MOM - MAP ORIENTED MACHINE
AN INNOVATIVE COMPUTING ARCHITECTURE

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

University of Kaiserslautern
P.B. 3049
D- 6750 Kaiserslautern , FRG
phone: ..49 / 631 / 205-2689

ABSTRACT

In this paper we describe an innovative computing architecture, called *Map Oriented Machine* (MOM). Concering speed, cost and flexiblity today there mainly exist two extreme solution: the totally flexible, but slow von Neumann computer and the very fast, but expensive and inflexible fully parallelized solution directly implemented on customized silicon. With some applications running on the MOM we show that it not only fills this gap, but also is a very good instrument to implement algorithms which have a map-oriented organization such as image processing. 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.

INTRODUCTION

The traditional approach in computer architectures is the well-known concept based on John von Neumann's work. Also well-known is the bottleneck between the processor and the memory of these computers. All data is accessed sequentially and furthermore the program access and program execution is sequential. Despite these drawbacks, the programming of von Neumann computers is very easy and also highly flexible.

Many efforts were and are done to improve the performance of this architecture. Special coprocessors were implemented to speed up various tasks in the computer. There are numeric processors like floating point processors to accelerate parts of a program, or there are I/O-processors to perform direct memory access, or graphic processors to speed up the display of immense amount of graphical data. Huge primary memories are installed to minimize the access to secondary storage. Reduced instruction sets are used to have faster processing units

available.

Another approach is it to extend the basic von Neumann concept into a multicomputer concept. These systems parallelize the programs by using special arrangements (arrays, hypercubes, vectors, and others) of a few computers. The programming of these systems is very difficult and they also have expensive operating systems and a large administrative overhead.

Totally different to the approaches above (all based on the von Neumann concept) is the use of special processing elements implemented in a special hardware. A certain number of these uniform processing elements (PEs) is arranged in linear or two-dimensional arrays. All these PEs perform the same operations in parallel and are connected for intercommunication purposes. So the degree of parallelism is direct proportional to the amount of PEs. The data is pulsed through these PEs, if the implementation is based on a systolic array concept [Kung 1979], or the complete data is stored in a PE square array. All these special-purpose architectures are very fast, but also very inflexible. If the special hardware is not capable of a certain algorithm, high design cost and a long turn-around time is necessary to implement new processing elements and their arrangement.

So we have the classical von Neumann concept on the one side and the special-purpose hardware solutions on the other. Both have their advantages and disadvantages concerning speed, cost and flexibility. Between these two main approaches is a remarkable gap, which is filled by the new concept presented here. The *Map-Oriented Machine* (MOM) is a medium speed solution at low cost. The basic idea of accelerating algorithms is to parallelize the program access by combinational hardware, where the data access remains sequential. The data itself is stored in a two-dimensional map-oriented memory and is accessed via a special window cache. The system can be personalized in a highly mechanized way to be adapted to a surprisingly wide range of applications. A CAD environment to develop applications includes the programming language MOPL (Map Oriented Programming Language) which supports the 'programming' of the MOM.

THE ARCHITECTURE OF THE MAP-ORIENTED MACHINE

The Map-oriented Machine consists of four main parts (see figure 1), the map-oriented data memory, the data cache, the move control unit and the programmable part, the problem oriented logic units. All these parts are connected to each other via a VME-bus.

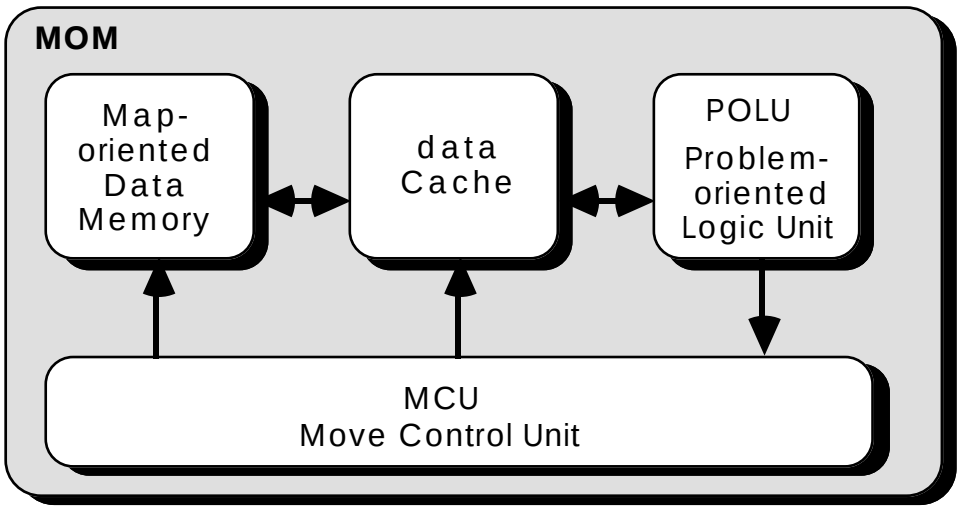

Figure 1: MOM Architecture

The map-oriented data memory has a two-dimensional organization and is addressable in x and in y direction. At each address a pixel is located (z direction), which has a variable size and a variable format (figure 2). The pixels can be used to store bit vectors, numeric values, they may have a tagged format or any mixed representation. In the whole memory all these types of pixel formats may be used at the same time to have many different types of variables and values available.

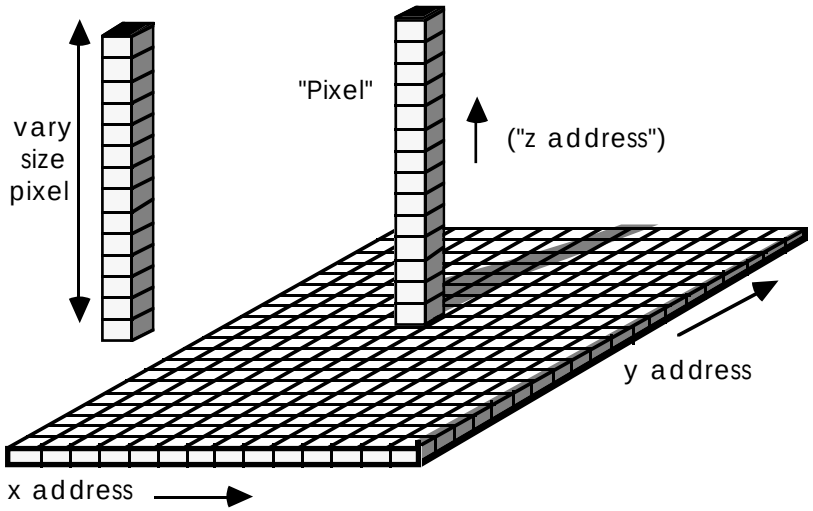

Figure 2: Map-oriented Data Memory

The data cache is a window to the map-oriented data memory located at the position which is addressed by the x and y address in the memory. The cache then holds a copy of a few pixels in the memory to make them accessable for the execution. All the pixels in the cache may be accessed in parallel and sent to the actual problem-oriented logic unit. The size of the cache is variable to adapt it to different applications (figure 3).

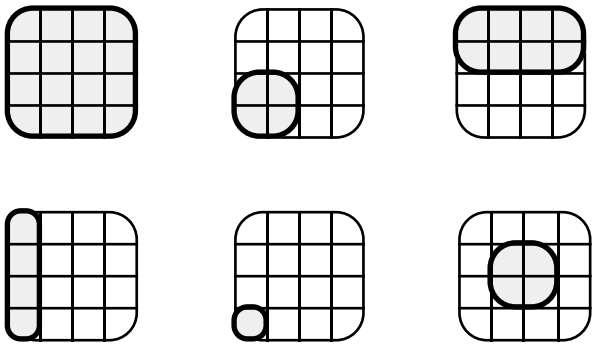

Figure 3: Window Cache Principle

In the move control unit (MCU) the addresses for the cache are generated to provide a controlled cache movement over the data in the memory. Two major move strategies are available. Schematic move strategies are independent of the data in the cache. There are single steps to neighbour positions possible, as well as gotos and relative jumps possible. A combination of these steps may result in linear step sequences, videoscan, shuffle sequences and other schematic move sequences. Non-schematic strategies are dependent on the data currently provided by the cache. These data driven movements are important to trace for example the contour of an object in an image. A, not complete, summary of possible movements is shown in figure 4.

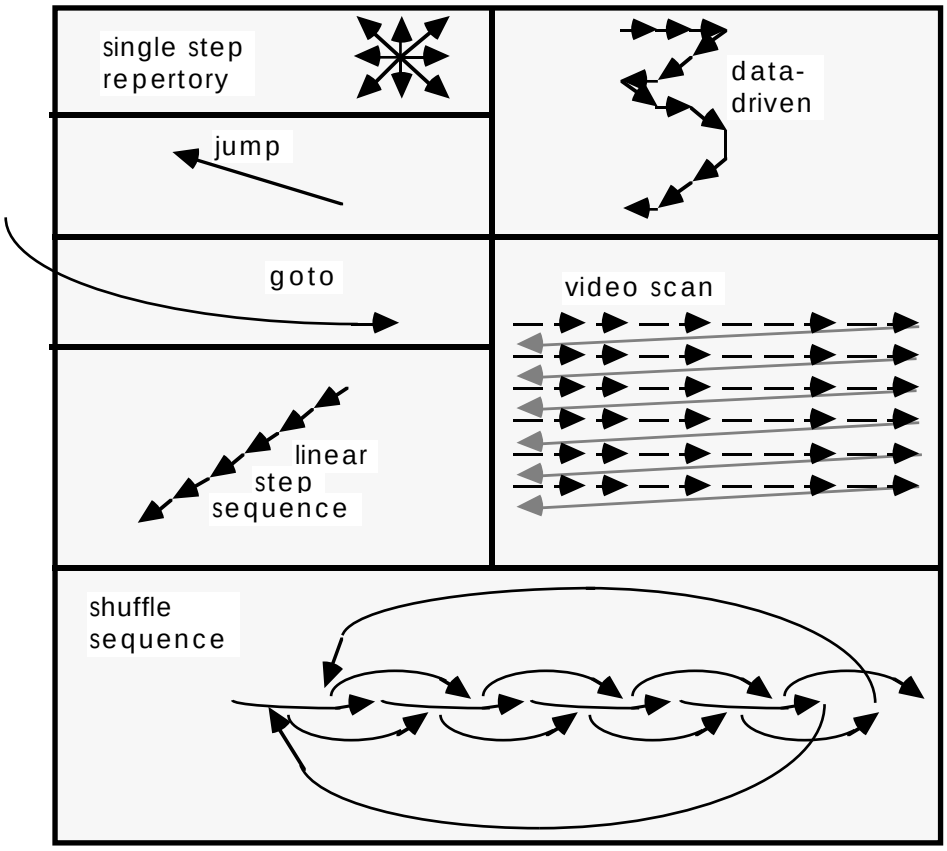

Figure 4: Cache Move Examples

All these movement strategies are supported by memory space limit registers to determine the boundary of the certain move area.

The programmable part of the MOM consists of several problem-oriented logic units (POLUs), one for each application. They can be selected by a special application select code (see figure 5). Such a POLU does not hold a sequential program, but is a CAD generated combinational hardware, which is obviously faster than a sequential program. The hardware contains a combinationally stored set of reference patterns, which is matched with the cache's contents in parallel. As a result of this pattern match a different pattern can be written back into the cache to cause the change of data in the cache and with this, in the memory. A second result, if the application requires data dependent cache movement, is the move code to be sent to the move control unit.

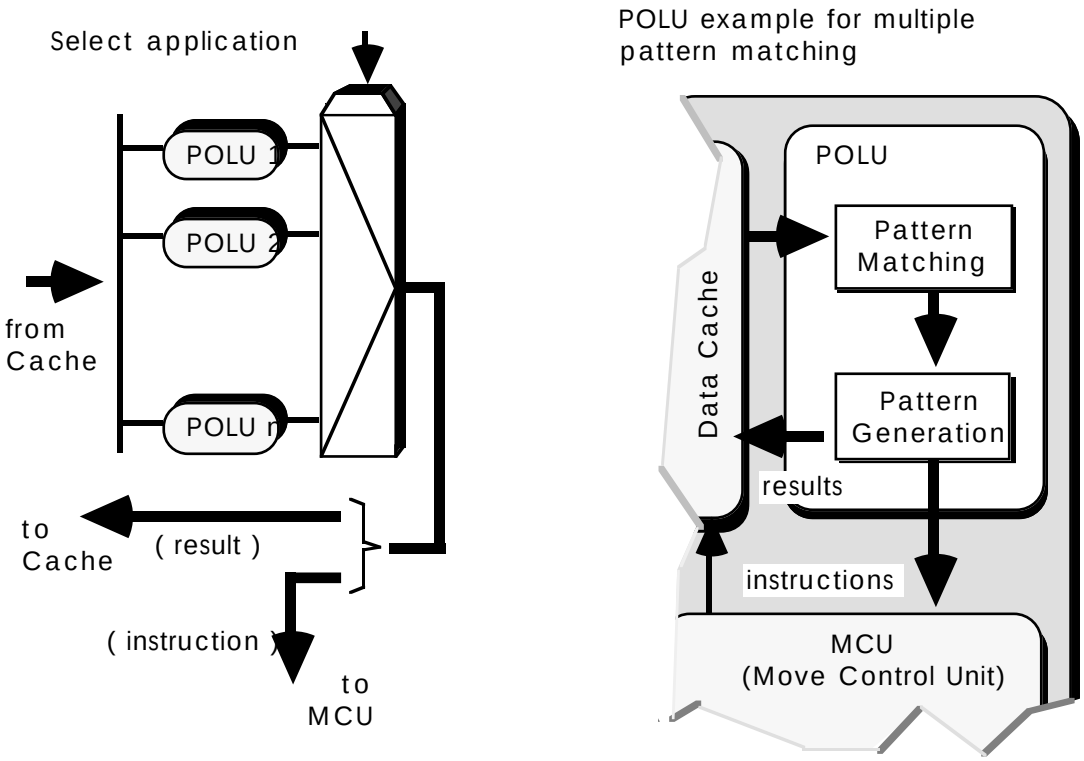

Figure 5: The Problem-Oriented Logic Units

To support numeric processing a traditional arithmetic logic unit (ALU) may be added as one of the POLUs. Then the cache is used as the register file for this ALU.
To implement new POLUs, that means to 'program' the MOM, a CAD toolbox is used. A comfortable editor to specify the reference patterns and the information necessary to interpret this is available. A translator generates the program code to store this reference data in EPROMs, PLAs, RAMs or ROMs. A high level language MOPL (Map Oriented Programming Language) may also be used to specify the patterns for new POLUs or to use existing POLUs as sort of standard functions and to control the movement of the cache.

The map-oriented machine is connected via a high speed bus to a workstation serving as a host. This bus (1 Mbyte per second) is used to load and unload the data memory, to have peripheral I/O equipment available and to provide the user interface to the MOM.

APPLICATION EXAMPLES

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 the 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 operation 'intersection'.

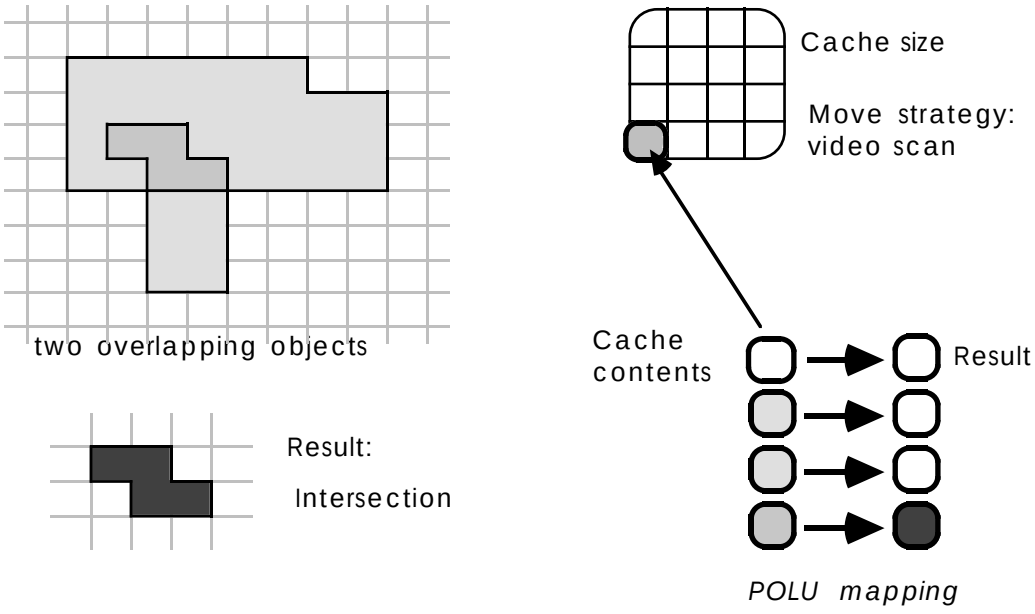

Figure 7: Set operation 'intersection'

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.

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.

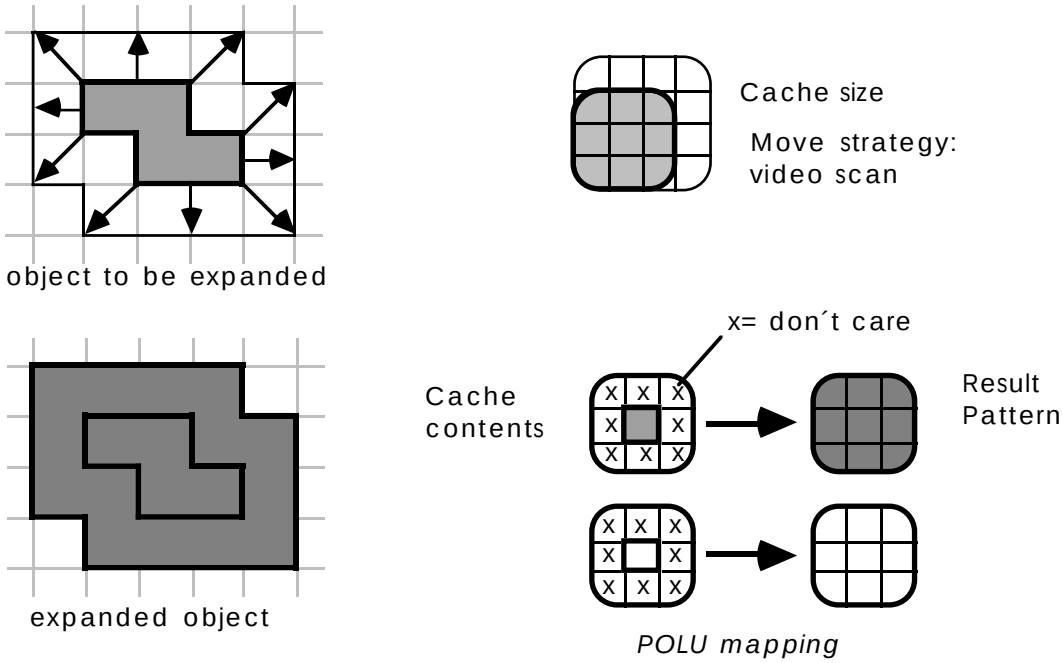

Figure 8: Expansion of an object

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 set of reference patterns, 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.

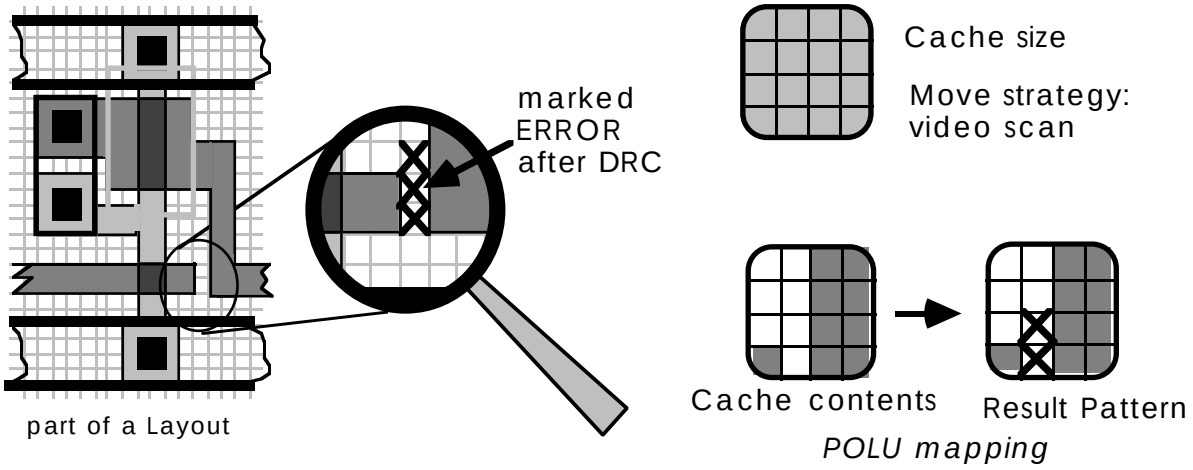

Figure 9: Design rule check example

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.

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.

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 the 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. Therefore 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 in parallel. 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.

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.

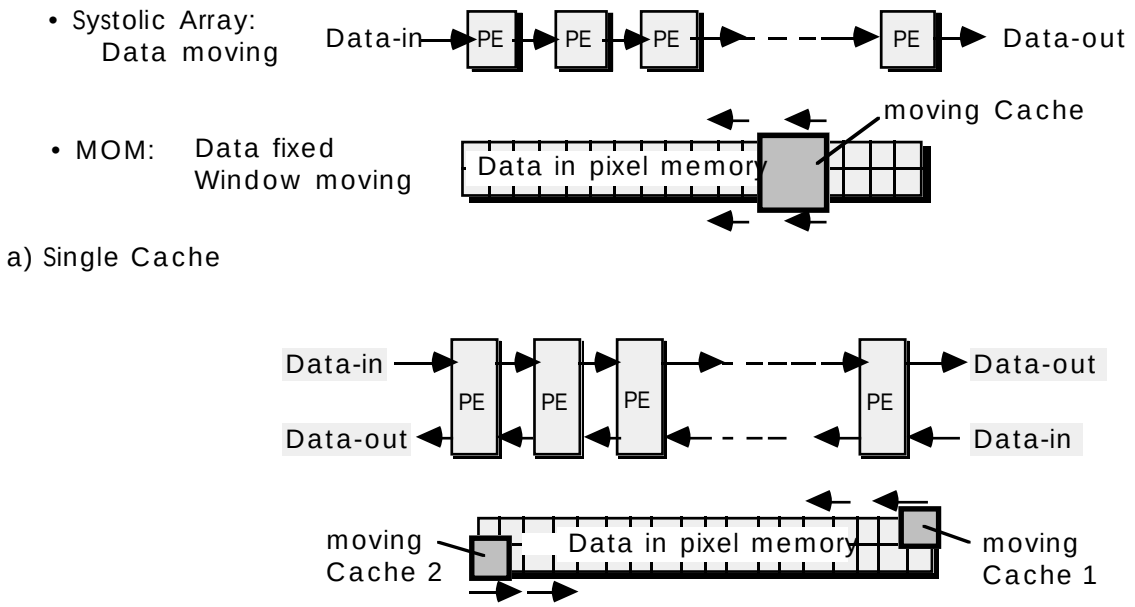

Figure 10: Aping of systolic arrays

CURRENT WORK

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 MOM programming language (MOPL) 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-designed 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 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 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 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, 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. 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