XPUTERS: AN OPEN FAMILY OF NON-VON NEUMANN ARCHITECTURES

R. W. Hartenstein, A. G. Hirschbiel, M. Weber, Universität Kaiserslautern,
ITG/GI Conf. "Architektur von Rechensystemen", Munich, Germany March 1990

Abstract. The paper introduces the principles of **x**puters - in contrast to the principles of von Neumann type **com**puters. The paper characterizes a class of algorithms which run by orders of magnitude faster on xputers than on computers and explains the novel execution mechanisms of xputers as well as novel compilation techniques to generate high performance xputer machine code. The paper proves, that xputers are as universal as computers. Based on a capacity analysis of communication mechanisms within the hardware the paper also shows the competitiveness of xputers against MIMD concurrent computers, VLIW computers, and data flow machines, and illustrates, that the design space of xputer architectures opens up a promising new area of research and development in processor architecture.

1. Introduction and about Systolizable Algorithms

Xputers are a class of universal processors based on innovative computing principles being a major deviation from von Neumann principles. Xputers, their principles and their "compilers" (which we call ***xpilers***) are described within this paper. Fig. 3 compares structure and throughput-relevant features of **com**puters and **x**puters. By performance xputers are superior to (von Neumann) computers in an important class of algorithms - which we call *systolizable algorithms* - in application areas such as digital signal processing, image processing, speech, radar, sonar, seismic and other kinds of processing [HH]. Fig. 1 summarizes experimental performance figures having been obtained for systolizable algorithms on the MoM xputer architecture [HH] and estimates of general acceleration factors which may be achieved. It is obvious that best performance is achieved for systolizable algorithms. Systolizable algorithms have intensive data dependencies (i.e. have a dense net of data interdependencies) of high regularity. Systolizable algorithms can be converted into systolic algorithms having highly regular **local** data dependencies. This is a reason why for such algorithms highly efficient optimization methods are available for compilers [Bul, Para]. More details and an introduction are given somewhere else, e.g. in [fast, SYT].

Section 2 classifies parallel computer systems with respect to performance issues. It shows, what type of parallelism has to be permitted by the hardware to accept code from highly optimizing compilers and thus to give best support for systolizable algorithms. Section 3 introduces xputer principles in contrast to (von Neumann) computer principles and explains the much better xputer performance by extreme compiler-friendliness. Section 4 shows the feasibility of families of xputer architectures with the goal of best performance for given major application areas. This section also summarizes the MoM as an example of an xputer architecture which efficiently supports pattern matching, image preprocessing and integrated circuit layout processing. Section 5 discusses technology issues and illustrates the feasibility of xputers at low hardware expense.

2. Levels of Parallelism in Computer Systems

Section 3 shows the superiority of xputers over von Neumann monoprocessors with respect to performance. To illustrate the competitiveness of xputers, however, also a comparison to contemporary parallel machines or systems as well as to customized VLSI solutions is desirable, what is subject of the following paragraphs. The very expensive and complex vector machines (usually classified as SIMD architectures) have fundamental performance bottle necks [Vec, Ve3].

algorithmic data dependence				achievable acceleration factors *)	
density	regularity		generics used 4)	single scan cache	double scan cache
	degree	locality			
low				*10*	
high	low			*10*	
	high	global	no	*10*	*50*
			yes	10 1) *100*	150 1) *1000*
		local		300 3) 2000 2)	

*) *italic: estimated* underlined: experimental (MoM) 2) design rule check 3) image preprocessing

1) 10-by-10 matrix multiplication only 4) hardwired generic data sequence available

Fig. 1: Some of the experimental results on a MoM using conservative technology compared to Motorola MC 68020 [HH] and some general performance estimates

Their sustained average performance is by several orders of magnitude lower, than their peak rate usually specified by vendors [Vec]. This even is the case, when creative coding techniques have helped the compiler to generate code which circumvents certain machine organization bottle necks [Ve2]. Because of their much higher degree of generality and flexibility and the availability of highly effective compiler strategies for them the following comparison stresses MIMD machines, rather than SIMD machine systems. Since xputers are data-driven this section also discusses optimization issues with respect to data flow machines.

Concurrent Computer Systems. Classes of MIMD computer systems may be distinguished by their different levels of parallelism. Conventional MIMD parallel processor systems (*concurrent computer systems*) exhibit high level parallelism (at process level). A program is split up into large tasks running concurrently. However, the exact processor time needed for each particular task is highly unpredictable, so that substantial amounts of time elapse for idling in waiting processors (compare fig. 2 a).

Communication between processes is asynchronous, which here is very expensive, since resources are frequently blocked by transaction-type synchronization (e.g. for handshakes implemented sequentially in software, or, for maintaining exclusive usage of critical regions). For process partitioning the compilers have to search for data **in**dependence (since searching for data **de**pendence would not be much helpful because of insufficient communication bandwidth available in the hardware). The weakness of this objective in distributing programs among processors has hamstrung early projects such as e.g. the Cm* project [ham]. Optimization success for systolizable algorithms is disappointing for fundamental reasons: the optimum very detailed instruction ordering cannot be mapped onto the hardware.

VLIW (Very Long Instruction Word) architectures [Eli, Ced] recently have drawn much attention in academic research. This is another class of MIMD architectures which, however, exhibits medium level parallelism (at instruction level). Its processors are coordinated by a global clock (fig. 2 b), where each clock cycle evokes the execution of exactly one instruction per processor in a coherent way such, that the juxtaposition of instructions currently activated forms a compound instruction, which evokes the execution of a compound function [Ga2]. During each such instruction cycle processors may communicate with each other synchronously (i.e. within nanoseconds) through one or more buses. It has been shown [Bul, Para] that - because of their high communication bandwidth - VLIW architectures are much more compiler-friendly by efficiently supporting highly efficient optimization based on data **de**pendence. Very good optimization results are achieved for systolizable algorithms, since excellent instruction ordering schemes can be efficiently mapped onto the hardware [Bul]. Since the performance of VLIW systems is extremely compiler-dependent, it has been proposed to develop the hardware **after** completion of the compiler implementation [Para].

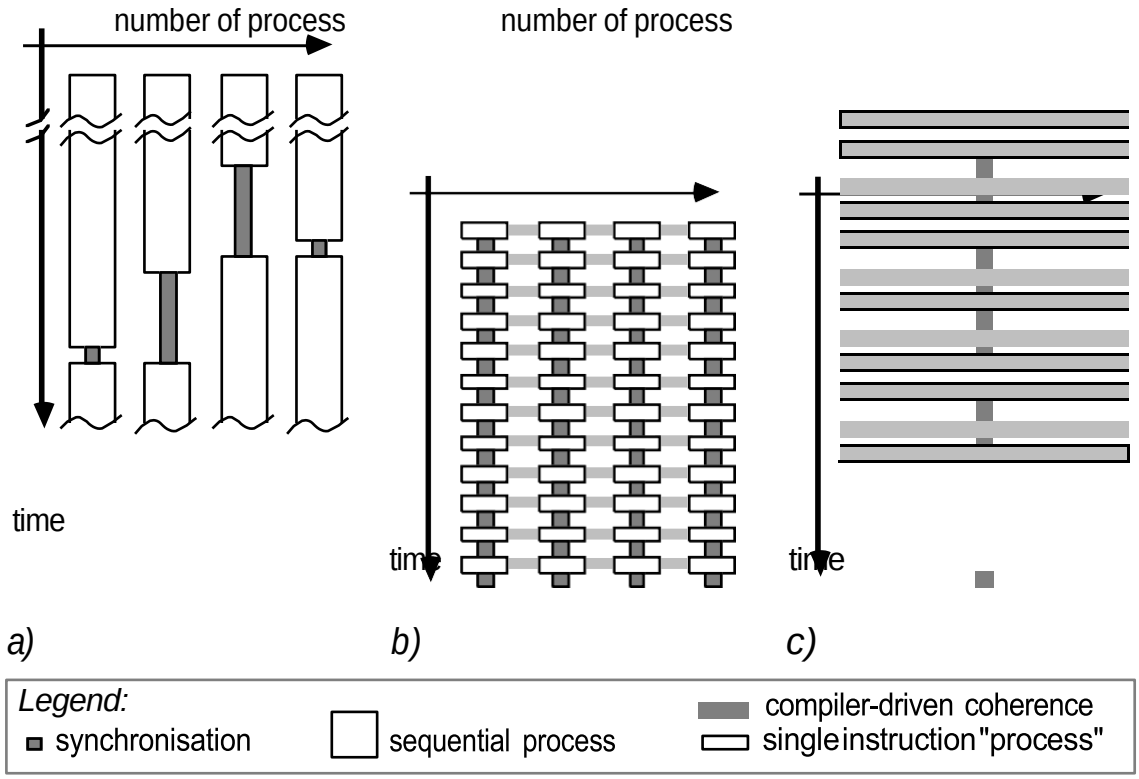


Fig. 2: Classes of Parallelism in Programmable Computing Machines: a) Concurrent Processes (Parallel Computer Systems), b) Parallel Processes by Compound Instructions (VLIW computer systems), c) intra-ALU Parallelism (Compound Operators) in Xputers

VLIW architectures still have drawbacks for three reasons. The efficient data sequencing schemes generated by optimization methods have to be mapped (from data sequencing) into control sequencing, which causes some overhead (1). Buses are used for inter-processor communication because of the need for programmability (2). A bus always causes multiplexing overhead, needs extra time for arbitration, and tends to be a bottle neck [Bus]. Inter-processor communication is done by **inter**-chip communication, which is very slow (3). Control flow overhead is avoided and much higher communication bandwidths are achieved within VLSI array processors (e.g. see [SYK]) for 4 reasons.

VLSI Array Processors (VAPs). In VAPs dedicated links are used which do not cause overhead (1). These links are **intra**-chip which are very fast (2). These links mostly use area-efficient *wiring by abutment* which leaves space for more PEs per chip (3) to increase parallelism. There is no controller, (and thus no control overhead) since the only "sequencer" is the global clock (4). The consequence is it, that VAPs permit the highest possible degree of parallelism at lowest possible level and thus of finest granularity. For this reason extremely good performance results are obtained for systolizable algorithms where the optimization (by *array compilers*, a sort of silicon compilers' front end) is fully based on data dependence analysis [Lem, SYK].

This lesson teaches us, that all PEs should be on the same chip and the lowest possible level of parallelism should be used to achieve best possible optimization results. Contemporary VAPs like systolic arrays, however, are not programmable. But the concept of xputers (subject of next section) also uses the lowest possible level of parallelism (synchronous parallelism **within** the ALU) in a way, so that real programmability is achieved by compilers using data-**de**pendence- driven optimization methods. Another drawback of VAPs is the fact, that conversion of data arrays into or from data streams requires additional computation capacity (mostly outside the VAP chip and thus highly inefficient). By the way: such a conversion is not needed for emulation of VAPs by xputers (also see section 3.1).

Data Flow Machines. The machine code for data flow machines is a data flow graph. At first glance this looks good (data dependence !). But at second glance it becomes obvious, that here this does not support dependence-based optimization. The data dependence information loaded into the machine is purely combinational and has not been mapped into a time dimension. The operation of the machine itself is highly indeterministic at the level of fine granularity, so that the hardware does not accept a predetermined order of executions and data accesses. The consequence is it, that - with respect to optimization - data flow machines are extremely compiler-hostile. Performance optimization cannot be done on data-dependence basis, nor at run time. Data flow machines have a number of other drawbacks [Gaj]. They e.g. have introduced several new kinds of bottlenecks and tend to suffer from data accessing conflicts [Gaj]. Existing high level programming language sources cannot be automatically translated for data flow machines [Bode, Gaj]. Code tends to be very large, and, redundance and extensive software pointer chasing causes an enormous addressing overhead [Bode, Gaj]. Parallel access to data flow machines' networks for selection, control and distribution is very difficult to implement for principal reasons [Bode].

3. Xputer Principles

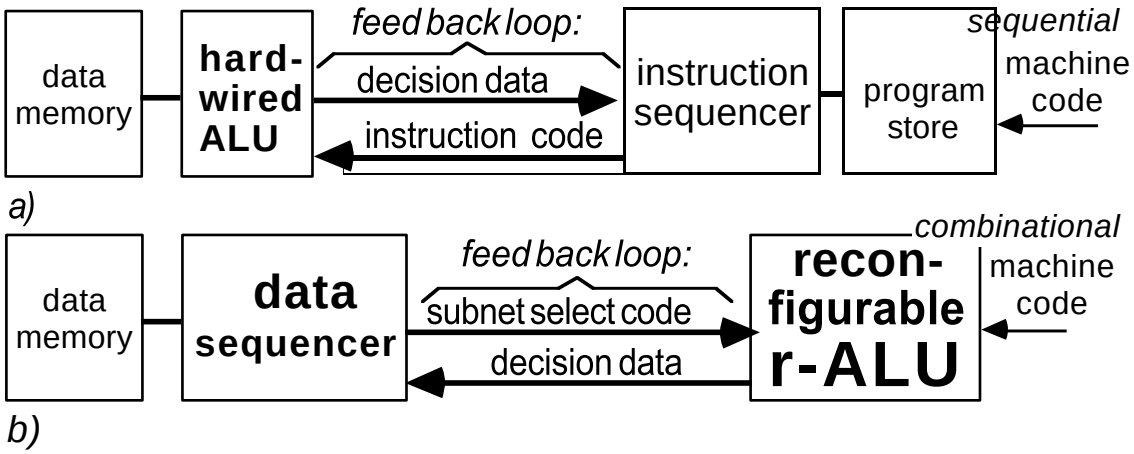
In VLSI digital hardware the communication tends to be much more difficult and expensive than processing power ([SYK] and others). The lecture from section 2 has taught us, that a very high degree of parallelism should be programmable into the hardware at lowest possible level, so that intelligent optimizing compilers find a massive amount of fine grain communication capacity within the hardware. This is impossible with the hardwired ALU of computers - also within parallel processor systems, since for such compilers the flexibility and bandwidth of communication mechanisms is as important as the number of processors and their power. Xputers use a fundamentally different approach. Their reconfigurable ALU (r-ALU) permits the configuration of communication at levels below instruction set level. That's why xputer principles offer exactly the kind of low level parallelism needed for code generation techniques available.

For illustration of xputer principles fig. 3 compares the structure of **com**puters (see fig. a for Harvard version of von Neumann machine) 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 use a reconfigurable *r-ALU*, which may be configured such, that several sets of highly parallel data paths form powerful *compound operators.* Xputers do not have a program store nor an instruction sequencer, since their combinational machine code is loaded directly into the r-ALU. Fig. 3 c summarizes computer to xputer comparison: 3 of 4 von Neumann bottlenecks have been eliminated. The 4th one has been greatly reduced because of highly optimizing "compilation" by xpilers, which achieve drastically better optimization results (especially for systolizable algorithms) than compilers for conventional parallel processing systems [Lem, Kil].

3.1 Data Sequencing

Unlike computers - which are control-driven - xputers are data-driven. But unlike data flow machines (which operate indeterministically), xputers feature fully deterministic principles of operation which we call *data sequencing.* The r-ALU is not controlled by an instruction sequencer (like computers, see fig. 3 a). Xputers have a *data sequencer* (see fig. 3 b). Such data sequencers feature hardwired *data scan sequences* making the locus of access travel step by step through the 1-, 2-, or n-dimensional data memory space. Examples of such *data scans* are [HH, Aq] *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, butterfly sequence, trellis sequence and others.

In most cases during such a generic scan sequence (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 interconnect between r-ALU and data sequencer (fig. 3 b) similar to that between controller and ALU in computers (fig. 3 a): A *subnet select code* fed into the r-ALU may evoke switching between different compound functions of the r-ALU. For data dependent scan sequences or scan tasks, such as e.g. in curve following (image processing, circuit extraction and other applications) *decision data* generated by the r-ALU may influence data sequencer operation. In xputers, however, this feed back loop is not hardwired, but may be reconfigured during 'programming' the r-ALU.



part of machine	computer	xputer
ALU	simple operations (instruction set) (bottleneck no.1: only 1 at a time)	powerful compound functions high degree of parellelism
	hardwired	re-configurable
instruction sequencer	bottleneck no.2: control flow overhead	has no instruction sequencer
program store	bottleneck no.3: slow access time	has no program store
data memory	bottleneck no.4: sequential and slow access	xpiler: number of accesses per data item drastically reduced

c)

Fig. 3: Difference between structures of: a) von Neumann machine (Harvard version) and b) Xputers. Comparison of throughput properties (c).

VAP Emulation. Before input to, or, after output from a VAP, data arrays have to be converted into somehow warped *data streams*, or, the other way around, respectively. This causes a huge I/ O overhead, normally to be done sequentially by the host. VAP emulation on xputers, however, does not cause such an overhead, since hardwired special generic scan sequences like *flushing video scan* have been developed [Kil] for this. Let's explain emulation in contrast to VAPs: xputers use only a single PE, connected to a double scan window. The VAP's PE array operation has been mapped from space to time, so that xputer data are not streaming, but stored as arrays at fixed memory locations. Instead of data the windows are moved, so that array / stream conversion is not needed. Thus VAP emulation on xputers is much more efficient than conventional VAP / host operation.

3.1 Task Sequencing

To obtain the generality of computers the introduction of control-driven elements is necessary also in xputers, so that organizational frameworks with irregular data dependencies may be implemented around xputer tasks (see above) derived mainly from systolizable algorithms. Such operations we call *task sequencing* [HH]. But - in contrast to computers - such a sequencer operates at very high level: instead of "tiny" instructions (von Neumann) powerful "huge tasks" have to be sequenced. Normally only a very small control store is needed. By inclusion of such a task se- quencer xputers become as universal as computers (also see section 4.2).

That's why xputers may run in stand-alone mode. 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 [xf] - also for xpiler implementation. As such a co-processor an xputer does not need a task sequencer, since task sequencing may be conveniently implemented on the host.

3.1 'Xpilers' - Compilers for Xputers

In section 2 the dominant role of optimizing compilers for high performance computer systems has been discussed. Since xputers do not have a hardwired instruction set xpilers are of vital importance to them. It has been shown that here dramatically good results are achieved in code optimization from systolizable algorithms. Even better results are achieved by array compilers for VAPs synthesis. Xpilers can be derived from such array compilers ([Lem, HH], also see section 4.1), since parallelism in xputers is of very low level - lower than with VLIW systems. This low level is the reason for the similarity of xpilation techniques to ASIC design methods: xputer concepts are a marriage between (processor) standard circuit use and VLSI design [HH].

1. Xputer Architectures

Within past four decades thousands of computer architectures have been developed because of the simplicity of basic von Neumann computer principles. Also basic xputer principles are sufficiently simple to develop numerous architectures on this basis. Fig. 4 gives some examples of architectural criteria and features for current and new xputer architectures.

resource	*criteria*	*features*
r-ALU	capacity of r-ALU resource	gate count, no. of particular operators, ..
	level of r-ALU architecture	gate level, functional level, mixed level ..
	r-ALU structural concept	EPLA, EGA, EPLD, mixed structures ?
	r-ALU configuration technology	floating gate, flipflops, fusible, ...?
	operator repertory of r-ALU	2 adders, 4 multipliers,.... ?
	programmability	programmable, customized, hybrid ?
	micro-architecture	design style, partitioning, ...
data sequencer	number of data scan caches	single cache, 2, 3, or more scan caches?
	kind of cache reconfigurability	max. size (5 by 5 ?), dimensions, ... ?
	data sequencing pattern repertory	video scan, shuffle, trellis, ...?
memory	memory segmentation features	segment nesting depth, dimensions, ...
	memory space dimensionality	1-D, 2-D, >2-D, reconfigurable, ...
	memory acceleration features	interleaved, array access, multi-port, .
interface	universality	stand-alone, embedded machine, hybrid

Fig. 4: Examples of Possible Architectural Features of Xputers

R-ALU architecture. The most important aspect is the architecture of the r-ALU, since its flexible low level internal parallelism is the basis for xputers' friendliness to intelligent optimizing xpilers. A good way to optimum performance for a given class of algorithms is it, to develop the xpiler first, and then to develop the hardware to support the kind of code generated. Various different objectives of r-ALU development could make sense. Should its concept stress universality or optimum support to a particular application area? Should it be programmable or customized? Should it stress very high performance or cheap mass production? What technology is available? The example of cordic processors [Cor, Co2, Co3] and the Warp machine [Warp] having been optimized for a class of algorithms similar to systolizable algorithms also is a source of good ideas for a family of xputer r-ALUs. The design space for r-ALUs is very large. The reconfigurability could be designed at gate level, at RT level, or at mixed levels. The elements to be configured could be masses of gates, but also e.g. one or more adders, a number of multipliers, several relational operators and a bunch of customized special operators.

An important problem for research here is the development of better r-ALU microarchitectures for programmable interconnect in order to achieve more capacity with less r-ALU chips or to achieve highly powerful single-chip solutions. Probably the use of PLA-like structures will be far from the optimum. Of course these structures highly depend on the code generation methods available for xpilers, which also are an important area of xputer research. Many important results from existing areas such as e.g. array compilation, systolic array synthesis, algorithms for digital signal processing, several sub-areas of VLSI design and ASIC development are an important source of good ideas and even methods ready for use or conveniently adaptable to the area of xputers.

Architecture of the Data Sequencer. Also the concept of the data sequencer is an important architectural aspect. Its repertory of generic data sequencing patterns is the key to optimum performance in executing particular classes of algorithms. For instance parametrized shuffle patterns [HH] are the key for extremely fast execution of FFT (Fast Fourier Transform) algorithms and other transforms in digital signal processing (e.g. see [fast, SYK]) on the MoM. Other examples are *flushing video scan sequences* to ape systolic arrays on xputers [Kil], which efficiently remap the data streams known from systolic arrays into data sequencing. This also is the key of using a systolic array synthesizer having been implemented at Kaiserslautern [Lem] as an optimizer within the MoMpiler for the MoM architecture (see section 4.1). Many kinds of more or less well known interconnect patterns (e.g. see [Bun, fast, SYK]), such as e.g. trellis, shuffle, butterfly, exchange, hypercube, X-tree and others are exactly the kind of generic schemes, which may be hardwired by surprisingly simple hardware to move through xputers' data memory without any overhead.

Memory Organization and Interfacing. Because of the high performance of xputers the communication with the data memory becomes a bottle neck. This problem is known from computers. Similar methods to alleviate memory contention may also be applied to xputer data memories. Since the xputer primary memory only stores data, however, it is much more simple to analyze bottle neck phenomena, so that known methods for data dependence remapping (e.g. see [fast, Lem, SYK] may be used to develop much better memory organizations supporting particular classes of algorithms much more efficiently. This makes possible a much more efficient cache utilization for xputers than known for computers.

Deterministic Cache Use. Instead of probabilistic strategies like by computers (with modest results only) xputers may use deterministic cache strategies, due to the concept of **data** sequencing. Xputer cache operation may be optimized on the basis of data dependence analysis at compile time. Experimental results on the MoM have shown suprisingly better results (see next section).

4.1 The MoM xputer architecture

The MoM represents one of many possible xputer architectures. It features (1) a special kind of reconfigurable register file called *window scan cache*, (2) a 2-dimensional memory organization supporting nested segments, and (3) the repertory of its data sequencer includes among others a *video scan* (within a given segment). The window cache is like a small 2-dimensional window to look locally into the memory. Fig. 5 [Ox] illustrates its reconfiguration feature permitting the adjustment of different window sizes and formats, such as e.g. 3-by-3 *"pixels"* for image processing or e.g. 4-by-4 pixels for grid-based layout analysis and others. The window scan cache is a reconfigurable multi-directional 2-D shift register. To minimize data memory accessing during video scan 2-dimensional bidirectional shift paths are available within the window cache. For a *video scan* within boundaries of a particular 2-dimensional memory segment only a single *data loop* is carried out where the data sequencer moves the window line by line over the entire segment such, that each word or "pixel" within this segment is visited by a particular reference point within the window exactly once.

If, for instance, segment size is 1024-by-1024 "pixels" and a 4-by-4 cache adjustment is used, such a video scan moves the window throughout the entire segment until each of all possible 1021*1021 window positions has been visited exactly once (not 1024*1024 positions in order to avoid going beyond segment limits). By such video scan movements the window scan cache of the MoM very efficiently supports all applications with *regular local data dependencies* in 2-dimensional arrays, such as e.g. in local pattern matching for image preprocessing. This will be shown in the next section by illustrating a design rule check application. The architectural features of the MoM have been derived from the PISA system including a specialized accelerator concept [San].

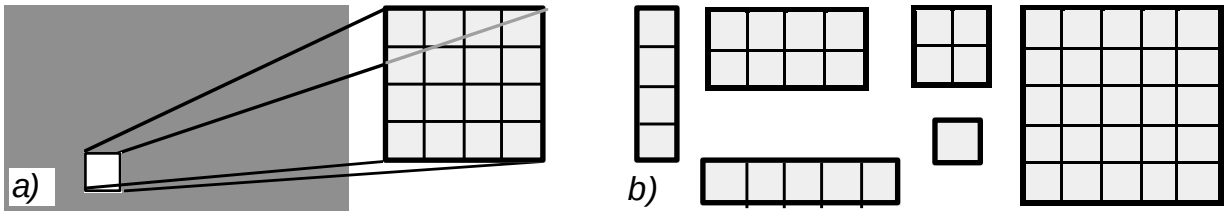


Fig. 5. MoM Architecture Example Detail: r-ALU of the MoM: a) illustration of reconfigurable Data Window Scan Cache; b) Examples of Window Cache Size Adjustments.

The MoM hardware as well as its MoM-DE application development environment [HH] have been implemented at the Kaiserslautern Xputer Lab (fig. 9). Via VME bus an ***eltec*** "68k" computer system (a 68020 system running under OS-9, a real-time version of UNIX) is connected to it as a host and user interface [elt]. The MoM-DE currently is running on a MicroVAX 3100, linked to the ***eltec*** "68k" system via Ethernet.

Integrated Circuit Layout Processing. It has been shown that by using grid-based design rules [Ly, MC] a design rule check (DRC) may be carried out via pattern matching methods (similar to those in image preprocessing) using a finite state machine approach [Eu] or combinational logic [San]. For this application dramatic acceleration factors (>2,000) have been achieved experimen- tally on the MoM [Aq, Ox]. For such a DRC the layout is stored in a 2-dimensional pixel map segment. The design rules are converted into a set of reference patterns, such that a pattern exists for each possible local design rule violation visible within the scan window. Fig. 6 b, for example, shows all reference patterns needed to detect a violation of the rule "minimum separation of poly from poly is 2 pixels". To represent CMOS design rules [IMS] 800 reference patterns have been needed which all fit into a 4-by-4 window.

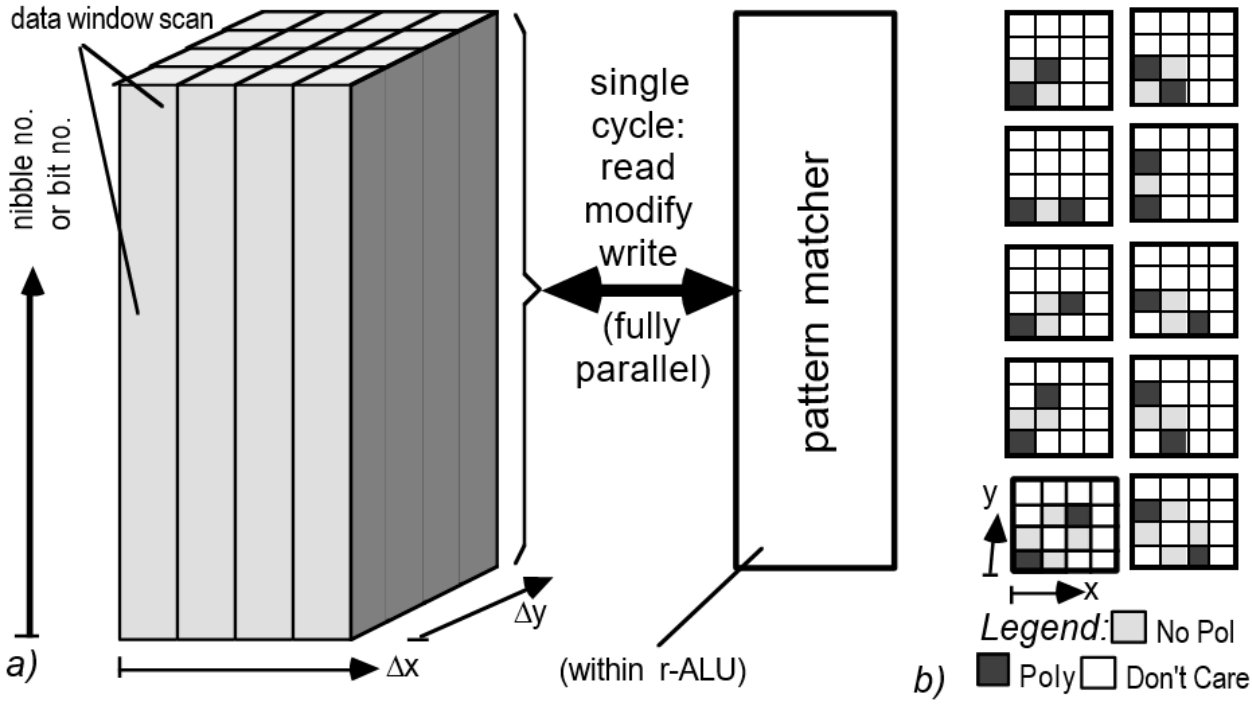


Fig 6: DRC based on pattern matching : a) r-ALU configuration, b) reference patterns example (automatically derived from the rule "noPoly separates Poly from Poly by ≥2 Lambda"

These reference patterns are configured into the r-ALU so that it is a huge combinational network ("pattern matcher", see fig. 6 a) connected in parallel to the window scan cache. This realizes a single but very powerful compound function linked with a video scan sequence. During each particular position of the window all 800 reference patterns are matched simultaneously within a single read-modify-write cycle, i.e. within nanoseconds. If an error has been detected an error code is written back into a reserved bit or byte of particular pixels held by the cache.

Experimental results with this hardware have proven acceleration factors by more than 2,000, where the comparison is intended to be fair: the technologically conservative MoM hardware has been compared with the relatively slow MC 68020 [HH]. For the same reasons substantial acceleration factors can be achieved for other kinds of grid-based layout processing, such as Lee routing [Aq, Ox, HH], circuit extraction [Bak], ERC (electrical rules check) [Bak], compaction [Cpn], processing of tiled layout [Geb, New], fault extraction from layout [CVT] and others.

The Application Development Environment. The MoM application development environment (*MoM-DE*) running on a host (currently a μVAX) features a special high level language *MoPL* (Map-oriented Programming Language), which roughly is a Pascal extension. Novel procedural constructs have been added which efficiently support the innovative MoM xputer features, such as e.g. generic data sequencing, data loops, configuring compound functions and loading tasks the sequencer's task register sets. Fig. 7 gives a survey.

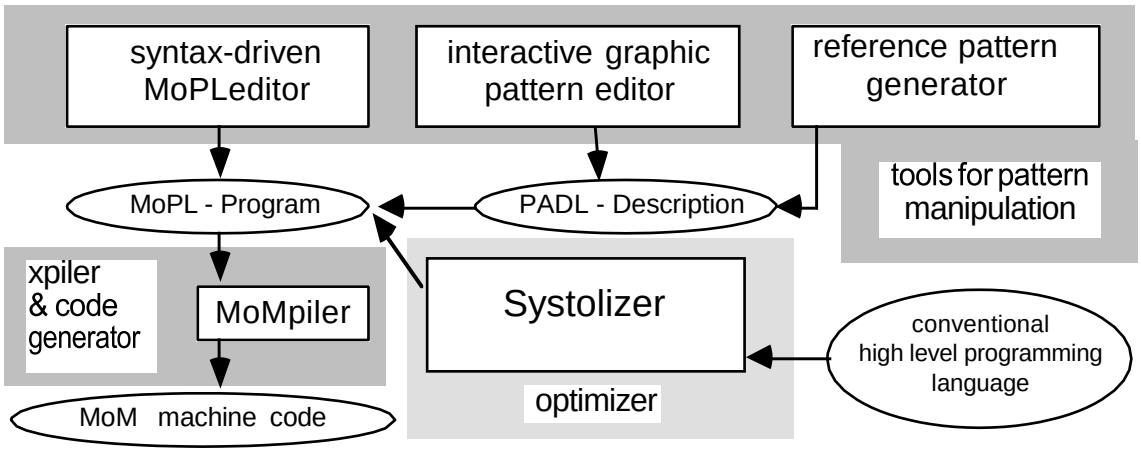


Fig. 7: Application Development Environment Supporting the MoM Architecture

Its xpiler (called *MoMpiler*) also features syntax-driven editing such, that it greatly alleviates the need for training the use of important paradigms of MoPL and its sublanguage *PaDL* (see below). At its front end MoM-DE includes a *Systolizer* for optimization by pseudo-systolic data dependence remapping [Kil] to achieve greatly minimized data memory accessing. The Systolizer accepts a conventional functional or imperative program notation quite similar to Pascal and generates MoPL code.

Image Processing. MoPL includes a sublanguage *PaDL* (Pattern Description Language), which efficiently supports Xputer programming for pattern matching applications in general. Especially for design rule check and similar applications an optimizing reference pattern generator has been implemented, accepting design rules expressed in a simple separation rule language (for an example illustration see fig. 6 b). These reference patterns having been generated automatically, how- ever, may not be suitable for many other kinds of layout processing. That's why an interactive graphic pattern editor supports convenient editing, modification, inspection and surveying of sets of reference patterns. Figure 8 shows a screen dump from its operation on a μVAX.

4.1 Universal Architectures

To obtain the generality of computers the introduction of a control-driven element is probably necessary also in xputers (the task sequencer: see fig. 10 a, also see section 3.2), so that not only systolizable algorithms may be carried out, but also anything else what is computable at all. In case the data sequencing steps available from the data sequencer will cause problems, a temporary switch over to control-driven sequencing is possible. If the data sequencer just picks a single memory word it is not really data *sequencing*, so that flow control is rendered to the task sequencer, i.e. control sequencing is applied. For the sake of universality it might be sometimes useful also to call such trivial tasks (e.g. single-step "tasks"), such as e.g. checking a bit or comparing a number. A properly designed section *rh-ALU* (reconfigurable high level ALU, see fig. 10 a) of the r-ALU might be useful for a more economic implementation of conventional simple single instructions.

We have the strong belief, that universality of xputers might also be achieved by data-driven task sequencing, i.e. without control-flow-driven sequencing. This would make xputer principles evenmore simple and more elegant. Probably a new programming paradigm would be needed for such a solution. The question then would be, how much training effort would be needed by users. To decide this problem further research might be needed or at least further creativity. However, it might also be, that some discipline we are currently not familiar with has already a solution.

Such universal xputers may run in stand-alone mode. But co-processor use together with a computer 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 [xf] - also for xpiler implementation. In such a configuration (fig.10 b) task sequencing may be conveniently implemented on the host, so that universality may be achieved by delegation.

4.2 Discussion of Technology Issues

For programmable r-ALUs the technology is available from a rapidly growing PLD market (one billion US dollars worldwide by 1990). The general term *PLD* stands for "programmable logic devices" which includes a wide variety of technology solutions [PLD]. This section discusses experiences about r-ALU hardware requirements from MoM applications to integrated circuit layout verification and routing using fully parallel pattern matching [DAC]. Because of the large number of reference patterns needed (e.g. 800) for CMOS design rule check (DRC) applications are critical. Technology selection is a key to minimize the number of ICs needed for the r-ALU. With the technology of the mid' eighties two r-ALU versions have been implemented: one using intel 5C180 components [itl] which have to be pulled off socket for programming, and, another one using DPLA-KL86 circuits [May] being dynamically programmable within milliseconds inside the MoM. This DPLA-KL86 using 3 micron nMOS technology has been designed and tested at Kaiserslautern and fabricated by Fraunhofer-IMS Duisburg (F.R.G.) in 1986. For the DRC example 254 of them are needed, whereas only 96 if the intel 5C180 have been needed, which fit onto 2 double Europe size PCBs.

A redesign study (DPLA-KL90) has been derived from the DPLA-KL86 to investigate r-ALU requirements with the technology of the early nineties. For a 0.8 micron technology on much larger chip area (10-by-10 millimeters) this study has shown, that only 4 ICs would be sufficient for the above DRC, which is a quite encouraging result. This time the bottleneck, however, is the number of pins available and the complexity of routing between window scan cache (concentrated onto a single chip) and these 4 r-ALU chips. 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 sub-words 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.

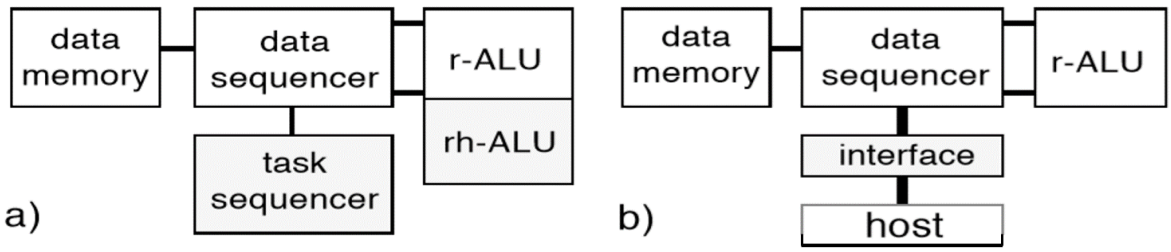


Fig. 10. Universality of Xputers: a) Stand-alone Xputer, b) Embedded Xputer

The (non-reconfigurable) "standard parts" of xputers are more simple than a RISC processor. It has been shown, that the technology needed for competitiveness of xputers is commercially available and steady improvements within next years are expected. It has been shown, that the concept of xputers supports smooth integration into existing application environments and of convenient utilization of contemporary programming and software engineering tools and environments. It has been shown, that xputers combine the flexibility, generality and low cost of microcomputers with the speed advantages of specialized hardware solutions.

It has been shown, that xputer architectures open up an interesting new area of research and de- delopment promising cheap very high performance solutions for several larger application areas. The high performance of xputers is due to the fact, that the data-driven hardware concept and its very low level parallelism permits extremely efficient optimization at compile time. This also in- cludes highly efficient compile-time support of cache utilization, since data-driven deterministic strategies permit substantially better results than conventional control-driven probabilistic strate- gies. The highly promising results having been summarized here are just a beginning. We expect much more interesting results from future research.

Technology and Architecture of PLD-like circuits are rapidly proceeding. Also electrically alterable gate arrays *(EGA)* are available [xil] which have a much different micro-architecture. Recently Plessey has announced a so-called *ERA* (**E**lectrically **R**econfigurable **A**rray), which uses sea of gates structures, currently with up to 40,000 gate equivalents per chip. With the 1-µ technology available at Plessey 80,000 gate equivalents per chip are feasible [Era]. In general we expect, that within the next few years the market will offer much more types of EPLDs featuring higher density and better micro-architectures. More and more powerful r-ALU architectures will be designable for low hardware cost, so that very cheap and simple xputers will become more competitive against highly complex and expensive parallel computer systems. This has opened up a promising research area to develop area-efficient clever PLD architectures which efficiently support varieties of powerful flexible xputers with a minimum of hardware expense.

5. Summary

Performance figures have been summarized showing that xputers have a much better performance than computers, and, that for systolizable algorithms dramatically high acceleration factors have been achieved. From a comparison of communication mechanisms within data flow machines, different classes of parallel computer systems and VLSI array processors a recipe for innovative computational devices has been derived, which advocates for programmable parallelism at the lowest possible level. On this basis it has been illustrated, why xputer monoprocessors are competitive even against parallel computer systems.

The (non-reconfigurable) "standard parts" of xputers are more simple than a RISC processor. It has been shown, that the technology needed for competitiveness of xputers is commercially available and steady improvements within next years are expected. It has been shown, that the concept of xputers supports smooth integration into existing application environments and of convenient utilization of contemporary programming and software engineering tools and environments. It has been shown, that xputers combine the flexibility, generality and low cost of microcomputers with the speed advantages of specialized hardware solutions.

6. Acknowledgements

Early versions of xputer parts 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, F.R.G., under coordination by the GMD, Schloß Birlinghoven. We also would like to acknowledge the various kinds of help by Elfriede Abel, Herwig Heckl, Gustl Kaesser from GMD, and Klaus Woelcken, now with the CEC (Commission of the European Communities). We also appreciate valuable ideas and encouragement especially by Klaus Singer, CEO of ***eltec*** GmbH at Mainz, F.R.G., Dr. Talbot from the CEC, but also by Prof. Maria Giovanna Sami from Politecnico di Milano, Prof. Francois Jutand from Paris Telecom University, and, the contributions by Karin Lemmert at BASF AG, Ludwigshafen, F.R.G. (formerly at Kaiserslautern University). We are also in debt of our ZMK (Zentrum für Mikroelektronik Kaiserslautern) for the availability of design tools. 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.

7. Literature

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

[Bak] C.M.Baker, C.J.Terman: A Tool for Verifying Integrated Circuit Designs; Lambda 1st Quarter. 1980

[Bode] A. Bode: Datenflussrechner: Alternative zur vN-Architektur; Computer-Technik Nov. 1989

[Bul] J.A.Fisher, J.R.Ellis, J.C.Ruttenberg, A.Nicolau: A Parallel Compiler and a Dumb Machine; Proc. ACM SIGPLAN '84 Symp on Compiler Correctness; SIGPLAN Notices **19**, 6 (June 1984)

[Bun] Ch. Wu, Tse-yun Feng: Tutorial: Interconnect Networks; IEEE EHO-217-0, IEEE, Silver Springs, MD 1984

[Bus] R. Hartenstein, G. Koch: The Universal Bus Considered Harmful; in: The Microarchitecture of Computing Systems (eds.: R. Hartenstein, R. Zaks); North Holland, Amsterdam 1975;

[Ced] D. D. Gajski, D. H. Lawrie, D. J. Kuck, A. H. Sameh: CEDAR; COMPCON Spring 1984

[Cor] G. Schmidt et al.: CMOS Implementation of Optimized 16 Bit Processors and Evaluation Tools; Int'l Symp. on Signals, Systems and Electronics, Erlangen Highly Parallel and Pipelined Computing Using CORDIC Processors - Application to, F.R.G., Sept 1989

[Cor2] J. F. Böhme, Bing Yang: Computer Graphics, Signal Processing and Linear Algebra; Proc. German/Chinese Electronics Week, Shanghai, Sep1989, VDE-Verlag, 1989

[Cor3] B. Yang et al.: Special Computer: Graphics - Robotics; IEEE Comp Euro, Hamburg, May 1987, IEEE Press, Washington, DC, 1987

[Cpn] D. Boyer, N.Weste: Virtual Grid Compaction using the most recent Layer Algorithms; Proceedings ACM/IEEE IC- CAD 1983

[CVT] I. Stamelos, M. Melgara, M. Paolini, S. Morpurgo, C. Segre: A Multi-Level Test Pattern Generation and Validation Environment; International Test Conference, 1986

[DAC] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: Acceleration of Layout Verification and Routing achieved by Xputer Use ; submitted for DAC 1990

[Eis] T.Mayer: *POLU (Problem Oriented Logic Unit)*, Diplomarbeit, Univ. Kaiserslautern, 1989.

[Eli] J. A. Fisher: Very Long Instruction Word Architectures and the ELI-512; 10th ISCA, 1983

[elt] N. N. (eltec): eltec 68k system user manual; eltec GmbH, Mainz, F.R.G., 1987

[ERA] P. Dillian, I. Phillips: Elektrisch rekonfigurierbare Arrays; ELEKTRONIK 21, Oct 13, 1989

[Eu] R. A. Eustace, A. Mukopadhyay: Deterministic Finite Automaton Approach to Design Rule Checking for VLSI; DAC 1982 (Las Vegas)

[fast] R. E. Blahut: Fast Signal Processing Algorithms; Addison-Wesley, 1987

[Gaj] D. D. Gajski, D. A. Padua, D. J. Kuck, R. H. Kuhn: A Second Opinion on Data Flow Machines; Computer, **15**, 2 (Febr. 1982)

[Ga2] D. D. Gajski, D. J. Kuck, D. A. Padua: Dependence Driven Computation; Proc. COMPCON Spring 1981, IEEE Press 1981

[Geb] J. Gebhardt et al.: Functional Extraction from Personality Matrixes of MOL (Matrix-oriented Logic) Circuits; IFIP CHDL'87, Amsterdam 1987

[ham] A. K. Jones, E. F. Gehringer (eds): The Cm* multiprocessor project: A research review; tech. report, CMU-CS-80-131, Carnegie-Mellon-University, July 1980

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

[IMS] G. Zimmer: Lambda Designregeln für das EIS-Projekt; report; IMS Institute for Microelectronic Systems, Duisburg, F.R.G., 1986.

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

[Lee] C.Y. Lee: *An Algorithm For Path Connections And Its Applications.* In: IEEE Trans. on Electronic Computers, vol. EC- 10, pp. 346-365, Sept. 1961.

[Lem] R.Hartenstein, K.Lemmert: SYS3 - *A CHDL-Based CAD System for the Synthesis of Systolic Architectures*, Proceedings IFIP CHDL '89, North Holland, 1989.

[LKL] I. Velten: *Implementierung des Lee-Algorithmus auf der MOM*, Diploma thesis, Fachbereich für Informatik, Kaiserslautern University, Kaiserslautern, 1987

[Ly] R. F. Lyon: Simplified Design Rules for VLSI Layout; Lambda, 1st quarter 1981

[May] T. Mayer: DPLA (dynamically programmable logic array) - Entwurf, Anwendung u. Programmierung; Diploma thesis, Fachbereich Informatik, TU Kaiserslautern, 1989

[MC] C. Mead, L. Conway: *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[New] R. Newton, S. D. Vadas: Topological Optimization of Multiple-Level Array Logic; IEEE Trans. CAD for Integrated Circuits and Systems, Nov. 1987

[Ox] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: *MOM - Map Oriented Machine,* in: Ambler, Agrawal, Moore: Hardware Accelerators, Adam Hilger, Bristol, 1988. Also: Preprints Int'l Workshop on Hardware Accelerators, Oxford, 1987.

[Para] D. A. Padua, D. J. Kuck, D. H. Lawrie: High Speed Multiprocessors and Compilation Techniques; IEEE TC-**29**, 9 (Sept 1980)

[PLD] N. N.: Programmable Logic Devices - A Second Generation; Electronics, May 1988

[San] R.Hartenstein, R.Hauck, A.Hirschbiel, W.Nebel, M.Weber: *PISA - A CAD package and special hardware for pixel oriented layout analysis,* Proc. ICCAD 1984, Santa Clara 1984.

[SYK] S. Y. Kung: VLSI Array Processors; Prentice-Hall, 1988

[Vec] J. J. Hack: Peak vs. Sustained Performance in Highly Concurrent Vector Machines; Computer, Sept 1989

[Ve2] J. J. Dongarra, E. Eisenstat: Squeezing the Most out of an Algorithm in Cray Fortran; report, ANL/MCS-TM-9, 1983 :

[Ve3] J. J. Dongarra: A Survey of High Performance Computers; COMPCON Spring 1987

[Warp] H. T. Kung: Systolic Algorithms for the CMU Warp Processor; in: (E. Swartzlander, ed.:) Systolic Signal Processing Systems; Marcel Dekker, New York 1987

[Xil] N. N.: The Programmable Gate Array Design Handbook; Xilinx Corp., San Jose, CA, USA, 1986

[xf] R. Hartenstein, A. Hirschbiel, M. Weber: Non-von-Neumann: Is the Technology Transfer an Achievable Goal?; report, Fachbereich Informatik, TU. Kaiserslautern, 1989