A Pseudo Parallel Architecture For Systolic Algorithms

by

R.W. Hartenstein, A.G.Hirschbiel, M.Weber
Universität Kaiserslautern
Postfach 3049, D - 6750 Kaiserslautern, FRG
phone: (+49-631) 205-2606 or (+49-7251) 3575
fax: (+49-631) 205-3200

Abstract

This paper introduces a family of non-von-Neumann innovative computing devices, called *Xputers*. The map-oriented machine (MoM), an example Xputer architecture, is a flexible non-von-Neumann accelerator machine having been developed at Kaiserslautern University. This machine uses a two-dimensional map-oriented data memory. Over this memory a variable-sized window cache can be moved in arbitrary move schemes to analyze and change the data via the problem-oriented logic unit, which delivers powerful, programmable pattern matching mechanisms. The MoM can be used to speed up signal processing, image processing and VLSI layout processing and many other applications, but it may also serve as a systolic array simulator and evaluator. Moreover it can be also used as a low-cost, simple, flexible, and programmable array emulation computer. In contrast to a systolic array, where **data streams are moving** through an array of PEs, the MoM keeps **data at fixed locations** in its memory and moves its scan cache window in a application-specific manner across this memory space.

1. Introduction

This paper introduces a non-von-Neumann innovative computing device, called *Xputer*. It has been shown elsewhere, that its innovative processor principles [1] fill the technology gap between universal, but slow von Neumann hardware and inflexible, but powerful tailored hardware solutions. The significance of the Xputer goes even further beyond just filling a gap:

- it is as universal as von Neumann principles
- it is much faster than von Neumann for most important applications [1, 2, 3]
- its standard parts are much more simple than that of a von Neumann processor (much more easy to design than a RISC processor)
- its reconfigurable parts are available from stock at a billion dollar 18 vendor PLD market [4]

Programmable von Neumann type computation is dependent on code having been laid down in, and, being (sequentially) scanned from a program memory. Xputer programmability and uni-

versality, however, makes use of alternative computing structures: electrically configurable end re-configurable hardware, implemented with PLDs ("programmable logic devices") or similar components [4, 5, 6]. That's why Xputer computation is dependent on the programmed interconnect between a cleverly prepared set of simple hardware operators.

Section 1.1 discusses the need for non-von-Neumann innovations. Chapter 2 introduces the innovative **X**puter principles - an alternative to **Com**puters. Chapter 3 highlights in more detail the MoM example of an Xputer architecture. Chapter 4 introduces the MoM development environment and illustrates the differences between Xputer application development support environments and its traditional counterparts in computer science.

1.1. Why non-von-Neumann ?

The shortcomings of the von Neumann machine and its "von Neumann bottle necks" have been frequently discussed throughout the decades [9, 10]. Within a von Neumann machine three major throughput bottle necks can be identified which result in any lack of parallelism. That's why von Neumann processors are by orders of magnitude less powerful, than what may be achieved by tailored ASIC or full custom VLSI solutions of comparable transistor count. Many different kinds of processor architectural remedies have been proposed or implemented, which, however, do not really result in deviations from von Neumann principles [12, 13, 14, 15]. However, only limited improvements have been achieved [16]. The concurrency approach, i.e. bundling several von Neumann processors to form parallel computer systems requires a expensive huge additional software overhead, which substantially increases the complexity and incomprehensibility of such a system.

Better principles for a universal machine would be highly desirable: possibly more efficient (acceleration factors up to several orders of magnitude - already by a single processor), and more easily to be designed (to keep up with technology progress more rapidly). Machine principles are desirable, which achieve a very high degree of parallelism already **within** a single processor: fine granularity parallelism. This would be much more efficient, than the coarse granularity parallelism of concurrent processes in contemporary parallel computer systems: bundling masses of inefficient hardware, and causing a huge overhead for coordination, scheduling and dispatching.

2. Xputer Principles

This section explains the non-von-Neumann *Xputer* principles in a more comprehensible way - different from having been published elsewhere [1]. It introduces Xputer principles by highlighting its differences to the von Neumann partitioning scheme. The Harvard computer partitioning scheme in fig. 1 a illustrates, that the three following throughput bottle necks can be identified within a von Neumann machine:

(1) sequential data access (only a single data word may be accessed at a time)

(2) "sequential" ALU (narrow bandwidth ALU: only a single data operator can be activated at a time)

(3) program scanning overhead (not every instruction being scanned is a data manipulation instruction)

In (2) for each operator selection also an instruction has to be fetched from program memory. Fig. 1 b (vs. fig. 1 a) illustrates how alternative machine ("Xputer") principles are derived from the von Neumann machine:

(a) sequencer and program memory are removed (bottle neck no. 3 has disappeared)

(b) the hardwired narrow bandwidth universal ALU is replaced by a highly parallel *reconfigurable ALU (RALU)* (bottle neck no. 2 has disappeared)

(c) bottle neck no. 1 has been left over: however, a special data sequencer (fig. b) supports highly efficient optimization of data access sequencing

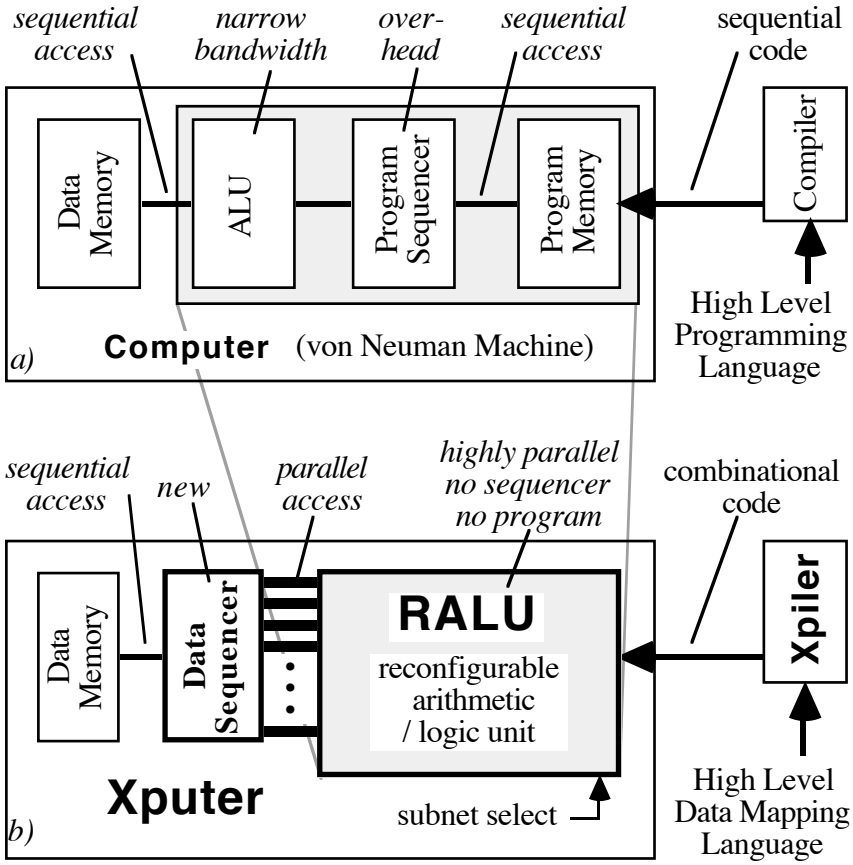

Fig. 1: Computer (Harvard Machine) vs. Xputer

Because of (a) the *Xputer* does not accept sequential code any more. That's why conventional compilers cannot be used for application development support. Because of (b) *Combinational Code* is needed instead, which (re)configures an electrically path-programmable PLD-based [4] or PGA-based [5, 6] ALU resource (called RALU) into a highly parallel network of data paths tailored to a particular application. For configuration a new type of application development software package is needed, which we call **X**piler (instead of **com**piler for **com**puters). That's why a RALU does not have a hard-wired instruction set: Xputers do not have a fixed word for-

mat. That's why extensions by adding more memory planes are possible conveniently.

The data sequencer includes a register array (called *data scan cache*) organized in a special way (see ch. 3 for more details). This register array is connected to the RALU by a very wide bandwidth data path (see fig. 1 b) in order to avoid a bottle neck. That's why the RALU (**r**econfigurable ALU) may execute so many data operations in parallel, so that within a single clock cycle the results of longer sequences of instruction executions on a von Neumann machine are met.

Such a highly powerful Xputer clock cycle we call an *X cycle* . No program store and no sequencer is needed to call such an *X cycle*. The set of functions carried out in parallel during such an X cycle we call an *X function*. Several X functions may reside simultaneously in a RALU. A particular X function may be activated from the RALU by its *subnet select* key (see fig. 1 b). The source of a subnet select code may be the data sequencer (see fig. 1 b), or the host interface (see chapter 4, also for more details on the data sequencer and its cooperation with the RALU).

The data sequencer features a hardwired data address sequence generator called *MCU* (*Move Control Unit*), which makes the data scan cache follow a particular path (*scan path*) through memory space. The MCU makes the scan cache behave 'traveling salesman'-like, subsequently 'visiting' different memory locations by following such a scan path. The shape of a particular scan path we call its *move pattern*. The *video scan* 's row by row move pattern through a 2-dimensional memory segment is an example of such a scan path (fig. 4 e). At each memory location visited on a scan path an X cycle may be evoked from the RALU. The sequence of X cycles thus associated with a scan path we call an *X loop*. By a stack mechanism within the data sequencer also sequences of X loops may be carried out. Such a sequence of X loops we call an *X task*. By a *task trigger* signal and a *task select key* the execution of an X task may be evoked from outside the MCU. No program-driven controller is needed for the MCU nor the TMU.

2.1 Programmable vs. Partly Customized Xputer

Currently there are two large flexibility gaps between universal computer solutions and these specialized solutions (1) and beyond von Neumann solutions (2) (see fig. 2). The Xputer concept offers three environmentally different classes of solutions:

- programmable Xputer (1)
- partly customized Xputer (2 a)
- embedded Xputer (2 b)

Xputer availability adds two more dimensions of flexibility: for many, many applications the universal *programmable Xputer* (using PLDs or programmable gate arrays) offers much more throughput without loss of flexibility (1). Another dimension of flexibility is the choice of alternative techniques (gate arrays or other ASIC technologies, or even full custom for even higher throughput) for RALU implementation (2): combination of tailored RALU with standard ICs for data sequencer and host interface (2 a), or tailored RALU combined with data sequencer and interfacing cells taken from a cell library (2 b).

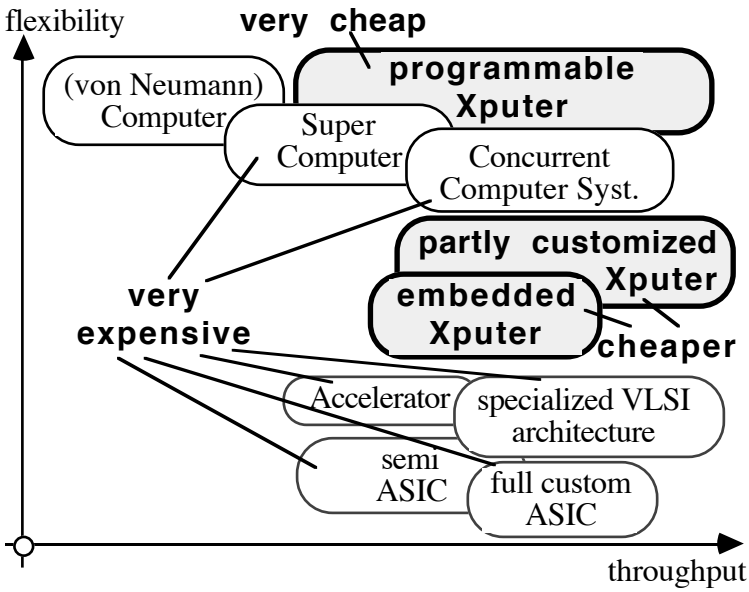

Fig. 2. Xputers: filling the flexibility/throughput gap between von Neumann and specialized solutions

The advantage of the *partially customized Xputer* over the "classical" ASIC approach is it, that only the RALU part has to be designed, whereas for the rest of the hardware standard integrated circuits may be used. Also the embedded Xputer solution substantially reduces design effort: only the RALU has to be designed, whereas the rest is made up from library cells provided by the technology vendor.

All this helps to save a substantial amount of design time and design cost, compared to a fully customized solution. This is an important aspect, since microprocessor development has entered a second phase of design crisis [39]. So the Xputer concept is not only a contribution to the area of computer structures, but also to the area of VLSI design practice [1].

3. The MoM Xputer Architecture

The general Xputer principles give room to develop many other Xputer architectures. This is similar to the von Neumann domain where also many processor architectures and computer architectures have been proposed and developed through the decades. One such Xputer architecture, having been implemented at Kaiserslautern is called "Map-oriented Machine" *(MoM)* or *MoMputer* [42, 43, 44]. It is introduced by this chapter. Its name is derived from the fact, that this architecture provides a powerful hardware support for mapping comprehensibly an important class of data dependencies [40, 41, 45] onto Xputer data memory space. Such a mapping technique is achieved by the design of the data sequencer (fig. 1 b). Essentials of the mapping support implementation are its subunits: the *data scan cache,* and the *move control unit (MCU).*

3.1 The Data Scan Cache of the MoM

The MoM's data *Data Scan Cache* is a reconfigurable register file for an efficient communication between RALU and data memory, such as e.g. for a read/modify/write communication. All words in this cache can be accessed in parallel by the RALU. The scan cache size is adaptable to different applications; it can be reconfigured at run time under parameter control. During win-

dow mode it maps a vari-size scan window onto the MoM data memory space [2, 3, 19]. It then holds and updates copies of a few neighbour words from memory for read/modify/write access. Fig. 3 shows some 2-dimensional cache size adjustment examples.

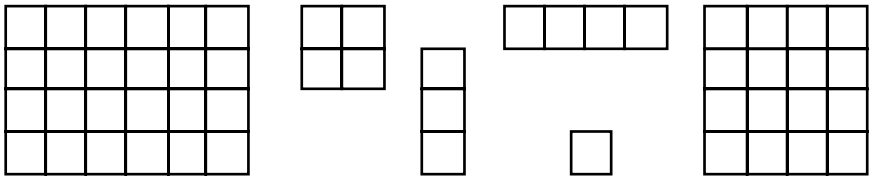

Fig. 3. Data Scan Cache: 2-dimensional Size Adjustment Examples

This buffer is called a *scan cache* or *scan window*, because the MoM architecture supports efficient cache updating [47, 48, 49, 50] when the cache is scanning a memory segment. This hardwired support is available for a repertory of scan patterns or move patterns (for illustration see e.g. fig. 4 a, d through j and l) being hardwired into the MCU. By a 4 by 4 cache format, for example, 16 operands are directly connected to the RALU (see the right end of fig. 3).

This cache format has been used together with a video scan move pattern for design rule check acceleration demo [18, 51, 52]: at each X cycle a 4 by 4 pixel segment of (grid-based) layout including an error layer is read into the cache, is matched in parallel with 800 reference patterns [52] (within the RALU, where its error layer is updated), and written back into data memory. For most image preprocessing examples also a 3 by 3 scan cache size adjustment would do [19]. The above design rule check example includes the highly parallel read/modify/write path as the only result path. As a second result path a feedback to the move control unit may be generated. This feedback is used for data-dependent cache movements (as e.g. in curve following, fig. 4 l).

In contrast to 1-dimensional von Neumann memory space, the MoM *map-oriented data memory* is primarily 2-dimensional. But at run-time the dimensionality of the memory also can be switched to 1-D, 3-D, 4-D or more dimensions [19]. Inside the data memory segments of arbitrary number and size can be defined to provide a memory map similar to floor plans in VLSI (if 2-dimensional mode is used). Also nesting of segments is hardware supported by a stack mechanism. Modern user interfaces encourage graphical presentation of such memory maps [53].

3.2 The Data Sequencer of the MoM

The data sequencer of the MoM (including the *Move Control Unit (MCU)* [19, 54, 55, 56]) provides hardwired move patterns for a repertory of scan paths (for terminology definition see last paragraphs of section 3.2). The MCU hardware supports all kinds of generic move patterns, which are completely defined by the name (or key) of a pattern, by a parameter, and by the segment limits [44]. The shuffle pattern in fig. 4 l, for instance, is completely defined by its name, its step width parameter [45] (which is 3), and by the segment length (which is 12.) Two major cache movement strategies are available from the MCU and may also be combined:

- independent move patterns (e.g. video scan or 'shuffle jumps')
- data-dependent move patterns (e.g. in curve following)

Cache movements are controlled by the **move pattern generator** [54] within the MCU using a structured jump generator hardware [55, 56]. It generates the address sequences needed for

moving the data scan cache along a particular scan path Fig. 4 illustrates part of the repertory of move patterns, such as e.g. for single steps to one of the 8 nearest neighbors (a), for relative (b) and absolute jumps (c) in memory space, as well as step sequences for a 'video scan' (e), skewed fill patterns (f, g), circular scan patterns (h, j) such as useful for the Lee algorithm [20, 21], shuffle sequences (k) useful for a large number of algorithms [57] (see fig. 4 for a double cache shuffle transfer example), or other '*travel paths*' through memory space.

By this powerful repertory of hardwired data scan paths the data sequencer may carry out long *sequences of X cycles,* also without the need to use a programmable controller. The removal of controllers driven from control code laid down in a program memory strictly avoids overhead and bottle necks at fine granularity level. The benefit of this is a very high throughput and still a high degree of flexibility by:

- hardwired support of regular interconnect schemes (shuffle [82], butterfly, trellis, spiral scan, n-dimensional fill scan, and others, also see fig 4)
- hardwired support far beyond nearest neighbour schemes
- extremely high parallelism at low level in pattern matching
- simple straight forward data sequencing optimization strategy is highly efficient

A third part of the data sequencer is the *task manager unit (TMU),* which also controls the coordination of X loops [58], which may be linked together to form an X task. Due to the architecture of the TMU, which is tightly coupled to the host interface [59, 60, 61, 62], the Kaiserslautern MoM architecture minimizes MoM/host interaction for the benefit of speed.

However, also a partly or fully sequential MCU implementation inside the host would be possible. If typical ALU operators are added to the RALU (standard circuit or taken from a cell library) even a hybrid architecture may be created, which adds von Neumann features to the MoM (also see section 3.3).

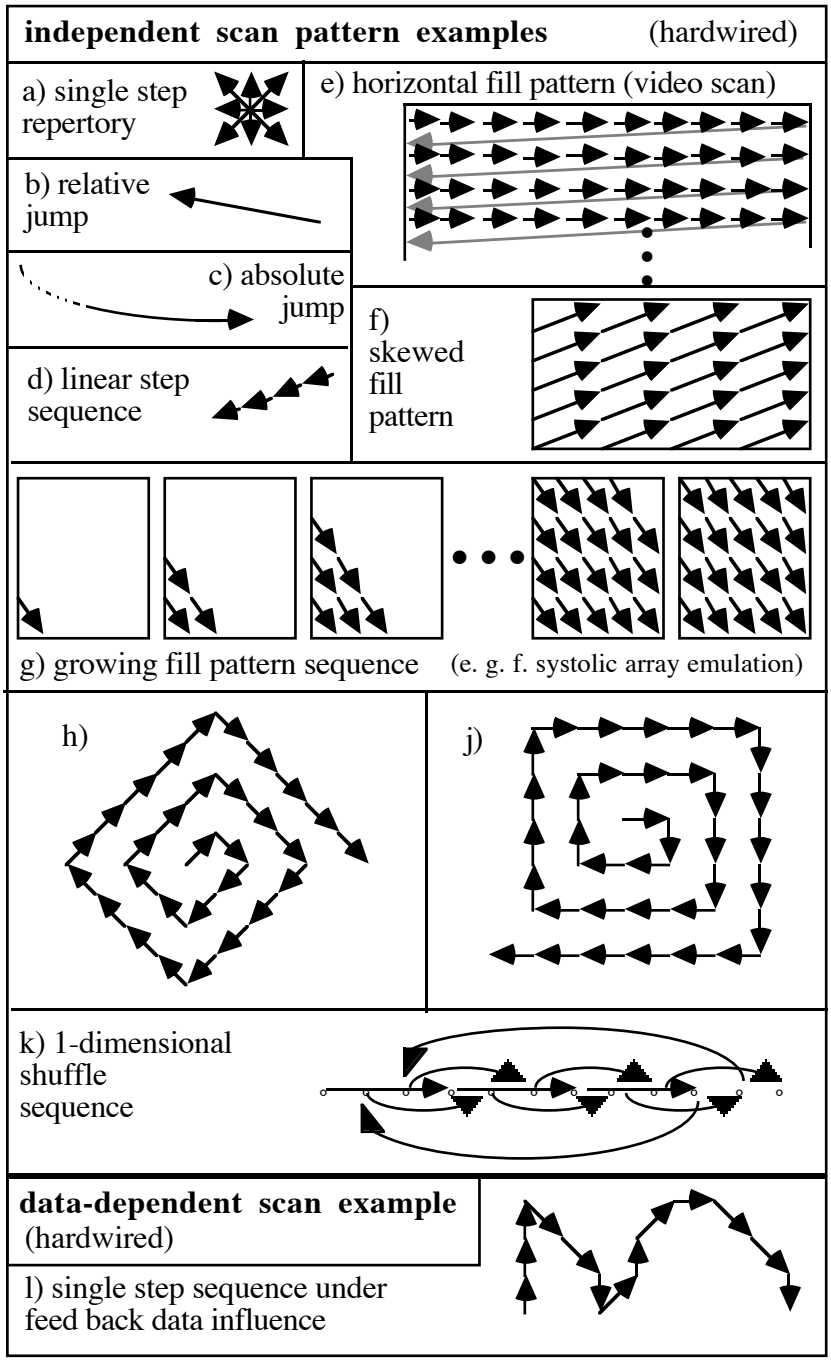

Figure 4: Some MoMputer Scan Pattern Scheme Examples

3.3 Advantages of Multiple Data Scan Caches

An Xputer can have 2, 3 or more scan caches, capable of moving independently at the same time. But such simultaneous cache movements may be synchronized by a common clock, i.e. by X cycles, such, that the coordination results in very attractive functionality. For example: cache no. 1 reads operands from data memory in a linear step sequence and cache no. 2 writes

results to data memory in a shuffle transfer sequence, such as e.g. in fast Fourier transform applications. This multiple scan cache strategy may result in substantial speed up. Also see the performance figures of the application demo example in fig. 5 e: for a 10 by 10 matrix multiplication dual scan cache use yields in an acceleration factor of sixteen over the single cache version. By the way: the combination of several X loops, including any mixes of parallel loops and series of loops, we call an *X task*.

3.4 MoM Implementation

Xputers can carry out any kind of data processing, since being as universal as von Neumann computers. However, Xputers achieve extraordinarily much better performance in processing such problems, where data dependencies can be efficiently mapped onto a two- or more-dimensional memory space. Such problems we call *map-oriented* problems. Fig. 5 gives a few acceleration factor examples of such "map-oriented" problems. For Xputers aping systolic arrays is of fundamental importance because all (pseudo-)systolizable problems are *map-oriented* problems. That's why Xputers are also very efficient for all kinds of problems, which may be efficiently implemented on systolic arrays [1, 37].

The speed benefit of the MoM is illustrated by comparison of the benchmarks of a few physical demos having been set up at Kaiserslautern using the MoM architecture. The comparison (fig. 5) is intended to be fair: a conservative technology MoM demo set-up is compared with a VAX 11/750, which also is technologically no more up to date. These benchmarks include quite a number of areas, such as: VLSI layout processing (design rule check, circuit extraction, compaction) [17, 18], image processing (pattern recognition, pattern matching, shrink, expand, contour following, segmentation, set operations) [18, 19], minimum-cost path Lee algorithm [20, 21], other non-numerics (sorting [23, 81], searching), bit map graphics [30], arithmetics processing [23], matrix operations, convolution, signal processing [22, 23, 24, 25, 26, 27] Fourier transform, other transforms, Viterbi algorithm [31], filtering, ape systolic systems [1, 37], emulation of neuro nets and other cellular systems [32, 33], encryption [34, 35], grid-based algorithms [36], and many applications [12, 13, 14].

row #	Operation (512x512 sized memory)	MoM* msec	VAX 11/750 sec	VAX 11/750 acceleration factor	68000 sec	68000 acceleration factor
a)	Grey-Image operations Binary-Image operations	60	7.8	>130	14.7	>240
b)	Erosion, Dilation, Skeleton, Edge Detection	210	38	>180	64	>300
c)	Design Rule Check(λ-based)	260	300	>1000	600**	>2000
d)	Minimum-cost Path (Lee)	150	20	>130	40**	>160
e)	10x10 Matrix multiplication ☐ aping systolic array ☐ using 2 caches	16 1	0.04	(>2) 40	0.15	(>9) 150
f)	Viterbi Algorithm	42	1**	>20	1.8**	>40
g)	bubble sort algorithms	100	4.2	>40	8.0**	80

* conservative TTL demo set-up, which has not been tuned ** estimated

Fig. 5: Some MoM acceleration factor examples

Dramatic improvements have been achieved in design rule check, routing and image preprocessing applications, for instance. E.g. the check of a one million square lambda NMOS design with grid-based design rules takes only one second, compared to many minutes or hours using mini computers of the von Neumann type. This is an acceleration by several orders of magnitude. There are many, many other application areas, where Xputer use yields such dramatic acceleration factors (e.g. see above list). Most benefit by Xputer use will be achieved e.g. in the areas of digital signal processing and image processing.

The MoM xputer architecture has been developed and implemented at Kaiserslautern, using an *eltec 68k System* [60, 61, 62] and an *IBM PC-AT* as a user interface [59]. Also an EDIF interface is available [63]. For the MoM essential standard hardware parts in nMOS technology have been designed, fabricated and successfully tested: scan cache [47, 48, 49, 50] and data sequencer [54, 55, 56, 58]. Also hardware and software of two host interfaces [59, 60, 61, 62, 63, 64] have been implemented.

4. Xputer (and MoM) Principles of Operation

Comparing with systolic array operation is a good illustration of Xputer principles of operation. For Xputer execution each (moving) systolic data stream is converted into data array stored at fixed memory location. On an Xputer not the data, but a data access location is moving along the *scan path* such, that the personalized RALU serves as a single systolic "basic processing element" (here used in a time multiplex mode). Instead of an array of processing elements working in parallel on the entire array of data stream variables only a single processing element's locus of activity travels through memory space to visit one variable after the other.

Thus by the Xputer the systolic parallelism is sequentialized. However, compared to a von Neumann approach still a dramatic acceleration is achieved, since the RALU is operating combinationally (i.e.: not sequentially) and also all other von Neumann bottle necks and overheads are avoided. The mapping of systolic arrays onto the MOM takes advantage of existent systolic array synthesis methods. The output of these systems serves as input to the MOM mapping procedure. This procedure falls into four steps:

1. Determine the cache size by the number of registers inside of one processing element plus the number of external registers.

2. Determine the memory map of the systolic array by tiling the cache onto the map-oriented memory of the MoM.

3. Map the work of one processing element onto the MoM's problem-oriented logic unit. The POLU (RALU) has to perform the computational work load of one PE as well as the data shifts to move data according to data stream directions.

4 Sequentialize the parallel work of the PE-array by using a special move pattern of the cache over the two-dimensional memory. During this sequence of cache moves two requirements have to be assured:

   •Only data items which are manipulated during a single time step of the systolic array must be manipulated during the according sequence of cache moves.

   •The cache moves have to be ordered in a way that no new data overwrites data which is still needed later.

The run-time performance of the real systolic VLSI array compared to the MoM implementation results in a slow-down factor of *(number of PEs + number of data items to be shifted in + number of data items to be shifted out)* for MoM execution.

We believe, that an elaborate Xputer application theory is already existing, which provides a growing rich repertory of methods and tools. It just has to be adapted to the Xputer principles: it is the rapidly developing theory of systolic processing. For a surprisingly wide variety of application areas (e.g. see in [22 - 27]) this currently very active scene has developed a powerful methodology of data dependency mapping onto the mostly 2-dimensional implementation space of VLSI resources. So this also is a kind of map-oriented processing, similar to the principles of the MoM. For further application problem preparation, needed for efficient MoM execution, these results have mainly to be remapped into a more refined time step scale.

This mapping is relatively simple, so that the derivation of the theory of MoM (and Xputer) processing from the theory of systolic processing is a relatively easy task. The conclusion is, that the theory of MoM application support is almost ready for use. We in Kaiserslautern, for instance, have implemented a systolic array synthesizer SYS^3 [67, 68], based on this theory. A modified version of SYS^3 also serves as part of our MoM development environment (MoM-DE, see next section).

4.1 MoM-DE: an Xputer Development System

Because of Xputer principles of operation the code to be generated by a MoMpiler is fundamentally different from the code generated by a conventional compiler. Whereas a compiler generates fine granularity sequential code to be laid down in a program store, a MoMpiler generates 2 kinds of code: combinational code for path-programming (configuring) the RALU, and only small amounts of large granularity sequential code for Xputer task sequencing.

For MoM application development *MoM-DE* (MoM Development Environment) has been implemented [76]. Also a special high level language *MoPL* (Map-oriented Programming Language) [75] has been implemented, which is an extended Pascalish procedural language. It has been extended by adding procedural features which are useful for Xputer programming, such as e.g. to concisely express generic data sequencing patterns (scan cache move patterns). MoM-DE includes a *MoMpiler* (phase II) accepting MoPL, and, which provides CAE tools needed for RALU personalization [69, 70, 71, 73] and loading MCU task register sets (for more details also see [58, 71]). It also includes a high level part (phase I) for optimization by pseudo-systolic data dependence remapping [46]. Phase I uses a modified version *SYS^3-MoM* of a systolic array synthesizer *SYS^3* [67, 68] having been implemented at Kaiserslautern.

MoPL includes *PaDL* (Pattern Description Language) [72, 73, 74] as a sublanguage, which efficiently supports Xputer programming for pattern matching applications. Image preprocessing [19], Lee Routing [10, 21] and Design Rule Check [18, 51, 52] demo applications on the MoM have been implemented with these tools. The PISA Pattern Editor [73, 74], a simple graphical editor, supports convenient editing and inspection of sets of reference patterns. For design rule check and similar applications also an automatic reference pattern generator and optimizer has been implemented [72], which accepts design rules expressed in a simple separation rule language. For Mead & Conway nMOS design rules [51] it has generated 256 reference patterns, for Lambda-based CMOS design rules [52] it has generated 800 reference patterns (which are RALU-executed completely in parallel).

Most parts of the MoPL language use the traditional procedural style, so that training effort for

introducing Xputer programming principles and practices is greatly minimized. This helps to smoothly embed Xputer supported paradigms into a conventional programming scene, so that it may be conveniently introduced to programmers with conventional background.

4.2 MoM and Xputer Paradigm

A computation problem can be expressed as a dependence graph (DG) with each node describing an operator and the directed edges describing its data dependency [40]. One can convert a DG into a computation graph (CG) spanning over the time-space domain, where DG nodes are annotated with time/space subscripts indicating when and where each of the operation nodes is mapped onto (a) processor resource(s). Each CG vertex represents a variable and each edge describes the data dependency of a vertex pair. Phase I of the MoM-DE converts systolizable DGs and many other type DG schemes into pseudo-systolic MoM-optimal data memory maps which make use of the powerful hardwired MoM repertory of generic data sequencing schemes *(move patterns)* [46].

A pseudo-systolic memory map is a computation graph (CG) that features optimum data sequencing: duality of memory map and move pattern: (re)arrange location of data within a memory segment such, that an optimum move pattern can be achieved. Most efficient are generic move patterns, which are hardwired into the Xputers data sequencer. The domain of the duality concept of pseudo-systolic memory map and generic data sequencing schemes - supported by Xputer data sequencing hardware and MoM-DE - by far exceeds the domain of systolizable DGs, since it also features individual jumps throughout data memory space. With a high level task sequencer use (see section 3.3) or a conventional host the universality of the von Neumann machine is reached.

It may be summarized, that for computation based on Xputers the development of a new theory is not needed (available). Experiments with MoM-DE have shown, that fundamentals for powerful application development support environments can be derived from scientific disciplines already existing. Normal programmers do not need to learn this theory. Due to its universality the Xputer may mainly be programmed also from conventional procedural programs expressed in languages like Pascal. That's why for standard applications no substantial retraining of programmers is required.

5. Conclusions

Xputer principles are a promising alternative to von Neumann: the narrow bandwidth standard ALU (its computation depends on fine granularity sequential code having been laid down in a program memory) is replaced by a very wide bandwidth hardware resources (where computation depends on compiled interconnect definition). In many important application areas Xputer use yields dramatic acceleration factors, which has been shown by a number of demo applications.

The MoM xputer architecture has been developed and implemented at Kaiserslautern, along with a number of demo applications and an experimental hardware demo set-up using an *IBM PC-AT* and an *eltec-68k-system* as user interfaces. Essential standard hardware parts in nMOS technology have been designed, fabricated and successfully tested. Many other Xputer architectures are feasible. Also the MoM architecture example combines von Neumann's flexibility and generality with speed advantages of specialized hardware solutions. It is much cheaper and much more universal than fully customized hardware solutions. Its standard parts are much

more simple, than a von Neumann microprocessor: the design is much less costly than designing a RISC processor.

We have shown, that new theories and methodologies needed for Xputer application support have been easily derived from the rapidly growing powerful theories of systolic processing and digital signal processing as well as from the methodology of integrated circuit design. Also an experimental application development environment (MoM-DE) has been developed and implemented at Kaiserslautern. The data-dependence-driven Xputer paradigm is also an excellent subject of modern visualization techniques. This data-driven paradigm and is typical way of memory space mapping may be more easily supported by visual presentations, than the more string-oriented von Neumann world of sequential processes.

The universality known from von Neumann computers generality has been achieved and also proven for an Xputer architecture [82]. Xputer principles still have a high potential for future improvements and continuous product innovations and thus for dynamic long range marketing strategies based on growing product families. Already today it will not be difficult to create a market for Xputer hardware and software.

By introducing Xputer innovative computational principles a new academic discipline in computer science research and engineering is supported, much of the theoretical and practical background of which is readily available to be adapted from the areas of digital signal processing, systolic arrays, wavefront arrays, as well as ASIC development and VLSI design. Also a new direction of research in compilers, programming languages, and its architectural support, which stresses fine granularity parallelism and data dependence analysis (e.g. see [78 - 80]) is efficiently supported by Xputer principles. So also progress in conventional computing will benefit from Xputer research.

6. Literature

[1] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: MoM - a partly custom-designed architecture compared to standard hardware, *(eds.:) W.E. Proebster, H. Reiner: Proceedings Comp Euro '89,* IEEE Press, Washington, DC, 1989.

[2] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: MoM - Map Oriented Machine; (*eds.) E. Chiricozzi, A. D´Amico: Parallel Processing and Applications;* North Holland / Elsevier, Amsterdam / New York 1988.

[3] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: MoM - Map-oriented Machine; *eds.: T. Ambler, P. Agrawal, W. Moore: Hardware Accelerators;* Adam Hilger, Bristol 1988.

[4] N. N.: Programmable Logic Devices, The Second Generation; *Electronics,* May 1988.

[5] R.Freeman: User-programmable Gate Arrays; *IEEE Spectrum*, p. 32-35, December 1988.

[6] Xilinx Corp.: *The Programmable Gate Array Design Handbook;* San José, Calif., 1986

[7] J.B. Dennis: Data Flow Supercomputers, IEEE Transaction on Computers, **13**, 11 (Nov. 1980)

[8] J. B. Dennis, D. P. Misunas: A Preliminary Architecture for a Basic Data Flow Processor, *Proc. 2nd Ann. Symp. on Computer Architecture, Houston, Tex., Jan. 1975*, ACM New York 1975, p. 126-132.

[9] H. W. Lawson, B. Magnhagn: Advantages of Structured Hardware; *Proc. 2nd Ann. Symp. on Computer Architecture (Houston, TX, 1975)*, IEEE New York 1975

[10] R. Hartenstein, G. Koch: The Universal Bus Considered Harmful; in [12];

[11] R.Hartenstein, R.Zaks: *Microarchitecture of Computer Systems;* North Holland, Amsterdam / New York 1975

[12] G. S. Sohi, S. Vajepeyam: Tradeoffs in Instruction Format Design f.Horizontal Architectures; *Proc. ASPLOS-III (Boston 1989),* IEEE Computer Soc.Press, Washington, DC 1989

[13] N. N.: Inside Technology; *Electronics,* April 1988

[14] D. A. Patterson: Reduced Instruction Set Computers; *Comm. ACM, **28**, 1 (Jan. 1985)*

[15] R. S. Nikhil, Arvind: Can Dataflow Subsume von Neumann Computing?; *Proc. 16th ISCA (Jerusalem 1989);* IEEE Computer Society Press, Washington, DC 1989

[16] N. P. Youpi, D. W. Wall: Available Instruction Level Parallelism for Superscalar and Superpipeline Architectures; *Proc. ASPLOS-III (Boston 1989),* IEEE Computer Society Press, Washington, DC 1989

[17] W. Nebel, *CAD-Entwurfskontrolle in der Mikroelektronik.* Teubner, Braunschweig, F. R. G., 1985.

[18] 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, CA);* IEEE Computer Soci. Press, Los Angeles et al. 1984

[19] R. Hartenstein, A. Hirschbiel, M. Weber: A Flexible Architecture for Image Processing; *Microprocessing and Microprogramming 21*, 1987.

[20] C. Lee: An Algorithm For Path Connections And Its Applications; *IEEE Trans. EC-10*, September, 1961.

[21] I. Velten: Implementierung des Lee-Algorithmus mit der MoM-Entwicklungsumgebung; Proj.-Arb., Kaiserslautern 1986

[22] W. Moore, A. McCabe, R. Urquhart: *Proc. 1st Int`l Conf. on Systolic Arrays (Oxford, UK, 1986);* Adam Hilger, Bristol / Boston 1986

[23] K. Bromley, S.Y. Kung, E. Swartzlander jr.: *Proc. 2nd Int`l Conf. on Systolic Arrays (San Diego, CA, 1988);* IEEE Computer Society Press, Los Angeles et al. 1988

[24] J. G. McCanney, J. McWhirter, E. Swartzlander jr.: *Systolic Array Processors,* Prentice Hall, 1989.

[25] E. Swartzlander jr.: *Systolic Signal Processing Systems;* Marcel Dekker, Inc., New York 1987

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

[27] S. Y. Kung, H. J. Whitehouse,T. Kailath: *VLSI and Modern Signal Processing;* Prentice-Hall, Englewood Cliffs 1985

[28] K. A. Collins, J. B. G. Roberts: Stereo Matching of Satellite Images with Transputers; in [23], pp. 175 - 182

[29] T. J. Fountain: The Use of Linear Arrays for Image Processing; in: [23], p. 183 - 192

[30] N. Charachorlo, et al.: A Million Transistor Systolic Array Graphics Machine; in: [23] p. 193 - 202

[31] K. A. Wen et al.: Implementation of Array Structured Maximum Likelihood Decoders; in: [23], pp. 227 - 236.

[32] S. Y. Kung: Parallel Architectures for Artificial Neural Nets; in [23], p. 163 - 174

[33] F, Blayo, P. Marchal: Generic Systolic VLSI Chip for Cellular Automata; in [24], p. 655 - 664

[34] National Bureau of Standards: *Data Encryption Standard;* Fed. Inf. Process. Standard Publ. 46, January 1977

[35] National Bureau of Standards: *DES Modes of Operation;* Fed. Inf. Process. Standard Publ. 81, January 1980

[36] K. Stüben, U. Trottenberg: Multigrid Methods: Fundamentals, Algorithms, Models, Problem Analysis and Applications; report, SFB 72, Univ. Bonn, F.R.G., Sept 1982

[37] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: Mapping Systolic Arrays onto the Map-Oriented Machine (MoM); in: [24]

[38] J. Holzer: *Hardware Implementation of a Pattern Recognizer,* Projektarbeit, Univ. Kaiserslautern 1988.

[39] R. Newton, D. Ditzel et al.: (panel on) CAD Tool Needs for High Performance Systems; Proc. 26th Design Automation Conference, June 1989, Las Vegas, IEEE Computer Society Press / ACM, 1989.

[40] W. T. Lin, J. P. Hwang: A High Speed Shuffle Bus for VLSI Arrays; *IEEE J. Solid-State Circuits **1** (1)* 41-68 (1983)

[41] R. W. Hartenstein, A. G. Hirschbiel, M. Weber: Mapping Systolic Arrays onto the Map-oriented Machine; in: [24].

[42] R.W. Hartenstein, A.G. Hirschbiel, M.Weber: *MoM - Map-oriented Machine, An Innovative Computing Architecture;* Interner Bericht, 181/88, Fachbereich Informatik, Universität Kaiserslautern, 1988.

[43] A.G. Hirschbiel: *PISA-Maschine: Eine spezielle Hardware für pixelorientierte Bildverarbeitung,* Diplomarbeit, Universität Kaiserslautern, 1985.

[44] A.G. Hirschbiel, M. Weber: *The Kaiserslautern MoM Architecture and its Implementation,* internal report, Univ. Kaiserslautern 1989

[45] R. Hartenstein, U. Welters: Higher Level Simulation and CHDLs; in: *(eds.:) W. Fichtner, M. Morf: VLSI CAD Tools and Applications;* Kluwer Acad. Publ., Boston, Mass, 1987

[46] M. Weber: Transforming Systolic Algorithms into MoPL Programs; internal report; Univ. Kaiserslautern 1989

[47] T. Becker: *PISA - CACHE als sequentielles Schieberegister - Beschreibung eines strukturierten Entwurfes,* Projektarbeit, Univ. Kaiserslautern 1986.

[48] R. Müller: *Multshift - eine Vollkundenschaltung für die Map-oriented Machine - Layout und Simulation,* Projektarbeit, Univ. Kaiserslautern 1987.

[49] H. Nicklaus: *Multshift - eine Vollkundenschaltung für die Map-oriented Machine - Spezifikation und Problemanalyse;* Projektarbeit, Univ. Kaiserslautern 1987.

[50] B. Mank: *Eine PLD Cache Implementierung für die MoM,* Projektarbeit, Univ. Kaiserslautern 1989.

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

[52] N. N.: *CMOS Design Rules für das E.I.S.-Projekt;* Fraunhofer-Inst. f. Mikroel. Systeme; Duisburg, F.R.G. 1988

[53] M. H. Brown: Perspectives of Algorithm Animation; *(eds.) E. Soloway, D. Frye, S. B. Sheppard: CHI'88 Conf. Proce. (Washington DC) - Human Factors in Computing Systems;* Association for Computing Machinery, New York, 1988

[54] A. Schaffer: *MCU - Move Control Unit for the MoM;* Diplomarbeit, Univ. Kaiserslautern 1989.

[55] A. Schaffer: *Single Step Control Unit,* Projektarbeit, Univ. Kaiserslautern, 1989.
[56] T. Mayer: *RANGECOUNTER - eine Vollkundenschaltung für die MoM (Map-oriented Machine),* Projektarbeit, Univ. Kaiserslautern 1987.
[57] H. S. Stone: Parallel Computing with the Perfect Shuffle; *IEEE TC-20 (1971),* p. 153 - 161
[58] R. Müller: Task Manager - A Hardware Task Concept for the MoM; Diplomarbeit, Univ. Kaiserslautern 1989.
[59] J. Westphal: MOM-Interface, ein Host Interface für die Map Oriented Machine, Diplomarbeit, Un. Kaiserslautern 1986.
[60] S. Burkhardt: *Implementierung eines MoM-Treiberprogramms für ein 68000-System unter OS-9;* Projektarbeit, Univ. Kaiserslautern 1989.
[61] N.N.: eltec-68k-system user manual; eltec GmbH, Mainz, F.R.G., 1986.
[62] T. Blüthner: VME-Bus Interface for the MoM; Diplomarbeit, Univ. Kaiserslautern 1989.
[63] S. Reibnegger: *EDIF Interface for the MoM;* Diplomarbeit, Univ. Kaiserslautern 1989.Univ. Kaiserslautern 1989.
[64] A.G. Hirschbiel: *Interface zum Anschluß einer Design-Rule-Checker-Maschine an die Siemens Rechenanlage 7.751/7.760,* Projektarbeit, Univ. Kaiserslautern 1983.
[65] T. Mayer: *POLU (Problem Oriented Logic Unit);* Diplomarbeit, Univ. Kaiserslautern 1989.
[66] C. Q. Zhu, P.-C. Yew: A Scheme to Enforce Data Dependence on Large Multiprocessor Systems; *IEEE Trans. on Software Engineering*, (June 1987) p. 726 - 739
[67] R. Hartenstein, K.Lemmert: SYS^3 - A CHDL-Based CAD System for the Synthesis of Systolic Architectures; *J. Darringer, F. Rammig: Hardware Description Languages and their Applications;* North Holland, Amsterd./ NY 1989.
[68] K.Lemmert: *SYS^3 - A Systolic Synthesis System around KARL,* Ph. D. Thesis, Univ. Kaiserslautern 1989.
[69] N.N.: *LOG / iC User Manual;* ISDATA, Karlsruhe, 1988.
[70] J.Holzer:*Optimal Programming Code for the POLU (Problem-Oriented Logic Unit);* Dipl.-Arb., Kaiserslautern 1989.
[71] W. Müller: Implementierung des MoMpiler-Codegenerators für die Move Control Unit der MoM; Diplomarbeit, Univ. Kaiserslautern 1989.
[72] M. Weber: *Ein Patterngenerator zur automatischen Referenzmustererzeugung;* report, Univ. Kaiserslautern 1985.
[73] C. Münster: *Implementierung eines graphischen Editors zur Eingabe der POLU-Operationen der MoM;* Diplomarbeit, Univ. Kaiserslautern 1989.
[74] N. N.: PISA Reference Manual, Univ. Kaiserslautern 1984
[75] P. Dewes: *Implementierung eines syntaxgesteuerten MoPL- Editors für die MoM;* Diplomarb., Un. Kaiserslautern 1989
[76] A.G. Hirschbiel, M. Weber: *Methodology of MoM application support,* internal report, Univ. Kaiserslautern, 198
[77] H.-M. Su, P.-C. Yew: On Data Synchronization for Multiprocessors; *Proc. 16th ISCA (Jerusalem 1989);* IEEE Computer Society Press, Washington, DC 1989
[78] P. Tang, P.-C. Yew, C. Q. Zhu: Impact of Self-Scheduling Order on Performance of Multiprocessor Systems; *1988 Int'l Conf. on Supercomputing*, (July 1988) p. 593 - 603
[79] D. Kuck, R. Kuhn, D. Padua, B. Leasure, M. Wolfe: Dependence Graphs and Compiler Optimizations; *ACM Conf. on Principles of Programming Languages (July 1981);*
[80] D. Kuck, A. Sansh, R. Cytron, A. Veidenbaum, C. Polychronopoulos, G. Lee, T. MacDaniel, B. Leasure, C. Beckman, J. Davies, C. Kruskal: The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance; *1984 Int`l Conf. on Parallel Processing;* Aug 1984.
[81] M. Weber: Using the MoPL Language for Programming the Bubble Sort Algorithm for the MoM; internal report, Univ. Kaiserslautern 1989.
[82] R.W. Hartenstein, A.G. Hirschbiel, M. Weber: Non-von-Neumann: is the Technology Transfer an Achievable Goal?; interner Bericht, Univ. Kaiserslautern 1989.