number of processors by 2/3. Zhong and Rajopadhye [31] show that such clustering can be determined automatically, using quasi-linear mappings* In most work on linear mappings, the array size is taken to be the volume V of the image of N under π. The clustering result of Zhong and Rajopadhye shows that this is not an accurate view—the proper cost measure is V/α. The determination of linear (or quasi-linear) processor maps $\pi(v)$ that minimize V/α is an open problem. Such clustering may affect the input pipelining period, γ. Depending on the clustering chosen, γ may be reduced by a constant factor or may not change. However, the block pipelining period remains unaffected by any such clustering.

For a large class of problems, the dag is infinite. Such a situation may arise when dealing with signal processing applications where the input is a sampled signal. In such cases, the computation time and the period are always infinite, thus not subject to minimization. In this case, the only cost function that can be optimized is the input pipelining period, γ. S. Y. Kung [14] presents a procedure to minimize α (over linear transformations). This procedure can be adapted to optimize γ.

2.2 Relation between T and $T_{\mathbf{tot}}$

There is an important difference between computation time, T, and total computation time, T_{tot}: $T \leq T_{\text{tot}}$. We are concerned with mappings that schedule a dag with $O(n^k)$ nodes on a processor array with (usually) $O(n^{k-1})$ processors. Of these, only $O(n^{k-2})$ are on the boundary (since the processor array is itself 'convex'). Ideally, the $O(n^{k-1})$ nodes in N_{in} and N_{out} are mapped by π to boundary processors. When this is not the case, an input (output) node in the dag is mapped to an *internal* processor, even though it has no predecessors (successors) in the dag. The data required for (produced by) this computation comes from (goes to) the external world, which cannot directly communicate with this processor. Hence, one pays a *time* penalty for the data

*These are mappings where π is given by the floor of a rational linear transformation.

that is propagated to (from) this processor from (to) the boundary. In this case, the optimal schedule for the dag as determined by T is not an accurate measure of the time required by the array. An example of this is the array presented for the algebraic path problem presented by Rote [25]; it has a total computation time of $7n$. However, the first (and last) n time steps are for I/O. The actual computation time is $5n$. Indeed, if this array is clustered using the techniques developed by Zhong and Rajopadhye [31], then one obtains a processor-time-minimal array which has the same cost as those developed by Scheiman and Cappello [26] and Benaini and Robert [1].

3 Hierarchical combination of cost measures

We now are interested in combining the various cost measures, with a view to understanding the inherent properties of the dag—the perspective of concrete computational complexity. While it is true that the total computation time is an accurate measure of the time cost, it remains an open problem whether this can be expressed in an architecture-independent manner. The definition of 'boundary' processors seems architecture-dependent. For the purpose of this paper, we therefore consider only the computation time. As described above, the optimization of processor pipelining period is relevant typically for problems with infinite dags. We however shall deal only with families of finite dags. We illustrate algorithm-dependent complexity issues with a specific problem: matrix product. Its standard algorithm can be modeled by the cubical mesh (defined later)*. The standard algorithm does not imply the cubical mesh—it is but 1 dag obtainable by *localizing* the algorithm. Nevertheless, most systolic array designers use this dag as *the representation* of the algorithm, and all complexity issues are addressed in the context of such a specific dag. Systolic array designers map the cubical mesh into processor-time with a transformation that typically is *linear*. That is, the nodes of the dag are labelled with *index vectors*, $[i\ j\ k\ ]^T$, and the *time step*

*The dependence dag associated with banded matrix product is a subgraph of the cubical mesh.

and *processor location* to which $[\, i \;\; j \;\; k \,]^T$ is mapped is given by:

$$\begin{bmatrix} step \\ location_1 \\ location_2 \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix},$$

for some matrix $M \in \mathbf{Z}^{3\times 3}$. Fig. 1 depicts a linear map of the cubical mesh into processor-time. Linear maps of iterative depen-

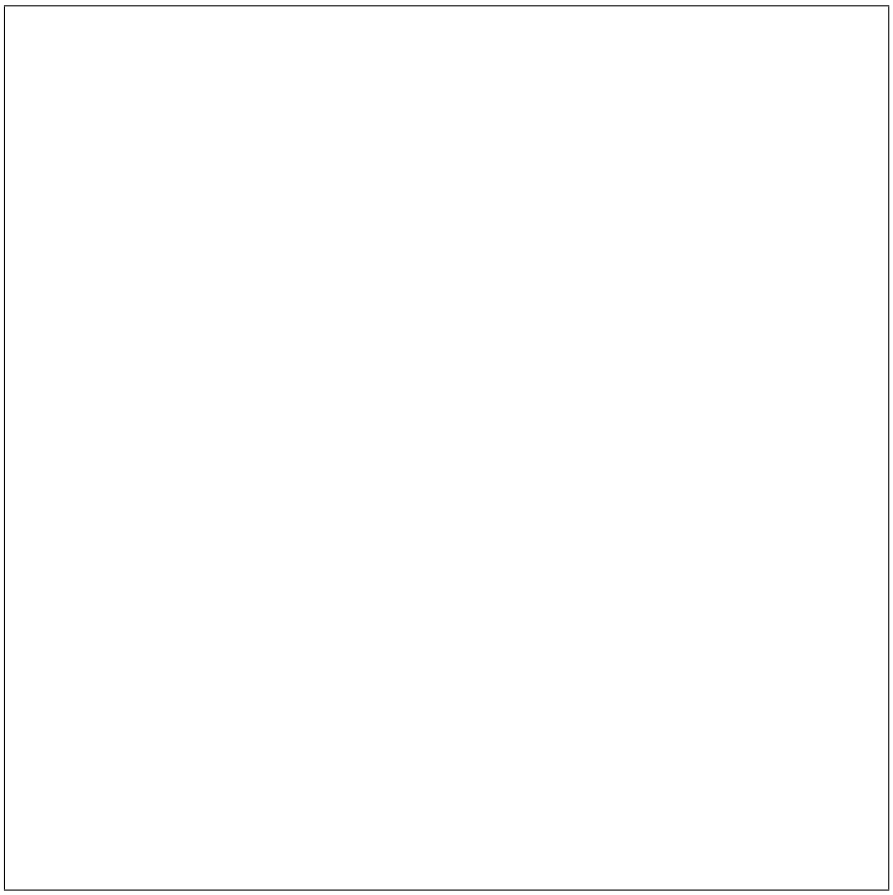

Figure 1: *A geometric representation of a schedule for the cubical mesh, for* $n = 2$*. The projection below is the processor array. The schedule uses* $3n - 2$ *steps,* n^2 *processors.*

dence dags have been researched intensely. There also has been a great deal of work on optimizing systolic arrays. The work pursued by Li and Wah [11], Fortes and Parisi-Presicce [9], Rao [24], Delosme and Ipsen [5], Chen [4], Lee and Kedem [15], Shang and

Fortes [28], and most recently by Wong and Delosme [29, 30], all contribute to methods for optimizing systolic arrays. These efforts constrain the processor-time mapping to be a linear or affine transformation of the problem's index set. The first reason that this constraint is used is because it yields systolic arrays that are both intuitively appealing and practical to implement. The question nonetheless arises as to whether relaxing the linearity constraint results in an even more efficient use of time and space. This question leads to the second reason that extant optimization efforts constrain the processor-time mapping to be linear or affine: the general problem of *precedence constrained scheduling* onto a set of processors is NP-complete. Computations that are NP-complete may be dealt with in several ways. One way is to isolate a fundamental, indexed family of problem instances, and find an optimal parameterized solution for that family. The dag family we shall use is the cubical mesh, $G_n = (N_n, A_n)$, where

- $N_n = \{(i, j, k) \mid 0 \leq i, j, k \leq n - 1\}$.
- $A_n = \{[(i, j, k), (i', j', k')] \mid$ where exactly 1 of the following conditions holds

  1. $i' = i + 1$
  2. $j' = j + 1$
  3. $k' = k + 1$

  for $0 \leq i, j, k \leq n - 1\}$.

We henceforth focus on 3 properties of schedules of unweighted dags: computation time, processors, and period.

Definition 7 A schedule for a dag is *time-minimal* when the number of steps in the schedule equals the number of nodes in a longest directed path in the dag.

The longest directed path in the cubical mesh clearly has $3n - 2$ nodes. The array depicted in Fig. 1 is time-minimal. Time-minimality measures the dag's maximum parallelism. It is more interesting than minimizing either processors or period. Minimizing processors is easy—scheduling all nodes onto a single processor is optimal. Minimizing the period also is easy—assign

a distinct processor to each node, interconnecting them isomorphic to the dag. This construction always achieves a period of 1 step. For example, we can construct a matrix product array with a period of 1 step by assigning a distinct processor to each of the n^3 nodes in Fig. 1, interconnecting them as a cubical mesh. *Within* time-minimality, it is more challenging to minimize processors than to minimize period; achieving the latter is easy, and does not affect time-minimality.

Definition 8 A time-minimal schedule for a dag is *processor-time-minimal* when it uses as few processors as any time-minimal schedule for the dag.

Processor-time-minimality discloses the minimum number of processors that are sufficient to extract the dag's maximum parallelism. Research on processor-time-minimal schedules recently has been conducted by Benaini and Robert [1] on a dag family for the algebraic path problem, by Louka and Tchuente [16] on a dag family for Gauss-Jordon elimination, by Benaini and Robert [1] on a dag family for Gaussian elimination, by Cappello [3, 2] on a dag family for matrix product, and by Scheiman and Cappello [26] on a dag family for transitive closure.

In order to sketch this lower bound for the cubical mesh, we define the notion of a concurrent set of nodes.

Definition 9 Let $G = (N, A)$ be a dag. We uniquely label each node $v \in N$ with number:

- i, when v is the ith node on a longest directed path in G;
- 0, otherwise.

Let L be the set of longest directed paths in G. This labeling partitions N into equivalence classes:

$$Q_0 = \{v \in N \mid \nexists l \in L \wedge l \text{ contains } v\}$$

$$Q_{i\neq 0} = \{v \in N \mid v \text{ is the } i\text{th node on some } l \in L\}$$

We refer to each nonzero equivalence class as a *concurrent* set of nodes.

Using this definition, we state a simple but useful theorem.

Theorem 1 *Let $G = (N, A)$ be a dag, $Q_i \subseteq N$ be a concurrent set of nodes, and P be the number of processors implementing a time-minimal schedule. Then $|Q_i| \leq P$.*

The cubical mesh, it turns out [3, 2], contains a concurrent set of size $\lceil 3n^2/4 \rceil$. Thus, according to Theorem 1, any time-minimal schedule of the cubical mesh requires at least $\lceil 3n^2/4 \rceil$ processors. The topology of the processor-time-minimal systolic array reported by Cappello [3, 2] is that of a hexagonally shaped, cylindrically connected 2D directed mesh. We now define the most stringent dag property illustrated in this paper.

Definition 10 A processor-time-minimal schedule for a dag is *period-processor-time-minimal* when it has the minimum period among all processor-time-minimal schedules for the dag.

Although only one of many performance measures, period-processor-time-minimality measures the *maximum throughput* obtainable when using the *minimum number of processing elements* that are sufficient to extract the dag's *maximum parallelism*. The dag family and an optimal schedule jointly imply *architectural* requirements for an optimal realization.

What is a lower bound for the *period* of a processor-time-minimal schedule? Let $G = (N, A)$ be a dag that has been scheduled using P processors. Consider the processor that uses the most steps in the schedule (determining the schedule's period). This number of steps, β, can be bounded from below by the following inequality:

$$|N| \text{ nodes} \leq P \text{ processors} \times \beta \text{ nodes/processor}. \tag{1}$$

For a processor-time-minimal systolic array for the cubical mesh, the lower bound for period is

$$\frac{n^3 \text{ nodes}}{3n^2/4 \text{ processors}} = \frac{4n}{3}\text{nodes/processor} \leq \beta \text{ nodes/proc.}$$

These bounds are not merely asymptotic, they are precise. For example, the $30 \times 30 \times 30$ mesh for computing a 30×30 matrix product requires at least 88 steps. Any multiprocessor that achieves this time-minimal schedule needs at least 675 inner-product-step processors. At least 1 of these processors requires at

least 40 steps for each matrix product. Scheiman and Cappello [27] present a schedule for the cubical mesh that achieves *exactly* this period-processor-time lower bound, whenever n is a multiple of 6. The optimal architecture implied by their schedule is a toroidally connected $n/2 \times n/2 \times 3$ systolic array.

One may be interested in trading among period, processors, and time in order to minimize their product: $period \times processors \times time$ (whose unit is $processor \times step^2$). From Eq. 1, we see that the $period \times processors$ product is bounded from below by $|N|$. This bound always is achievable (e.g., when $P = 1$ processor and the period $\beta = |N|$ steps, or when $P = |N|$ processors and the period $\beta = 1$ step). Thus, we have the following.

Lemma 1 *A schedule for dag $G = (N, A)$ is $period \times processors$-minimal if and only if its $period \times processor$ product $\beta \times P = |N|$.*

Multiplying this bound by the lower bound on time, we obtain the following lower bound for the cubical mesh: $3n^4 - 2n^3 \leq period \times processors \times time$. A schedule can be (period $\times$ processor)-minimal and time-minimal, and still not be period-processor-time-minimal. For example, the systolic array of Fig. 1 is both (period $\times$ processor)-minimal, and time-minimal, but it is *not* period-processor-time-minimal. We however do have the following.

Theorem 2 *If a schedule is (period $\times$ processor)-minimal and processor-time-minimal, then it is period-processor-time-minimal.*

A period-processor-time-minimal schedule is, in effect, an optimal concurrent communication program for the dag [family]. Investigating *fundamental algorithms* with respect to period-processor-time-minimality increases our basic understanding of their potential for concurrent realization. Finding the best [systolic] algorithm (i.e., [localized] dag) for a given problem is a difficult research problem that remains open.

References

[1] Abdelhamid Benaini and Yves Robert. Spacetime-minimal systolic arrays for gaussian elimination and the algebraic path problem. In *Proc. Int. Conf. on Application Specific Array Processors*,

pages 746–757, Princeton, September 1990. IEEE Computer Society.

[2] Peter Cappello. A processor-time-minimal systolic array for cubical mesh algorithms. *IEEE Trans. on Parallel and Distributed Systems*, recommended for acceptance.

[3] Peter R. Cappello. A spacetime-minimal systolic array for matrix product. In John V. McCanny, John McWhirter, and Earl E. Swartzlander Jr., editors, *Systolic Array Processors*, pages 347–356, Killarney, IRELAND, May 1989. Prentice-Hall.

[4] Marina C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *J. of Parallel and Distributed Computing*, pages 461–491, Dec. 1986.

[5] Jean-Marc Delosme and Ilse C. F. Ipsen. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *Proc. 2nd Int. Symp. on VLSI Technology, Systems and Applications*, pages 268–273, Taipei, 1985.

[6] Vincent Van Dongen and Patrice Quinton. Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays. In *Proc. Int. Conf. on Systolic Arrays*, pages 473–482, San Diego, May 1988. IEEE Computer Society.

[7] José A. B. Fortes. *Algorithm Transformations for Parallel Processing and VLSI Architecture Design.* PhD thesis, University of Southern California, Los Angeles, Dec. 1983.

[8] José A. B. Fortes and Dan I. Moldovan. Parallelism detection and algorithm transformation techniques useful for VLSI architecture design. *J. Parallel Distrib. Comput*, 2:277–301, Aug. 1985.

[9] José A. B. Fortes and F. Parisi-Presicce. Optimal linear schedules for the parallel execution of algorithms. In *Int. Conf. on Parallel Processing*, pages 322–328, Aug. 1984.

[10] H. V. Jagadish, Sailash K. Rao, and Thomas Kailath. Multiprocessor architectures for iterative algorithms. *Proc. IEEE*, September 1987.

[11] Guo jie Li and Benjamin W. Wah. The design of optimal systolic algorithms. *IEEE Trans. on Computers*, C-34(1):66–77, 1985.

[12] Richard M. Karp, Richard E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14:563–590, 1967.

[13] H.-T. Kung. Why systolic architectures. *Computer*, 15(1):37–45, Jan 1982.

[14] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, 1988.

[15] Peizong Lee and Zvi Meir Kedem. Synthesizing linear array algorithms from nested for loop algorithms. *IEEE Trans. on Computers*, 37(12):1578–1598, Dec. 1988.

[16] Basile Louka and Maurice Tchuente. An optimal solution for Gauss-Jordon elimination on 2D systolic arrays. In John V. McCanny, John McWhirter, and Earl E. Swartzlander Jr., editors, *Systolic Array Processors*, pages 264–274, Killarney, IRELAND, May 1989. Prentice-Hall.

[17] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley Publishing Co., Menlo Park, CA, 1980.

[18] Dan I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. Comput.*, C-31:1121–1126, Nov. 1982.

[19] Dan I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, 71(1):113–120, Jan. 1983.

[20] Nikolay Petkov. *Systolische Algorithmen und Arrays*. Akademie–Verlag, Berlin, 1989.

[21] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc. 11th Ann. Symp. on Computer Architecture*, pages 208–214, 1984.

[22] Sanjay V. Rajopadhye and Richard M. Fujimoto. Systolic array synthesis by static analysis of program dependencies. In J. W. DeBakker, A. J. Nijman, and P. C. Treleaven, editors, *Lecture Notes in Computer Science*, number 258: Parallel architectures and languages, Europe, pages 295–310. Springer Verlag, June 1987.

[23] Sanjay V. Rajopadhye, S. Purushothaman, and Richard M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In K. V. Nori, editor, *Lecture Notes in Computer Science*, number 241: Foundations of Software Technology and Theoretical Computer Science, pages 485–503. Springer Verlag, December 1986.

[24] Sailash K. Rao. *Regular Iterative Algorithms and Their Implementation on Processor Arrays*. PhD thesis, Stanford University, October 1985.

[25] Günter Rote. A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion). *Computing*, 34(3):191–219, 1985.

[26] Chris Scheiman and Peter R. Cappello. A processor-time minimal systolic array for transitive closure. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 19–31, Princeton, September 1990. IEEE Computer Society.

[27] Chris Scheiman and Peter R. Cappello. A period-processor-time minimal systolic array for cubical mesh algorithms. Technical Report CS91-4, UC, Santa Barbara, Santa Barbara, March 1991.

[28] Weijia Shang and José A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. In *Int. Conf. on Systolic Arrays*, pages 393–402, San Diego, May 1988.

[29] Yiwan Wong and Jean-Marc Delosme. Optimization of computation time for systolic arrays. Dept. of Computer Sci. RR-651, Yale Univ., May 1989.

[30] Yiwan Wong and Jean-Marc Delosme. Optimization of processor count for systolic arrays. Dept. of Computer Sci. RR-697, Yale Univ., May 1989.

[31] Xiaoxiong Zhong and Sanjay V. Rajopadhye. Deriving fully efficient systolic arrays by quasi-linear allocation functions. In *Parallel Architectures and Languages, Europe*, Eindhoven, the Netherlands, June 1991. Springer Verlag. extended version submitted to J. VLSI Signal Processing.