Algorithms and architectures of multimedia systems

Systolic Parallel Processing

Lecture Notes

Matej Zajc

University of Ljubljana
2011/12

1 Introduction to systolic parallel processing

This chapter introduces the area of systolic arrays and systolic algorithms. The systolic mode of parallel processing has gained a tremendous interest due to elegant exploitation of data parallelism inherent in computationally demanding algorithms from different fields of research. The fundamental theory behind the systolic arrays is presented to give a needed background for understanding the systolic mode of parallel processing.

1.1 Parallel processing and parallel computing

The most widely known mode of information processing is sequential processing in which the operations that have to be performed to obtain the solution of a problem are carried out in a sequence by a single processing unit [Petkov93]. In case the processor is computer it is termed sequential computing. The most common computer design is still based on a von Neumann architecture, which is composed of a single processor executing single sequence of instructions on a single stream of data known also under SISD machine. In von Neumann architecture both instructions and data come from a common memory, while in a Harvard architecture data and instructions are stored in different memories and are fed to the processor by two different paths [Hennessy96]. The conceptual simplicity of the sequential approach and associated technology at the time led to digital computer revolution.

For most applications sequential computers present an efficient solution providing the required processing power. Nevertheless, there were, are, and will be problems that require computational power that exceeds the resources of the most powerful sequential computers at the time for several orders of magnitude. These problems are not only those referred to as large-scale scientific computing problems, but also several real-time signal and image processing problems. Some of them were covered in the previous chapter.

In order to achieve radically higher computational throughput new approaches must be considered. One of already prominent approaches is solving problems in parallel. As opposite to sequential processing, parallel processing could be regarded as a mode of information processing in which at least two processing units cooperate while carrying out in parallel the information processing operations that belong to a problem. During the process each processing unit works on a different part of a problem[1].

Usually, multiprocessing and parallel computing are used as synonyms referring to a number of electronic processing units involved in parallel computing of a particular computational problem. If number of these processing units is large, in order of hundreds or thousands, the attribute massively is added.

[1] For the terminology we refer to [Petkov93].

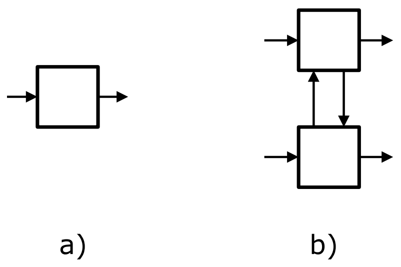


Figure 1: Sequential processing a), and parallel processing b).

The main reason for parallel execution of a problem is the acceleration of computations. If single processing unit can accomplish the task in time *T* then *n* processing units could ideally accomplish it in time *T/n*. This is termed a linear speed-up. In most cases the actual speed-up achieved on parallel computers is considerably smaller then the desired linear speed-up.

1.2 Classification of parallel computer architectures

With the tremendous development of the VLSI technologies a wide variety of computer architectures have been proposed. In the last 20 years infinite number of new computer architectures for parallel processing have been innovated based on major approaches of parallel computing developed in 1960s and 1970s. With increasing number of computer architectures, it has become important to find a way for efficient classification. The classification should distinguish those structures that are significantly different and at the same time reveal the similarities between apparently divergent designs.

Taxonomies, or classification schemes, are widely used for classifying the world. For example, whales are grouped with other mammals rather than with fish. Their relationship with mammals is far more important than their relationship to fish even though it might be less obvious at the first glance [Duncan90].

Due to diversity of parallel architectures, diverse definitions have been proposed. Up to now many authors classified the computer architectures [Flynn66, Duncan90, Skillicorn88, KungSY88, Krishnamurthy89]. One of the most widely adopted classifications is Flynn's taxonomy [Flynn66] based on instruction and data stream organisation. The disadvantage of Flynn's classification of architectures is that it does not discriminate clearly between different multiprocessor architectures. Duncan's taxonomy [Duncan90] on the other hand solves some of the problems in Flynn's taxonomy. These two taxonomies are briefly presented in a sequel.

Two taxonomies [Flynn66, Duncan90] are presented with a purpose to expose parallel architectures from different points of view. The classification is also interesting from evolution point of view. The main reason for inclusion of the subject is mainly to place systolic mode of operation in the area of parallel processing.

1.2.1 Flynn's classification

Flynn’s taxonomy divides the entire computer world into four groups: SISD, SIMD, MISD and MIMD. Flynn classifies architectures on the presence of single or multiple streams of instructions and data into:

- SISD (Single Instruction Single Data); defines serial computers.
- SIMD (Single Instruction Multiple Data); involves multiple processors simultaneously executing the same instruction on different data.
- MISD (Multiple Instruction Single Data); involves multiple processors applying different instructions to a single datum; this hypothetical possibility is generally unrealistic but which Flynn affirms to include specialised streaming organisation.
- MIMD (Multiple Instruction Multiple Data); involves multiple processors autonomously executing diverse instructions on diverse data.

The four well-known categories are presented in Figure 2 exposing the architectural differences. Single processor computers are representatives of SISD category. SIMD category, also refereed to as synchronous parallelism, contain vector computers as well as array computers. MISD category has quite free interpretation and is even non-existing by some authors. Pipeline computers are classified into this category by Bräunl [Bräunl93]. As opposite to SIMD, MIMD category is refereed to as asynchronous parallelism and covers multiprocessor distributed computer systems.

The classification provides useful shorthand for characterising computer architectures. There are several structures that do not smoothly fit into any of the four categories. Therefore Flynn’s taxonomy became insufficient for classifying various modern computers such as pipelined vector processors, systolic arrays, etc. [Duncan90].

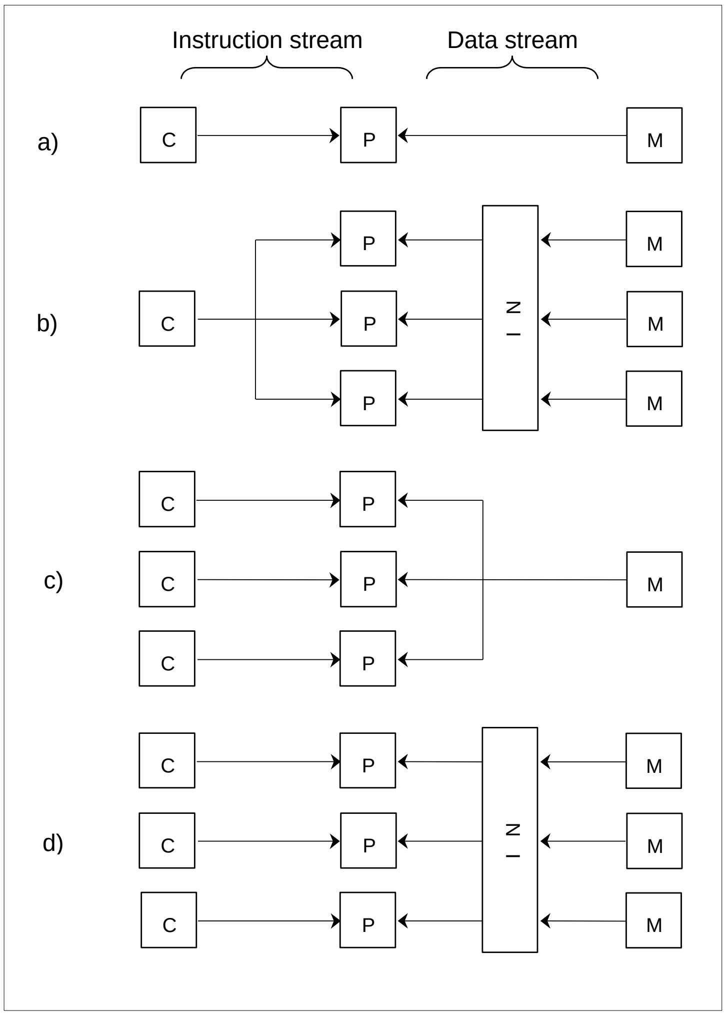


Figure 2: Flynn’s taxonomy of computer architectures: a) SISD, b) SIMD, c) MISD, and d) MIMD (C: control unit, P: processor, M: memory, I N: interconnection network).

1.2.2 Duncan's classification

Duncan's taxonomy placed recent architecture innovations in the broader context of parallel architectures. Accordingly to Duncan, taxonomy should satisfy the following essentials [Duncan90]:

- Maintain elements of Flynn's useful taxonomy based on instruction and data stream;
- Exclude architectures incorporating only low-level parallel mechanism that have become commonplace features of modern computers;

- Include pipelined vector processors and other architectures that intuitively seem to merit inclusion as parallel architectures, but which are difficult to gracefully accommodate within Flynn's scheme.

Under the above conditions, a parallel architecture can be defined as an explicit, high-level framework for the development of parallel programming solutions by providing multiple processors, whether simple or complex, that co-operate to solve problems through concurrent execution.

Figure 3 shows the classification of processor structures corresponding to Duncan's taxonomy, which uses high-level categories to delineate the principle approaches and to define coherent spectrum of architectural alternatives. For details on different categories we refer to [Duncan90, KungSY88, Skillicorn88, Krishnamurthy89].

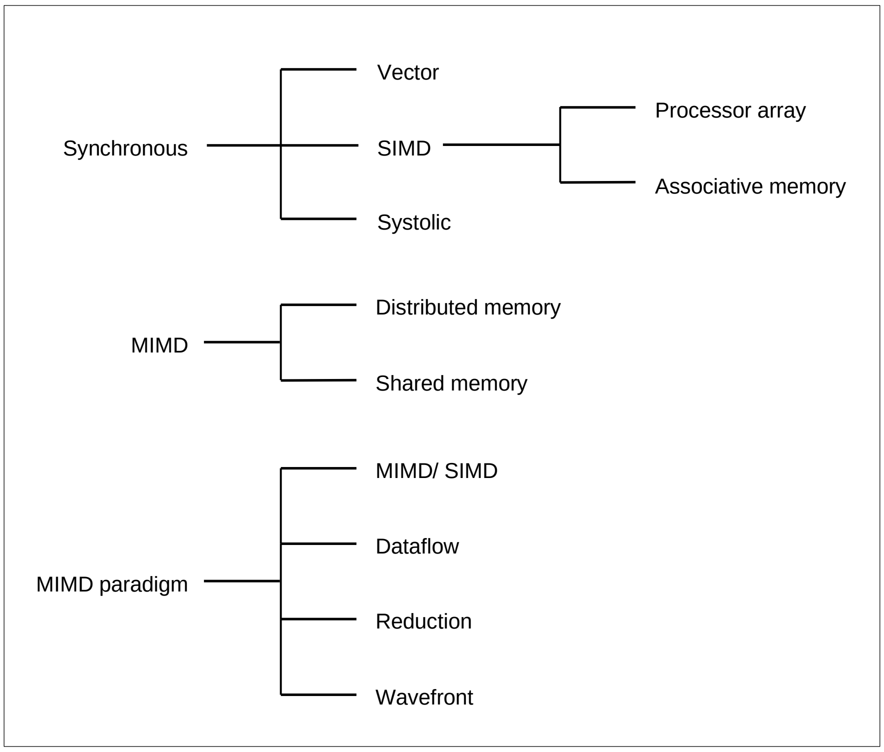


Figure 3: Duncan’s taxonomy of parallel computer architectures.

1.2.3 VLSI processor arrays

The systolic and wavefront arrays are also called VLSI processor arrays. Figure 4 places systolic arrays and wavefront arrays together with SIMD and MIMD architectures exposing the similarities and differences. The classification is made according to the synchronicity timing scheme and data I/O [KungSY88].

In systolic arrays and wavefront arrays data is pipelined through the boundary processors concurrently with processing. Systolic arrays use local instructions synchronised globally, while wavefront arrays utilise data driven capability. SIMD and MIMD, as a difference to pipelined data, use global data and control allowing broadcasts from a memory and a control unit. The main characteristics of the four computer architectures are briefly described in sequel [Petkov93, KungSY88, Bräunl93, Krishnamurthy89].

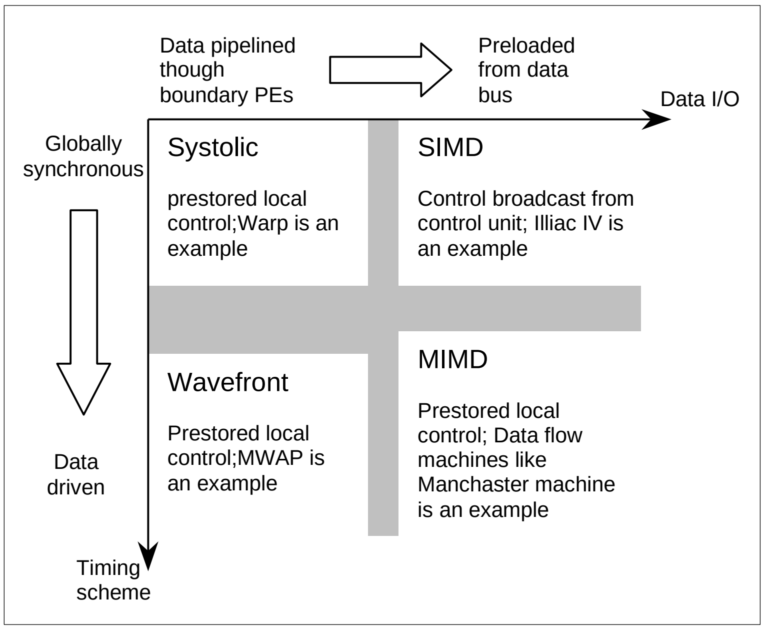

Data pipelined though boundary PEs
Preloaded from data bus
Data I/O
Globally synchronous
Data driven
Timing scheme
Systolic
prestored local control;Warp is an example
SIMD
Control broadcast from control unit; Illiac IV is an example
Wavefront
Prestored local control;MWAP is an example
MIMD
Prestored local control; Data flow machines like Manchaster machine is an example

Figure 4: Similarities and differences of SIMD architectures, MIMD architectures, systolic arrays, and wavefront arrays [KungSY87].

SIMD architectures

SIMD architectures typically employ a central control unit, multiple processors, and an interconnection network for either processor-to-processor or processor-to-memory communications. The control unit broadcasts a single instruction to all processors, which execute the instruction on local data. The interconnection network allows instruction results calculated at one processor to be communicated to another processor for the use as operands in a subsequent instruction.

MIMD architectures

MIMD architectures employ multiple processors that execute independent instruction stream using local data. MIMD can support parallel solutions that require processors to operate in a largely autonomous manner. MIMD architectures are asynchronous computers, characterised by decentralised hardware control. Software processes executing on MIMD architectures are usually synchronised by passing messages through an interconnection network or by accessing data in shared memory. MIMD computers support high-level parallelism at subprogram and task level.

Systolic architectures

The systolic architectures were first introduced in 1978 by Kung and Leisserson [KungHT82]. Systolic arrays are in general classified as high-performance, special-purpose VLSI computer systems that are suitable for specific application requirements that must balance intensive computations with demanding I/O bandwidths. Systolic architectures (systolic arrays) are organised as networks which are composed of a large number of identical, locally connected elementary processing elements. Data in systolic arrays are pulsed in rhythmic fashion from memory through PEs before returning to memory. A global clock and explicit timing delays synchronise the system. Modular processors united by regular and local interconnections provide basic building blocks for a variety of special-purpose systems. Systolic arrays address the performance requirements of special-purpose systems by achieving significant parallel computations and by avoiding I/O and memory bandwidth bottlenecks. The rest of the work is dedicated to systolic arrays.

Wavefront array architectures

Wavefront array processors combine systolic data pipelining with the asynchronous dataflow execution paradigm. Wavefront and systolic architectures are both characterised by modular processors and regular, local interconnection networks. However, wavefront arrays replace the global clock and explicit time delays used for synchronising systolic data pipelining with asynchronous handshaking as the mechanism for co-ordinating inter-processor data movements. Thus, when a processor has performed its computations and is ready to pass data to its successor, it informs the successor, sends data when the successor indicates it is ready, and receives an acknowledgement from the successor. The handshaking mechanism makes computational wavefronts pass smoothly through the array without intersecting, as the array's processors act as a wave-propagating medium. In this manner, correct sequencing of computations replaces the correct timing of systolic architectures.

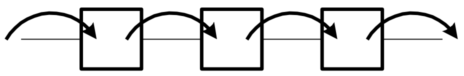
1.3 Systolic arrays and algorithms

1.3.1 The term systolic array

The term systolic array was introduced in the computer science by H. T. Kung et al. [KungHT78] at the end of 70's. A systolic array typically consists of a large number of similar processing elements interconnected in an array. The interconnections are local meaning that each processing element can communicate only with a limited number of neighbouring processing elements. Data move at a constant velocity through the systolic array passing from one processing element to the next processing element. Each of the processing elements performs computations, thus contributing to the overall processing needed to be done by the array.

Systolic arrays are synchronous systems. A global clock synchronises the exchange of data between directly communicating processing elements. Data can be exchanged only at the ticks of the clock. Between two consecutive clock ticks, each processing element carries out computation on the data which it has received upon the last tick and produces data which is sent to neighbouring processing elements at the next clock tick. The processing element can also hold data stored in the local memory of the processing element.

A data flow, which moves through systolic array, resembles the pulsing movement of the blood under contractions of the heart, which are called *systoles* in physiology (Figure 5). This analogy was the reason to give such computing structures the attribute systolic[2] [Petkov93].



[2] Search on the Internet or in the library's database for term *systolic* and one gets an interesting list of computer science books as well as various medical books.

Figure 5: Analogy of data movement in a systolic array with the pulsing movement of the blood [Petkov93].

Another explanation of the systolic term is connected with the overall system in which the systolic array plays a role of co-processor. A host computer supplies the systolic array with data and extracts the data, which has been processed, by the systolic array. The system resembles the blood circulatory system, in that the host pumps data through systolic array like the heart, which pumps blood through the arteries, capillaries and veins (Figure 6) [Petkov93].

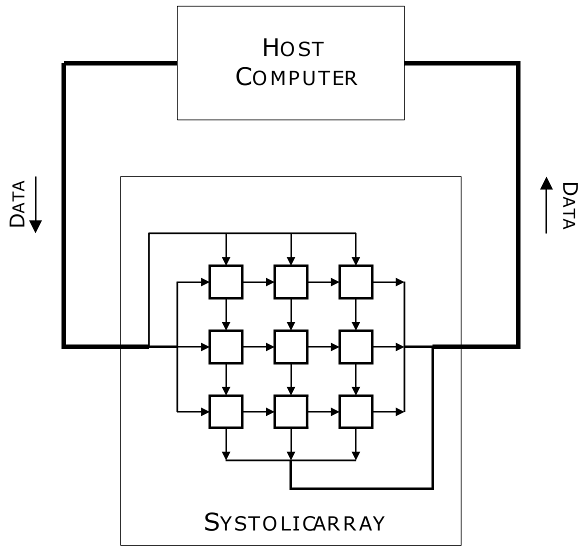


Figure 6: Analogy of the computing system with the blood circulatory system [Petkov93].

1.3.2 Features of systolic arrays

There are several similar definitions of systolic arrays [KungHT78, Ullman84, KungSY88, Megson92, Evans91, Johnson93]. According to Kung and Leiserson [KungHT78], we have the following well known definition:

"A systolic system is a network of processors which rhythmically compute and pass data through the system."

A more consistent definition of systolic arrays is presented below. A systolic array is a computing system, which possesses the following features [Petkov93]:

- **Network.** It is computing network employing a number of processing elements (PEs)[3] or cells with interconnections.
- **Rhythm.** The data are computed and passed through the network in a rhythmic and continual manner.
- **Synchrony.** A global clock times the execution of instructions and the communication data.
- **Modularity**. The network consists of one or, at most, a few types of processing elements. If there is more than one type of processors, the systolic array can usually be decomposed into distinct parts with one processor type.
- **Regularity**. The interconnections between the processing elements are regular and homogeneous. The numbers of interconnections for processing elements are independent on the problem size.
- **Locality.** The interconnections are local. Only neighbouring processing elements can communicate directly.
- **Boundary.** Only boundary processing elements in the network communicate with the outside world.
- **Extensibility.** The computing network may be extended indefinitely.
- **Pipelinebility.** All data is proceeded and transferred by pipelining. As presented in the sequel, it means that at least one delay element (register) is present between each two directly connected combinatorial PEs.

From the features presented above we can summarise that a large number of PEs work in parallel on different parts of the computational problem. Data enter the systolic array at the boundary. Once input into the systolic array, data is used many times before it is output. In general, several data flows move at constant velocities through the array and interact with each other during this movement, where processing elements execute repeatedly one and the same function. There is no transfer of intermediate results from the array to the host computer. Only initial data and final results are transferred between the host and the systolic array.

As long as VLSI technology is used, certain constraints should be analysed and incorporated in the design methodology. A few, most important, are short communication paths, limited input/output interaction, time consuming data transfer and synchronisation. These constraints

[3] Both notations are used.

are all incorporated in the above features of the systolic array. Features like regularity, modularity and locality are especially favourable from the VLSI point of view and make systolic arrays good candidates for the VLSI implementation [KungSY85, KungSY88].

1.4 Formal representation of systolic arrays

The systolic arrays are usually represented as array of processing elements and array of interconnections connecting the PEs with some particular pattern. In other words, systolic array is a collection of PEs that are arranged in a particular pattern. The pattern and the PEs are defined with the implemented algorithm.

The PEs are composed of combinatorial part and/or memory. The combinatorial part is responsible for arithmetic operations required by the systolic algorithm. By memory (registers) we denote delay elements within the PE that hold data and thus control data flow into and out of the PE. In general, no large memory is associated with PEs.

The PE can be formally defined as an abstract automaton [Petkov93]. Automata notations are extensively used as a fundamental representation of different computing structures. Automata with processing capabilities and memory are models of basic data storage and transfer functions.

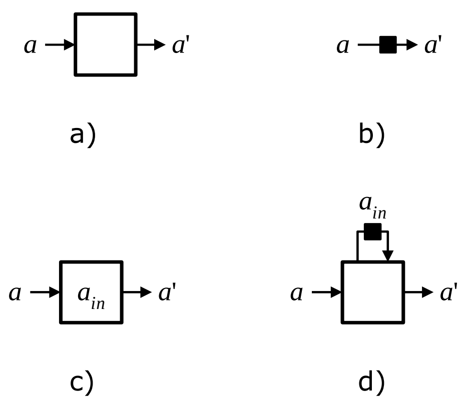


Figure 7: Automaton a) combinatorial automaton, b) delay element, c) automaton with internal variable (memory), and d) equivalent representation using delay element.

The automaton is in general a system that consists of *input*, *output* and *internal variables.* The functional behaviour of the automaton is usually described by the state equations that specify the values of its internal and output variables.[4] If the automaton does not contain an internal variable it is named a *combinatorial* automaton. And further, if the automaton merely delays the flow of the input data by one time step it is called *delay* element and thus presents a memory element.

Rippling, Broadcasting, Pipelining

The three important terms needed for the presentation of systolic mode of operation are presented next, namely rippling, broadcasting, and pipelining [Petkov93].

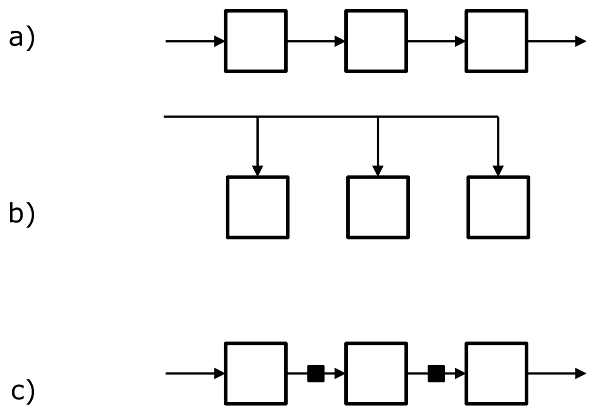


Figure 8: a) Rippling: the direct connection of combinatorial PEs results in rippling. b) Broadcasting: data is simultaneously input to the number of PEs via a common input. c) Pipelining: chaining of combinatorial PEs via delay elements.

Rippling occurs if combinatorial automata are directly connected. Since there is no delay elements data items propagate along the chain as a front of a wave. The next data item can only be input after the previous has exited the chain. Therefore the clock period is defined by the overall combinatorial delay of the chain.

Broadcasting is the case where a number of PEs receive the same signal from the common input line. This form of global data transfer is called broadcasting and is undesirable from VLSI implementation point of view, since long connections present long delay in fast clocked systems.

[4] In general, the automaton can have several inputs, outputs and internal variables.

Pipelining is a contrast to rippling. The cells are chained via delay elements. A data item moves under the control of the clock from cell to cell. As a result the clock period is determined by the combinatorial delay of a single cell. In pipeline, a sequence of data items can be processed in parallel, as opposed to just one datum at a time in a rippling chain.

Parallel computing structures can be efficiently represented with the appropriate combination of many automata. In addition, if the automata are connected to certain neighbouring automata, the resulting structure is called *cellular* automaton. In cellular automaton, the delay elements serve to quantise the data flow in time steps.

Systolic automata can be considered as a special class of cellular automata [Petkov93]. The systolic automaton is completely pipelined version of a cellular automaton. In addition, the PEs have equal combinatorial delay. The complete pipelining is achieved by introducing the delay element into each connection between two PEs of the systolic automaton. As a result the broadcasting is eliminated[5]. As mentioned, it is important that the PEs exhibit equal combinatorial delays. This restriction does not mean that the PEs must be of one type but rather that the PEs have comparable combinatorial delays. This guarantees a balanced load of the PEs. Otherwise the efficiency of the system would be dominated by its slowest component.

Systolic array is then defined in the context of systolic automata with an additional requirement that the PEs are of one type and regularly arranged and interconnected [Petkov93]. The regularity of the structure is especially important from the VLSI implementation point of view. The regularity contains the arrangement of the PEs as well as the interconnections between them. The requirement of PEs of the same type is an advantage from economic as well as from programming point of view. The restriction to one type of PEs also results in equal combinatorial delay and load balancing.

[5] If broadcasting is allowed in systolic processing the algorithm would be called *semi-systolic*.

1.4.1 Examples of systolic array structures

Linear systolic array

In a linear systolic array, processing elements are arranged in one-dimension. The inter-connections between the processing elements are nearest neighbour only. Linear systolic arrays differ relative to the number of data flows and their relative velocities. Representatives of linear systolic arrays are matrix-vector multiplication, one-dimensional convolution (FIR filtering), etc. [Petkov93].

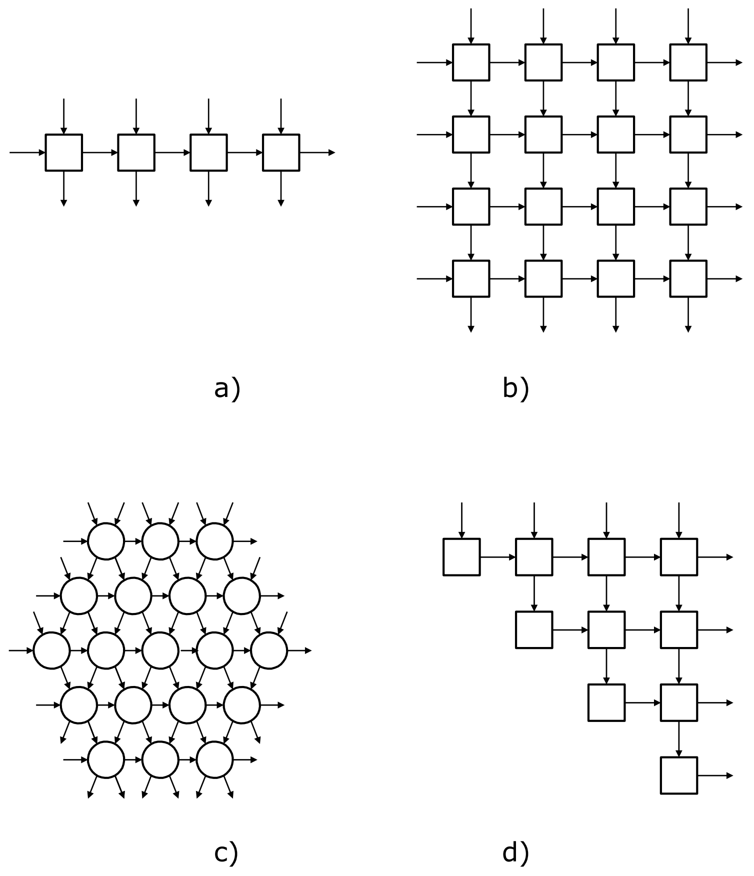


Figure 9: Examples of mesh-connected systolic structures. a) linear systolic array, b) orthogonal systolic array, c) hexagonal systolic array and d) triangular systolic array.

Orthogonal systolic array

In an orthogonal systolic array processing elements are arranged in a two-dimensional grid. Each processing element is interconnected to its nearest neighbours to the north, east, south and west. Again, the systolic arrays differ relative to the number and direction of data flows and the number of delay elements arranged in them. The most typical representative of this class is one of possible mappings of the matrix-matrix multiplication algorithm (0).

Hexagonal systolic array

In a hexagonal systolic array processing elements are arranged in a two-dimensional grid. The processing elements are connected with its nearest neighbours where inter-connections have hexagonal symmetry. A particular mapping of the matrix-matrix multiplication algorithm results in a hexagonal array [Wan98].

Triangular systolic array

The term triangular systolic array refers to two-dimensional systolic array where processing elements are arranged in a triangular form. This topology is mostly used in different algorithms from linear algebra. In particular it is used in Gaussian elimination and other decomposition algorithms [Petkov93].

1.5 Formal representation of systolic algorithms

Array algorithms are a subset of parallel algorithms. Generally speaking, an array algorithm is a set of rules solving a problem in a finite number of steps by a multiple number of interconnected processors [KungSY88]. The most suitable problems for array algorithms are the compute bound algorithms, where the number of computations is much greater then the number of I/O communications. The I/O bound problems are in contrast to compute bound problems where the number of I/O operations is dominant. In general, computation bound algorithms that can be represented in a recursive form and locally dependent are candidates for array algorithms.

Systolic algorithm is the algorithm implemented on a systolic array. In the systolic algorithm PE performs a fraction of the necessary computations and thus contributing a part to the overall computations carried out by the systolic array.

Systolic algorithms could be classified into the following three groups:

- Hard systolic algorithms.
- Hybrid systolic algorithms.
- Soft systolic algorithms.

The classification is based on the degree of programmability. Hard-systolic algorithms have a very low degree of programmability. The topology of the systolic array as well as PE's operation are fixed. This type of algorithms is highly desirable from VLSI implementation point of view. Due to the limited flexibility the designs are rather special purpose. Hybrid-systolic algorithms have a medium degree of flexibility. As a consequence, micro programming or simple control tags are added to data flows. The idea is to allow a number of PE types to be implemented by the same processor where the connection topology is still systolic. The last group of algorithms, soft-systolic, provides a high degree of programmability. The main idea is to keep the systolic computation as a design principle.

Here we present the notation used in for the representation of systolic algorithms. A systolic algorithm is usually presented in a sort of graphical form. It consists of a systolic array, PE assignments, and input data positioning. The systolic array reveals the topology (organisation of the PEs) and the interconnection pattern. PE assignments present the operation required by the algorithm. The input data positioning holds the information how the data must be organised.

In previous section the formal representation of systolic arrays was given in the context of automata notation. Usually, the functionality of automata is presented with state functions. To provide systolic algorithms as simple as possible, the function of the PEs is rather presented in the form of pseudo-code procedures [Petkov93].

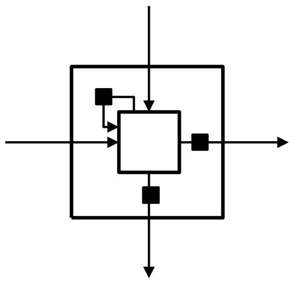


Figure 10: Processing element.

Since delay elements are associated with each interconnection they could be omitted from the presentation. Actually they are hidden within PEs (Figure 10). The decision is influenced by widely adopted presentation of the systolic arrays.

Figure 11 presents a systolic algorithm for the operation $\mathbf{y} = \mathbf{y}^{(0)} + \mathbf{A}\ \mathbf{x}$. The above systolic algorithm specification for matrix-vector multiplication serves as an example of formal representation of systolic algorithms. The systolic algorithm in Figure 11 consists of a systolic array with input data specification and processing element (PE) specification.

The systolic array has linear topology and consists of identical PEs. The components of the vector **x** are input from the left and transferred from PE to PE in a pipelined way. The matrix **A** is input row-wise, where the first row in input into the first PE, the second row is input to the second PE, etc.

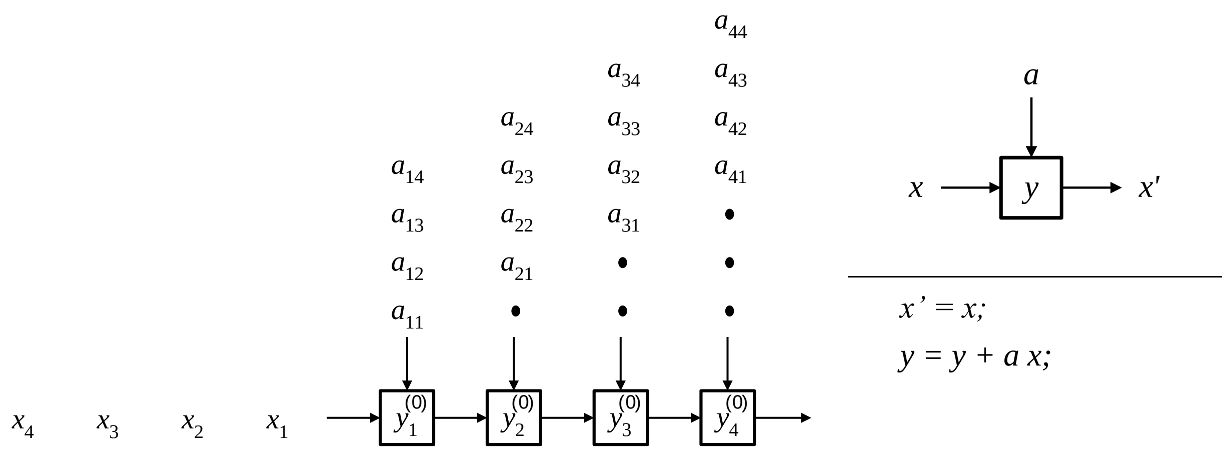


Figure 11: Systolic algorithm for the operation $\mathbf{y} = \mathbf{y}^{(0)} + \mathbf{A}\,\mathbf{x}$.

Neighbouring rows of the matrix **A** are delayed by one clock period, watching from the left to the right, to provide that the first component of each row enters the respective PE together with the first component x_1 of the vector **x**. These delays are represented with black dots in the input data flow [Petkov93].

The function of a PE in the systolic array is specified as a pseudo-code procedure. In any clock period a PE in the systolic array receives a component *x* of the vector **x** and a component *a* of the matrix **A**, multiplies the two values and accumulates the product in a variable *y* which resides in the PE. The component *x* is output to the right unchanged. Initially, the components of the additive vector $\mathbf{y}^{(0)}$ are preloaded in PEs.

In general, there is no need for control. The correct operation of the structure is provided by the coordinated input of data flows. If control is needed, it is realised by an instruction flow, which moves at a constant velocity through the systolic array and is coordinated with the data flows. The instruction flow is considered simply as another data path in the systolic array. If present, the processing element's operation depends on its value. In case that the systolic arrays' operation changes during the operation instruction flow is also added. Instruction flow is handled in the same manner as data. The instruction flow enters the systolic array at the upper left processing element. It than propagates down similarly as data. In case that the velocity of the instruction flow differs from the velocity of data in the systolic array extra delay elements are introduced in the inter-processor connections.

1.6 Mapping an algorithm onto a systolic array

The main issue of mapping an algorithm onto a desired parallel architecture is to explore and exploit the parallelism of a given problem. Usually, parallelism and concurrency are not obvious. Several transformation and design techniques were proposed in the past with the aim to reveal the hidden concurrency in the algorithm [VLSI98a]. Fortes et al. [Fortes88] collected 19 methodologies from different authors, as for example [Moldovan82, Li85, Moldovan86, Rao88, Quinton89] and many new have been proposed since then [KungSY88, Zimmermann96]. Among other issues, these methods differ in the way the original algorithms are represented.

The first systolic arrays were derived in an ad-hoc manner meaning that the topology and the PE assignments were guessed from the description of the algorithm. The development of a systematic design methodology for transforming an algorithm represented in some high level constructs into a systolic architecture specification was one of the important design problems in 80's. The main goal of these approaches is to systematically derive a set of systolic arrays from which the optimal solution can be derived. For example Wan et al. [Wan98] have derived 19 ways for systolic matrix multiplication.

The original algorithm can be represented in the forms of:

- Algebraic expressions: Uniform Recurrence equations (URE) [Quinton89], Regular Iterative Algorithms (RIA) [Rao88], Affine Recurrence Equations (ARE), indexes of nested loops [Moldovan82, Moldovan86].
- Graph based descriptions: signal flow graphs (SFG) [KungSY88].
- High level languages [Lam85].

The two most popular types of algorithm representation are algebraic expressions and graphical descriptions. In algebraic based methods the regularised description given as a set of algebraic expressions, and transformations are applied to these expressions to obtain an implementation. The RIA design methodology for systolic arrays are similar to the methodologies derived by [Moldovan82, Quinton89, Li85]. We could say that the RIA methodology generalises and extends this class of methodology.

A different line of research uses graphical notations to describe an algorithm. Graph-based methods represent an algorithm as a graph, and apply transformations to the graph to render it more suitable for later steps. The regularised graph is then mapped onto an array either directly or through other intermediate representations.

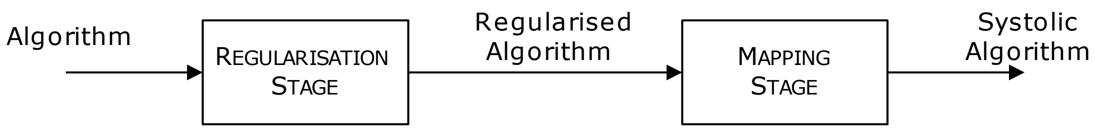


Figure 12: Stages in a transformational design methodology.

The common characteristic of most of these methodologies is the use of transformational approach. Systolic architectures are derived by transforming the original algorithm description that is unsuitable for direct VLSI or systolic implementation. Starting from the representation of an original algorithm, a transformational design methodology consists of algorithm regularisation and array mapping (Figure 12). The regularisation stage corresponds to the derivation of a regularised representation of an algorithm from an original form. The regularised representation describes the algorithm in a manner suitable for manipulation in the remaining steps of a particular method. Consequently, different approaches are characterised by different regularised representations. The mapping stage uses the regularised description to determine the topology and the structure of the systolic array, the characteristics of the PEs, the allocation of data and operations to PE, the dataflow, the I/O, the control and so on [Fortes88].

The above mentioned design methodologies allow systematic synthesis of systolic arrays. Another important issue for systolic array design is the automation of these transformational design methodologies. A number of ongoing attempts to develop synthesis software tools have been made. One most notable attempt among these is ADVIS [Moldovan86].

1.6.1 Notes on mapping of matrix triangularisation algorithms

This section presents basics of mapping matrix triangularisation algorithms, such as Gaussian elimination and Givens rotations, and LU and QR decomposition as well. Later, these methods and the corresponding systolic arrays will take part in systolisation of Faddeev algorithm. The results could be obtained with one of the mapping techniques. Since mapping algorithms onto VLSI processor arrays is not the main issue, only resultant systolic arrays are presented. For details we refer to [KungSY88].

The triangularisation systolic arrays for Gaussian elimination and Givens rotations both utilise the same principle. In both methods we have three data streams from systolic algorithm point of view, namely the input matrix, the resulting triangular matrix, and stream of factors. These factors are named in the case of Gaussian elimination multiplication factors and are denoted as *m* and in the case of Givens rotations rotation factors and denoted as *c* and *s*.

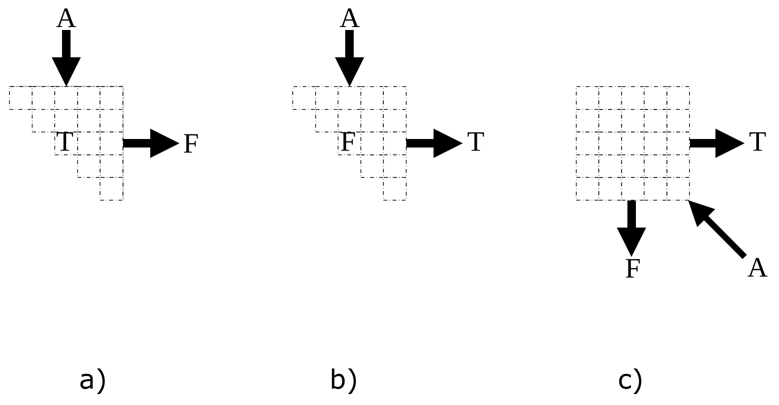


Figure 13: Triangular topologies systolic arrays a) version 1: A and F flow, T resident, b) version 2: A and T flow, F resident, and c) square topology systolic array: all streams flow. The three data streams are input data stream A, the resulting triangular stream T and stream of factors F.

The solutions are classified on the amount of data movements. The systolic solutions differ depending on which data streams are flowing and which are resident in the systolic array. In general, in triangular topology systolic arrays one data stream is resident on the other hand in square topology systolic arrays all data streams flow through the array.

There are two systolic solutions with the triangular topology of the PEs. In the first solution Figure 13a, the resulting triangular matrix is resident in the systolic array at the end. In case the result is needed, data unloading must be performed for which additional execution time is added to the total processing time. The multiplication factors in Gaussian elimination or rotation factors in Givens rotations flow out of the systolic array. In the second solution Figure 13b, the resulting triangular matrix is output at the end. In this case, the multiplication factors in Gaussian elimination or rotation factors in Givens rotations are resident in the systolic array.

The solution with all data streams flowing results in a square systolic array (Figure 13c). As a consequence the resulting triangular matrix is output of the systolic array as well as multiplication or rotation factors, depending on the method.

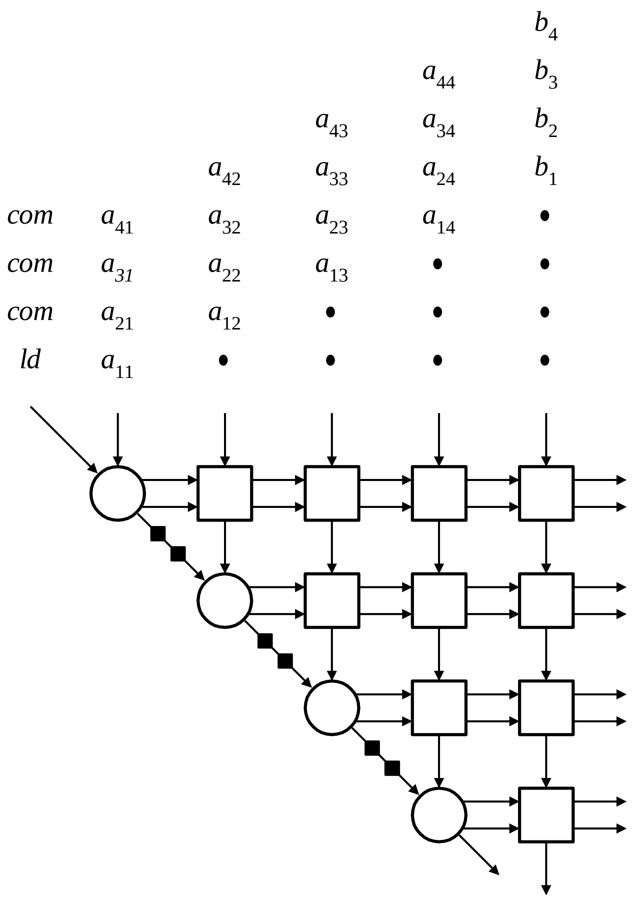
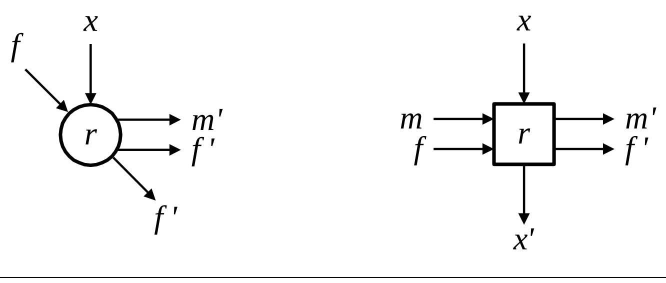

b_4
a_{44} b_3
a_{43} a_{34} b_2
a_{42} a_{33} a_{24} b_1
com a_{41} a_{32} a_{23} a_{14} •
com a_{31} a_{22} a_{13} • •
com a_{21} a_{12} • • •
ld a_{11} • • • •

Figure 14: Systolic array for forward elimination step of Gaussian elimination method.



```
f' = f;
if (f == ld)
    r = x;
if (f == com)
    m = – x / r;
```

```
f' = f; m' = m;
if (f == ld)
    r = x;
if (f == com)
    x' = x + m r;
```

Figure 15: Processing elements for forward elimination step of Gaussian elimination method.

1.7 Schematic representation of systolic algorithms

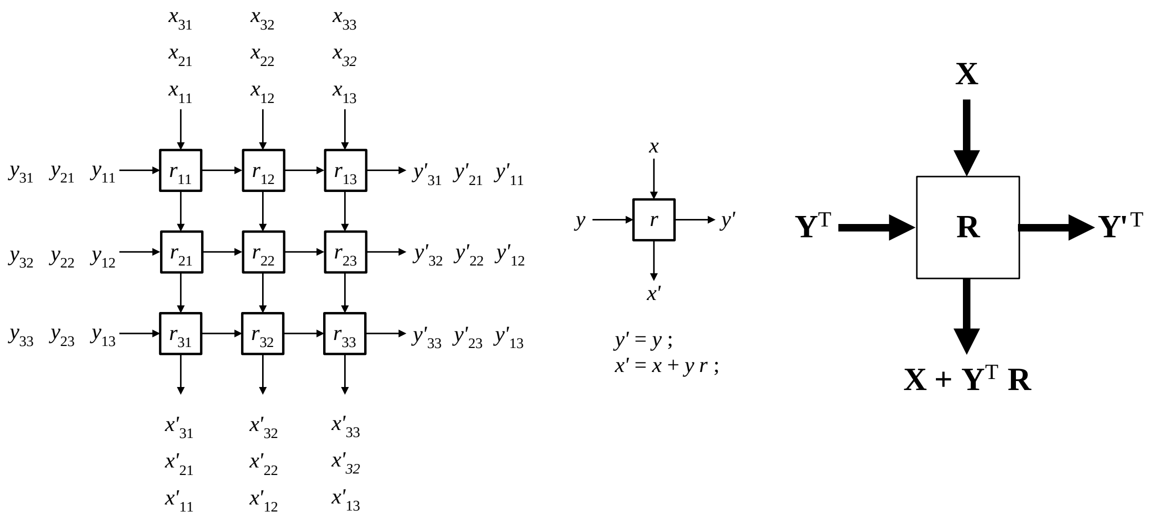
Next we consider a sort of abbreviation of the systolic algorithm description. We call this formulation schematic representation of systolic algorithms. Different authors have successfully used this representation for the description of systolic algorithms. The main purpose is to efficiently represent the computations of the systolic array at the matrix level and not at the matrix coefficient level as it is usual in systolic algorithm description. As a result, the coefficient presentation is substituted with matrices and vectors. Using this approach, scalar operations associated with PEs of the systolic array are generalised in matrix form on the systolic array level.

McWhirter et al. [McWhirter92, Proudler93] extensively used this form of representation of systolic arrays for the development of new systolic algorithms and structures. The procedure is known as *algorithmic engineering* and was extensively used in the area of least squares computation.

Schematic representation provides a framework for describing and manipulating building blocks commonly used to define parallel algorithms and architectures for digital signal processing. Each building block is treated as a mathematical operator with its parallel structure and interconnections represented in terms of Signal Flow Graph (SFG) [KungSY88]. The basic difference between the systolic array and signal flow graph is that the delay elements are omitted. As a result the computations are executed instantly. The results are available with no delay. The main advantage is the absence of detailed timing features, which would be required on systolic array level. In other words, SFG hides the detailed timing features associated with synchronous systolic array but, when required, this information can easily be re-inserted [KungSY88].

The schematic formulation of systolic algorithms will be mainly used for presentation of computations within the systolic arrays. As it will be demonstrated later, with schematic representation intermediate computations become transparent. The intermediate results also reveal different matrix forms obtainable during the operation. This fact widens the set of operations of the systolic array[6]. The knowledge of intermediate computation in the systolic array can also be efficiently used as a verification tool for verifying computations at the output of the systolic array at the end of the operation.

In this section we introduce two basic matrix operators in schematic formulation. Other mathematical operators will be introduced when needed. We start with the systolic array and associated PE assignments. The next step is the derivation of the Signal Flow Graph, which is the foundation for the construction of the fixed mathematical operator. Their function remains constant and is not affected by the data which they process. Within the schematic presentation blocks of different shapes are used. The shape of the blocks is defined by the underlying systolic array's topology. Labels are added to distinguish between the operators of the same shape and different operation.



x_{31} x_{32} x_{33}
x_{21} x_{22} x_{32}
x_{11} x_{12} x_{13}
y_{31} y_{21} y_{11}
r_{11} r_{12} r_{13}
y'_{31} y'_{21} y'_{11}
y_{32} y_{22} y_{12}
r_{21} r_{22} r_{23}
y'_{32} y'_{22} y'_{12}
y_{33} y_{23} y_{13}
r_{31} r_{32} r_{33}
y'_{33} y'_{23} y'_{13}
x'_{31} x'_{32} x'_{33}
x'_{21} x'_{22} x'_{32}
x'_{11} x'_{12} x'_{13}
x
y
r
y'
x'
$y' = y$;
$x' = x + y\,r$;
$\mathbf{X}$
$\mathbf{Y}^T$
$\mathbf{R}$
$\mathbf{Y'}^T$
$\mathbf{X} + \mathbf{Y}^T\,\mathbf{R}$

Figure 16: Derivation of a rectangular fixed matrix operator. a) SFG, b) PE assignments, c) schematic representation.

The input data could be oriented column or row wise regarding the inputs of the systolic array. According to data dependencies we must determine if the data matrices enter directly or transposed. The propagation of the data must meet the requirements of the algorithms. The operation of the mathematical operator is defined by the operation of PEs.

[6] As an example we can consider the computation of the orthogonal matrix **Q**, which is not directly obtainable with systolic array for the QR decomposition.

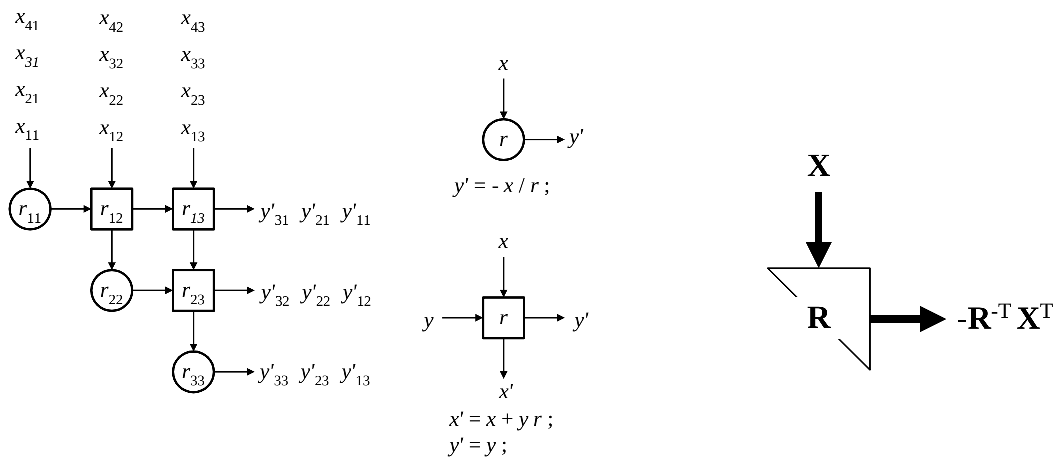


Figure 17: Derivation of a triangular fixed matrix operator. a) SFG, b) PE assignments, c) schematic representation.

1.8 Systolic array performance analysis

There are many performance measures for parallel structures. Table 1 lists some of the widely used performance measures for the systolic arrays [KungSY88]. The table is followed with a brief description of each performance parameter. These parameters will be used for the evaluation and comparison of different systolic solutions.

A	Array size
T	Computation time
α	Pipelining period
β	Block pipelining period
S	Speedup
E	Efficiency of the systolic array
IO	I/O channels
AT^2	Area-Time complexity

Table 1: Basic systolic array performance measures.

Array size (A): Equals the number of the processing elements (PEs) in the systolic array and thus determines the basic hardware cost. It measures requirements of the systolic algorithm on computational resources. On the one hand, the number of processors required by the systolic algorithm presents a measure of cost of the algorithm, i.e. of how expensive this algorithm is to run. On the other hand, it reflects the extend to which the algorithm makes use of the parallelism inherent in the respective computational problem supposing that the processors are used efficiently.

For instance, one can expect that an algorithm, which requires n^2 processors for the multiplication of $(n \times n)$ matrices makes better use of the parallelism inherent in matrix multiplication than an algorithm which needs only n processors. At the same time the former algorithm is more expensive to run than the later.

Computation time (T): The time interval between starting the first computation and finishing the last computation of a problem instance by the processor array is the computation time. In other words, time requirements of the algorithm can be expressed in the number of clock periods needed by a systolic algorithm to solve a problem of a given size. The computation time for a given algorithm can be divided into two terms

$$T = T_c + T_d$$

The first part, denoted T_c, is the time interval between starting the first computation and finishing the last computation of a problem instance by the systolic array. The second part, denoted T_d, is the sum of the time for input data (required by the first computation) to propagate to the appropriate PE from certain boundaries of the systolic array, and the time for the result (generated by the last computation) to propagate to the boundary of the systolic array.

This quantity presents the measure for that how fast an algorithm is in this way, of how well it does in achieving the goal of parallel computing. In general, the clock intervals are counted from the first input operation to the last output operation. Thus, for example, different systolic algorithms for matrix multiplication have different time requirements, e.g. $4n - 1$, $2n$, or n clock periods.

Pipelining period (α): The time interval between two successive computations in a PE is referred to as pipelining period. In other words, the processor is busy for one out of every α time intervals. The particular activity pattern is present in a systolic algorithm and is directly associated with the efficiency of the algorithm as discussed later on. For example, each PE in the systolic array could be active every third clock cycle. Algorithms with this particular activity pattern are also called 3-slow systolic algorithms.

Block pipelining period (*β*): The time interval between the initiations of two successive problem instances by the processor array is block pipelining period. In other words, the block-pipelining period is actually the largest time-span of any PE in the systolic array. Time span indicates the difference between the last time step and the first step while a PE is busy for a given algorithm.

The number of clock periods considered above refers to the time requirements of an algorithm as far as a single problem is concerned. In several cases in which multiple problems of the same type have to be solved one after another using the same systolic algorithm, a smaller number of clock periods is needed per problem. A new problem is started before the previous task has been completed. For a certain period of time operations, which belong to two different problems, are carried out in different parts of the systolic array.

Speedup (*S*): The speedup can be defined as the ratio of systolic arrays' processing time and single processor's processing time

$$S = T_s / T$$

where T_s is the processing time of a single processor for a given algorithm. Clearly, the larger the speedup, the better the parallel algorithm. The processing time of a single processor is equivalent to array active computing time, which is the sum of the active computing time of all PEs of the systolic array. The PE active computing time equals the number of clock periods during which a given PE is active with executing useful computational operations, which contribute to the solution of the problem. This time can be of course different for different PEs in the same systolic array.

In case when M instances are fed into the systolic array to be processed in a pipeline way, the total time required for a single processor is MT_s and the total time required for the systolic array is $T + (M - 1)\beta$. Thus the speedup of the systolic array is defined with

$$S_M = MT_s \ / T + (M - 1)\beta$$

In case that $M \to \infty$ S_M becomes

$$S_\infty = T_s \ / \beta$$

Efficiency (*E*): The efficiency of the systolic array is defined as the ratio of its speedup to the number of PEs in the array

$$E = S / A = T_s / T A$$

This quotient can be used to characterise how efficient an algorithm is using the available computational resources. It is refereed to as algorithm efficiency or as array utilisation.

The efficiency for multiple problem instances can be given by

$$E_M = S_M / A = MT_s / A(T + (M - 1)\beta)$$

In case that $M \to \infty$ the E_M becomes

$$E_\infty = S_\infty / A = T_s / A\beta$$

I/O channels (*IO*): Another important performance measure is the number of I/O lines to communicate with the outside world (the host computer). I/O channels are directly connected to hardware cost. The number of I/O channels can be directly derived from projected systolic array.

Area- Time complexity (AT^2): There are different area-time complexity measures. The AT^2 performance measure combines two factors A and T and is the most useful measure for the processor arrays. It provides hardware cost-effectiveness. In the case of many problem instances to be proceeded (multiple problems), the average time for a single problem instance is approaching to the block-pipelining period β. Thus, the area-time complexity measure becomes $A\beta^2$.

A systolic algorithm for a given problem which requires a smaller number of processors than another alternative systolic algorithm for the same problem will typically need more clock periods to complete both in the single problem mode and for multiple problems. A measure to compare the performance of the algorithm could be the product of the number of processors with the number of clock periods required by a given systolic algorithm.

1.8.1 Relativity of performance measures

The simple quantitative measures introduced above are to be used only for comparison of different systolic algorithms for one and the same problem. Such a comparison is legitimate only if the algorithms to be compared make use of the same or at least very similar PEs. The number of PEs and the number of clock periods can be used as a basis for the comparison of the hardware requirements and the actual processing time of different systolic algorithms only under these conditions.

The meaning of performance measures greatly depends also on a given situation. For example, when we consider a single problem with finite input data, the computation time T is perhaps a more important criterion than the pipelining period β. But when we consider a single problem instance with indefinite input data (filtering and DSP applications) the pipelining period α may become the main performance parameter. On the other hand the block-pipelining period β is of main concern for the cases when several problem instances are to be proceeded by the same systolic array.

1.9 Summary and discussion

This chapter covers the main characteristics of systolic arrays. The systolic arrays, representatives of VLSI processor arrays, were put in the context of parallel computer architectures. Different taxonomies were presented to show this interesting area of computing from different points of view.

The notation of systolic arrays and algorithms was also specified. The systolic algorithms are usually represented with the systolic array, PEs assignment and the corresponding input data stream. The data and computations are given at a coefficient level. Therefore the overall operation and data flow is rather unclear. Schematic representation of systolic arrays presents an attractive approach, which will prove its worth through out the rest of the work.

The mapping of an algorithm on a VLSI processor array was only briefly covered. One of the reasons is that mapping techniques were of great interest in the past. Therefore a large number of systolic arrays are already well known and well understood. From a selected a large set of mapping solutions are obtained from where only a small portion satisfy the strict specifications of systolic arrays. The obtained solutions can be compared according to different performance measures. The systolic arrays used here are all generally accepted and extensively covered in the literature.

2 Matrix-matrix multiplication systolic array algorithm

This chapter presents one of the elementary systolic algorithms: matrix-matrix multiplication systolic algorithm. Simplicity and regularity of this algorithm ease the understanding of systolic parallel processing principles. Furthermore, it is excellent starting point for algorithm engineering. Parallel Hotelling algorithm is used as representative example.

2.1 Matrix multiplication systolic array

Let **A** be $(m \times n)$ matrix and **B** be $(n \times l)$ matrix. The product of the matrices **A** and **B** is a $(m \times l)$ matrix **C**

$$\mathbf{C} = \mathbf{A}\,\mathbf{B} \tag{ 1}$$

or in component form

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}, \quad i = 1:m,\ j = 1:l$$

Usually, a more general operation is realised. Besides computing the matrix product a matrix is also added to it. So (1) is modified as

$$\mathbf{C} = \mathbf{C}^{(0)} + \mathbf{A}\,\mathbf{B} \tag{ 2}$$

or in component form

$$c_{ij} = c_{ij}^{(0)} + \sum_{k=1}^{n} a_{ik}\, b_{kj}, \qquad i = 1:m,\ j = 1:l$$

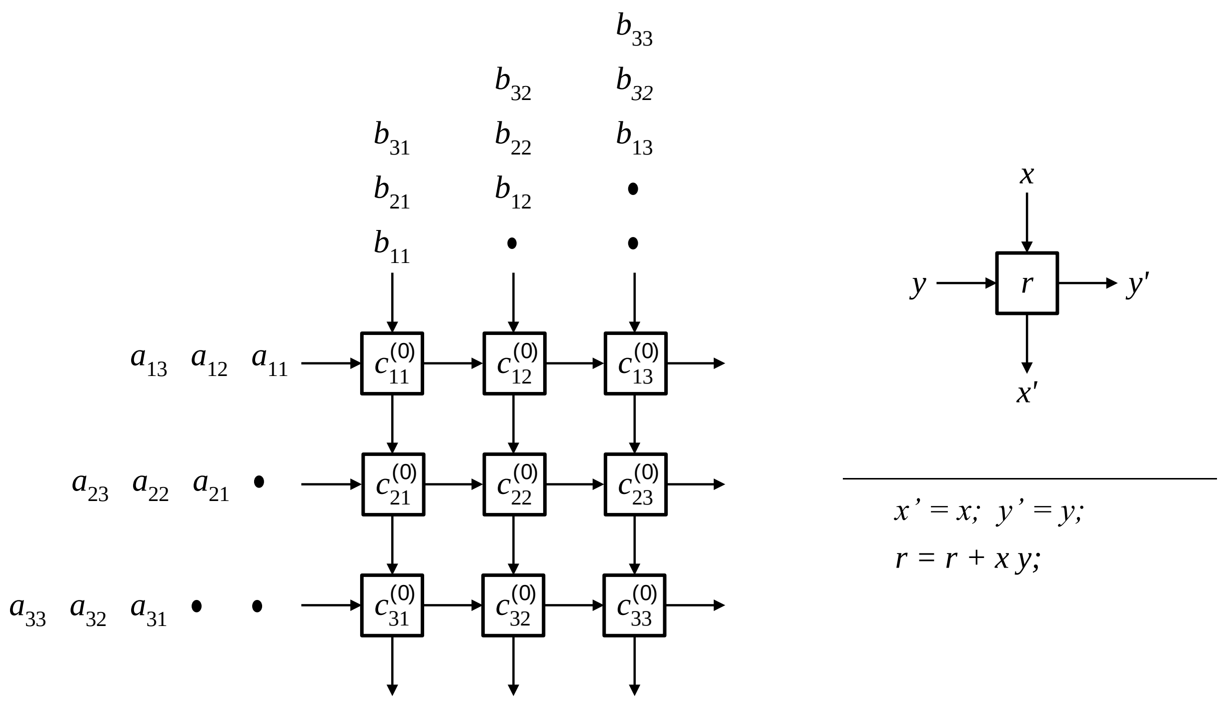
From the above algorithm a variety of systolic array designs can be systematically generated using mapping techniques [Wan98]. These systolic designs can be compared to obtain the best solution under appropriate objective functions or performance measures.

The selected mapping results in an orthogonal mesh topology systolic array where one of the three matrices from (2) is stationary and the other two matrices are passing the systolic array. Thus we have three different cases for which we can define three versions of the selected systolic array. In each of these cases (versions) one matrix is stationary and preloaded in the systolic array before the operation starts. During the operation the other two matrices are passing the array executing the required operations.

2.1.1 Version 1

Systolic array in Figure 18 can compute $\mathbf{C} = \mathbf{C}^{(0)} + \mathbf{A}\,\mathbf{B}$ while **C** is stationary and **A** and **B** are non-stationary. At the matrix level one can observe that the matrix **A** flows through the systolic array from left to the right, while the matrix **B** flows through the systolic array from the top to the bottom. The resultant matrix is stored in the systolic array at the end of computation.

In every time cycle[7], each PE in the systolic array receives input from the left and another from the top, then performs multiplication on the two input operands and accumulation on the product and the previous partial result stored in the PE. Considering PE$_{11}$ of the systolic array shown in Figure 18, we see, that the operation of the PE is used to compute the product of the first row of the matrix **A** with the first column of the matrix **B** and add it to the first element of **C** thus obtaining the first element c_{11} of the result matrix. Passing the first row of the matrix **A** to the left it can be used in the PE$_{12}$ to compute its product with the second column of the matrix **B** and to add it to the element of $c_{12}^{(0)}$ obtaining c_{12}.



b_{33}
b_{32} b_{32}
b_{31} b_{22} b_{13}
b_{21} b_{12} •
b_{11} • •
a_{13} a_{12} a_{11}
a_{23} a_{22} a_{21} •
a_{33} a_{32} a_{31} • •
$c_{11}^{(0)}$ $c_{12}^{(0)}$ $c_{13}^{(0)}$
$c_{21}^{(0)}$ $c_{22}^{(0)}$ $c_{23}^{(0)}$
$c_{31}^{(0)}$ $c_{32}^{(0)}$ $c_{33}^{(0)}$
x
y r y'
x'
$x' = x;\ y' = y;$
$r = r + x\,y;$

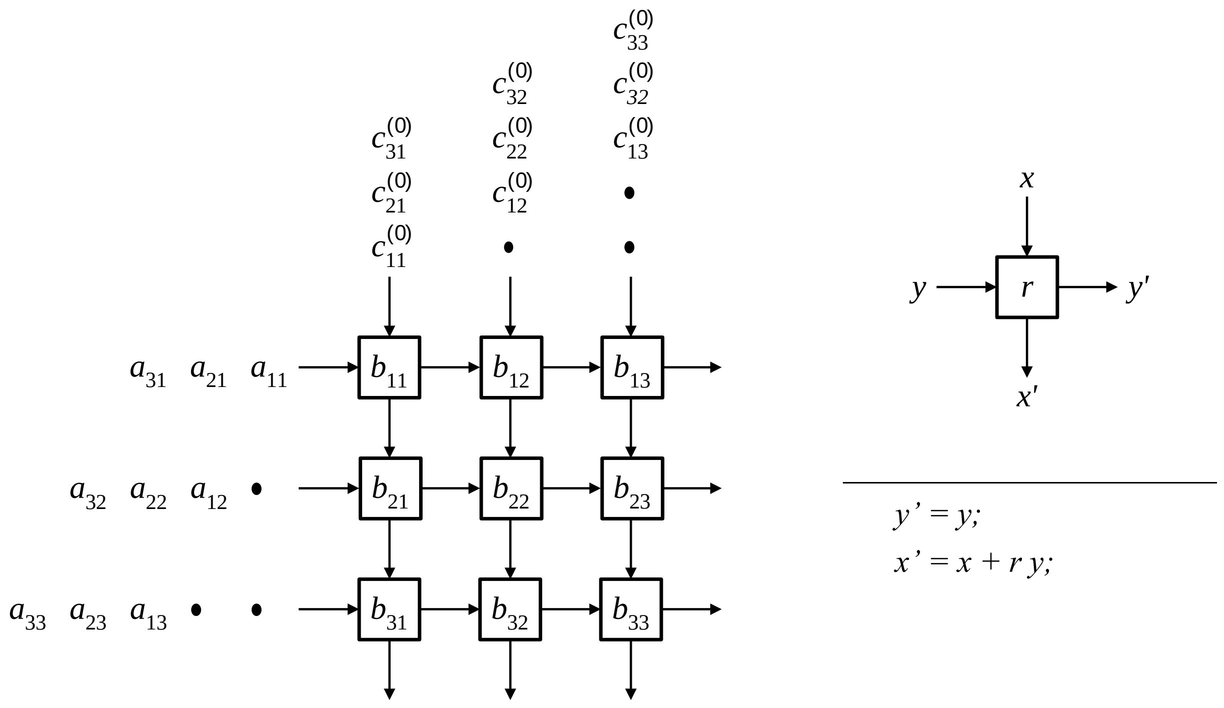
Figure 18: Systolic algorithm for matrix multiplication version 1. Matrix **A** is input row-wise, matrix **B** is input column-wise, matrix $\mathbf{C}^{(0)}$ is preloaded, and matrix **C** is accumulated in the systolic array by replacing the matrix $\mathbf{C}^{(0)}$. $n = m = l = 3$.

[7] Systolic cycle.

2.1.2 Version 2

Systolic array in Figure 19 can also compute $\mathbf{C} = \mathbf{C}^{(0)} + \mathbf{A}\mathbf{B}$ while **B** is stationary and **A** and **C** are non-stationary. It is evident that the systolic array differs from the systolic array in Figure 18 only in data positions. In this case matrix **B** is preloaded in the systolic array. The matrix **A** in transposition flows through the systolic array from left to the right. The initial matrix $\mathbf{C}^{(0)}$ enters the systolic array form the top, then flows through the systolic array downwards, and finally the resultant matrix **C** comes out of the systolic array at the bottom.

In every time cycle, each PE receives two inputs from the left and the top, then performs multiplication and accumulation, and finally the result is sent to its downward neighbour.



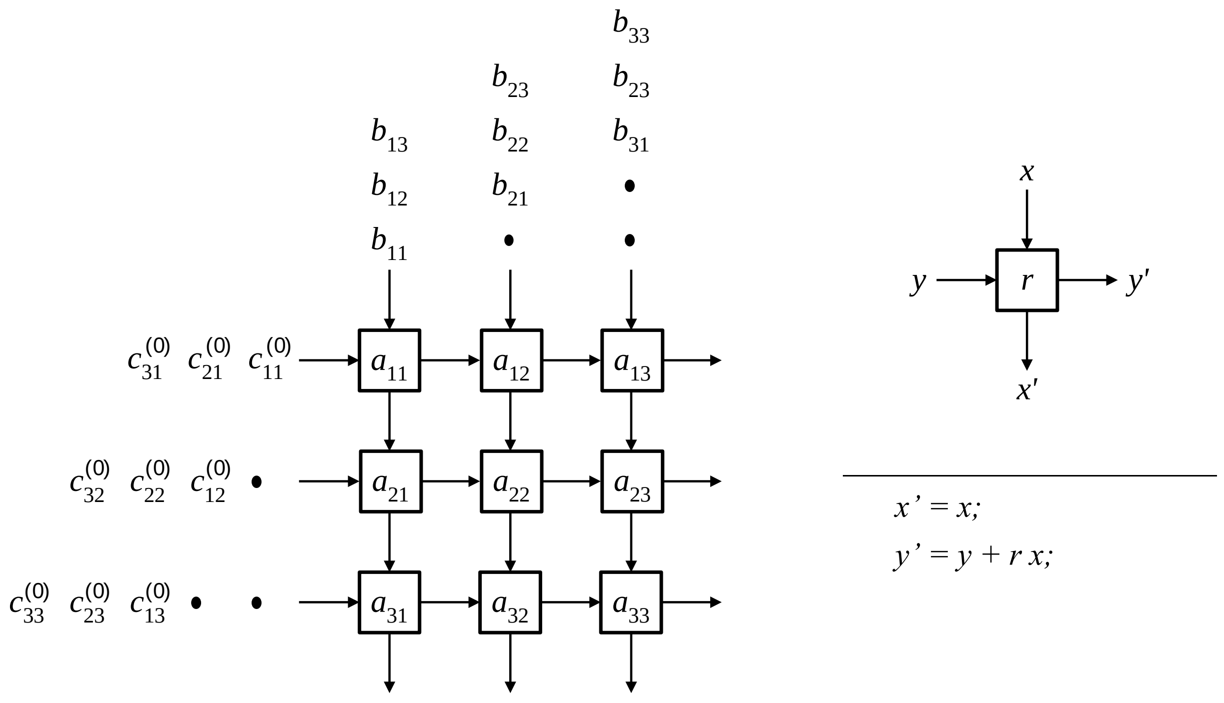
$c_{33}^{(0)}$
$c_{32}^{(0)}$ $c_{32}^{(0)}$
$c_{31}^{(0)}$ $c_{22}^{(0)}$ $c_{13}^{(0)}$
$c_{21}^{(0)}$ $c_{12}^{(0)}$ •
$c_{11}^{(0)}$ • •
a_{31} a_{21} a_{11}
a_{32} a_{22} a_{12} •
a_{33} a_{23} a_{13} • •
b_{11} b_{12} b_{13}
b_{21} b_{22} b_{23}
b_{31} b_{32} b_{33}
x
y
r
y'
x'
$y' = y;$
$x' = x + r\,y;$

Figure 19: Systolic algorithm for matrix multiplication version 2. $n = m = l = 3$.

2.1.3 Version 3

Systolic array in Figure 20 can also compute $\mathbf{C} = \mathbf{C}^{(0)} + \mathbf{A}\mathbf{B}$ while **A** is stationary and **B** and **C** are non-stationary. This is an alternative to version 2 systolic array (Figure 19). It differs only in data positions. Matrix **A** is preloaded in the systolic array. The matrix **B** in transposition flows through the systolic array from the left to the right. The initial matrix $\mathbf{C}^{(0)}$ enters the systolic array from the top, then flows through the systolic array downwards, and finally the resultant matrix **C** comes out of the systolic array at the bottom.

Each PE receives two inputs from the left and the top, then performs multiplication and accumulation, and finally the result is sent to its downward neighbour.



b_{33}
b_{23} b_{23}
b_{13} b_{22} b_{31}
b_{12} b_{21}
b_{11}
$c_{31}^{(0)}$ $c_{21}^{(0)}$ $c_{11}^{(0)}$
$c_{32}^{(0)}$ $c_{22}^{(0)}$ $c_{12}^{(0)}$
$c_{33}^{(0)}$ $c_{23}^{(0)}$ $c_{13}^{(0)}$
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
x
y
r
y'
x'
$x' = x;$
$y' = y + r\,x;$

Figure 20: Systolic algorithm for matrix multiplication version 3. $n = m = l = 3$.

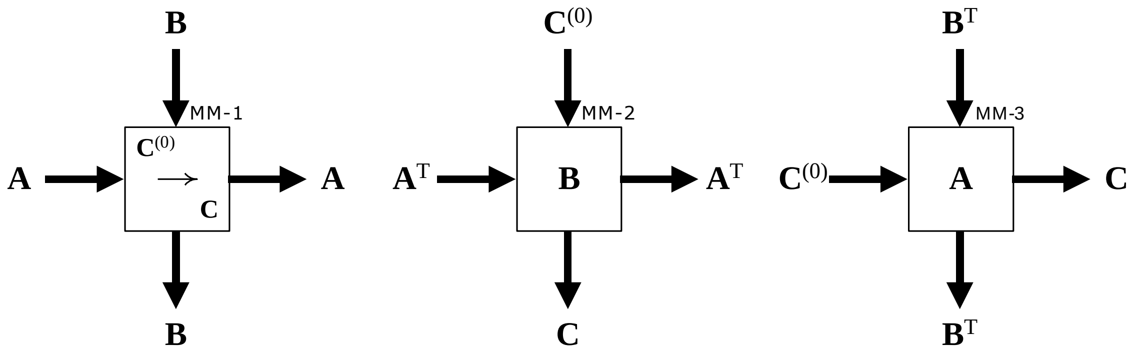


Figure 21: The three versions of matrix multiplication systolic array using schematic representation. Versions are specified with a label.

Figure 21 schematically presents the three versions of matrix multiplication systolic array. The description at matrix level clearly defines proper data orientation. The operation is defined with (2). The derivation of such blocks, treated as mathematical operators, was discussed in 1.7. Since blocks are based on Signal Flow Graphs (SFGs) (1.7) the computations are executed instantly. The three blocks are used as building blocks for the derivation of compound operations presented in sequel. Firstly, triple matrix multiplication is considered, followed with matrix inverse and Schur complement computation.

In Chapter 0 different performance parameters were defined. Performance parameters are collected in Table 2 for the three versions of matrix multiplication systolic array. The parameters are equal for all three versions due to their evident similarity.

Performance parameters	**Matrix multiplication: MM-1, MM-2, and MM-3**
Array size	$A = n^2$
Computation time	$T_c = 3n - 1 \quad T_d = n$
Pipelining period	$\alpha = 1$
Block pipelining period	$\beta = n$
Speedup	$S = n^2/4 \quad S_\infty = n^2$
Efficiency	$E = 1/4 \quad E_\infty = 1$
I/O channels	$IO = 4n$
Area-Time complexity	$A\beta^2 = n^4$

Table 2: Performance parameters for the selected matrix multiplication systolic array.

With these performance measures one can find the best topology of a systolic array for the specific application. The selection was based on different performance parameters, mainly on area and time parameters, as well as on particular property of the selected systolic array where one of the three matrices is stationary. Note, that the selected three versions are part of nineteen different ways of systolic matrix multiplication derived by Wan et al. [Wan98].

2.2 Triple matrix product computation

An important and computationally demanding operation is a triple matrix product frequently encountered in DSP and IP problems. For example, two-dimensional transforms can be represented by triple matrix products [Pratt78].

The computation of the triple matrix product

$$\mathbf{Y} = \mathbf{A\ X\ B} \tag{3}$$

is decomposed into two matrix-matrix multiplication operations

$$\mathbf{Z} = \mathbf{X\ B}$$
$$\mathbf{Y} = \mathbf{A\ Z} \tag{4}$$

The formulation (4) is appropriate for execution on the matrix multiplication systolic array presented in the previous section. Note, that the systolic array must be used twice before the result of triple matrix multiplication (3) is available.

In order to minimise the I/O operations the following approach seems appropriate exploring the properties of the three versions of matrix multiplication systolic array. Instead of using one of the above versions of the matrix multiplication systolic arrays twice systolic arrays version 1 and version 2 could be combined together. The advantage is presented next. If version 1 systolic array (Figure 18) is used twice, the resultant matrix is stored in the systolic array at the end of computations. On the other hand, if version 2 systolic array (Figure 19) is used twice the pre-loading operation is needed. The advantage of the combined approach (version 1 and version 2 systolic arrays) is that no pre-loading operation is needed at the beginning if $\mathbf{Z}^{(0)}$ is zero and no downloading is needed at the end since the resultant matrix is available at the output of the systolic array. This is best presented in schematic form (Figure 22) where data follows (4).

The PEs in the combined systolic array have two distinct modes, where each mode of operation defines a particular version of the matrix multiplication systolic array. In the first mode, the PE is capable of storing partial sums, while in second mode it retains the constant.

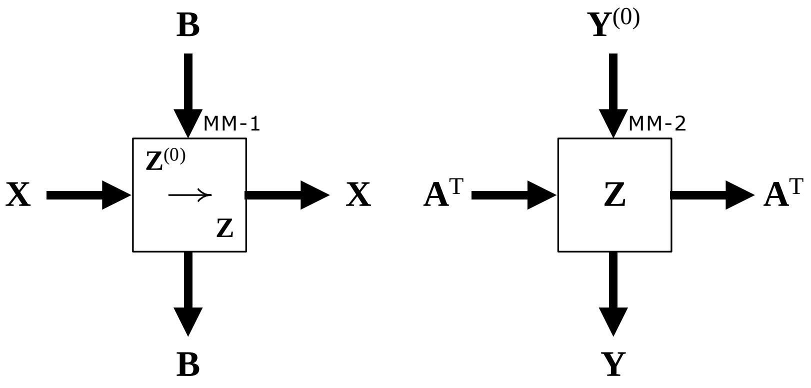


Figure 22: Utilisation of the matrix multiplication systolic array for triple matrix product computation.

2.3 Hotelling method for matrix inversion

The computational load for finding the inversion of a matrix is enormous. This makes this operation interesting for the systolic solution. Matrix inversion can be computed either by a direct method or by an iterative method. Direct methods compute the inversion by elimination methods (Gauss elimination) or matrix decomposition (LU, QR). The iterative methods find the approximate matrix inversion iteratively. Usually, several iterations are required before the final result is obtained. The computation starts with an initial approximation. The result converges toward the solution during iterations [Evans94].

2.3.1 Newton iteration method

A commonly used root finding algorithm is Newton's method [Kreyzsig93]. A continuously differentiable function $f(x)$ is given together with an initial approximation x_0 for one of its roots *z*. The method computes

$$x_{n+1} = x_n - f(x_n)/f'(x_n) \qquad n = 0, 1, 2, \ldots \tag{5}$$

until $|x_{n+1} - x_n| < e$. Here $f'(x)$ is the derivative of $f(x)$ and e is the desired accuracy. A geometric interpretation of Newton's method is shown in Figure 23.

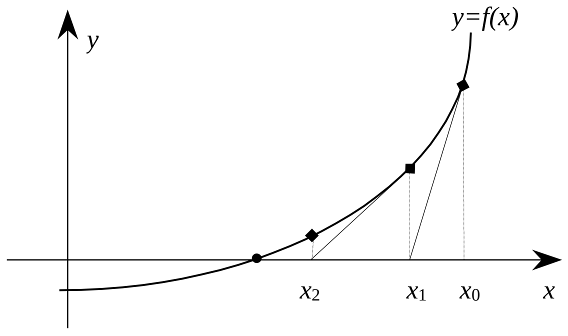


Figure 23: Newton's method for finding a root. The next approximation x_{n+1} is the intersection with the *x* axis of the tangent to the curve *f(x)* at x_n [Kreyzsig93].

The main reason for method's popularity is its rapid convergence when x_0 is sufficiently close to *z*. More precisely, if $f(x)$ and its first and second derivatives $f'(x)$ and $f''(x)$ respectively, are continuous on an interval containing a root *z*, with $f'(x) \neq 0$, and $|x_0 - z| < 1$, then for large *n*, $|x_{n+1} - z| = k|x_n - z|^2$, where *k* is a constant of proportionality that depends on $f'(x)$ and $f''(x)$. In other words, the error in x_{n+1} is proportional to the square of the error in x_n. The method has a quadratic convergence under the stated conditions. In practice, this means that the number of correct digits in the answer doubles within each iteration.

Different matrix algebra problems can be solved iteratively using Newton's iteration, for example square root of a matrix can be computed. With proper substitution of the function $f(x)$, matrix inverse can also be computed. Newton's method modified for the matrix inverse computation is known as Hotelling method [Higham95, Evans94].

2.3.2 *Hotelling method*

Let **A** denote and $(n \times n)$ matrix to be inverted and $\mathbf{X}_k$ denote the *k*-th approximation to $\mathbf{A}^{-1}$. Then we can compute the (*k*+1)-th approximation according to

$$\mathbf{X}_{k+1} = \mathbf{X}_k\,(2\mathbf{I} - \mathbf{A}\mathbf{X}_k) = \mathbf{X}_k\,(\mathbf{I} + \mathbf{R}_k) \tag{6}$$

where **I** is an $(n \times n)$ unit matrix. Equation (6) presents the principle of Hotelling method for matrix inversion. It can be derived from Newton's method by modifying the expression (5)

$$f(\mathbf{X}_k) = \mathbf{X}_k^{-1}\,\mathbf{A}$$

$$f'(\mathbf{X}_k) = -\ \mathbf{X}_k^{-2} \qquad (7)$$

The residual matrix $\mathbf{R}_k$ in (6) measures how far is $\mathbf{X}_k$ from $\mathbf{A}^{-1}$

$$\mathbf{R}_k = (\mathbf{I} - \mathbf{A}\,\mathbf{X}_k) \qquad (8)$$

It can be shown that

$$\begin{aligned}\mathbf{R}_{k+1} &= \mathbf{I} - \mathbf{A}\,\mathbf{X}_{k+1} \\ &= \mathbf{I} - \mathbf{A}\,(2\,\mathbf{X}_k - \mathbf{X}_k\,\mathbf{A}\,\mathbf{X}_k) \\ &= (\mathbf{I} - \mathbf{A}\,\mathbf{X}_{k+1})^2 \\ &= \mathbf{R}_k^2\end{aligned}$$

This means that $\mathbf{R}_k^2 = \mathbf{R}_0^{2^k}$, and thus $\mathbf{R}_k$ converges very rapidly to zero provided that $\mathbf{X}_0$ is a reasonably good initial approximation to $\mathbf{A}^{-1}$ [Leighton92].

This process converges only if $\mathbf{X}_0$ is chosen so that every eigenvalue of $\mathbf{I} - \mathbf{A}\mathbf{X}_0$ is of absolute value less than 1, where $\mathbf{X}_0$ is initial approximation to the matrix inverse [Kreyzsig93]. In addition, if the initial approximation is chosen that $||\mathbf{I} - \mathbf{A}\mathbf{X}_k|| < 1$, the correct number of decimal places increases in geometrical progression [Faddeeva59].

```
$k = 0$;
$\mathbf{R}_0 = \mathbf{I} - \mathbf{A}\,\mathbf{X}_0$;
while $||\mathbf{R}_k||_F < e$
    $k = k + 1$;
    $\mathbf{R}_k = \mathbf{I} - \mathbf{A}\,\mathbf{X}_k$;
    $\mathbf{X}_{k+1} = \mathbf{X}_k\,(\mathbf{I} + \mathbf{R}_k)$;
end
```

Table 3: Algorithm for matrix inversion by Hotelling method. $\mathbf{X}_0$ is a given initial approximation to $\mathbf{A}^{-1}$ and e is a given small positive number for convergence test.

Table 3 sums up the Hotelling iterative method for matrix inverse computation in an algorithmic form [Evans94]. The body of the loop computes the iteration of the Hotelling method accordingly to (6). If the condition in the *while* statement is true, another iteration is computed. Otherwise the result has acceptable precision and the computation stops. The Frobenius norm of the residual matrix is used as a measure how far is $\mathbf{X}_k$ from $\mathbf{A}^{-1}$.

2.3.3 Systolic array for matrix inversion by Hotelling method

The systolic array implementation of Hotelling method is considered next. The first operation of the algorithm in Table 3 is of the form

$$\mathbf{R}_k = \mathbf{I} - \mathbf{A}\,\mathbf{X}_k \tag{9}$$

The systolic array version 1 from Figure 18 can be used to perform this operation. identity matrix **I** is pre-loaded in the systolic array at the beginning of the operation An. The matrices –**A** and **X** enter the systolic array from the left and the top respectively. At the end of the computation the result is stored in the systolic array.

The second matrix multiplication operation in the algorithm from Table 3 is of form

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{X}_k\,\mathbf{R}_k \tag{10}$$

The systolic array MM-2 from Figure 19 can be employed to perform this task. At the start matrix $\mathbf{R}_k$ must be pre-loaded in the systolic array. The matrices $\mathbf{X}_k^{\mathrm{T}}$ and $\mathbf{X}_k$ enter the systolic array from the left and the top respectively. After the completion of computations one iteration is computed and the resultant matrix $\mathbf{X}_{k+1}$ flows out of the systolic array at the bottom.

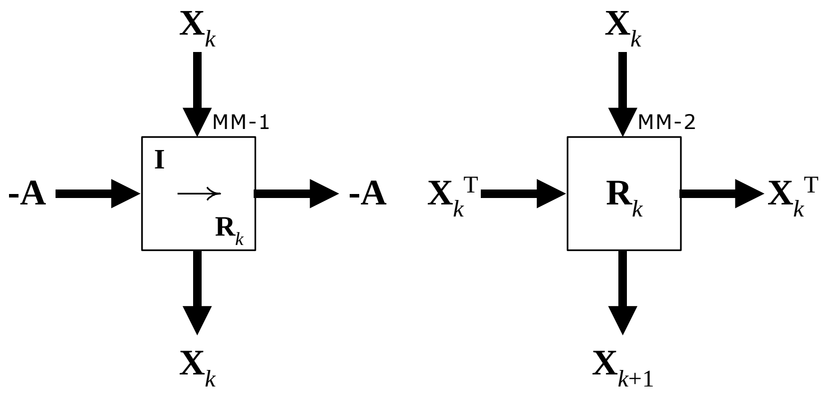


Figure 24: Utilisation of systolic arrays MM-1 and MM-2 for one iteration of Hotelling method.

Figure 24 shows one iteration of Hotelling method for matrix inversion executed by the two systolic arrays for matrix multiplication. It is straightforward to perform more iterations by the systolic array since the data flow is continuous between adjacent iterations.

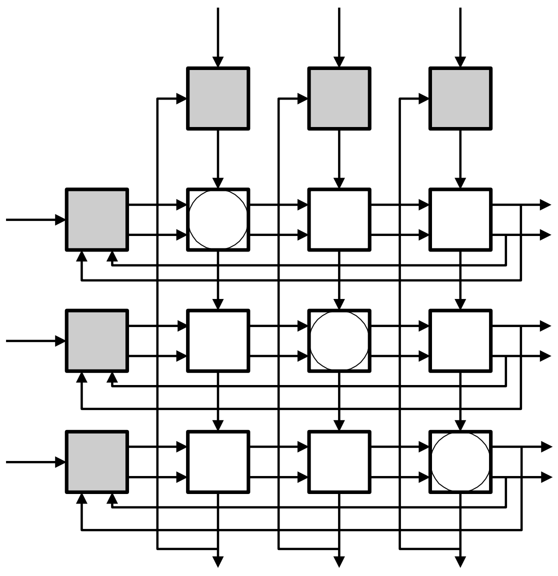
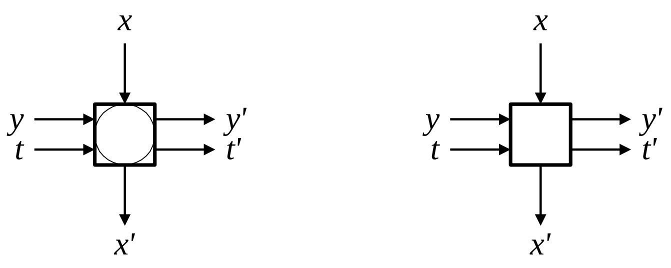


Figure 25: Systolic array with data feedback connections.

Next we consider the overall architecture for the computation of Hotelling method. The fact that the resultant matrix $\mathbf{R}_k$ is stored in the systolic array after the first step and required to be preloaded at the beginning of the second step suggests the use of a single systolic array to perform the tasks from Figure 24.

The feedback connections are added to feed the data from the outputs of the systolic array back to the inputs. According to algorithm requirements in Table 3 data feedback connections are needed in order to feed the outputs of the first computation back to the inputs for the second computation. The grey cells perform simple data interface tasks by sending the input data to the systolic array and feeding the output back to the systolic array.

There are two types of PEs in the systolic array with two different modes of operation. The two modes of operation correspond to the two versions of matrix multiplication systolic array, namely version 1 and version 2. Therefore in Mode 1 the PEs store the result of the multiply-accumulate operation and in Mode 2 the PEs end the result to the bottom neighbour. The command *initial* is needed to pre-store an identity matrix in the PEs as required by the algorithm in Table 3. The internal register of the diagonal PEs is initialised to 1 and the internal register of the non-diagonal PEs is initialised to 0.



MODE 1	$y' = y;\ t' = x;\ x' = x;$ if *initial* $\quad r = 1;$ else $\quad r = r + y\,x;$	$y' = y;\ t' = t;\ x' = x;$ if *initial* $\quad r = 0;$ else $\quad r = r + y\,x;$
MODE 2	$x' = x + t\,r;$ $y' = y;\ t' = t;$	$x' = x + t\,r;$ $y' = y;\ t' = t;$

Figure 26: PE definitions for Hotelling method.

Note that during Mode 1 transposition of the matrix $\mathbf{X}_k$ is required. The diagonal PEs in the array must perform an additional task, matrix transposition. The vertical data flow x is redirected into the horizontal data flow t. The details on systolic array for a matrix transposition are presented next.

Matrix transposition systolic array

Matrix transposition is for example needed to feed the properly oriented data to inputs of the systolic array. In the context of systolic arrays, the matrix transposition is a redirection of data from vertical direction to horizontal direction.

The transpose of $\mathbf{A}^{\mathrm{T}}$ of an $(m \times n)$ matrix $\mathbf{A}$ is the $(n \times m)$ matrix that has the first row of $\mathbf{A}$ as its first column, the second row of $\mathbf{A}$ as its second column and so on [Kreyzsig93]. Thus the transpose of $\mathbf{A}$ is defined as

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \\ a_{m1} & & & a_{mn} \end{bmatrix} \xrightarrow{\mathbf{T}} \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & & \\ \vdots & & \ddots & \\ a_{1n} & & & a_{mn} \end{bmatrix} \tag{11}$$

Figure 27 depicts a triangular systolic array for matrix transposition where n = 3. Matrix **A** enters the systolic array column-wise from the top of the array. The transposed matrix flows out of the array at the right-hand side.

The circular PE receives the input from the top and redirects it to the right. The rectangular PE receives two inputs form the left and the top, and simply passes them to the right and the bottom, respectively.

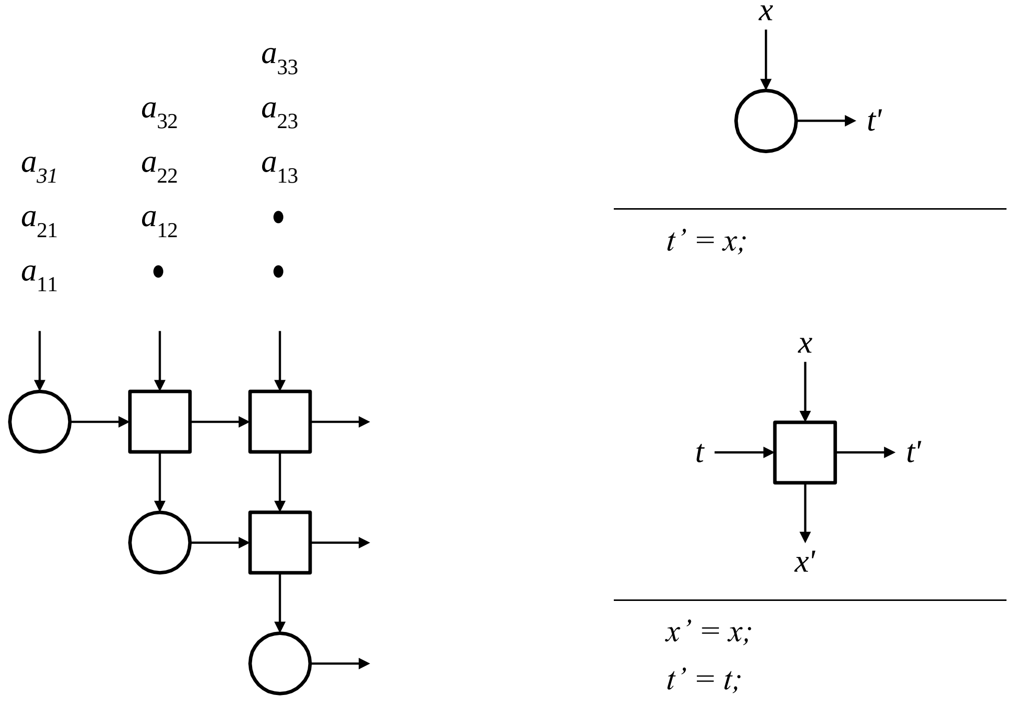


Figure 27: Systolic algorithm for matrix transposition. PE definitions: a) operation: redirection, b) operation: pass through.

2.3.4 Other issues related to matrix inverse computation

Initial approximation computation

The convergence rate and thus performance of the Hotelling method greatly depends on the selection of the initial approximation $\mathbf{X}_0$. A good approximation for a general matrix is [Leighton92, Kreyszig93]

$$\mathbf{X}_0 = \frac{1}{m}\mathbf{A}^{\mathrm{T}}$$

(12)

where m is the trace of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$

$$m = \left\|\mathbf{A}^{\mathrm{T}}\mathbf{A}\right\|_1$$

The calculation of the initial approximation can be simplified by exploring the properties of the matrix **A**. Next some special cases are considered with a simplified version of initial approximation from (12).

Initial approximation computation for some special matrices

For special matrices the initial approximation computation (12) can be modified in order to provide better initial approximation and therefore reduce the number of needed iterations [Evans94, Wan96].

- *Symmetric positive definite case*. For symmetric positive definite matrix **A** the initial approximation can be chosen as

$$\mathbf{X}_0 = \frac{1}{m}\mathbf{I}, \; m = \left\|\mathbf{A}\right\|_1$$

(13)

- *Non-singular case*. The inversion of an arbitrary non-singular matrix **A** can be replaced by the inversion of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, which is always positive definite. A general alternative choice to (12) can be

$$\mathbf{X}_0 = \frac{1}{m}\mathbf{I}, \; m = \left\|\mathbf{A}^{\mathrm{T}}\mathbf{A}\right\|_1$$

(14)

- *Diagonally dominant case*. For a diagonally dominant matrix **A** the initial approximation can be chosen as

$$\mathbf{X}_0 = \mathrm{diag}(a_{11}^{-1}, a_{22}^{-1}, \ldots, a_{nn}^{-1})$$

(15)

Convergence test

In practice we need to know when the iteration should stop. In order to know when the iteration can stop we need to know how far $\mathbf{X}_k$ is from $\mathbf{A}^{-1}$. In Table 3 the convergence test is accomplished by the condition in the *while* statement. According to Table 3 Frobenius norm of the matrix **R**

$$\left\|\mathbf{R}_k\right\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij}{}^2} \tag{16}$$

is used to generate a signal to control the iteration process in the systolic array. This signal is used to stop the computation when the required precision is obtained. The square-root computation can be omitted using a square of the matrix norm as a stopping criterion [Evans94].

Systolic array for Hotelling method with convergence test

In the following section the extension of the systolic array from Figure 25 is presented. The systolic array for Hotelling method with convergence test is presented in Figure 28. From Table 3 we observe that the systolic array must compute $||\mathbf{R}_k||_F$ of the residual matrix $\mathbf{R}_k$. The systolic array for matrix Frobenius norm computation can be obtained in a straightforward manner. From Figure 24 we observe that the elements of the matrix $\mathbf{R}_k$ are stored in PEs. According to (16) squared elements of matrix $\mathbf{R}_k$ must be summed. Therefore each PE must compute r^2 and add it to the incoming value propagated along the additional vertical interconnection from its above neighbour. As a result, at the bottom of the systolic array the squared sums of the columns of the matrix $\mathbf{R}_k$ are output. These values must be then summed in order to obtain the square of (16).

Figure 28 presents a modified systolic array implementing the convergence test. It is evident that adding the convergent test increases the complexity of the systolic array. Extra vertical connections and a row of extra PEs (rhombic) at the bottom are added for the computation of the Frobenius norm of the residual matrix.

The operation of PEs in the systolic array changes in Mode 2 (Figure 29) when the square and the circular PEs in the array perform another multiplication in order to obtain r^2 and add it to z. The result is then output downwards as z'. The additional linear systolic array at the bottom, composed of rhombic PEs sums the vertical data flow. The output of the right most rhombic PE gives the square of the Frobenius norm of the residual matrix. This quantity can be used to generate a control signal to stop the computation when the solution with the required precision is obtained.

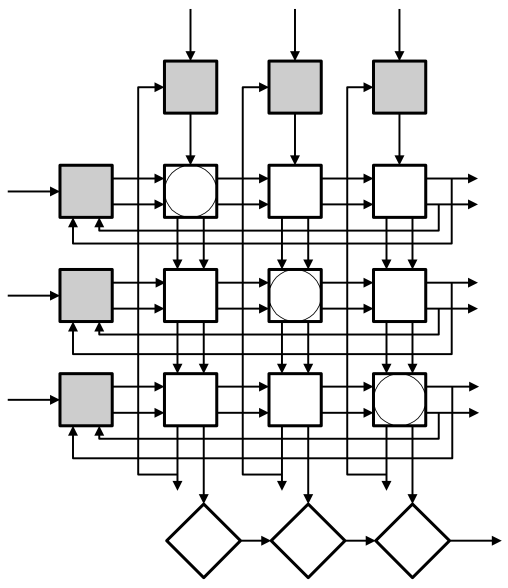
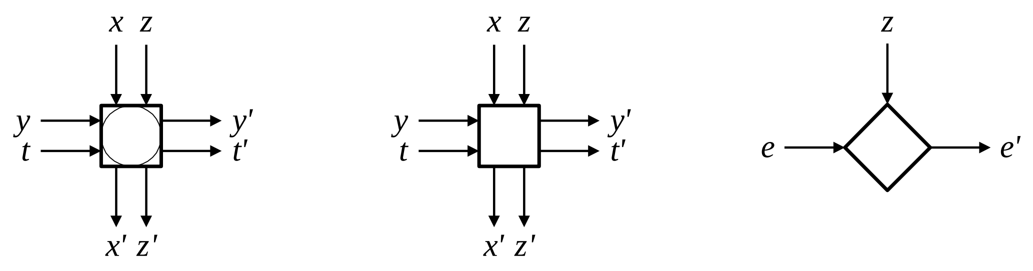


Figure 28: Systolic array with added convergence test.



MODE 1	$y' = y$; $t' = x$; $x' = x$; if *initial* $r = 1$; else $r = r + y\,x$;	$y' = y$; $t' = t$; $x' = x$; if *initial* $r = 0$; else $r = r + y\,x$;	*NoOp*
MODE 2	$x' = x + t\,r$; $y' = y$; $t' = t$; $z' = z + r^2$;	$x' = x + t\,r$; $y' = y$; $t' = t$; $z' = z + r^2$;	$e' = e + z$
MODE 3	$r = r + y\,x$; $y' = y$; $t' = t$; $x' = x$;	$r = r + y\,x$; $y' = y$; $t' = t$; $x' = x$;	*NoOp*
MODE 4	$y' = y + x\,r$; $t' = t$; $x' = x$;	$y' = y + x\,r$; $t' = t$; $x' = x$;	*NoOp*

Figure 29: PE definitions for alternative Schur complement computation. Mode 1 and Mode 2 contain definitions for Hotelling method with added convergence test. Mode 3 and Mode 4 contain definitions for triple matrix product.

Two additional modes of operation are also added in Figure 29. Mode 3 and Mode 4 are needed for alternative Schur complement computation. During these two modes of operation a triple matrix product is formed. The details on Schur complement computation are presented next.

2.3.5 Schur complement computation

In order to obtain Schur complement with the matrix multiplication based systolic structure the form needs to be decomposed into the following basic operations:

- Computation of matrix inverse with Hotelling iterative method.
- Computation of triple matrix product using the result of the first step.

The computation of the above operations with matrix multiplication systolic array was already presented.

Figure 30 presents the basic steps needed to form Schur complement in a schematic representation. The three blocks for the three versions of the matrix multiplication systolic array, presented in Figure 21, are used as building blocks. The first iteration of matrix inversion algorithm is presented in the first row. The second row covers the computation of the triple matrix product that completes the computation after which Schur complement is available. Note how operation of the blocks is changing from MM-1, MM-2, MM-1 to MM-3.

Figure 30 presents only *k*-th iteration of Hotelling method. The computation of additional iteration would be carried out in a similar manner with the increased iteration index since data flow is continuous between the adjacent iterations. The iteration starts with the initial approximation $\mathbf{X}_0$ whose computation is covered in 2.3.4. The schematic representation enables us to easily investigate different possibilities and verify the correctness of the selected solution.

The systolic array for alternative Schur complement computation is equivalent to systolic array in Figure 28 since this systolic array combines all the needed features. The difference is in PE definitions in Figure 29. For alternative Schur complement computation two additional modes of operation are added. The two modes, namely Mode 3 and Mode 4, are responsible for triple matrix multiplication computation.

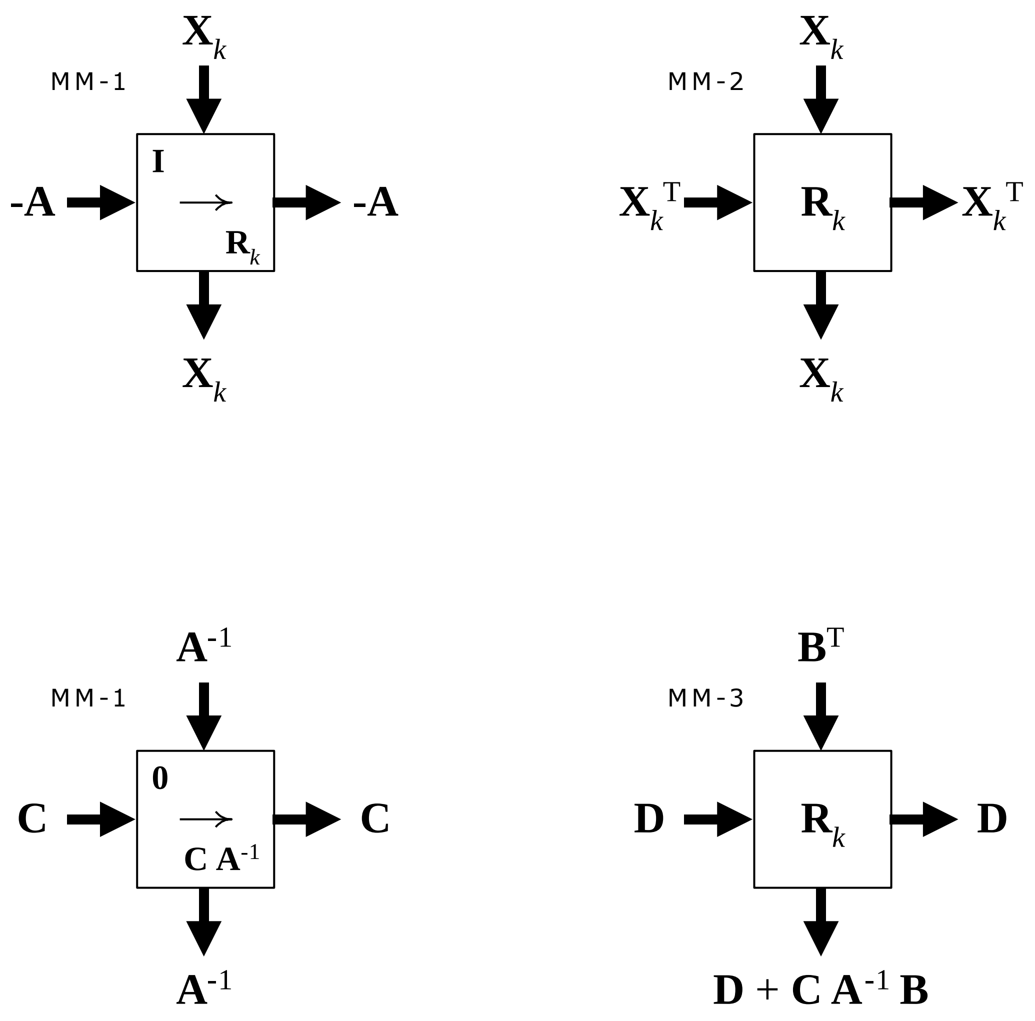


Figure 30: Utilisation of matrix multiplication systolic array for Schur complement computation. The first row represents one iteration of matrix inversion computation. The second row shows triple matrix product computation. X_{k+1} is substituted with A^{-1} in the third block.

2.3.6 Numerical example

The following numerical example illustrates the convergence of Hotelling iterative method. The aim is to present some of the issues discussed in this section. For this example five randomly generated matrices were selected as test matrices $(\mathbf{A}_1,\ \mathbf{A}_2,\ \ldots,\ \mathbf{A}_5)$. The five matrices and the corresponding initial approximations were generated in Matlab with the following commands

```
>> A=randn(3);

>> Xo=A'/trace(A'*A);
```

Computation of the initial approximations ($\mathbf{X}_0$) agrees with (12).

The systolic algorithm specification in Figure 28 and Figure 29 was used for modelling and simulation in Simulink. Correctness of the simulation results was then verified with the sequential algorithm specified in Table 3.

Figure 31 presents the convergence rate of matrix inverse computation for each of the five examples. The horizontal axis displays the number of iterations and the vertical axis displays the norm of the residual matrix (16).

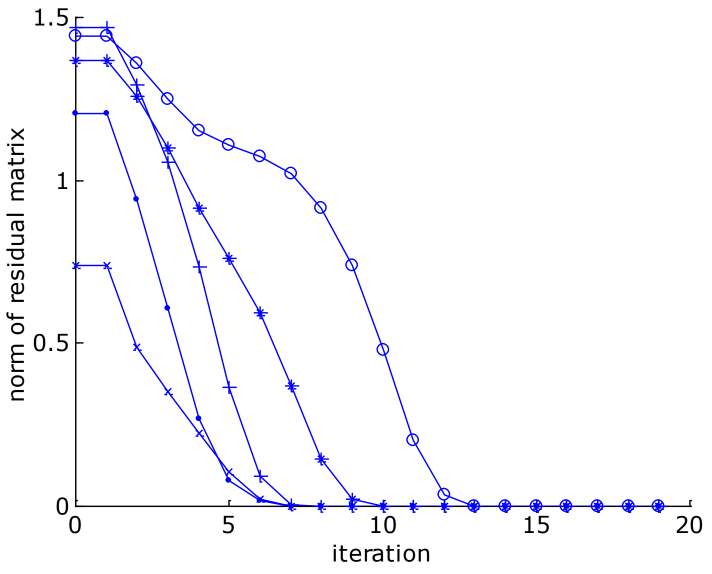


Figure 31: Convergence of Hotelling iterative method.

Table 4 presents some details from Figure 31. For each of the five cases we investigate how many iterations are needed before the norm of the residual matrix becomes smaller than the selected error *e* (Table 3). The figures are given for three different cases: $e = 10^{-2}$, 10^{-4}, and 10^{-8}. The results prove the fact that the convergence is very rapid after the approximation is close to the solution. Let consider the results of $\mathbf{A}_1$ from Table 4. Within the eight iteration the approximation to the matrix inverse satisfies e_1 as well as e_2. In the following iteration the number of the correct decimal digits doubles and the approximation satisfies e_3 as well.

		Error		
Matrix	**Symbol**	$e_1 = 10^{-2}$	$e_2 = 10^{-4}$	$e_3 = 10^{-8}$
$\mathbf{A}_1$	**x**	8	8	9
$\mathbf{A}_2$	**.**	7	8	9
$\mathbf{A}_3$	**o**	14	14	15
$\mathbf{A}_4$	*	10	11	12
$\mathbf{A}_5$	+	8	8	9

Table 4: Convergence of Hotelling iterative method. The number of iterations for particular precision.

Bibliography

[Ahmed82] H. M. Ahmed, J. M. Delosme, M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing", *Computer*, vol. 15, no. 1, 192-207, January 1982.

[Akl89] S. G. Akl, *The design and analysis of parallel algorithms*, Prentice-Hall, 1989.

[Array90] *Proceedings of Application Specific Array Processors*, editors: S. Y. Kung et al., IEEE Computer Society Press, 1990.

[Array94] *Proceedings of Application Specific Array Processors*, editors: P. Cappello et al., IEEE Computer Society Press, 1994.

[Bräunl93] T. Bräunl, *Parallel Programming: an Introduction*, Prentice Hall, 1993.

[Chuang85] H. Y. H. Chuang, G. He, "A Versatile Systolic Array for Matrix Computations," *Proc. 12th Int. Conf. Comput. Architecture*, Boston, MA, pp. 315-322, June 1985.

[Codenotti92] B. Codenotti, M. Leoncini, *An Introduction to Parallel Processing*, Addison-Wesley, 1992.

[Computer82] Special issue on: High-Speed Parallel Computing, *Computer*, vol. 15, no. 1, January 1982.

[Computer83] Special issue on: Applications for Array Processors, *Computer*, vol. 16, no. 6, June 1983.

[Computer87] Special issue on: Systolic Arrays, *Computer*, vol. 20, no. 7, July 1987.

[Computer97] Special issue on: The Future of Micro Processors, *Computer*, vol. 30, no. 9, September 1997.

[Datta95] B. N. Datta, *Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Company, 1995.

[Dougherty95] E. R. Dougherty, P. A. Laplante, *Introduction to Real-Time Imaging*, SPIE Optical Engineering Press, IEEE Press, 1995.

[DSP98] *The Digital Signal Processing Handbook*, editors: V. K. Madisetti, D. B. Williams, IEEE Press, 1998.

[Duncan90] R. Duncan, "A Survey of Parallel Computer Architectures", *Computer*, pp. 5-16, February 1990.

[Dudgeon84] D. E. Dudgeon, R. M. Mersereau, *Multidimensional digital signal processing*, Prentice-Hall Signal Processing Series, 1984.

[El-Amawy87] A. El-Amawy, W. A. Porter, J. L. Aravena, "Array architectures for iterative matrix calculations", *IEE Proc.*, pt. E, vol. 134, no. 3, pp. 149-154, May 1987.

[El-Amawy89] A. El-Amawy, "A Systolic Architecture for Fast Dense Matrix Architecture", *IEEE Trans. on Computers*, vol. 38, no. 3, pp. 449-455, March 1989.

[Elliot92] D. G. Elliot, W. M. Snelgrove, M. Strumm, "Computational RAM: a Memory-SIMD Hybrid and its Application to DSP", *Proc. of CCIC'92*, Hawaii, 1992.

[Flynn66] M. J. Flynn, " Very High Speed Computing Systems", *Proc. of the IEEE*, pp. 1901- 1909, December 1996.

[Flynn95] M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Jones and Barlett Publishers, 1995.

[Flynn96] M. J. Flynn, K. W. Rudd, "Parallel Architectures", *ACM Computer Surveys*, March 1996.

[Freeman92] T. L. Freeman, C. Phillips, *Parallel numerical Algorithms*, Prentice Hall, 1992.

[Hennessy96] J. L. Hennessy, D. A. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann, 1996.

[Higham96] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Siam, 1996.

[Krishnamurthy89] E. V. Krishnamurthy, *Parallel Processing: Principles and Practise*, Addison-Wesley, 1989.

[KungHT78] H. T. Kung, C. E. Leiserson "Systolic Arrays (for VLSI)," *Technical Report CS 79-103*, Carnegie Mellon University, 1978.

[KungSY88] S. Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[Leighton92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, 1992.

[Li85] G.-J. Li, B. W. Wah, "The design of optimal systolic arrays", *IEEE Trans. on Computers*, vol. C-34, no. 1, pp. 66-77, January 1985.

[Madisetti95] V. K. Madisetti, *VLSI Digital Signal Processors*, IEEE Press, 1995.

[McWhirter92] J. G. McWhirter, "Algorithmic Engineering in Adaptive Signal Processing", http://siwg.dra.hmg.gb/signal/www/AlgAndArch/papers/AEI.ps, DRA, Electronic Division, RSRE, 1992.

[Ortega88] J. M. Ortega, *Introduction to parallel and vector solution of linear systems*, Plenum, 1988.

[Parhami02] B. Parhami, *Introduction to Parallel Processing*, Kluwer Academic Press, 2002.

[Parhi99] K.K. Parhi, *VLSI Signal Processing Systems*, John Wiley & Sons, 1999.

[Patterson97] D. Patterson, et al., “A case for intelligent RAM”, *IEEE Micro*, vol. 17, no. 2, pp. 34-44, March/April 1997.

[Petkov93] N. Petkov, *Systolic Parallel Processing*, North-Holland, 1993.

[Proakis92] J. G. Proakis, D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 2nd edition, Macmillan, 1992.

[Proceedings96] *Proceedings of the IEEE, Special Issue on Parallel Architectures for Image Processing*, editor: M. Moresca, vol. 84, no. 7, July 1996.

[Proudler93] I. K. Proudler, J. G. McWhriter, “Algorithmic Engineering in Adaptive Signal Processing II- worked examples”, Internet: http://siwg.dra.hmg.gb/signal/www/AlgAndArch/papers/AEII.ps, DRA, Electronic Division, RSRE, England, 1993.

[Quinn94] M. J. Quinn, *Parallel Computing; Theory and Practice*, 2nd edition, McGraw Hill, 1994.

[Quinton89] P. Quinton, V. van Dongen, “The Mapping of Linear Recurrence Equations on Regular Arrays”, *Journal of VLSI Signal Processing*, vol. 1, no. 2, pp. 95-113, 1989. Reprinted in [VLSI98a].

[Quinton91] P. Quinton, Y. Robert, *Systolic algorithms and architectures*, Prentice Hall, 1991.

[Rader96] C. M. Rader, “VLSI Systolic Arrays for Adaptive Nulling”, *IEEE Signal Processing Magazine*, vol. 13, no. 4, pp. 29-49, July 1996.

[Rao88] S. K. Rao, T. Kailath, “Regular Iterative Algorithms and their Implementation on Processor Arrays”, *Proceedings of the IEEE*, vol. 76, no. 3, pp. 259-269, March 1988. Reprinted in [VLSI98a].

[Schaller97] R. R. Schaller, “Moore’s law: past, present and future”, *IEEE Spectrum*, vol. 34, no. 6, pp. 53-59, June 1997.

[Skillikorn88] D. B. Skillikorn, “A Taxonomy for Computer Architectures”, *IEEE Computer*, vol. 21, no. 11, pp. 46-57, November 1988.

[Trefethen97] L. N. Trefethen, D. Bau III, *Numerical linear algebra*, SIAM, 1997.

[Urquhart84] R. B. Urquhart, D. Wood, “ Systolic matrix and vector multiplication methods for signal processing”, *IEEE Proc.*, pt. F, vol. 131, no. 6, pp. 623-631, October 1984.

[VLSI85] *VLSI and Modern Signal Processing*, editors: S. Y. Kung, H. J. Whitehouse, T. Kalaith, Prentice Hall, 1985.

[VLSI94] *VLSI Design methods for Digital Signal Processing Architectures*, editors: M. A. Bayoumi, Kluwer Academic Publishers, 1994.

[VLSI94a] *VLSI Signal Processing Technology,* editors: M. A. Bayoumi and E. E. Swartzlander, Kluwer Academic Publishers, 1994.

[VLSI98a] *High Performance VLSI Signal Processing: Innovative Architectures and Algorithms*, vol. I: Algorithms and Architectures, Edited by: K. J. R. Liu, K. Yao, IEEE Press, 1998.

[VLSI98b] *High Performance VLSI Signal Processing: Innovative Architectures and Algorithms*, vol. II: Systems Design and Applications, Edited by: K. J. R. Liu, K. Yao, IEEE Press, 1998.

[Wan93] C. R. Wan, D. J. Evans, “A systolic array architecture for linear and inverse matrix systems”, *Parallel Computing*, vol. 19, pp. 303-321, North-Holland, 1993.

[Wan93a] C. R. Wan, D. J. Evans, “A Systolic Architecture for Capon’s Directions-of-Arrival (DOA) Estimation Method”, *Parallel Algorithms and Applications*, vol. 1, pp. 99-114, 1993.

[Wan94] C. R. Wan, D. J. Evans, “A systolic array architecture for QR decomposition of block structured sparse systems”, *Parallel Computing*, vol. 20, pp. 903-914, 1994.

[Wan98] C. R. Wan, D. J. Evans, “Nineteen ways of systolic matrix multiplication”, *Intern. J. Computer Math.*, vol. 68, pp. 39-69, 1998.

[Wanhammar99] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, 1999.

[Zajc98] M. Zajc, R. Sernec, J. Tasič, "Fixed Structure Systolic Array and Processing of Images", *Proc. of MELECON’98*, vol. 1, pp. 96-100, Tel Aviv 1998.

[Zajc98a] M. Zajc, R. Sernec, J. Tasič, "Matrix Multiplication Based Systolic Array: Schur Complement Computation", *Proc. of WDTA’98*, pp. 57-60, Dubrovnik 1998.

Matej Zajc

is assistant professor at University of Ljubljana, Slovenia. He received M.Sc. in VLSI Systems Design at University of Westminster, Great Britain in 1996. In 1999 he received Ph.D. at University of Ljubljana in the field of parallel digital signal processing.

Since 1999 he works at University of Ljubljana lecturing and tutoring at different undergraduate and graduate modules at Communications department such as Microprocessor systems in telecommunications and Parallel architectures for real-time signal processing, Electronics, Project management in engineering.

His research interests include parallel processing, massively parallel algorithms, digital communication systems. Professionally he is involved in different European IST projects and domestic applied projects.