11 Systolic arrays

Types of special-purpose computers:

1. Inflexible and highly dedicated structures
2. Structures, enabling some programmability and reconfiguration.

Are there applications for array of simple processors on chip?

Systolic and wavefront arrays are determined by pipelining data concurrently with the (multi)processing - *data and computational pipelining*.

- Wavefront arrays use data-driven processing capability.
- Systolic arrays use local instruction codes synchronized globally.

Timing scheme \ Data I/O	Data pipelined through boundary processors	Data preloaded from data bus
Globally synchronous	Systolic (prestored local control)	SIMD (broadcasted control)
Data driven	Wavefront (prestored local control)	MIMD (prestored local control)

Definition: A *systolic array* is a network of processors that rhythmically compute and pass data through the system.

11.1 Applications

Basic matrix algorithms:

- **matrix-vector multiplication**
- **matrix-matrix multiplication**
- **solution of triangular linear systems**
- **LU and QR decomposition (resulting triangular matrices)**
- **preconditioned conjugate gradient (solving a set of positive definite symmetric linear equations), etc.**

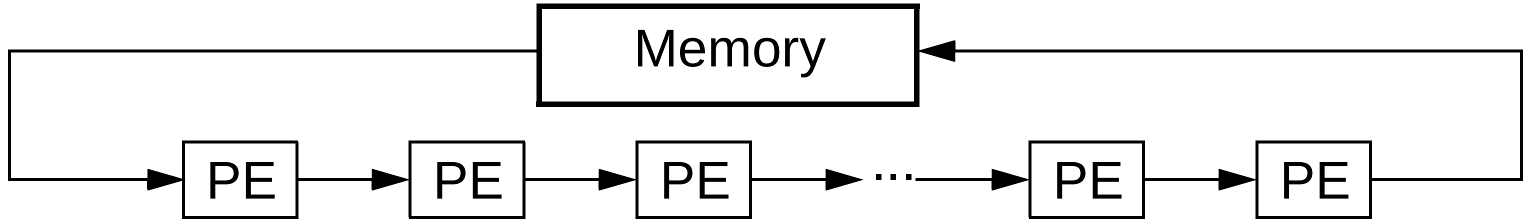
In general, various DSP algorithms can be mapped to systolic arrays:

- **FIR, IIR, and ID convolution**
- **Interpolation**
- **Discrete Fourier Transform**
- **Template matching**
- **Dynamic scene analysis**
- **Image resampling, etc.**

Non-numeric applications:

- **Data structures - stacks and queues, sorting**
- **Graph algorithms - transitive closure, minimum spanning trees**
- **Language recognition**
- **Dynamic programming**
- **Relational database operations, etc.**

11.2 Basic configuration



Data item is not only used when it is input but also *reused* as it moves through the pipelines in the array.

Factors to consider:

- Balancing the processing and input/output bandwidths
- Balancing between general-purpose and special-purpose systolic systems
- Cost of implementation
- Modularity and expansion capabilities
- Trade-off between simplicity and complexity (capabilities of the cell)

Thoroughly investigated in eighties [**IEEE Computer**, Jan. 1982, **July 1987**].
“Systolic array” - analogy with the human circulatory system:

- heart == global memory,
- network of veins == array of processors and links.

11.2.1 VLSI processor arrays

Design criteria:

- Modularity and simplicity
- Locality and regularity
- Extensive pipelineability and parallelism
- Efficiency and speedup
- Data streams in different directions.

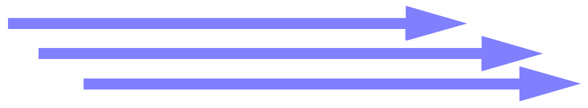
TOPOLOGY:

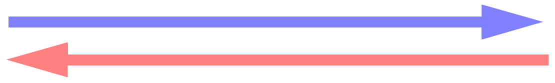
Planar arrays:

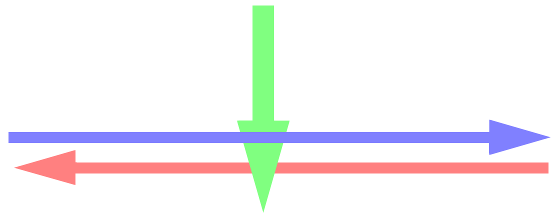
- Triangular
- Square
- Hexagonal

Linear arrays:

- **U**nidirectional **L**inear **A**rray (one, two or three data-paths (**Why?**))
- **B**idirectional **L**inear **A**rray (opposite streams)
- **T**hree-path communication **L**inear **A**rray



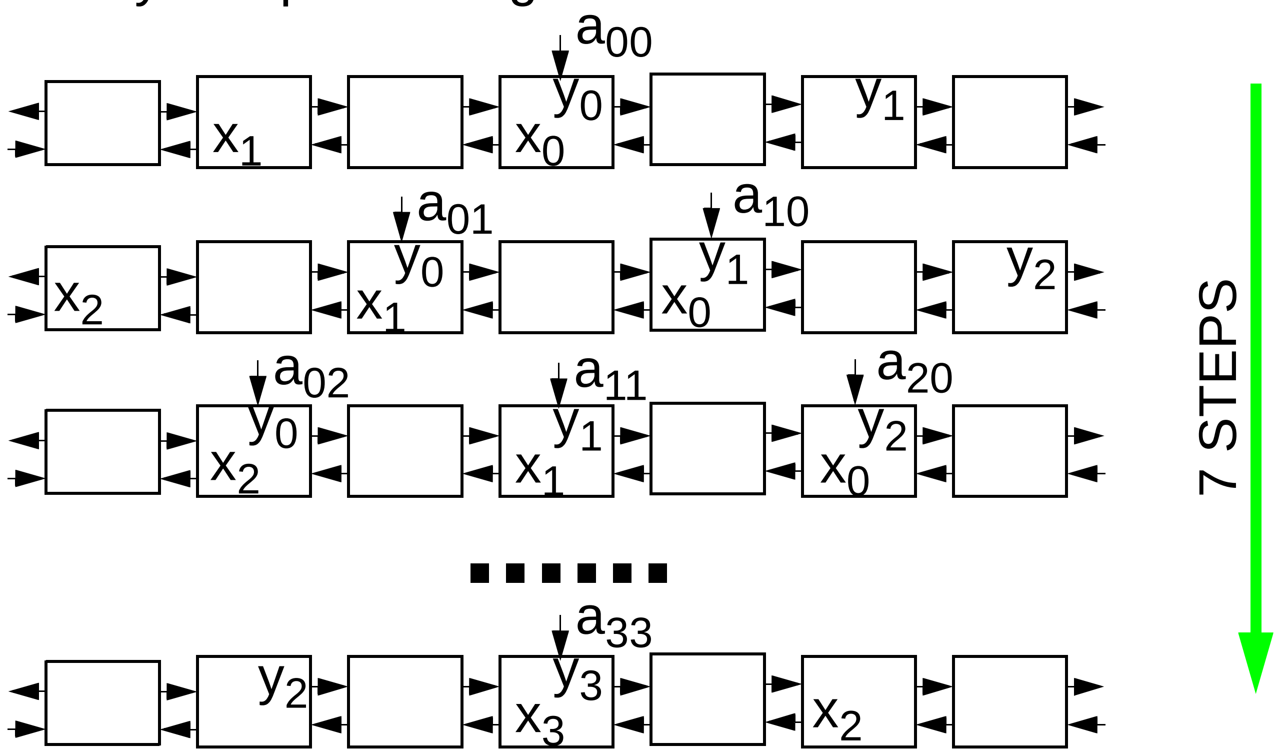




11.2.2 Systolic vector-matrix multiplication

Invented by Kung and Leiserson (1978).

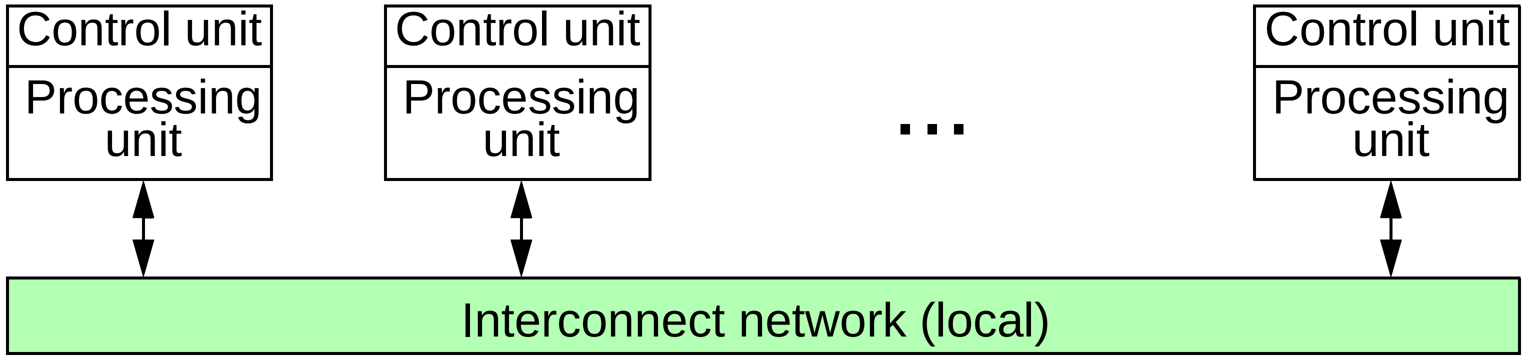
Let's consider vector-matrix multiplication $\|X\| \times \|A\| = \|Y\|$, where A is $n \times n$ matrix. In case of n=4, the multiplication can be organized using bidirectional linear array of 7 processing elements:



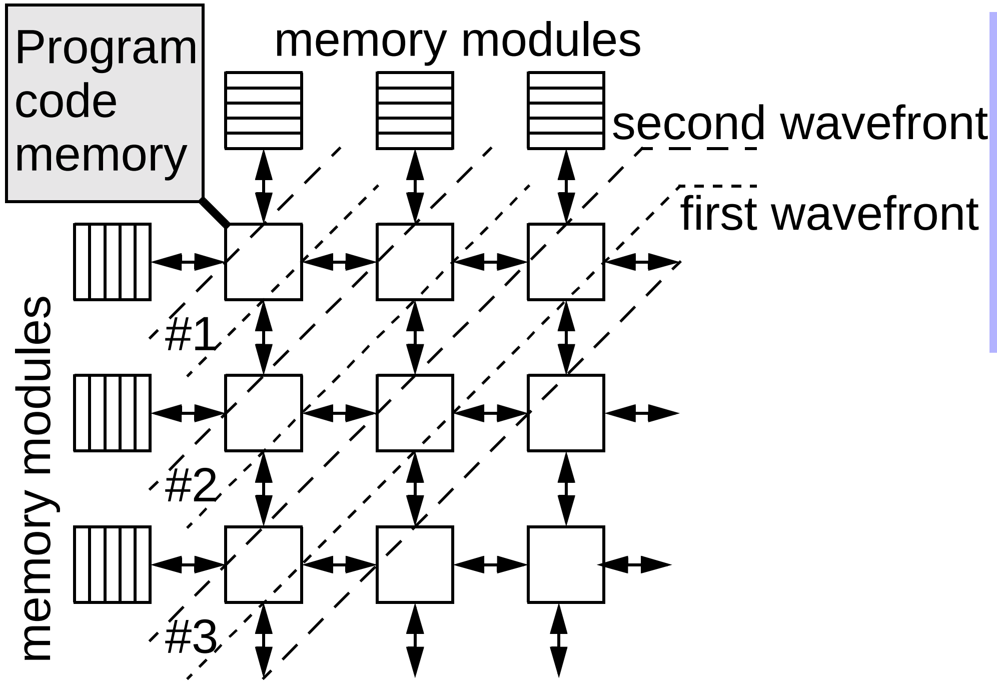
Inner product step (ISP) cell:

$$y_{out} = y_{in} + x_{in} \times a_{in}$$
$$x_{out} = x_{in}$$

11.2.3 Wavefront systolic multiplication



Example of matrix-matrix multiplication:



$$C = A \times B$$

Decomposed into multiplication of columns of A_i and rows of B_j.

$$C = A_1 \times B_1 + A_2 \times B_2 + \ldots + A_N \times B_N$$

At #1 and processor (1,1):
First wave:

$$c_{11}^{(1)} = c_{11}^{(0)} + a_{11} \times b_{11}$$

Second wave:

$$c_{11}^{(2)} = c_{11}^{(1)} + a_{12} \times b_{21}$$

11.2.4 Implementation issues I

Design of large-purpose systems:

- Adding HW mechanism to reconfigure the topology and interconnection matrix;
- Using SW to map different algorithms into a fixed-array architecture;
- Combination of both.

Design and mapping techniques:

To synthesize a systolic array from the description of an algorithm, a designer needs a through understanding of:

1. systolic computing
2. application
3. algorithm
4. technology

Granularity:

- bit-wise
- word-level
- complete program

Extensibility:

- Execution the same algorithm for a problem of a larger size?

11.2.5 Implementation issues II

Clock synchronization:

- Avoiding clock skews (specific layout, data flow in one direction)
- Self-timed system - *wavefront arrays*

Reliability:

$$\text{Reliability of an array of processors} = \text{Reliability of processor}^{\text{\# of processors in array}}$$

- Introducing the fault-tolerant mechanisms
- run-time testing
- maximizing reliability while minimizing the corresponding overhead

Partitioning of large problems:

Executing a large problem without building a large systolic array.

- Identifying and exploiting algorithm partitions (array “travels through the set of computations of the algorithm in the right order until it “covers” all the computations)
- restating the problem so that the problem becomes a collection of smaller problems.

- Holy grail - universal building block
- Integration into existing systems

11.2.6 VLSI processor array characteristics

Timing:

⌘ *Execution time* - all the computations

⌘ *Input time* - data to processor where the first computation takes place

⌘ *Output time*[1] - data from processor where the last computation finished

⌘ *Load time* - data elements to array

⌘ *Unload time* - resulting data elements exit the array

⌘ *Total processing time*

Systolic array algorithms:

⌘ *Synchronicity* - asynchronous mode - wavefront, synchronous mode - SA

⌘ *Concurrency control* - dependency between computations

⌘ *Granularity* - simplicity of the cells first

⌘ *Communication geometry* - local interconnections

Speedup and efficiency: $S_p = \frac{T_1}{T_p}, E_p = \frac{S_p}{p}$

1. Usually, the input and output time equal to the processor array size.

11.3 Algorithm mapping and programming

Efficiency of systolic array implementation in VLSI depends of locality of interconnections.

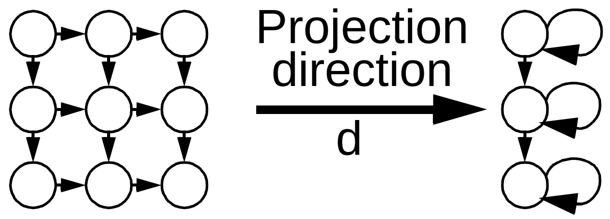
Algorithmic notation expressing the recurrence and parallelism (space-time activities) — Data Graph (DG) design

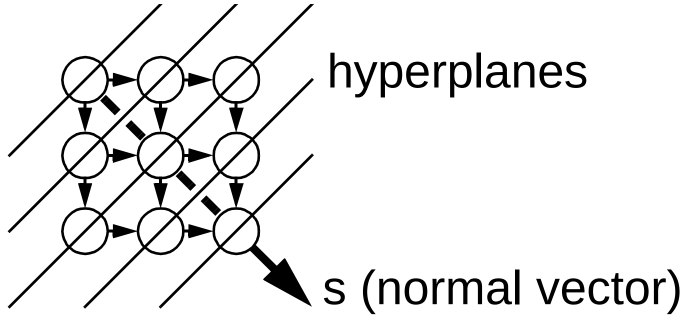
↓

VLSI hardware description or executable array processor machine code — Data Graph implementation

Straightforward implementation of a DG (assigning each node in DG to a PE) is not (area) efficient.

- Mapping DGs to systolic arrays
  - **linear assignment (projection) -** nodes along a straight line are mapped to a PE
  - **linear schedule -** mapping a set of parallel equitemporal hyperplanes to a set of linearly increased time indices.





11.3.1 Rules to obtain systolic array from DG

Let’s denote time index of a node $s^T i$.

- $s^T e > 0$. Here e denotes any edge in the DG. The number $s^T e$ denotes the number of delays (Ds) on the edge of the systolic array. The schedule vector s must obey the data dependencies of the DG; that is, if node i depends on the output of node j, then j must be scheduled before i.
- $s^T d > 0$. The projection vector d and the schedule vector s cannot be orthogonal to each other; otherwise, sequential processing will result.

In systolic mapping, the following rules are adopted:

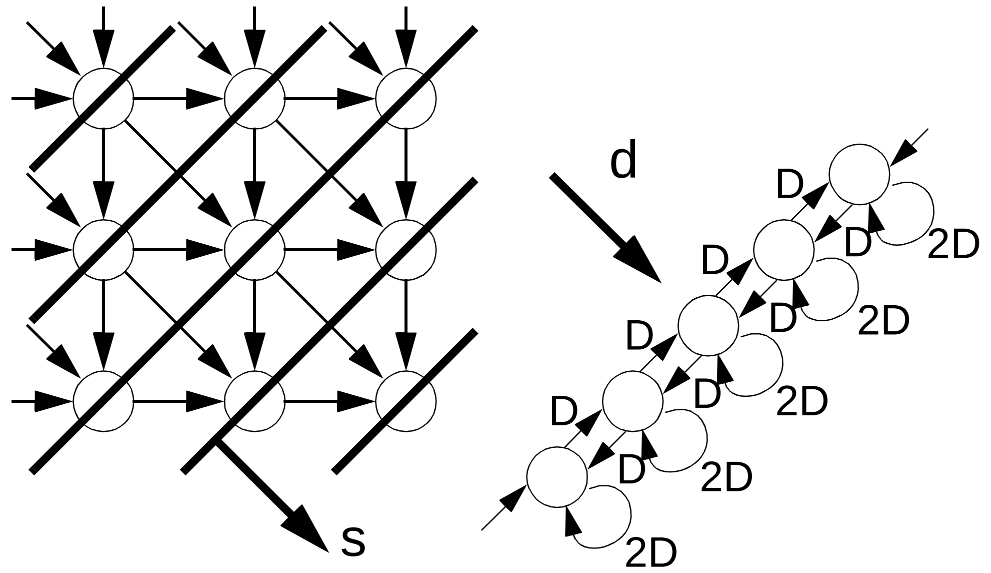
- The nodes in the systolic array must correspond to the projected nodes in the DG.
- The arcs in the systolic array must correspond to the projected components of arcs in DG.
- The input data must be projected to the corresponding arcs in the systolic array.

The DG is *shift-invariant* if the dependency arcs corresponding to all the nodes in the index space do not change with respect to the node positions.

(Matrix multiplication, convolution, autoregressive filtering, discrete Fourier transform, discrete Hadamard transforms, Hough transforms, least squares solutions, sorting, perspective transforms, LU decomposition, QR decomposition belong to class of shift-invariant algorithms)

11.3.2 Example of mapping DG to a systolic array

DG for convolution-like algorithm:



s - hyperplane’s normal vector
d - projection direction
D - delay

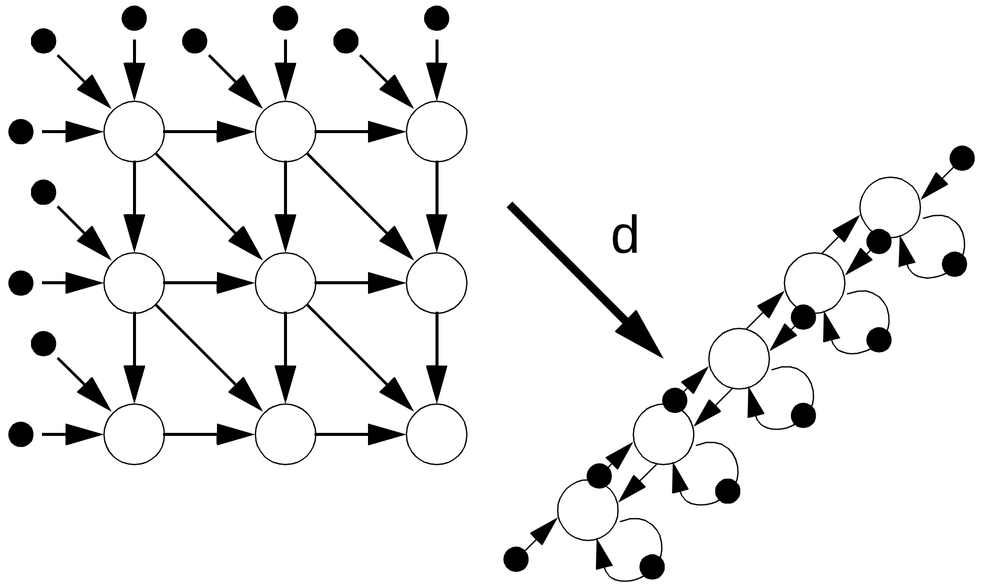
11.3.3 Mapping DGs to wavefront arrays

A wavefront array does not have a fixed schedule => operation is didacted by:

- data dependency structure;
- initial data tokens.

A wavefront array can be modeled by a dataflow graph (DFG).

A node is *enabled* when all input arcs contain tokens and all output arcs contain empty queues.



d - projection direction

Connection with Petri nets?

11.3.4 Process of mapping DG to DFG

- To get shift-invariant DG (to get uniform appearance), some of boundary nodes have to be assigned some initializing data.
- Each input data token in the DG is mapped to a initial token on the corresponding arc in the DFG.
- The queue size for each DFG arc is assumed to be large enough to accommodate the target algorithms.

✔ **Wavefront design is more appealing in cases where there is timing uncertainty among the nodes in DG.**

✘ **Analyzing the exact performance of a wavefront array, which is data-driven and sometimes data-dependent, is very difficult.**

- ✔ **There exists upper bound of the execution time of wavefront arrays.**

11.3.5 Queues

Using queues is a way to implement asynchronous communication in a wavefront array.

- Can be implemented in SW or in HW.

Let the DG node computation times to be data-independent. Hence, the DG can be scheduled *a priori* and the minimum computation time can be determined.

- ✘ **Insufficient queue size usually results in an additional slowdown of the computation.**

Let’s suppose that DG arc *a* is projected to DFG arc *a’*. Then:

- The scheduled completion time, t_1, for the initiating node of *a* indicates when the output data of the node are produced (put) on *a’*.
- The scheduled completion time, t_2, for the terminating node of *a* indicates when the data are consumed from *a’*. Apparently, if τ is the node computation time, then $t_2 - t_1 + \tau$ represents the length of time a data token stays in *a’* and its two end nodes.
- The pipelining period, α, which is the time period between two consecutive data being put on *a’*, can be determined from the schedule. Thus the queue size for *a’*, *Q*, can be calculated as: $Q = \lceil (t_2 - t_1 + \tau) / \alpha \rceil$. This is correct when sustained rate is equal at initiating and at terminating node.

11.4 Systolic array programming

Let’s look the case with *fixed interconnections* between PEs and *fixed queue length* for each data link. *Programming a wavefront array* means specifying the sequence of operations for each PE. Each operation includes:

- the type of computation (addition, multiplication, division, etc.)
- the input data link (north, south, east, west, or internal register),
- the output data link,
- an additional specification i.e. time scheduling, when an operation in PE actually occurs (not required in wavefront array).

Programming languages

- **Occam** (historically for Inmos Transputers). Multiplication example:

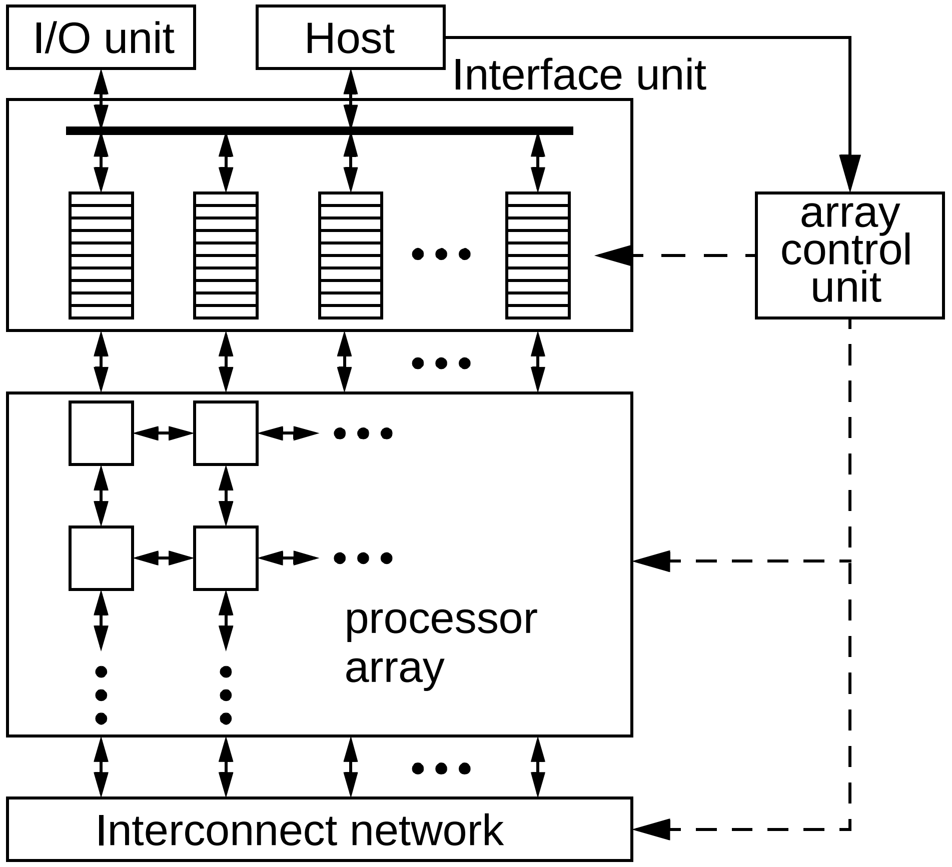
```
CHAN vertical [n*(n+1)]:
CHAN horizontal[n*(n+1)]:
 PAR i=[0 FOR n]
   PAR j=[0 FOR N]
     mult (vertical[(n*i)+j], vertical[(n*i)+j+1],
     horizontal[(n*i)+j], horizontal[(n*(i+1))+j]):
```

- **MDFL** - Matrix Data Flow Language

11.5 Architecture

The system components:

- processor array(s)
  - **1-D (linear),**
  - **2-D (mesh, hexagonal),**
  - **3-D ((hyper)cube connected),**
- interconnect network(s) - local, global, inter-array, intra-array,...
- a host computer, interface unit.



Processing element components:

- ALU (fixed or floating point)
- memory unit
- control unit (RISC, CISC)
- I/O unit (processing concurrently with data transfer)

Host:

- system monitoring, batch data storage, management, data formatting
- determines the schedule program for interface unit and network
- generates global control codes and object codes for PEs.

11.6 Conclusion

Why wavefront arrays?

Factor	Systolic array	Wavefront array
Clock distribution	Critical, but may result conceptually simpler design. Clock skew problem.	Minor problems
Processing speed	Suffer when processing times in PEs are not uniform.	Handshaking overhead. Beneficial when processing is data dependent (skipping multiplication with zero).
Programming	Assignment of computations to PEs and scheduling	Only assignment of computations to PEs.
Fault tolerance: ✦ fabrication-time ✦ compile-time ✦ runtime	None. For self-testing full array must be interrupted.	Runtime fault tolerance (due to data-driven behaviour). Self-testing of a single PE is possible.