MOM - MAP ORIENTED MACHINE

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

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

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.

1. 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 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 flexible, 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 [12]. It is difficult to achieve the parallel operations the hardware resources seem to offer.

- Array Architectures using simple processing elements [1]
  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) [14] 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 [7], 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 repertory 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.

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. With the support of a CAD environment this hardware may be personalized in a highly mechanized way to be adapted to a surprisingly wide range of applications. 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.

2. ARCHITECTURE AND MODE OF OPERATION

The basic architecture of the MOM is shown in Figure 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, forms the 'data sequencer'. The POLU is the problem specific section, whereas all other modules are parts of the universal section of MOM.

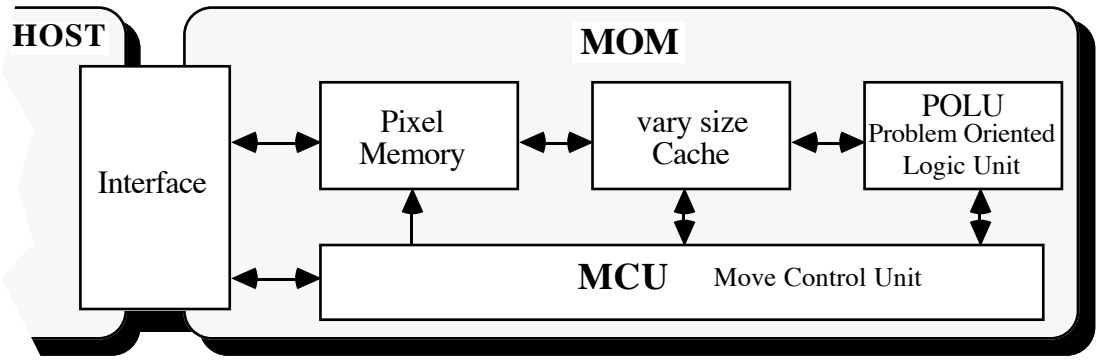

Figure 1: MOM Architecture

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 1). Its design is straightforward, further details are described in [6]. The cache is a parallel accessible data memory which by parameter control can be varied in its sized (Figure 3) to be adaptable to different applications. It is connected to the POLU which observes the cache's contents and delivers an application specific result depending on the matched reference data. It holds a copy of a few pixels located next to each other and addressed by the MCU in the pixel memory. 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. To move the cache there are different move primitives such as *single step, goto* and *jump*. Sequences of these primitives provide all possible cache move strategies necessary for the various applications of MOM. Two fundamentally different moving strategies may be applied: schematic moves such as e.g. videoscan to meet all pixels exactly once (Figure 2), 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. All

move sequences are supported by address limits to specify a particular memory space (Figure 2).

Since the *memory* is not used for program storage we are not forced to use a particular Instruction format. So a pixel in the memory can have various formats representing e.g. a bit vector, a numeric value, or could be divided into a input vector and an output vector. Some tags can be added to every pixel to influence the cache movement. Therefore the format and the size of the pixels is adaptable to the particular application.

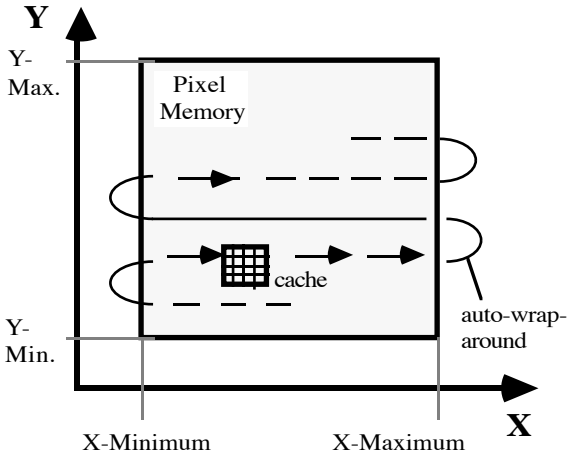

Figure 2: Video Scan

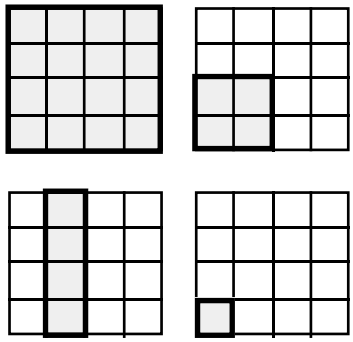

Figure 3: Vary Size Cache

2.2. 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 set of reference patterns 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*, or, for numeric processing, it could contain a standard ALU.

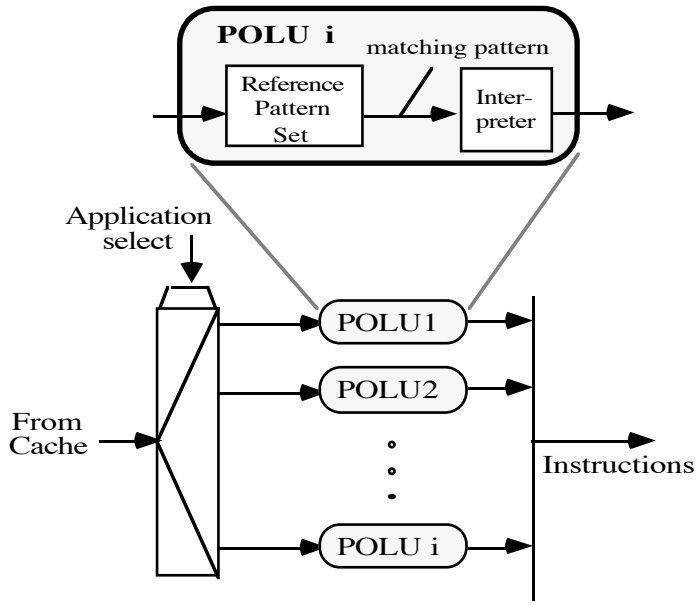

Figure 4: Problem Oriented Logic Units

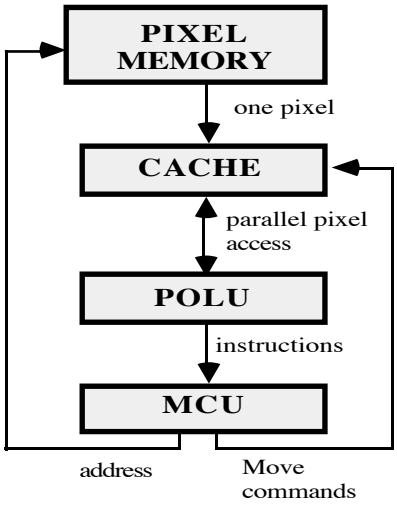

Figure 5: MOM Cycle

By means of a select code one of these POLUs can be selected (Figure 4), each performing a different application by comparing the cache's contents in parallel with a set of reference patterns. The resulting list of matching patterns is interpreted, and several instructions and result patterns are generated. Therefore the 'program' for the MOM is stored in the reference patterns which are the 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. Each application is performed in several cycles (Figure 5). In each cycle the cache is loaded with new data which will be compared with the reference patterns. The result pattern is generated, and the cache and memory size is selected. Finally the cache is move to the next position.

3. GENERATION OF NEW POLUS

To program the MOM a new set of reference patterns and a new interpreter have to be constructed. For this purpose we use a high level language as an aid to program MOM . This language, called MOPL (Map Oriented Programming 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. A translator generates the program code to store this reference data in RAMs, ROMs, PLAs, etc.. 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.

4. 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 MOM are described.

4.1. Image Preprocessing

Since the memory is pixel oriented, image preprocessing is a natural application for MOM execution. The simplest problems in this area are bit operations and operations on sets of pixels. Boolean operations 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 move strategy is to visit each pixel exactly once by performing a videoscan. Figure 6 gives an example of the set operation *intersection*. These operations can be combined to perform functions on colour graphics data, e.g. to mix colours or to make colours invisible.

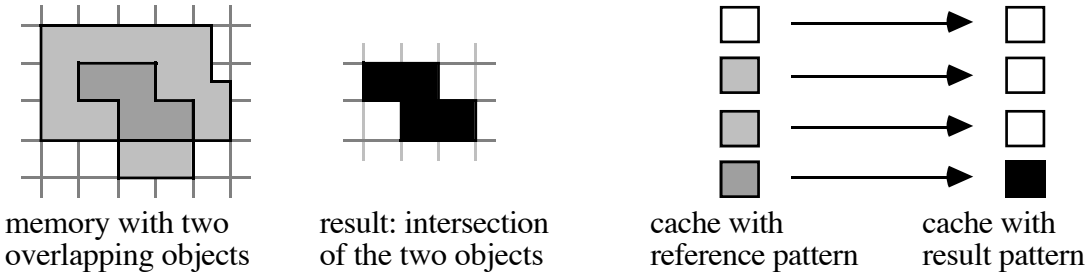

Figure 6: Set operation 'intersection'

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.

4.2. Layout Processing

Another suitable field for MOM executions is VLSI layout processing, such as e.g. design rule check or circuit extraction. An 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 7 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 [11] design rule check requires 258 reference patterns and takes about 30 seconds for chip area of $1cm^2$.

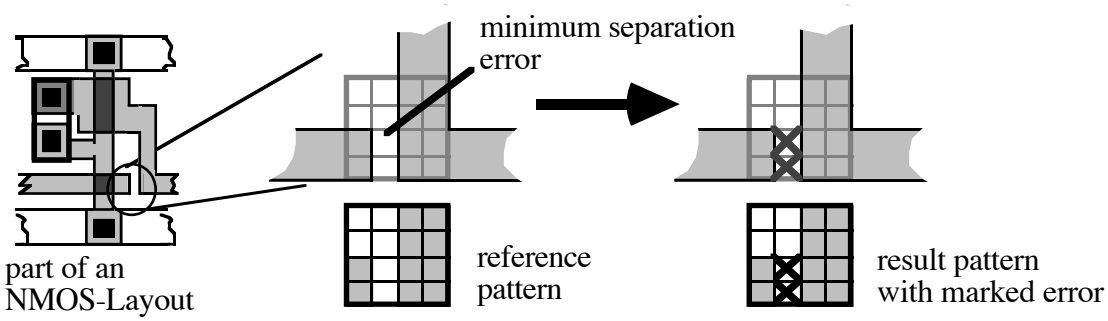

Figure 7: Design rule check example

4.3. Other Applications

To find a *minimum-cost path* using the Lee routing algorithm [2,9] is another MOM application example in CAD for VLSI. Performing a spirally movement, a 3-by-3 pixel cache propagates an arrow wavefront around a start cell, until the target cell is found. Then a single pixel cache follows the direction of the arrows back to the start cell marking the minimum-cost path.

Matrix-oriented Logic [4] are logic circuits which may be specified by means of a personality matrix, e.g. PLA, Weinberger array [15], dense gate matrix [10] and others. MOM can be used to simulate and verify this symbolic description given by the personality matrix.

Systolic arrays are widely used to speed up algorithms by using hardware [3,7,8]. 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. So, as a matter of fact, everything which fits onto a systolic array implementation can be directly mapped onto the MOM system.

There are many other feasible applications of the MOM, especially when it 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.

5. CURRENT WORK

A main topic of our present work is to design the high level language MOPL (Map Oriented Programming Language) as an aid to program MOM easily, safely and quickly. PADL (Pattern Definition Language) which is a subset of MOPL provides a efficient way to describe patterns. 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. To cover a wider area of data processing problems the MOM can be extended to have two or more caches running simultaneously.

6. CONCLUSIONS

We have illustrated a flexible computing resource and accelerator environment concept for a wide variety of applications. 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 [5]. 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 (approx. $3mm^2$) with Mead-&-Conway design rules takes 1 second, compared to minutes or hours in using super mini computers. MOM may be used for CAD applications, such as e.g. in VLSI design and verification, or for real data processing in a wide variety of applications.

REFERENCES

[1] T. Blank, A Survey of Hardware Accelerators used in Computer-Aided Design, IEEE Design and Test of Computers, August 1984.
[2] M.Breuer and K.Shamsa, A Hardware Router, Journal Of Digital Systems, Vol.4, 1981
[3] M.Foster, H.Kung, The Design of Special-Purpose VLSI Chips, Computer, Jan. 1980.
[4] R.Hartenstein, R.Hauck , Gebhardt, Oelcke, Functional Extraction from Personality Matrices of MOL (Matrix Oriented Logic) Circuits, report, Kaiserslautern Univ.1986.
[5] R.Hartenstein, R.Hauck, A.Hirschbiel, W.Nebel, M.Weber, PISA, A CAD Package and Special Hardware for Pixel-Oriented Layout Analysis, ICCAD 1984, Santa Clara.
[6] A. Hirschbiel, PISA Maschine, Eine spezielle Hardware für pixel-orientierte Bildverarbeitung, report, Kaiserslautern University, 1985.
[7] H.Kung, Let's Design Algorithms for VLSI, Caltech Conference on VLSI, 1979.
[8] H.Kung, Why Systolic Architectures?, Computer, 1982
[9] C.Y.Lee, An Algorithm For Path Connections And Its Applications, IEEE Transactions on Electronic Computers, EC-10, 1961.
[10] A. Lopez, H.Law, A Dense Gate Matrix Layout Method for MOS VLSI, IEEE Journal of solid-state circuits, SC-15, 1980.
[11] C. Mead, L.Conway, Introduction to VLSI Systems, Addison Wesley, 1980.
[12] K.Preston, L.Uhr, Multicomputers and Image Processing, Academic Press, 1982.
[13] L.Snyder, L.H.Jamieson, D.B.Gannon, H.J.Siegel, Algorithmically Specialized Parallel Computers, Academic Press,1985.

[14]S.R.Sternberg, Parallel architectures for image processing, in "Real/Time Parallel Computers" (M.Onoe, K.Preston Jr., A.Rosenfeld, eds.), Plenum, New York, 1981.
[15]A.Weinberger, Large Scale Integration of MOS Complex Logic: A Layout Method, IEEE Journal of Solid-State Circuits, SC-2, 1967.