MOM - Map Oriented Machine

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

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

ABSTRACT

In this paper we describe a new architecture, called 'Map Oriented Machine' (MOM), which fills the gap between the totally flexible, but slow von Neumann computer and the very fast, but expensive and inflexible fully parallelized solution directly implemented on customized silicon. Some applications of MOM, presented here, show that algorithms which have a map-oriented organization, such as image processing, can be implemented in a very efficient way on MOM. The basic idea of speeding up the algorithms is to parallelize the program access by combinational hardware, whose development is supported by some CAD tools.

MOM - MAP ORIENTED MACHINE

R.W. HARTENSTEIN, A. HIRSCHBIEL, M. WEBER

INTRODUCTION

For some algorithms, such as e.g. image processing algorithms, a classical software solution on a von Neumann computer is too slow. On the other hand a hardware solution on full custom circuits based on a cellular array concept is expensive and very inflexible. The discussion of general-purpose versus special-purpose systems (see Snyder et al 1985, Blank 1984 and Figure 1) indicates, that the main disadvantage of the fully parallelized solutions is high design cost, often too high with respect to market size.

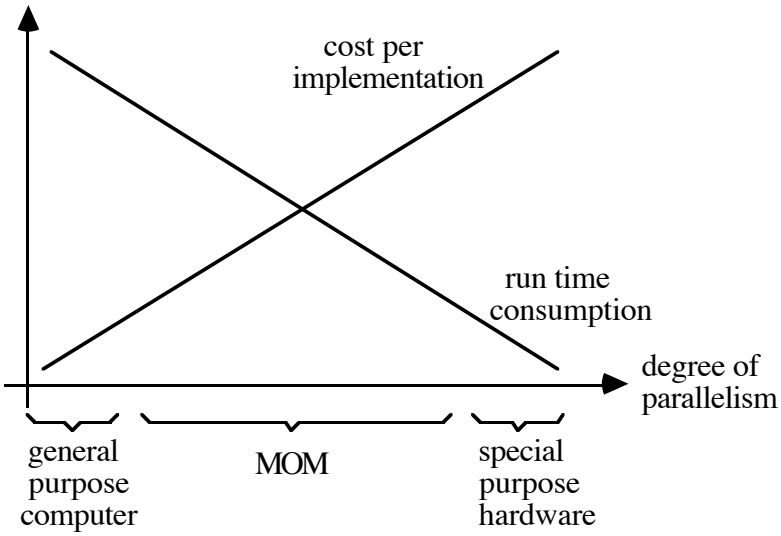

Figure 1: General purpose versus special purpose machines

The two main approaches to solve data processing problems are:

• Von Neumann computers or multicomputers
These general-purpose architectures use a classical software solution which is flexi-

ble, but rather slow, because they have sequential program and data access. Multicomputer systems parallelize the programs, but are difficult to program and have a large administrative overhead (Preston and Uhr 1982, Young and Fu 1986). It is difficult to achieve the parallel operations the hardware resources seem to offer.

- Array Architectures using simple processing elements

  This approach has the intention to speed up processes using VLSI implementations. They use a certain number of uniform processing elements (PEs) which are connected for intercommunication purposes. The arrangement of these PEs can be linear arrays (pipelines) (Sternberg 1981) or two-dimensional arrays. The number of PEs determines the degree in parallelization achieved. The data is pulsed through the PEs, if the implementation is based on a systolic array (Kung 1979), or the complete data is stored in a PE square array and each PE performs the same computations. The PEs are limited to a rather small range of operations or even they are able to perform just one single operation. The use of special very simple processing elements (PE) makes this approach very fast, but also very inflexible and expensive. These architectures use parallelized control parts and data manipulation parts.

The acceleration factor from the fully parallel hardware solution to the software solution usually exceeds more than five orders of magnitude, but in many cases a speed up of three or four orders of magnitude would be sufficient.

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. In the universal part of MOM a data sequencer provides controlled access to the data memory. The problem-oriented part is combinational to avoid sequential mechanisms. The computational principle is based on a parallel match of the cache with reference data. The system may be personalized in a highly mechanized way to be adapted to a surprisingly wide range of applications. A CAD environment to develop applications supports 'programming' this part. It supports a variety of technologies ranging from (E)EPROM, ROM over (E)PLA, PALs to semi- and full-custom solutions. If MOM performs an algorithm which has a two-dimensional *Map Oriented* (MO) organization, such as e.g. in image processing, it is an efficient alternative to conventional software solutions.

ARCHITECTURE AND MODE OF OPERATION

The basic architecture of the MOM is shown in Figure 2. 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, forms the 'data sequencer'. Also more than one cache may be used, which can be very useful for certain applications (see next section). The POLU is the problem-specific section, whereas all other modules are parts of the universal section of MOM.

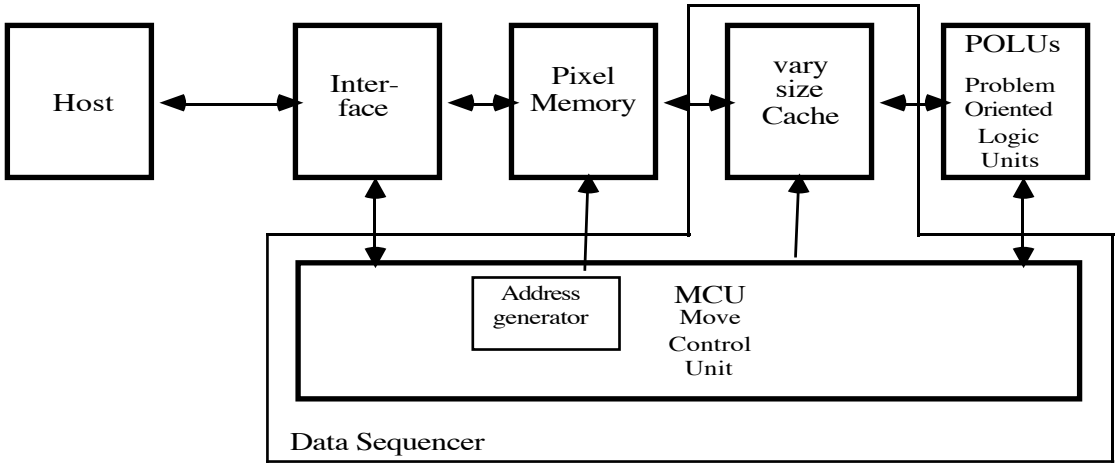

Figure 2: MOM Architecture

The Universal Part

The data sequencer in the MOM consists of the vary-size cache and the move control unit (MCU, see Figure 2). Its design is straightforward, further details are described in Hirschbiel 1985. The cache is a parallel accessible data memory. It holds a copy of a few pixels located next to each other and addressed by the MCU in the pixel memory. Partly it can be compared with the register file of a von Neumann style computer. The address itself 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, called a 'pixel'. Since the memory is partitioned into single-bit memory planes, a pixel is a column in these planes. We call this multiple bit map memory 'pixel memory'. To move the cache over the pixel memory simply means to 'look' through a movable peep-hole at the entire part of the pixel memory, though the cache actually buffers a copy of the 'visible' pixels. To move the cache there are two different move primitives: *jump absolute (x,y)* and *jump relative (x,y)*. Sequences of these primitives provide all possible cache move strategies necessary for the various applications of MOM.

The size of the memory is stored during an initialization period, keeping the minima and maxima of the x and y direction, to recognize the end of a memory line and the end of the memory itself. To meet all the contents in the memory exactly once an automatic videoscan mode is implemented (Figure 3). The cache is variable in its size (Figure 4) to be adaptable to different applications. It is connected to the problem oriented logic unit (POLU) which observes the cache's contents and delivers an application specific result depending on the matched reference data. So a read/modify/write cycle is organized for the cooperation between POLU and cache on one side, and the pixel memory on the other side.

Not only two-dimensional memory organizations may be used, also other multi-dimensional organizations can be taken into consideration. The difference to the von Neumann computer is that move primitives are used at data side, and not at the program side, furthermore 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. videoscan, and data-dependent moving such as e.g. in curve following. In the first strategy the move scheme is initialized by the host, in the latter strategy the POLU output is used to control the MCU and provide the move commands to compute the next memory address.

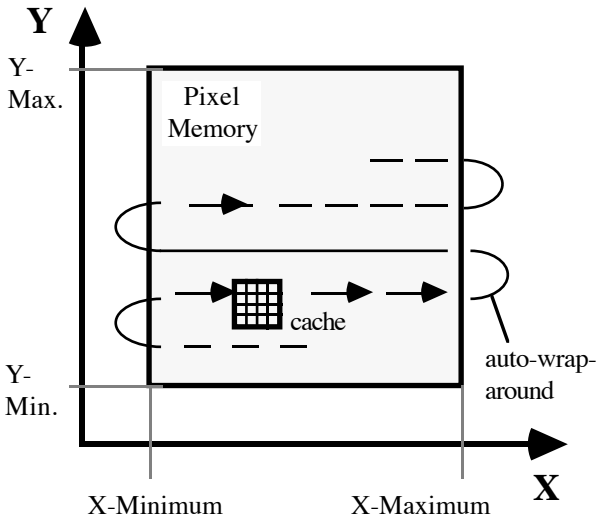

Figure 3: Video Scan

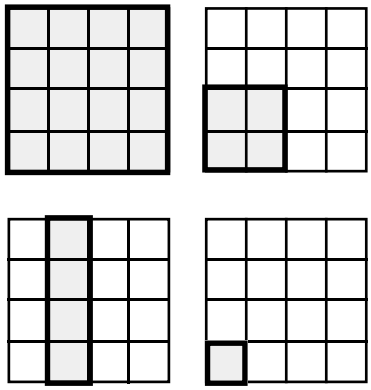

Figure 4: Vary Size Cache

The "Programmable" Part

The programmable part of the MOM consists of several problem-oriented logic units (POLUs). A POLU does not hold a sequential program, but is a CAD-generated special purpose hardware, which is obviously faster than a sequential program. It is subdivided into the combinationally stored reference pattern set and an interpreter which derives instructions from the matched patterns. It also contains 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 preprocesses such as shrink, expand and set operations

By means of a select code one of these POLUs can be selected (Figure 5), each performing a different application by comparing the cache's contents parallely with a set of reference patterns stored in a combinational hardware. The result is a list of numbers of the matching patterns. This list is interpreted, and several instructions are given depending on these references. 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.
- the next cache size
- the memory size

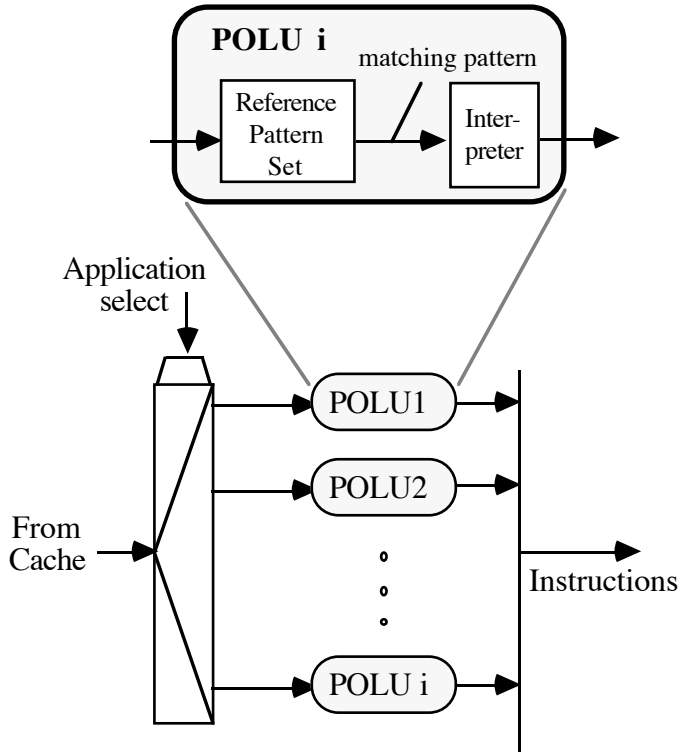

Figure 5: Problem Oriented Logic Units

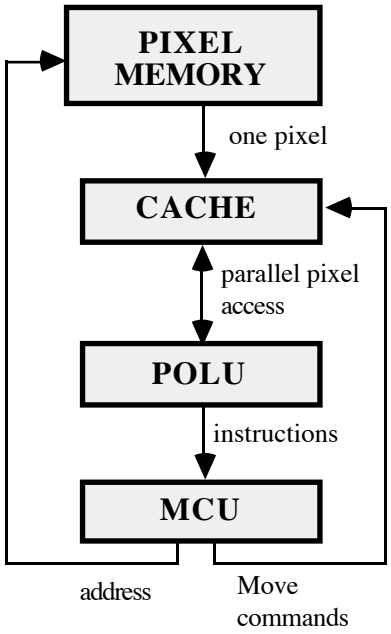

Figure 6: MOM Cycle

Therefore the 'program' for the MOM is stored in the reference pattern set which is part to be programmed for a new application (see section 'generation of new POLUs'). This 'program' can be stored in ROMs, PLAs, PALs, gate arrays, other semi custom or full custom devices or in mixtures of all these devices. For fast turn-around prototyping EPROMs, PROMs, RAMs, etc. may be used. Each application is performed in several cycles. One cycle consists of (compare Figure 6):

- load the cache
- compare the cache with the reference patterns in the selected POLU
- store result pattern in the cache
- select an application
- set the cache size
- set the memory area size
- move the cache (= store cache to memory and load it with the new addressed data)

There are 'no operations' for each cycle part so not all of the above operations have to be performed at every cycle. For example different operations can be performed at the same position in the memory (without cache movement) or the cache is moved and its contents remains unchanged. Before the program can actually start the host must set up the MOM with initial values. These are:

- preset the 4 limit registers for the memory size

- load the bit map memory with data
- select an initial cache size
- select an initial application
- move the cache to an initial position

GENERATION OF NEW POLUS

To program the MOM a new set of reference patterns and a new interpreter have to be constructed. A CAD-toolbox supports this. A comfortable editor for the reference patterns and the information necessary to interpret this is available. A translator generates the program code to store this reference data 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 to be able to apply mainly local operations. Then these local operations have to be defined and implemented to be performed on this data. This will be clarified with some examples in the next section. Local data can be accessed parallely that means it can be observed, compared and changed very quickly (in the demo set up of the MOM one of these cycles takes just one microsecond). These local data manipulations can be spent in large numbers compared to a von Neumann computer, where every comparison of data and access to data costs a lot. Sometimes it can be necessary to keep the same data in a few copies in the memory to be able to use just local operations, what can be accepted by the benefit in speed. After the algorithm is specified considering these special conditions the above CAD-tools help to install the algorithm on the MOM.

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. Classes of problems with their algorithms showing the power of MOM are described.

Image Preprocessing

Since the pixel memory is a multilayer bit map, it obviously is an excellent representation medium for two-dimensional images. This fact opens the wide range of image preprocesses for MOM execution. The simplest problems in this area are bit operations (boolean operations applied to 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 videoscan, i.e. the cache is moved from the left bottom corner, row by row, to the right top corner of the pixel memory. To provide the required operation 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 7 gives an example of the set operation 'intersection'.

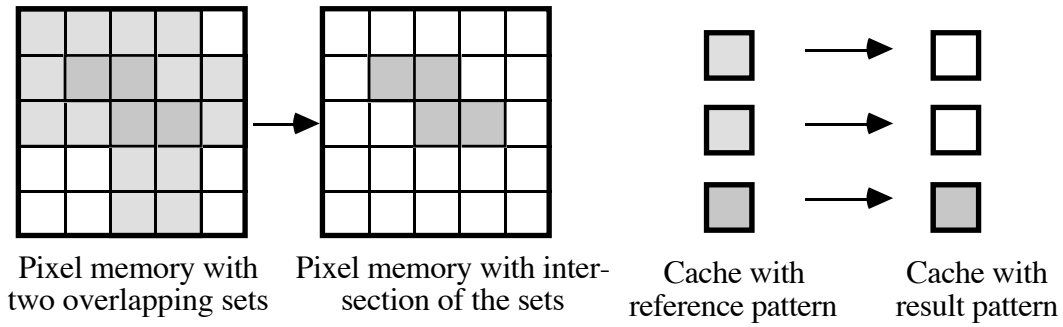

Figure 7: Set operation 'intersection'

These bit and set operations can be combined to perform functions on colour graphics data (colours stored in different layers), e.g. to mix colours or to make colours invisible. They may also be used to implement topological algorithms for image analysis.

The next slightly more complex level in image preprocessing needs the evaluation of neighbourhood information. To meet this requirement a larger cache is used. In most cases a 3-by-3 pixels wide cache is sufficient to get the necessary local informations. Usually the eight neighbours of the center pixel determine the change of the cache's contents. Shrinking and expansion of image structures are examples for this type of applications. To expand an object it is necessary to know whether a neighbour of the center pixel is already blocked or whether it is still free to expand the object (see Figure 8). The two reference patterns and their result patterns shown in Figure 8 are the first two encountered when performing a videoscan from the left bottom to the right top of the pixel memory section in the example of Figure 8. Naturally there are more different patterns necessary to provide a complete expansion as shown in the example.

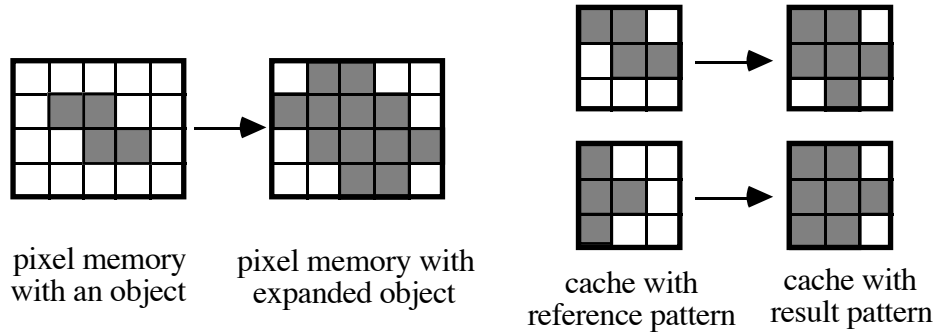

Figure 8: Expansion of an object

The expansion algorithm may also be implemented using a varied move strategy. The cache may follow the outline of the object and add new pixels at the edge of the existing object. In this case the moves of the cache are dependent of the matching reference patterns. Now the POLU provides the MCU with the new move commands. This explains that problems can be realized in different MOM algorithms, either in different reference patterns, or in different move strategies, or in both. The user has to decide whether to have a more complex reference pattern set, usually resulting in less execution time, or whether to have a simple move strategy,

resulting in an easier programming of the reference patterns.

Layout Processing

Another suitable field for MOM executions is VLSI layout processing, which can be regarded as image processing too. 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.

To perform 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 (Hartenstein et al 1984). The entire design rule check is done by a single videoscan over the layout. The reference patterns represent all possible design rule violations. A match of one or more of these patterns indicates a violation. For all reference patterns there is only one result pattern directly marking the location of a design rule violation in using a special error layer at the position where a reference pattern has matched. Figure 9 shows an example of a design rule violation and its detection by a reference pattern. The number of possible design rule violations is rather limited compared to the number of designs possible in an area as large as the cache. A Mead-&-Conway (1980) design rule check for example requires only 258 reference patterns and takes 1 second for a layout size of 1,000-by-1,000 lambdas. Clearly only orthogonal layout structures can be processed, because the underlying pixel memory allows only orthogonal structures to be stored.

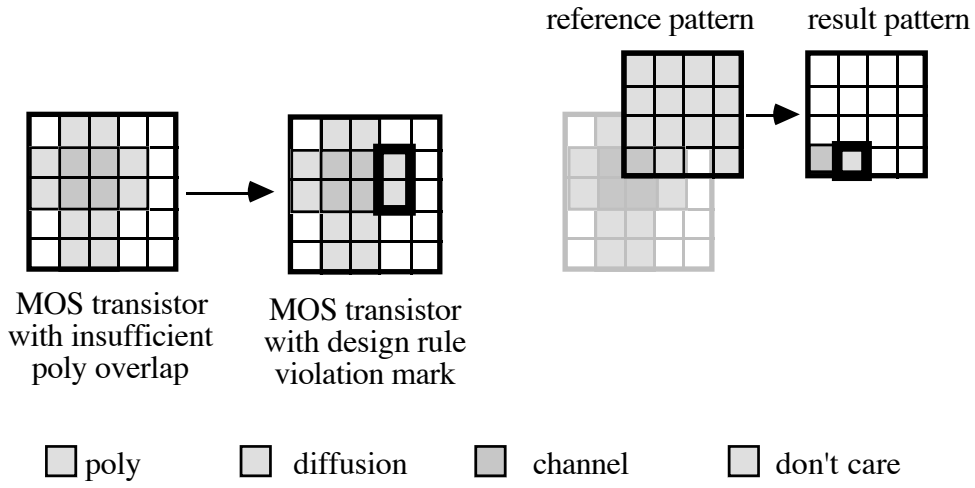

Figure 9: Design rule check example

Minimum-cost Path

To find a minimum-cost path using the Lee routing algorithm (Lee 1961, Breuer and Shamsa 1981) is another MOM application example in CAD for VLSI. Starting from a specially marked pixel (the start cell), the minimum-cost 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. First a single pixel cache is videoscanned 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 spirally outward from the start cell, expanding the wave continuously, until a certain matched reference pattern indicates that the target cell is found. In the third part of the routing algorithm a single pixel cache follows the direction of the arrows back from the target to the start cell to complete the routing process.

Aping Matrix-Oriented Logic

Matrix-oriented Logic (MOL, see Hartenstein et al 1986) are logic circuits which may be specified by means of a personality matrix, e.g. PLA, Weinberger array (Weinberger 1967), dense gate matrix (Lopez and Law 1980) 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. The reference patterns ape the logic function implied by the personality matrix by scanning over this matrix.

Non-CAD Applications

Driven by our own processing needs we mainly used the MOM for accelerator applications in CAD for VLSI. However, there are many other feasible applications of the MOM other than in VLSI design or in image processing. Especially, when the MOM is working in twin-cache mode (or sometimes in another multiple-cache mode) all types of convolution-like processes 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.

Aping Systolic Arrays. Systolic arrays are widely used to speed up algorithms by using hardware (Kung 1979, Foster and Kung 1980, Kung 1982). 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. Thus data is pulsed through these PE's and a pipelined processing is achieved, such that all PE's compute their current data parallely. The degree of parallelization is equivalent to the number of processing elements used. It is possible to ape these systolic arrays with the MOM. 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 one PE of the systolic array does. Figure 10 illustrates the transfer of the problem from systolic arrays onto a MOM.

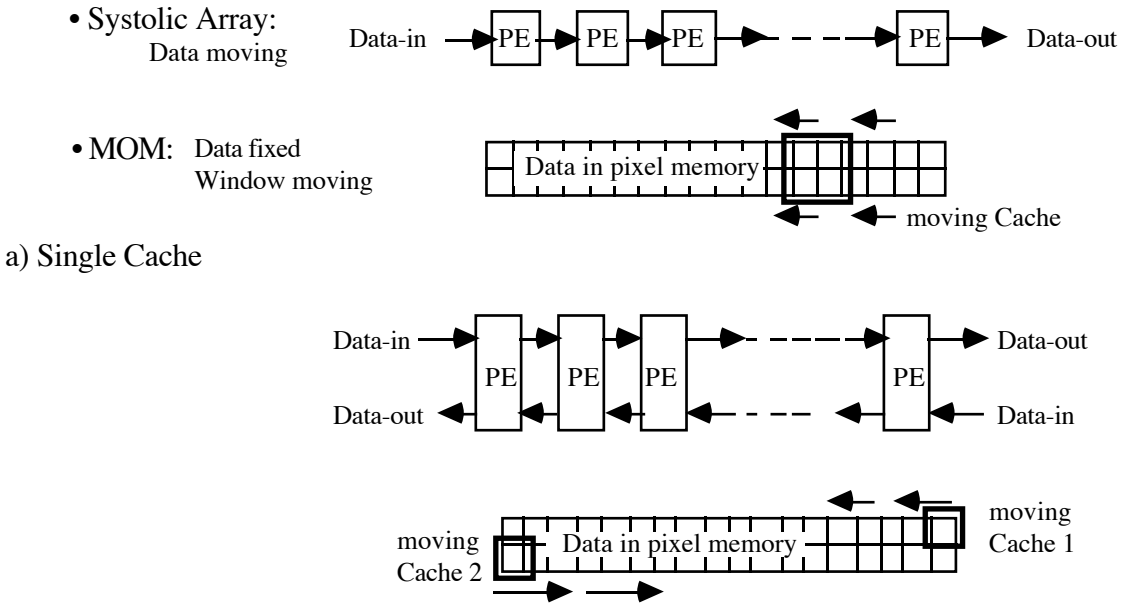

Figure 10: Aping of systolic arrays

MOM can 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 personalized in another way. In the latter application MOM would be used as a CAD tool (like a programmable 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.

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 (see also Figure 10).

A main topic of our present work is to design a high level language as an aid to program MOM easily, safely and quickly. This language includes 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 algorithms are transformed into new POLUs to extend MOM. This language is not only used to install desired new POLUs in the MOM, it also serves as a user's programming language to run and control the MOM using the existing POLUs.

Currently the MOM prototype based on standard TTL-circuits is reassembled using self-de-

signed full custom NMOS circuits to get a higher degree in integration and in speed. E.g. an extendible 4-by-4 pixel cache on a single chip has been designed and manufactured to substitute a whole board of TTL-circuits. (This work was funded by the German Ministry of Research and Technology within the E.I.S.-project.)

CONCLUSIONS

With the *Map Oriented Machine* 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 (Hartenstein et al 1984). 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. Nevertheless a lot of work has to be done to explore application methodologies of this new type of computational resource.

REFERENCES

Blank T 1984 A Survey of Hardware Accelerators used in Computer-Aided Design, IEEE Design and Test of Computers, August 1984

Breuer M and Shamsa K 1981 A Hardware Router, Journal Of Digital Systems, Vol. 4

Foster M J and Kung H T 1980, The Design of Special-Purpose VLSI Chips, Computer, Jan. 1980

Hartenstein R, Hauck R, Gebhardt J and Oelcke D 1986 Functional Extraction from Personality Matrices of MOL (Matrix Oriented Logic) Circuits, report, Kaiserslautern University

Hartenstein R, Hauck R, Hirschbiel A, Nebel W and Weber M 1984 PISA, A CAD Package And Special Hardware For Pixel-Oriented Layout Analysis, ICCAD - 84, Digest Of Technical Papers, Santa Clara

Hirschbiel A 1985, PISA Maschine, Eine spezielle Hardware für pixel-orientierte Bildverarbeitung, report, Kaiserslautern University

Kung H T 1979 Let's Design Algorithms for VLSI, Caltech Conference on VLSI

Kung H T 1982 Why Systolic Architectures?, Computer

Lee C Y 1961 An Algorithm For Path Connections And Its Applications, IEEE Transactions on Electronic Computers, EC-10

Lopez A and Law H 1980 A Dense Gate Matrix Layout Method for MOS VLSI, IEEE Journal

of solid-state circuits, SC-15
Mead C and Conway L 1980 Introduction to VLSI Systems, Addison Wesley
Preston K and Uhr L 1982 Multicomputers and Image Processing, Academic Press
Snyder L, Jamieson L H, Gannon D B and Siegel H J 1985 Algorithmically Specialized Parallel Computers, Academic Press
Sternberg S R 1981 Parallel architectures for image processing, in "Real/Time Parallel Computers" (M. Onoe, K. Preston Jr., and A. Rosenfeld, eds.), Plenum, New York
Weinberger A 1967 Large Scale Integration of MOS Complex Logic: A Layout Method, IEEE Journal of Solid-State Circuits, SC-2
Young T and Fu K 1986 Handbook of Pattern Recognition and Image Processing, Academic Press