The Machine Paradigm of Xputers: and its Application to Digital Signal Processing Acceleration

by

R. W. Hartenstein, A. G. Hirschbiel, M. Weber
Universitaet Kaiserslautern, Postfach 3049,
D - 6750 Kaiserslautern, F. R. G.

category: hardware-oriented or other papers

key words:
- Impact of VLSI to parallel processor architecture
- parallel / distributed logic circuits
- processor-memory interconnections
- Compilers

Abstract. The paper gives an introduction to xputer use for acceleration of digital signal processing applications. Xputers are a novel class of high performance processors, the operation of which is deterministically data-driven. The innovative high performance machine principles of **x**puters efficiently support innovative compilation techniques for much better optimization of parallelism than possible on (conventional) **com**puters. These compilation techniques are mainly derived from compilation methods for ASAPs (application-specific array processors). The programming paradigm, which stems from the deterministically data-driven xputer machine paradigm, is illustrated by introducing a few xputer application examples in

digital signal processing.

1. Introduction

Xputers are a novel class of non-von Neumann universal processors, the principles and application development support tools of which have been described somewhere else [HHW89, Kil89, HHW90]. With respect to cost / performance ratio xputers are highly superior to (von Neumann) computers in a large class of algorithms within very important application areas such as digital signal processing, image processing, speech, radar, sonar, seismic and other kinds of processing such as e. g. layout verification and routing in EDA (electronics design automation) [HHW89, HHW90]. Xputers are superior also in many important applications, which require extraordinarily high throughput - up to the range of (von Neumann type) kilo-MIPS - at very low hardware cost. The superiority of xputers has several reasons:

- Xputers' fundamental operation principles are based on *data sequencing* with only *sparse control*, so that xputers are deterministically data-driven and most known sources of overhead are avoided from beginning.
- Xputer hardware supports very fine granularity parallelism (also below instruction level: data path level or gate level) in a such way that internal communication mechanisms are cheaper and more simple, and, by orders of magnitude more powerful, than known from all kinds of parallel computer systems.
- Xputer hardware is substantially more compiler-friendly and flexible than any kind of other computer or processor hardware in such a way, that dramatically better optimization (parallelization) strategies (based on highly detailed *data scheduling*) may be used by *xpilers* ("compilers" for xputers) than by (conventional) compilers.

This paper introduces the implementation of digital signal processing algorithms on xputers by means of some examples which also illustrate the xputer programming paradigm and its basic machine paradigm. Then some experimental performance figures and hardware expense figures will be shown. For introduction the following section briefly highlights xputer principles and those features which cause the high superiority of xputers.

2. Xputer Principles

For illustration fig. 1 compares the basic structure of **com**puters (fig. a) with the structure of **x**puters (b). The ALU of computers is a very narrow bandwidth device: it can carry out only a single simple operation at a time. Xputers, however, use a reconfigurable *r-ALU*, which may be configured such, that several sets of highly parallel data paths form powerful *compound operators* which need only a single basic clock cycle (a few nanoseconds only). Since their combinational machine code is loaded directly into the r-ALU xputers do not have a program store nor an instruction sequencer. That's why xputers are data-driven [Mch90]. But unlike data flow ma-

chines (which operate indeterministically), xputers feature fully deterministic principles of operation which we call *data sequencing*.

Xputers avoid most of the well-kown von Neumann bottlenecks except one: access to primary memory (being a data memory only) is still sequential. This one, however, has been greatly reduced because the highly optimizing "compilation" by *Xpilers* ("compilers" for xputers) achieve drastically better optimization results than compilers for conventional parallel processing systems [Mch90]. That's because the hardware of xputers is much more compiler-friendly (than that of computers), so that much more powerful optimization strategies can be mapped onto the (xputer) hardware.

2.1 The reconfigurable ALU

Xputers do not have a hardwired instruction set. An xputer's set of *compound operators,* which may be dramatically more powerful than "instructions" of computers, is defined by an xpiler (an innovative kind of compiler). That's why xputers have a reconfigurable *r-ALU* based on the use of path-programmable PLD-like circuitry (*PLD* stands for *programmable logic device*).

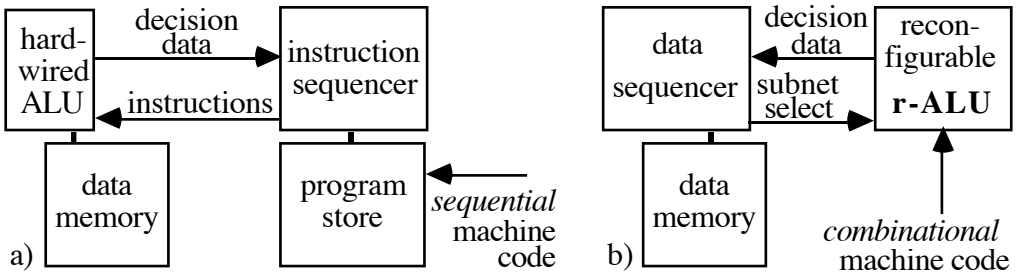


Fig. 1: Basic structures of: a) von Neumann (Harvard) machine and b) xputers.

PLDs, a special kind of ICs (integrated circuits), are available from about 20 vendors of a billion US-dollar world market (see fig.13). Path-programmability is the reason, why xputer machine code is not sequential code (known from computers), but *combinational machine code*. Many PLD types support dynamic reconfiguration of the r-ALU, i. e. new compound operators may be configured at run time. Several compound operators may reside simultaneously in a r-ALU by configuring a separate r-ALU *subnet* for each compound operator, which may be selected by a subnet select code inputted to the r-ALU (fig. 1 b or "SCC" in fig. 2 a).

2.2 The Data (Window) Scan Cache -

an Innovative Processor-to-Memory Interface

Xputers have - in contrast to computers - a completely different processor-to-memory interface which efficiently supports much better exploitation of parallelism within algorithms, than possible with any other kind of parallel computer systems. We call such an interface a *data scan cache* or *window scan cache,* or briefly: *scan cache*. This innovative interface also supports two- or more-dimensional memory organization. For simplicity and comprehensibility all examples within this paper are based on 2-dimensional memory organization.

A scan cache is a hardwired window to the memory space, which will be illustrated here for the 2-dimensional case (fig. 2 b). The scan cache format is electrically adjustable at run time. Fig. 2 b shows a few scan cache format examples, where the 1 by 1 format is a trivial case (the cache holds only a single memory word). A scan cache is very tightly connected to the r-ALU. This fully parallel bidirectional interconnect uses fully dedicated direct wiring (no bus! [HaKo75]) between the r-ALU and each bit of each word held by the cache. In designing xputer architectures with extraordinarily large r-ALU resources this sometimes causes wiring congestion [HHW90]. During redesign efforts we have experienced, that such problems can be solved by clever partitioning schemes [HHW90], or, even systematically.

To achieve substantially higher throughput xputers may have *multiple scan caches,* i. e. several such window caches may connect an xputer processor to the same primary memory (fig. 2 a). This will be illustrated by section 3.2. We call such interfaces a cache, since for many important applications its concept efficiently supports such throughput improvement methods, where the scheduling strategy of the xpiler can achieve, that the right data are at the right time at the right place (for illustration of such a very fine granularity *data scheduling* also see section 2.3). That's why scan caches support *deterministic caching* strategies which are substantially better than traditional cache philosophies being probabilistic (i. e. are based on gambling [AlGo09]).

Scan caches usually are much smaller than traditional caches, and have an internal multi-dimensional shift mechanism to minimize primary memory access rate at least during local cache movements ([HHW88], for an example see fig. 3 c). Due to the concept of very fine granularity data scheduling supported by this kind of memory interface it is relatively easy to efficiently explore interleaved memory access organizations in order to substantially accelerate processor-to-memory traffic.

2.3 Data Sequencing and Data Schedules

Xputers are data-driven [Mch90], since the r-ALU is not controlled by an instruction sequencer.

That's why xputers have a data-driven *data sequencer* (see fig. 1 b). But unlike data flow machines (which operate indeterministically), xputers feature fully deterministic principles of operation which we call *data sequencing*. It will be illustrated, that such data sequencing may be programmed by very fine granularity data schedules, which are an excellent basis for using highly effective throughput optimization strategies within xpilers.

Such data sequencers feature hardwired *data scan sequences* (which we also call *scan patterns*) making the location of a scan cache travel step by step or jump by jump through the (1-,) 2- (, or n-)dimensional data memory space. Examples of such *data scans* are [HHW88, HHW89, Kil89]: *single scan steps* such as e. g. *data goto, data jump, data next* (goto one of the nearest neighbors) as well as longer generic *data scan sequences* , such as for instance *video scan sequence, shuffle sequence, trellis sequence* and others.

In most cases stepping through such a generic scan pattern (e.g. a video scan) the compound function being activated within the r-ALU remains the same. Such a data-driven sequence we call a *data loop*. Several such scan sequences may be linked to form a compound data scan sequence which we call a *scan task*. This is achieved because xputers have feed back loop inter-

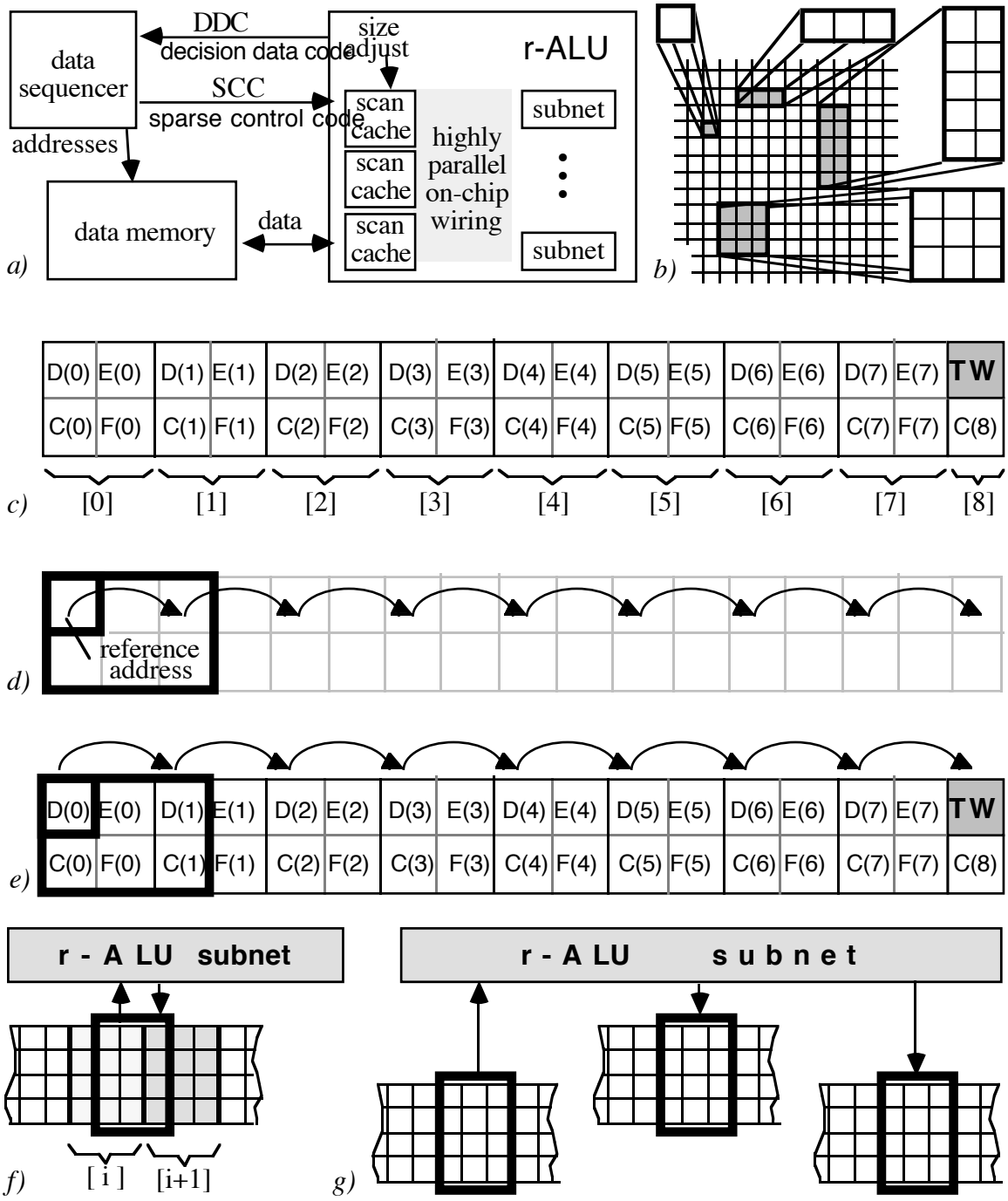


Fig. 2. Data (window) scan cache and data schedule: a) the role of scan caches within xputers , b) illustrating the scan cache as a window to data memory and several examples of window size adjustments, c) example of a data map (from fig. 3), d) its "shuttle" which is a 2 by 3 scan cache adjustment associated with a linear scan pattern (8 steps to the right: step width = 2), e) a data schedule example (derived from figs. c and d), f) local tight communication by single cache overlapping window, g) global tight communication by multiple caches.

connect between we call a *scan task*. This is achieved because xputers have feed back loop in-

terconnect between r-ALU and data sequencer (fig. 1 b) similar to that between controller and ALU in computers (fig. 1 a): A *subnet select code* fed into the r-ALU (for an example see fig. 2 a) may evoke switching between different compound functions of the r-ALU. A *tagged control word* having been inserted into the data map (see TW in fig. 3 e) may be recognized and decoded within the r-ALU (see "tag check ..." in fig. 3 d) to derive a suitable *DDC* code (see fig. 2 a) to select the next scan pattern.
For data dependent scan sequences or scan tasks, such as e.g. in curve following (image processing [HHW88], circuit extraction [HHW89], Lee routing [HHW90] and other applications), *decision data* generated by the r-ALU may influence the data sequencer operation (see fig. 1 b, or "DDC" in fig. 2 a). In xputers this feed back loop is not hardwired, but may be reconfigured with the r-ALU.

2.4 The Data Scheduling Paradigm

The novel machine paradigm of xputers is the basis of the highly successful parallelization strategies thus applicable to xpilers [HHW89, Mch90]. The essential concept of this paradigm we call *data scheduling*, because it takes care, that during computation the r-ALU finds the right data at the right time at the right place. We call this data *scheduling*, because by this structuring of data the execution order is almost completely determined at compile time - in contrast to data flow machines, where the order of execution is determined at run time.

Let us illustrate this by means of an example data schedule, derived from the algorithm example in fig. 3. For this example the data schedule (fig. 2 e) consists of a data map (fig. 2 c), and a *shuttle* (fig. 2 d). A shuttle (fig. 2 d) consists of 3 items: (1:) a cache format adjustment (cache no. 1 :: 2 by 3), (2:) a starting location for this cache (fig. 2 d/e), and, (3:) a scan pattern being associated to this cache and this starting location (8 steps to the right where step width is 2, see fig. 2 d/e).

To get a more clear view on the data scheduling paradigm we metaphorically say, that data communicate with each other through the processor (i. e., we do not say it the usual way: where "processors communicate with each other" by data). Within the above example, for instance (see fig. 2 f), the data subarray [i] communicates with the data subarray [i+1]. This is local communication (nearest neighbor communication) by overlapping scan window, similar to that used in the example from fig. 3 e, where c[i] (shaded area) is the communication field (during cache position [i] it is written, whereas at position [i+1] is is read).

A data schedule may also consist of a data map and several shuttles (examples in figs. 7 and 9), or even of several data maps and several shuttles. In this case the tight interconnect of several caches through the r-ALU (fig. 2 g) supports global communication: long distance data communication between different data locations being far apart from each other or between different data maps.

2.5 Universal Xputers

Xputers are as universal as computers for two reasons [HHW89, Mch90]. By a path for decision data sequencer and r-ALU are linked into a feed back loop (see fig. 1 b), similar as known from the computer (fig. 1 a). Also the highly flexible xputer concept of sparse control allows to insert tagged control words into data maps at any location desired (e. g. compare TW in fig. 3). Because of their universality xputers may run in stand-alone mode, since being as universal as computers. But co-processor use together with a **com**puter as a host has the advantage of smooth integration into existing application environments and of convenient utilization of contemporary programming and software engineering tools and environments - also for xpiler implementation [Kil89, Txf89].

3. Xputers in Digital Signal Processing

To meet real-time requirements for many digital signal processing (DSP) algorithms extraordinarily high throughput is needed. Many DSP algorithms have very high inherent parallelism based on highly regular data dependencies. Excellent methods to exploit this parallelism are have been developed for such algorithms with highly regular data dependencies. On this basis we could consider three fundamentally different approaches to achieve the high throughput needed: (1) using application-specific array processors (ASAPs), (2) using parallel computer systems, and: (3) using xputers.

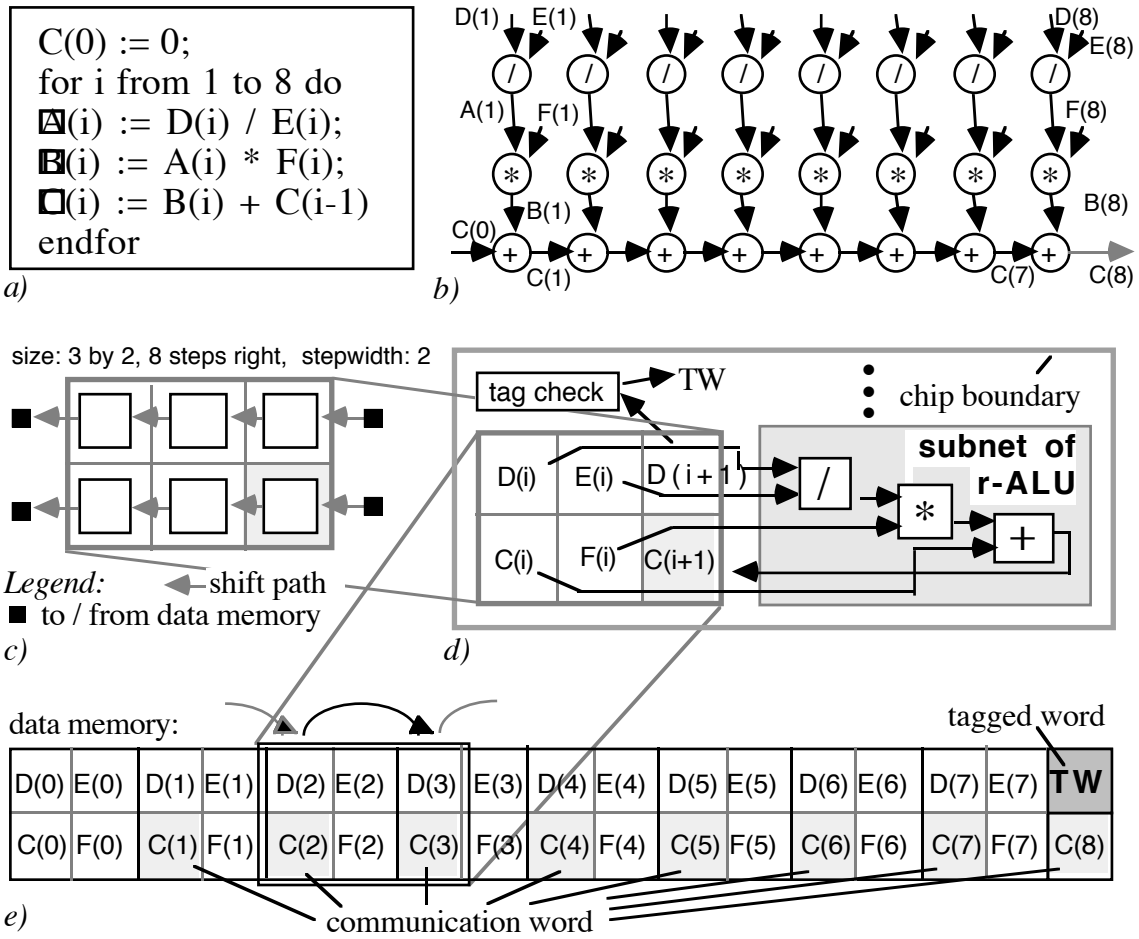


Fig. 3. Illustration of the xputer machine paradigm: a) an algorithm example, b) its data flow graph, c) its scan cache size adjustment, d) r-ALU subnet configuration used, e) data map used,

The main bottleneck in exploitation of parallelism is the communication between processors or processing elements (PEs), rather than the number of PEs. In ASAPs communication indigestion between PEs is avoided by locality: PEs are placed such, that only nearest neighbor communication is needed, so that interconnect between PEs is wired by abutment. These dedicated links are much faster than buses and avoid any overhead [HaKo75]. Efficient ASAP implementations are possible only for algorithms having local regular data dependencies (systolic algorithms). Also a huge overhead is caused by scrambling and unscrambling the data before or after processing. The ASIC design or full custom design needed for an ASAP implementation is very expensive. Programmable ASAPs also are very expensive and very complex, if available at all.

Programmable parallel computer systems at first glance seem to be sufficiently flexible also for efficient implementation of non-systolic DSP algorithms (having only globally regular data dependencies). A closer look and also practical experience reveals, that the communication mech-

anisms offered by this hardware suffer from huge overhead and do not support parallelism of sufficiently fine granularity (e. g. [Mch90]). So mostly only a small fraction of the parallelism can be utilized, which the number of available processors seems to offer. An exception are VLIW® architectures, which achieve a less disappointing degree of parallelism at least for systolic algorithms [Gaj81] (VLIW®: trade mark of Multiflow Computer Corp.).

That's because VLIW architectures use a lower level of parallelism (instruction level) than other parallel computer systems (process level parallelism). We strongly believe, that even lower level parallelism (below instruction level) is needed, to achieve the required very high throughput at very low hardware cost. A much higher flexibility of communication mechanisms is achieved by parallelism within the "ALU" at data path level or at gate level. To achieve machine programmability of parallelism at that level the only approach currently known is the concept of xputers with its programmable ("reconfigurable") r-ALU. The following sections illustrate this xputer programming paradigm by illustrating the implementation of a few DSP algorithms on xputers.

3.1 DSP algorithms with locally regular dependencies.

Two DSP algorithm examples will be used to illustrate using the xputer data scheduling paradigm for locally regular dependencies. It is typical for running such systolic algorithms on xputers, that only a single data scan cache is needed - in contrast to such algorithms which exhibit only globally regular dependencies (see section 3.2). For both examples also an estimation of hardware expense will be given in terms of gates needed for the r-ALU.

3.1.1 Digital Filtering

It is assumed, that the reader of this section is familiar with with 2-D filtering or image preprocessing. Many algorithms for image enhancement and digital filtering may be performed on a xputer using a 3 by 3 window data scan cache (see e. g. example coefficients in fig. 4 a). In xputer use a video scan pattern may be used to scan a 2-D pixel map, line by line.

By the xpiler a r-ALU subnet is configured into a compound operator which simultaneously reads all 9 pixels from the cache and writes back the result into the center pixel (c4) of the cache (fig. 4 b). Fig. 4 c shows the r-ALU configuration in detail. Since direct wiring is used (no bus !) this read/modify/write action exhibits fine granularity parallelism such, that only a single r-ALU cycle is needed (about 50 nsec or less). The straight-on r-ALU configuration solution, using predefined universal arithmetic modules needs about 60,000 gates, which easily fits on a single PLD chip. A minimized solution based on the restricted domain of weight constants in this example requires only about 700 gates (multiply by 2 or four just requires a shift left).

3.1.2 The Convolution Example

Another example of an algorithm with locally regular dependencies is the following operation (required for the convolution) which may be executed efficiently on the xputer is the computation of the inner product of the form:

$$Y(i+1) = Y(i) + X(i+1) * W(i)$$

In figure 5 a more detailed illustration is given including the r-ALU subnet configuration and the operational principle of the data sequencing. This time the r-ALU (not the data map) holds the weights as constants. The data scan cache performs a single line scan in single steps to the right:

for k = 1 step 1 to 9 do $Y[k] := \sum_{i=-2}^{2} W[i] * X[k-i]$ endfor

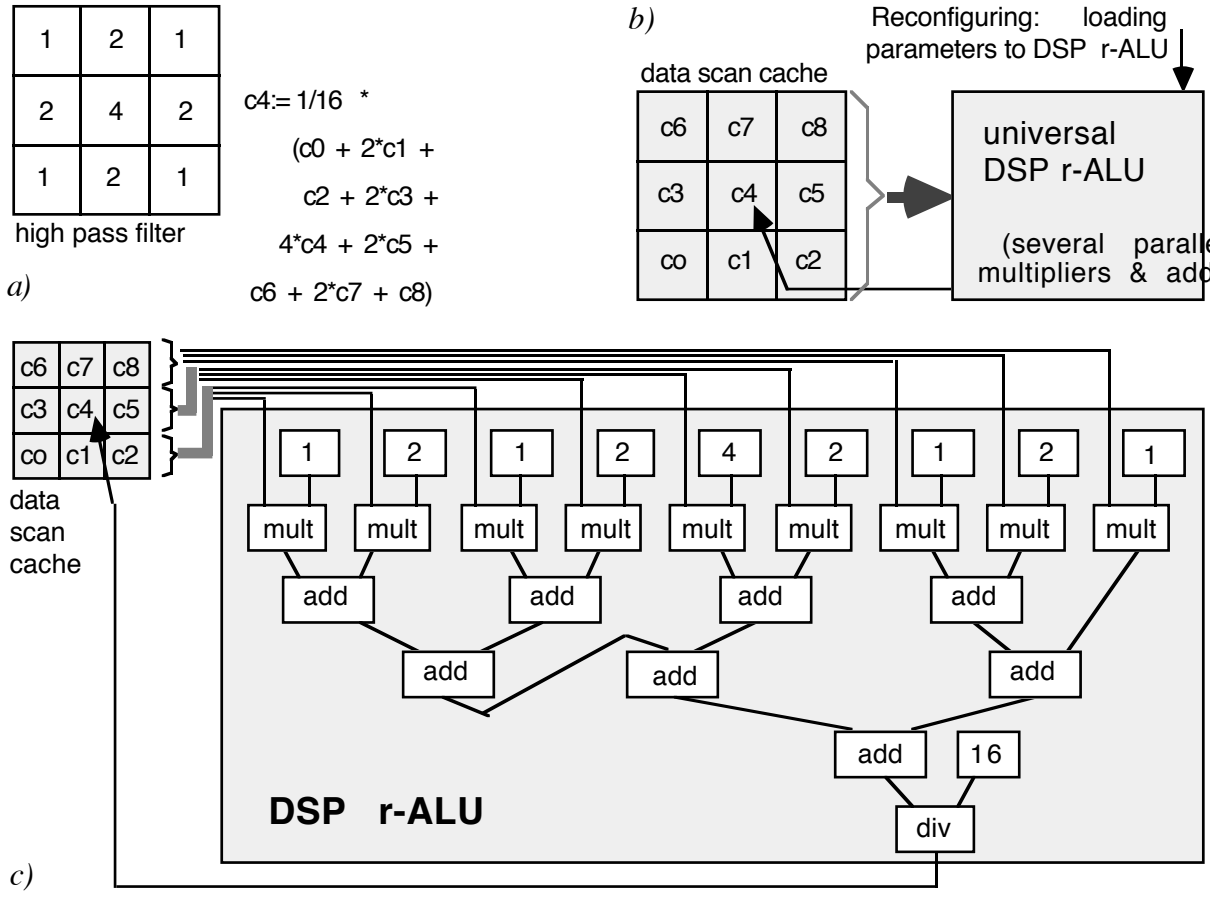


Fig. 4 . Two-dimensional filtering example: a) coefficients, b) basic r-ALU configuration

structure, c) detailed r-ALU subnet configuration.

Fig. 5 a illustrates this serialized operation. Fig. 5 b shows the xputer data map which combines the input data vector and the output data vector from fig. a into a data array. Together with its associated scan pattern this data map forms a data schedule. Fig. 5 c shows the r-ALU configuration tailored to this data schedule, and a suitable 2 by 5 scan cache size adjustment.

Note, that due to the very tight coupling (no bus ! [HaKo75]) between r-ALU and cache(s) the entire read / modify / write compound operation onto the current cache contents only needs a single clock cycle of a few nanoseconds only, what explains the high acceleration factors.

3.2 Algorithms with globally regular generic dependencies

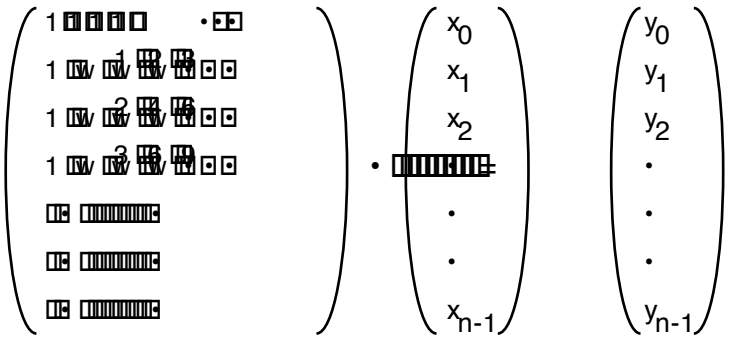
Two DSP example will be used to illustrate xputer use to implement algorithms with only globally regular data dependencies. In contrast to systolic ASAPs xputers can efficiently handle this by hardwired parametrized scan patterns using generic sequences of long distance data jumps, such as e. g. butterfly scans, shuffle scans etc. To achieve an optimum performance the use of multiple data scan caches moving simultaneously is highly useful, which will be illustrated by 3 examples. The examples used for illustration are two Fast Fourier Transform (FFT) algorithms and the Viterbi algorithm. It is assumed that the reader is familiar with these algorithms, so that no introduction nor explanation will be given.

3.2.1 Fast Fourier Transform (FFT)

Fast Fourier transform algorithms are examples of algorithms which exhibit only globally regular data dependencies. The n-point Fourier transform computes $(y_0, y_1, \ldots, y_{n-1})$ for given n input points $(x_0, x_1, \ldots, x_{n-1})$ such that y_i is defined by

$$y_i = x_0 w^{i(n-1)} + x_1 w^{i(n-2)} + \ldots + x_{n-1}$$

where w is a primitive n^{th} root of unity. This may be written as matrix vector multiplication:



The straightforward method to solve an n-point Fourier Transform, as shown above, requires $O(n^2)$ operations. An efficient algorithm for the Fourier transform is the Cooley-Tukey FFT [CoTu65] which solves the problem with O(n logn) operations.

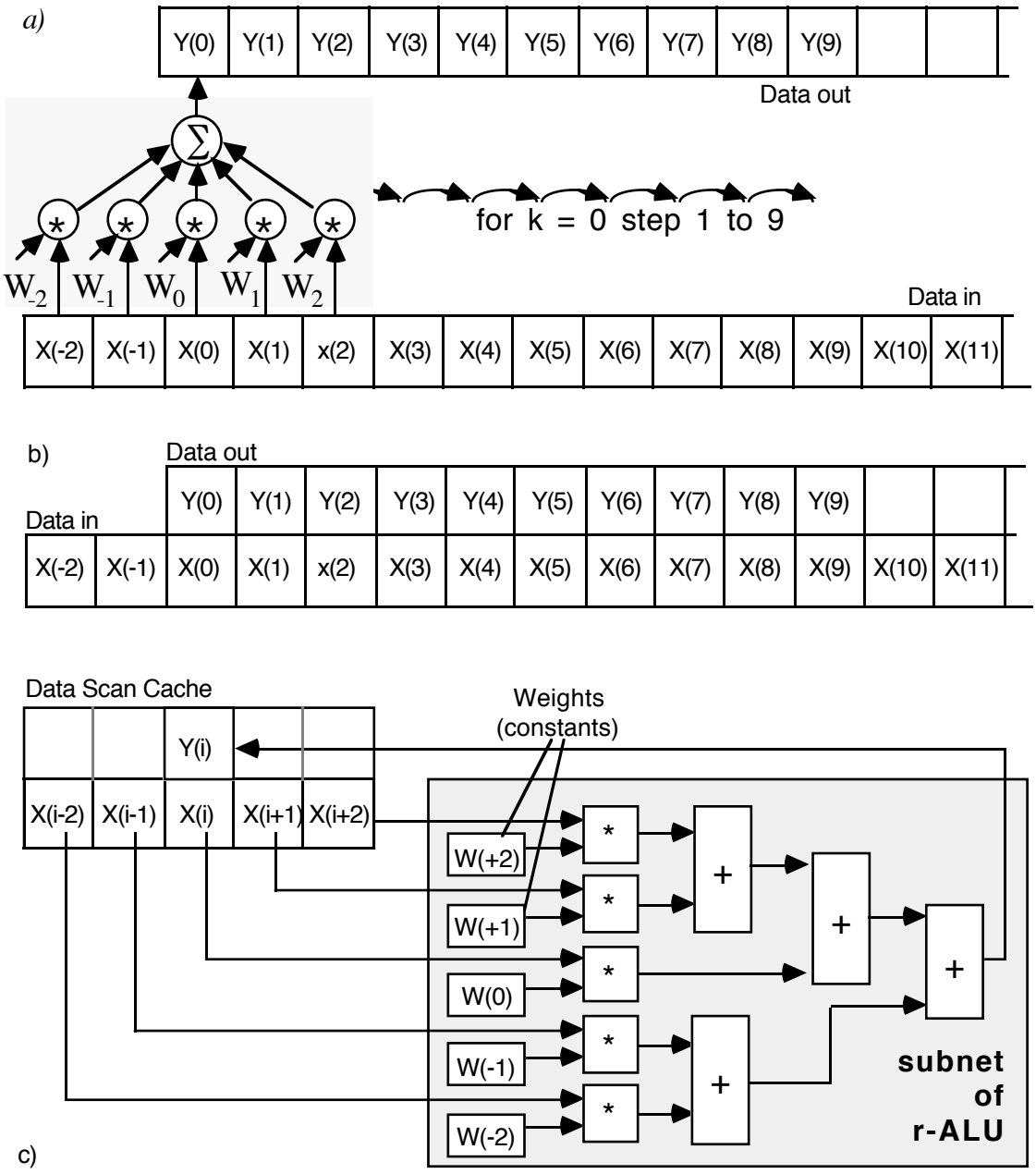


Fig. 5 . Convolution-like algorithms on xputers. a) illustrating the algorithm, b) its data map, c) its cache and r-ALU configuration.

FFT computing structures thus call for spatially global interconnect, and cannot be easily mapped onto systolic arrays [Kun84], since these permits only local interconnect between processing elements. VLSI solutions for FFT thus use a pipeline of processing elements the layers of which are connected to each other via butterfly wiring patterns [Swa87], which are expensive with respect to VLSI design objectives [Kun84].

3.2.2 The Cooley-Tukey FFT

Figure 6 illustrates the 16-point Cooley-Tukey FFT [HwBr84]. Storage locations are shown by white squares. Operations are shown by white circles. The weights are used by corresponding operation as factors for the multiplication within the operation. (The storage cells needed for the weights are also indicated by gray square areas.) For Xputer execution the storage locations shown in fig. 6 are directly mapped (1-to-1) into the data memory shown in figure 7 b.

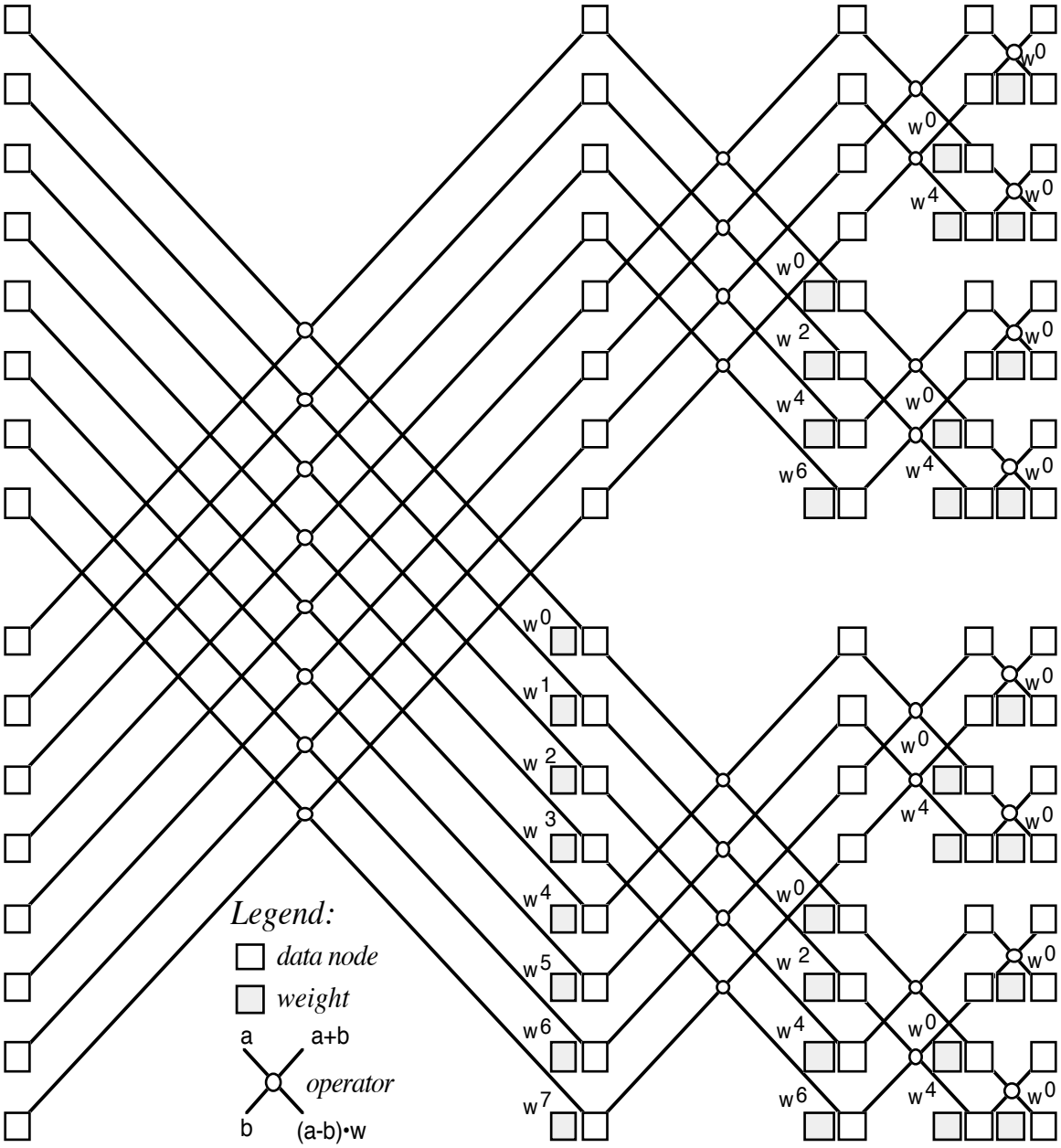


Figure 6. Signal flow graph of the Cooley-Tukey FFT and its mapping into xputer data map

To cover the long distance dependencies a two separate 1 by 1 scan caches are configured to be used in a sequence of four phases, where each phase needs a different scan pattern (compare fig. 7 b). During a particular phase both caches use the same scan pattern, which however is started from different starting points (compare fig. 7 b). Fig. 7 a shows the r-ALU configuration for this example and fig.7 b shows its data schedule. A sequence of 4 different scan patterns is needed for the scan task. The first scan pattern (the leftmost one in fig. 7 b) is just an 8 step linear scan pattern (step width = 1) applied to both caches simultaneously, but with different starting locations. The other 3 scan patterns within this sequence are 2-level nested scan patterns, where each higher level step goes into the starting point of a lower level scan pattern.

3.3.3 Constant-geometry Version of the FFT

The constant-geometry version of the FFT algorithm allows the same data shuffling to be used for all the stages [RaGo75]. This is depicted in figure 7 for a 16-point FFT. The operations are again represented by white dots. The weight by each of these operations indicates that this weight is used as a factor for the multiplication within the operation.

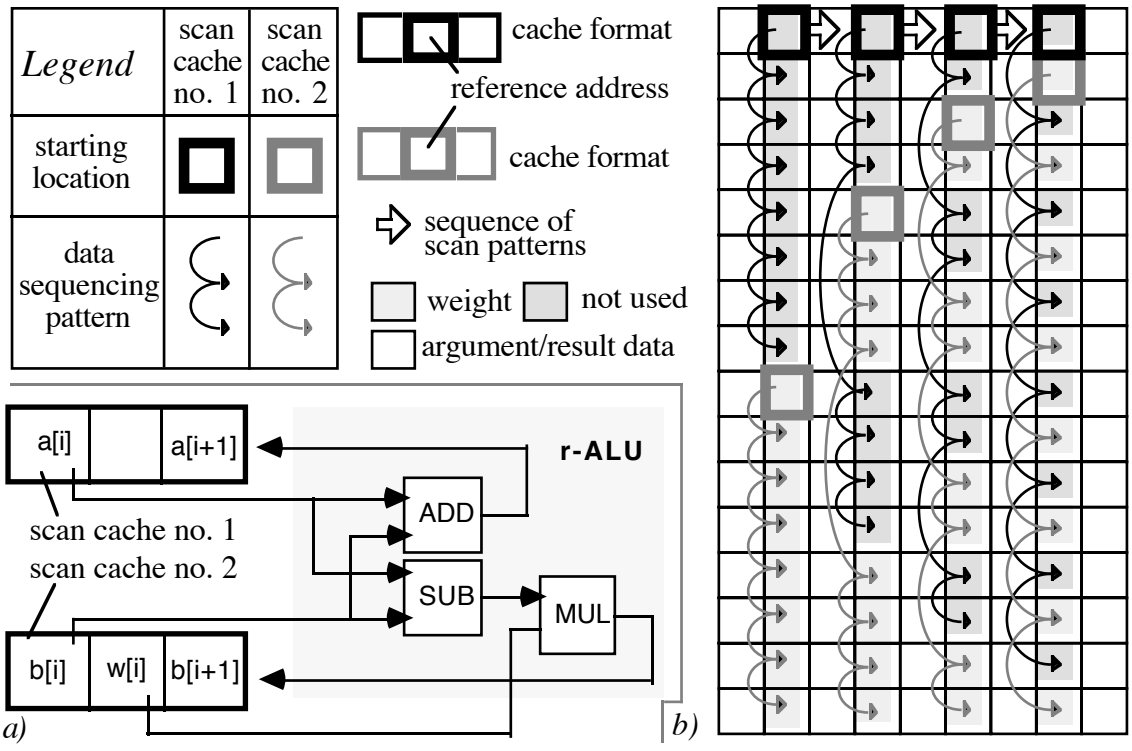


Figure 7. Illustrating the xputer implementation of example in fig. 6: a) r-ALU subnet configuration, b) its data schedule (using a sequence of four double cache scan patterns)

For Xputer execution one cache is used to provide the input values for each operation, while a spatially distributed output cache is used to accept the results. The r-ALU has to perform the operations "a + w·b" and "a - w·b" as a compound function. Figure 8 shows the data paths from the input cache via the r-ALU to the output cache.

The data map is shown in fig. 9 a. To perform one stage of the constant-geometry FFT the input cache is moved down two positions, while simultaneously the output cache is moved down by one position, starting at the top of the segment until the bottom is reached. At each position in this loop the r-ALU is activated to produce intermediate results of the FFT, which are stored in the memory via the output cache. When the last two input values have been reached by the moving caches, the second stage of the FFT algorithm is started by placing both caches again at the top of the memory segment, but two words to the right. The results of the previous stage are now the input to the next iteration. Again the caches are moved downwards to perform another stage of the FFT. In all (log n) of these iterations are necessary to complete the FFT algorithm (figure 9).

Also an implementation is possible where the distribution of the two results delivered by the r-ALU is done using a shuffle move pattern for a single word output cache instead of distributing the output cache itself. For this solution two different subnets are configured into the r-ALU which can be selected by the subnet select code (figure 10).

3.3.2 The Viterbi algorithm on xputers

The following section has been written for readers being familiar with the Viterbi algorithm [For73]. To illustrate xputer application we use the simple example in fig. A.

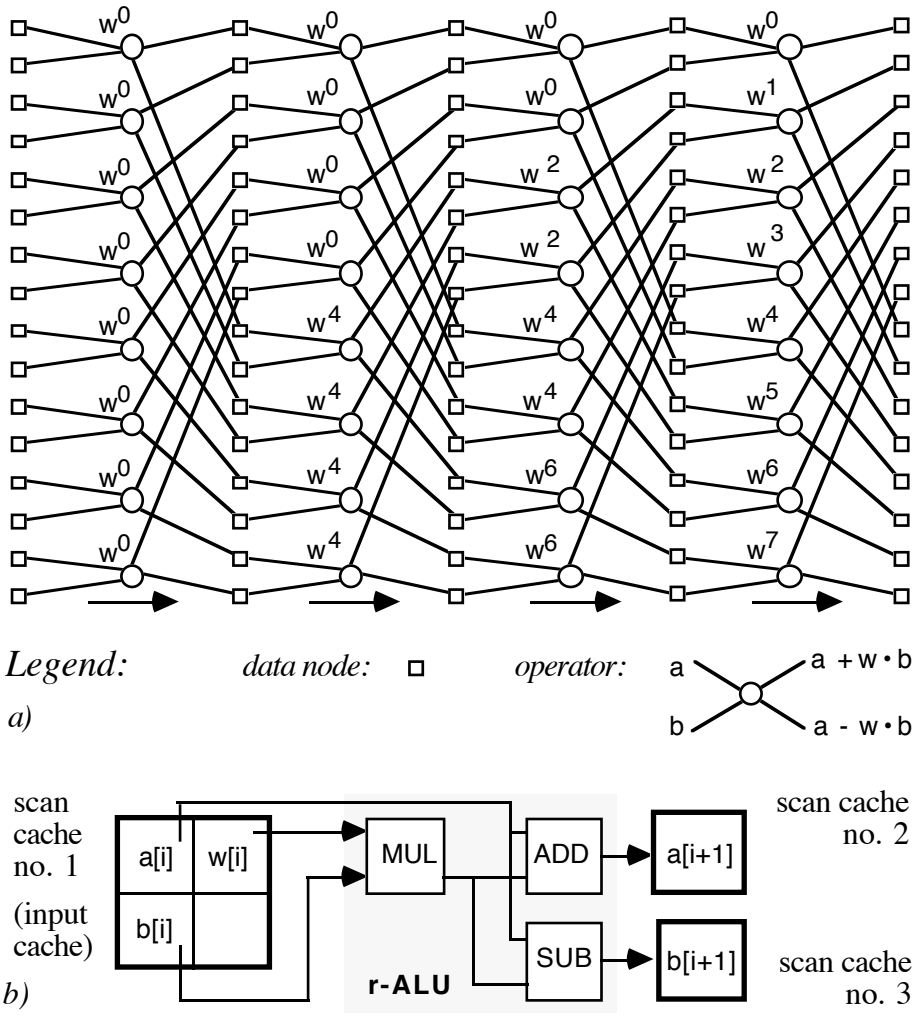


Fig. 8. Constant-geometry version of the example from fig. 8: a) signal flow graph, b) r-ALU subnet configuration

To carry out decoding and maximum likelihood detection we have to transform the trellis diagram (see fig. A b) into a memory map. For the transition from time step k to k+1 eight computations corresponding to the branches of the trellis are needed, here carried out in four cycles, as shown in fig. B.

Each step of the chosen sequence of computations involves three active data elements. Those on the left side of each step have been compounded together in one input cache and the other active data element on the right side represents an output cache. The order of these four steps facilitates the sequence of movements of the caches involved. As can be clearly seen the output cache moves at twice the speed of the input cache. Such a sequence of movements can be easily expressed with nested loops in the MOPL input language.

Decoding the Viterbi algorithm contains two tasks, first the computing of path weights and sec-

ond the determination of the shortest path through the trellis. The path weight of a node in time step k+1 is computed as the minimum of all the weights of paths leading to this node. The path weights of all nodes of time step k are known. To compute the weight of a path leading to a

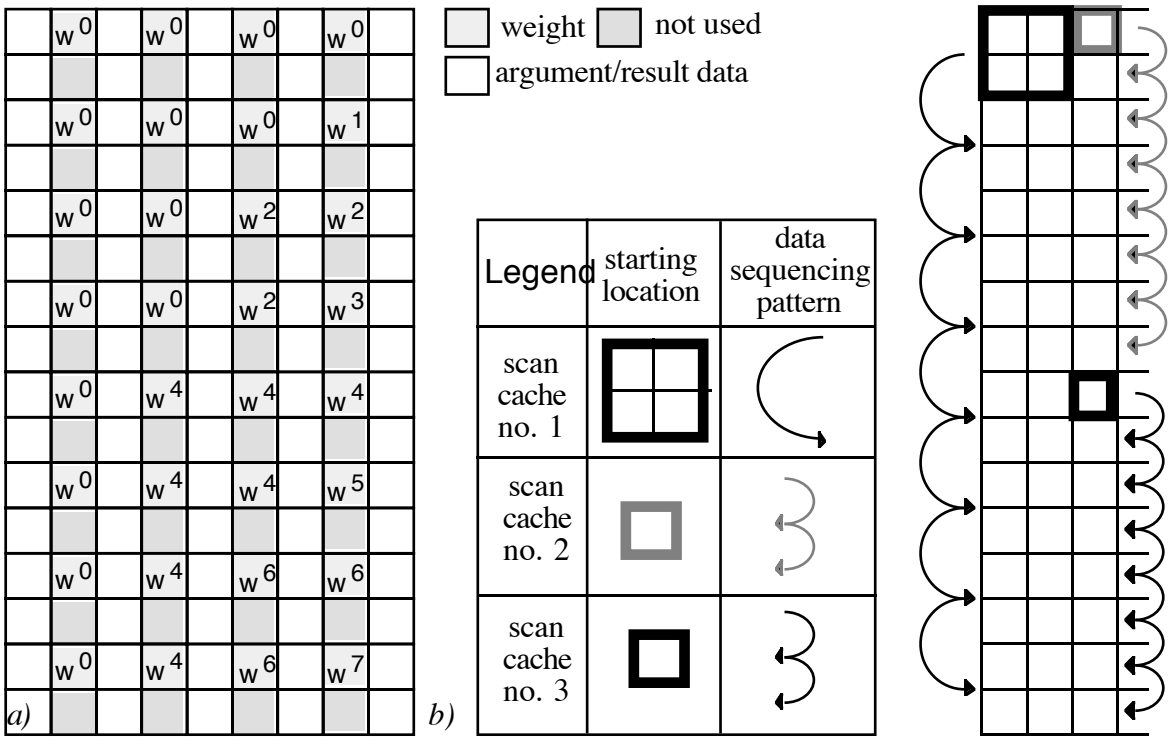


Fig. 9. Data map (a) and (first phase) data scan pattern (b) of the example in fig. 8 - note, that the scan pattern (b) will be carried out four times, in steps 2 locations to the right

node at time step k+1 we need to know the weights of the branches. In our example the weight of a branch is determined by the hamming distance between the decoder input and the expected symbol for a transition via the branch in question (for further discussion we call the expected symbol 'value'). Up to now we have already identified three of five variables needed for the computation in each stage. These variables are the decoder input, the expected symbol (value) and the sum of the weights for each node (compare figure 5).

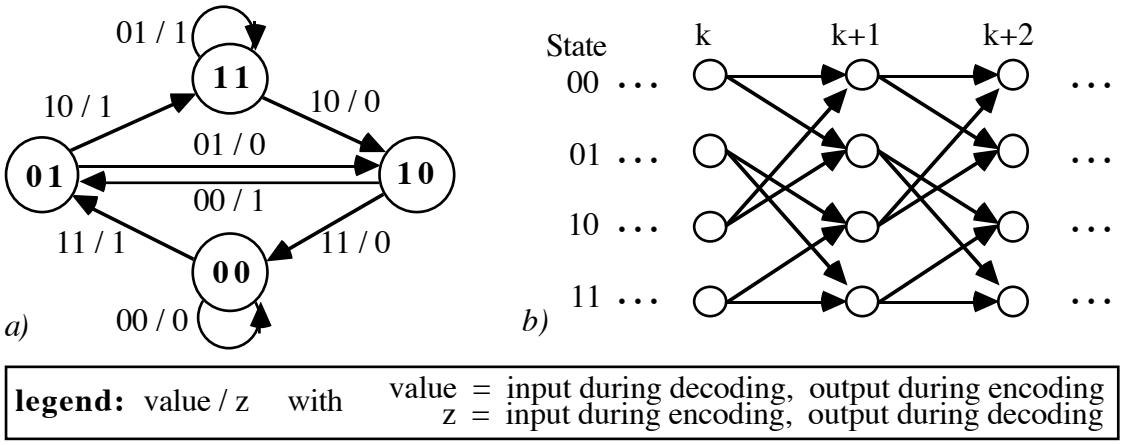


Fig. 10. Viterbi example used for illustration; a) state graph, b) illustration of its trellis diagram

The computation of the weights of all paths at each time step is performed by traversing the trellis from left to right. Each transition from time step k to k+1 is subdivided in the above mentioned subtasks (figure 4). To facilitate the reconstruction of the shortest path in the second phase by traversing the trellis from right to left, the address of the predecessor in this shortest path is stored together with the other information for each node. This is already done in the first phase and thus the reconstruction does not need any further time consuming computations. The decoded sequence with the highest probability is found during the second phase simply by delivering the contents of the field 'decoder output'. This sequence is in reversed order but can easily be rearranged by using a LIFO storage.

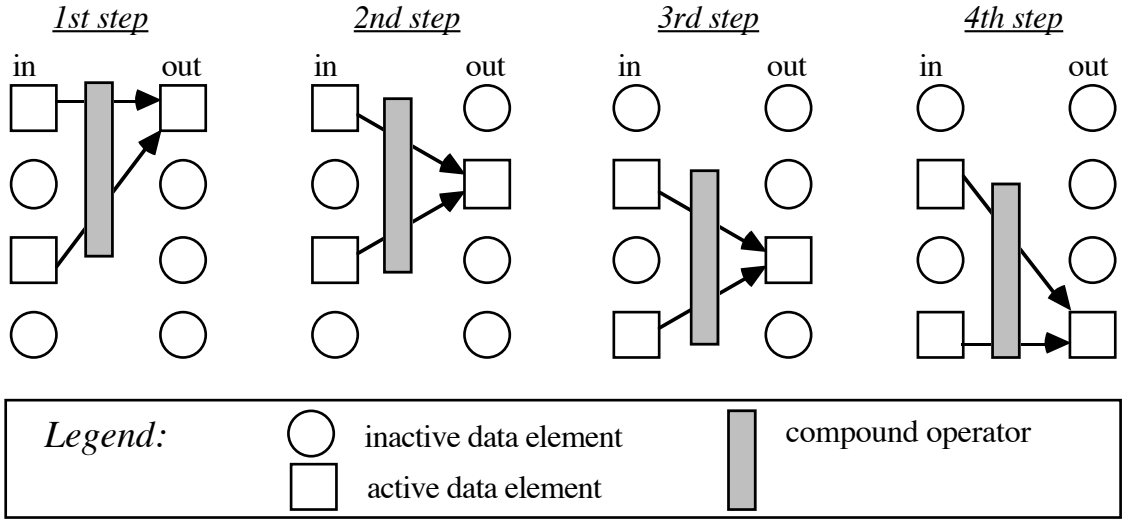


Fig. 11: Data schedule illustration for example in fig. 10.

On an Xputer the operations mentioned above are performed using a customized r-ALU. In figure

6 we show how the input cache containing the values of the two active elements (represented by the two rows) of time step k are used to compute the new values of the output cache of time step k+1. Figure 7 specifies the configuration of the r-ALU for the specified task.

4. Acceleration factors having been achieved by xputers

MIPs have been defined as a measure of computer performance. Since xputers do not have "instructions" MIPs would not be an adequate measure of xputer performance. Since a single xputer compound operator could be equivalent to up to hundreds of instructions (e. g. see [HHW90]) it would be highly unfair to force it into line with a computer instruction. Since being highly technology-driven MIPs are not a durable measure, nor a suitable measure to evaluate the merits of particular architectures.

Our main goal is it, not to measure the performance of the latest processor design using the most advanced technology, but to prove and to measure the superiority of the fundamental principles of xputers over those of von Neumann principles in a technology-independent way. We believe this to be fair, since both, future computers and future xputers will benefit from future progress of technology. That's why we prefer to use the notion of *acceleration factor* to measure the relative benefit of xputer use over the use of a computer of comparative technology.

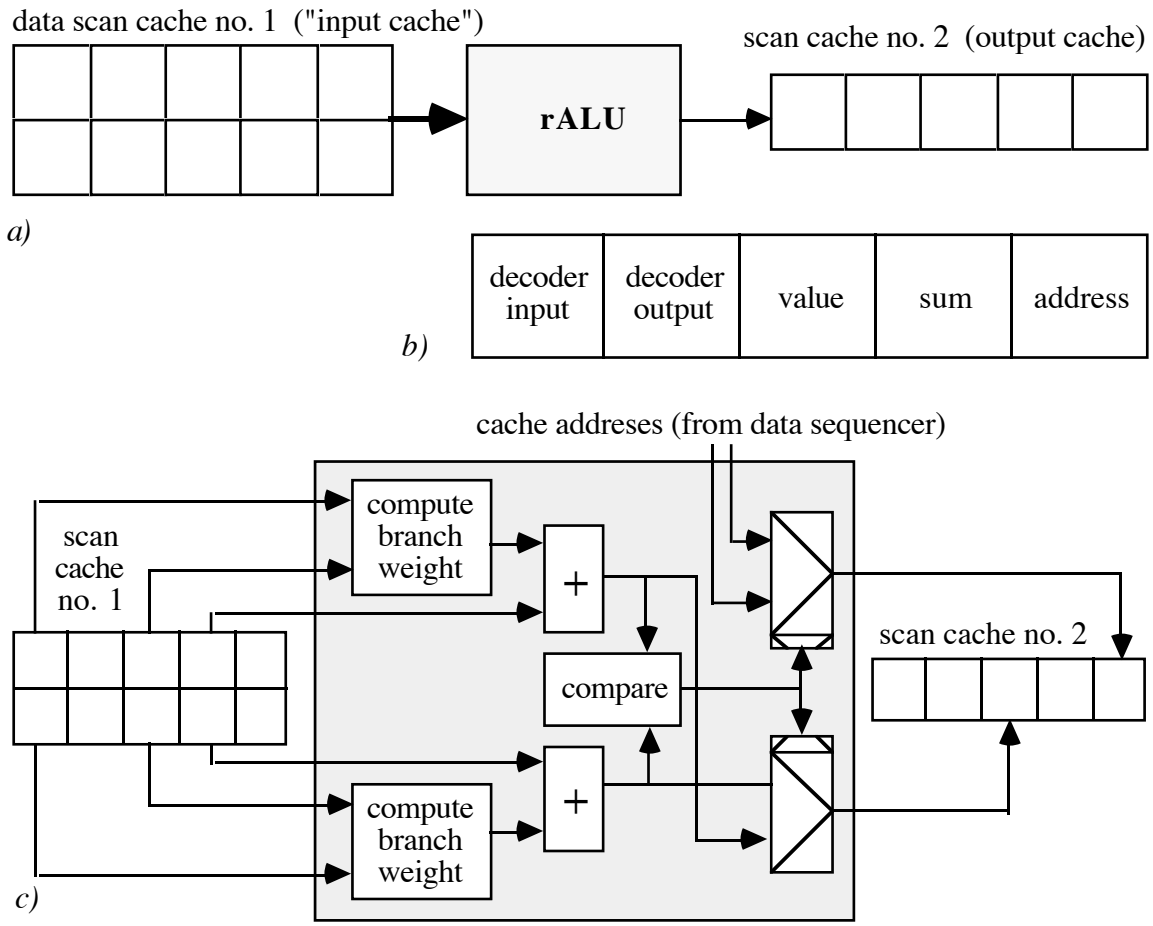


Fig. 12: Cache configuration for the example in fig. 10

For performance evaluation we have implemented a number of algorithms on the MoM xputer architecture [HHW89]. To obtain acceleration factors we used two different approaches: an experimental approach and a calculated estimation approach. Our experimental approach compares the "CPU time" needed by a VAX 11/750 or a Motorola 68 020 to that needed by a first generation MoM hardware [HHW86] having been set up using technology of the early and mid' 80s (a mix of TTL-based breadboards and 3 micron nMOS integrated circuits, see fig. 14). The calculated estimation approach compares the total number of primary memory access cycles needed. Fig. 13 a gives a survey on acceleration factors obtained this way so far, including 2-D digital filtering (>300) having been illustrated within this paper. Acceleration factors for the other DSP examples within this paper (FFT, convolution and Viterbi algorithm) will be available by end of March 1990. (We expect figures of at least >100.)

5. Discussion of Technology Issues

The r-ALU resource of xputers is electrically path-programmable quite similar to a very large EPLA, where the capacity may be measured by the number of terms or gates which can be "programmed". Xputer architecture families are extensible by adding more "*PLA space*" to the r-ALUs so that more terms can be programmed. When integrated PLD circuits are used from commercial sources (a billion US-dollars market world-wide) the number of ICs (integrated circuits) needed for a particular application is a rough measure of r-ALU hardware cost. A good estimate for having several applications reside within a single r-ALU simultaneously is the total sum of the numbers of integrated circuits needed for the individual applications.

algorithm	*a)* acceleration factor	*b)* no. of chips needed for functional part of r-ALU								
		DPLA KL'86	Altera EP900 intel 5C090	Altera EP1800 intel 5C180	Altera MAX EPM 5128	Xilinx XC 3090	Plessey ERA 60100	Xilinx LCA XC* 3200	Plessey ERA* 60400	DPLA KL'90
	**	700	900	1800	2800	9000	10,000	20,000	40,000	40,000
design rule [1] check(CMOS)	>2000	254	190	96	45	18	16	8	4	4
electrical rules check [1]	>300	24	20	10	5	2	2	1	< 1	< 1
Lee routing [1]	>160	24	20	10	5	2	2	1	< 1	< 1
shuffle sort [2]	80	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1
matrix mult. [2] • single cache	>9	28	24	12	7	3	2	1	< 1	< 1
• double cache	150									
FFT [3]		70	60	30	18	6	6	3	2	2
Viterbi [4]		< 1	< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1
Image Prepr. [5]	>300									
• straight-on			8	4	3	< 1	< 1	< 1	<< 1	<< 1
• minimized		1	< 1	< 1	<< 1	<< 1	<< 1	<< 1	<< 1	<< 1
Convolution [6]		36	28	14	9	3	3	2	1	1

[1]) [HHW90] [2]) [HHW89] [3, 4, 5, 6]) see this paper **) gate equivalents *) announced for

Fig. 13. Some xputer acceleration factor performance figures (a) , and (b) r-ALU hardware cost examples (in terms of number of PLD integrated circuits needed)

Some applications, like design rule check (DRC, compare fig. 13 b) [HHW90] are critical w. r. to r-ALU capacity. (This application is very interesting, since acceleration factors of >2000 have bean achieved for it experimentally [HHW89]). That's why the following discussion focuses on

DRC. Technology is developing fast which results in rapidly decreasing PLD chip count (see fig. 13 b). The first generation MoM xputer architecture at Kaiserslautern (fig. 14) has used Technology of the mid'80es. First the intel 5C180 IC has been used (second source: Altera). For the DRC example 96 of them have been needed (column 5 in fig. 7), which fit onto 2 double Europe size PCBs. In using the Plessey ERA 60400, however, only 4 chips would be needed for the same application.

For many other important applications, however, meanwhile several applications' r-ALU subnets would fit onto a single PLD chip (compare fig. 13 b). Since being very simple (much more simple than a RISC processor) also the (non-programmable) standard xputer parts conveniently would fit onto a single chip, provided PLD technology is accessible to the designer. This definitely means, that in using advanced technology available to-day single chip solutions in DSP and DRC acceleration are possible now which can replace conventional kiloMIPS hardware.
For programming with early PLD types each IC had to be inserted into a piece of programmer equipment, which takes many minutes. That's why to achieve dynamic programmability (which needs milliseconds) a special *DPLA* (dynamically programmable logic array) has been designed at Kaiserslautern [May89] using 3 micron nMOS technology, where only limited chip size has been available. This DPLA-KL'86 may be dynamically programmed - also at run time, which needs only a few milliseconds or microseconds. It has been fabricated by Fraunhofer-IMS Duisburg (F.R.G.), and successfully tested, and, has been used within the first generation MoM hardware at Kaiserslautern [HHW88, HHW89].

MoM stands for "Map-oriented Machine". Meanwhile dynamically programmable PLDs are available also from Xilinx and Plessey. These are used within the second generation MoM hardware having been almost completed (December 1989) at Kaiserslautern [MoM90]. From Plessey also gate arrays are available being compatible to Plessey PLDs. That's why tested and debugged second generation MoM xputer machine code can be used directly (i. e. without any design effort) to freeze an implementation into a gate array version.

5.1 Wiring Congestion Problems within very large r-ALUs

The fully parallel bidirectional interconnect between the r-ALU and each bit of each word held by the cache uses fully dedicated direct wiring (see section 2.2). In designing xputer architectures with extraordinarily large r-ALU resources this sometimes causes wiring congestion [HHW90]. During redesign efforts we have experienced, that such problems can be solved by clever partitioning schemes [HHW90], or, even systematically. This redesign study (DPLA-KL'90), based

on 0.8 micron technology on much larger chip area (10 by 10 millimeters) has shown, that only 4 ICs would be sufficient for the above DRC application example, which is a quite encouraging result.

This time the bottleneck, however, has been the number of pins available per IC. Another problem is the complexity of routing between window scan cache and the r-ALU chips (all this concentrated onto a single chip). A solution has been found by distributing the scan cache, such that each DPLA chip carries on board a segment of the cache covering just a subword of the data memory word. By systematic overlap between these subwords a high flexibility of the r-ALU has been achieved also with this kind of partitioning. Another approach would be a similar partitioning, but in time instead of in space, to save hardware, such that e.g. for the DRC example 2, 3 or 4 video scans would be needed instead of one.

We believe that it is an important and promising area of future research to find better partitioning schemes which best fit to the needs of universal xputer architectures. This would be only a part of a very promising area of xputer-driven research and innovative data processing hardware development to substantially improve the competitiveness of companies from industry in consumer electronics and industrial electronics as well as other high tech industries.

6. Conclusions

By xputer use for algorithms in digital signal processing, image preprocessing, VLSI layout verification and detailed routing and other algorithms dramatically high acceleration factors have been achieved experimentally. The paper has paper given an introduction to xputer use for acceleration of digital signal processing applications. Xputers are a novel class of high performance processors, the operation of which is deterministically data-driven. The innovative machine principles of **x**puters efficiently support innovative compilation techniques for much better optimization of parallelism than possible on (conventional) **com**puters. The programming paradigm, which stems from the deterministically data-driven xputer machine paradigm, has been illustrated by introducing a few xputer application examples in digital signal processing (DSP). In 2-D digital filtering acceleration factors of up to > 300 have been achieved. Acceleration factors for the other DSP examples having been used within this paper (FFT, convolution and Viterbi algorithm) will be available by end of March 1990. (We expect figures of at least >100.)

Figure 14: The "MoM Lab" xputer laboratory at Kaiserslautern

With PLDs commercially announced for 1990 per DSP example from this paper ≤ 2 integrated circuits will be sufficient for the r-ALU. Xputer's (non-reconfigurable) "standard parts" are more simple than a RISC processor. This shows that xputers are very cheap and useful accelerators for important DSP applications. Xputers combine the flexibility, generality and low cost of microcomputers with the speed advantages of specialized hardware solutions. Xputers also support smooth integration into existing application environments and of convenient utilization of contemporary programming and software engineering tools and environments.

Critical comparison with previous work (high performance computing, optimizing compilers and ASAPS) has been published elsewhere [Mch90]. Second generation xpiler software and second generation MoM xputer hardware is currently being implemented at Kaiserslautern [MoM90]. By using modern visualization techniques the xputer data scheduling paradigm would be easy to learn and easy to use. But xputers may also be programmed with conventional procedural notations like MoPL accepted by the MoMpiler having been implemented at Kaiserslautern [HHW89, Kil89, MoM90] Also xputer architectures with extraordinarily large r-ALU resources are feasible at very low hardware cost by using clever partitioning schemes, or, even systematically. Xputers have opened up a very promising area of future research and product development.

7. Acknowledgements

Early versions of parts of the MoM concepts have been developed within the multi university E.I.S. project, having been jointly funded by the German Federal Ministry of Research and Technology and the Siemens-AG, Munich, under coordination by GMD, Schloß Birlinghoven. We also would like to acknowledge the various kinds of personal support from Elfriede Abel, Herwig Heckl, Gustl Kaesser and Klaus Woelcken (now with the Commission of the European Communities) on leave from GMD. We also appreciate valuable ideas from Klaus Singer at eltec GmbH at Mainz, and Karin Lemmert at BASF Ludwigshafen (formerly at Kaiserslautern University). Last but not least we appreciate the contributions of our students: T. Blüthner, S. Burkhardt, P. Dewes, J. Holzer, B. Mank, T. Mayer, R. Müller, W. Müller, C. Münster, H. Nicklaus, S. Reibnegger, H. Reinig, A. Schaffer, K. Schmidt, and J. Westphal.

8. Literature

[AlGo09] G. S. Almasi, A. Gotlieb: Highly Parallel Computing; Benjamin / Cummings 1989

[CoTu65] J. W. Cooley, J. W. Tukey: *An algorithm for the machine computation of complex Fourier series;* Mathematics of Computation, April 1965.

[Eis89] T. Mayer: *POLU:Problem-oriented Logic Unit*, Diplomarbeit, Univ. Kaiserslautern 1989.

[For73] G. D. Forney: *The Viterbi Algorithm;* Proc. IEEE **61**, 3 (March 1973)

[Gaj81] D. D. Gajski, D. J. Kuck, D. A. Padua: Dependence-driven Computation; Proc. COMPCON Spring 1981

[HaKo75] R. Hartenstein, G. Koch: The Universal Bus Considered Harmful; in: R. Hartenstein, R. Zaks: The Microarchitecture of Computer Systems, North Holland, 1975

[HHW87] R. Hartenstein, A. Hirschbiel, M. Weber: *MOM - Map Oriented Machine,* in: Ambler et al.: Hardware Accelerators, Adam Hilger 1988. also: Preprints Int'l Works.on Hardware Accelerators, Oxford, 1987.

[HHW88] R. W. Hartenstein, A. G. Hirschbiel, M.Weber:*MOM - Map Oriented Machine*, in: Chiricozzi, D'Amico: Parallel Processing and Applications, North Holland,1988. also: Preprints Int'l Conf. Parallel Processing and Applications, L'Aquila, Italy, 1987

[HHW89] R. Hartenstein, A. Hirschbiel, M. Weber: *MOM - a partly custom-designed architecture compared to standard hardware;* Proc. IEEE Comp Euro '89, Hamburg, F.R.G., IEEE Press Washington DC, 1989

[HHW90] R. Hartenstein, A. Hirschbiel, M. Weber: *Acceleration of Layout Verification and Routing achieved by Xputer Use;* (submitted to an Int'l IEEE Conf.)

[HwBr84] K. Hwang, F. A. Briggs: *Computer Architecture and Parallel Processing;* McGraw- Hill, 1984.

[Kil89] R. Hartenstein,A. Hirschbiel,M. Weber:*Mapping systolic Arrays onto the Map-oriented Machine (MoM)*, Proc. 3rd Int'l Conference on Systolic Arrays, Kilarney, Ireland, May 1989.

[Kun84] S.Y.Kung: *On Supercomputing with Systolic/Wavefront Array Processors;* Proc. IEEE **72**, 7 (July '84)

[Lem89] R.Hartenstein, K.Lemmert: *SYS^3 - A CHDL-based CAD System for the Synthesis of Systolic Architectures*, Proc. IFIP CHDL'89, North Holland, 1989.

[May89] T. Mayer: *DPLA (dynamically programmal logic array) - Entwurf, Anwendung u. Programmierung;* Diplomarbeit, Univ. Kaiserslautern, Aug. 1989

[Mch90] R. Hartenstein, A. Hirschbiel, M. Weber: *Xputers: an Open Family of non-von Neumann Architectures;* Proc. GI/ITG Conf. on the Architecture of Computing Systems, Munich, F.R.G., March 1990; VDE-Verlag, Düsseldorf, FRG, 1990;

[MoM90] The second generation MoM hardware and Application Development Environment; (in preparation)

[RaGo75] L. R. Rabiner, B. Gold: *Theory and application of digital signal processing;* Prentice-Hall, 1975.

[Swa87] E.Swartzlander: *Systolic FFT Processors;* in: Moore et al.: Systolic Arrays, Adam Hilger, 1987.

[Txf89] R. Hartenstein, A. Hirschbiel, M. Weber: *Non-von Neumann: is the Technology Transfer an*

Achievable Goal ?; report, Univ. Kaiserslautern, 1989