A Flexible Architecture for Image Processing

by
R.W. Hartenstein
A. Hirschbiel
M.Weber

April 1987
Kaiserslautern University
Computer Science Department
Postfach 3049
D-6750 Kaiserslautern, F.R.G.

Phone: xx49 631 205 2606

Keywords:

- image processing• flexible architecture
- pattern matching, recognition• segmentation, shrink,expand
- VLSI layout processing• routing

- VLSI impact on architectures
- Tools and methods for architecture design and description
- computer architecture
- Map oriented data processing

ABSTRACT

The paper describes an innovative computation resource concept which for a class of data processing problems is an alternative to the von Neumann machine. The 'processor' , called 'Map Oriented Machine' (MOM), used for this concept is faster than a von Neumann-type computer, however, it is substantially less expensive than a fully parallel hardwarized implementation using full custom or semi custom circuits. Instead of a program store with a program sequencer a personalized hardware is used, and, to 'program' this machine CAD tools are used instead of conventional compilers. The MOM concept is a compromise between the purely sequential von Neumann concept (sequential control part and sequential data part) and fully parallel solutions (parallelized control part and parallelized data manipulation side) insofar, as the control part has been parallelized, the data manipulation side, however, still uses a universal sequential access organisation.

b) State of the Art in this Field :

There are several approaches on speeding up image processing using VLSI implementations. All of them use a certain number of uniform processing elements (PEs) which are connected for intercommunication purposes. The arrangements of these PEs can be linear arrays (pipelines) or square arrays. The number of PEs determines the degree in parallelization achieved. The image data is pulsed through the PEs, if the implementation is based on a systolic array concept, or the complete image is stored in a PE square array and the PEs perform the same computations simultaneously based on a cellular array concept. The PEs are limited to a rather small range of operations or even they are able to perform just one single operation.

c) Stating of Problem :

There are mainly three methods to solve image processing problems.

- Von Neumann computers or multicomputers

  They use a classical software solution which is flexible, but rather slow, because they have a sequential program access and sequential data access. Multicomputer systems parallelize the programs, but are difficult to program and have a large administrative overhead. It is difficult to achieve the parallel operations the hardware resources seem to offer.
- Array Architectures using simple processing elements

  The use of special very simple processing elements make this approach very fast, but also very inflexible and expensive. These architectures use parallelized control parts and data manipulation parts. In most of these architectures the data is pulsed through the array and are known as systolic arrays.
- Pipeline Architectures

  One example for this type is the cytocomputer [STER81]. It is severely inflexible because of

a very rigid scanning scheme, it can perform only a videoscan scheme. Therefore it is not suitable for data dependent scanning schemes.

A relatively inexpensive but highly flexible accelerator for moderate performance requirements, for direct applications, as well as for simulations is not available.

d) Method Chosen for Solving the Problem :

The gap between the totally flexible but slow von Neumann solution and the very fast but expensive and inflexible array and pipeline architectures is filled with a medium speed, low cost solution called Map Oriented Machine (MOM). The basic idea of speeding up the algorithms is to parallelize the program access by combinational hardware. Unlike other solutions which pump the image data through fixed processor arrays, we use fixed data in a map-organized memory. A vari-sized cache is part of a universal data sequencer. This data sequencer has a simple but powerful instruction set. It is the universal part of the MOM. It can be moved across the image in arbitrary directions for example following any contour in the image or visiting all data by a self moving video scan command. The machine also permits the use of multiple data sequencers each following individual move commands, such as e.g. two data sequencers moving in opposite directions in aping systolic arrays.

The problem-oriented part is combinational so that sequential mechanisms and the von Neumann bottle-neck are avoided. A CAD environment to develop applications supports "programming" this part. It supports a variety of technologies ranging from EEPROM, EPROM, ROM over (E)PLA, PALs to semi- and full-custom solutions.

e) Results Received :

With our flexible hardware architecture the acceleration factors of specialized hardware solutions cannot be reached, but this factor is still between 100 and 10000 compared to conventional software programs. For example in a CAD application for VLSI design a 1024 square pixels image can be processed

in one second performing mighty instructions such as finding and marking errors. A substantial benefit, however, is achieved in the flexibility of the system. MOM is a compromise between special hardware and computer use, not losing the advantages of control side parallelism, but achieving more generality and flexibility.
Due to the flexible move features not only image preprocesses can be carried out, but also algorithms for other applications , such as e.g. design rule check, Lee routing, arithmetic problems, matrix operations, systolic algorithms and many other applications where the data can be efficiently stored in a two-dimensional memory.

Contents:

1. INTRODUCTION

To implement a data processing problem there are two extreme classes of solutions: the classical software solution running on a general purpose von Neumann-style computer on one side, and the other extreme would be a fully parallelized solution directly implemented on customized silicon. The classical solution is very slow since it is highly sequential, whereas the fully parallelized solution

yields the highest throughput. In many cases the fully parallel solution would be an overkill with respect to performance requirements. In many cases, however, an acceleration by about one or two orders of magnitude would be sufficient to meet the throughput requirements.
To achieve an acceleration by far less than 6 orders of magnitude (such as in [BHN85]) a much cheaper solution would be feasible, which may be implemented on the environment described in this paper. The discussion of general-purpose vs. special-purpose systems (e.g. see [SJGS85], [BLA84]) indicate, that the greatest disadvantages of the fully parallelized solutions are high design cost, very often too high with respect to market size (Figure 1.1). The concept presented here is a semi von Neumann computer, called Map Oriented Machine (MOM). Its acceleration factor is not as high as in the silicon solution, but anyhow it might be still up to between 100 and even 10000 in some cases, what is sufficient for many tasks.

Figure 1.1:
General purpose versus
special purpose machines

In the system described here the data access is still performed sequentially. However, the 'program side' has been parallelized. This can be achieved in

a way, that instead of a program a problem-oriented hardware is used (see Figure 1.2). The system may be personalized in a highly mechanized way to be adapted to a surprisingly wide range of applications. This is supported by a CAD tool, so that the design for a particular application is cheap. The analogon to the von Neuman computer is: instead of a compiler a CAD tool accepting a relatively high language is used to 'program' it (compare Figure 1.2).

Figure 1.2: MOM versus von Neumann Computer

In principle the system can solve almost any problem, which may be solved on a conventional computer (see Table 1 for examples). But for non-time-critical problems it may sometimes be better to use a conventional general purpose computer, because almost anybody knows to program it. However, if MOM performs an algorithm which has a two-dimensional Map Oriented (MO) organisation, such as e.g. in image processing, it is an efficient alternative to conventional software solutions: its programming is map-oriented, which may be easily visualized. The application's data structure is represented by a two-dimensional data map.

VLSI layout processing: design rule check, circuit extraction, compaction [HNW84].

image preprocessing: pattern recognition, pattern matching, shrink, expand, contour following, separation, colour mixing, set operations, etc. [PU82].

arithmetic problems:multiplication, addition, incre- menting, decrementing, etc.
routing: the Lee algorithm [LCY61, BS81].
ape systolic arrays: very wide variety of applications
matrix operations: multiplication, conversion, etc.
convolution type algorithms: FFT, convolutions, and similar problems
ape map oriented logic: PLA, PAL, Weinberger array, Kolte array, Lopez/Law Dense Gate Matrix etc.

Table 1: examples of data processing problems feasible for MOM execution.

2. ARCHITECTURE AND MODE OF OPERATION

The basic architecture of the MOM is shown in Figure 2.1. It consists of a data memory, a vary size cache, a problem oriented logic unit (POLU) and a move control unit (MCU). All these parts are connected via an interface to a host computer. The vary-size cache, together with the MCU, form the "data sequencer" (compare Figure 1.2 b). Also more than one cache may be used, which be very useful (see next section). The POLU is the problem-specific section, whereas all other modules are parts of the universal section of MOM.

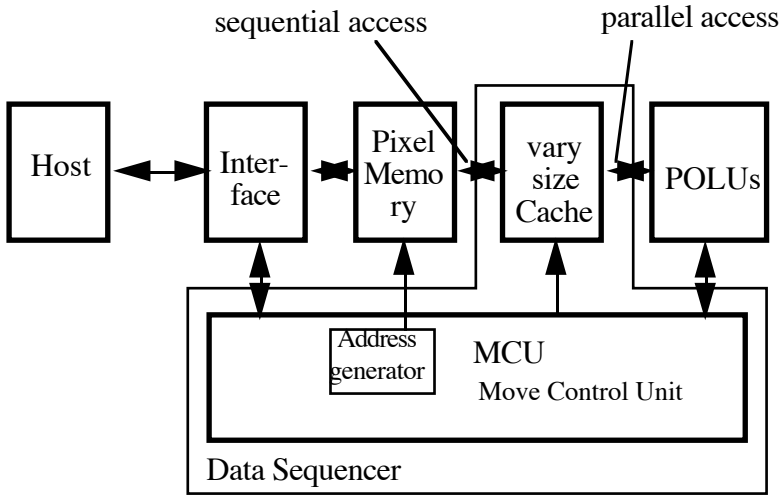

Figure 2.1: MOM Architecture

2.1. The Universal Part

The data sequencer in the MOM consists of the vary-size cache and the move control unit (MCU, see Figure 2.1). It is described in detail in [HIR85], and its design is straight forward, that's why we do not show any details within this paper. The cache is a parallel accessible data store. Its contents is a copy of the addressed part of the memory. It could be compared with part of as the register file of a von Neumann style computer. The address is generated by the MCU, and, since the memory is organized in a two dimensional way, there are two parts of the address, the x part and the y part. Each pair addresses a word in the memory, which is called a “pixel“.The memory is patitioned into single-bit organized memory plans. Thus the memory is a multiple bit map memory, and, it is called "pixel memory". When we are talking about "moving the cache over the pixel memory" we mean that the cache buffers the contents of a "window" , and after a new address is selected the cache is loaded with the contents of the next position of this window of the memory. To do this there are two different move primitives:

- **jump absolute (x,y)**
- **jump relative (x,y).**

The set of move instructions is derived from the von Neumann concept, which makes the MOM concept very universal. For example, if you would like to meet all the contents in the memory you could start in the lower left corner and step through it by single steps to the right with the following primitives:

jump absolute (0,0)
while not end of memory do **jump relative (1,0)** endwhile.

Because of an auto-wrap-around at the end of one memory line this is all you need to do. This mode is called video scan and is depicted in Figure 2.2. To recognize the end of a memory line and the end of

memory the size of the memory is set during an initilization by storing the minima and maxima for the x and y direction. The cache is variable in its size (Figure 2.3) to adapt it to its application. It is connected to the problem oriented logic unit (POLU) which observes the cache's contents and delivers an application specific result. So a read/modify/write cycle is organized for cooperation between POLU and cache on one side, and the pixel memory on the other side.

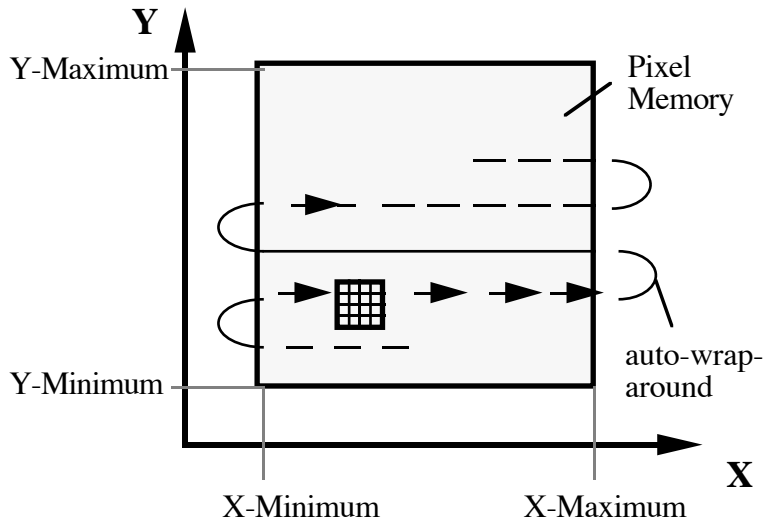

Figure 2.2: Video Scan

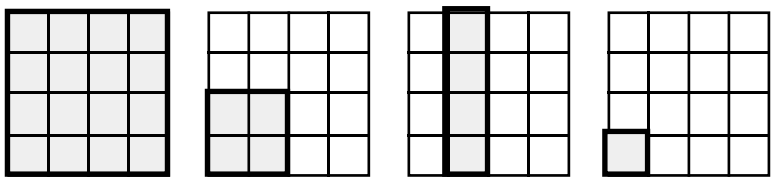

Figure 2.3: Vary Size Cache

Not only two-dimensional memory organisations may be used but also other multi-dimensional organisations. The difference to the von Neumann Computer is that move primitives are used at data side, and not at the program side and, that the memory-space is multi-dimensional, instead of being one-dimensional. The data sequencer hardware of MOM has been implemented such, that two fundamentally different moving strategies may be applied: schematic moves such as e.g. video scan, and

data-depentant moving such as e.g. in curve following. For the latter strategie POLU output is used to control the MCU.

2.2. The "programmable" Part

The programmable part of the MOM is the POLU. It does not hold a sequential program, but is a CAD-generated special purpose hardware which is obviously faster than a sequential program. It also consists of predefined hardware patterns from a CAD library which can be used within the programmable part and can be seen as some sort of "standard functions". Some examples are:

- Arithmetic functions like addition, multiplication
- Image preprocessing such as shrink, expand and set operations

By means of a select code one of these parts can be selected (Figure 2.4), each performing a different application by comparing the cache's contents with a set of reference patterns. The result is a list of numbers of the matching patterns. This list then is interpreted, and as a result several instructions are given. These are:

- the result pattern to be written back to the cache which will be stored back to the memory at the next move.
- the next move command for the cache.
- the select application code.

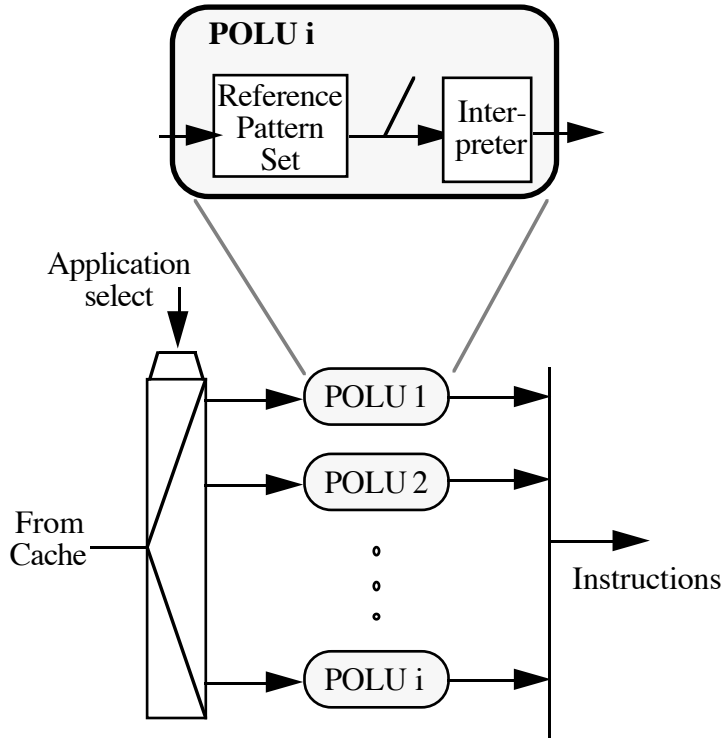

Figure 2.4: Problem oriented Logic Units

The "program" for the MOM therefore is stored in the reference pattern set and so this is the part which has to be programmed for a new application (see section 3). This "program" can be stored in ROM, PLA, PAL, gate arrays, other semi custom or full custom devices or in mixtures of all these. For fast turn-around prototyping EPROM, PROM, etc. may be used. Each application is performed in several cycles. One cycle consists of (compare Figure 2.5):

- load the cache
- compare the cache with the selected reference patterns in the POLU
- store result pattern in cache
- select application
- set cache size

• set memory area size
• move cache (= store cache to memory and load it with the new addressed data)

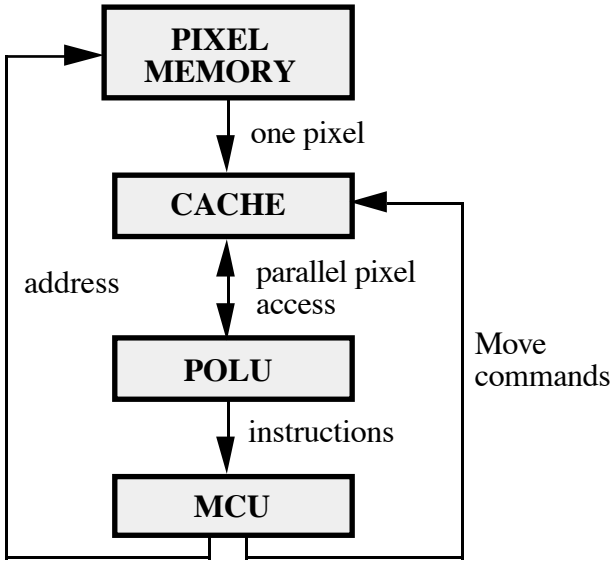

Figure 2.5: MOM Cycle

There are "no operations" for each cycle part so not all of these are performed at any cycle. For example different operations can be performed at the same position in the memory (no cache movement) or the cache will be moved and its contents remains unchanged. Before the program can start the host must set up the MOM. This is:

• preset the 4 limit registers for the memory
• load the bit map memory

• select a initial cache size
• select a initial application
• move the cache to a initial position

3. GENERATION OF NEW POLUS

To program the MOM a new set of reference patterns and a new interpreter has to be constructed. A CAD-Toolbox supports this. There is a comfortable reference pattern editor and a translator to generate the program code to store these in RAMs or ROMs. The way an algorithm is constructed for MOM is different from programming a von Neumann computer. It is important to arrange the data in the pixel memory in an efficient way. Then we have to find local operations to be performed on this data. This will be clarified with some examples in the next section. Local data can be accessed parallel that means it can be observed, compared and changed very quickly, so this is what can be spent in large numbers in opposite to a von Neumann computer, where every comparison of and access to data costs a lot. Sometimes it is neccesary to hold the same data in many copies in the memory to be able to use just local operations, what can be accepted because of decreasing memory costs. After the algorithm is found it can be installed on MOM with the help of some CAD-tools.

4. APPLICATIONS AND ALGORITHMS

The MOM is capable to execute almost any kind of data processing. Anyhow there are problems more or less feasible for MOM execution. Groups of problems with their algorithms, which show the power of MOM are described as are problems, that can be solved, but where MOM is less powerful than an ordinary von Neumann machine [VNEU61] with a mighty ALU.

4.1. Bit Manipulations and Set Operations

Since the pixel memory is a multilayer bit map it is obviously an excellent representation of two-dimensional images. This fact provides the first class of processes suitable for MOM. The simplest problems in this class are bit operations (boolean instructions between different layers in a pixel) and set operations. Sets can be stored as areas of set bit in different pixel layers, and boolean instructions are

applied to compute functions like 'contained', 'intersection', 'union' or 'difference'. These operations require just an one-pixel cache to be moved over the memory. The movement strategy is to visit each pixel exactly once and to match it with the set of reference patterns. This is done by a video scan, i.e. the cache is moved from the left bottom corner, row by row, to the right top corner of the pixel memory. Each reference pattern represents a possible input-instance of the entire boolean operation, the corresponding result pattern represents the result of the operation applied to the input-instance. Figure 4.1 gives an example of a set operation.

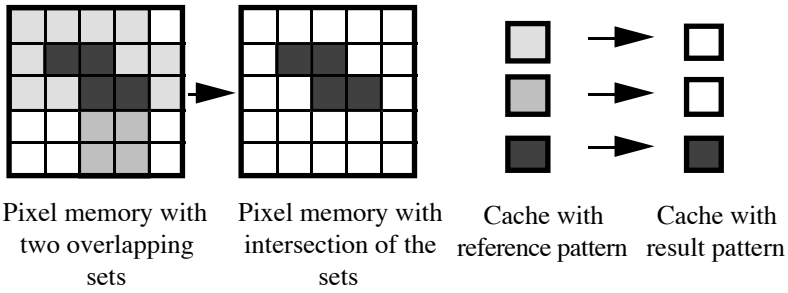

Figure 4.1: Set operation 'intersection'

4.2. Image Preprocessing

In the field of image preprocessing these bit and set operations can be used to mix colours stored in different layers or to make colours invisible. Other image preprocesses need the evaluation of neighbourhood information. To meet this requirement a bigger cache is used. In most cases a 3-by-3 pixel wide cache is sufficient to get local informations. Shrinking and expansion of image structures are examples for this type of applications. To expand an object it is necessary to know whether the neighbour pixel is already blocked or whether it is still free to expand the object (see Figure 4.2). The two reference patterns and their result patterns are the first two encountered when performing a video scan from the left bottom to the right top of the pixel memory section in the example of Figure 4.2. Naturally there are more different patterns necessary to provide a complete expansion as shown in the example.

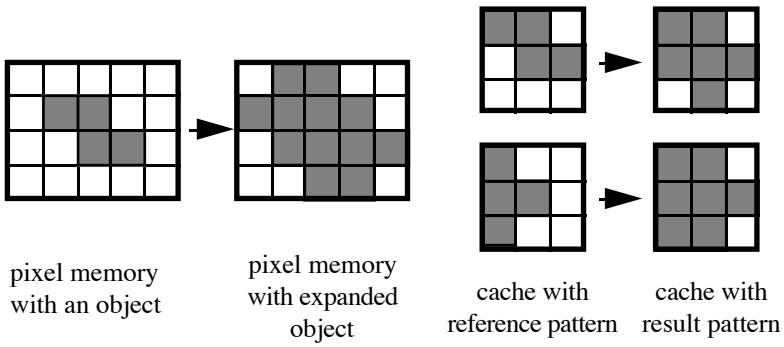

Figure 4.2: Expansion of an object (an image pre-processing example)

A variation of the expansion algorithm could be not to perform a video scan movement strategy, but to follow the outline of the object and add new pixels at the edge of the existing object. This explains that problems can be realized in different MOM algorithms, either in different reference patterns or in different movement strategies or in both.

4.3. Layout Processing

Another suitable field for MOM executions is VLSI layout processing. Since layout can be excellently stored in multi-coloured bit maps the ability to solve problems by scanning a local window over the layout data is obvious. Classical problems feasible for MOM are design rule check, Lee routing or circuit extraction.

4.3.1. Design Rule Check Application of MOM

To carry out a design rule check of the layout of integrated circuits it is necessary to have a cache, which is one pixel larger than the largest design rule, e.g. if the rule "minimum metal spacing is three lambda" has the biggest lambda factor of all design rules, the cache has to have a size of 4-by-4 pixels (one pixel is equivalent to one lambda unit) in order to cover this rule, and, at least one of its direct neighbour pixels [HNW84]. The entire design rule check is done by a single video scan over the layout. The reference patterns represent all possible design

rule violations. A match to one or more of these patterns indicates a violation. For all reference patterns there is only one result pattern which directly marks the location of a design rule violation in using a special error layer at the position where a reference pattern has matched. Since design rules are locally bounded only relatively few reference patterns are needed. For the Mead-&-Conway design rules [MECO80] only 258 patterns are needed for example. Figure 4.3 shows an example of a design rule violation and its detection by a reference pattern. Clearly only orthogonal layout structures can be processed, because the underlying pixel memory allows only orthogonal structures to be stored.

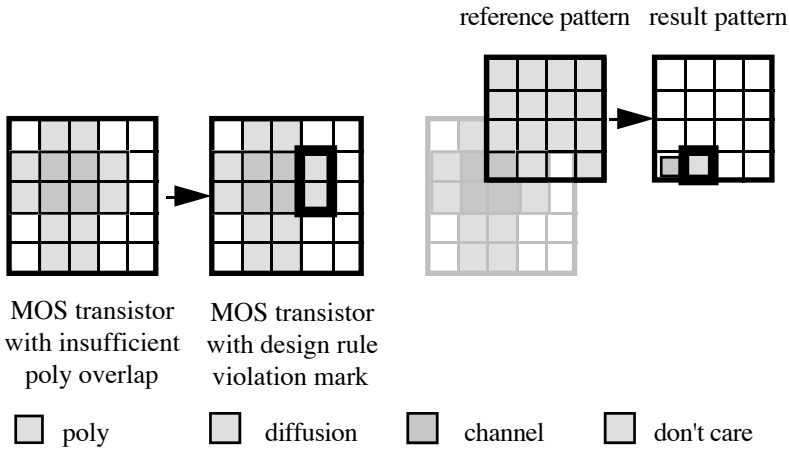

Figure 4.3: Design rule check example

4.3.2. Lee Routing Application

Lee routing [LCY61] is another MOM application example in CAD for VLSI. Starting from a specially marked pixel (the start cell), the shortest path to a target cell is searched by propagating arrows from the start cell, thus propagating a wavefront to the target cell. By backtracking the path, back along the arrows, the shortest connection between the two points is achieved. This routing process requires the combination of different sets of reference patterns and different movement strategies [VEL86]. First a single pixel cache is video-scanned over the image to find the start cell. The waveform expansion of arrows requires a 3-by-3 pixel cache to get information about arrows positioned in neighbour pixels.

The movements of the cache are somewhat spiralic outward from the start cell until a certain matched reference pattern indicates that the target cell is found. In the third part of the routing a single pixel cache follows the direction of the arrows back from the target to the start cell to complete the routing process. Altogether only 33 reference patterns and 33 result patterns are needed for the Lee routing algorithm with MOM. Figure 4.4 shows a snapshot during the waveform expansion. The movements are anticlockwise. The two reference patterns match at the numbered positions (1, 2) in the example. The waveform has the shape of a growing diamond until the target cell is reached (Figure 4.5).

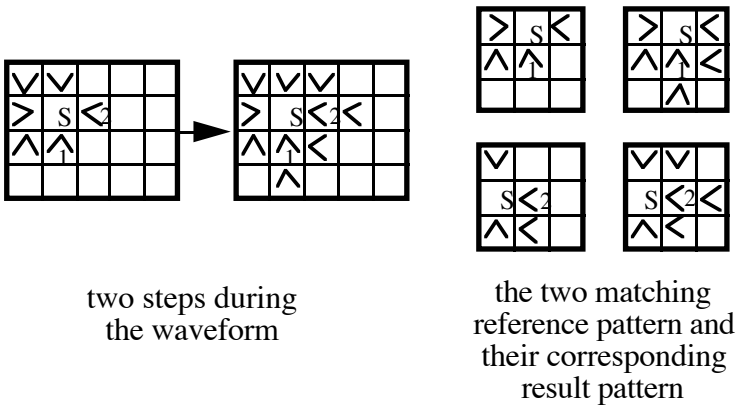

Figure 4.4: Lee routing example (snapshots)

Figure 4.5: Lee routing example (waveform scheme illustration)

4.3.3. Circuit Extraction

MOM can also perform a circuit extraction by tracing layout structures with a 2-x-2 cache to receive a netlist of the microchip [NEB85]. The data structure to hold this netlist and the capacitance of all nodes have to be stored outside the MOM architecture. MOM provides the entries for this data structure.

4.4. Aping Matrix-Oriented Logic

Matrix-oriented Logic (MOL, see [GHHO86]) are logic circuits which may be specified by means of a personality matrix, e.g. PLA, Weinberger array [WEI67], dense gate matrix [LOLA80] and others. MOM can be used to simulate and verify this symbolic description given by the personality matrix which is stored in the pixel memory. By means of the AND matrix of a small PLA the principle to ape matrix-oriented logic is illustrated (Figure 4.6). Each function in the AND matrix is evaluated by one scan, left to right, over this entire row of the personality matrix using a 2-by-1 pixel cache. The result layer is completely preset with '1' at the beginning of the execution. In each scan step the result is passed on to the right neighbour in the result layer. If after the scan the '1' is still present at the right edge of the row, the corresponding function in the PLA is true.

Figure 4.6: Aping a PLA

Other matrix-oriented logics can be processed in the same way though the reference patterns and result patterns get more sophisticated.

4.5. Non-CAD Applications

Driven by our own processing needs we mainly used the MOM machine for accelerator applications in CAD for VLSI. However, there are many other feasible applications of the MOM other than in VLSI design. Especially, when MOM is working in twin-cache mode (or sometimes in other multiple-cache mode) all types of convolution-like processing can be directly mapped onto the multi-dimensional MOM address space. Since the MOM concept is an excellent resource for low-cost aping of systolic arrays.

4.5.1. Aping systolic arrays

Systolic arrays are widely used to speed up algorithms by using hardware [KUN79], [FOKU80]. Generally an array or a matrix of uniform processing elements (PE's) are linked together which are able to transfer information to their neighbour processing elements. Data thus is pulsed through these PE's and a pipelined processing is achieved, such that all PE's compute their current data parallel. It is possible to ape these systolic arrays with the MOM machine. The cache substitutes one PE, the scan of the cache substitutes the systolic pulse of the data through the PE. Reference patterns and result patterns have to perform the same function as the PE of the systolic array does. Figure 4.7 illustrates the transfer of the problem from systolic arrays onto a MOM.

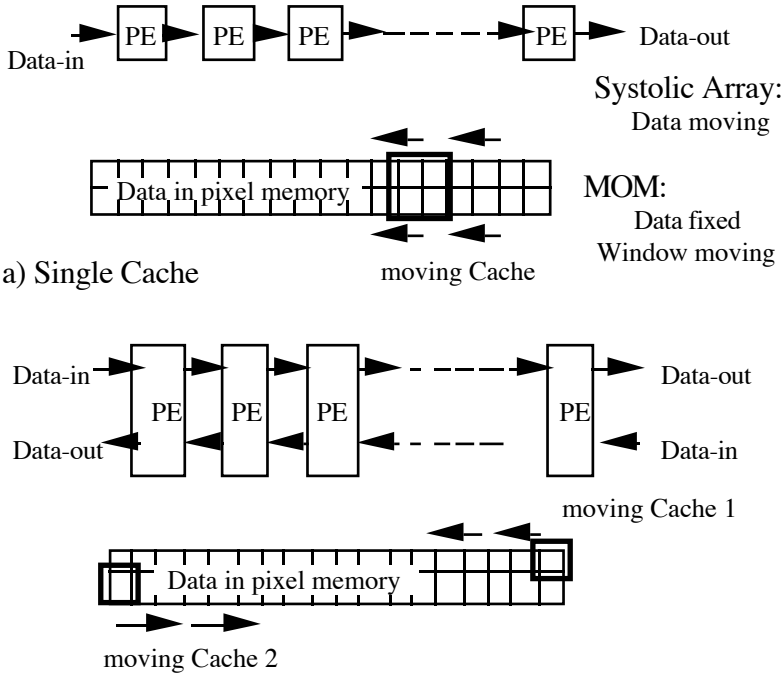

Figure 4.7: Aping of systolic arrays

That's why MOM can also be used to ape systolic arrays for "real applications", whenever the high performance of a systolic array is not needed, as well as to provide an environment for the development of systolic arrays, or, for programming such systolic arrays, which can be programmed or can be personalized in another way. In the latter application MOM would be used as a CAD tool (like a programmable systolic array compiler, or a systolic array compiler) within a toolbox for the design of systolic architectures.
So, as a matter of fact, everything which fits onto a systolic array implementation can be directly mapped onto the MOM system. That's why also the following application areas [KUN82] are feasible for using MOM resources.

Signal and image processing:

- FIR, IIR filtering, 1-D convolution
- 2-D convolution and correlation
- discrete Fourier transformation
- interpolation

- 1-D and 2-D median filtering
- geometric warping

Matrix arithmetic:

- matrix-vector multiplication
- matrix-matrix multiplication
- matrix triangularization
- QR decomposition
- solution of triangular systems

Non-numeric applications:

- data structures (sorting, searching)
- dynamic programming
- graph algortihms (transitive closure, connected components)
- language recognition (string matching, regular expression)
- relational data-base operations

For references to these application areas see [KUN82]. This list shows that the MOM concept has a very wide range of applications. Especially those applications which are based on repetitive computations on large sets of data are feasible for MOM.

4.6. Applications less feasible for MOM

MOM is inferior to von Neumann-style machines when this machine is able to compute the desired task in only one or a few execution steps with its ALU. Besides such applications which need huge dynamic intermediate memory allocations often are more suitable for von Neumann machines. Nevertheless there could be the possibility to run MOM as an accelerating extension of a von Neumann machine with this von Neumann machine as the host of MOM.

5. EXTENSIONS AND FUTURE ASPECTS

To cover a wider area of data processing problems the MOM having been introduced here, can be extended to have two or more caches running simultaneously. This leads to two data selections at a time,

which can be interleaved and it leads to two parallel data accesses. E.g. to multiply two matrices one cache scans the first matrix row by row, the second cache the second matrix column by column. The two caches exchange information via the POLU to provide a multiplication of the two matrices.
A main topic of future work would be to design a high level language as an aid to program MOM easily, safely and quickly. Such a language should include features for the description of the shape of reference patterns and result patterns as well as the behavioural description of the cache movements. Via existing CAD-tools and a language interpreter the described algorithms could be transformed into new POLUs to extend MOM.
Since the cost for primary memory is decreasing more and more, it will be affordable soon to have a main memory of a gigabyte or more. Thus very large data maps could be kept in primary memory (the pixel memory). Also very large pixels could be feasible, such as using pixel word formats of up to a hundred bits or more. The swapping rates will be reduced drastically. This is a good development also for MOM applications.

6. CONCLUSIONS

With the map oriented machine MOM a new approach to speed up algorithms has been introduced. MOM combines the flexibility advantages of von Neumann-type machines with the speed advantages of special hardware solutions. MOM is slower but more universal than fully customized special hardware, however it is more flexible, i.e. faster than a von Neumann-type machine, but less general in its usage. The MOM has been developed at Kaiserslautern University, F.R.G., where a prototype, which has been personalized for a design rule check application demo, is running successfully. In addition to this 'real' MOM, a software simulation system for the map-oriented machine has been implemented, which also serves as a MOM application development environment and CAD toolbox. This software version of MOM is called the PISA program [HNW84]. The speed benefit in using MOM varies

from application to application. A dramatic improvement has been achieved in design rule check applications. E.g. the check of an one million square lambda NMOS design with Mead-&-Conway design rules takes 1 second, compared to minutes or hours in using super mini computers.
We have illustrated a flexible computing resource and accelerator environment concept for a wide variety of applications. It may be used for CAD applications, such as e.g. in VLSI design and verification. It may also be used for 'real' data processing in a wide variety of applications. A lot of work has to be done to explore application methodologies of this new type of computational resource.

7. LITERATURE

[BHN85]K. Bastian, R. Hartenstein, W. Nebel: *VLSI-Algorithmen: Innovative Schaltungstechnik statt Software - Shuffle Sort*, VDI-Berichte Nr. 550, 1985

[BLA84]T. Blank: *A Survey of Hardware Accelerators used in Computer-Aided Design*, IEEE Design and Test of Computers, August 1984

[BS81] M.A. Breuer and K. Shamsa: *A Hardware Router*. In: Journal Of Digital Systems, Vol.4, Issue 4, 1981.

[FOKU80]M.J. Foster, H.T. Kung: *The Design of Special-Purpose VLSI Chips*, Computer, January 1980

[GHHO86]R. Gebhardt, R. Hartenstein, R. Hauck, D. Oelcke: *Functional Extraktion from Personality Matrixes of MOL (Matrix Oriented Logic) Circuits*, report, Kaiserslautern University, 1986

[HIR85]A. Hirschbiel: *PISA Maschine, Eine spezielle Hardware für pixel orientierte Bildverarbeitung*, report, Kaiserslautern University, 1985

[HNW84]R.W. Hartenstein, R. Hauck, A.G. Hirschbiel, W. Nebel, M. Weber : *PISA, A CAD Package And Special Hardware For Pixel-Oriented Layout Analysis*. In: ICCAD - 84, Digest Of Technical Papers, Santa Clara, 1984.

[KUN79]H.T. Kung: *Let's Design Algorithms for VLSI*, Caltech Conference on VLSI, 1979

[KUN82]H.T. Kung: *Why Systolic Architectures?*, Computer, January 1982

[LCY61]C.Y. Lee: *An Algorithm For Path Connections And Its Applications*. In: IEEE Trans. on Electronic Computers, vol EC-10, pp. 346-365, Sep. 1961.

[LOLA80]A. Lopez, H. Law: *A Dense Gate Matrix Layout Method for MOS VLSI*, IEEE Journal of solid-state circuits, Vol. SC-15, No. 4, Aug. 1980

[MECO80]C. Mead, L. Conway: *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[NEB85]W. Nebel: *CAD-Entwurfskontrolle in der Mikroelektronik* (in german), B.G.Teubner, 1985

[PU82]K. Preston jr., L. Uhr: *Multicomputers and Image Processing*. Academic Press, 1982.

[SJGS85]L. Snyder, L. H. Jamieson, D. B. Gannon and H.J. Siegel: *Algorithmically Specialized Parallel Computers*. Academic Press, 1985.

[VEL86]I. Velten: *Implementierung des Lee-Algorithmus auf MOM* (in german), report, Kaiserslautern University, 1986

[VNEU61]J. von Neumann: *Collected Works, Volume 5*, Pergamon Press, 1961.

[WEI67]A. Weinberger: *Large Scale Integration of MOS Complex Logic: A Layout Method*, IEEE Journal of Solid-State Circuits, Vol. SC-2, No. 4, Dec. 1967

d) References :

[BAT80]K. E. Batcher: *Design of a massively parallel processor,* IEEE Trans. Comput. C-29 836-840, 1980.

[BHN85]K. Bastian, R. Hartenstein, W. Nebel: *VLSI-Algorithmen: Innovative Schaltungstechnik statt Software - Shuffle Sort*, VDI-Berichte Nr. 550, 1985

[BLA84]T. Blank: *A Survey of Hardware Accelerators used in Computer-Aided Design*, IEEE Design and Test of Computers, August 1984

[BS81] M.A. Breuer and K. Shamsa: *A Hardware Router*. In: Journal Of Digital Systems, Vol.4, Issue 4, 1981.

[FOKU80]M.J. Foster, H.T. Kung: *The Design of Special-Purpose VLSI Chips*, Computer, January 1980

[HIR85]A. Hirschbiel: *PISA Maschine, Eine spezielle Hardware für pixel orientierte Bildverarbeitung*, report, Kaiserslautern University, 1985

[HNW84]R.W. Hartenstein, R. Hauck, A.G. Hirschbiel, W. Nebel, M. Weber : *PISA, A CAD Package And Special Hardware For Pixel-Oriented Layout Analysis*. In: IC-CAD - 84, Digest Of Technical Papers, Santa Clara, 1984.

[KUN79]H.T. Kung: *Let's Design Algorithms for VLSI*, Caltech Conference on VLSI, 1979

[KUN82]H.T. Kung: *Why Systolic Architectures?*, Computer, January 1982

[LCY61]C.Y. Lee: *An Algorithm For Path Connections And Its Applications*. In: IEEE Trans. on Electronic Computers, vol EC-10, pp. 346-365, Sep. 1961.

[MECO80]C. Mead, L. Conway: *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[NEB85]W. Nebel: *CAD-Entwurfskontrolle in der Mikroelektronik* (in german), B.G.Teubner, 1985

[PU82]K. Preston jr., L. Uhr: *Multicomputers and Image Processing*. Academic Press, 1982.

[SJGS85]L. Snyder, L. H. Jamieson, D. B. Gannon and H.J. Siegel: *Algorithmically Specialized Parallel Computers*. Academic Press, 1985.

[STER81]S. R. Sternberg, *Parallel architectures for image processing,* in "Real/Time Par-

allel Computers" (M. Onoe, K. Preston, Jr., and A. Rosenfeld, eds.) pp. 347-359. Plenum, New York 1981.