MAPPING SYSTOLIC ARRAYS ONTO THE MAP-ORIENTED MACHINE (MoM)

Reiner W. Hartenstein, Alexander G. Hirschbiel, Michael Weber
Universität Kaiserslautern, D-675 Kaiserslautern, F.R.G.

ABSTRACT

A method SYS^{2}toMoM to map systolic systems onto the MoM (**m**ap-**o**riented **m**achine) is introduced in this paper. This mapping method is needed to derive a methodology of MoM application development support from the theory of systolic array synthesis. The MoM is a flexible non-von-Neumann computer principle having been developed at Kaiserslautern. MoM "programming" uses combinational code (for path programming) instead of sequential code (for sequencing). That's why for a wide variety of computation problems the MoM provides substantial acceleration factors over von Neumann machines. The MoM can also be used as an inexpensive and highly flexible programmable pseudo-systolic processor for emulation of systolic arrays, such as e.g. in experimenting with alternative systolic architectures at very early phases of the systolic array design process.

INTRODUCTION

This paper introduces a methodology of application development support for using the MoM [13], a monoprocessor based on an innovative non-von-Neumann computing principle. It explains SYS^{2}toMoM, its essential part to remap data dependencies from systolic arrays to MoM use. This methodology is indirectly based on the systolic array concept [1, 2, 3]. It uses derivatives of systolic array synthesis methods [4, 5, 6, 7, 8, 9] and optimization methods [11, 12].

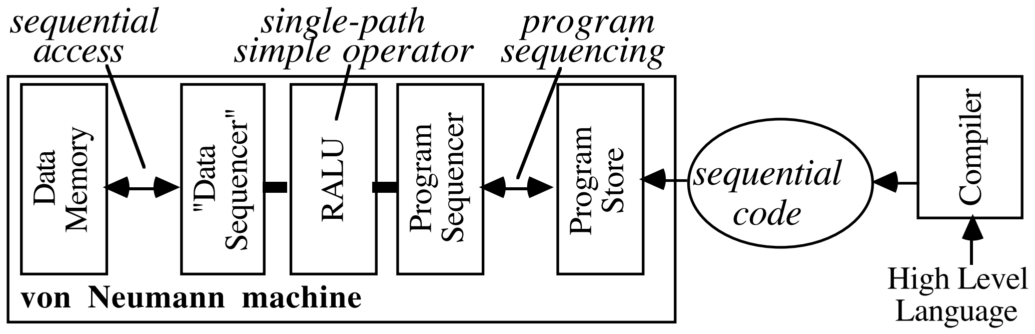


Fig. 1: von Neumann basic processor architecture (Harvard version)

First a brief introduction of the MoM and its architectural principles is given. Then MoM principles are compared with the von Neumann principle. The main section of this paper introduces the philosophy, how systolic synthesis methods are mapped onto the MoM principles to derive a MoM application development support methodology. Finally the MoM is compared with the WARP^{SM} [10] machine and with the use of systolic arrays.

WHY THE MoM IS FASTER THAN VON NEUMANN

The principles of the MoM may be illustrated by comparison with von Neumann principles. The von Neumann machine may be divided into five major parts: data memory, data sequencer, RALU (registers and ALU), program sequencer and program store (fig. 1). Well-known are the three von Neumann bottlenecks: both, data and program side are accessed sequentially. The RALU provides only a very low degree of parallelism: due to its single data path principles it is only able to perform a single simple operation at a time. To determine the next operations the program memory has to be scanned several times. That's why von Neumann hardware is very slow. Specialized accelerator machines have been developed for certain problem classes [19], what is expensive and offers only very little flexibility and limited acceleration. For higher acceleration factors (e.g. by millions as e.g. in [20]), a fully customized ASIC approach is needed, which is even more expensive, but has no flexibility at all.

Operation Example (512x512 sized memory)	MoM* msec	VAX 11/750 sec	VAX 11/750 acceleration factor	Motorola 68000 sec	Motorola 68000 acceleration factor
grey image operations binary image operations	60	7.8	>130	14.7	>240
erosion, dilation, skeleton, edge detection	210	38	>180	64	>300
design rule check (λ-based)	260	300	>1000	600**	>2000
minimum cost Path (Lee)	150	20	>130	40**	>160
10x10 matrix multiplication ☐ aping systolic array ☐ using 2 caches	 16 1	0.04	(>2) 40	0.15	(>9) 150

* conservative TTL demo set-up, which has not been tuned ** estimated

Table 1: Some acceleration factor examples

To fill the wide performance gap between the von Neumann approach and tailored ASIC solutions a *universal accelerator principle* is needed. Such an *almost universal* accelerator principle is used by the ***Map-oriented Machine*** (MoM) [21, 22]. Its three main acceleration ideas are based on minimizing "program" scanning:

(a) highly parallel path programmable operation unit (POLU)

(b) no program scan sequencing needed by the POLU

(c) powerful highly parametrized repertory of hardwired data scan sequences

(d) most branching is directly data driven

Replacing the single-simple-operation-at-a-time RALU by an electrically path- programmed fully combinational data manipulation hardware called ***POLU*** (a) avoids sequencing and thus program scanning between main memory data accesses (b). (POLU stands for "problem-oriented logic unit" using RAMs, PLDs [23] and other programmable hardware [24]). Also at Kaiserslautern an electrically programmable gate array has been designed in nMOS technology ([29], fig. 14), which has been fabricated and tested. That's why one of the three von Neumann bottlenecks definitely is removed: the program side bottle neck [14, 21, 22]. Only data access remains sequential, however with substantially reduced need for next address lookup from relatively slow standard memory (c). Most branching is data driven (d), such that no control instructions to be scanned are needed. So the need for explicit branching is drastically reduced. Such 'strategic branching' is carried out under host interaction.

For a surprisingly wide variety of application areas (wider than the area of systolic array application) the MoM offers substantial acceleration factors over the von Neumann machine, even up to several orders of magnitude. Table 1 shows some acceleration factor examples having been experienced at Kaiserslautern [15, 21,22].

ARCHITECTURAL PRINCIPLES OF THE MoM

The Map-oriented Machine consists of four major parts (figure 2), the map-oriented data memory, the data scan cache, the move control unit and the programmable part, the problem oriented logic units (POLUs). The POLU part is programmed by loading combinational code into it, being generated by a MoMpiler (see fig. 2). The Kaiserslautern MoM demo setup (fig. 15) is connected to a host and to an eltec image processing computer system [30] via a high speed bus.

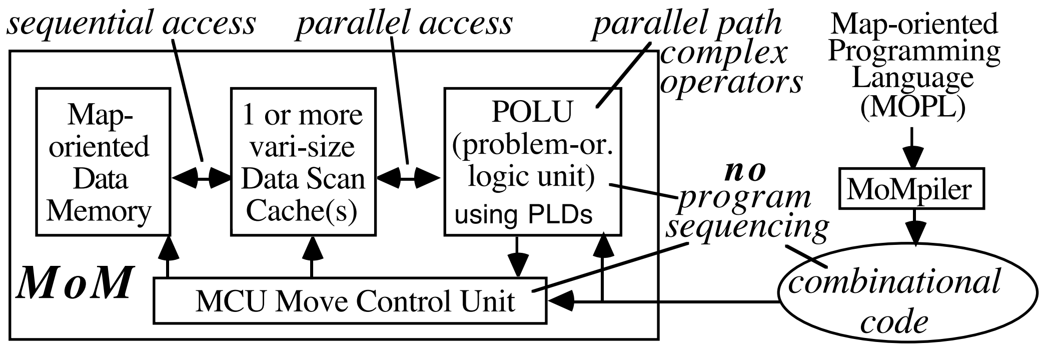


Fig. 2: Illustration of MoM Architecture Principles

The ***problem-oriented logic units*** (POLU, also see fig. 3) does not have a hard-wired instruction set. It only holds user-defined functions, 'combinationally' path-programmed into PLDs, RAMs or electrically programmable gate arrays [14]. Due to the capability to store many terms, the POLU operations can much more powerful than instructions of even complex instruction set von Neumann processors. For example, hundreds of "if-clauses" may be coded to be executed in parallel.

Let's explain this parallelism with a pattern matching application example (e.g. in image preprocessing): the POLU contains a set of reference patterns which are matched in the analyzer (fig. 3b) with the cache's contents in parallel within a single machine cycle. As an interpretation of this pattern match a different ("result", see fig. 3b) pattern can be written back into the cache to complete a read-modify-write operation for the modification of data in main memory via data scan cache. Numeric functions can be implemented as a set of quasi-reference patterns and result patterns derived from the corresponding function table. As a second result feedback to the move control unit may be generated (cache(s) move "instructions", see fig. 3b). This feedback is used for data-dependent cache movements (as e.g. in curve following during image preprocessing).

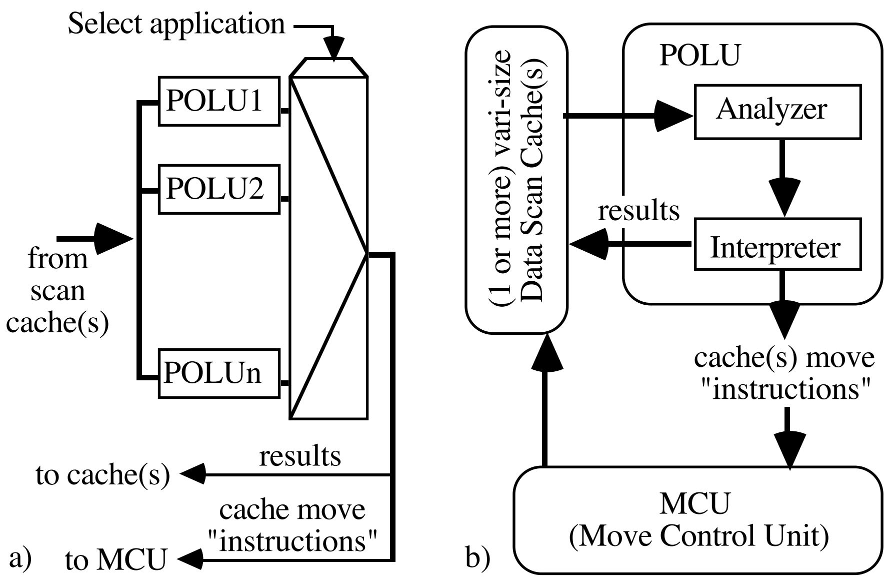


Fig. 3: POLU examples: a) multi tasking b) parallel pattern matching

The acceleration effect behind POLU implementation is based on the idea, that user-defined data operations should be that powerful, that no scanning of instruction sequences in some control store or program store is needed between two accesses to data memory. To achieve complex data operations the compiler for a von Neumann machine generates sequential code, i. e. sequences of simple instructions. Parallelism

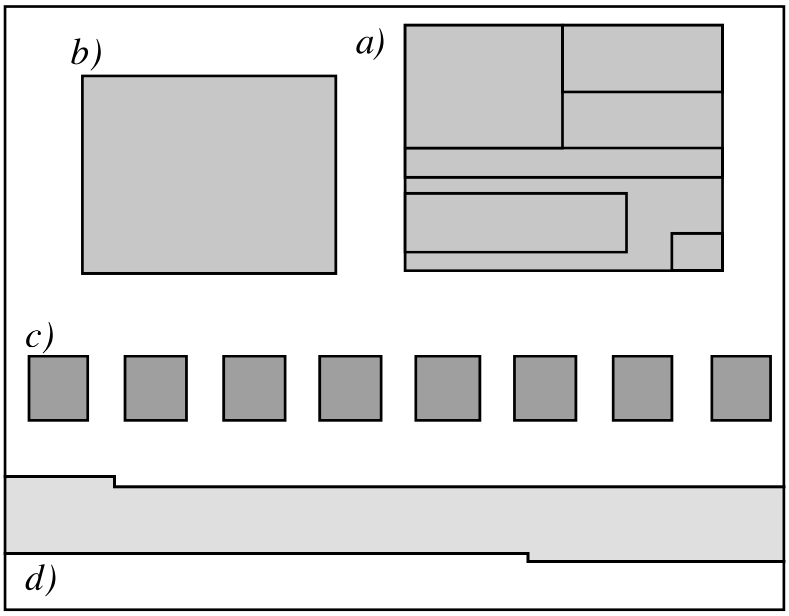


Fig. 4: 2-dimensional memory map example showing dimensional modes of MoM data memory: a) nested 2-D segments, b) 2-D, c) 3-D, d) 1-D.

is not provided, since from a classical ALU only one of such simple operations may be selected at a time. The MoMpiler for "programming" a MoM, however, generates personalization code for creating user-defined data paths - also highly parallel data paths - within its POLU. That's why here the flexibility of MoM goes far beyond the severe limitations of the *narrow-bandwidth fixed instruction set ALU* of the classical processor.

In contrast to the 1-dimensional von Neumann memory space, the ***map-oriented data memory*** is primarily 2-dimensional. At run-time the dimensionality of the memory can be switched to 1-D, 3-D, 4-D or more dimensions [14] (see fig. 4). The MoM data word format is highly flexible, since it is not bound to instruction formats. Its word size may be easily increased by adding more memory planes. This is one major difference to conventional computers, which have a fixed word length. Inside the data memory segments of arbitrary number and size can be defined under fully hard-wired MCU support such, that a MoM memory map (see fig. 4) looks similar to floor plans in VLSI (if 2-dimensional mode is used). Modern user interfaces encourage a graphical presentation of such memory maps.

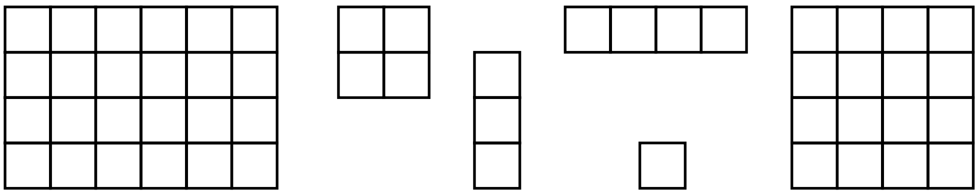
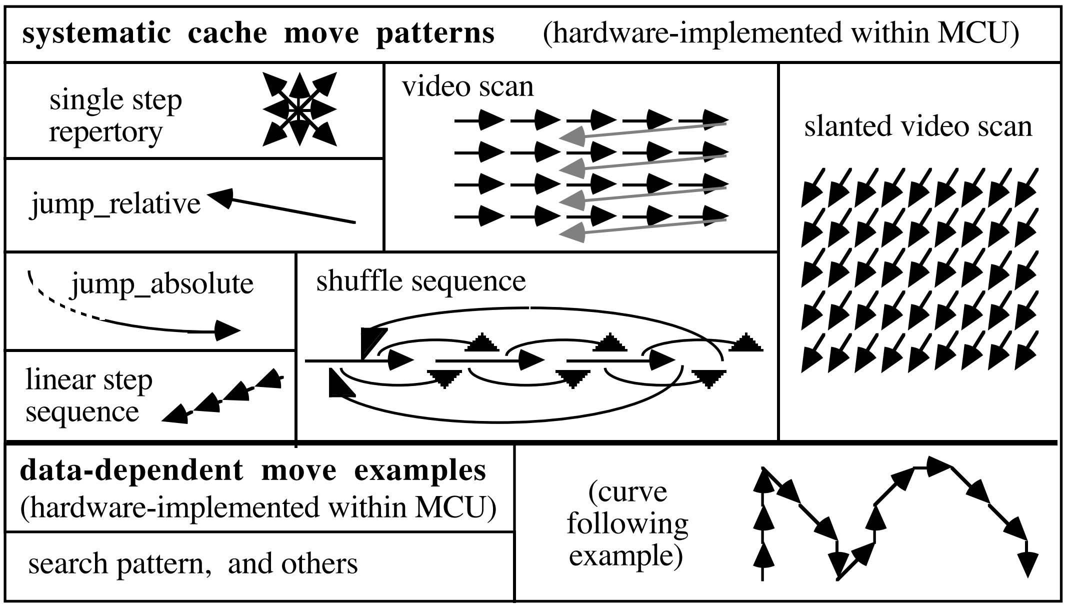


Fig. 5: Different format examples of the data scan cache

The vari-size ***data cache*** is a window to scan the MoM memory space [13]. It holds and automatically updates copies of the few adjacent words visible within the *scan* *window* , ready for fully parallel read/modify/write access by the POLU (fig. 3b). Cache size is adaptable to different applications and is reconfigurable at run time



systematic cache move patterns (hardware-implemented within MCU)
single step repertory
video scan
slanted video scan
jump_relative
jump_absolute
shuffle sequence
linear step sequence
data-dependent move examples (hardware-implemented within MCU)
search pattern, and others
(curve following example)

Figure 6: Cache move scheme examples

such, that much more than two or three words (like in a von Neumann ALU) are accessible at a time. By a 4-by-4 cache format, for example, 16 operands are directly connected to the POLU (also see fig. 5). A maximum-5-by-5 nMOS cache has been designed at Kaiserslautern, fabricated and tested (fig. 13). The memory / cache communication is minimized by an elaborate network of shift data paths within the cache which is automatically adapted to cache format and move step width and direction. The vary size cache has been designed, manufactured (in NMOS technology [27, 28]) and successfully tested, within the German multi university E.I.S. project, having been funded by the German Federal Ministry of Research and Technology.

The ***move control unit*** (MCU) hardware provides accessing sequences for a controlled cache 'movement' over the memory space. So this unit serves as the data sequencing part of the MoM. Two major cache movement strategies are available and may be combined:

- systematic move patterns (e.g. video scan or 'shuffle jumps')
- data-dependent move patterns (e.g. in curve following)

Cache movements are controlled by the **move manager** [14] using a structured jump generator hardware. It generates addresses for single steps to one of the eight nearest neighbours, for jumps in memory space, as well as step sequences for a 'video scan', shuffle sequence or other 'travel paths'. An incomplete summary of possible movements is shown in figure 6. Using the MoPL language cache movements may be specified in a procedural way to be translated by MoMpiler into MoM-executable task register sets (more details are found in [14]).

A second part of the MCU, the **task manager**, controls the coordination of several move patterns [14], which may be linked together to a so-called "MoM-Application". This part also is the communication interface between a MoM and its host. The Kaiserslautern MoM architecture minimizes MoM/host interaction for maximum throughput [14].

ONE PRINCIPLE - MANY ARCHITECTURES_

We have explained the innovative MoM principle by illustrating the Kaiserslautern example of a MoM architecture. But there is room to develop many other MoM architectures. This is similar to the von-Neumann domain where also many architectures have been proposed and developed through the decades. However, since a MoM architecture does **not** have the restrictions of a narrow-bandwidth fixed instruction format (inevitable property of all von Neumann architectures), it has a second dimension of flexibility.

This second dimension of flexibility - superior to the flexibility of the von Neumann principle - is the flexibility in the choice of techniques for the POLU implementation. Using PLDs or programmable gate arrays we get the "*programmable MoM*" with substantial acceleration factors, where electrical personalization requires only seconds. Taking normal gate arrays or even full-custom circuits for POLU implementation may achieve very high speed solutions, but needing a turn-around time of weeks or months (ASIC prototype delivery time). We call this alternative the "*partly customized MoM*" ("partly", since data memory, cache, and MCU still use standard circuits). For implementation of systolic systems the availability of the MoM provides additional alternatives, so that in total we have the choice between:

- simulation on a von Neumann machine
- monoprocessor emulation on a programmable MoM
- implementation on a partly customized MoM
- emulation on a WARP or similar machine
- fully customized ASIC or VLSI systolic array

Considering the throughput / cost tradeoffs and the range of applications to be covered by a specific MoM architecture, we could choose between different repertories of cache movement patterns (different "move instruction" sets), multiple caches (to emulate multiple data streams) or large caches. Also mixes between programmable MoM and partly customized MoM are feasible. The Kaiserslautern MoM architecture minimizes MoM/host interaction for the benefit of speed. However, also a partly or fully sequential MCU implementation inside the host would be possible. If an ALU is added to the POLU as one of the sub-POLUs (catalog circuit or library cell) even a hybrid von-Neumann / MoM architecture may be created.

SYSTOLIC EMULATION: MoM APPLICATION SUPPORT

The MoM with its host can execute any kind of data processing. Anyhow the MoM achieves very good performance in processing such problems, where the data can be efficiently mapped onto a two-(or more-)dimensional memory space. That's why systolic arrays, for example, can be directly mapped onto the MoM memory space.

Although the systolic parallelism is sequentialized, a surprisingly efficient implementation may be achieved. The slanted wavefront systolic data streams are converted into MoM data segments stored at fixed memory locations. The cache access location is moving such, that a single, but powerful processing element (POLU) connected to it serves as an equivalent to a basic systolic processing element (here used in a time multiplex mode). For the cache movements within such a data segment no program sequencing is needed, due to the hardwired move sequence repertory of the MCU (move control unit).

Tens of thousands of papers, representing millions of man years in research and development, have contributed to the theory of using the von Neumann architecture and its application development and the methodology has grown over 4 decades. Revolutionarily new processor principles such as the MoM have the disadvantage, that this know-how in existing application support tools (compilers, environments, operating systems) cannot be used any more. Even portations are impossible, since the underlying principles and their theoretical fundamentals do not fit any more. So a new theory of computation and application development is needed. This would be needed to avoid severe acceptance problems.

In fact, an elaborate MoM application theory is already existing, which provides a growing rich repertory of methods and tools. It just has to be adapted to the MoM principles: it is the rapidly developing theory of systolic processing. It just has to be mapped onto the MoM principles.

For a surprisingly wide variety of application areas (see [25] and many papers in [16]) this currently very active scene has developed a powerful methodology of data dependency mapping onto the mostly 2-dimensional implementation space of VLSI resources. So this also is a kind of map-oriented processing, similar to the principles of the MoM. For further application problem preparation, needed for the MoM, these results have mainly to be remapped into a more refined time step scale. This mapping is very simple, so that the derivation of the theory of MoM processing from the theory of systolic processing is a relatively easy task. The conclusion is, that the theory of MoM application support is almost ready for use. We in Kaiserslautern, for instance, have implemented a systolic array synthesizer SYS^3 [17], which also will serve as part of our MoM development environment (MoM-DE), currently being implemented (compare fig. 7).

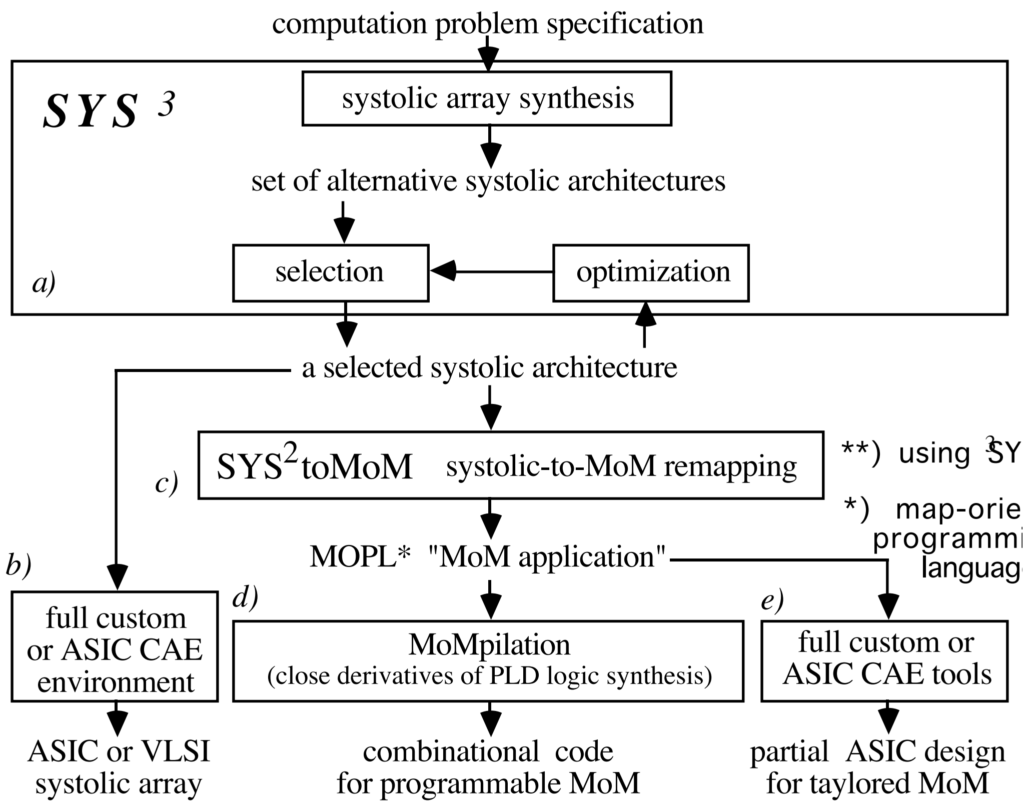

computation problem specification
SYS 3
systolic array synthesis
set of alternative systolic architectures
selection
optimization
a)
a selected systolic architecture
SYS2toMoM systolic-to-MoM remapping
c)
**) using SY
*) map-orie
programmi
languag
b)
MOPL* "MoM application"
d)
e)
full custom or ASIC CAE environment
MoMpilation
(close derivatives of PLD logic synthesis)
full custom or ASIC CAE tools
ASIC or VLSI systolic array
combinational code for programmable MoM
partial ASIC design for taylored MoM

Fig. 7: Application support: MoM-DE (a, b, d) and ASIC solution support (c, e)

For application development a special MoM development environment (MoM-DE) featuring a high level language MoPL (MoM Programming Language [14]) is available, which also supports systolic-to-MoM remapping. (Conventional compilers cannot be used, since they generate sequential code.) That's why the MoM-DE includes the 'MoMpiler', which provides CAE tools like PLD 'programming' tools [26] needed to code desired operations as POLU personalizations. Cache movements may be expressed in MoPL in a procedural way to be translated into MoM-executable MCU task register sets (more details are found in [14]).

MAPPING SYSTOLIC ARRAYS ONTO THE MoM

A typical two-dimensional systolic array (fig. 8) is used to illustrate the mapping onto the MoM. Two major methods can be used for this mapping, depending on the number of independent caches provided by the MoM. The dynamic method requires only one cache, which is used to access all data streams in a temporary execution field in the map-oriented memory (fig. 8). Using the dynamic method, the data streams are shifted during the read / modify / write cycle inside the cache. The static

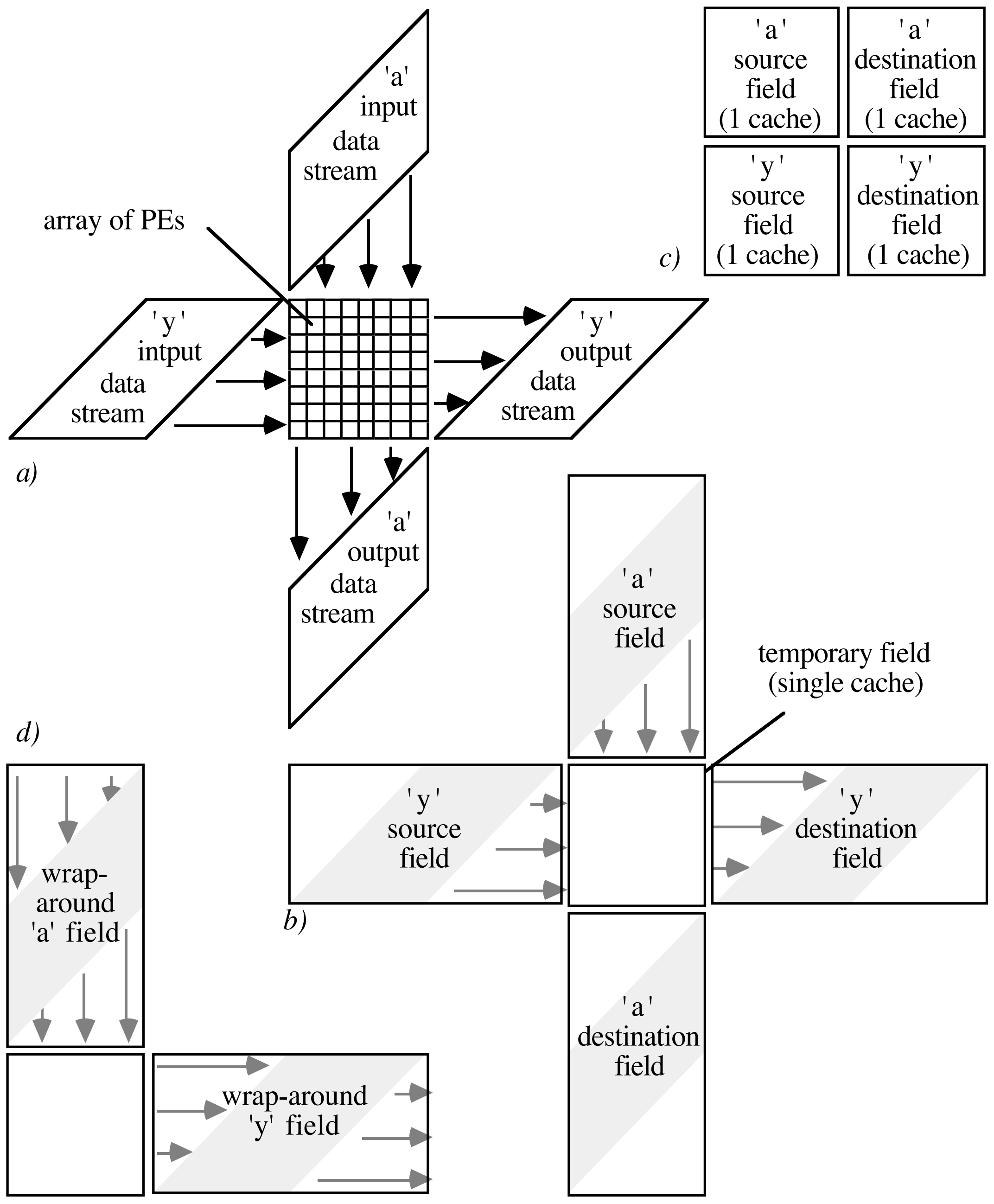


Figure 8: MoM memory maps for 2-dimensional systolic array: a) systolic array illustration, b) map for dynamic method (single cache), c) map for static method (multiple caches), d) wrap-around-version of c).

method uses several caches, one for each data stream. Here the different data streams are statically stored in different memory maps, with a cache each to access data in this entire map (fig. 8). The several caches are connected to one POLU, which performs the operations of one processing element.

The following algorithm and its corresponding systolic array will explain its mapping onto the MoM using the dynamic method. The algorithm

```
for j0 := 1 to n1 do
    for j1 := 1 to n2 do
        for j2 := 1 to n3 do
            y(j0) := y(j0) + a (j1, j2) ;
        end j2;
    end j1;
end j0;
```

is transformed (considering n1 = n2 = n3 = 3 and using SYS^3[9]) into the systolic array shown in figure 5a. The data dependency vectors found by SYS^3 are

$$d\_a = (\ 0\ 1\ 0\ ) ;\ d\_y = (\ 0\ 0\ 1\ )$$

and the space-time transformation matrix is

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

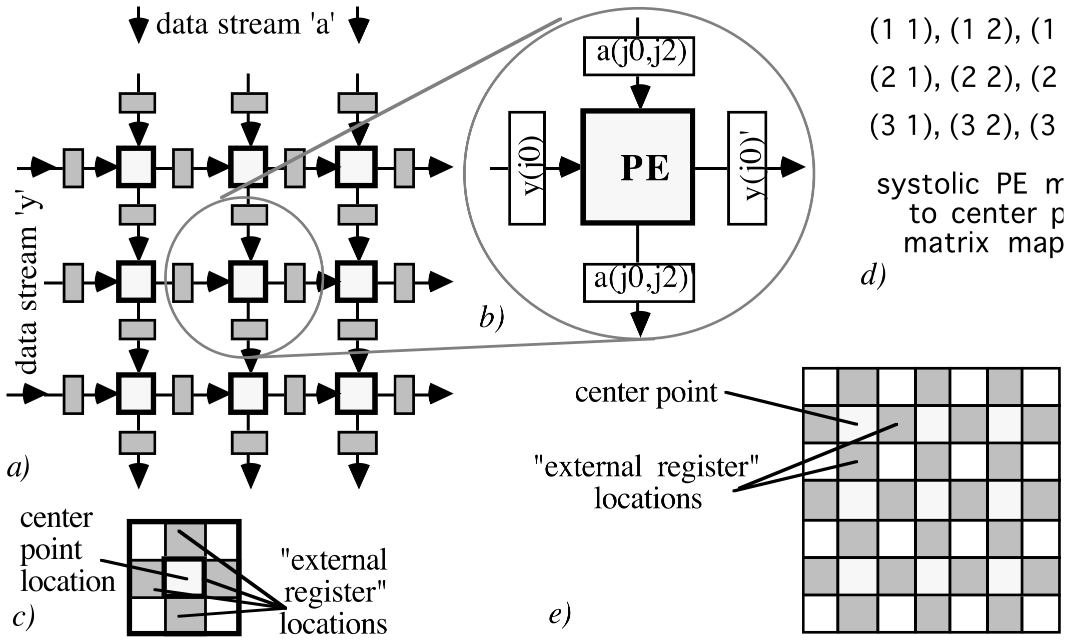


Fig. 9: A 2-dimensional systolic array example: a) global view, b) local view ,c) scan cache size and allocation, d) systolic PE to MoM mapping, e) temporary field map

The time transformed data dependencies, also provided by SYS^3, are

$$M{*}d\_a = (\ 1\ 1\ 0\ ) ;\ M{*}d\_y = (\ 1\ 0\ 1\ ),$$

where the first component gives the number of registers used for this data stream and the second and third component determine the direction of the data stream.

These time transformed data dependencies are used to determine the cache adjustment for the MoM mapping. For each external register (direction is not (0 0)) two 'pixels' are reserved, internal registers (direction is (0 0)) require only one 'pixel' in the cache. The number of external registers also provides the stepwidths for the cache movements. The cache adjustment can also be directly derived from the local view (fig. 9b) including **all** external registers around a PE. For the PE a center element is reserved. Fig 9c shows the result: a 3-by-3 cache adjustment. Fig. 10 shows more complex examples: one with pipelined external registers (a) and its cache adjustment (b). Also hexagonal systolic arrays can be mapped: local view example (c), its orthogonalization scheme (d) and its cache adjustment (e).

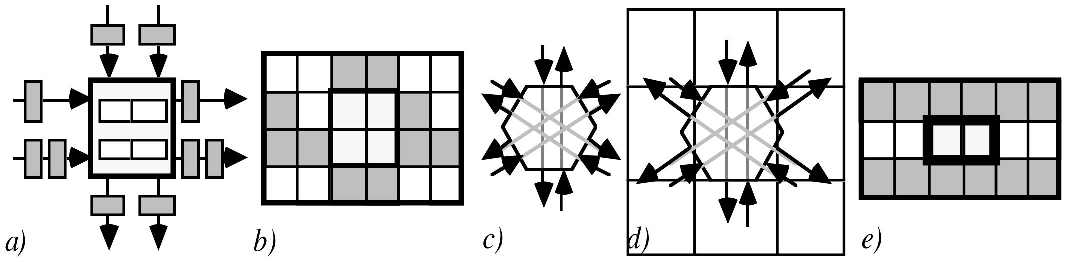


Fig.10: Complex processing element mapping examples: a) orthogonal systolic array local view, b) its scan cache mapping adjustment, c) hex array, d) its orthogonalization scheme, e) its scan cache mapping adjustment

Define the problem-oriented logic unit: The POLU of the MoM has to perform the data manipulations and data shifts done in one PE of the systolic array. In the example the output y(j0)' := y(j0) + a(j0, j2) is produced at the right external register location, while a(j0) is shifted down unchanged.

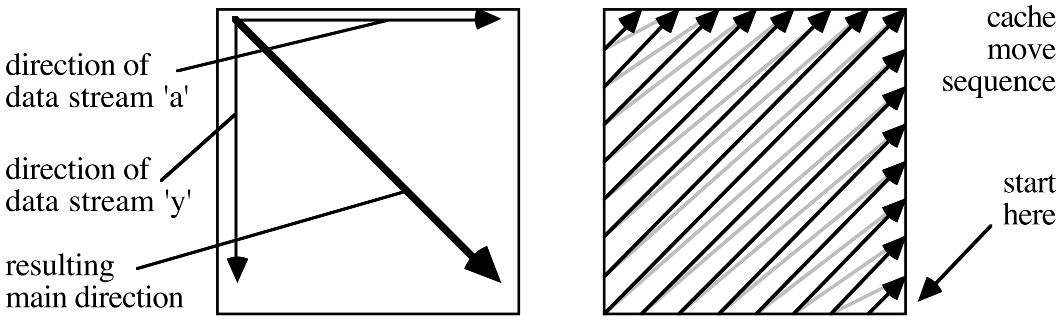


Figure 11: Cache moves for temporary field

Map the floorplan of the systolic array onto the Map-oriented memory: The space transformation (M * vector of index variables) done by SYS3 provides a list of coordinates used to place the processing elements for the floorplan of the systolic array. This list, together with the cache size, is used to determine the size and mapping of the temporary field in the map-oriented memory. This size directly implies the limits of this field, which are used as jumplimits for the cache move patterns. The example in figure 10 c, d, e illustrates this mapping.

properties \ processing principle used	**systolic array**	**WARP machine**	**MoM** (programmable version)
dimension of PE array	1-D, 2-D	1-D	single PE only
flexibility of PE	taylored	programmable	programmable
location of PE(s)	fixed within PE array	fixed within PE array	PE accessing location: random access within RAM memory space
location of data at run time	moving data stream	moving data stream	data within fixed segments within RAM data memory
multiple data streams capability	several data streams: up to 8 different directions	maximum of two data streams (opposite directions)	highly efficient emulation of any multiple data streams by multi-port mode of RAM using multiple data scan caches

Table 2: Comparison of systolic arrays, WARP and programmable MoM

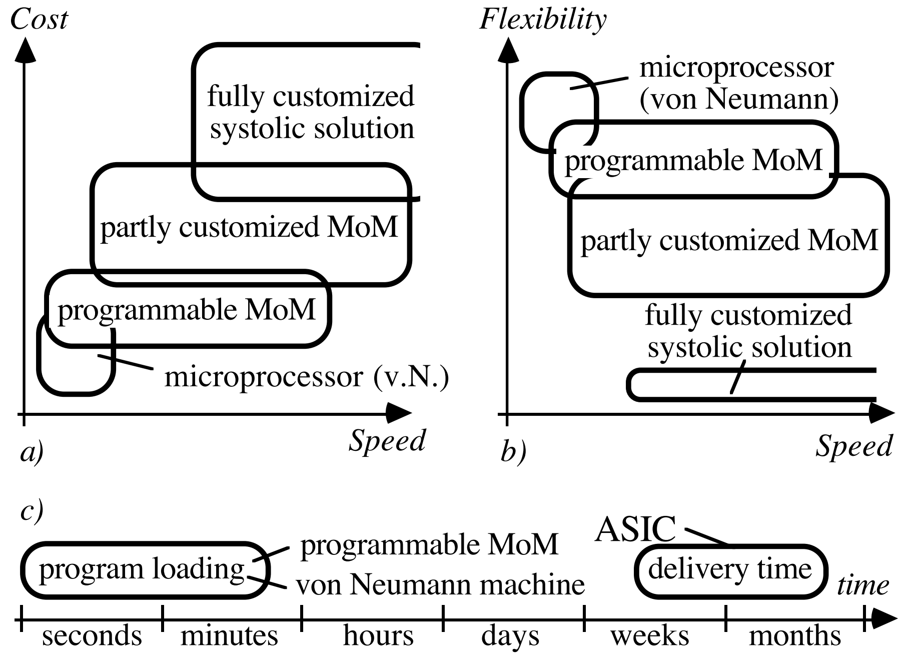


Figure 12: The MoM - filling a gap in algorithm implementation space

Defining the cache move pattern: The parallel work of the many PEs in the systolic array has to be done sequentially by a single cache in the MoM. I.e.: the cache has to move in a way that reflects one time step t_i of the systolic array by many time steps

t_{i1} - t_{in} of the MoM, where n is the number of processing elements in the systolic array.

Fig. 13. Plot of max.-5-by-5 version nMOS scan cache circuit feasibility study

To meet this requirement the cache movements are in opposite directions of the data streams in the systolic array. In the example array the data streams flow from left to right and simultaneously from top to bottom. The two data stream directions are combined to a resulting main direction (diagonally down and left) which also reflects the front of executions in the array. Now the cache is moved in opposite direction to this resulting main direction (figure 11) along the locations having the same execution time in the systolic array. This move pattern has to be performed for each systolic time step. Such a move pattern is provided by the parametrized repertory of hardwired data scan sequences and needs no program sequencing at all.

If no resulting main direction can be found (i.e. data streams move in opposite directions), there are two possible solutions for the cache movements.

1.: A single cache performs two move sequences. First, the cache is moved opposite to the one data stream using a POLU which only produces new data for this data stream. Second, the cache follows in opposite direction the second data stream using a POLU manipulating this data.

Fig. 14. Plot of nMOS electrically programmable gate array feasibility study

2.: A second cache is spent. The first cache follows in opposite direction the first data stream while simultaneously the second cache moves against the second stream. The caches influence "their" data stream by matching different POLUs.

A dynamic and a static method for the mapping is possible. The dynamic method requires only one cache, but the data streams are shifted in the memory. The static method uses several caches on fixed data. Comparing the run-time performance of the 'real' VLSI array and the MoM implementation of the same array (using one cache and dynamic method) we get a slow-down factor of

(number of PE's * number of different data stream directions)

+ number of data items to be shifted in

+ number of data items to be shifted out.

COMPARISON OF THE MoM WITH SYSTOLIC ARRAYS AND THE WARP MACHINE

The main properties of VLSI systolic arrays, the WARP machine and a programmable version of the MoM are compared in table 2. The MoM processor principles fill the wide performance gap between the slow but most flexible von Neumann solution (also for systolic processing) and the very high performance fully customized VLSI solutions (fig. 12). It also fills the performance gap between von Neumann monoprocessors and the WARP machine, but it is substantially cheaper than the WARP and other parallel computer systems. The MoM-DE supports the highly flexible programmable MoM as well as VLSI systolic array design.

CONCLUSIONS

Procedure SYS^{2}toMoM to map systolic systems onto the MoM has been introduced in this paper. This mapping allows to use the MoM as a systolic array simulator or emulator to implement systolic algorithms and/or to experiment with alternative systolic designs. It has been illustrated, that SYS^{2}toMoM serves as an important part of MoM-DE (MoM development environment) which is a powerful application development environment for the highly flexible programmable MoM, for the partly tailored MoM, and for fully tailored systolic array hardware. The principles of the MoM have been explained: acceleration principles for non-von-Neumann monoprocessors avoiding 2 of the 3 von Neumann performance bottle necks.

The MoM has been developed at Kaiserslautern, where a demo set-up, connected to an *eltec* image processing system, has been personalized with design rule check, Lee routing, image preprocessing and other applications. For the Kaiserslautern architecture example of the MoM essential standard hardware parts in nMOS technology have been designed, fabricated and tested, and parts of the MoM-DE have been developed within the German multi university E.I.S. project, having been funded by the German Federal Ministry of Research and Technology. Many other MoM architectures are feasible. In data processing and also in systolic processing the MoM processor principles fill the wide performance gap between the slow but most flexible von Neumann solution (classical software) and fully customized ASIC solutions. A dramatic acceleration factor has been demonstrated on an experimental hardware for several application examples.

Fig. 15. Experimental MoM demo set-up with ***IBM*** *PC/AT user communication interface and VME-bus-based* ***eltec*** *image processing computer system [30]*

REFERENCES

[1] H.T. Kung: Let's Design Algorithms for VLSI. Caltech Conf. VLSI, 1979.

[2] C.E. Leiserson, H.T. Kung: Algorithms for VLSI Processor Arrays; in: C. Mead, L. Conway: Introduction to VLSI Systems, Addison-Wesley, 1980.

[3] H.T. Kung: Why Systolic Architectures?; Computer, Jan. 1982.

[4] D.I. Moldovan: On the Design of Algorithms for VLSI Systolic Arrays; Proceedings of the IEEE, Jan. 1983.

[5] P. Quinton: Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations; 11th International Symposium on Computer Architecture, 1984.

[6] G. Li, B.W. Wah: The Design of Optimal Systolic Arrays; IEEE TC Jan'85.

[7] P. Quinton, P. Gachet: DIASTOL User's Manual; res. report, IRISA, 1984.

[8] D.I. Moldovan: ADVIS: A Software Package for the Design of Systolic Arrays; IEEE Transactions on Computer-Aided Design, Jan. 1987.

[9] R.Hartenstein, K.Lemmert: SYS3 - A CHDL-Based CAD System for the Synthesis of Systolic Architectures; IFIP CHDL'89, Washington, U.S.A., June 1989; North Holland, Amsterdam / New York 1989. - *also see* [17]

[10] M. Annaratone et al.: The WARP^{SM} Computer: Architecture, Implementation and Performance; Carnegie Mellon University, 1987.

[11] S. Kuppuswami, F. André: MISS: A Distributed Simulator for Systolic Archi-

tectures; in: M. Cosnard et al.: Parallel Algorithms and Architectures, North-Holland, Amsterdam / New York 1986.

[12]L. Omtzigt, E. Theodore: SYSTARS: A CAD Tool for the Synthesis and Analysis of VLSI Systolic/Wavefront Arrays; Int'l Conf. on Syst. Arrays '88.

[13]A. Hirschbiel, M. Weber: Methodology of MoM application support. Kaiserslautern, 1989.

[14]A. Hirschbiel, M. Weber: The Kaiserslautern MoM Architecture and its Implementation, report, Kaiserslautern, 1989.

[15]R. Hartenstein, A. G. Hirschbiel, M. Weber: MoM - map-oriented machine: a partly custom-designed architecture compared to standard hardware; Proc. IEEE CompEuro '89, Hamburg, FRG, IEEE Press, New York, 1989.

[16] K. Bromley, S-Y. Kung, E. Swartzlander: Proc. Int'l Conf. on Systolic Arrays, San Diego '88; IEEE Computer Soci. Press, Washington, DC, 1988.

[17] J. McCanney, J. McWhirter, E. Swartzlander: Proc. Int'l Conf. on Systolic Arrays, Killarney '89; IEEE Computer Society Press, Washington, DC 1989.

[18] H.T. Kung: Computational Models for Parallel Computers; CS-report CMU-CS-88-164, Carnegie-Mellon Univesity, Pittsburgh, PA, 1988.

[19]T. Blank: A Survey of Hardware Accelerators used in Computer-Aided Design; IEEE Design&Test, August 1984.

[20] K. Bastian et al., VLSI-Algorithmen: Innovative Schaltungstechnik statt Software - Shuffle Sort, VDI-Berichte 550, 1985.

[21] R. Hartenstein, A. Hirschbiel, M. Weber, A Flexible Architecture for Image Processing, Microprocessing and Microprogramming 21, 1987.

[22] R. Hartenstein, A. Hirschbiel, M. Weber: Map-oriented Machine; Ambler, Agrawal, Moore): Hardware Accelerators for Electr. CAD, Adam Hilger, 1988.

[23] B.C. Cole: Programmable Logic Devices: The second generation, Electronics, May 12, 1988.

[24] R. Freeman: User-programmable Gate Arrays; IEEE Spectrum, Dec. 1988.

[25]A. Fisher, H. Kung: Special-Purpose VLSI Architectures: General Discussion and a Case Study; (S. Y. Kung et al.:) VLSI and Modern Signal Processing, Prentice Hall, 1985.

[26]N.N.: LOG/iC User Manual; ISDATA GmbH, Karlsruhe, 1988.

[27]R. Müller: Multshift, Layout und Simulation; Projektarb.; Kaiserslautern, '88.

[28]H. Nicklaus: Multshift - Spezifikation und Problemanalyse; Projektarbeit; Kaiserslautern 1988.

[29]T. Mayer: Dynamically programmable Logic Array; Kaiserslautern, 1988.

[30] N.N.: eltec - 68k - system user manual; eltec GmbH, Mainz, F.R.G., 1986