MOM - Map Oriented Machine

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

Kaiserslautern University
Computer Science Department
Postfach 3049
D-6750 Kaiserslautern, F.R.G.
phone: (..49 631) 205-2689

ABSTRACT
In this paper we describe an 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 organisation, 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.

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 an very inflexible (see [SJGS85], [BLA84] and Figure 1.1).

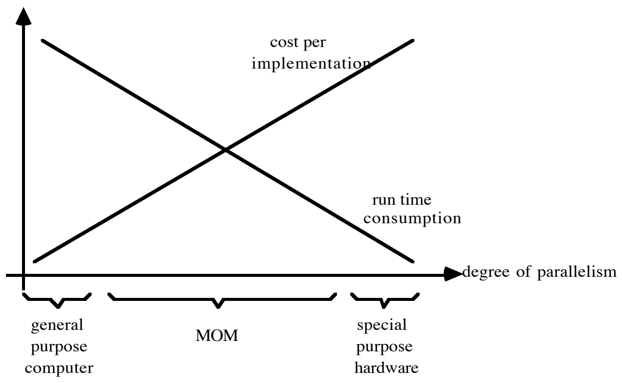


Figure 1.1: General purpose versus special purpose machines

There are mainly two approaches to solve data processing problems.

• 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 ([PU82], [YOFU86]). 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) [STER81] 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 [KUN79], 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 accelaration factor from the fully parallel hardware solution to the software solution exeeds 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) organisation, 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 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'. Also more than one cache may be used, which could 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.

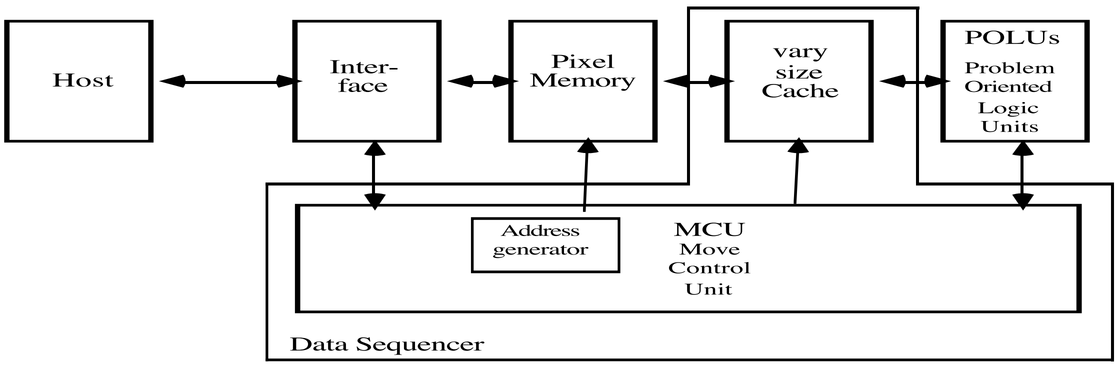


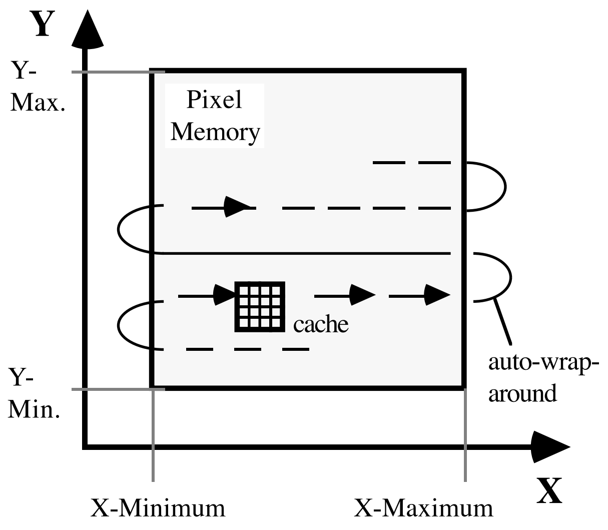
Figure 2.1: MOM Architecture

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). Its design is straightforward, further details are described in [HIR85]. 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 could 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'. The memory is partitioned into single-bit memory planes. Thus the memory is a multiple bit map memory, called '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).*

The size of the memory is stored during the initialization, keeping the minima and maxima for the x and y direction, to recognize the end of a memory line and the end of memory itself. To meet all the contents in the memory an automatic videoscan mode is implemented (Figure 2.2). The cache is variable in its size (Figure 2.3) to be adaptable to different applications. It is connected to the problem oriented logic unit (PO-LU) which observes the cache's contents and delivers an application specific result. 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 organisations may be used, also other multi-dimensional organisations 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. video scan, and data-dependant moving such as e.g. in curve following. For the latter strategy the POLU output is used to control the MCU.



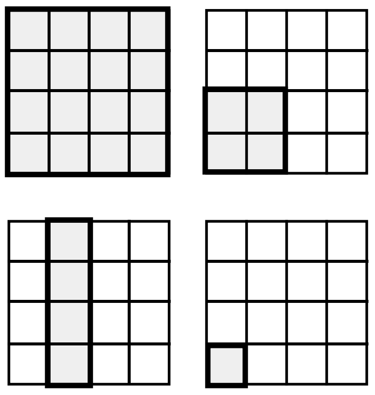


Figure 2.2: Video Scan Figure 2.3: Vary Size Cache

2.2. The "programmable" Part

The programmable part of the MOM is called 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.

The 'program' for the MOM therefore is stored in the reference pattern set which is part 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 devices. For fast turn-around prototyping EPROM, PROM, RAM, 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)

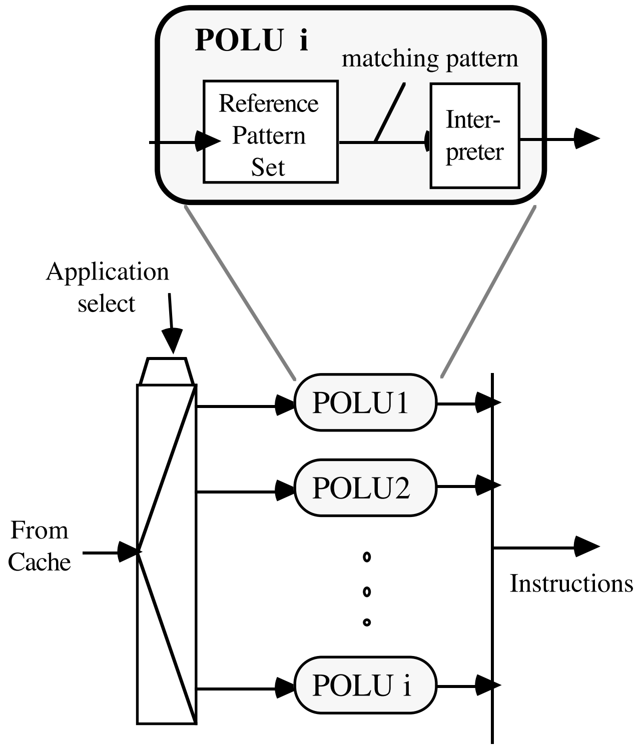


Figure 2.4: Problem Oriented Logic Units

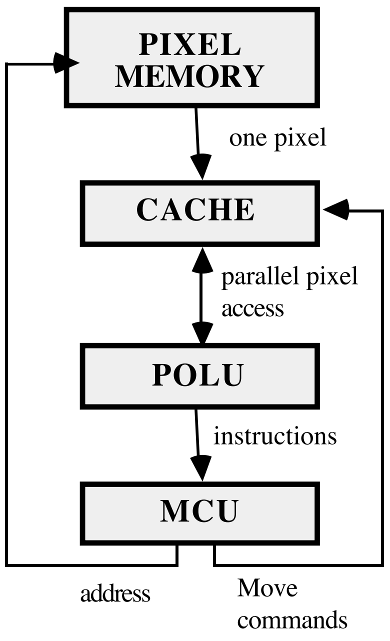


Figure 2.5: MOM Cycle

There are 'no operations' for each cycle part so not all of the above operations are performed at every cycle. For example different operations can be performed at the same position in the memory (without 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 size
- load the bit map memory with data
- select an initial cache size
- select an initial application

- move the cache to an 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. 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 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 parallelly that means it can be observed, compared and changed very quickly. These local data manipulations can be spent in large numbers in opposite to a von Neumann computer, where every comparison of data and access to data costs a lot. Sometimes it can be necessary to hold 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 MOM.

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.

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 perform functions on colours stored in different layers, e.g. to mix colours 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 the necessary 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 shown in Figure 4.2 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

A variation of the expansion algorithm could be to perform not a videoscan move strategy, but to 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. This explains that problems can be realized in different MOM algorithms, either in different reference patterns or in different move strategies or in both.

4.3. Layout Processing

Another suitable field for MOM executions is VLSI layout processing, which can be regarded as images 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. Classical problems feasible for MOM are design rule check, Lee routing or circuit extraction.

4.3.1. Design Rule Check Application of MOM

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 [HNW84]. 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 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. The Mead-&-Conway design rules [MECO80] require only 258 patterns 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], [BS81]) 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 [VEL87]. 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 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. 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 4.4: Lee routing example (snapshots)

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.

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], [KUN82]. 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 parallel. 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 the PE of the systolic array does. Figure 4.5 illustrates the transfer of the problem from systolic arrays onto a MOM.

Figure 4.5: 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.

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. Especially those applications which are based on repetitive computations on large sets of data could be executed by the MOM extension.

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 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 described algorithms is transformed into new POLUs to extend MOM.

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. Nevertheless 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, Jan. 1980

[GHHO86] R. Gebhardt, R. Hartenstein, R. Hauck, D. Oelcke: *Functional Extraction from Personality Matrices 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.

[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

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

[VEL87] I. Velten: *Implementierung des Lee-Algorithmus auf MOM* (in german), report, Kaiserslautern University, 1987

[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

[YOFU86] T. Young, K. Fu: *Handbook of Pattern Recognition and Image Processing*, Academic Press, 1986